

# An efficient distributed security policy management for gateway-less VPNs

**Yuichi Ishikawa, Norihito Fujita and Akira Tsukamoto**

System Platforms Research Laboratories

NEC Corporation

1753, Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa 211-8666, Japan

E-mail: {y-ishikawa@df, n-fujita@bk, a-tsukamoto@bu}.jp.nec.com

## Abstract

An efficient security policy management scheme for gateway-less virtual private network (VPN) architectures is discussed. Although gateway-less VPN architectures are more scalable than gateway-based one, conventional gateway-less VPN architectures are difficult to deploy for collaborative groups with dynamic membership. In conventional architectures, the entire VPN security policy is pre-installed in each node, and the management server advertises the security policy updates associated with membership changes to all nodes in the VPN whenever a group member changes, which imposes a heavy load on the management server. To support dynamic membership changes in a scalable manner, we propose an “on-demand” security policy resolution scheme. By utilizing DNS name resolution mechanism, each node pulls only the part of the security policy required to connect to the destination from the management server in an on-demand fashion, instead of obtaining the entire security policy locally. Therefore, when a group member changes, only updates of security policies maintained on the management server are necessary; the server does not advertise policy updates to member nodes, which considerably reduces the load on the server. Evaluation of the load imposed on the server when a member changes demonstrated that this scheme performs efficiently for an increased number of nodes and frequent membership changes.

## Keywords

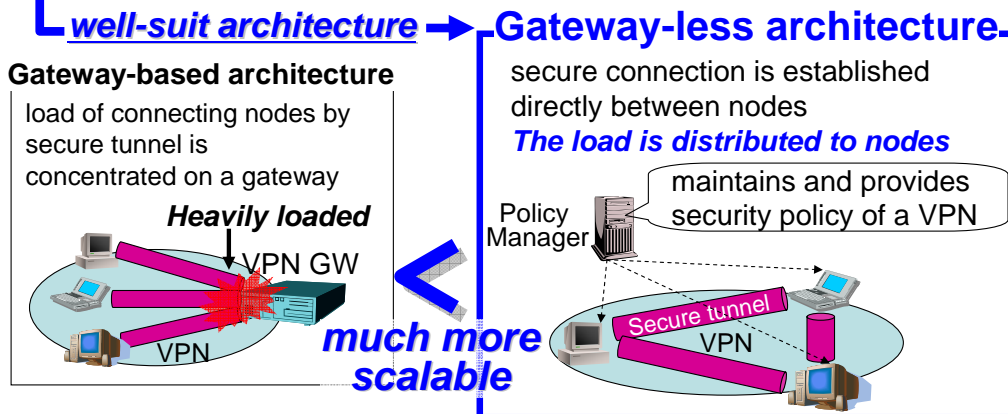
Virtual Private Network, Security Policy, DNS, Gateway-less, Policy Manager

## Contact Person

Yuichi Ishikawa (y-ishikawa@df.jp.nec.com)

# 1. Introduction

**New target of VPN services:  
provide collaborative groups with secure network platform  
e.g. temporal business projects, club activities in a school**



**Key Issue: How to support dynamic membership changes**  
security policy update associated with membership changes must be immediately reflected to communication in a VPN

## 1. Introduction

With the increasing risks in networks, such as spyware and identity theft, various new types of virtual private network (VPN) services are now being commercially provided in addition to legacy VPN services, such as remote access VPNs and intra/extranet VPNs [1]–[4]. One of the most important targets of the new VPN services is to provide a secure closed network for collaborative groups with dynamic membership, such as temporal business project teams and student clubs that forbid the use of legacy, pre-configured, static VPN services because of their dynamic properties.

To provide such closed networks to these groups, two architectures have been introduced; one is a gateway-based VPN architecture [1][2] and the other is a gateway-less VPN architecture [3][4]. In the gateway-based architecture, all secure tunnels between member nodes are established in a centralized manner via a VPN gateway, in which communication traffic is concentrated at the gateway, resulting in heavy loading. In the gateway-less architecture, secure tunnels are established directly between member nodes. The load for providing the VPN's security policy (a set of rules that defines which nodes are group members, where the group members are, and how to connect to the member nodes (e.g., member nodes' IP addresses/hostnames and encryption algorithm for securing tunnels)) to nodes is concentrated on a special policy manager server, which means that this architecture is much more scalable than the gateway-based architecture. In this paper, we focus on a gateway-less VPN architecture for providing secure, closed networks to collaborative groups.

Because collaborative groups are created and managed on an on-demand basis, gateway-less architectures must support dynamic membership changes. Specifically, updates of the VPN security policy related to group membership changes must be reflected in the members' communication in a gateway-less VPN.

