CHAPTER 5

Issues with Current VoIP Technologies

This chapter examines some of the issues that are faced by VoIP systems, particularly systems that would be used by carriers for true end-to-end anytime-anyplace connectivity, comparable to what one enjoys today with traditional PSTN voice telephony. We only focus on issues and opportunities that can be addressed by IPv6, namely scalability and end-to-end robustness. The flow mechanism of IPv6 can be employed to manage QoS-specific paths which is critical to VoIP support—the value of flows is already highlighted in MPLS and the deployment/applications it is already experiencing at this time. (To be fair, it should be noted that not all issues faced by VoIP are addressed by IPv6—it is not a complete panacea. Examples here include: general security concerns; equipment interworking; carrier interworking; VoWi-Fi to cellular and/or 3G interworking; and, straightforward and reliable Unified Messaging deployments).

In the sections that follow we first briefly introduce the issue of security (Section 5.1); then we look at the NAT issue (Section 5.2). As a potential solution to some of the NAT problems, we then look in Section 5.3 at Simple Traversal of User Datagram Protocol Through Network Address Translators (STUN). STUN is a lightweight protocol that allows applications to discover the presence and types of NATs and firewalls between them and the public Internet. It also provides the ability for applications to determine the public IPv4 addresses allocated to them by the NAT. STUN works with many existing NATs and does not require any special behavior from them. As a result, it allows a variety of applications to work through existing NAT infrastructure [ROS200301] (however, up to now STUN has not experienced major acceptance/deployment). In Section 5.4 we look at Middlebox Communication (MIDCOM) as a possible other approach to dealing with the issues at hand. Finally, we look at some pragmatic short-term approaches, as embodied in the Session Border Controller (SBC) technology (Section 5.5). None of these solutions are optimal in all factors, hence, the utility of IPv6-based solutions.

5.1 General Enterprise Security Issues

Network and host security continue to be major concerns for enterprise-, institutional-, and service-provider environments. Well-documented recent studies show that cyber attacks continue to remain a substantial threat to organizations of all types. On average, companies experience several dozen attacks per week on their Information Technology resources. About 20% of large companies suffer at least two severe events a year. The challenge to corporate planners just continues to get more onerous. It has been conservatively forecasted that in 2010, around 100,000 new vulnerabilities will be discovered in software applications in that year alone; this will force companies to assess and mitigate one new risk every few minutes of every hour each day.

Considering that each vulnerability instance has the potential to disrupt or bring a company's business to a complete halt, organizations must take risk assessment seriously and determine how each risk will be handled. The increased number of vulnerabilities being discovered also drives up the number of security incidents worldwide and it will increase to a point where 8,000 incidents a week will affect organizations that

have not properly addressed and mitigated their risks. It is estimated that the worldwide financial impact of malicious code is around $100 B year. Beyond the original venue of proving technical bravado, in the recent past attacks have been aimed at stealing customer data, obtaining proprietary information, and deliberately hampering a corporation's ability to do business [POL200401].

If a company loses its information technology (computer and/or voice/data networking) resources for more than a day or two, the company may well find itself in financial trouble. Obviously brokerage firms, banks, airports, medical establishments, and homeland security concerns would be impacted faster than, say, a manufacturing firm or a book publishing firm. However, the general concern is universal. If a company is unable to conduct business for more than a week, the company may well be permanently incapacitated. Therefore, there is a clear need to protect the enterprises from random, negligent, malicious, or planned attacks on its Information Technology resources. As more and more companies send their IT business abroad under the rubric of "outsourcing," the potential IT (and, hence, corporate) risks are arguably growing at a geometric pace; these risks can have ultimate negative implications, particularly in view of cumulative exposures to risks which, in the aggregate, take on nontrivial probability.

Many companies are (now) shifting to a highly mobile work force. To support this mobility firms are upgrading their network architectures to support remote workforces. Mobile users need access to centrally located applications and data over the Internet; voice is also an issue. This, once again, raises the issue of security.

5.1.1 Typical Enterprise Network Approaches

Firewalls are a basic mechanism to support perimeter security, even if by themselves they tend to be inadequate. See Figure 5.1 for a typical environment. Firewalls provide a method of guarding a private network by analyzing the data leaving and entering the intranet. Typically they are implemented as a network appliance (dedicated/standalone hardware), although it can also be a just a software program (for example, for a PC client). [CSO200501]. The majority of packet-inspection firewalls are designed to secure and apply policy to the transport level. Firewalls range in functionality from basic protocol/port inspection, to stateful session-oriented packet inspection, to sophisticated application-layer proxy firewalls. A typical firewall may support the following functions: packet filtering, object grouping, proxy services, URL filtering, stateful inspection, and inline authentication (with or without access to a RADIUS (remote access dial-in user service) server. Firewalls can also provide network address translation, so the actual IP addresses of devices inside the firewall stay hidden from public view; but this is precisely one of the issues of concern for end-to-end connectivity.

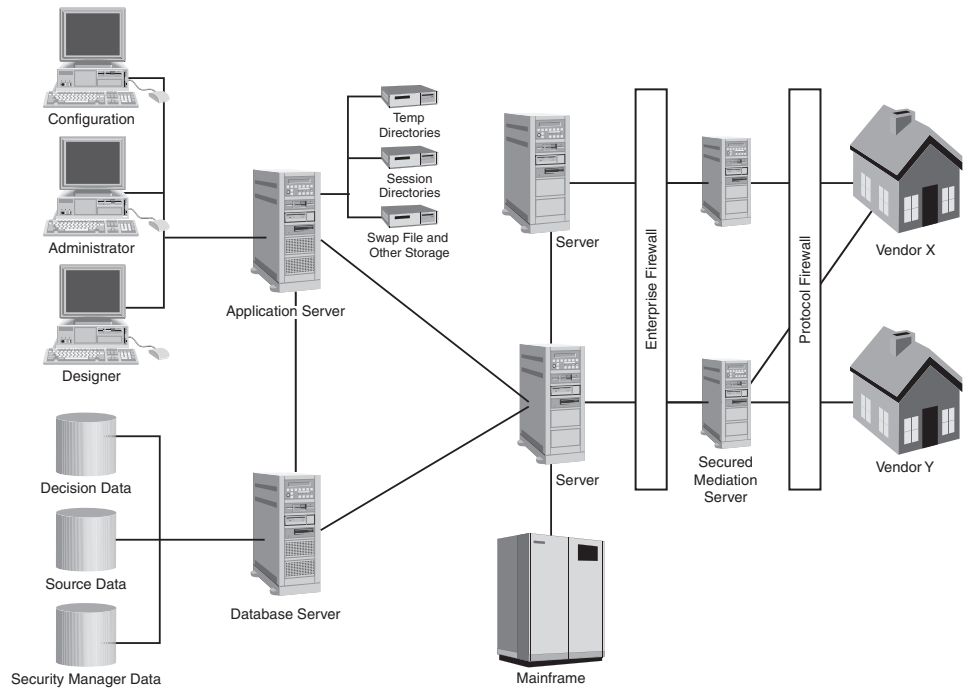


Figure 5.1: Typical firewall environment.

Most companies implement security in layers. The layering can be in terms of domains or in terms of assets categories. It is not effective to rely on a single point-of-protection when addressing the panoply of threats that can impact an IT environment; robust information security requires a multilayered approach.

Companies typically see the environment as being comprised of the following zones (also known as domains). (See Figure 5.2, which depicts both a logical view and an example of a physical view):

Externally-Controlled Zone (ECZ) (such as a particular extranet or 3rd-party environment with an established business relationship): Here the physical access, the IT administration, and the security authority are controlled by a third party.

Uncontrolled Zone (UZ) (such as the Internet and also carrier networks): No established business relationship exists where the firm can assess the security of the environment. Here the physical access, the IT administration, and the security authority are basically unknown.

Controlled Zone (CZ): Network point (zone) where all inbound and outbound communications are mediated (such as the firewall complex). Here the physical access, the IT administration, and the security authority are controlled by the firm in question. This domain separates the ECZ and UZ from the Restricted Zone (typically the intranet) of the firm.

Restricted Zone (RZ): Here the physical access, the IT administration, and the security authority are controlled by the firm in question. Access is granted only to authorized/authenticated users or systems.

Secure(d) Zone (SZ): Network location (zone) that provides isolation from the RZ. This zone may contain more critical assets such as the firm's data warehouse, the Directory, or specialized applications (such as financials, payroll, etc.). Here the physical access, the IT administration, and the security authority are controlled by the firm in question.

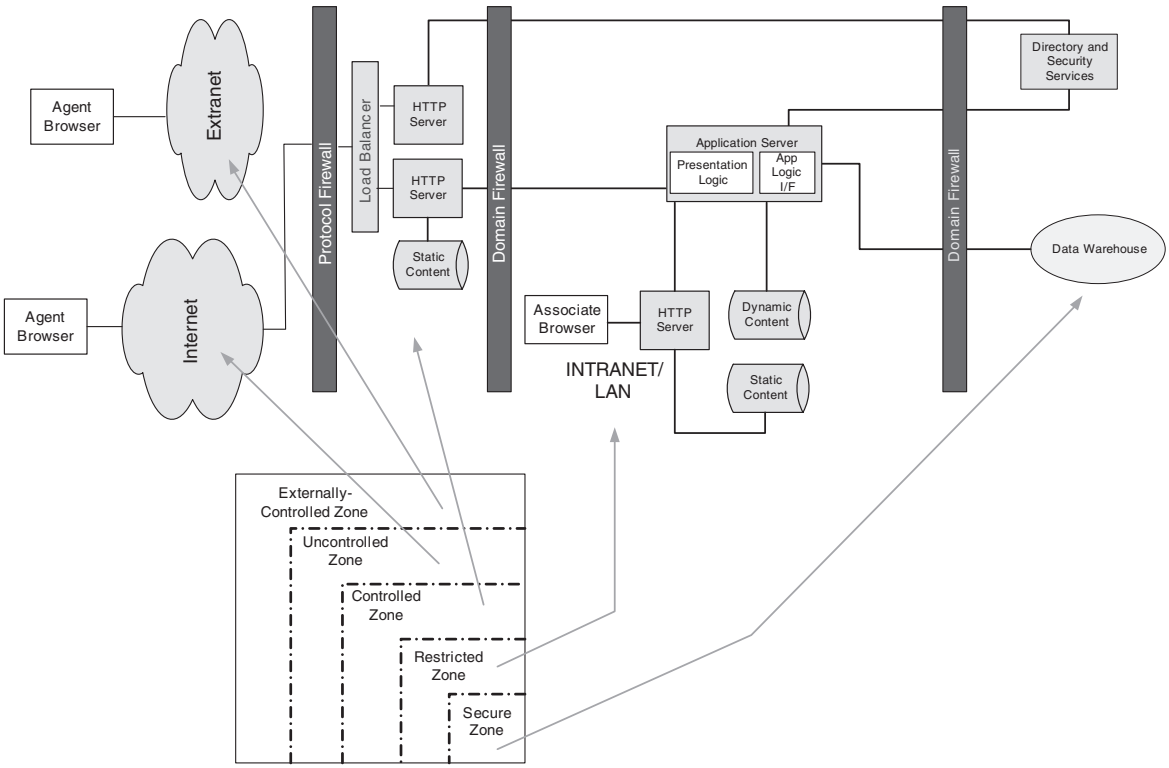


Figure 5.2: Layered security apparatus for typical enterprise environment.

It is also useful to look at layers from an asset category perspective. One example of this is Microsoft's Defense-in-Depth Model, as shown for illustrative purposes in Figure 5.3 [MIC200501].

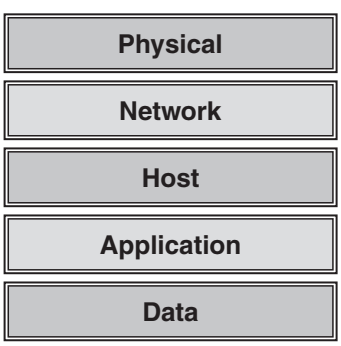


Figure 5.3: Asset category layering per Microsoft's Defense-in-Depth Model.

A very basic firewall glossary as included in Table 5.1 (for a more extensive glossary the reader may refer to [MIN200601]).

Most corporations today address security with a number of technical solutions ranging from login/password, hardware tokens, and RADIUS servers for authentication to Virtual Private Networks (VPNs) for data encryption; hardware (appliance) firewalls at corporate locations for data packet filtering; antivirus software on remote PCs; and encrypted storage (e.g., per the new IEEE standard P1619) [POL200401]. Hardware firewalls (routers and/or appliances), generally protect the corporate network from external attacks but cannot provide protection against attacks originating from within the corporate network (as noted above, however, the Secure Zone (Domain) is delimited by firewalls that are inside of the corporate intranet itself). As noted, increasingly enterprises make use of a "layered" security approach. While authentication mechanisms ensure user/machine authorization and VPNs ensure data privacy in transit, the conventional security tools (e.g., hardware firewalls and antivirus software) cannot fully protect the environment. Malicious code, such as "spyware" can use peer-to-peer file sharing, instant messaging, and file downloading as a vehicle and enter the corporate network to create damage or hog network bandwidth. These are the reasons why XML firewalls (which inspect deep into the transmitted text) can be useful [POL200401].

TCP/IP-based networking uses the TCP-Port apparatus to identify the protocol and/or applications with which a given TCP session should be associated. Firewall technology is very much dependent on this arrangement for proper functioning (other/supplementary techniques such as specifying an IP address or IP address range are also utilized). Two general observations are useful:

- Applications using TCP are easier to manage through a firewall than applications using UDP;
- Protocols/applications that have a smaller range of allowed ports are easier to manage through a firewall than applications using a larger range—those using a single port are the easiest of all.

As it can be seen in Table 5.1, RTP and H.323 have some wire ranges making VoIP based on these protocols something of a challenge (the RTP issue is the same whether one uses SIP or H.323).

In the context of "layered" security, it should be mentioned that many organizations end up using the mechanism of NAT as part of the "toolkit" of available techniques by providing what some call *security through obscurity*. This entails keeping outside entities "unaware" of what the address of internal devices (servers, etc.) is, so that these entities cannot then launch a direct attack against said devices (for example, via a TELNET or a specifically-targeted flow of PDUs and-the-like). Clearly, NAT is a means-to-an-end; hence, if every device has a globally-unique address as in IPv6, then other methods will have to be put in place to provide a layer of security comparable to that provided by the previous state of "obscurity."

Figure 5.4 [ISL200501] depicts today's security environment as compared to what is possible/desirable in an IPv6 future state. The new (NAT-free) security mechanisms facilitate end-to-end connectivity, mobility, and collaboration, under a VoIP and/or 3G wireless environment in the coming years. Today's environment is very different, as discussed in the section that follows.

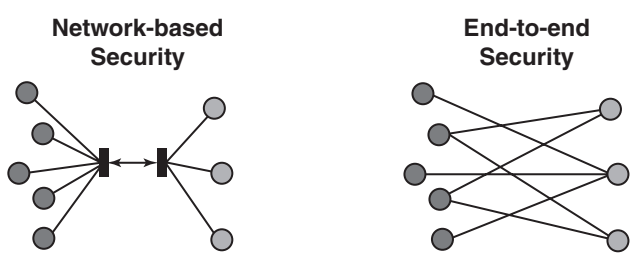


Figure 5.4: End-to-end security environment in IPv6.

Table 5.1: Basic security glossary.

Demilitarized Zone (DMZ)	*(We prefer the expansion "demarcation zone.") An area of an intranet that is a barrier, or a buffer, between a company's internal network and resources connected to the network, and the outside public network. That portion of the intranet-to-extranet or intranet-to-Internet interface apparatus that supports a highly constrained access environment. An area between the hostile Internet and protected services; may be implemented as a Layer 2 switch that support a number of Ethernet-attached devices "sandwiched" between a front-end and a back-end firewall. The purpose of the DMZ is to prevent external users from getting direct access to a server or other corporate IT resources. A DMZ is usually comprised of routers, packet filters, firewalls, proxies, and/or mediation devices.* *A neutral zone, or buffer, that separates the internal and external networks. The DMZ usually exists between two firewalls. External users can access servers in the DMZ, but not the computers on the internal network. The servers in the DMZ act as an intermediary for both incoming and outgoing traffic [BRA200501].* *The DMZ designates the area of protection that lies between the corporate computing environment and the Internet or publicly-accessible network. The DMZ is typically where the firewalls, gateways, application proxies, and other protective computing devices are connected, and employs protective software such as filtering and intrusion detection applications.*
Filter	*Packet matching information that identifies a set of packets to be treated a certain way by a middlebox (security mediation device). A set of terms and/or criteria used for the purpose of separating or categorizing. This is accomplished via single- or multifield matching of traffic header and/or payload data. 5-Tuple specification of packets in the case of a firewall and 5-tuple specification of a session in the case of a NAT middlebox function are examples of a filter [SRI200201].*
Firewall	*A method of guarding a private network by analyzing the data leaving and entering. Typically implemented as a network appliance (dedicated/standalone hardware), although it can also be a just a software program (for example for a PC client.) [CSO200501]. The majority of packet-inspection firewalls are designed to secure and apply policy to the transport level. Firewalls range in functionality from basic protocol/port filtering devices to stateful session-level packet-inspection systems and sophisticated application-layer proxy firewalls. Firewalls can also provide network address translation, so the actual IP addresses of devices inside the firewall stay hidden from public view.* *A policy-based packet filtering middlebox function, typically used for restricting access to/from specific devices and applications. The policies are often termed Access Control Lists (ACLs) [SRI200201].* *Includes four basic types: (1) Application-layer gateway; (2) Stateful-inspection firewall at the Session Layer; (3) Circuit-level gateway at the Network Layer; and (4) Packet-filtering firewall. Firewalls form the fundamental gateway that controls (at different layers of the OSI protocol stack) traffic entering and leaving the network, and all security issues of this type (such as Denial of Service attacks) come under this heading [LIG200501].* *Packet-filtering firewalls use rules based on basic information, such as a packet's source, destination, or port, to determine whether or not to allow it into the network. More advanced stateful packet-filtering firewalls have access to more information from which to make their decisions. Stateful firewalls examine related inbound-outbound traffic for expected/predicted patterns.)* *Proxy firewalls that look at content and can involve authentication and encryption can be more flexible and secure but also tend to be slower. Although firewalls require configuration expertise they are a critical component of network security [INF200501], [CSO200501].*
Layer 2	*The protocol layer below Layer 3 (that therefore offers the services used by Layer 3). Forwarding, when done by the swapping of short fixed length labels, occurs at layer 2 regardless of whether the label being examined is an ATM VPI/VCI, a frame relay DLCI, or a Multiprotocol Label Switching (MPLS) label.*
Layer 2 VPN (L2VPN)	*(aka L2 VPN) Three types of L2VPNs are currently defined [AND200501]: Virtual Private Wire Service (VPWS); Virtual Private LAN Service (VPLS); and IP-only LAN-like Service (IPLS).*
Layer 3	*The protocol layer at which IP and its associated routing protocols operate.*
Layer 3 Security Mechanisms	*Encryption mechanisms such as IPsec or Multilayer IPSec (ML-IPsec).*

Layer 3 VPN (L3VPN)	*(a.k.a L3 VPN) An L3VPN interconnects sets of hosts and routers based on Layer 3 addresses; see [CAL200301].*
Middlebox	*A middlebox is a network intermediate device (in IETF parlance) that implements one or more of the middlebox services. A NAT middlebox is a middlebox implementing NAT service. A firewall middlebox is a middlebox implementing firewall service. Traditional middleboxes embed application intelligence within the device to support specific application traversal. Proposed middleboxes supporting the Middlebox Communications (MIDCOM) protocol, as defined in RFC 3303, will be able to externalize application intelligence into MIDCOM agents. In reality, some of the middleboxes may continue to embed application intelligence for certain applications and depend on MIDCOM protocol and MIDCOM agents for the support of remaining applications [SRI200201].*
Proxy	*An intermediary program (system) that acts both as a server and as a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A software agent that acts on behalf of a user, typical proxies accept a connection from a user, make a decision as to whether or not the user or client IP address is permitted to use the proxy, perhaps does additional authentication, and then completes a connection on behalf of the user to a remote destination [INF200501].* *An intermediate relay agent between clients and servers of an application, relaying application messages between the two. Proxies use special protocol mechanisms to communicate with proxy clients and relay client data to servers and vice versa. A Proxy terminates sessions with both the client and the server, acting as server to the end-host client and as client to the end-host server. Applications such as FTP, SIP, and RTSP use a control session to establish data sessions. These control and data sessions can take divergent paths. While a proxy can intercept both the control and data sessions, it might intercept only the control session. This is often the case with real-time streaming applications such as SIP and RTSP [SRI200201].* *May include a function that replaces the IP address of a host on the internal (protected) network with its own IP address for all traffic passing through it.*
Proxy Firewall	*Unlike packet-filtering, this type of firewall does more than simply block port access. Instead, it acts as a proxy server, processing access requests on behalf of the network on which it is located. This protects individual computers on the network because they never interact directly with incoming client requests [CSO200501].* *Firewalls that look at content and can involve authentication and encryption can be more flexible and secure but may require more processing power [INF200501], [CSO200501].*
Proxy Servers	*Specialized application or server programs that run on a firewall host or on a dedicated appliance: either a dual-homed host with an interface on the internal network and one on the external network, or some other bastion host that has access to the Internet and is accessible from the internal devices. These programs take users' requests for Internet services (such as FTP and Telnet) and forward them, as appropriate according to the site's security policy, to the actual services. The proxies provide replacement connections and act as gateways to the services. For this reason, proxies are sometimes known as application-level gateways. Proxy services intervene, often transparently, between a user on the inside (on the internal network) and a service on the outside (on the Internet). Instead of talking to each other directly, each talks to a proxy. Proxies handle all the communication between users and Internet services behind the scenes. To the user, a proxy server gives the appearance that the user is dealing directly with the real server. To the real server, the proxy server presents the illusion that the real server is dealing directly with a user on the proxy host (as opposed to the user's real host). Proxy servers have two main purposes:* ***Improve Performance:** Proxy servers can improve performance for groups of users. This is because it saves the results of all requests for a certain amount of time. Consider the case where both user X and user Y access the World Wide Web through a proxy server. First user X requests a certain Web page, say Page 1. Sometime later, user Y requests the same page. Instead of forwarding the request to the Web server where Page 1 resides, which can be a time-consuming operation, the proxy server simply returns the Page 1 that it already fetched for user X. Since the proxy server is often on the same network as the user, this is a much faster operation.* ***Filter Requests:** Proxy servers can also be used to filter requests. For example, a company might use a proxy server to prevent its employees from accessing a specific set of websites.*

<table>
<tr><td>Proxy Services</td><td>Proxy services intervene, often transparently, between a user on the inside (on the internal network) and a service on the outside (on the Internet). Proxy services are effective only when they are used in conjunction with a mechanism that restricts direct communications between the internal and external hosts. Dual-homed hosts and packet filtering are two such mechanisms. If internal hosts are able to communicate directly with external hosts, there is no need for users to use proxy services, and so (in general) they will not; such bypass, however, is typically not in accordance with an organization's security policy.

A proxy service requires two components: a proxy server and a proxy client. In this situation, the proxy server runs on the dual-homed host. A proxy client is a special version of a normal client program (i.e., a Telnet or FTP client) that talks to the proxy server rather than to the "real" server out on the Internet; in addition, if users are taught special procedures to follow, normal client programs can often be used as proxy clients. The proxy server evaluates requests from the proxy client and decides which to approve and which to deny. If a request is approved, the proxy server contacts the real server on behalf of the client (thus the term "proxy"), and proceeds to relay requests from the proxy client to the real server, and responses from the real server to the proxy client. In some proxy systems instead of installing custom client proxy software, one employs standard software, but set up custom user procedures for using it. A proxy service is not a firewall architecture; proxy services are used in conjunction with a firewall architecture.</td></tr>
<tr><td>Proxying</td><td>Approach that involves mediating a connection at an intermediate point. In this case the TCP connection is not between the client and the (application) host, but from the client to the intermediate proxy-server/gateway. In turn, the proxy will decide (based on some criteria) if/where a companion session to the ultimate (application) host needs to be established. Proxy servers can also be used to filter requests.

