Workshop: Workshop on Computer Networks, ICS 2002

Title: A Service Probing and Channel Establishment Protocol across Multiple Network Address Translation Realms

Abstract - NAT (Network Address Translator) has been widely used to protect private networks and resolve IPv4 address depletion. All hosts behind the NAT have only private IP addresses instead of public IP addresses. Due to the nature of NAT that hides the internal network topology, a service located behind a NAT is unreachable and thus protected to some degree from the outside world. Despite some schemes have been proposed to help a host on a public network access an internal service, it is still difficult to access a service that is protected behind multiple NAT boundaries. In this paper, we propose a protocol that allows an outsider to access services protected behind disparate address realms. With the collaboration of service-probing servers, a bidirectional communication channel can be established between the outsider and the protected service. To release resource occupied by closed or dead sessions, this paper addresses the detection of channels that are no longer used. In addition, a comparison with the related work is also given in this paper.

Authors: Fu-Shen Ho, Shiuhpyng Shieh, Yu-Lun Huang and Kun-Lai Tsai

Address: Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta-Hsueh Rd., HsinChu, Taiwan 30050, ROC

Phone: +886-3-5711437, Fax: +886-3-5724176

Email: {fsho, ssp, ylhuang, kltsai}@csie.nctu.edu.tw

Contact Author: Shiuhpyng Shieh (ssp@csie.nctu.edu.tw)

Keywords – Network Address Translator, NAT and service probing protocol

A Service Probing and Channel Establishment Protocol across Multiple Network Address Translation Realms*

Fu-Shen Ho, Shiuhpyng Shieh, *Senior Member, IEEE*, Yu-Lun Huang and Kun-Lai Tsai

Department of Computer Science and Information Engineering,

National Chiao Tung University,

1001 Ta-Hsueh Rd., HsinChu, Taiwan 30050, ROC

Abstract - NAT (Network Address Translator) has been widely used to protect private networks and resolve IPv4 address depletion. All hosts behind the NAT have only private IP addresses instead of public IP addresses. Due to the nature of NAT that hides the internal network topology, a service located behind a NAT is unreachable and thus protected to some degree from the outside world. Despite some schemes have been proposed to help a host on a public network access an internal service, it is still difficult to access a service that is protected behind multiple NAT boundaries. In this paper, we propose a protocol that allows an outsider to access services protected behind disparate address realms. With the collaboration of service-probing servers, a bidirectional communication channel can be established between the outsider and the protected service. To release resource occupied by closed or dead sessions,

* This work is supported in part by the Ministry of Education, National Science Council of Taiwan and Lee & MTI Center of National Chiao Tung University.

this paper addresses the detection of channels that are no longer used. In addition, a comparison with the related work is also given in this paper.

Keywords – Network Address Translator, NAT and service probing protocol

I. INTRODUCTION

With the rapid deployment of broadband Internet services, depletion of IPv4 address space and protection of enterprise networks have become important issues recently. A long-term solution would be IPv6 [3] with IPSec [7], but it requires a major upgrade to the whole internetworking infrastructure. In contrast, NAT (Network Address Translator) [13][14] and NAT variants by themselves provide a transparent routing and protection solution to end hosts that need to communicate between disparate address realms without changing to end hosts. For these reasons, NAT has been widely deployed by network planners. A NAT device modifies either the source or destination address and maintains states for these updates so that datagrams pertaining to a session can be transparently routed to the right host in either realm. According to current deployment scenarios, NAT can be categorized into two types: *single-level NAT* and *multi-level NAT*. A single-level NAT means address translation is performed once during the delivery of a packet; while a multi-level NAT means an address is translated multiple times before arriving at the destination.

Despite the convenience brought by network address translation, there are also some limitations

[12-14]. Naturally, NAT hides internal network topology from the external world so that no internal service behind NAT is visible to an outsider. Besides, NAT allows only uni-directional (outbound) connections instead of bidirectional (inbound and outbound) connections [12-14], preventing any protected services from being reached from the outside world.