However, conventional gateway-less architectures have a scalability problem when they are used for collaborative groups with dynamic membership changes. Because security policy updates must be advertised to all member nodes by the policy manager whenever the group membership changes, the policy manager is heavily loaded in conventional architectures.

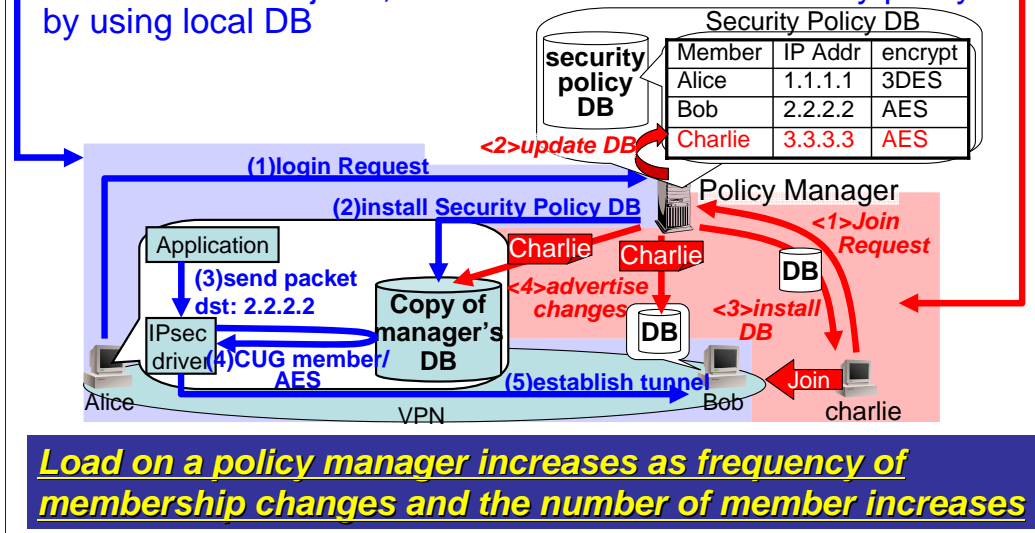
In this paper, we describe an efficient security policy resolution scheme for gateway-less VPN architectures. Our scheme uses on-demand based security policy resolution instead of the pre-installed security policies found in conventional architectures. In our scheme, each node pulls only the required parts of the security policy from the policy manager using a DNS name resolution mechanism [5]. Our scheme eliminates the advertisement-based policy update and makes the load on the policy manager much less than that of the conventional architectures.

## 2. Previous Work

**Whenever the membership changes, the policy manager must advertise security policy update to all members**

**Security policy is pre-installed in member nodes**

Whole security policy of a VPN is installed in a node's local DB when a node joins, and a node resolves a security policy by using local DB



## 2. Previous Work

There have been several studies on gateway-less VPN architectures. The Policy-Based Network Management System developed at the University of Murcia (UMU-PBNM) [3] addresses secure communication services based on the public key infrastructure (PKI). It supports dynamic reconfiguration of security policies for membership changes using the common open policy service (COPS) protocol. The dynamic VPN controller (DVC) [4] takes a similar approach to UMU-PBNM. It allows users to dynamically initiate and tear down secure tunnels by using a security policy negotiation mechanism.

In these architectures, the policy manager maintains the entire VPN security policy. When a node participates in a VPN, the policy manager installs the policy in nodes ((1) and (2) in the slide), and when the node starts to communicate with another node in the VPN, it refers to the pre-installed security policy to establish a secure tunnel to the destination. Specifically, this pre-installed security policy, which includes such parameters as the node IP addresses and encryption algorithms, is used to determine whether packets sent from an application are destined for the VPN or not and which encryption algorithm should be used to establish a secure tunnel to a destination ((3),(4), and (5) in the slide). Therefore, when a change occurs in a VPN, such as when a new node joins or one leaves, the security policy installed in each node must be updated to enable each node to communicate correctly.