Companies use proxy servers to improve performance (through caching Web pages and graphics), to filter requests to certain sites, to make sure that only certain users can get to the Internet, or as a way of accounting for Web use (logging sites that users visit). Most proxy servers can perform all of these tasks.</td></tr>
<tr><td>TCP Ports</td><td>Transport layer end-to-end protocol identifiers of traffic being carrier in a network. Ports of interest to VoIP include (but are not limited to):

<table>
<tr><td>H.323 RAS</td><td>TCP 1719</td><td></td></tr>
<tr><td>H.323 (H.225)</td><td>TCP 1720</td><td>GW → CM, Call Setup</td></tr>
<tr><td>MGCP</td><td>TCP 2427/2428</td><td>GW → CM, Call Setup</td></tr>
<tr><td>RTP</td><td>UDP 16384–32767</td><td>Bearer Channel</td></tr>
<tr><td>SIP</td><td>TCP 5060</td><td></td></tr>
<tr><td>Skinny Client</td><td>TCP 2000</td><td>Call Setup and Control</td></tr>
<tr><td>Skinny GW (Digital)</td><td>TCP 2002</td><td>Call Setup and Control</td></tr>
</table>
GW = Gateway; CM = Call Manager (server)

(long lists of well-known ports are published by the IETF)</td></tr>
<tr><td>XML Firewall</td><td>A (relatively) new type of firewall intended to secure XML messages and Web Services (WS). Traditional firewalls are not designed to understand/interpret the XML message-level security and they cannot defend against new XML message-based attacks. The majority of packet-inspection firewalls are designed to secure and apply policy to the transport level, therefore they generally do not scan for content in Simple Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI), SAML or other Web services protocols. The difference between an XML firewall and other firewalls is that much of the features in an XML firewall exist at the application layer and within the data payload or content, as opposed to the transport and session layer. Many modern XML firewalls act like high-performance proxies: they can approach wire speed performance by offloading crytpo and XML validation functions to dedicated hardware (features such as message routing, encryption and forwarding are somewhat of a commodity). In this role, the XML firewall performs security services such as Authentication, Authorization, Auditing (AAA) and XML validation at a message level. The features are a separation of message-level security from transport-level security (these XML features do not act as transport-level connection security such as done in SSL) [WRE200401].</td></tr>
</table>

5.1.2 Typical Enterprise Network VoIP Security/Integration Approaches

This section briefly describes typical Best-in-Class designs for enterprise VoIP/converged networks. It highlights some of the architecture/design issues. This discussion is loosely based on reference [KUI200501].

Security issues affecting VoIP networks include, but are not limited to, the following: Toll fraud, packet/call eavesdropping, viruses, worms, Denial-of-Services, TCP vulnerabilities, Layer 3 exploits, rogue device in the network, man-in-the-middle issues, DHCP spoofing, DHCP starvation, and DNS spoofing.

A rogue device has access to the voice stream (packets and/or session/call) between the two communicating endpoints. Products/programs have appeared (e.g., Voice Over Misconfigured Internet Telephones (VOMIT)) that facilitate such eavesdropping by assembling tcpdumps of conversations into .WAV files. A rogue device (an unauthorized device that has been able to inject itself into the network) can undertake theft of telephone service; rogue voice gateways can cause even more harm.

Countermeasures include all IP-based security mechanisms such as (the relatively weak) VLAN switch/port management methods, Layer 3 firewalls, proxies, intrusion prevention systems, encryption/tunneling, H.245 security, certificates, authentication/RADIUS/IEEE 802.1 services, physical hardening.

Figure 5.5 depicts a typical corporate VoIP arrangement, somewhat similar to the figures included in Chapter 1 (keep in mind that a carrier arrangement would be quite different). As can be seen from this figure, there are several places (at least at the firewall locations) where the NAT/firewall issues can be problematic.

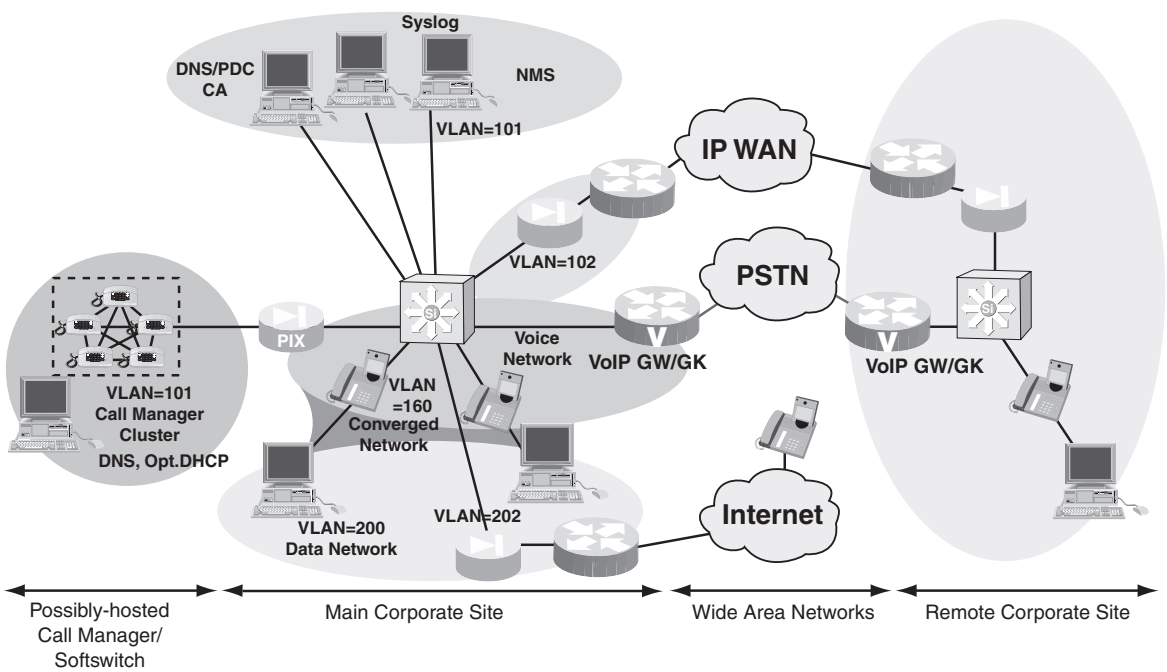


Figure 5.5: A typical corporate VoIP arrangement.

VoIP security is built in layers, as was the case for the more general intranet discussion earlier in the chapter. Note the firewall arrangement facing the IP WAN as well as facing the call manager/softswitch cluster (which can be co-located or can be at a hosted network/ASP-resident site). Again, these are impacted by NAT.

Security related to the call manager/softswitch cluster centers on hardened Operating System (OS) (such as Linux and/or a high-quality maintenance process on other less reliable OSs), IPSec tunneling (for remote/hosted arrangements), and Host-based Intrusion Prevention system. IPSec in also affected by NAT.

Security related to the firewall connecting to the call control manager deals with Access Control Lists, control of source addresses, and proper filtering (e.g., to allow only call control, directory/LDAP functions, and network management). NAT impacts the overall setup.

Connection to the outside world (Internet) is handled over a Layer 3 VPN mechanism. Network Intrusion Detection Systems/Network Protection Systems are typically used.

Endpoints (clients) use separate voice and data VLANs (in support of the already mentioned relatively weak VLAN switch/port management mechanism), authentication, and encryption (especially if over a wireless LAN or VPN, here for a softphone.) Endpoint encryption, particularly for VoWi-Fi, is still evolving in terms of broad vendor support.

The campus network typically makes use of the normal Layer 3/Application Layer firewalls, and IP filters between voice and data. NAT use should be minimized.

Most deployments today make use a distinct VLAN for voice and a VLAN for data traffic, as already mentioned and further depicted in Figure 5.6. This is done for administrative, QoS, and pseudo-security[23] considerations (the voice VLAN is called an *auxiliary* VLAN). However, firms want to use the same access, core, and distribution layers for the two segments in order to be able to make the claim and gain the operational and financial advantage of a converged network (see Figure 5.7.) This is supported by mechanisms such as Layer 3 access control and stateful firewalls (firewalls that examine related inbound-outbound traffic for expected/predicted patterns.)

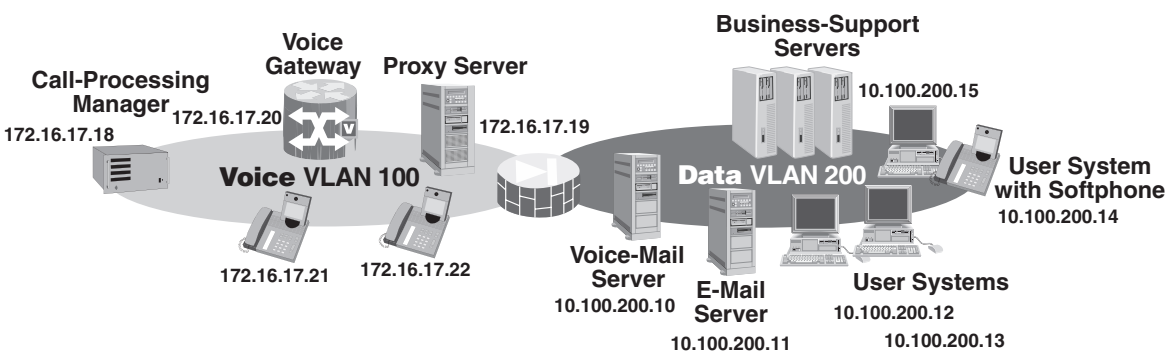


Figure 5.6: Use of two VLANs—typical 2G VoIP arrangement.

[23] Pseudo-security is a term we use to describe an environment or technology with a weak (and/or false sense of) protection.

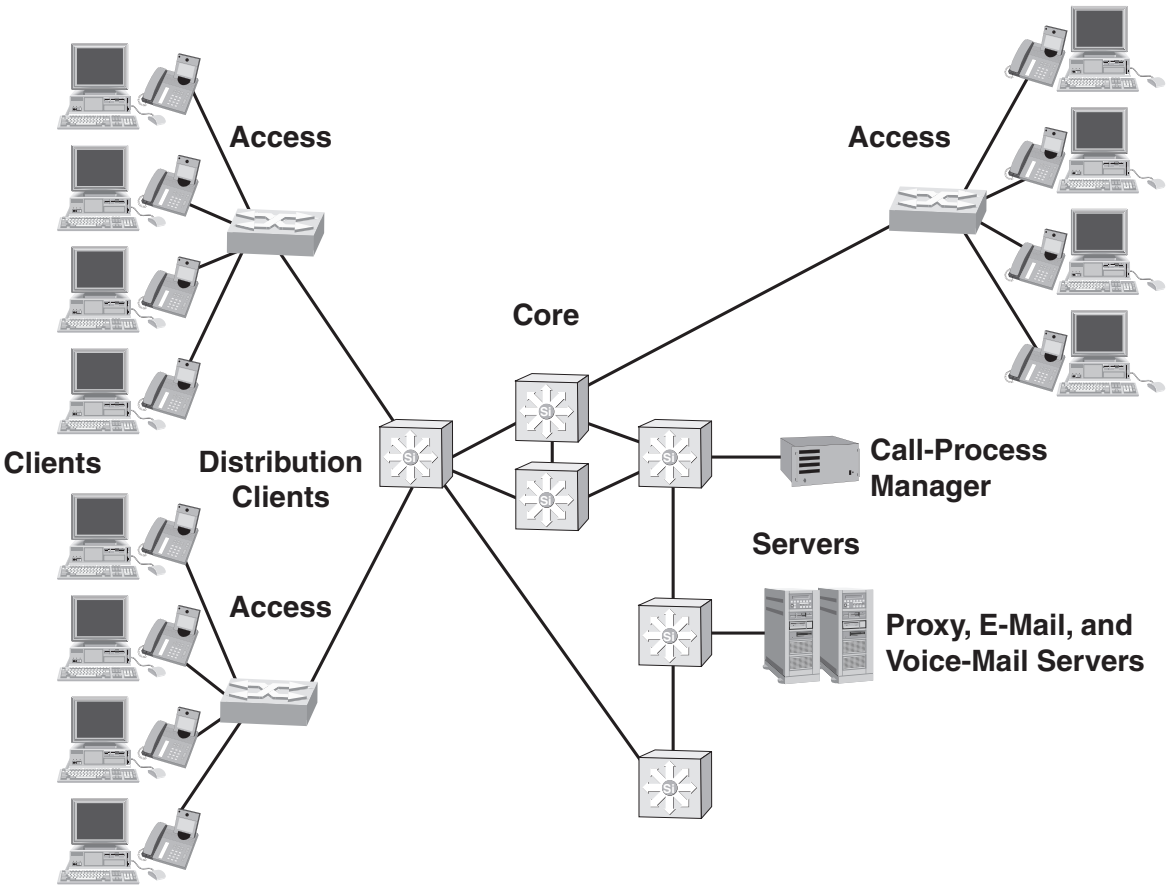


Figure 5.7: Converged intranet.

IP phones typically support access to both segments (IP phones have a "data port"/Ethernet for the local PC to connect—this uses a single Ethernet cable to the desk, often with in-line power.) Planners need to make sure that the phone supports separation of the two segments. However, firms should not rely solely on VLANs for separation: to support more robust security one needs to make provision for Layer 3 filtering between the data and voice VLANs.)

A stateful firewall between the two VLAN segments is typically used to manage the data/voice VLAN interaction. The stateful firewall provides dynamic access and mitigation against TCP connection starvation, UDP flooding, and spoofing attacks.

As seen in Figure 5.6 in 2G VoIP one typically makes use of a private address space (RFC 1918) for the data and for the voice VLAN segments. The partitioned addressing facilitates filtering and recognition. The approach in RFC 1918 does not support routability, but this can be utilized to reduce the likelihood of reconnaissance scans even if NAT happened to be misconfigured. Spoof-mitigation filtering addresses the identity issue (that nodes are who they claim to be) in local segments.

Related to the end-points, blocking PC access to the voice VLAN at the VLAN switch (even if the PC has physical access to the network or to the Layer 2 switch) greatly reduces the possibility of eavesdropping

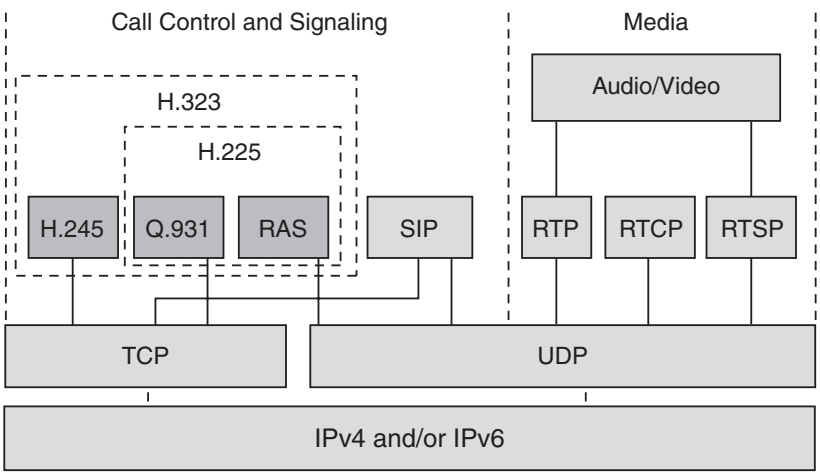
attacks (such as those that may be unleashed with VOMIT-like products); techniques also exist to prevent man-in-the-middle attacks or traffic interception. Access Control Lists (ACLs) can be used to prevent directed-TCP attacks. DHCP snooping stops DHCP spoofing and starvation attacks. Digitally-signed firmware and configuration files on clients mitigate security liabilities. Certificates can be used to prevent rogue call managers, gateways, and phone set insertion (particularly in a VoWi-Fi environment). Finally, encryption prevents interception. Similar techniques can be used to protect the servers that support VoIP, e.g., Call Managers, Gateways, Gatekeepers, etc.

5.1.3 Firewall Issues for VoIP

As noted earlier, firewalls are a basic mechanism to support perimeter security; packet-inspection firewalls are designed to apply policy to the transport level. As discussed, firewalls range in functionality from a basic stateful packet-inspection engine to sophisticated application-layer proxy firewalls; firewalls can also provide network address translation. TCP/IP-based networking uses the TCP/UDP-Port apparatus to identify the protocol and/or applications with which a given TCP session should be associated. As we already observed, two general observations are useful in a networking context that are also useful in a VoIP context:

- Applications using TCP are easier to manage through a firewall than applications using UDP;
- Protocols/applications that have a smaller range of allowed ports are easier to manage through a firewall than applications using a larger range—those using a single port are the easiest of all.

Figure 5.8 depicts the protocol stacks of interest to VoIP. As it can be seen in Table 5.1, RTP and H.323 have some wide ranges making VoIP based on these protocols something of a challenge (the RTP issue is the same whether one uses SIP or H.323.)



H.323 Version 3 and 4 supports H.245 over UDP/TCP, Q.931 over UDP/TCP, and RAS over UDP.
SIP supports TCP and UDP.

Figure 5.8: VoIP protocol stack.

We limit the rest of this discussion to SIP. Some of the NAT-related issues are highlighted next. As we discussed in Chapter 3, the Via field in SIP indicates the path taken by the SIP request under discussion up to the present point. This prevents request looping and ensures replies take the same path as the requests, which, in principle, assists in firewall traversal and other unusual routing situations [HAN199901].

According to [HAN199901] (on which the discussion that follow is based) if a SIP proxy server[24] forwards a SIP request, it must add itself to the beginning of the list of forwarders noted in the Via headers. The Via trace ensures that replies can take the same path back, ensuring correct operation through compliant firewalls and avoiding request loops. On the response path, each host must remove its Via, so that routing internal information is hidden from the callee and outside networks. A proxy server must check that it does not generate a request to a host listed in the Via sent-by, via-received, or via-maddr parameters (the maddr parameter provides the server address to be contacted for this user, overriding the address supplied in the host field; this address is typically a multicast address but could also be the address of a backup server.)

Hence, the client originating the request inserts into the request messages a Via field containing its host name or network address and, if not the default port number, the port number at which it wishes to receive responses. (Note that this port number can differ from the UDP source port number of the request.) A fully-qualified domain name is typically used. Each subsequent proxy server that sends the request onwards must add its own additional Via field before any existing Via fields. A proxy that receives a redirection (3xx) response and then searches recursively, must use the same Via headers as on the original proxied request. A SIP proxy should check the top-most Via header field to ensure that it contains the sender's correct network address, as seen from that proxy. If the sender's address is incorrect, the proxy must add an additional "received" attribute.

A host behind a network address translator or firewall may not be able to insert a network address into the Via header that can be reached by the next hop beyond the NAT. Use of the received attribute allows SIP requests to traverse NATs that only modify the source IP address. NATs that modify port numbers, called *Network Address Port Translators (NAPTs),* will not properly pass SIP when transported on UDP, in which case an application layer gateway is required[25]. When run over TCP, SIP stands a better chance of traversing NATs, since its behavior, in this case, is similar to HTTP (but of course on different ports).

A proxy sending a request to a multicast address must add the "maddr" parameter to its Via header field, and should add the "ttl" parameter. If a server receives a request that contained an "maddr" parameter in the top-most Via field, it should send the response to the multicast address listed in the "maddr" parameter. If a SIP proxy server receives a request which contains its own address in the Via header value, it must respond with a 482 (Loop Detected) status code. A proxy server must not forward a request to a multicast group which already appears in any of the Via headers. This prevents a malfunctioning proxy server from causing loops. Also, it cannot be guaranteed that a proxy server can always detect that the address returned by a location service refers to a host listed in the Via list, as a single host may have aliases or several network interfaces.

Normally, every host that sends or forwards a SIP message adds a Via field indicating the path traversed. However, it is possible that NATs changes the source address and port of the request (e.g., from net-10 to a globally routable address), in which case the Via header field cannot be relied on to route replies. To prevent this, a proxy should check the top-most Via header field to ensure that it contains the sender's correct network address, as seen from that proxy. If the sender's address is incorrect, the proxy must add a "received" parameter to the Via header field inserted by the previous hop. Such a modified Via header field is known as a receiver-tagged Via header field. An example is:

Via: SIP/2.0/UDP erlang.bell-telephone.com:5060
Via: SIP/2.0/UDP 10.0.0.1:5060;received=199.172.136.3

In this example, the message originated from 10.0.0.1 and traversed a NAT with the external address border. ieee.org (199.172.136.3) to reach erlang.bell-telephone.com. The latter noticed the mismatch, and added a

[24] Refer to Chapter for definition of functionality.

[25] An example is a Border Session controller.

parameter to the previous hop's Via header field, containing the address that the packet actually came from. (Note that the NAT border.ieee.org is not a SIP server.)

Via header fields in responses are processed by a proxy or UAC according to the following rules:

1. The first Via header field should indicate the proxy or client processing this response. If it does not, discard the message. Otherwise, remove this Via field.
2. If there is no second Via header field, this response is destined for this client. Otherwise, the processing depends on whether the Via field contains a "maddr" parameter or is a receiver-tagged field:
   a. If the second Via header field contains a "maddr" parameter, send the response to the multicast address listed there, using the port indicated in "sent-by," or port 5060 if none is present. The response should be sent using the TTL indicated in the "ttl" parameter, or with a TTL of 1 if that parameter is not present. For robustness, responses must be sent to the address indicated in the "maddr" parameter even if it is not a multicast address.
   b. If the second Via header field does not contain a "maddr" parameter and is a receiver-tagged field, send the message to the address in the "received" parameter using the port indicated in the "sent-by" value, or using port 5060 if none is present.
   c. If neither of the previous cases apply, send the message to the address indicated by the "sent-by" value in the second Via header field.

This discussion implicitly highlights the private address/NAT issues faced in 2G VoIP systems. Some of these issues can be mitigated in certain IPv6 implementations.

5.2 What is NAT?

We mentioned NAT a number of times. In this section we provide some detailed information on it. Basic Network Address Translation or Basic NAT is a method by which IP addresses are mapped from one group to another, transparent to end users. Network Address Port Translation, or NAPT is a method by which many network addresses and their TCP/UDP (Transmission Control Protocol/User Datagram Protocol) ports are translated into a single network address and its TCP/UDP ports. Together, these two operations, referred to as traditional NAT, provide a mechanism to connect a realm with private addresses to an external realm with globally unique registered addresses. As discussed, NAT has impact on 2G VoIP systems; hence, the reason for our coverage. The NAT operation described in this section is based on IETF RFC 3022 [SRI200101]. Developers should refer directly to the RFC for any normative guidance.

Note: IPv4 NAT is described in RFC 2663 and RFC 3022, but has also been is extended beyond IPv4 networks to include the IPv4-v6 NAT-PT described in RFC 2766. While the IPv4 NAT translates one IPv4 address into another IPv4 address to provide routing between private v4 and external V4 address realms, IPv4-v6 NAT-PT (RFC 2766) translates an IPv4 address into an IPv6 address, and vice versa, to provide routing between a v6 address realm and an external v4 address realm. Unless specified otherwise, NAT is a proxy (middlebox) function referring to both IPv4 NAT, as well as IPv4-v6 NAT-PT [SRI200101], [TSI200001], [SRI200201].

5.2.1 Introduction

The need for IP address translation arises when a network's internal IP addresses cannot be used outside the network either for privacy reasons or because they are invalid for use outside the network. Network topology outside a local domain can change in many ways. Customers may change providers, company backbones may be reorganized, or providers may merge or split. Whenever external topology changes with time, address assignment for nodes within the local domain must also change to reflect the external changes. Changes of this type can be hidden from users within the domain by centralizing changes to a single address translation router.