In the last few years, some schemes were proposed to solve the problems caused by NAT, focusing on either service probing or internal service connectivity. SLP (Service Location Protocol) proposed by E. Guttman, et al. [4] was introduced to provide a framework for a host to probe Internet services dynamically. There are four protocols based on the concept of SLP: LDAP (Light Directory Access Protocol) [6][16], DNS (Domain Name System) [10], Sun's Jini [5][17] and Berkeley's SDS (Secure Service Discovery Service) [2]. Unfortunately, compatibility issues with NAT were not considered in the designs of these SLP-based protocols. For internal service connectivity, four schemes were proposed to deal with NAT: port forwarding [8], DNS_ALG (DNS Application Level Gateway) [15], Expanded NAT [9] and RSIP (Realm-Specific IP) [1]. Port forwarding creates a fixed port for each internal service, but it cannot handle applications with dynamic changing ports. DNS_ALG is an extension to DNS, providing internal address mappings only good for one-level NAT deployment. Since Expanded NAT is IP-tunneling based and it modifies the NAT mapping table to support internal server connection ability, it neglects the problem of private IP address conflict. RSIP is intended as a replacement to NAT in which end-to-end communication can be maintained. RSIP is different from NAT in that RSIP requires

each host must be RSIP-aware in layer 3 and layer 4 of the TCP/IP protocol stack, which might be considered difficult in a wide deployment scenario.

In this paper, we propose a service probing protocol for probing services protected behind multi-level NAT, based on the concepts of SLP. The proposed protocol assists an outsider to access the protected services with the helps of a group of service-probing servers, which may or may not belong to the same probing hierarchy. When a match is found, our protocol is responsible for reserving a bidirectional communication path between the two hosts by interfacing with the associated NAT devices to setup the necessary address mapping and filtering rules on the NAT device dynamically.

This paper is organized as follows. In Section II, we present a protocol for probing services across multiple NAT boundaries. In Section III, we discuss the detection of a session that is no longer used. At last, our protocol is compared with other schemes and a conclusion is given in Section IV and V, respectively.

II. SERVICE PROBING PROTOCOL

In this section, we present a protocol for discovering services protected across multiple NAT realms. The protocol comprises three kinds of participants: *user agent (UA)*, *service agent (SA)* and *probe server (PS)*. UA is a process working on the user's behalf to setup communication channels for services. SA is a process working on behalf of one or more services to perform the service advertisement function. PS is

an application that collects service advertisements and exchanges the advertisement information with other cooperative PSs. SA must advertise its service to the PS of its NAT realm upon service startup. In terms of functionally, PS also performs the following two tasks: *service probing* and *channel establishment*. Service probing is a procedure to locate the service requested by the UA. Channel establishment is a procedure to setup a bidirectional communication path between the UA and SA, which is fulfilled by configuring the associated address mapping rules on the NAT devices [11] along the path. Since a PS and NAT device of the same realm have to work together to achieve the channel establishment, they can be integrated into a physical device for the sakes of security and efficiency.

A *probing hierarchy* that forms an administrative group is a rooted tree structure in which each node represents a PS. A typical probing hierarchy is illustrated with Fig. 1. For simplicity, a probe server is denoted by $P[m,(a_1,a_2,...,a_i,...,a_m)]$, where *m* is the level of the PS in the hierarchy, a_i represents the a_i-th node of level *i,* and the sequence $(a_1,a_2,...,a_i,...,a_m)$ is called the *index* of the PS. The hierarchy root is denoted by $P[0,\phi]$ in particular, with a *null* index. In a probing hierarchy, the *common index* of a group of PSs is the index of their closest common ancestor. For example, the closest common ancestor of P[2,(1,1)] and P[2,(1,2)] is P[1,(1)],and thus their common index is (1).

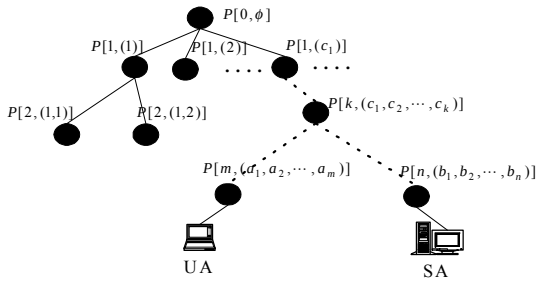

Fig. 1. A typical probing hierarchy.