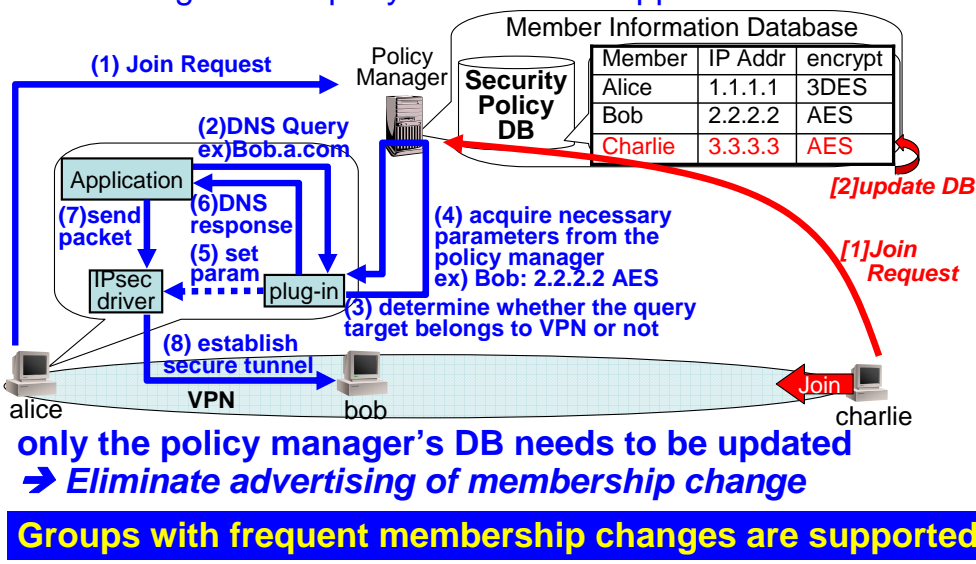
When membership changes, the updated security policy is installed in each node by the policy manager. When a new node joins the VPN, the policy manager is notified (<1> in the slide); it updates its security policy database (<2> in the slide) and then advertises the security policy update to all nodes in the VPN (<4> in the slide). When a node leaves, another advertisement is sent out .

However, this advertisement-based policy update scheme imposes a heavy load on the policy manager, especially when membership changes frequently or a large number of nodes participate. Thus, it is difficult for the conventional architectures to support groups with dynamic membership changes in a scalable manner.

### 3. On-demand Security Policy Resolution scheme(1/2)

#### Security policy is pulled from the policy manager on an on-demand basis

a node determines security policy needed to be pulled by monitoring a DNS query sent from an application



### 3. On-demand security policy resolution scheme

In this section, we describe our “on-demand” security policy resolution scheme for gateway-less VPN architectures that support dynamic group membership changes in a scalable manner.

In our scheme, member nodes “pull” the necessary security policy from the policy manager in an on-demand fashion instead of having the whole security policy locally. When a node communicates with another node, it pulls only the parameters needed to establish a secure tunnel to the destination. Thus, to support dynamic membership changes, only the policy manager’s database needs to be updated, which is performed in a way similar to that in conventional architectures.

This pull-based policy resolution scheme eliminates the advertising of security policy updates to all members. Therefore, the policy manager’s load to support dynamic membership changes is much less than for conventional architectures.

To perform pull-based policy resolution, it is necessary to determine which part of the security policy needs to be pulled from the policy manager (i.e., with which node an application starts to communicate) before an application starts sending packets to its destination. To achieve this, our scheme utilizes a DNS name resolution mechanism. Before an application starts communicating, it resolves the destination’s IP address from the hostname by sending a DNS query. Thus, by monitoring the domain name in the DNS query, the security policy required to establish a secure tunnel to the destination can be determined before the application starts sending packets.

The specific process is as follows (the numbers in the explanation correspond to those shown in the slide). To monitor the DNS query an application sends, a plug-in module which intercepts all DNS queries is installed in a member node. When it intercepts a DNS query (2), it determines whether the query is destined for a VPN or not by the host name in the query (3). Then, if the query’s destination is a VPN, the host acquires the necessary parameters to establish a secure tunnel to the destination by sending a request to the policy manager (4). After that, the plug-in module sends the acquired parameters to an IPsec driver (5) and a DNS response to the application (6). When the application starts sending packets (7), the IPsec driver refers to the parameters to determine whether the packets are destined for a VPN and establishes a secure tunnel if they are (8).

The process described above assumes that an application performs name resolution when it starts communication. However, when an application uses an IP address to designate the destination, name resolution is not performed. This is also supported by our scheme, which are described in section 5.2.

### 3. On-demand Security Policy Resolution scheme(2/2) ~ Security Policy “Caching” ~

- Member nodes cache resolved policy for a certain period ( TTL ) designated by the policy manager
- **Further server load reduction**
- Policy manager adjusts TTL value based on average residence time\*
- **Eliminating obsolete cache entries, which cause extra latency to establish tunnels due to policy re-resolution**