Basic address translation would (in many cases, except as noted in RFC 2663 and section 6 of RFC 3022) allow hosts in a private network to transparently access the external network and enable access to selective local hosts from the outside. Organizations with a network setup predominantly for internal use, with a need for occasional external access, are good candidates for this scheme.

Many Small Office and Home Office (SOHO) users as well as telecommuting employees have multiple network nodes in their office running TCP/UDP applications, but have a single IP address assigned to their remote access router by their service provider to access remote networks. This community of remote access users typically employs NAPT, which permits multiple nodes in a local network to simultaneously access remote networks using the single IP address assigned to their router.

There are limitations to using the translation method. It is mandatory that all requests and responses pertaining to a session be routed via the same NAT router. One way to ascertain this would be to have NAT based on a border router that is unique to a stub domain, where all IP packets are either originated from the domain or destined to the domain. There are other ways to ensure this with multiple NAT devices. For example, a private domain could have two distinct exit points to different providers and the session flow from the hosts in a private network could traverse through whichever NAT device has the best metric for an external host. When one of the NAT routers fails, the other could route traffic for all the connections. There is however a caveat with this approach, in that rerouted flows could fail at the time of switchover to the new NAT router. A way to overcome this potential problem is to have the routers share the same NAT configuration and exchange state information to ensure a fail-safe backup for each other.

Address translation is application-independent and often accompanied by Application Level Gateways (ALGs) to perform payload monitoring and alterations. FTP is the most popular ALG resident on NAT devices. Applications requiring ALG intervention must not have their payload encoded, as doing that effectively disables the ALG, unless the ALG has the key to decrypt the payload.

This solution has the disadvantage of taking away the end-to-end significance of an IP address, and making up for it with increased state in the network. As a result, end-to-end IP network level security assured by IPSec cannot be assumed to end hosts, with a NAT device enroute. The advantage of this approach, however, is that it can be installed without changes to hosts or routers.

Definition of terms such as “Address Realm,” “Transparent Routing,” “TU Ports,” “ALG,” and others may be found in RFC 2663.

5.2.2 Overview of Traditional NAT

The Address Translation operation presented in this RFC is referred to as “Traditional NAT.” There are other variations of NAT that are explored in this RFC. Traditional NAT would allow hosts within a private network, in most cases, to transparently access hosts in the external network. In a traditional NAT, sessions are unidirectional, outbound from the private network. Sessions in the opposite direction may be allowed on an exceptional basis using static address maps for pre-selected hosts. Basic NAT and NAPT are two variations of traditional NAT, in that translation in Basic NAT is limited to IP addresses alone, whereas translation in NAPT is extended to include IP address and Transport identifier (such as a TCP/UDP port or ICMP query ID).

Unless mentioned otherwise, Address Translation or NAT throughout this section will pertain to traditional NAT—namely Basic NAT—as well as NAPT. Only the stub border routers as described in Figure 5.9 may be configured to perform address translation.

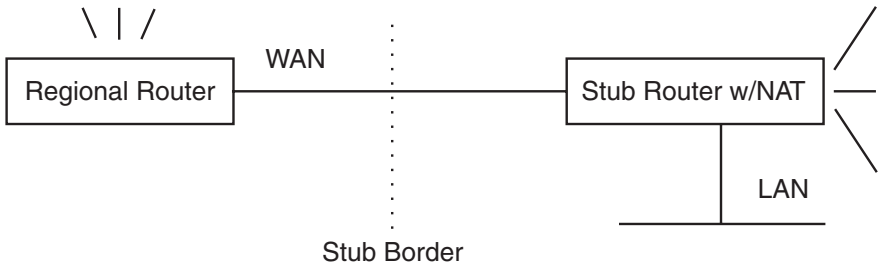


Figure 5.9: Traditional NAT configuration.

5.2.2.1 Overview of Basic NAT

Basic NAT operation is as follows. A stub domain with a set of private network addresses could be enabled to communicate with an external network by dynamically mapping the set of private addresses to a set of globally valid network addresses. If the number of local nodes is less than or equal to addresses in the global set, each local address is guaranteed a global address to map to. Otherwise, nodes allowed to have simultaneous access to external network are limited by the number of addresses in global set. Individual local addresses may be statically mapped to specific global addresses to ensure guaranteed access to the outside or to allow access to the local host from external hosts via a fixed public address. Multiple simultaneous sessions may be initiated from a local node using the same address mapping.

Addresses inside a stub domain are local to that domain and not valid outside the domain. Thus, addresses inside a stub domain can be reused by any other stub domain. For instance, a single Class A address could be used by many stub domains. At each exit point between a stub domain and backbone, NAT is installed. If there is more than one exit point, it is of great importance that each NAT have the same translation table.

For instance, in the example of Figure 5.10, both stubs A and B internally use class A private address block 10.0.0.0/8 (see RFC 1918). Stub A's NAT is assigned the class C address block 198.76.29.0/24, and Stub B's NAT is assigned the class C address block 198.76.28.0/24. The class C addresses are globally unique—no other NAT boxes can use them.

When stub A host 10.33.96.5 wishes to send a packet to stub B host 10.81.13.22, it uses the globally unique address 198.76.28.4 as destination, and sends the packet to its primary router. The stub router has a static route for net 198.76.0.0 so the packet is forwarded to the WAN-link. However, NAT translates the source address 10.33.96.5 of the IP header to the globally unique 198.76.29.7 before the packet is forwarded. Likewise, IP packets on the return path go through similar address translations.

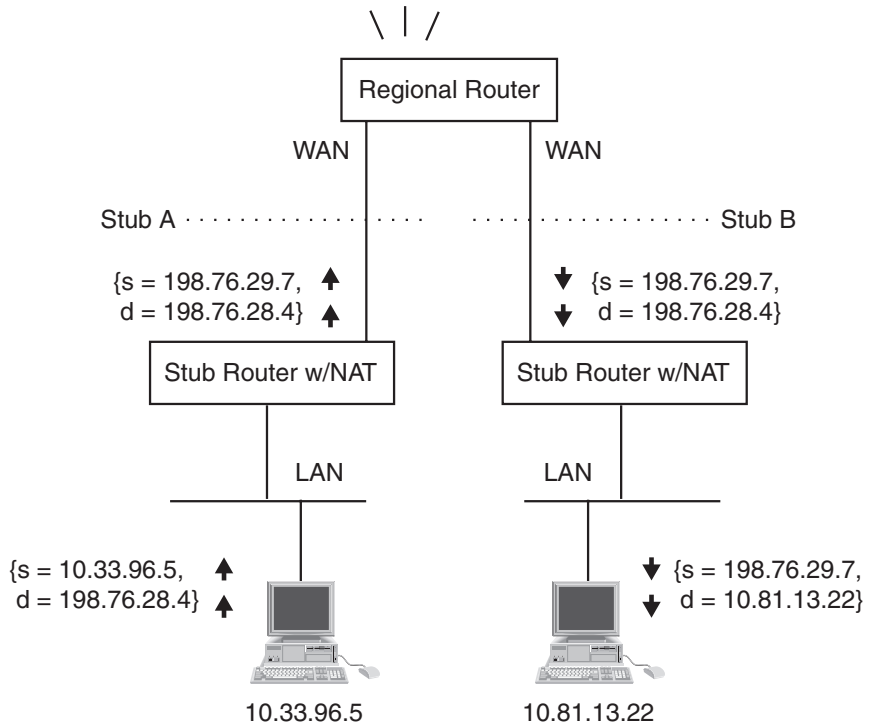


Figure 5.10: Basic NAT operation.

Notice that this requires no changes to hosts or routers. For instance, as far as the stub A host is concerned, 198.76.28.4 is the address used by the host in stub B. The address translations are transparent to end hosts in most cases. Of course, this is just a simple example. There are numerous issues to be explored.

5.2.2.2 Overview of NAPT

Say, an organization has a private IP network and a WAN link to a service provider. The private network's stub router is assigned a globally valid address on the WAN link and the remaining nodes in the organization have IP addresses that have only local significance. In such a case, nodes on the private network could be allowed simultaneous access to the external network, using the single registered IP address with the aid of NAPT. NAPT would allow mapping of tuples of the type (local IP addresses, local TU port number) to tuples of the type (registered IP address, assigned TU port number).

This model fits the requirements of most Small Office/Home Office (SOHO) groups to access external network using a single service provider assigned IP address. This model could be extended to allow inbound access by statically mapping a local node per each service TU port of the registered IP address.

In the example of Figure 5.11, stub A internally uses class A address block 10.0.0.0/8. The stub router's WAN interface is assigned an IP address 138.76.28.4 by the service provider.

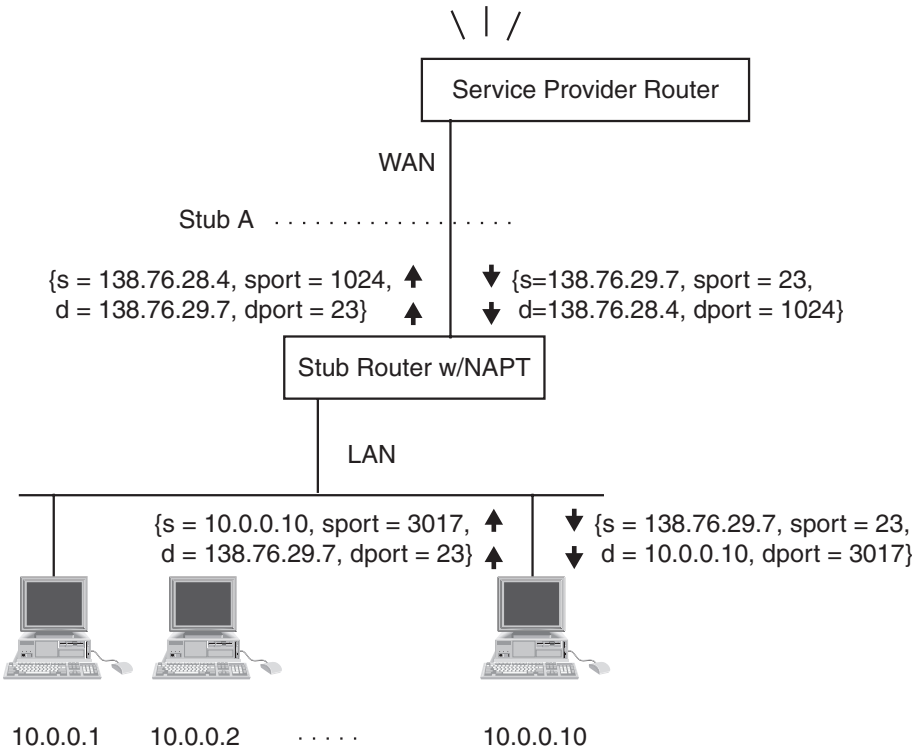


Figure 5.11: Network Address Port Translation (NAPT) operation.

When stub A host 10.0.0.10 sends a telnet packet to host 138.76.29.7, it uses the globally unique address 138.76.29.7 as destination, and sends the packet to it's primary router. The stub router has a static route for the subnet 138.76.0.0/16 so the packet is forwarded to the WAN-link. However, NAPT translates the tuple of source address 10.0.0.10 and source TCP port 3017 in the IP and TCP headers into the globally unique 138.76.28.4 and a uniquely assigned TCP port, say 1024, before the packet is forwarded. Packets on the return path go through similar address and TCP port translations for the target IP address and target TCP port. Notice that this requires no changes to hosts or routers. The translation is completely transparent.

In this setup, only TCP/UDP sessions are allowed and must originate from the local network. However, there are services such as DNS that demand inbound access. There may be other services for which an organization wishes to allow inbound session access. It is possible to statically configure a well known TU port service (RFC 1700) on the stub router to be directed to a specific node in the private network.

In addition to TCP/UDP sessions, ICMP messages, with the exception of REDIRECT message types, may also be monitored by a NAPT router. ICMP query type packets are translated in a manner similar to the way TCP/UDP packets are translated in that the identifier field in an ICMP message header will be uniquely mapped to a query identifier of the registered IP address. The identifier field in ICMP query messages is set by Query sender and returned unchanged in a response message from the Query responder. So, the tuple of (`Local IP address, local ICMP query identifier`) is mapped to a tuple of (`registered IP address, assigned ICMP query Identifier`) by the NAPT router to uniquely identify ICMP queries of all types from any of the local hosts. Modifications to ICMP error messages are discussed in a later section as that involves modifications to the ICMP payload as well as the IP and ICMP headers.

In NAPT setup, where the registered IP address is the same as the IP address of the stub router WAN interface, the router has to be sure to make distinction between TCP, UDP, or ICMP query sessions originated from itself versus those originated from the nodes on a local network. All inbound sessions (including TCP, UDP, and ICMP query sessions) are assumed to be directed to the NAT router as the end node, unless the target service port is statically mapped to a different node in the local network.

Sessions other than TCP, UDP and ICMP query type are simply not permitted from local nodes serviced by a NAPT router.

5.2.3 Translation Phases of a Session

The translation phases with traditional NAT are the same as those described in RFC 2663. The following subsections identify items that are specific to traditional NAT.

5.2.3.1 Address Binding

With Basic NAT, a private address is bound to an external address when the first outgoing session is initiated from the private host. Subsequent to that, all other outgoing sessions originating from the same private address will use the same address binding for packet translation.

In the case of NAPT, where many private addresses are mapped to a single globally unique address, the binding would be from the tuple of (private address, private TU port) to the tuple of (assigned address, assigned TU port). As with Basic NAT, this binding is determined when the first outgoing session is initiated by the tuple of (private address, private TU port) on the private host. While not a common practice, it is possible to have an application on private host establish multiple simultaneous sessions originating from the same tuple of (private address, private TU port). In such a case, a single binding for the tuple of (private address, private TU port) may be used for translation of packets pertaining to all sessions originating from the same tuple on a host.

5.2.3.2 Address Lookup and Translation

After an address binding or (address, TU port) tuple binding in case of NAPT is established, a soft state may be maintained for each of the connections using the binding. Packets belonging to the same session will be subject to session lookup for translation purposes. The exact nature of translation is discussed in the follow-on section.

5.2.3.3 Address Unbinding

When the last session based on an address or (address, TU port) tuple binding is terminated, the binding itself may be terminated.

5.2.4 Packet Translations

Packets pertaining to NAT-managed sessions undergo translation in either direction. Individual packet translation issues are covered in detail in the following subsections.

5.2.4.1 IP, TCP, UDP, and ICMP Header Manipulations

In Basic NAT model, the IP header of every packet must be modified. This modification includes IP address (source IP address for outbound packets and destination IP address for inbound packets) and the IP checksum.

For TCP and UDP sessions, modifications must include update of checksum in the TCP/UDP headers. This is because TCP/UDP checksum also covers a pseudo header which contains the source and destination IP addresses. As an exception, UDP headers with 0 checksum should not be modified. As for ICMP Query packets ([ICMP]), no further changes in ICMP header are required as the checksum in ICMP header does not cover IP addresses.

In a NAPT model, modifications to an IP header are similar to that of Basic NAT. For TCP/UDP sessions, modifications must be extended to include translation of TU port (source TU port for outbound packets and destination TU port for inbound packets) in the TCP/UDP header. The ICMP header in ICMP Query packets must also be modified to replace the query ID and ICMP header checksum. Private host query ID must be translated into assigned ID on the outbound and the exact reverse on the inbound. ICMP header checksum must be corrected to account for Query ID translation.

5.2.4.2 Checksum Adjustment

NAT modifications are applied on a packet-by-packet basis and can be very compute intensive, as they involve one or more checksum modifications in addition to simple field translations. Luckily, we have an algorithm below, which makes checksum adjustment to IP, TCP, UDP and ICMP headers very simple and efficient. Since all these headers use a one's complement sum, it is sufficient to calculate the arithmetic difference between the before-translation and after-translation addresses and add this to the checksum. The algorithm below is applicable only for even offsets (i.e., optr below must be at an even offset from start of header) and even lengths (i.e., olen and nlen below must be even). Sample code (in C) for this is as follows.

```
void checksumadjust(unsigned char *chksum, unsigned char *optr,
int olen, unsigned char *nptr, int nlen)
/* assuming: unsigned char is 8 bits, long is 32 bits.
  - chksum points to the chksum in the packet
  - optr points to the old data in the packet
  - nptr points to the new data in the packet
*/
{
  long x, old, new;
  x=chksum[0]*256+chksum[1];
  x=~x & 0xFFFF;
  while (olen)
  {
      old=optr[0]*256+optr[1]; optr+=2;
      x-=old & 0xffff;
      if (x<=0) { x--; x&=0xffff; }
      olen-=2;
  }
  while (nlen)
  {
      new=nptr[0]*256+nptr[1]; nptr+=2;
      x+=new & 0xffff;
      if (x & 0x10000) { x++; x&=0xffff; }
      nlen-=2;
  }
  x=~x & 0xFFFF;
  chksum[0]=x/256; chksum[1]=x & 0xff;
}
```

5.2.4.3 ICMP Error Packet Modifications

Changes to ICMP error message will include changes to IP and ICMP headers on the outer layer as well as changes to headers of the packet embedded within the ICMP-error message payload.

In order for NAT to be transparent to end-host, the IP address of the IP header embedded within the payload of ICMP-Error message must be modified, the checksum field of the embedded IP header must be modified, and lastly, the ICMP header checksum must also be modified to reflect changes to payload.

In a NAPT setup, if the IP message embedded within ICMP happens to be a TCP, UDP, or ICMP Query packet, you will also need to modify the appropriate TU port number within the TCP/UDP header or the Query Identifier field in the ICMP Query header.

Lastly, the IP header of the ICMP packet must also be modified.

5.2.4.4 FTP Support

One of the most popular applications, "FTP," would require an ALG to monitor the control session payload to determine the ensuing data session parameters. FTP ALG is an integral part of most NAT implementations.

The FTP ALG requires a special table to correct the TCP sequence and acknowledge numbers with source port FTP or destination port FTP. The table entries should have source address, destination address, source port, destination port, delta for sequence numbers and a timestamp. New entries are created only when FTP PORT commands or PASV responses are seen. The sequence number delta may be increased or decreased for every FTP PORT command or PASV response. Sequence numbers are incremented on the outbound and acknowledge numbers are decremented on the inbound by this delta.

FTP payload translations are limited to private addresses and their assigned external addresses (encoded as individual octets in ASCII) for Basic NAT. For NAPT setup, however, the translations must be extended to include the TCP port octets (in ASCII) following the address octets.

5.2.4.5 DNS Support

Considering that sessions in a traditional NAT are predominantly outbound from a private domain, DNS ALG may be obviated from use in conjunction with traditional NAT as follows. DNS server(s) internal to the private domain maintain mapping of names to IP addresses for internal hosts and possibly some external hosts. External DNS servers maintain name mapping for external hosts alone and not for any of the internal hosts. If the private network does not have an internal DNS server, all DNS requests may be directed to the external DNS server to find address mapping for the external hosts.

5.2.4.6 IP Option Handling

An IP datagram with any of the IP options Record Route, Strict Source Route, or Loose Source Route would involve recording or using IP addresses of intermediate routers. A NAT intermediate router may choose not to support these options or leave the addresses untranslated while processing the options. The result of leaving the addresses untranslated would be that private addresses along the source route are exposed end-to-end. This should not jeopardize the traversal path of the packet, per se, as each router is supposed to look at the next hop router only.

5.2.5 Miscellaneous Issues

5.2.5.1 Partitioning of Local and Global Addresses

For NAT to operate as described in this RFC, it is necessary to partition the IP address space into two parts—the private addresses used internal to stub domain and the globally unique addresses. Any given address must either be a private address or a global address. There is no overlap.

The problem with overlap is the following. Say a host in stub A wished to send packets to a host in stub B, but the global addresses of stub B overlapped the private addressees of stub A. In this case, the routers in stub A would not be able to distinguish the global address of stub B from its own private addresses.

5.2.5.2 Private Address Space Recommendation

RFC 1918 has recommendations on address space allocation for private networks. Internet Assigned Numbers Authority (IANA) has three blocks of IP address space, namely 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16 for private internets. In pre-CIDR notation, the first block is nothing but a single class A network number, while the second block is a set of 16 contiguous class B networks, and the third block is a set of 256 contiguous class C networks.

An organization that decides to use IP addresses in the address space defined above can do so without any coordination with IANA or an Internet registry. The address space can thus be used privately by many independent organizations at the same time, with NAT operation enabled on their border routers.

5.2.5.3 Routing Across NAT

The router running NAT should not advertise the private networks to the backbone. Only the networks with global addresses may be known outside the stub. However, global information that NAT receives from the stub border router can be advertised in the stub the usual way.

Typically, the NAT stub router will have a static route configured to forward all external traffic to service provider router over WAN link, and the service provider router will have a static route configured to forward NAT packets (i.e., those whose destination IP address fall within the range of NAT managed global address list) to NAT router over WAN link.

5.2.5.4 Switch-Over from Basic NAT to NAPT

In Basic NAT setup, when private network nodes outnumber global addresses available for mapping (say, a class B private network mapped to a class C global address block), external network access to some of the local nodes is abruptly cut off after the last global address from the address list is used up. This is very inconvenient and constraining. Such an incident can be safely avoided by optionally allowing the Basic NAT router to switch over to NAPT setup for the last global address in the address list. Doing this will ensure that hosts on private network will have continued, uninterrupted access to the external nodes and services for most applications. Note, however, it could be confusing if some of the applications that used to work with Basic NAT suddenly break due to the switch-over to NAPT.

5.2.6 NAT Limitations

RFC 2663 covers the limitations of all flavors of NAT, broadly speaking. The following subsections identify limitations specific to traditional NAT.

5.2.6.1 Privacy and Security

Traditional NAT can be viewed as providing a privacy mechanism since sessions are unidirectional from private hosts, and the actual addresses of the private hosts are not visible to external hosts. The same characteristic that enhances privacy potentially makes debugging problems (including security violations) more difficult. If a host in a private network is abusing the Internet in some way (such as trying to attack another machine or even sending large amounts of spam) it is more difficult to track the actual source of trouble because the IP address of the host is hidden in a NAT router.

5.2.6.2 ARP responses to NAT Mapped Global Addresses on a LAN Interface

NAT must be enabled only on border routers of a stub domain. The examples provided in the document to illustrate Basic NAT and NAPT have maintained a WAN link for connection to external router (i.e., service provider router) from NAT router. However, if the WAN link were to be replaced by a LAN connection and if part or all of the global address space used for NAT mapping belongs to the same IP subnet as the LAN segment, the NAT router would be expected to provide ARP support for the address range that belongs to the same subnet. Responding to ARP requests for the NAT mapped global addresses with its own MAC address is a must in such a situation with Basic NAT setup. If the NAT router did not respond to these requests, there is no other node in the network that has ownership of these addresses and hence will go unresponded.

This scenario is unlikely with NAPT setup except when the single address used in NAPT mapping is not the interface address of the NAT router (as in the case of a switch-over from Basic NAT to NAPT explained in 5.2.5.4 above, for example).

Using an address range from a directly connected subnet for NAT address mapping would obviate static route configuration on the service provider router.

It is the opinion of the authors that a LAN link to a service provider router is not very common. However, vendors may be interested to optionally support proxy ARP just in case.

5.2.6.3 Translation of Outbound TCP/UDP Fragmented Packets in NAPT Setup

Translation of outbound TCP/UDP fragments (i.e., those originating from private hosts) in NAPT setup are doomed to fail. This is because only the first fragment contains the TCP/UDP header that would be necessary to associate the packet to a session for translation purposes. Subsequent fragments do not contain TCP/UDP port information, but simply carry the same fragmentation identifier specified in the first fragment. Say, two private hosts originated fragmented TCP/UDP packets to the same destination host. And, they happened to use the same fragmentation identifier. When the target host receives the two unrelated datagrams, carrying the same fragmentation ID, and from the same assigned host address, it is unable to determine which of the two sessions the datagrams belong to. Consequently, both sessions will be corrupted.