The PS from which a UA initiates a service request is called the *originating PS*, while the PS to which the SA belongs is called the *terminating PS*. The path from the originating PS to the terminating PS is called the *probing path* of the service request. Since a probing hierarchy is a tree-like structure, the probing path between any two PSs of the same hierarchy is unique. Given any two PSs $P[m,(a_1,a_2,...,a_m)]$ and $P[n,(b_1,b_2,...,b_n)]$, if their common index is $(c_1,c_2,...,c_k)$, where $0 \le k \le m$ and $0 \le k \le n$, we define the probing path (denoted by ω) of a service request initiated from $P[m,(a_1,a_2,...,a_m)]$, to be the sequence of all PSs on this path, which is

$$\omega = \{ P[m,(a_1,a_2,...,a_m)],\ P[m-1,(a_1,a_2,...,a_{m-1})], \ldots,\ P[m-k,(a_1,a_2,...,a_{m-k})],$$

$$P[n-k+1,(b_1,b_2,...,b_{n-k+1})], \ldots,\ P[n,(b_1,b_2,...,b_n)] \}.$$

The number of nodes (denoted by σ) of the probing path ω would be

$$\sigma = (m + n - 2k + 1).$$

Since a probing request flows from the first PS on a probing path to the last PS, the previous PS of any PS on a probing path is called the *upstream PS* of this PS, while the next PS is called the

downstream PS. Similarly, the previous NAT realm of any PS on a probing path is called the *upstream realm* of this PS, while the next NAT realm is called the *downstream realm*.

According to the life cycle of a communication session, the proposed protocol consists of three procedures: *Service advertisement, Channel setup* and *Channel release procedures*.

A. Service Advertisement Procedure

The service advertisement procedure is used for exchanging information about service locations and types. Each PS collects such information in a database called the *service directory*. Typically, entries in a service directory are accessed through the indexes of PSs. Each indexed entry in a service directory is associated with a list of services provided by the SAs belonging to the PS of that index. The construction of a service directory is achieved through the following sub-procedures, which can be activated in an arbitrary order throughout the lifetime of each participant.

1) *Registering a service*: Each SA periodically advertises a service registration message (*RegRqst*) to register its service with the PS in the same NAT realm. *RegRqst* can be either a unicast or broadcast message depending on whether the SA has been configured the address of its PS in advance. The detailed contents of *RegRqst* are not defined in this paper, but they should at least include a service profile with a unique service identifier, the service type and address. If there is any PS that is willing to serve the SA, the PS will respond with a service registration reply message (*RegRply*), informing the SA

that the service has been registered. Meanwhile, the PS that accepts the registration also keeps the service profile in its service directory. If any security policy is enforced, *RegRply* should contain a credential that can be verified by the SA. A PS keeps an expiration timer for each registered service. If the timer of any service expires, the service will be considered unavailable and taken out of the service directory.

2) *Building a service directory*: Each PS in the system periodically advertises a probe advertisement message (*PrbAD*) on all connected NAT realms using a broadcasting or multicasting mechanism. *PrbAD* is a one-way notification message, and it does not require a response. The contents of *PrbAD* are not defined in this paper, but they should include the service directory stored on the PS and the index of the PS. Once a PS receives any *PrbAD* from the neighborhoods, it iterates through all entries in *PrbAD*. If the entry cannot be found in the PS's service directory, it will be added to the service directory. Otherwise, the entry will replace the existing entry in the service directory. The time when the service directory on each PS becomes stable depends on the interval of advertisement. Once a service directory becomes stable, it will stay unchanged until any service change is made.

3) *Finding a probe server*: Each UA learns about the address of the active PS and the service directory by monitoring the *PrbAD* message over the local network on which the UA currently resides. If a UA has the capability to travel among different NAT realms, it can also switch between different PSs dynamically.

B. Channel Setup Procedure

The purpose of this procedure is to help a UA probe the requested service across multiple NAT realms and establish a bidirectional channel hop-by-hop between the UA and SA. Channel establishment can be expressed with the following four steps and shown with bold lines in Fig. 2. Note the SA that provides the service is not involved in this procedure.