**\*residence time:**

**duration for which a node participates in a VPN**

#### **Security policy caching**

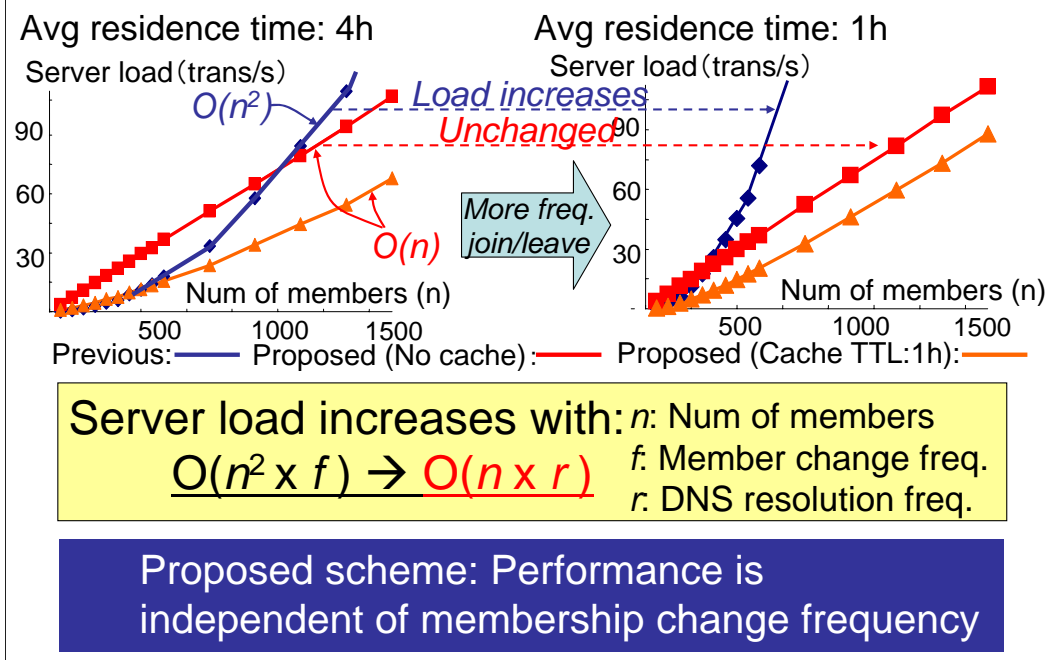
While the pull-base policy resolution scheme can support frequent membership changes, it can also induce too many server-client interactions if the member nodes communicate frequently, such as when security policy resolution is performed frequently. To address this problem, our scheme adopts a cache mechanism.

The plug-in module caches the resolved policy during the time-to-live (TTL) designated by the policy manager in the same way as normal DNS name resolution. If no record is found on the policy resolution, a negative cache is created to explicitly recognize the absence of the node that corresponds to the destination IP address.

Although a long TTL effectively reduces the policy manager’s load due to reduced server-client interactions, it increases the risk of obsolete cache entries that cause communication failures. If obsolete entries are used, a certain period of time has to elapse until a timeout is detected and a corresponding security policy is re-resolved, which would cause either packet drops or extra latency until the timeout.

To minimize the number of obsolete cache entries, the policy manager dynamically adjusts the TTL. The TTL is determined based on the average time that member nodes participate in the VPN (referred to as “residence time” in this paper). If the average residence time is relatively long, the TTL is lengthened, and vice versa. The average residence time is calculated by monitoring the join and leave requests at the policy manager. More specific guidelines for determining the TTL are discussed in section 4.2.