5.3 STUN—Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)

STUN is a lightweight protocol described in RFC 3489 that allows applications to discover the presence and types of NATs and firewalls between them and the public Internet. It also provides the ability for applications to determine the public IP addresses allocated to them by the NAT. STUN works with many existing NATs and does not require any special behavior from them. As a result, it allows a variety of applications to work through existing NAT infrastructure [ROS200301] (however, up to now it has not experienced major acceptance/deployment). The STUN operation described in this section is based on IETF RFC 3489 [ROS200301]. Developers should refer to the original RFP for any normative guidance.

5.3.1 Applicability Statement

It is recognized that STUN is not a cure-all for the problems associated with NAT. It does not enable incoming TCP connections through NAT. It allows incoming UDP packets through NAT, but only through a subset of existing NAT types. In particular, STUN does not enable incoming UDP packets through symmetric NATs, which are common in large enterprises. STUN's discovery procedures are based on assumptions on NAT treatment of UDP; such assumptions may prove invalid down the road as new NAT devices are deployed. STUN does not work when it is used to obtain an address to communicate with a

peer that happens to be behind the same NAT. STUN does not work when the STUN server is not in a common shared address realm.

5.3.2 Introduction

NATs, while providing many benefits, also come with many drawbacks. The most troublesome of those drawbacks is the fact that they break many existing IP applications, and make it difficult to deploy new ones. Guidelines have been developed that describe how to build "NAT friendly" protocols, but many protocols simply cannot be constructed according to those guidelines. Examples of such protocols include almost all peer-to-peer protocols, such as multimedia communications, file sharing, and games.

To combat this problem, Application Layer Gateways (ALGs) have been embedded in NATs. ALGs perform the application layer functions required for a particular protocol to traverse a NAT. Typically, this involves rewriting application layer messages to contain translated addresses, rather than the ones inserted by the sender of the message. ALGs have serious limitations, including scalability, reliability, and speed of deploying new applications. To resolve these problems, the Middlebox Communications (MIDCOM) protocol has been developed (see RFC 3303). MIDCOM allows an application entity, such as an end client or network server of some sort (like a SIP proxy discussed in Chapter 3 in the context of RFC 3261) to control a NAT (or firewall) in order to obtain NAT bindings and open or close pinholes. In this way, NATs and applications can be separated once more, eliminating the need for embedding ALGs in NATs and resolving the limitations imposed by current architectures. MIDCOM is covered in Section 5.4 of this chapter.

Unfortunately, MIDCOM requires upgrades to existing NATs and firewalls in addition to application components. Complete upgrades of these NAT and firewall products will take a long time, potentially years. This is due, in part, to the fact that the deployers of NATs and firewalls are not the same people who are deploying and using applications. As a result, the incentive to upgrade these devices will be low in many cases. Consider, for example, an airport Internet lounge that provides access with a NAT. A user connecting to the NATed network may wish to use a peer-to-peer service, but cannot, because the NAT does not support it. Since the administrators of the lounge are not the ones providing the service, they are not motivated to upgrade their NAT equipment to support it, using either an ALG or MIDCOM.

Another problem is that the MIDCOM protocol requires that the agent controlling the middleboxes know the identity of those middleboxes, and have a relationship with them which permits control. In many configurations, this will not be possible. For example, many cable access providers use NAT in front of their entire access network. This NAT could be in addition to a residential NAT purchased and operated by the end user. The end user will probably not have a control relationship with the NAT in the cable access network, and may not even know of its existence.

Many existing proprietary protocols, such as those for online games and VoIP, have developed "tricks" that allow them to operate through NATs without changing those NATs. RFC 3489 is an attempt to take some of those ideas, and codify them into an interoperable protocol that can meet the needs of many applications. STUN allows entities behind a NAT to first discover the presence of a NAT and the type of NAT, and then to learn the address bindings allocated by the NAT. STUN requires no changes to NATs and works with an arbitrary number of NATs in tandem between the application entity and the public Internet.

5.3.3 Applicability to VoIP

The primary usage STUN has found is in the area of VoIP, facilitating allocation of addresses for receiving RTP traffic. In that application, the periodic keepalives are provided by the RTP traffic itself. However, several practical problems arise for RTP. First, RTP assumes that RTCP traffic is on a port one higher than the RTP

traffic. This pairing property cannot be guaranteed through NATs that are not directly controllable. As a result, RTCP traffic may not be properly received. Protocol extensions to SDP have been proposed which mitigate this by allowing the client to signal a different port for RTCP. However, there will be interoperability problems for some time. For VoIP, silence suppression can cause a gap in the transmission of RTP packets. This could result in the loss of a binding in the middle of a call, if that silence period exceeds the binding timeout. This can be mitigated by sending occasional silence packets to keep the binding alive. However, the result is additional brittleness; proper operation depends on the silence suppression algorithm in use, the usage of a comfort noise codec, the duration of the silence period, and the binding lifetime in the NAT.

5.3.4 Definitions

STUN Client: A STUN client (also just referred to as a client) is an entity that generates STUN requests. A STUN client can execute on an end system, such as a user's PC, or can run in a network element, such as a conferencing server.

STUN Server: A STUN Server (also just referred to as a server) is an entity that receives STUN requests, and sends STUN responses. STUN servers are generally attached to the public Internet.

5.3.5 NAT Variations

It has been observed that NAT treatment of UDP varies among implementations. The four treatments observed in implementations are:

Full Cone: A full cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Furthermore, any external host can send a packet to the internal host, by sending a packet to the mapped external address.

Restricted Cone: A restricted cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Unlike a full cone NAT, an external host (with IP address X) can send a packet to the internal host only if the internal host had previously sent a packet to IP address X.

Port Restricted Cone: A port restricted cone NAT is like a restricted cone NAT, but the restriction includes port numbers. Specifically, an external host can send a packet, with source IP address X and source port P, to the internal host only if the internal host had previously sent a packet to IP address X and port P.

Symmetric: A symmetric NAT is one where all requests from the same internal IP address and port, to a specific destination IP address and port, are mapped to the same external IP address and port. If the same host sends a packet with the same source address and port, but to a different destination, a different mapping is used. Furthermore, only the external host that receives a packet can send a UDP packet back to the internal host.

Determining the type of NAT is important in many cases. Depending on what the application wants to do, it may need to take the particular behavior into account.

5.3.6 Overview of Operation

This section is descriptive only (normative behavior is described in Sections 5.3.8 and 5.3.9.) The typical STUN configuration is shown in Figure 5.12. A STUN client is connected to private network 1. This network connects to private network 2 through NAT 1. Private network 2 connects to the public Internet through NAT 2. The STUN server resides on the public Internet.

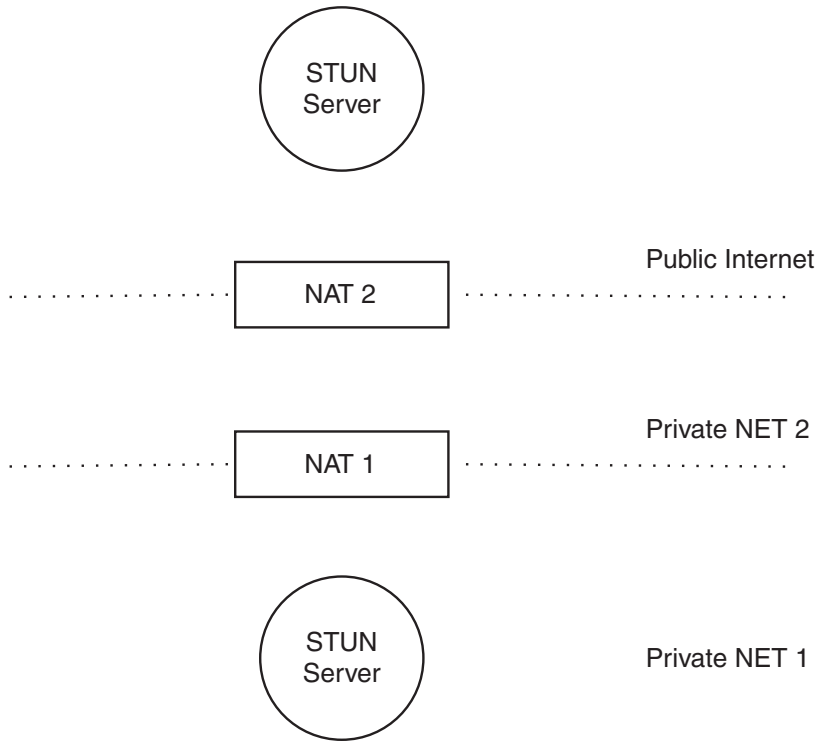


Figure 5.12: STUN configuration.

STUN is a simple client-server protocol. A client sends a request to a server, and the server returns a response. There are two types of requests—Binding Requests, sent over UDP, and Shared Secret Requests, sent over TLS over TCP. Shared Secret Requests ask the server to return a temporary username and password. This username and password are used in a subsequent Binding Request and Binding Response, for the purposes of authentication and message integrity.

Binding requests are used to determine the bindings allocated by NATs. The client sends a Binding Request to the server, over UDP. The server examines the source IP address and port of the request, and copies them into a response that is sent back to the client. There are some parameters in the request that allow the client to ask that the response be sent elsewhere, or that the server send the response from a different address and port. There are attributes for providing message integrity and authentication.

The trick is using STUN to discover the presence of NAT, and to learn and use the bindings they allocate.

The STUN client is typically embedded in an application which needs to obtain a public IP address and port that can be used to receive data. For example, it might need to obtain an IP address and port to receive Real Time Transport Protocol (RTP) traffic. When the application starts, the STUN client within the application sends a STUN Shared Secret Request to its server, obtains a username and password, and then sends it a Binding Request. STUN servers can be discovered through DNS SRV records, and it is generally assumed that the client is configured with the domain it needs to use to find the STUN server. Generally, this will be the domain of the provider of the service the application is using (such a provider is incented to deploy

STUN servers in order to allow its customers to use its application through NAT). Of course, a client can determine the address or domain name of a STUN server through other means. A STUN server can even be embedded within an end system.

The STUN Binding Request is used to discover the presence of a NAT, and to discover the public IP address and port mappings generated by the NAT. Binding Requests are sent to the STUN server using UDP. When a Binding Request arrives at the STUN server, it may have passed through one or more NATs between the STUN client and the STUN server. As a result, the source address of the request received by the server will be the mapped address created by the NAT closest to the server. The STUN server copies that source IP address and port into a STUN Binding Response, and sends it back to the source IP address and port of the STUN request. For all of the NAT types above, this response will arrive at the STUN client.

When the STUN client receives the STUN Binding Response, it compares the IP address and port in the packet with the local IP address and port it bound to when the request was sent. If these do not match, the STUN client is behind one or more NATs. In the case of a full-cone NAT, the IP address and port in the body of the STUN response are public, and can be used by any host on the public Internet to send packets to the application that sent the STUN request. An application need only listen in on the IP address and port from which the STUN request was sent. Any packets sent by a host on the public Internet to the public address and port learned by STUN will be received by the application.

Of course, the host may not be behind a full-cone NAT. Indeed, it does not yet know what type of NAT it is behind. To determine that, the client uses additional STUN Binding Requests. The exact procedure is flexible, but would generally work as follows. The client would send a second STUN Binding Request, this time to a different IP address, but from the same source IP address and port. If the IP address and port in the response are different from those in the first response, the client knows it is behind a symmetric NAT. To determine if it is behind a full-cone NAT, the client can send a STUN Binding Request with flags that tell the STUN server to send a response from a different IP address and port than the request was received on. In other words, if the client sent a Binding Request to IP address/port A/B using a source IP address/port of X/Y, the STUN server would send the Binding Response to X/Y using source IP address/port C/D. If the client receives this response, it knows it is behind a full cone NAT.

STUN also allows the client to ask the server to send the Binding Response from the same IP address the request was received on, but with a different port. This can be used to detect whether the client is behind a port restricted cone NAT or just a restricted cone NAT.

It should be noted that the configuration in Figure 5.12 is not the only permissible configuration. The STUN server can be located anywhere, including within another client. The only requirement is that the STUN server is reachable by the client, and if the client is trying to obtain a publicly routable address, that the server reside on the public Internet.

5.3.7 Message Overview

STUN messages are TLV (type-length-value) encoded using big endian (network ordered) binary. All STUN messages start with a STUN header, followed by a STUN payload. The payload is a series of STUN attributes, the set of which depends on the message type. The STUN header contains a STUN message type, transaction ID, and length. The message type can be Binding Request, Binding Response, Binding Error Response, Shared Secret Request, Shared Secret Response, or Shared Secret Error Response. The transaction ID is used to correlate requests and responses. The length indicates the total length of the STUN payload, not including the header. This allows STUN to run over TCP. Shared Secret Requests are always sent over TCP (indeed, using TLS over TCP).

Several STUN attributes are defined. The first is a MAPPED-ADDRESS attribute, which is an IP address and port. It is always placed in the Binding Response, and it indicates the source IP address and port the server saw in the Binding Request. There is also a RESPONSE-ADDRESS attribute, which contains an IP address and port. The RESPONSE-ADDRESS attribute can be present in the Binding Request, and indicates where the Binding Response is to be sent. It's optional, and when not present, the Binding Response is sent to the source IP address and port of the Binding Request.

The third attribute is the CHANGE-REQUEST attribute, and it contains two flags to control the IP address and port used to send the response. These flags are called *change IP* and *change port flags*. The CHANGE-REQUEST attribute is allowed only in the Binding Request. The "change IP" and "change port" flags are useful for determining whether the client is behind a restricted cone NAT or restricted port cone NAT. They instruct the server to send the Binding Responses from a different source IP address and port. The CHANGE-REQUEST attribute is optional in the Binding Request.

The fourth attribute is the CHANGED-ADDRESS attribute. It is present in Binding Responses. It informs the client of the source IP address and port that would be used if the client requested the "change IP" and "change port" behavior.

The fifth attribute is the SOURCE-ADDRESS attribute. It is only present in Binding Responses. It indicates the source IP address and port where the response was sent from. It is useful for detecting twice NAT configurations.

The sixth attribute is the USERNAME attribute. It is present in a Shared Secret Response, which provides the client with a temporary username and password (encoded in the PASSWORD attribute). The USERNAME is also present in Binding Requests, serving as an index to the shared secret used for the integrity protection of the Binding Request. The seventh attribute, PASSWORD, is only found in Shared Secret Response messages. The eighth attribute is the MESSAGE-INTEGRITY attribute, which contains a message integrity check over the Binding Request or Binding Response.

The ninth attribute is the ERROR-CODE attribute. This is present in the Binding Error Response and Shared Secret Error Response. It indicates the error that has occurred. The tenth attribute is the UNKNOWN-ATTRIBUTES attribute which is present in either the Binding Error Response or Shared Secret Error Response. It indicates the mandatory attributes from the request which were unknown. The eleventh attribute is the REFLECTED-FROM attribute which is present in Binding Responses. It indicates the IP address and port of the sender of a Binding Request used for traceability purposes to prevent certain denial-of-service attacks.

5.3.8 Server Behavior

The server behavior depends on whether the request is a Binding Request or a Shared Secret Request.

5.3.8.1 Binding Requests

A STUN server must be prepared to receive Binding Requests on four address/port combinations—(A1, P1), (A2, P1), (A1, P2), and (A2, P2). (A1, P1) represent the primary address and port, and these are the ones obtained through the client discovery procedures below. Typically, P1 will be port 3478, the default STUN port. A2 and P2 are arbitrary. A2 and P2 are advertised by the server through the CHANGED-ADDRESS attribute, as described below.

It is recommended that the server check the Binding Request for a MESSAGE-INTEGRITY attribute. If not present, and the server requires integrity checks on the request, it generates a Binding Error Response with an ERROR-CODE attribute with response code 401. If the MESSAGE-INTEGRITY attribute was present,

the server computes the HMAC over the request as described in Section 5.3.11.2. The key to use depends on the shared secret mechanism. If the STUN Shared Secret Request was used, the key must be the one associated with the USERNAME attribute present in the request. If the USERNAME attribute was not present, the server must generate a Binding Error Response. The Binding Error Response must include an ERROR-CODE attribute with response code 432. If the USERNAME is present, but the server does not remember the shared secret for that USERNAME (because it timed out, for example), the server must generate a Binding Error Response. The Binding Error Response must include an ERROR-CODE attribute with response code 430. If the server does know the shared secret, but the computed HMAC differs from the one in the request, the server must generate a Binding Error Response with an ERROR-CODE attribute with response code 431. The Binding Error Response is sent to the IP address and port the Binding Request came from, and sent from the IP address and port the Binding Request was sent to.

Assuming the message integrity check passed, processing continues. The server must check for any attributes in the request with values less than or equal to 0x7fff which it does not understand. If it encounters any, the server must generate a Binding Error Response, and it MUST include an ERROR-CODE attribute with a 420 response code.

That response must contain an UNKNOWN-ATTRIBUTES attribute listing the attributes with values less than or equal to 0x7fff which were not understood. The Binding Error Response is sent to the IP address and port the Binding Request came from, and sent from the IP address and port the Binding Request was sent to.

Assuming the request was correctly formed, the server must generate a single Binding Response. The Binding Response must contain the same transaction ID contained in the Binding Request. The length in the message header must contain the total length of the message in bytes, excluding the header. The Binding Response must have a message type of "Binding Response."

The server must add a MAPPED-ADDRESS attribute to the Binding Response. The IP address component of this attribute must be set to the source IP address observed in the Binding Request. The port component of this attribute must be set to the source port observed in the Binding Request.

If the RESPONSE-ADDRESS attribute was absent from the Binding Request, the destination address and port of the Binding Response must be the same as the source address and port of the Binding Request. Otherwise, the destination address and port of the Binding Response must be the value of the IP address and port in the RESPONSE-ADDRESS attribute.

The source address and port of the Binding Response depend on the value of the CHANGE-REQUEST attribute and on the address and port the Binding Request was received on, and are summarized in Table 5.2.

Let Da represent the destination IP address of the Binding Request (which will be either A1 or A2), and Dp represent the destination port of the Binding Request (which will be either P1 or P2). Let Ca represent the other address, so that if Da is A1, Ca is A2. If Da is A2, Ca is A1. Similarly, let Cp represent the other port, so that if Dp is P1, Cp is P2. If Dp is P2, Cp is P1. If the "change port" flag was set in the CHANGE-REQUEST attribute of the Binding Request, and the "change IP" flag was not set, the source IP address of the Binding Response must be Da and the source port of the Binding Response must be Cp. If the "change IP" flag was set in the Binding Request, and the "change port" flag was not set, the source IP address of the Binding Response must be Ca and the source port of the Binding Response MUST be Dp. When both flags are set, the source IP address of the Binding Response MUST be Ca and the source port of the Binding Response MUST be Cp. If neither flag is set, or if the CHANGE-REQUEST attribute is absent entirely, the source IP address of the Binding Response MUST be Da and the source port of the Binding Response must be Dp.

Table 5.2: Impact of flags on packet source and CHANGED-ADDRESS.

Flags	Source Address	Source Port	CHANGED-ADDRESS
none	Da	Dp	Ca:Cp
Change IP	Ca	Dp	Ca:Cp
Change port	Da	Cp	Ca:Cp
Change IP and Change port	Ca	Cp	Ca:Cp

The server must add a SOURCE-ADDRESS attribute to the Binding Response, containing the source address and port used to send the Binding Response.

The server must add a CHANGED-ADDRESS attribute to the Binding Response. This contains the source IP address and port that would be used if the client had set the "change IP" and "change port" flags in the Binding Request. As summarized in Table 5.2, these are Ca and Cp, respectively, regardless of the value of the CHANGE-REQUEST flags.

If the Binding Request contained both the USERNAME and MESSAGE-INTEGRITY attributes, the server must add a MESSAGE-INTEGRITY attribute to the Binding Response. The attribute contains an HMAC over the response, as described in Section 5.3.11.2. The key to use depends on the shared secret mechanism. If the STUN Shared Secret Request was used, the key must be the one associated with the USERNAME attribute present in the Binding Request.

If the Binding Request contained a RESPONSE-ADDRESS attribute, the server MUST add a REFLECTED-FROM attribute to the response. If the Binding Request was authenticated using a username obtained from a Shared Secret Request, the REFLECTED-FROM attribute MUST contain the source IP address and port where that Shared Secret Request came from. If the username present in the request was not allocated using a Shared Secret Request, the REFLECTED-FROM attribute must contain the source address and port of the entity which obtained the username, as best can be verified with the mechanism used to allocate the username. If the username was not present in the request, and the server was willing to process the request, the REFLECTED-FROM attribute should contain the source IP address and port where the request came from.

The server should not retransmit the response. Reliability is achieved by having the client periodically resend the request, each of which triggers a response from the server.

5.3.8.2 Shared Secret Requests

Shared Secret Requests are always received on TLS connections. When the server receives a request from the client to establish a TLS connection, it must proceed with TLS, and should present a site certificate. The TLS ciphersuite TLS_RSA_WITH_AES_128_CBC_SHA should be used. Client TLS authentication must not be done, since the server is not allocating any resources to clients, and the computational burden can be a source of attacks.

If the server receives a Shared Secret Request, it must verify that the request arrived on a TLS connection. If it did not receive the request over TLS, it must generate a Shared Secret Error Response, and it must include an ERROR-CODE attribute with a 433 response code. The destination for the error response depends on the transport on which the request was received. If the Shared Secret Request was received over TCP, the Shared Secret Error Response is sent over the same connection the request was received on. If the Shared Secret Request was receive over UDP, the Shared Secret Error Response is sent to the source IP address and port that the request came from.

The server must check for any attributes in the request with values less than or equal to 0x7fff which it does not understand. If it encounters any, the server must generate a Shared Secret Error Response, and it

must include an ERROR-CODE attribute with a 420 response code. That response must contain an UNKNOWN-ATTRIBUTES attribute listing the attributes with values less than or equal to 0x7fff which were not understood. The Shared Secret Error Response is sent over the TLS connection.

All Shared Secret Error Responses must contain the same transaction ID contained in the Shared Secret Request. The length in the message header must contain the total length of the message in bytes, excluding the header. The Shared Secret Error Response must have a message type of "Shared Secret Error Response" (0x0112).

Assuming the request was properly constructed, the server creates a Shared Secret Response. The Shared Secret Response must contain the same transaction ID contained in the Shared Secret Request. The length in the message header must contain the total length of the message in bytes, excluding the header. The Shared Secret Response must have a message type of "Shared Secret Response." The Shared Secret Response must contain a USERNAME attribute and a PASSWORD attribute. The USERNAME attribute serves as an index to the password, which is contained in the PASSWORD attribute. The server can use any mechanism it chooses to generate the username. However, the username must be valid for a period of at least 10 minutes. Validity means that the server can compute the password for that username. There MUST be a single password for each username. In other words, the server cannot, 10 minutes later, assign a different password to the same username. The server must hand out a different username for each distinct Shared Secret Request. Distinct, in this case, implies a different transaction ID. It is recommended that the server explicitly invalidate the username after ten minutes. It must invalidate the username after 30 minutes. The PASSWORD contains the password bound to that username. The password must have at least 128 bits. The likelihood that the server assigns the same password for two different usernames must be vanishingly small, and the passwords must be unguessable. In other words, they must be a cryptographically random function of the username.

These requirements can still be met using a stateless server, by intelligently computing the USERNAME and PASSWORD. One approach is to construct the USERNAME as:

```
USERNAME = <prefix,rounded-time,clientIP,hmac>
```

Where prefix is some random text string (different for each shared secret request), rounded-time is the current time modulo 20 minutes, clientIP is the source IP address where the Shared Secret Request came from, and hmac is an HMAC over the prefix, rounded-time, and client IP, using a server private key. The password is then computed as:

```
password = <hmac(USERNAME,anotherprivatekey)>
```

With this structure, the username itself, which will be present in the Binding Request, contains the source IP address where the Shared Secret Request came from. That allows the server to meet the requirements specified in Section 5.3.8.1 for constructing the REFLECTED-FROM attribute. The server can verify that the username was not tampered with, using the hmac present in the username.