Step 1: When a UA wants to query a service, it sends a service request message (*SrvRqst*) to the originating PS, $P[m,(a_1,a_2,...,a_i,...,a_m)]$ (abbreviated as P_{orig}) on UA's NAT realm. *SrvRqst* contains the service profile to be requested. When P_{orig} receives *SrvRqst* from the UA, it searches the requested service and finds out the corresponding terminating PS in the service directory. Then, P_{orig} computes the probing path ω to the terminating PS, $P[n,(b_1,b_2,...,b_n)]$ (abbreviated as P_{term}).

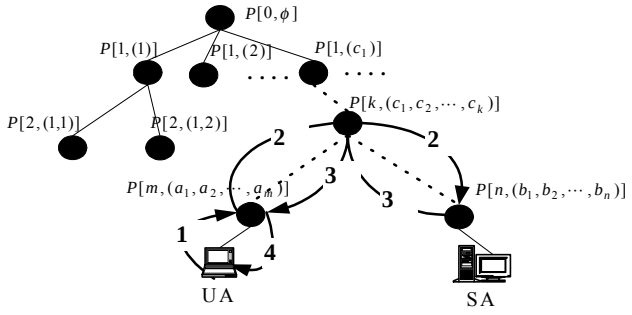

Fig. 2. The channel setup procedure.

Step 2: If the UA is authorized to request the service, P_{orig} instructs the NAT device to allocate a free address on the downstream realm for address translation and setup a NAT mapping entry on the NAT device. Then, P_{orig} initiates a probe request message (*PrbRqst*) to its downstream PS, in which it

contains ω, the service profile to be requested and the source address on the downstream realm for this service request. *PrbRqst* traverses hop-by-hop from P_{orig} to P_{term}. Each PS on ω must also instruct its NAT device to allocate a free address on the downstream realm for address translation and setup a NAT mapping entry for address translation between its own realm and the downstream realm. The source address of the service request contained in *PrbRqst* is taken as the source address in the NAT mapping entry. If any of the nodes on ω fails to reserve the address translation resource, the failed node will return a probe reply message (*PrbRply*) to its upstream PS. In this case, *PrbRply* contains the status code indicating an error of this probe request. The *PrbRply* flows in upstream direction back to P_{orig}. Each node on the return path instructs its NAT device to release the allocated address and delete the temporarily reserved NAT mapping entry.

Step 3: When *PrbRqst* arrives at P_{term}, it verifies if the requested service is still in service at the time being. If the service is available, P_{term} will respond a *PrbRply* with a status code indicating that the request can be fulfilled and the NAT mapping entry has been setup successfully. The *PrbRply* flows in upstream direction back to P_{orig}. If quality of service (QoS) should also be ensured, bandwidth allocation and any other QoS parameter configuration are done in this step.

Step 4: When P_{orig} receives a successful *PrbRply* from its downstream PS, it implies that the bidirectional communication channel between the UA and SA has been setup properly. Then, P_{orig} sends a service reply message (*SrvRply*) back to the UA, indicating that the request has been fulfilled.

Otherwise, if the *PrbRply* shows an error, P_{orig} sends a *SrvRply* with a status code that points out the failure. Upon the reception of *SrvRply*, the UA verifies the status code

C. Channel Release Procedure

This procedure is activated by P_{orig} or P_{term} in one of the following conditions:

1) *Implicit release*: After the session is up, both the originating and terminating NAT devices start to monitor the traffic activity between the UA and the SA. If either one detects that an existing session may no longer be used or idle exceeding expected period of time, it will immediately notify its PS to initiate the channel release procedure.

2) *Explicit release*: SA sends a release request message (*RlsRqst*) to P_{term}, indicating that the session has been closed explicitly. *RlsRqst* contains the profile of the service to be released.

Fig. 3 illustrates the scenario of an implicit-release procedure initiated by P_{orig}. Explicit-release procedure or implicit-release procedure initiated by P_{term} works in a similar way as the procedure by P_{orig}, and is omitted for simplicity. In Fig. 3, two steps are required to release the channel.