## 4.Evaluation: (1)Policy manager load



### 4. Evaluation

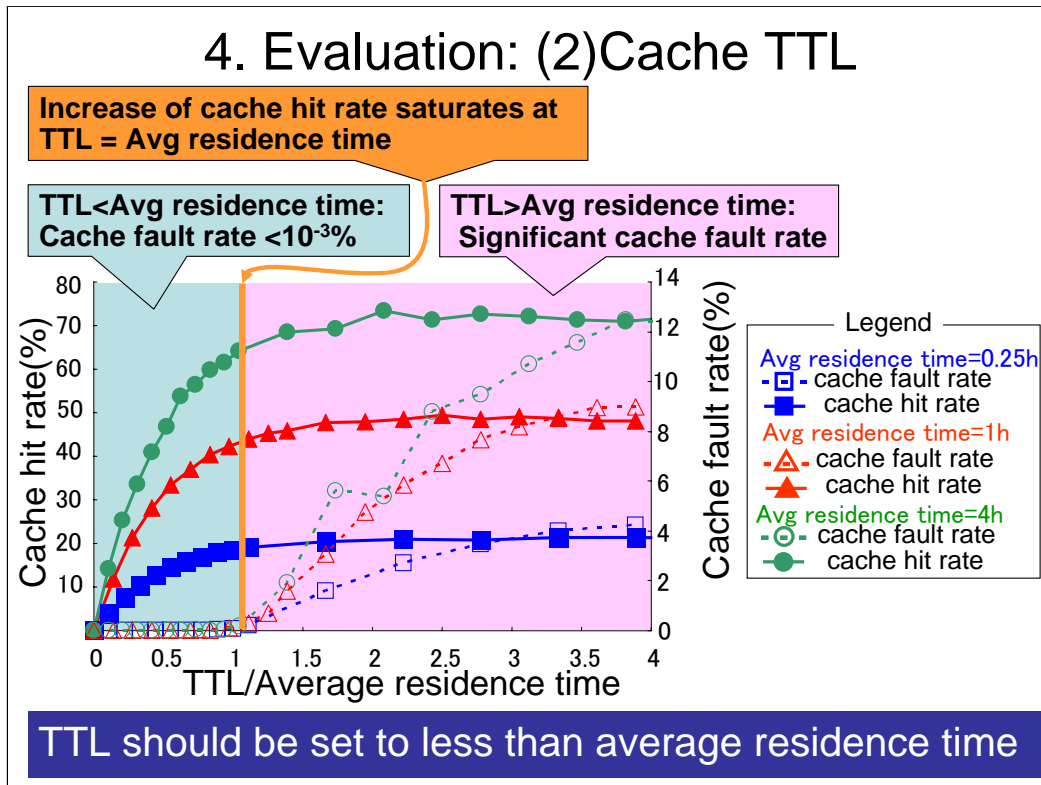
#### 4.1. Policy manager load

We simulated a load on the policy manager to evaluate the performance of our proposed scheme.

In the simulation, nodes dynamically joined and left a VPN, and their average residence times varied from one to four hours. We compared the server load in our scheme with that in conventional schemes, where the security policy is advertised to all member nodes whenever nodes join or leave. For simplicity, the server load was assumed to be proportional to the number of security policies resolved and/or distributed by the server during a period of time. In this simulation, each node initiated a session with a randomly selected node every ten seconds on average. When the TTL was zero, each node resolved the security policy every ten seconds. Although this was too frequent, we wanted to evaluate the performance under the harshest conditions.

These graphs show the results of the server load, measured for a different number of nodes participating in the VPN. From them, we can see how the advertisement-based scheme places a heavier load on the server because of the increase in the number of nodes. This is because the number of nodes causes a higher frequency of node joins and leaves and an increase in the number of target nodes receiving updated IPsec policies. In this scheme, the server load increases roughly with the square of the number of nodes. Moreover, as the average residence time decreases, the previous scheme has a higher server load due to the higher frequency of node joins and leaves. In contrast, in our scheme, the server load increases linearly with the number of nodes. The performance characteristics are independent of the residence times of the member nodes. Therefore, our scheme effectively reduces the server load in a VPN whose member nodes can dynamically join and leave.

## 4. Evaluation: (2)Cache TTL



### 4.2 Cache TTL

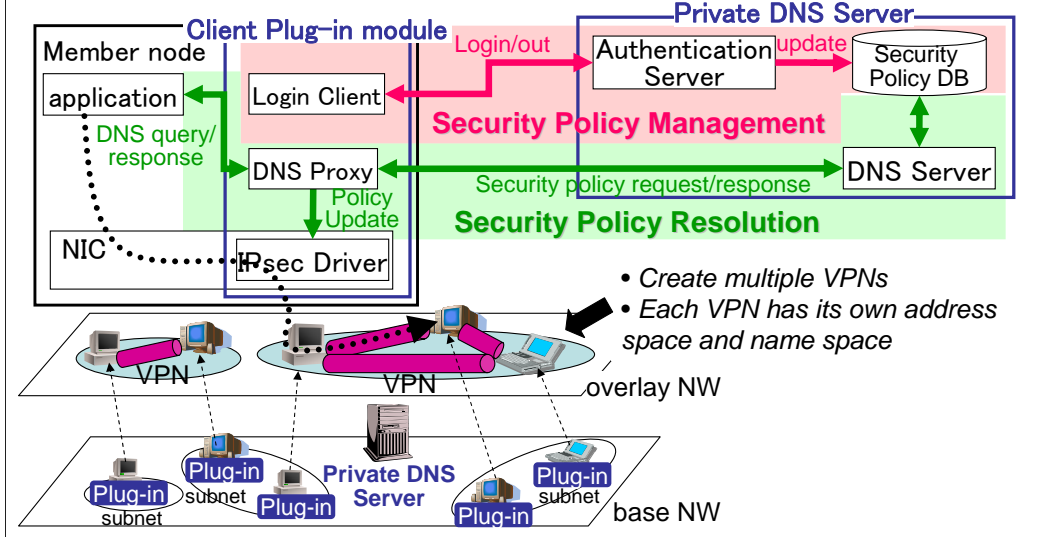
Next, to determine an appropriate cache TTL, we evaluated the server load reduction obtained by caching resolved security policies and the risk of hitting obsolete cache entries.