The Shared Secret Response is sent over the same TLS connection the request was received on. The server should keep the connection open, and let the client close it.

5.3.9 Client Behavior

The behavior of the client is very straightforward. Its task is to discover the STUN server, obtain a shared secret, formulate the Binding Request, handle request reliability, and process the Binding Responses.

5.3.9.1 Discovery

Generally, the client will be configured with a domain name of the provider of the STUN servers. This domain name is resolved to an IP address and port using the SRV procedures specified in RFC 2782.

Specifically, the service name is "stun." The protocol is "udp" for sending Binding Requests or "tcp" for sending Shared Secret Requests. The procedures of RFC 2782 are followed to determine the server to contact. RFC 2782 spells out the details of how a set of SRV records are sorted and then tried. However, it only states that the client should "try to connect to the (protocol, address, service)" without giving any details on what happens in the event of failure. Those details are described here for STUN.

For STUN requests, failure occurs if there is a transport failure of some sort (generally, due to fatal ICMP errors in UDP or connection failures in TCP). Failure also occurs if the transaction fails due to timeout. This occurs 9.5 seconds after the first request is sent, for both Shared Secret Requests and Binding Requests. See Section 5.3.9.3 for details on transaction timeouts for Binding Requests. If a failure occurs, the client should create a new request, which is identical to the previous, but has a different transaction ID and MESSAGE INTEGRITY attribute (the HMAC will change because the transaction ID has changed). That request is sent to the next element in the list as specified by RFC 2782.

The default port for STUN requests is 3478, for both TCP and UDP. Administrators should use this port in their SRV records, but may use others.

If no SRV records were found, the client performs an A record lookup of the domain name. The result will be a list of IP addresses, each of which can be contacted at the default port.

This would allow a firewall admin to open the STUN port, so hosts within the enterprise could access new applications. Whether they will or will not do this is a relevant question.

5.3.9.2 Obtaining a Shared Secret

There are several attacks possible on STUN systems. Many of these are prevented through integrity of requests and responses. To provide that integrity, STUN makes use of a shared secret between client and server, used as the keying material for an HMAC in both the Binding Request and Binding Response. STUN allows for the shared secret to be obtained in any way (for example, Kerberos). However, it must have at least 128 bits of randomness. In order to ensure interoperability, this specification describes a TLS-based mechanism. This mechanism, described in this section, must be implemented by clients and servers.

First, the client determines the IP address and port that it will open a TCP connection to. This is done using the discovery procedures in Section 5.3.9.1. The client opens up the connection to that address and port, and immediately begins TLS negotiation. The client must verify the identity of the server. To do that, it follows the identification procedures defined in Section 3.1 of RFC 2818. Those procedures assume the client is dereferencing a URI. For purposes of usage with this specification, the client treats the domain name or IP address used in Section 5.3.9.1 as the host portion of the URI that has been dereferenced.

Once the connection is opened, the client sends a Shared Secret request. This request has no attributes, just the header. The transaction ID in the header must meet the requirements outlined for the transaction ID in a binding request, described in Section 5.3.9.3 below. The server generates a response, which can either be a Shared Secret Response or a Shared Secret Error Response.

If the response was a Shared Secret Error Response, the client checks the response code in the ERROR-CODE attribute. Interpretation of those response codes is identical to the processing of Section 5.3.9.4 for the Binding Error Response.

If a client receives a Shared Secret Response with an attribute whose type is greater than 0x7fff, the attribute must be ignored. If the client receives a Shared Secret Response with an attribute whose type is less than or equal to 0x7fff, the response is ignored.

If the response was a Shared Secret Response, it will contain a short-lived username and password encoded in the USERNAME and PASSWORD attributes, respectively.

The client may generate multiple Shared Secret Requests on the connection, and it may do so before receiving Shared Secret Responses to previous Shared Secret Requests. The client should close the connection as soon as it has finished obtaining usernames and passwords.

Section 5.3.9.3 describes how these passwords are used to provide integrity protection over Binding Requests, and Section 5.3.8.1 describes how it is used in Binding Responses.

5.3.9.3 Formulating the Binding Request

A Binding Request formulated by the client follows the syntax rules defined in Section 5.3.11. Any two requests that are not bit-wise identical, and not sent to the same server from the same IP address and port, must carry different transaction IDs. The transaction ID must be uniformly and randomly distributed between 0 and 2**128 – 1. The large range is needed because the transaction ID serves as a form of randomization, helping to prevent replays of previously signed responses from the server. The message type of the request must be "Binding Request."

The RESPONSE-ADDRESS attribute is optional in the Binding Request. It is used if the client wishes the response to be sent to a different IP address and port than the one the request was sent from. This is useful for determining whether the client is behind a firewall, and for applications that have separated control and data components. See Section 5.3.10.3 for more details. The CHANGE-REQUEST attribute is also optional. Whether it is present depends on what the application is trying to accomplish. See Section 5.3.10 for some example uses.

The client should add MESSAGE-INTEGRITY and USERNAME attributes to the Binding Request. This MESSAGE-INTEGRITY attribute contains an HMAC. The value of the username, and the key to use in the MESSAGE-INTEGRITY attribute depend on the shared secret mechanism. If the STUN Shared Secret Request was used, the USERNAME must be a valid username obtained from a Shared Secret Response within the last nine minutes. The shared secret for the HMAC is the value of the PASSWORD attribute obtained from the same Shared Secret Response.

Once formulated, the client sends the Binding Request. Reliability is accomplished through client retransmissions. Clients should retransmit the request starting with an interval of 100ms, doubling every retransmit until the interval reaches 1.6s. Retransmissions continue with intervals of 1.6s until a response is received, or a total of nine requests have been sent. If no response is received by 1.6 seconds after the last request has been sent, the client should consider the transaction to have failed. In other words, requests would be sent at times 0ms, 100ms, 300ms, 700ms, 1500ms, 3100ms, 4700ms, 6300ms, and 7900ms. At 9500ms, the client considers the transaction to have failed if no response has been received.

5.3.9.4 Processing Binding Responses

The response can either be a Binding Response or Binding Error Response. Binding Error Responses are always received on the source address and port the request was sent from. A Binding Response will be received on the address and port placed in the RESPONSE-ADDRESS attribute of the request. If none was present, the Binding Responses will be received on the source address and port the request was sent from.

If the response is a Binding Error Response, the client checks the response code from the ERROR-CODE attribute of the response. For a 400 response code, the client should display the reason phrase to the user. For a 420 response code, the client should retry the request, this time omitting any attributes listed in the UNKNOWN-ATTRIBUTES attribute of the response. For a 430 response code, the client should obtain a

new shared secret, and retry the Binding Request with a new transaction. For 401 and 432 response codes, if the client had omitted the USERNAME or MESSAGE-INTEGRITY attribute as indicated by the error, it should try again with those attributes. For a 431 response code, the client should alert the user, and may try the request again after obtaining a new username and password. For a 500 response code, the client may wait several seconds and then retry the request. For a 600 response code, the client must not retry the request, and should display the reason phrase to the user. Unknown attributes between 400 and 499 are treated like a 400, unknown attributes between 500 and 599 are treated like a 500, and unknown attributes between 600 and 699 are treated like a 600. Any response between 100 and 399 must result in the cessation of request retransmissions, but otherwise is discarded.

If a client receives a response with an attribute whose type is greater than 0x7fff, the attribute MUST be ignored. If the client receives a response with an attribute whose type is less than or equal to 0x7fff, request retransmissions must cease, but the entire response is otherwise ignored. If the response is a Binding Response, the client should check the response for a MESSAGE-INTEGRITY attribute. If not present, and the client placed a MESSAGE-INTEGRITY attribute into the request, it must discard the response. If present, the client computes the HMAC over the response as described in Section 5.3.11.2. The key to use depends on the shared secret mechanism. If the STUN Shared Secret Request was used, the key must be the same as that used to compute the MESSAGE-INTEGRITY attribute in the request. If the computed HMAC differs from the one in the response, the client must discard the response, and should alert the user about a possible attack. If the computed HMAC matches the one from the response, processing continues.

Reception of a response (either a Binding Error Response or Binding Response) to a Binding Request will terminate retransmissions of that request. However, clients must continue to listen for responses to a Binding Request for 10 seconds after the first response. If it receives any responses in this interval with different message types (Binding Responses and Binding Error Responses, for example) or different MAPPED-ADDRESSes, it is an indication of a possible attack. The client must not use the MAPPED-ADDRESS from any of the responses it received (either the first or the additional ones), and should alert the user.

Furthermore, if a client receives more than twice as many Binding Responses as the number of Binding Requests it sent, it must not use the MAPPED-ADDRESS from any of those responses, and should alert the user about a potential attack.

If the Binding Response is authenticated, and the MAPPED-ADDRESS was not discarded because of a potential attack, the CLIENT may use the MAPPED-ADDRESS and SOURCE-ADDRESS attributes.

5.3.10 Use Cases

The rules of Sections 8 and 9 describe exactly how a client and server interact to send requests and get responses. However, they do not dictate how the STUN protocol is used to accomplish useful tasks. That is at the discretion of the client. Here, we provide some useful scenarios for applying STUN.

5.3.10.1 Discovery Process

In this scenario, a user is running a multimedia application and needs to determine which of the following scenarios applies to it:

1. On the open Internet;
2. Firewall that blocks UDP;
3. Firewall that allows UDP out, and responses have to come back to the source of the request (like a symmetric NAT, but no translation; this is called a *symmetric UDP firewall*);
4. Full-cone NAT;

5. Symmetric NAT;
6. Restricted cone or restricted port cone NAT.

The determination of which of the six scenarios applies can be achieved through the flow chart shown in Figure 5.13. The chart refers only to the sequence of Binding Requests; Shared Secret Requests will, of course, be needed to authenticate each Binding Request used in the sequence. The flow makes use of three tests. In test I, the client sends a STUN Binding Request to a server, without any flags set in the CHANGE-REQUEST attribute, and without the RESPONSE-ADDRESS attribute. This causes the server to send the response back to the address and port that the request came from. In test II, the client sends a Binding Request with both the "change IP" and "change port" flags from the CHANGE-REQUEST attribute set. In test III, the client sends a Binding Request with only the "change port" flag set.

The client begins by initiating test I. If this test yields no response, the client knows right away that it is not capable of UDP connectivity. If the test produces a response, the client examines the MAPPED-ADDRESS attribute. If this address and port are the same as the local IP address and port of the socket used to send the request, the client knows that it is not NATed. It executes test II.

If a response is received, the client knows that it has open access to the Internet (or, at least, it's behind a firewall that behaves like a full-cone NAT, but without the translation). If no response is received, the client knows it is behind a symmetric UDP firewall.

In the event that the IP address and port of the socket did not match the MAPPED-ADDRESS attribute in the response to test I, the client knows that it is behind a NAT. It performs test II. If a response is received, the client knows that it is behind a full-cone NAT. If no response is received, it performs test I again, but this time, does so to the address and port from the CHANGED-ADDRESS attribute from the response to test I. If the IP address and port returned in the MAPPED-ADDRESS attribute are not the same as the ones from the first test I, the client knows it's behind a symmetric NAT. If the address and port are the same, the client is either behind a restricted or port restricted NAT. To make a determination about which one it is behind, the client initiates test III. If a response is received, it is behind a restricted NAT, and if no response is received, it is behind a port-restricted NAT.

This procedure yields substantial information about the operating condition of the client application. In the event of multiple NATs between the client and the Internet, the type that is discovered will be the type of the most restrictive NAT between the client and the Internet. The types of NAT, in order of restrictiveness, from most to least, are: symmetric, port-restricted cone, restricted cone, and full cone.

Typically, a client will redo this discovery process periodically to detect changes, or look for inconsistent results. It is important to note that when the discovery process is redone, it should not generally be done from the same local address and port used in the previous discovery process. If the same local address and port are reused, bindings from the previous test may still be in existence, and these will invalidate the results of the test. Using a different local address and port for subsequent tests resolves this problem. An alternative is to wait sufficiently long to be confident that the old bindings have expired (half an hour should more than suffice).

5.3.10.2 Binding Lifetime Discovery

STUN can also be used to discover the lifetimes of the bindings created by the NAT. In many cases, the client will need to refresh the binding, either through a new STUN request, or an application packet, in order for the application to continue to use the binding. By discovering the binding lifetime, the client can determine how frequently it needs to refresh.

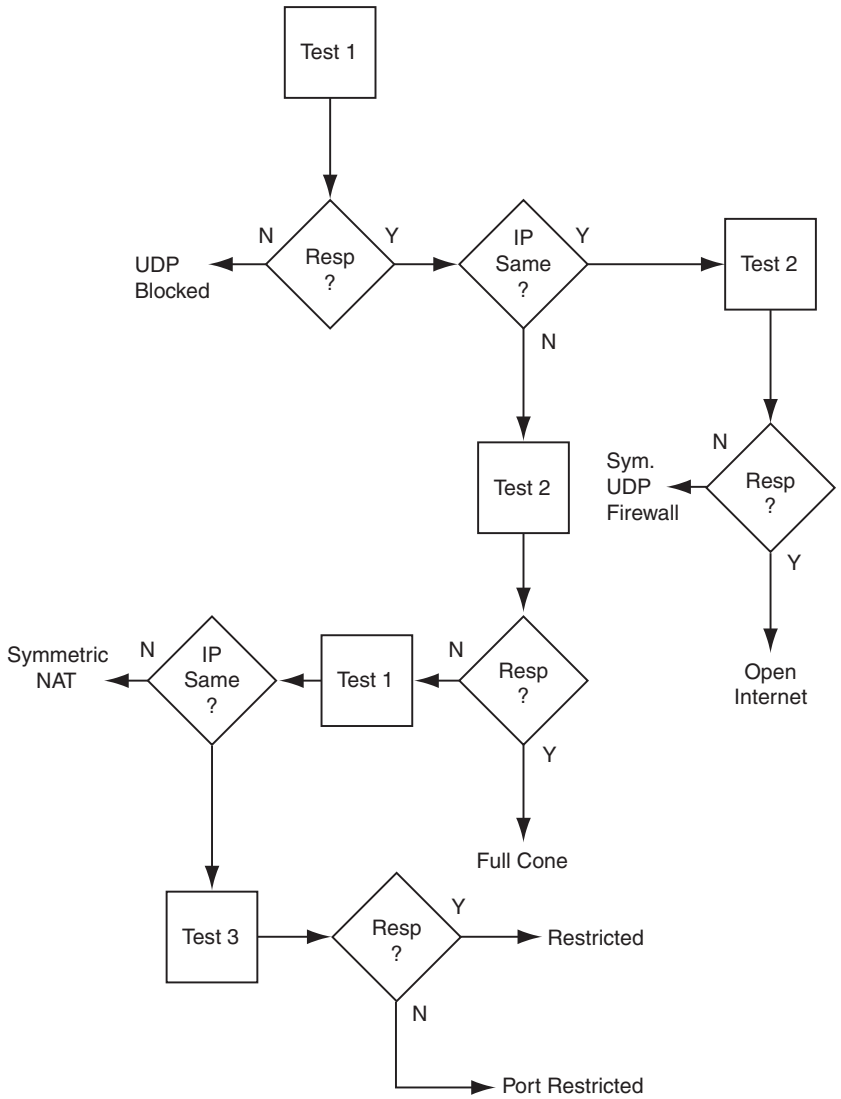


Figure 5.13: Flow for type discovery process.

To determine the binding lifetime, the client first sends a Binding Request to the server from a particular socket, X. This creates a binding in the NAT. The response from the server contains a MAPPED-ADDRESS attribute, providing the public address and port on the NAT. Call this Pa and Pp, respectively. The client then starts a timer with a value of T seconds. When this timer fires, the client sends another Binding Request to the server, using the same destination address and port, but from a different socket, Y. This request contains a RESPONSE-ADDRESS address attribute, set to (Pa,Pp). This will create a new binding on the NAT, and cause the STUN server to send a Binding Response that would match the old binding, if it still exists. If the client receives the Binding Response on socket X, it knows that the binding has not expired. If the client receives the Binding Response on socket Y (which

is possible if the old binding expired, and the NAT allocated the same public address and port to the new binding), or receives no response at all, it knows that the binding has expired.

The client can find the value of the binding lifetime by doing a binary search through T, arriving eventually at the value where the response is not received for any timer greater than T, but is received for any timer less than T.

This discovery process takes quite a bit of time, and is something that will typically be run in the background on a device once it boots.

It is possible that the client can get inconsistent results each time this process is run. For example, if the NAT should reboot, or be reset for some reason, the process may discover a lifetime than is shorter than the actual one. For this reason, implementations are encouraged to run the test numerous times, and be prepared to get inconsistent results.

5.3.10.3 Binding Acquisition

Consider once more the case of a VoIP phone. It used the discovery process above when it started up to discover its environment. Now, it wants to make a call. As part of the discovery process, it determined that it was behind a full-cone NAT.

Consider further that this phone consists of two logically separated components—a control component that handles signaling, and a media component that handles the audio, video, and RTP. Both are behind the same NAT. Because of this separation of control and media, we wish to minimize the communication required between them. In fact, they may not even run on the same host.

In order to make a voice call, the phone needs to obtain an IP address and port that it can place in the call setup message as the destination for receiving audio.

To obtain an address, the control component sends a Shared Secret Request to the server, obtains a shared secret, and then sends a Binding Request to the server. No CHANGE-REQUEST attribute is present in the Binding Request, and neither is the RESPONSE-ADDRESS attribute. The Binding Response contains a mapped address. The control component then formulates a second Binding Request. This request contains a RESPONSE-ADDRESS which is set to the mapped address learned from the previous Binding Response. This Binding Request is passed to the media component, along with the IP address and port of the STUN server. The media component sends the Binding Request. The request goes to the STUN server which sends the Binding Response back to the control component. The control component receives this, and now has learned an IP address and port that will be routed back to the media component that sent the request.

The client will be able to receive media from anywhere on this mapped address.

In the case of silence suppression, there may be periods where the client receives no media. In this case, the UDP bindings could timeout (UDP bindings in NATs are typically short; 30 seconds is common). To deal with this, the application can periodically retransmit the query in order to keep the binding fresh.

It is possible that both participants in the multimedia session are behind the same NAT. In that case, both will repeat this procedure above, and both will obtain public address bindings. When one sends media to the other, the media is routed to the NAT, and then turns right back around to come back into the enterprise, where it is translated to the private address of the recipient. This is not particularly efficient, and unfortunately, does not work in many commercial NATs. In such cases, the clients may need to retry using private addresses.

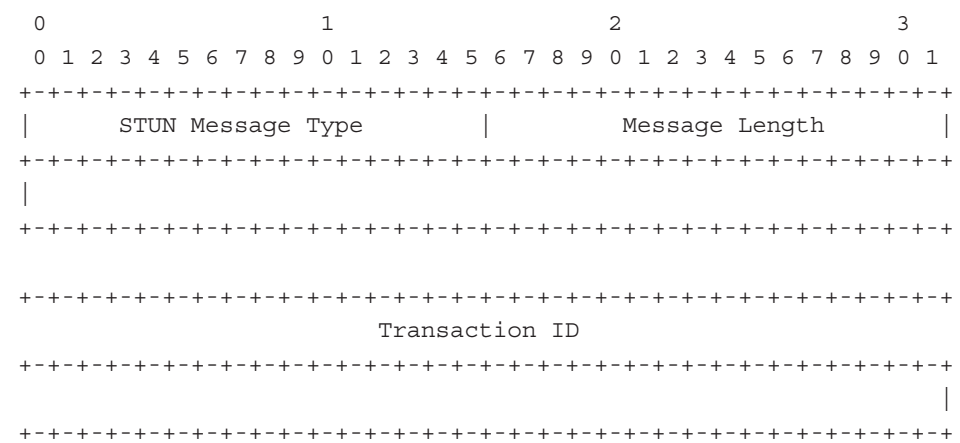
5.3.11 Protocol Details

This section presents the detailed encoding of a STUN message. As noted, STUN is a request-response protocol. Clients send a request, and the server sends a response. There are two requests, Binding Request, and Shared Secret Request. The response to a Binding Request can either be the Binding Response or Binding Error Response. The response to a Shared Secret Request can either be a Shared Secret Response or a Shared Secret Error Response.

STUN messages are encoded using binary fields. All integer fields are carried in network byte order, that is, most significant byte (octet) first. This byte order is commonly known as big-endian. The transmission order is described in detail in Appendix B of RFC 791. Unless otherwise noted, numeric constants are in decimal (base 10).

5.3.11.1 Message Header

All STUN messages consist of a 20 byte header:



The Message Types can take on the following values:

0x0001 : Binding Request

0x0101 : Binding Response

0x0111 : Binding Error Response

0x0002 : Shared Secret Request

0x0102 : Shared Secret Response

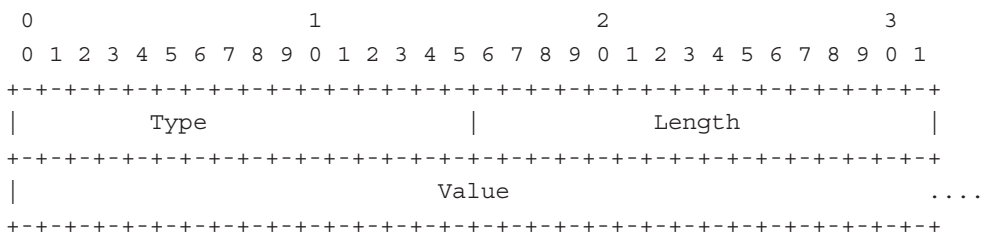
0x0112 : Shared Secret Error Response

The Message Length is the count, in bytes, of the size of the message, not including the 20 byte header.

The Transaction ID is a 128 bit identifier. It also serves as salt to randomize the request and the response. All responses carry the same identifier as the request they correspond to.

5.3.11.2 Message Attributes

After the header are 0 or more attributes. Each attribute is TLV encoded, with a 16-bit type, 16-bit length, and variable value:



The following types are defined:

0x0001: MAPPED-ADDRESS

0x0002: RESPONSE-ADDRESS

0x0003: CHANGE-REQUEST

0x0004: SOURCE-ADDRESS

0x0005: CHANGED-ADDRESS

0x0006: USERNAME

0x0007: PASSWORD

0x0008: MESSAGE-INTEGRITY

0x0009: ERROR-CODE

0x000a: UNKNOWN-ATTRIBUTES

0x000b: REFLECTED-FROM

To allow future revisions of the specification to add new attributes if needed, the attribute space is divided into optional and mandatory ones. Attributes with values greater than 0x7fff are optional, which means that the message can be processed by the client or server even though the attribute is not understood. Attributes with values less than or equal to 0x7fff are mandatory to understand, which means that the client or server cannot process the message unless it understands the attribute.

The MESSAGE-INTEGRITY attribute must be the last attribute within a message. Any attributes that are known, but are not supposed to be present in a message (MAPPED-ADDRESS in a request, for example) must be ignored.

Table 5.3 indicates which attributes are present in which messages. An M indicates that inclusion of the attribute in the message is mandatory, O means its optional, C means it is conditional based on some other aspect of the message, and N/A means that the attribute is not applicable to that message type.