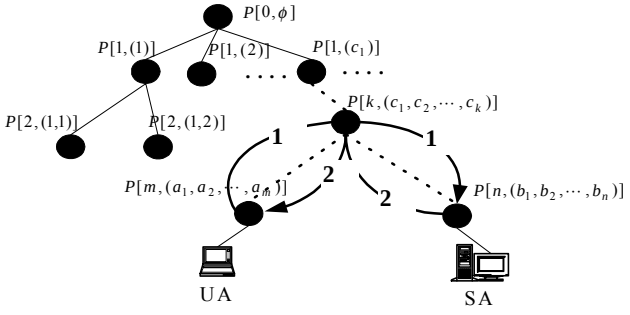

Fig. 3. The implicit channel release procedure.

Step 1: P_{orig} first issues a *RlsRqst*, which is then forwarded along the probing path to P_{term}.

Step 2: When P_{term} receives the *RlsRqst*, it instructs its NAT device to release the allocated address and delete the corresponding NAT mapping entry. Then, P_{term} returns a release reply message (*RlsRply*) to its upstream PS. Each PS on the probing path releases the allocated address and deletes the reserved NAT mapping entry, respectively.

III. DECTECTION OF A CLOSED SESSION

Detection of a closed session that triggers an implicit channel release varies from applications to applications. There is no such a universal algorithm to precisely distinguish between sessions that are idle for an exceeded time or have already been closed. However, to prevent the possibility that channel resource on a NAT device is occupied by unused sessions, we address two practical methods to handle the issues on both UDP and TCP sessions as follows.

1) *Connectionless (UDP) sessions*: UDP-based sessions are always application-specific. We can keep a timer on each active session and retire the expired sessions. Generally, a network administrator can choose a timeout value for each application, depending on the behavior of the application and the infrastructure of the network.

2) *Connection-oriented (TCP) sessions*: A NAT device can detect the termination of each session precisely, for a TCP-based session usually ends with the packet carrying a SYN or RST flag. According

to the suggestion by the authors of NAT [14], a TCP-based session could only be assumed to have been terminated after a period of 4 minutes subsequent to this detection. However, since the UA or SA may be disconnected from the network or closed abnormally, the NAT device cannot just reply on the reception of such flags. A timer mechanism should also be enforced on each active session instead. The selection of a timeout value for a TCP session depends on the infrastructure of the network only, and is application independent.

IV. COMPARISONS

In this section, the proposed protocol is compared with the related work mentioned in the first section and summarized in TABLE I. Two factors are compared: protocol efficiency and NAT compatibility.

TABLE I

THE COMPARISONS

	Numbers of messages for service registration	Numbers of messages for service probing	Single-level NAT compatible	Multi-level NAT compatible
SLPv2	*N/A*	*N/A*	*No*	*No*
DNS_ALG	*N/A*	$\geq 2(n+1)$	*Yes*	*No*
SDS	$\leq (n+1)$	$2(n+1)$	*No*	*No*
Our protocol	*2*	$2(n+1)$	*Yes*	*Yes*

In terms of protocol efficiency, we compare the numbers of messages for service registration and

service probing. For simplicity, we assume that there is exactly one PS in each NAT realm and each PS in the system forms a full *n*-level binary tree with 2^{n+1}-1 nodes. In DNS_ALG, since the DNS database is manually setup by the network administrator, no service registration message can be made dynamically. A DNS query in the tree may exceed *2(n+1)* messages. As for SDS, when a server registers its service, the registration message propagates toward the root node and at most (*n+1)* messages are required. For a service probing, SDS requires *2(n+1)* messages. In SLPv2, since directory agents do not form a tree structure, the number of messages for service registration cannot be estimated. Service probing is done by periodically multicasting the request using the multicast convergence algorithm [4]. The number of registration messages cannot be estimated as well. In our protocol, two messages are required for service registration, and at most *2(n+1)* messages are needed for service probing depending on the locations of the UA and SA. The result is similar to other schemes.

Since NAT compatibility were not considered by SDS and SLP-based protocols, they have difficulty to function well under the networks with NAT. Despite DNS_ALG was designed to work with NAT, it can only survive under the networks with single-level NAT. Instead, our protocol has been designed to be interoperable with single-level as well as multi-level NAT. As a result, our protocol can perform service probing and channel establishment across multiple NAT boundaries without scarifying the efficiency.