The simulation model was the same as that used to simulate the load on the policy manager. Nodes dynamically joined and left a VPN, and each node initiated a session with a randomly selected node every ten seconds on average. When a node initiated a session, it first checked whether the corresponding security policy to connect to the selected node had already been cached. If the security policy had not been cached, the node cached the resolved security policy for a duration corresponding to the TTL. If the security policy had been cached and the destination node had been participating in the VPN since the security policy was cached, the “cache hit count” was incremented. Otherwise, if a departure of the destination node had been detected even though the corresponding security policy had already been cached, the cache entry was regarded as obsolete and the “cache fault count” was incremented. We measured the cache hit rate and fault rate by changing the cache TTL value.

The graph in the slide shows the simulation results. Although the longer TTL provides a higher cache hit rate, i.e., a lower server load, a TTL much greater than the residence time significantly increased the risk of hitting obsolete cache entries. We observed that a TTL value of less than the average residence time produced a negligible cache fault rate. By setting the TTL to the average residence time of the VPN nodes, we can effectively reduce the server load while keeping the security policies in each node almost up-to-date.

## 5. Implementation: Overview

- Proposed scheme is realized by
  - Security Policy Management Mechanism**  
maintain security policy DB up-to-date and adjust cache TTL value
  - Security Policy Resolution Mechanism**  
resolves security policy by using DNS name resolution behavior

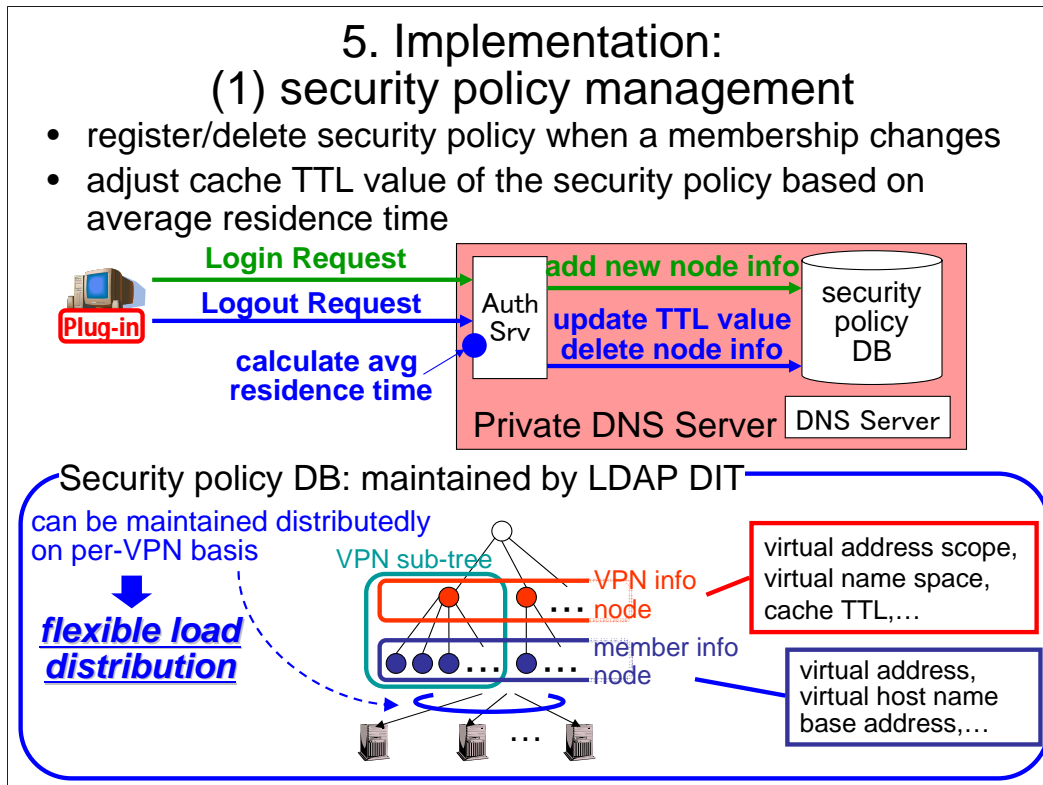


## 5. Implementation

We constructed a prototype gateway-less VPN system that uses the proposed security policy resolution scheme. This slide shows an overview of the system. VPNs are provided as subnets, overlaid on a base network; a VPN is created for each collaborative group. Each VPN has its own virtual IP address space (e.g., 192.168.1/24) and name space (e.g., \*.groupA.private.nec.com). A virtual IP address and hostname are assigned to each member node; they are used to assign destinations by the member nodes when they communicate with each other. In each VPN, secure communication is achieved using IPsec tunnels, established between communicating nodes, and an authentication mechanism that allows only approved nodes to participate in the VPN.