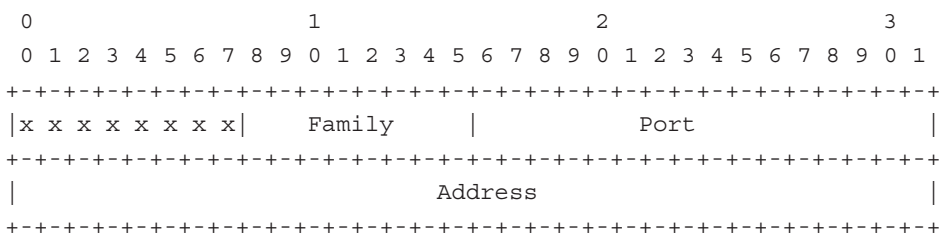
Table 5.3: Summary of Attributes

Att.	Binding Req.	Binding Resp.	Binding Error Resp.	Shared Secret Req.	Shared Secret Resp.	Secret Shared Error Resp.
MAPPED-ADDRESS	N/A	M	N/A	N/A	N/A	N/A
RESPONSE-ADDRESS	O	N/A	N/A	N/A	N/A	N/A
CHANGE-REQUEST	O	N/A	N/A	N/A	N/A	N/A
SOURCE-ADDRESS	N/A	M	N/A	N/A	N/A	N/A
CHANGED-ADDRESS	N/A	M	N/A	N/A	N/A	N/A
USERNAME	O	N/A	N/A	N/A	M	N/A
PASSWORD	N/A	N/A	N/A	N/A	M	N/A
MESSAGE-INTEGRITY	O	O	N/A	N/A	N/A	N/A
ERROR-CODE	N/A	N/A	M	N/A	N/A	M
UNKNOWN-ATTRIBUTES	N/A	N/A	C	N/A	N/A	C
REFLECTED-FROM	N/A	C	N/A	N/A	N/A	N/A

The length refers to the length of the value element, expressed as an unsigned integral number of bytes.

MAPPED-ADDRESS

The MAPPED-ADDRESS attribute indicates the mapped IP address and port. It consists of an eight-bit address family, and a sixteen bit port, followed by a fixed length value representing the IP address.



The port is a network byte-ordered representation of the mapped port. The address family is always 0x01, corresponding to IPv4. The first 8 bits of the MAPPED-ADDRESS are ignored, for the purposes of aligning parameters on natural boundaries. The IPv4 address is 32 bits.

RESPONSE-ADDRESS

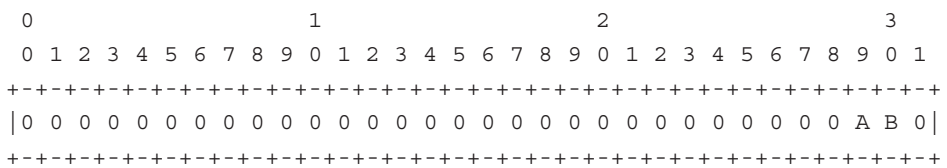
The RESPONSE-ADDRESS attribute indicates where the response to a Binding Request should be sent. Its syntax is identical to MAPPED-ADDRESS.

CHANGED-ADDRESS

The CHANGED-ADDRESS attribute indicates the IP address and port where responses would have been sent from if the "change IP" and "change port" flags had been set in the CHANGE-REQUEST attribute of the Binding Request. The attribute is always present in a Binding Response, independent of the value of the flags. Its syntax is identical to MAPPED-ADDRESS.

CHANGE-REQUEST

The CHANGE-REQUEST attribute is used by the client to request that the server use a different address and/or port when sending the response. The attribute is 32 bits long, although only two bits (A and B) are used:



The meaning of the flags is:

A: This is the "change IP" flag. If true, it requests the server to send the Binding Response with a different IP address than the one the Binding Request was received on.

B: This is the "change port" flag. If true, it requests the server to send the Binding Response with a different port than the one the Binding Request was received on.

SOURCE-ADDRESS

The SOURCE-ADDRESS attribute is present in Binding Responses. It indicates the source IP address and port that the server is sending the response from. Its syntax is identical to that of MAPPED-ADDRESS.

USERNAME

The USERNAME attribute is used for message integrity. It serves as a means to identify the shared secret used in the message integrity check. The USERNAME is always present in a Shared Secret Response, along with the PASSWORD. It is optionally present in a Binding Request when message integrity is used.

The value of USERNAME is a variable length opaque value. Its length MUST be a multiple of 4 (measured in bytes) in order to guarantee alignment of attributes on word boundaries.

PASSWORD

The PASSWORD attribute is used in Shared Secret Responses. It is always present in a Shared Secret Response, along with the USERNAME.

The value of PASSWORD is a variable length value that is to be used as a shared secret. Its length MUST be a multiple of 4 (measured in bytes) in order to guarantee alignment of attributes on word boundaries.

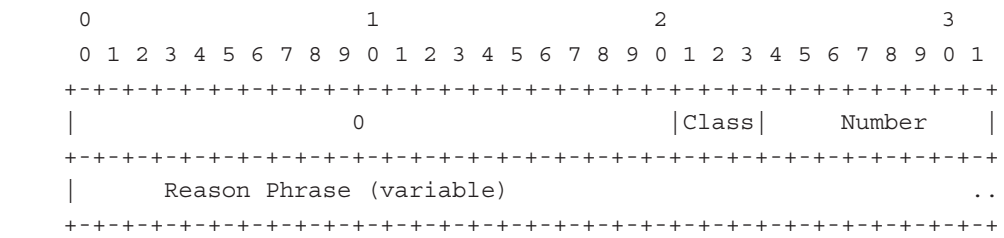
MESSAGE-INTEGRITY

The MESSAGE-INTEGRITY attribute contains an HMAC-SHA1 of the STUN message. It can be present in Binding Requests or Binding Responses. Since it uses the SHA1 hash, the HMAC will be 20 bytes. The text used as input to HMAC is the STUN message, including the header, up to and including the attribute preceding the MESSAGE-INTEGRITY attribute. That text is then padded with zeroes so as to be a multiple of 64 bytes. As a result, the MESSAGE-INTEGRITY attribute must be the last attribute in any STUN message. The key used as input to HMAC depends on the context.

ERROR-CODE

The ERROR-CODE attribute is present in the Binding Error Response and Shared Secret Error Response. It is a numeric value in the range of 100 to 699 plus a textual reason phrase encoded in UTF-8, and is consistent in its code assignments and semantics with SIP and HTTP. The reason phrase is meant for user consumption, and can be anything appropriate for the response code. The lengths of the reason phrases must be a multiple of 4 (measured in bytes). This can be accomplished by added spaces to the end of the text, if necessary. Recommended reason phrases for the defined response codes are presented below.

To facilitate processing, the class of the error code (the hundreds digit) is encoded separately from the rest of the code.



The class represents the hundreds digit of the response code. The value must be between 1 and 6. The number represents the response code modulo 100, and its value must be between 0 and 99.

The following response codes, along with their recommended reason phrases (in brackets) are defined at this time.

400 (Bad Request): The request was malformed. The client should not retry the request without modification from the previous attempt.

401 (Unauthorized): The Binding Request did not contain a MESSAGE-INTEGRITY attribute.

420 (Unknown Attribute): The server did not understand a mandatory attribute in the request.

430 (Stale Credentials): The Binding Request did contain a MESSAGE-INTEGRITY attribute, but it used a shared secret that has expired. The client should obtain a new shared secret and try again.

431 (Integrity Check Failure): The Binding Request contained a MESSAGE-INTEGRITY attribute, but the HMAC failed verification. This could be a sign of a potential attack or client implementation error.

432 (Missing Username): The Binding Request contained a MESSAGE-INTEGRITY attribute, but not a USERNAME attribute. Both must be present for integrity checks.

433 (Use TLS): The Shared Secret request has to be sent over TLS, but was not received over TLS.

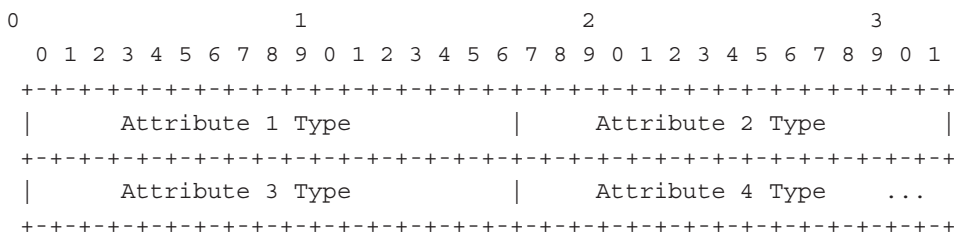
500 (Server Error): The server has suffered a temporary error. The client should try again.

600 (Global Failure): The server is refusing to fulfill the request. The client should not retry.

UNKNOWN-ATTRIBUTES

The UNKNOWN-ATTRIBUTES attribute is present only in a Binding Error Response or Shared Secret Error Response when the response code in the ERROR-CODE attribute is 420.

The attribute contains a list of 16 bit values, each of which represents an attribute type that was not understood by the server. If the number of unknown attributes is an odd number, one of the attributes must be repeated in the list, so that the total length of the list is a multiple of 4 bytes.



REFLECTED-FROM

The REFLECTED-FROM attribute is present only in Binding Responses, when the Binding Request contained a RESPONSE-ADDRESS attribute. The attribute contains the identity (in terms of IP address) of the source where the request came from. Its purpose is to provide traceability, so that a STUN server cannot be used as a reflector for denial-of-service attacks. Its syntax is identical to the MAPPED-ADDRESS attribute.

5.4 Overview of MIDCOM Approaches

This section looks at the newly-defined topic of Middlebox Communications (MIDCOM), which was alluded to above in the context of STUN. A principal objective of RFC 3303 is to describe the underlying framework of MIDCOM to enable complex applications through the middleboxes, seamlessly using a trusted third party. This discussion is based on RFC 3303 [SRI200201]. Developers should refer to the original RFC and all supportive extensions, updates, etc., for normative development guidance.

5.4.1 Background

There are a variety of intermediate devices in the Internet today that require application intelligence for their operation. Datagrams pertaining to real-time streaming applications, such as SIP and H.323, and peer-to-peer applications, such as Napster and NetMeeting, cannot be identified by merely examining packet headers. Middleboxes implementing Firewall and Network Address Translator services typically embed application intelligence within the device for their operation. The document specifies an architecture and framework in which trusted third parties can be delegated to assist the middleboxes to perform their operation, without resorting to embedding application intelligence. Doing this will allow a middlebox to continue to provide the services while keeping the middlebox application agnostic.

Intermediate devices requiring application intelligence are the subject of RFC 3303. These devices are referred to as middleboxes throughout the document. Many of these devices enforce application-specific policy-based functions such as packet filtering, VPN (Virtual Private Network) tunneling, Intrusion detection, security, and so forth. Network Address Translator service, on the other hand, provides routing transparency across address realms (within IPv4 routing network or across V4 and V6 routing realms) independent of applications. Application Level Gateways (ALGs) are used in conjunction with NAT to examine and optionally modify application payload so the end-to-end application behavior remains unchanged for many of the applications traversing NAT middleboxes. There may be other types of services requiring embedding application intelligence in middleboxes for their operation. The discussion scope of this RFC is however limited to Firewall and NAT services. Nonetheless, the MIDCOM framework is designed to be extensible to support the deployment of new services.

Tight coupling of application intelligence with middleboxes makes maintenance of middleboxes hard with the advent of new applications. Built-in application awareness typically requires updates of operating systems with new applications or newer versions of existing applications. Operators requiring support for newer applications will not be able to use third party software/hardware specific to the application and are at the

mercy of their middlebox vendor to make the necessary upgrade. Further, embedding intelligence for a large number of application protocols within the same middlebox increases complexity of the middlebox and is likely to be error prone and degrade in performance.

RFC 3303 describes a framework in which application intelligence can be moved from middleboxes into external MIDCOM agents. The premise of the framework is to devise a MIDCOM protocol that is application independent so the middleboxes can stay focused on services such as firewall and NAT. The framework document includes some explicit and implied requirements for the MIDCOM protocol. However, it must be noted that these requirements are only a subset. A separate requirements document lists the requirements in detail.

MIDCOM agents with application intelligence can assist the middleboxes through the MIDCOM protocol in permitting applications such as FTP, SIP and H.323. The communication between a MIDCOM agent and a middlebox will not be noticeable to the end-hosts that take part in the application, unless one of the end-hosts assumes the role of a MIDCOM agent. Discovery of middleboxes or MIDCOM agents in the path of an application instance is outside the scope of this RFC. Further, any communication amongst middleboxes is also outside the scope of RFC 3303.

RFC 3303 describes the framework in which middlebox communication takes place and the various elements that constitute the framework. Section 5.4.2 describes the terms used in the document. Section 5.4.3 defines the architectural framework of a middlebox for communication with MIDCOM agents. The remaining sections cover the components of the framework, illustration using sample flows, and operational considerations with the MIDCOM architecture. Section 5.4.4 describes the nature of MIDCOM protocol. Section 5.4.5 identifies entities that could potentially host the MIDCOM agent function. Section 5.4.6 considers the role of Policy server and its function with regard to communicating MIDCOM agent authorization policies. Section 5.4.7 is an illustration of SIP flows using a MIDCOM framework in which the MIDCOM agent is co-resident on a SIP proxy server. Section 5.4.8 addresses operational considerations in deploying a protocol adhering to the framework described here. Section 5.4.9 is an applicability statement, scoping the location of middleboxes.

5.4.2 Terminology

Below are the definitions for the terms used in RFC 3303.

5.4.2.1 Middlebox Function/Service

A middlebox function or a middlebox service is an operation or method performed by a network intermediary that may require application-specific intelligence for its operation. Policy-based packet filtering (a.k.a. firewall), Network Address Translation (NAT), Intrusion detection, Load balancing, Policy-based tunneling, and IPsec security are all examples of a middlebox function (or service).

5.4.2.2 Middlebox

A middlebox is a network intermediate device that implements one or more of the middlebox services. A NAT middlebox is a middlebox implementing NAT service. A firewall middlebox is a middlebox implementing firewall service.

Traditional middleboxes embed application intelligence within the device to support specific application traversal. Middleboxes supporting the MIDCOM protocol will be able to externalize application intelligence into MIDCOM agents. In reality, some of the middleboxes may continue to embed application intelligence for certain applications and depend on MIDCOM protocol and MIDCOM agents for the support of remaining applications.

5.4.2.3 Firewall

Firewall is a policy-based packet-filtering middlebox function, typically used for restricting access to/from specific devices and applications. The policies are often termed Access Control Lists (ACLs).

5.4.2.4 NAT

Network Address Translation is a method by which IP addresses are mapped from one address realm to another, providing transparent routing to end-hosts. Transparent routing here refers to modifying end-node addresses en route and maintaining state for these updates so that when a datagram leaves one realm and enters another, datagrams pertaining to a session are forwarded to the right end-host in either realm. Refer to RFC 2663 for the definition of Transparent routing, various NAT types, and the associated terms in use. Two types of NAT are most common. Basic-NAT, where only an IP address (and the related IP, TCP/UDP checksums) of packets is altered and NAPT (Network Address Port Translation), where both an IP address and a transport layer identifier, such as a TCP/UDP port (and the related IP, TCP/UDP checksums), are altered.

The term NAT here is very similar to the IPv4 NAT described in RFC 2663, but is extended beyond IPv4 networks to include the IPv4-v6 NAT-PT described in RFC 2766. While the IPv4 NAT translates one IPv4 address into another IPv4 address to provide routing between private v4 and external V4 address realms, IPv4-v6 NAT-PT (RFC 2766) translates an IPv4 address into an IPv6 address, and vice versa, to provide routing between a v6 address realm and an external v4 address realm. Unless specified otherwise, NAT is a middlebox function referring to both IPv4 NAT, as well as IPv4-v6 NAT-PT.

5.4.2.5 Proxy

A proxy is an intermediate relay agent between clients and servers of an application, relaying application messages between the two. Proxies use special protocol mechanisms to communicate with proxy clients and relay client data to servers and vice versa. A proxy terminates sessions with both the client and the server, acting as server to the end-host client and as client to the end-host server.

Applications such as FTP, SIP, and RTSP use a control session to establish data sessions. These control and data sessions can take divergent paths. While a proxy can intercept both the control and data sessions, it might intercept only the control session. This is often the case with real-time streaming applications such as SIP and RTSP.

5.4.2.6 ALG

Application Level Gateways are entities that possess the application-specific intelligence and knowledge of an associated middlebox function. They examine application traffic in transit and assist the middlebox in carrying out its function.

An ALG may be a co-resident with a middlebox or reside externally, communicating through a middlebox communication protocol. It interacts with a middlebox to set up state, access control filters, use middlebox state information, modify application specific payload, or perform whatever else is necessary to enable the application to run through the middlebox.

ALGs are different from proxies in that they are not visible to end-hosts, unlike the proxies which are relay agents terminating sessions with both end-hosts. They do not terminate sessions with either end-host. Instead, they examine, and optionally modify, application payload content to facilitate the flow of application traffic through a middlebox. ALGs are middlebox centric, in that they assist the middleboxes in carrying out their function, whereas, the proxies act as a focal point for application servers, relaying traffic between application clients and servers.

ALGs are similar to Proxies, in that both ALGs and proxies facilitate application-specific communication between clients and servers.

5.4.2.7 End-Hosts

End-hosts are entities that are party to a networked application instance. End-hosts referred to in this RFC, are specifically those terminating Real-time streaming Voice-over-IP applications such as SIP and H.323, and peer-to-peer applications such as Napster and NetMeeting.

5.4.2.8 MIDCOM Agents

MIDCOM agents are entities performing ALG functions, logically external to a middlebox. MIDCOM agents possess a combination of application awareness and knowledge of the middlebox function. This combination enables the agents to facilitate traversal of the middlebox by the application's packets. A MIDCOM agent may interact with one or more middleboxes.

Only "In-Path MIDCOM agents" are considered in this RFC. In-Path MIDCOM agents are agents which are within the path of those datagrams that the agent needs to examine and/or modify in fulfilling its role as a MIDCOM agent. "Within the path" here simply means that the packets in question flow through the node that hosts the agent. The packets may be addressed to the agent node at the IP layer. Alternatively, they may not be addressed to the agent node, but may be constrained by other factors to flow through it. In fact, it is immaterial to the MIDCOM protocol which of these is the case. Some examples of In-Path MIDCOM agents are application proxies, gateways, or even end-hosts that are party to the application.

Agents not resident on nodes that are within the path of their relevant application flows are referred to as "Out-of-Path (OOP) MIDCOM agents" and are out of the scope of this RFC.

5.4.2.9 MIDCOM PDP

MIDCOM Policy Decision Point (PDP) is primarily a Policy Decision Point (PDP) as defined in RFC 3198; and also acts as a policy repository, holding MIDCOM-related policy profiles in order to make authorization decisions. RFC 3198 defines a PDP as "a logical entity that makes policy decisions for itself or for other network elements that request such decisions"; and a policy repository as "a specific data store that holds policy rules, their conditions and actions, and related policy data."

A middlebox and a MIDCOM PDP may communicate further if the MIDCOM PDP's policy changes or if a middlebox needs further information. The MIDCOM PDP may, at any time, notify the middlebox to terminate authorization for an agent.

The protocol facilitating the communication between a middlebox and MIDCOM PDP need not be part of the MIDCOM protocol. Section 5.4.6 in the document addresses the MIDCOM PDP interface and protocol framework independent of the MIDCOM framework.

Application-specific policy data and policy interface between an agent or application endpoint and a MIDCOM PDP is out of bounds for this RFC. The MIDCOM PDP issues addressed in the document are focused at an aggregate domain level as befitting the middlebox. For example, a SIP MIDCOM agent may choose to query a MIDCOM PDP for the administrative (or corporate) domain to find whether a certain user is allowed to make an outgoing call. This type of application-specific policy data, as befitting an end user, is out of bounds for the MIDCOM PDP considered in this RFC. It is within bounds, however, for the MIDCOM PDP to specify the specific end-user applications (or tuples) for which an agent is permitted to be an ALG.

5.4.2.10 Middlebox Communication (MIDCOM) protocol

The protocol between a MIDCOM agent and a middlebox allows the MIDCOM agent to invoke services of the middlebox and allow the middlebox to delegate application specific processing to the MIDCOM agent. The MIDCOM protocol allows the middlebox to perform its operation with the aid of MIDCOM agents, without resorting to embedding application intelligence. The principal motivation behind architecting this protocol is to enable complex applications through middleboxes, seamlessly using a trusted third party, i.e., a MIDCOM agent.

This is a protocol yet to be devised.

5.4.2.11 MIDCOM Agent Registration

A MIDCOM agent registration is defined as the process of provisioning agent profile information with the middlebox or a MIDCOM PDP. MIDCOM agent registration is often a manual operation performed by an operator rather than the agent itself.

A MIDCOM agent profile may include agent authorization policy (i.e., session tuples for which the agent is authorized to act as ALG), agent-hosting-entity (e.g., Proxy, Gateway, or end-host which hosts the agent), agent accessibility profile (including any host level authentication information), and security profile (for the messages exchanged between the middlebox and the agent).

5.4.2.12 MIDCOM Session

A MIDCOM session is defined to be a lasting association between a MIDCOM agent and a middlebox. The MIDCOM session is not assumed to imply any specific transport layer protocol. Specifically, this should not be construed as referring to a connection-oriented TCP protocol.

5.4.2.13 Filter

A filter is packet matching information that identifies a set of packets to be treated a certain way by a middlebox. This definition is consistent with RFC 3198, which defines a filter as "A set of terms and/or criteria used for the purpose of separating or categorizing. This is accomplished via single- or multifield matching of traffic header and/or payload data."

5-Tuple specification of packets in the case of a firewall and 5-tuple specification of a session in the case of a NAT middlebox function are examples of a filter.

5.4.2.14 Policy action (or) Action

Policy action (or Action) is a description of the middlebox treatment/service to be applied to a set of packets. This definition is consistent with RFC 3198, which defines a policy action as "Definition of what is to be done to enforce a policy rule, when the conditions of the rule are met. Policy actions may result in the execution of one or more operations to affect and/or configure network traffic and network resources."

NAT Address-BIND (or Port-BIND in the case of NAPT) and firewall permit/deny action are examples of an Action.

5.4.2.15 Policy Rule(s)

The combination of one or more filters and one or more actions. Packets matching a filter are to be treated as specified by the associated action(s). The Policy rules may also contain auxiliary attributes such as individual rule type, timeout values, creating agent, etc.

Policy rules are communicated through the MIDCOM protocol.

5.4.3 Architectural Framework for Middleboxes

A middlebox may implement one or more of the middlebox functions selectively on multiple interfaces of the device. There can be a variety of MIDCOM agents interfacing with the middlebox to communicate with one or more of the middlebox functions on an interface. As such, the middlebox communication protocol must allow for selective communication between a specific MIDCOM agent and one or more middlebox functions on the interface. Figure 5.14 identifies a possible layering of the service supported by a middlebox and a list of MIDCOM agents that might interact with it.

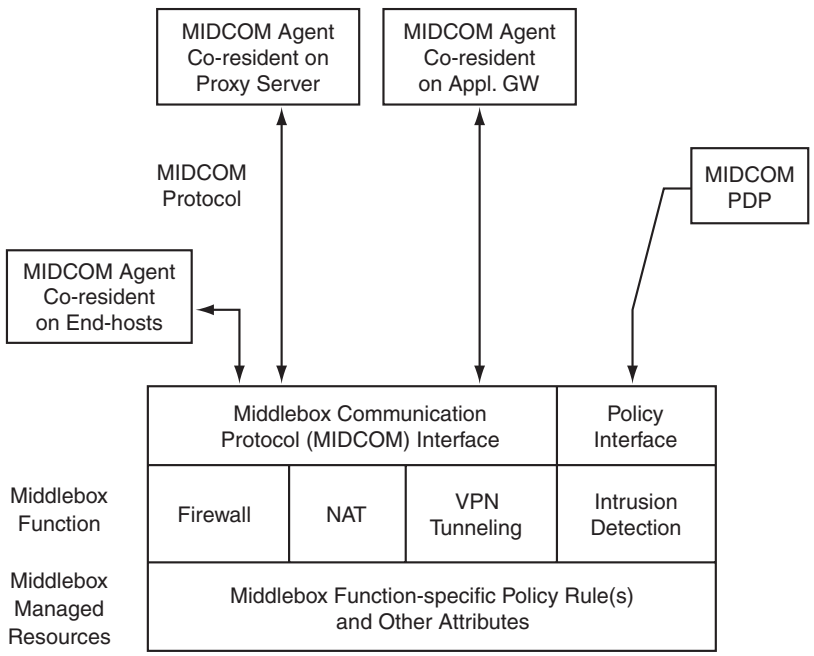


Figure 5.14: MIDCOM agents interfacing with a middlebox.