V. CONCLUSION

The increasing deployment of home networks and virtual private networks has greatly encouraged the application of NAT. We can foresee in the near future that new users of broadband services may be assigned only private addresses. In this way, the risk of being attacked by a hacker can be reduced. However, NAT also has the difficulty of providing network services of a private network to the public networks. It is desirable to protect the internal hosts of a private network and at the same time provide network services to public networks. To cope with the problem, we proposed a service probing and channel establishment protocol that allows an outsider to access an internal service across multiple network address translators. In our protocol, a probe server is located in each disparate NAT realm, handling service probing requests and interfacing with the associated NAT device to establish a bidirectional channel along the communication path between the host and service.

The protocol comprises three procedures for handling each communication session: service advertisement, channel setup and release procedures. Service advertisement procedure is performed during the lifetime of each participant for knowing each other. Channel setup procedure is performed when an end host issues a service request. After service probing and channel establishment are accomplished in the channel setup procedure, the session activity is monitored. Once the session is considered to be no longer active, the channel release procedure is triggered to tear down the channel. Compared to other service probing protocols mentioned in Section I, our protocol can perform service

probing and channel establishment across multiple NAT boundaries without sacrificing its efficiency.

In an environment that involves privacy and trading, security policies must be enforced on each participant to eliminate the risk of a malicious attack. The enhancement of the proposed protocol to support secure probing and channel establishment is discussed in this paper, and is left as the future work.

REFERENCES

[1] M. Borella, J. Lo, D. Grabelsky, and G. Montenegro, "Realm Specific IP: Framework," Internet RFC 3102, Nov. 2001.

[2] S. E. Czerwinski, B. Y. Zhao, and T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a Secure Service Discovery Service," Proceedings of the Fifth Annual ACM/IEEE Intl. Conference on Mobile Computing and Networking, pp. 24-35, Aug. 1999.

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

[4] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," Internet RFC 2608, Jul. 1999.

[5] E. Guttman, and J. Kempf, "Automatic discovery of thin servers: SLP, Jini and the SLP-Jini Bridge," IECON '99 Proceedings of the 25th Annual Conference of the IEEE, pp. 722-727, Vol.2,

Dec. 1999.

[6] V. Hasller, “X.500 and LDAP Security: A Comparative Overview,” IEEE Network, pp. 54-64, Vol. 13, Issue: 6, Nov./Dec. 1999.

[7] S. Kent, R. Atkinson, “Security Architecture for the Internet Protocol,” Internet RFC 2401, Nov. 1998.

[8] D. Kosiur, *Building and Managing Virtual Private Networks*, John Wiley & Sons, New York, 1998.

[9] E.S. Lee, H.S. Chae, B.S. Park, and M.R. Choi, “An expanded NAT with server connection ability,” TENCON 99, Proceedings of the IEEE Region 10 Conference, pp. 1391-1394, Vol.2, Sep. 1999.

[10] P. Mockapetris, “DOMAIN NAMES - CONCEPTS AND FACILITIES,” Internet RFC 1034, Nov. 1987.

[11] J Rosenberg, Pyda Srisuresh, A. Molitor, Abdallah Rayhan, J Kuthan, “Middlebox Communication Architecture and Framework,” Internet draft, Mar. 2002.

[12] Shiuhpyng Shieh, Fu-Shen Ho, Yu-Lun Huang, and Jia-Ning Luo, “NETWORK ADDRESS TRANSLATORS: Effects on Security Protocols and Applications in the TCP/IP Stacks,” IEEE Internet Computing, pp. 42-49, Vol. 4, Number: 6, Nov./Dec. 2000.

[13] P. Srisuresh and K. Egevang, “Traditional IP Network Address Translator (Traditional NAT),” Internet RFC 3022, Jan. 2001.

[14] P. Srisuresh and M. Holdrege, “IP Network Address Translator (NAT) Terminology and Considerations,” Internet RFC 2663, Aug. 1999.

[15] P. Srisuresh, G. Tsirtsis, P. Akkiraju, and A. Heffernan, “DNS extensions to Network Address Translators (DNS_ALG),” Internet RFC 2694, Sep. 1999.

[16] M. Wahl, T. Howes, and S. Kille, “Lightweight Directory Access Protocol (v3),” Internet RFC 2251, Dec.1997.

[17] J. Waldo, “The JINI architecture for network-centric computing,” Communications of ACM, pp. 76-82, Vol. 42, Issue: 7, Jul. 1999.