The prototype system is comprised of a management server, which we call a “private DNS (PDNS) server”, and client plug-in modules. The PDNS server has functionality similar to that of the policy manager in the conventional architectures. It is implemented as CGI programs running on a Web server (Apache [6]), and all communications between the server and client plug-in modules are encrypted using the HTTPS protocol, which enables member nodes to access the PDNS server securely and seamlessly without interruptions from firewalls and network address translators. A client plug-in module is installed on each node, enabling the node to join the VPN (login client module), resolve the security policy (DNS proxy module), and establish IPsec tunnels (IPsec driver). The modules support Microsoft Windows OS.

The proposed “on-demand” security policy resolution is performed through the cooperation of the PDNS server and the client plug-in module. Specifically, it is performed using the security policy management mechanism and the security policy resolution mechanism. The former keeps the security policy database up-to-date and adjusts the cache TTL of the security policy. The latter resolves security policies using DNS name resolution behavior. Next, we will describe how these mechanisms are implemented.



## 5.1 Security Policy Management

Security policy management is performed by the authentication server module and the security policy database of the PDNS server.

### • Authentication Server Module

When the authentication server receives a login request from a node, it authenticates the node and assigns a virtual IP address and hostname. Then, the authentication server registers the node information, such as the IP address, hostname, and the base IP address, to the security policy database.

When it receives a logout request, it deletes the node information associated with the node from the database. The cache TTL values are also updated at the same time. The authentication server calculates the node's residence time, re-calculates the average residence time of the VPN, and sets the average residence time to the TTL value.

### • Security Policy Database

The security policy DB maintains the data in a lightweight directory access protocol [7] directory information tree (DIT). As the figure in the slide shows, each sub-tree corresponds to a VPN. A root node of a sub-tree maintains the VPN's configuration parameters, such as virtual address scopes, name spaces, and cache TTL of the security policy. A leaf node, which corresponds to a member node of the VPN, maintains node information, such as the virtual IP address and hostname assigned to the node, and the base IP address.

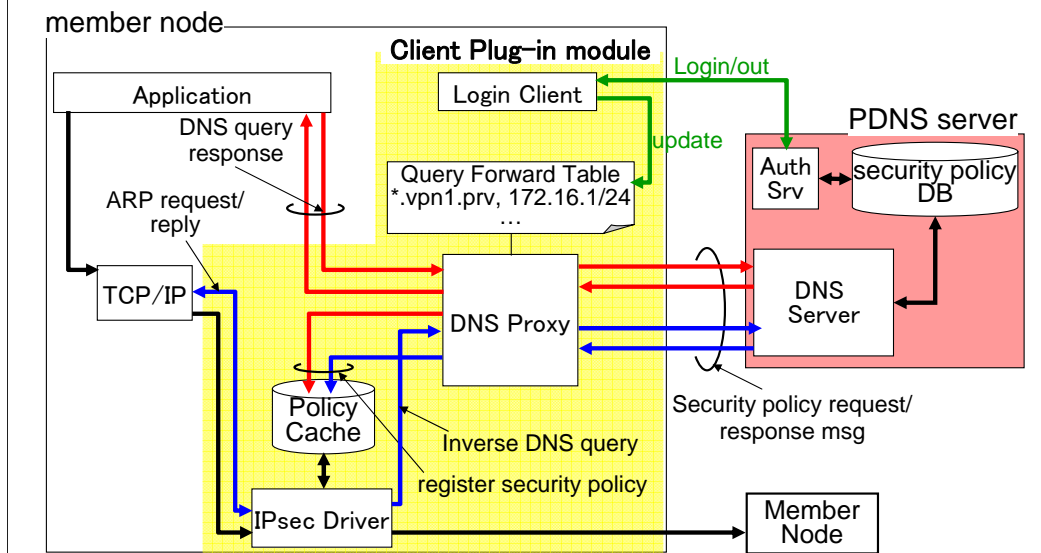
Each sub-tree can be maintained separately using different servers. Therefore, DB access can be distributed on a per-VPN basis, which enables efficient load distribution. For example, it is possible for an operator to maintain a sub-tree that corresponds to the VPN with a large number of members using multiple servers separately from the other parts of the DIT.

## 5.Implementation: (2) security policy resolution

The following events trigger the security policy resolution

- DNS name lookup by an application (→)
- ARP resolution by TCP/IP stack (→)

←An application can designate its destination by an IP address



### 5.2 Security Policy Resolution

Security policy resolution is performed by the DNS proxy module of the clients' plug-in and the DNS server module of the PDNS server. The DNS proxy module behaves as a local DNS server for a node. It intercepts DNS queries and satisfies the security policy requirements to establish a secure tunnel to the destination node, which corresponds to the domain name in the query, by sending a request message to the DNS server module. The DNS server module responds to the request message by sending the requested security policy to the proxy module.