Firewall ACLs, NAT-BINDs, NAT address-maps, and Session-state are a few of the middlebox function-specific policy rules. A session state may include middlebox function-specific attributes, such as timeout values, NAT translation parameters (i.e., NAT-BINDS), and so forth. As Session-state may be shared across middlebox functions, a Session-state may be created by a function, and terminated by a different function. For example, a session-state may be created by the firewall function, but terminated by the NAT function, when a session timer expires.

Application specific MIDCOM agents (co-resident on the middlebox or external to the middlebox) would examine the IP datagrams and help identify the application the datagram belongs to, and assist the middlebox in performing functions unique to the application and the middlebox service. For example, a MIDCOM agent, assisting a NAT middlebox, might perform payload translations, whereas a MIDCOM agent assisting a firewall middlebox might request the firewall to permit access to application-specific, dynamically-generated session traffic.

5.4.4 MIDCOM Protocol

The MIDCOM protocol between a MIDCOM agent and a middlebox allows the MIDCOM agent to invoke services of the middlebox and allow the middlebox to delegate application-specific processing to the

MIDCOM agent. The protocol will allow MIDCOM agents to signal the middleboxes, to let complex applications using dynamic port-based sessions through them (i.e., middleboxes) seamlessly.

It is important to note that an agent and a middlebox can be on the same physical device. In such a case, they may communicate using a MIDCOM protocol message format (but using a non-IP based transport, such as IPC messaging), (or) they may communicate using well-defined API/DLL, (or) the application intelligence is fully embedded into the middlebox service (as it is done today in many stateful inspection firewall devices and NAT devices).

The MIDCOM protocol will consist of a session setup phase, run-time session phase, and a session termination phase.

Session setup must be preceded by registration of the MIDCOM agent with either the middlebox or the MIDCOM PDP. The MIDCOM agent access and authorization profile may either be preconfigured on the middlebox (or) listed on a MIDCOM PDP; the middlebox is configured to consult. MIDCOM shall be a client-server protocol initiated by the agent.

A MIDCOM session may be terminated by either of the parties. A MIDCOM session termination may also be triggered by (a) the middlebox or the agent going out of service and not being available for further MIDCOM operations, or (b) the MIDCOM PDP notifying the middlebox that a particular MIDCOM agent is no longer authorized.

The MIDCOM protocol data exchanged during runtime is governed principally by the middlebox services the protocol supports. Firewall and NAT middlebox services are considered in this RFC. Nonetheless, the MIDCOM framework is designed to be extensible to support the deployment of other services as well.

5.4.5 MIDCOM Agents

MIDCOM agents are logical entities which may reside physically on nodes external to a middlebox, possessing a combination of application awareness and knowledge of middlebox function. A MIDCOM agent may communicate with one or more middleboxes. The issues of middleboxes discovering agents, or vice versa, are outside the scope of this RFC. The focus of the document is the framework in which a MIDCOM agent communicates with a middlebox using MIDCOM protocol, which is yet to be devised. Specifically, the focus is restricted to just the In-Path agents.

In-Path MIDCOM agents are MIDCOM agents that are located naturally within the message path of the application(s) they are associated with. Bundled session applications, such as H.323, SIP, and RTSP which have separate control and data sessions, may have their sessions take divergent paths. In those scenarios, In-Path MIDCOM agents are those that find themselves in the control path. In a majority of cases, a middlebox will likely require the assistance of a single agent for an application in the control path alone. However, it is possible that a middlebox function, or a specific application traversing the middlebox might require the intervention of more than a single MIDCOM agent for the same application, one for each sub-session of the application.

Application Proxies and gateways are a good choice for In-Path MIDCOM agents as these entities, by definition, are in the path of an application between a client and server. In addition to hosting the MIDCOM agent function, these natively in-path application-specific entities may also enforce application-specific choices locally, such as dropping messages infected with known viruses or lacking user authentication. These entities can be interjecting both the control and data sessions. For example, FTP control and Data sessions are interjected by an FTP proxy server.

However, proxies may also be interjecting just the control session and not the data sessions, as is the case with real-time streaming applications such as SIP and RTSP. Note, applications may not always traverse a proxy and some applications may not have a proxy server available.

SIP proxies and H.323 gatekeepers may be used to host MIDCOM agent functions to control middleboxes implementing firewall and NAT functions. The advantage of using in-path entities, as opposed to creating an entirely new agent, is that the in-path entities already possess application intelligence. You will need to merely enable the use of the MIDCOM protocol to be an effective MIDCOM agent. Figure 5.15 illustrates a scenario where the in-path MIDCOM agents interface with the middlebox. Let us say, the MIDCOM PDP has preconfigured the in-path proxies as trusted MIDCOM agents on the middlebox and the packet filter implements a 'default-deny' packet filtering policy. Proxies use their application-awareness knowledge to control the firewall function and selectively permit a certain number of voice stream sessions dynamically using MIDCOM protocol.

In the illustration below, the proxies and the MIDCOM PDP are shown inside a private domain. The intent however, is not to imply that they be inside the private boundary alone. The proxies may also reside external to the domain. The only requirement is that there be a trust relationship with the middlebox.

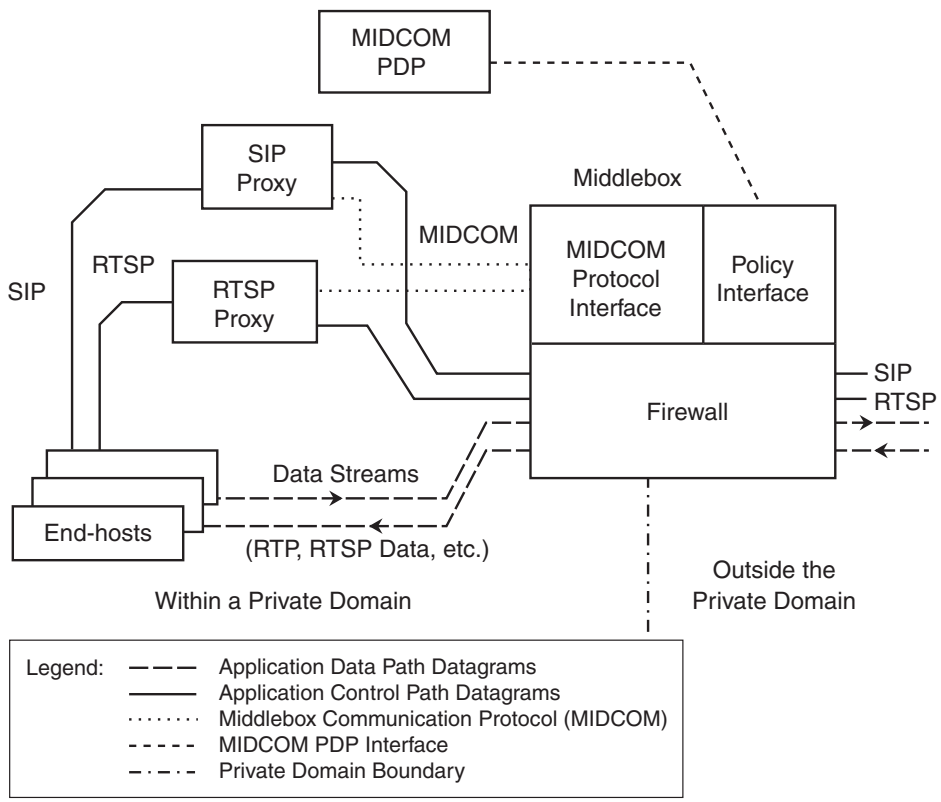


Figure 5.15: In-path MIDCOM agents for middlebox communication.

5.4.5.1 End-hosts as In-path MIDCOM Agents

End-hosts are another variation of In-Path MIDCOM agents. Unlike Proxies, End-hosts are a direct party to the application and possess all the end-to-end application intelligence there is to it. End-hosts presumably terminate both the control and data paths of an application. Unlike other entities hosting MIDCOM agents, end-host is able to process secure datagrams. However, the problem would be one of manageability—upgrading all the end-hosts running a specific application.

5.4.6 MIDCOM PDP Functions

The functional decomposition of the MIDCOM architecture assumes the existence of a logical entity, known as MIDCOM PDP, responsible for performing authorization and related provisioning services for the middlebox as depicted in Figure 5.14. The MIDCOM PDP is a logical entity which may reside physically on a middlebox or on a node external to the middlebox. The protocol employed for communication between the middlebox and the MIDCOM PDP is unrelated to the MIDCOM protocol.

Agents are registered with a MIDCOM PDP for authorization to invoke services of the middlebox. The MIDCOM PDP maintains a list of agents that are authorized to connect to each of the middleboxes the MIDCOM PDP supports. In the context of the MIDCOM Framework, the MIDCOM PDP does not assist a middlebox in the implementation of the services it provides.

The MIDCOM PDP acts in an advisory capacity to a middlebox, to authorize or terminate authorization for an agent attempting connectivity to the middlebox. The primary objective of a MIDCOM PDP is to communicate agent authorization information so as to ensure that the security and integrity of a middlebox is not jeopardized. Specifically, the MIDCOM PDP should associate a trust level with each agent attempting to connect to a middlebox and provide a security profile. The MIDCOM PDP should be capable of addressing cases when end-hosts are agents to the middlebox.

5.4.6.1 Authentication, Integrity and Confidentiality

Host authenticity and individual message security are two distinct types of security considerations. Host authentication refers to credentials required of a MIDCOM agent to authenticate itself to the middlebox and vice versa. When authentication fails, the middlebox must not process signaling requests received from the agent that failed authentication. Two-way authentication should be supported. In some cases, the two-way authentication may be tightly linked to the establishment of keys to protect subsequent traffic. Two-way authentication is often required to prevent various active attacks on the MIDCOM protocol and secure establishment of keying material.

Security services such as authentication, data integrity, confidentiality and replay protection may be adapted to secure MIDCOM messages in an untrusted domain. Message authentication is the same as data origin authentication and is an affirmation that the sender of the message is who it claims to be. Data integrity refers to the ability to ensure that a message has not been accidentally (maliciously or otherwise) altered or destroyed. Confidentiality is the encryption of a message with a key, so that only those in possession of the key can decipher the message content. Lastly, replay protection is a form of sequence integrity, so when an intruder plays back a previously-recorded sequence of messages, the receiver of the replay messages will simply drop the replay messages into bit-bucket. Certain applications of the MIDCOM protocol might require support for nonrepudiation as an option of the data integrity service. Typically, support for nonrepudiation is required for billing, service level agreements, payment orders, and rec eipts for delivery of service.

IPsec IP Authentication Header (AH) offers data-origin authentication, data integrity and protection from message replay. IPsec Encapsulating Security Payload (ESP) provides data-origin authentication to a lesser degree (same as IPsec AH if the MIDCOM transport protocol turns out to be TCP or UDP), message confidentiality, data integrity, and protection from replay. Besides the IPsec based protocols, there are other security options as well. TLS based transport layer security is one option. There are also many application-layer security mechanisms available. Simple Source-address based security is a minimal form of security and should be relied on only in the most trusted environments, where those hosts will not be spoofed.

The MIDCOM message security shall use existing standards, whenever the existing standards satisfy the requirements. Security shall be specified to minimize the impact on sessions that do not use the security option. Security should be designed to avoid introducing, and to minimize the impact of, denial of service

attacks. Some security mechanisms and algorithms require substantial processing or storage, in which case the security protocols should protect themselves as well against possible flooding attacks that overwhelm the endpoint (i.e., the middlebox or the agent) with such processing. For connection-oriented protocols (such as TCP) using security services, the security protocol should detect premature closure or truncation attacks.

5.4.6.2 Registration and Deregistration of MIDCOM Agents

Prior to allowing MIDCOM agents to invoke services of the middlebox, a registration process must take place. Registration is a different process than establishing a MIDCOM session. The former requires provisioning agent profile information with the middlebox or a MIDCOM PDP. Agent registration is often a manual operation performed by an operator rather than the agent itself. Setting up a MIDCOM session refers to establishing a MIDCOM transport session and exchanging security credentials between an agent and a middlebox. The transport session uses the registered information for session establishment.

Profile of a MIDCOM agent includes agent authorization policy (i.e., session tuples for which the agent is authorized to act as ALG), agent-hosting-entity (e.g., Proxy, Gateway or end-host which hosts the agent), agent accessibility profile (including any host level authentication information), and security profile (i.e., security requirements for messages exchanged between the middlebox and the agent).

MIDCOM agent profile may be preconfigured on a middlebox. Subsequent to that, the agent may choose to initiate a MIDCOM session prior to any data traffic. For example, MIDCOM agent authorization policy for a middlebox service may be preconfigured by specifying the agent in conjunction with a filter. In the case of a firewall, for example, the ACL tuple may be altered to reflect the optional Agent presence. The revised ACL may look something like the following.

```
(<Session-Direction>, <Source-Address>, <Destination-Address>, <IP-
Protocol>, <Source-Port>, <Destination-Port>, <Agent>)
```

The reader should note that this is an illustrative example and not necessarily the actual definition of an ACL tuple. The formal description of the ACL is yet to be devised. Agent accessibility information should also be provisioned. For a MIDCOM agent, accessibility information includes the IP address, trust level, host authentication parameters, and message authentication parameters. Once a session is established between a middlebox and a MIDCOM agent, that session should be usable with multiple instances of the application(s), as appropriate. Note, all of this could be captured in an agent profile for ease of management.

The technique described above is necessary for the pre-registration of MIDCOM agents with the middlebox. The middlebox provisioning may remain unchanged, if the middlebox learns of the registered agents through a MIDCOM PDP. In either case, the MIDCOM agent should initiate the session prior to the start of the application. If the agent session is delayed until after the application has started, the agent might be unable to process the control stream to permit the data sessions. When a middlebox notices an incoming MIDCOM session, and the middlebox has no prior profile of the MIDCOM agent, the middlebox will consult its MIDCOM PDP for authenticity, authorization, and trust guidelines for the session.

5.4.7 MIDCOM Framework Illustration Using an In-Path Agent

In Figure 5.16, one considers SIP applications to illustrate the operation of the MIDCOM protocol. Specifically, the application assumes that a caller, external to a private domain, initiates the call. The middlebox is assumed to be located at the edge of the private domain. A SIP phone (SIP User Agent Client/Server) inside the private domain is capable of receiving calls from external SIP phones. The caller uses a SIP Proxy, node located external to the private domain, as its outbound proxy. No interior proxy is assumed for the callee. Lastly, the external SIP proxy node is designated to host the MIDCOM agent function.

Arrows 1 and 8 in the figure below refer to a SIP call setup exchange between the external SIP phone and the SIP proxy. Arrows 4 and 5 refer to a SIP call setup exchange between the SIP proxy and the interior SIP phone, and are assumed to be traversing the middlebox. Arrows 2, 3, 6 and 7 below, between the SIP proxy and the middlebox, refer to MIDCOM communication. Na and Nb represent RTP/RTCP media traffic path in the external network. Nc and Nd represent media traffic inside the private domain.

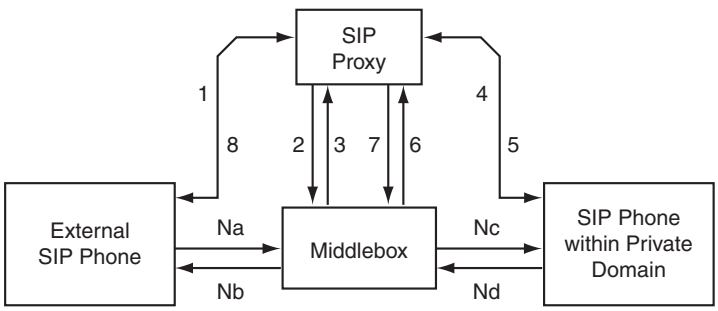


Figure 5.16: MIDCOM framework illustration with in-path SIP proxy.

As for the SIP application, we make the assumption that the middlebox is preconfigured to accept SIP calls into the private SIP phone. Specifically, this would imply that the middlebox implementing firewall service is preconfigured to permit SIP calls (destination TCP or UDP port number set to 5060) into the private phone. Likewise, middlebox implementing NAPT service would have been preconfigured to provide a port binding, to permit incoming SIP calls to be redirected to the specific private SIP phone. In other words, the INVITE from the external caller is not made to the private IP address but to the NAPT external address.

The objective of the MIDCOM agent in the following illustration is to merely permit the RTP/RTCP media stream through the middlebox, when using the MIDCOM protocol architecture outlined in the document. A SIP session typically establishes two RTP/RTCP media streams—one from the callee to the caller and another from the caller to the callee. These media sessions are UDP based and will use dynamic ports. The dynamic ports used for the media stream are specified in the SDP section of the SIP payload message. The MIDCOM agent will parse the SDP section and use the MIDCOM protocol to (a) open pinholes (i.e., permit RTP/RTCP session tuples) in a middlebox implementing firewall service, or (b) create PORT bindings and appropriately modify the SDP content to permit the RTP/RTCP streams through a middlebox implementing NAT service. The MIDCOM protocol should be sufficiently rich and expressive to support the operations described under the timelines. The examples do not show the timers maintained by the agent to keep the middlebox policy rule(s) from timing out.

MIDCOM agent Registration and connectivity between the MIDCOM agent and the middlebox are not shown in the interest of restricting the focus of the MIDCOM transactions to enabling the middlebox to let the media stream through. MIDCOM PDP is also not shown in the diagram below or on the timelines for the same reason.

The following subsections illustrate a typical timeline sequence of operations that transpire with the various elements involved in a SIP telephony application path. Each subsection is devoted to a specific instantiation of a middlebox service: NAPT, firewall, and a combination of both NAPT and firewall are considered.

5.4.7.1 Timeline Flow—Middlebox Implementing Firewall Service

Figure 5.17 assumes a middlebox implementing a firewall service. One further assumes that the middlebox is preconfigured to permit SIP calls (destination TCP or UDP port number set to 5060) into the private

phone. The following timeline illustrates the operations performed by the MIDCOM agent, to permit RTP/RTCP media stream through the middlebox.

The INVITE from the caller (external) is assumed to include the SDP payload. You will note that the MIDCOM agent requests the middlebox to permit the Private-to-external RTP/RTCP flows before the INVITE is relayed to the callee. This is because, in SIP, the calling party must be ready to receive the media when it sends the INVITE with a session description. If the called party (private phone) assumes this and sends "early media" before sending the 200 OK response, the firewall will have blocked these packets without this initial MIDCOM signaling from the agent.

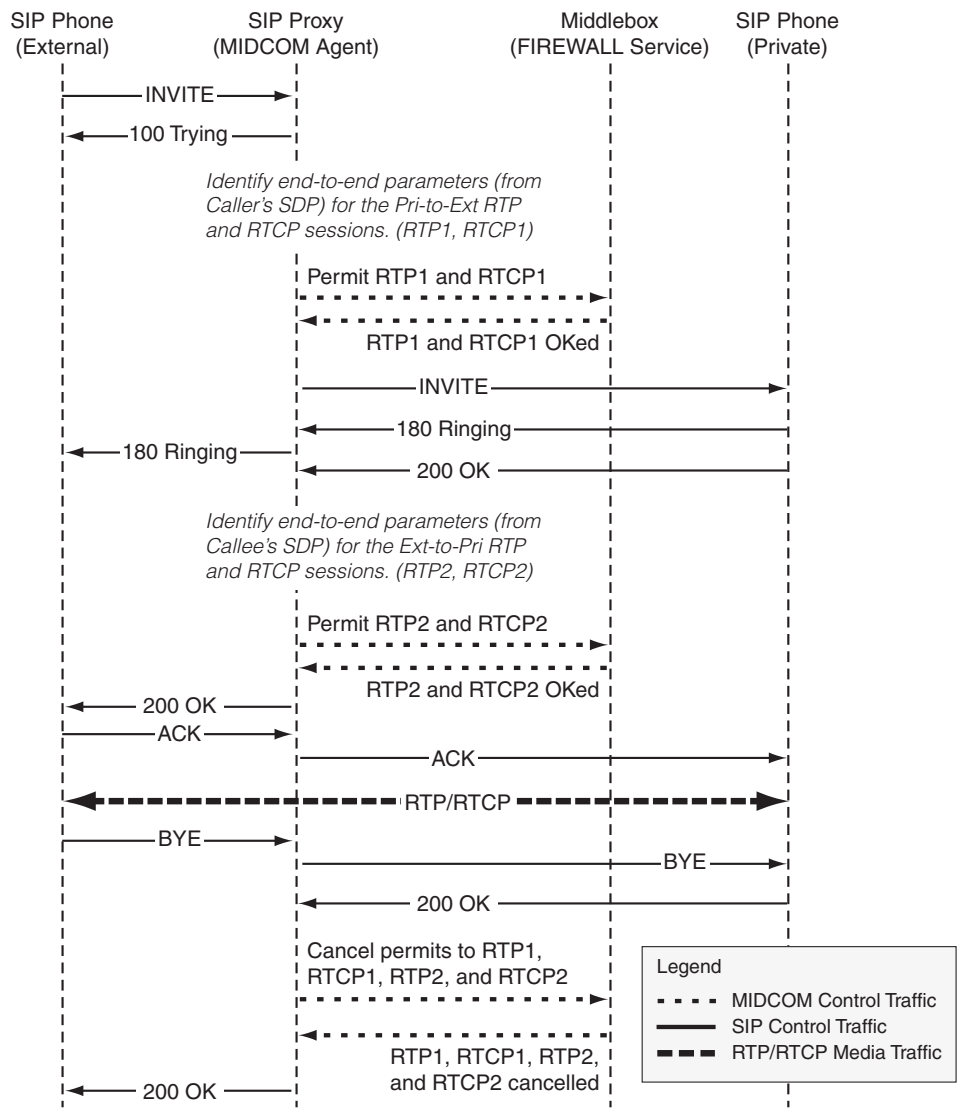


Figure 5.17: Timeline flow—Middlebox implementing firewall service.

5.4.7.2 Timeline Tow—Middlebox Implementing NAPT Service

Figure 5.18 assumes a middlebox implementing NAPT service. One makes the assumption that the middlebox is preconfigured to redirect SIP calls to the specific private SIP phone application. i.e., the INVITE from the external caller is not made to the private IP address, but to the NAPT external address. Let us say, the external phone's IP address is Ea, NAPT middlebox external Address is Ma, and the internal SIP phone's private address is Pa. SIP calls to the private SIP phone will arrive as TCP/UDP sessions, with the destination address and port set to Ma and 5060 respectively. The middlebox will redirect these datagrams to the internal SIP phone. The following timeline will illustrate the operations necessary to be performed by the MIDCOM agent to permit the RTP/RTCP media stream through the middlebox.

As with the previous example (Section 5.4.7.1), the INVITE from the caller (external) is assumed to include the SDP payload. You will note that the MIDCOM agent requests the middlebox to create NAT session descriptors for the private-to-external RTP/RTCP flows before the INVITE is relayed to the private SIP phone (for the same reasons as described in Section 5.4.7.1). If the called party (private phone) sends "early media" before sending the 200 OK response, the NAPT middlebox will have blocked these packets without the initial MIDCOM signaling from the agent. Also, note that after the 200 OK is received by the proxy from the private phone, the agent requests the middlebox to allocate NAT session descriptors for the external-to-private RTP2 and RTCP2 flows, such that the ports assigned on the Ma for RTP2 and RTCP2 are contiguous. The RTCP stream does not happen with a noncontiguous port. Lastly, you will note that even though each media stream (RTP1, RTCP1, RTP2 and RTCP2) is independent, they are all tied to the single SIP control session, while their NAT session descriptors were being created. Finally, when the agent issues a terminate session bundle command for the SIP session, the middlebox is assumed to delete all associated media stream sessions automatically.