The prototype system supports two events that trigger a security policy resolution: One is DNS name lookup by an application to resolve a destination's virtual IP address from the destination node's virtual hostname (event A). The other is an address resolution protocol (ARP) resolution by the TCP/IP stack of the node's OS to resolve the MAC address from the destination's virtual IP address (event B). This enables appropriate security policy resolution, even for applications like VoIP and video chat, which use non-DNS-based IP address resolution mechanisms, such as the session initiation protocol (SIP) [8].

For each event, security policy resolution is performed as follows.

- Security policy resolution triggered by event A (shown by the red lines in the slide):

A DNS query sent from an application is intercepted by the DNS proxy. The proxy then sends a security policy request message, which designates the required security policy using the virtual host name of the destination, to the PDNS server.

- Security policy resolution triggered by event B (shown by the blue lines in the slide)

When a node starts sending IP packets destined for a virtual IP address, the TCP/IP stack sends out a command that instructs the node to send an ARP request to resolve the MAC address from the destination's virtual IP address. (Because this address belongs to the same subnet address scope as the node's virtual IP address, the TCP/IP stack tries to resolve the MAC address directly, instead of resolving the default gateway's MAC address).

The IPsec driver, on receiving the ARP request, sends an inverse DNS query (i.e., a query to resolve a hostname from an IP address) that designates a resolution target by sending the destination's virtual IP address to the DNS proxy. Then, the DNS proxy sends a security policy request message designating the necessary security policy by sending the virtual IP address to the PDNS server. After the necessary security policy is registered to the policy cache, the IPsec driver notifies the TCP/IP stack of the MAC address, and the stack starts sending packets to the destination.

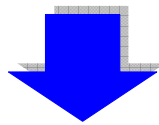
The above approach causes extra ARP resolution latency, and a packet drop could possibly occur before the MAC address is sent to the TCP/IP stack. However, for applications that use TCP as a transport layer protocol, the possibility of packet drops is negligible because the SYN packet waits in the kernel buffer until the SYN timeout (which is much longer than it takes for the policy resolution latency [9]) to elapse. For applications that use UDP, the communication quality degradation caused by the packet drop was negligible in our experiments, in which nodes performed video chat by using Microsoft Windows Messenger.

## 6. Conclusion

We propose

***“On-demand” security policy resolution scheme***  
for gateway-less VPN

- perform pull-base resolution by utilizing DNS name resolution mechanism
- Adopts cache mechanism to eliminate too-many server-client interactions



***reduce policy manager’s load especially when deployed to groups with frequent membership changes***

## 6. Conclusion

We have developed a security policy resolution scheme for scalable gateway-less VPN architectures to support collaborative groups with dynamic membership. By utilizing the DNS name resolution mechanism, the proposed scheme enables nodes to pull only the required security policy from the policy manager on an on-demand basis. When membership in the VPN changes, only the security policy on the policy manager needs to be updated, thus reducing the policy manager’s load, unlike conventional schemes in which policy updates are advertised to all member nodes. The server load is thus effectively reduced, especially for groups in which member nodes frequently join or leave. Furthermore, the proposed scheme uses a cache mechanism to mitigate frequent server-client interactions, which could present a problem for a pull-based resolution scheme. Implementation of our scheme in a gateway-less VPN prototype demonstrated its effectiveness. We plan to implement it in actual networks to evaluate its practical performance.

## Acknowledgments

This work was supported by the Ministry of Public Management, Home Affairs, Posts and Telecommunications.

## Reference

- [1] Freebit, “Emotion Link,” [http://www.freebit.com/english/solution/el\\_ipv4.html](http://www.freebit.com/english/solution/el_ipv4.html).
- [2] SoftEther, “SoftEther,” <http://www.softether.com/us/>.
- [3] A. Gomez, G. Martinez, and O. Canovas, “New Security Services based on PKI,” *Future Generation Computer Systems*, Vol. 19, pp. 251-262, Elsevier Science, Jan. 2003.
- [4] “Dynamic VPN Controller (DVC) Demonstrator Project Report,” Version 1.2, NRNS Incorporated, Canada, Oct. 2002.
- [5] P. Mockapetris, “Domain names - implementation and specification,” RFC1035, Nov. 1987.
- [6] The Apache Software Foundation, “The Apache HTTP Server Project,” <http://httpd.apache.org/>.
- [7] M. Wahl, T. Howes, and S. Kille, “Lightweight Directory Access Protocol (v3)”, RFC2251, Dec. 1997.
- [8] J. Rosenberg et al., “SIP: Session Initiation Protocol”, RFC 3261, Jun. 2002.
- [9] Microsoft, “TCP/IP and NBT configuration parameters for Windows XP,” <http://support.microsoft.com/default.aspx?scid=kb;en-us;314053>