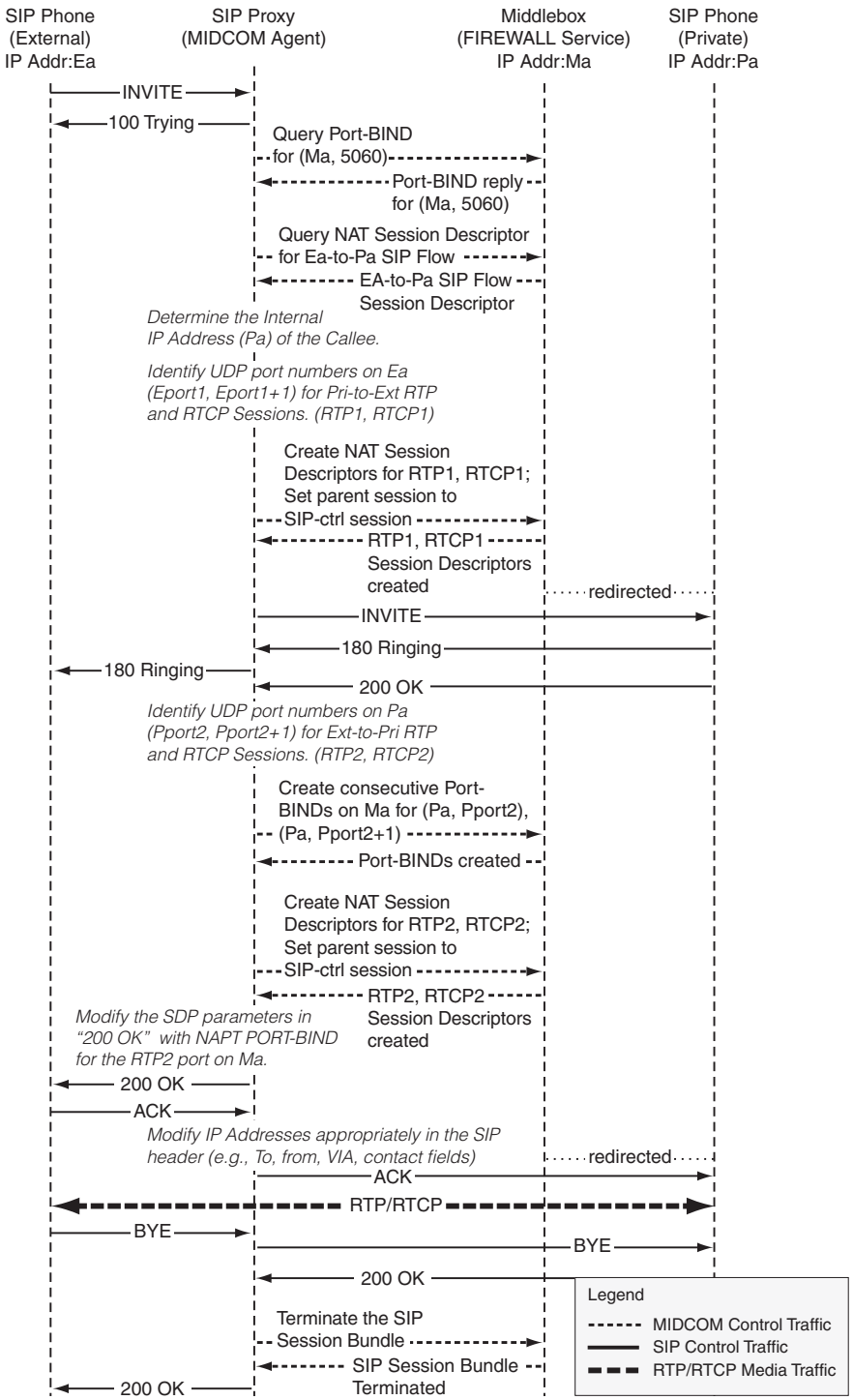


Figure 5.18: Timeline flow—Middlebox implementing NAPT service.

5.4.7.3 Timeline flow—Middlebox implementing NAPT and firewall.

Figure 5.19 assumes a middlebox implementing a combination of a firewall and a stateful NAPT service. One makes the assumption that the NAPT function is configured to translate the IP and TCP headers of the initial SIP session into the private SIP phone, and the firewall function is configured to permit the initial SIP session.

In the following timeline, it may be noted that the firewall description is based on packet fields on the wire (for example, as seen on the external interface of the middlebox). In order to ensure correct behavior of the individual services, you will notice that NAT specific MIDCOM operations precede firewall specific operations on the MIDCOM agent. This is noticeable in the timeline below when the MIDCOM agent processes the "200 OK" from the private SIP phone. The MIDCOM agent initially requests the NAT service on the middlebox to set up port-BIND and session-descriptors for the media stream in both directions. Subsequent to that, the MIDCOM agent determines the session parameters (i.e., the dynamic UDP ports) for the media stream, as viewed by the external interface, and requests the firewall service on the middlebox to permit those sessions through.

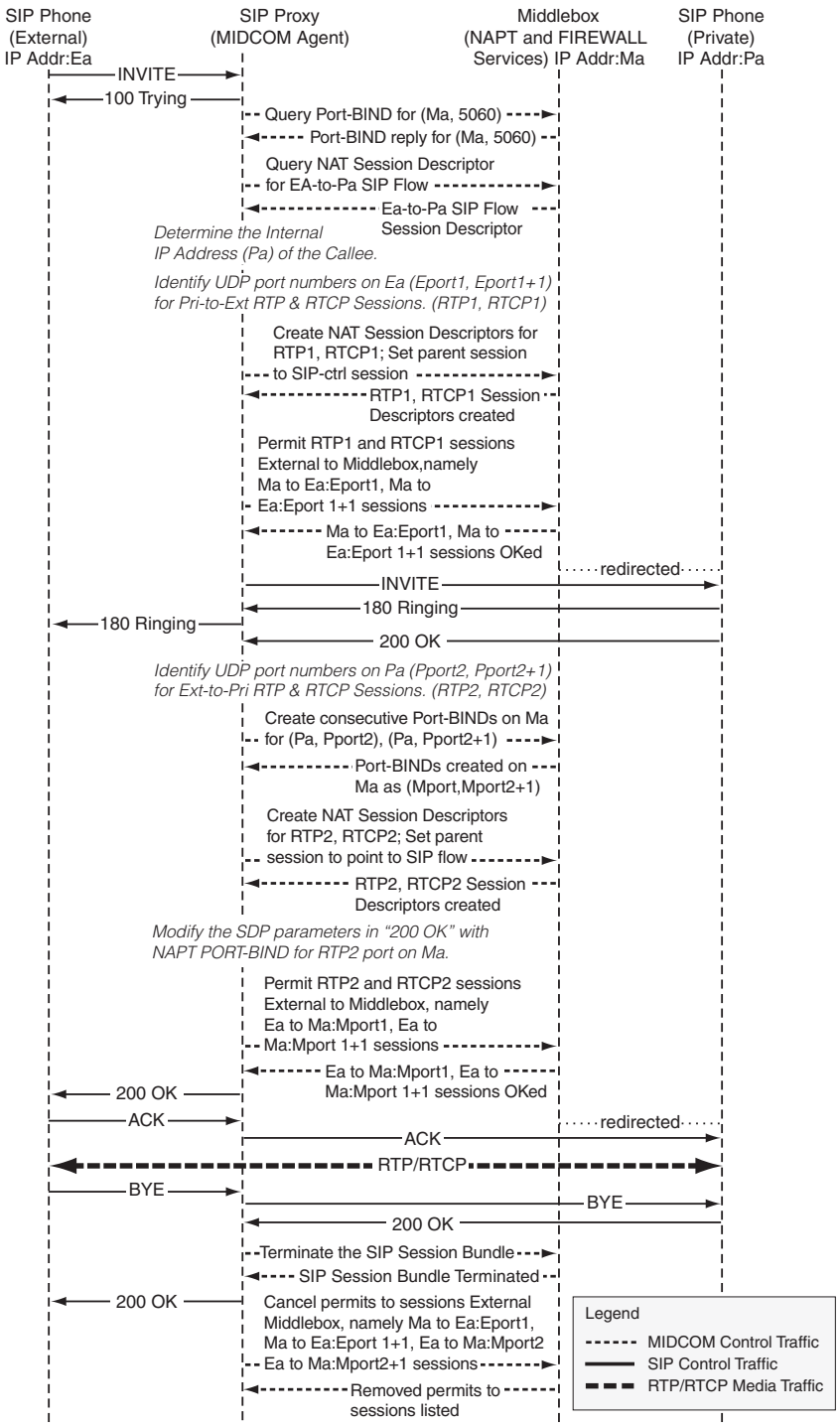


Figure 5.19: Timeline flow—Middlebox implementing NAPT and firewall.

5.4.8 Operational Considerations

5.4.8.1 Multiple MIDCOM sessions between agents and middlebox

A middlebox cannot be assumed to be a simple device implementing just one middlebox function and no more than a couple of interfaces. Middleboxes often combine multiple intermediate functions into the same device and have the ability to provision individual interfaces of the same device with different sets of functions and varied provisioning for the same function across the interfaces.

As such, a MIDCOM agent ought to be able to have a single MIDCOM session with a middlebox and use the MIDCOM interface on the middlebox to interface with different services on the same middlebox.

5.4.8.2 Asynchronous Notification to MIDCOM Agents

Asynchronous notification by the middlebox to a MIDCOM agent can be useful for events such as Session creation, Session termination, MIDCOM protocol failure, middlebox function failure or any other significant event. Independently, ICMP error codes can also be useful to notify transport layer failures to the agents.

In addition, periodic notification of various forms of data, such as statistics update, would also be a useful function that would be beneficial to certain types of agents.

5.4.8.3 Timers on Middlebox Considered Useful

When supporting the MIDCOM protocol, the middlebox is required to allocate dynamic resources, as specified in policy rule(s), upon request from agents. Explicit release of dynamically allocated resources happens when the application session is ended or when a MIDCOM agent requests the middlebox to release the resource.

However, the middlebox should be able to recover the dynamically allocated resources, even as the agent that was responsible for the allocation is not alive. Associating a lifetime for these dynamic resources and using a timer to track the lifetime can be a good way to accomplish this.

5.4.8.4 Middleboxes Supporting Multiple Services

A middlebox could be implementing a variety of services (e.g. NAT and firewall) in the same box. Some of these services might have interdependency on shared resources and sequence of operation. Others may be independent of each other. Generally speaking, the sequence in which these function operations may be performed on datagrams is not within the scope of this RFC.

In the case of a middlebox implementing NAT and firewall services, it is safe to state that the NAT operation on an interface will precede a firewall on the egress and will follow a firewall on the ingress. Further, firewall access control lists used by a firewall are assumed to be based on session parameters, as seen on the interface supporting firewall service.

5.4.8.5 Signaling and Data Traffic

The class of applications the MIDCOM architecture addresses focus around applications that have a combination of one or more signaling and data traffic sessions. The signaling may be done out-of-band, using a dedicated stand-alone session or may be done in-band, within a data session. Alternately, signaling may also be done as a combination of both stand-alone and in-band sessions.

SIP is an example of an application based on distinct signaling and data sessions. A SIP signaling session is used for call setup between a caller and a callee. A MIDCOM agent may be required to examine/modify SIP payload content to administer the middlebox so as to let the media streams (RTP/RTCP based) through. A MIDCOM agent is not required to intervene in the data traffic.

Signaling and context-specific Header information is sent in-band, within the same data stream for applications such as HTTP embedded applications, Sun-RPC (embedding a variety of NFS apps), Oracle transactions (embedding Oracle SQL+, MS ODBC, Peoplesoft) etc.

H.323 is an example of an application that sends signaling in both dedicated stand-alone sessions, as well as in conjunction with data. H.225.0 call signaling traffic traverses middleboxes by virtue of static policy, no MIDCOM control needed. H.225.0 call signaling also negotiates ports for an H.245 TCP stream. A MIDCOM agent is required to examine/modify the contents of the H.245 so that H.245 can traverse it.

H.245 traverses the middlebox and also carries Open Logical Channel information for media data. So, the MIDCOM agent is once again required to examine/modify the payload content needs to let the media traffic flow.

The MIDCOM architecture takes into consideration, supporting applications with independent signaling and data sessions as well as applications that have signaling and data communicated over the same session.

In the cases where signaling is done on a single stand-alone session, it is desirable to have a MIDCOM agent interpret the signaling stream and program the middlebox (that transits the data stream) so as to let the data traffic through uninterrupted.

5.4.9 Applicability Statement

Middleboxes may be stationed in a number of topologies. However, the signaling framework outlined in this RFC may be limited to only those middleboxes that are located in a DMZ (Demilitarized Zone) at the edge of a private domain, connecting to the Internet. Specifically, the assumption is that you have a single middlebox (running NAT or firewall) along the application route. Discovery of a middlebox along an application route is outside the scope of this RFC. It is conceivable to have middleboxes located between departments within the same domain or inside the service provider's domain and so forth. However, care must be taken to review each individual scenario and determine the applicability on a case-by-case basis.

The applicability may also be illustrated as follows. Real-time and streaming applications, such as Voice-Over-IP, and peer-to-peer applications, such as Napster and Netmeeting, require administering firewalls and NAT middleboxes to let their media streams reach hosts inside a private domain. The requirements are in the form of establishing a "pin-hole" to permit a TCP/UDP session (the port parameters of which are dynamically determined) through a firewall or retain an address/port bind in the NAT device to permit sessions to a port. These requirements are met by current generation middleboxes using adhoc methods, such as embedding application intelligence within a middlebox to identify the dynamic session parameters and administering the middlebox internally as appropriate. The objective of the MIDCOM architecture is to create a unified, standard way to exercise this functionality, currently existing in an ad-hoc fashion, in some of the middleboxes.

By adopting MIDCOM architecture, middleboxes will be able to support newer applications they have not been able to support thus far. MIDCOM architecture does not, and must not in anyway, change the fundamental characteristic of the services supported on the middlebox.

Typically, organizations shield a majority of their corporate resources (such as end-hosts) from visibility to the external network by the use of a DMZ at the domain edge. Only a portion of these hosts are allowed to be accessed by the external world. The remaining hosts and their names are unique to the private domain. Hosts visible to the external world and the authoritative name server that maps their names to network addresses are often configured within a DMZ in front of a firewall. Hosts and middleboxes within DMZ are referred to as DMZ nodes.

Figure 5.20 illustrates the configuration of a private domain with a DMZ at its edge. Actual configurations may vary. Internal hosts are accessed only by users inside the domain. Middleboxes, located in the DMZ may be accessed by agents inside or outside the domain.

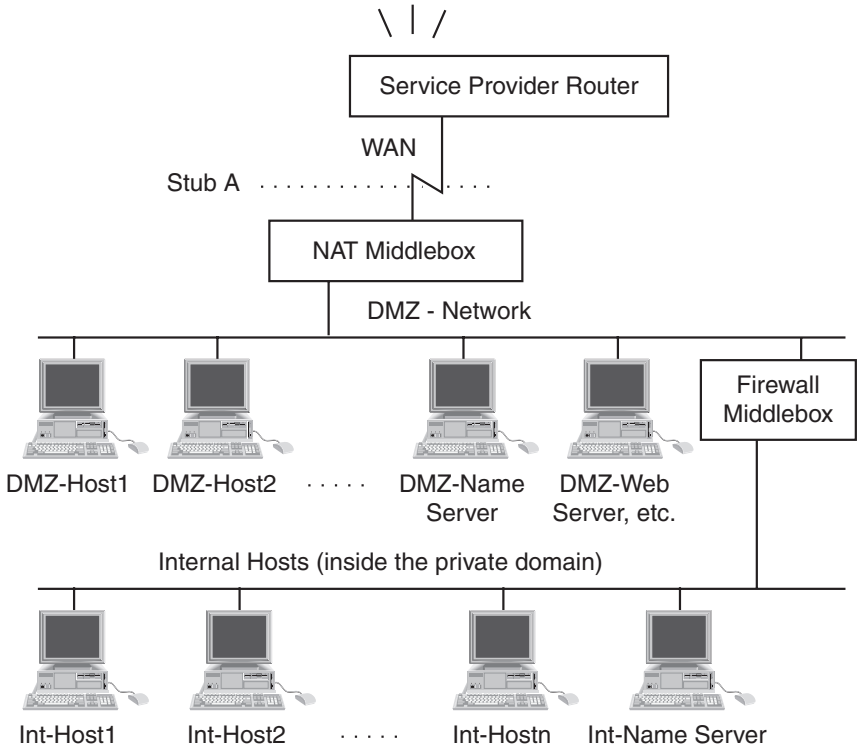


Figure 5.20: DMZ network configuration of a private domain.

5.5 Pragmatic Approaches using SIP Border Gateways

The previous sections of this chapter discussed some of the issues involved in supporting VoIP on a large scale due to addressing problems, and some current approaches (e.g., STUN, MIDCOM) to address these concerns. As an outgrowth of these limitations, Session Border Controllers (SBC) have emerged of late to assist service providers support VoIP and real-time interactive IP-based video/multimedia sessions in five areas: security, service reach maximization (end-to-end feasibility), SLA assurance, revenue and profit protection, and regulatory and law enforcement [OUE200501]. The interest in this context is on the first two items in this list.

A session border controller is a piece of network equipment or a collection of functions that control real-time session traffic at the signaling, call-control, and packet layers as they cross a notional packet-to-packet network border between networks or between network segments. SBCs are critical to the deployment of VoIP networks, because they address the inability of real-time session traffic to cross NAT device or firewall boundaries. Signaling protocols such as H.323, MGCP, and SIP transfer information including media session endpoint IP addresses and UDP port numbers in different layers above OSI Layer 4 (IETF TCP/UDP). This information cannot be seen by a normal firewall or NAT device, so the subsequent sessions set up are not

recognized, do not pass through firewalls, and have incompatible IP addresses across NAT boundaries. SBCs allow NAT and firewall traversal, normally by incorporating those elements with signaling controllers for the required signaling protocols [LIG200502].

SBCs are devices used in VoIP to deal with a number of interworking issues. The controller refreshes NAT bindings for SIP registrations. The SBC compresses SIP packets to less than 1492 bytes (UDP fragmentation). It hides routing information to the outside world. The SBC ensures that an end-to-end media path is established. Also it helps the teardown process after the call is completed.

SBCs address the requirements at the boundary where different service provider networks interconnect or "peer." In general, session border controllers integrate signaling and media control, encompassing the following three functional subelements: (a) Interconnect Border Control Function, (b) Interworking Function, and (c) Interconnect Border Gateway Function.

One example of SBC usage is in the IP Multimedia Subsystem (IMS). IMS is an architecture defined by the Third Generation Partnership Project (3GPP) for the delivery of real-time voice, video and multimedia services using SIP over packet-switched networks with a focus on mobile wireless access networks. This architecture has been extended by ETSI to more completely satisfy the service delivery requirements in fixed-wireline access networks. Some of these additional requirements include [OUE200501]:

- Premise-based NAT traversal;
- Overlapping private address space and enterprise MPLS VPN bridging;
- IPv4 to IPv6 interworking for signaling and media;
- SIP interworking for H.323 IP PBXs and gatekeeper trunking/termination networks;
- Media-based DTMF (RFC 2833) to signaling-based DTMF translations.

Within the extended IMS architecture, two different types of session border controllers that integrate signaling and media control play very important roles: the Access SBC and the Interconnect SBC. The integration of signaling and media control provides several architectural benefits:

Security: SBC prevents DoS attacks on core (IMS) elements by dynamically discovering and blocking malicious signaling and media attacks or nonmalicious overloads (e.g., endpoint re-registering very frequently). Advanced SBCs using hardware-based features, can protect themselves against attack without loss of service.

Scalability: SBC provides distributed edge processing function for signaling and media offloading core (IMS) elements for connection and encryption management (e.g., TCP, TLS, IPSec), NAT traversal processing and other processor-intensive tasks.

Manageability: SBC incorporates multiple (IMS) functions resulting in fewer network elements, fewer networking protocols, and more robust fault and performance management (e.g., media QoS monitoring incorporated with session layer accounting).

The Interconnect Session Border Controller addresses the requirements at the boundary where different service provider networks interconnect or "peer." The controller integrates three functional elements from the ETSI TISPAN architecture [OUE200501].

1. Interconnect Border Control Function (IBCF): Provides overall control of the boundary between different service provider networks. It provides security for the IMS core in terms of signaling information by implementing a Topology Hiding Internetwork Gateway (THIG) subfunction. This subfunction performs signaling-based topology hiding, IPv4-IPv6 interworking, and session screening based upon source and destination signaling addresses. The IBCF also invokes the interworking function (described below) when connecting non-SIP or non-IPv6 networks, and performs admission control and bandwidth allocation using local policies or via interface to ETSI TISPAN Resource and

Admission Control Subsystem (RACS). Lastly, the IBCF interacts with I-BGF (described below) for control of the boundary at the transport layers including pinhole firewall, NAPT, and numerous other features.

2. Interworking Function (IWF): Provides signaling protocol interworking between the SIP-based IMS network and other service provider networks using H.323 or different SIP profiles.
3. Interconnect Border Gateway Function (I-BGF): Controls the transport boundary at layers 3 and 4 between service provider networks. This function acts as a pinhole firewall and NAT device protecting the service provider's IMS core. It controls access by packet filtering on IP address/port and opening/closing gates (pinholes) into the network. It uses NAPT to hide the IP addresses/ports of the service elements in the IMS core. QoS packet marking, bandwidth and signaling rate policing, usage metering and QoS measurements for the media flows are additional features supported by the I-BGF.

The Access Session Border Controller satisfies the requirements at the border where subscribers access the IMS core. It integrates two functional elements from the IMS and ETSI TISPAN architectures [OUE200501].

1. Proxy-Call Session Control Function (P-CSCF): Is the SIP signaling contact point, the outbound/inbound "proxy," for subscribers within IMS as defined by 3GPP. However, the term "proxy" is deceiving since to fulfill its complete set of responsibilities it must be able to proactively initiate SIP requests. This requires implementation as a SIP Back-to-Back User Agent (SIP B2BUA), not a simple SIP proxy. The P-CSCF is responsible for forwarding SIP registration messages from the subscriber's endpoint, the User Element (UE), in a visited network to the Interrogating-CSCF (I-CSCF) and subsequent call set-up requests and responses to the Serving-CSCF (S-CSCF). The P-CSCF maintains the mapping between logical subscriber SIP URI address and physical UE IP address and a security association, for both authentication and confidentiality, with the UE using TLS for example. It supports emergency call (E911) local routing within the visited network, accounting, session timers, and admission control. Admission control requires an interface to an external IMS Policy Decision Function (PDF)/ESTI TISPAN RACS. The P-CSCF interacts with an Access Border Gateway Function (A-BGF) for control of the boundary at the transport layers including pinhole firewall, NAPT and numerous other features. In addition, for wireline networks, ETSI's RACS is responsible for network-based NAT traversal.
2. Access Border Gateway Function (A-BGF): Controls the transport boundary at layers 3 and 4 between subscribers and the service provider's network. It performs all of the functions and features of the I-BGF. In addition, in wireline networks, it provides network-based NAT traversal for the media flows.

Session border controller product typically integrate signaling and media control in a single platform. Alternatively, session border control may be implemented using a distributed architecture using separate physical signaling and media control products for the three functional elements described above.

Figure 5.21 depicts, for illustrative purposes, an example of a commercial Session Border Controller; in this case the functionality runs parallel to the Gatekeeper [SYS200501]. In this example the Gateway/ Session Border Controller can operate in three ways:

1. Direct/static Mode to allow call resolution without Registration, Admission, or Status (RAS) message control (RAS is a protocol used in the H.323 protocol suite for discovering and interacting with a Gatekeeper). This mode will allow number translation and dynamic call control given that the participating gateways support the canMapAlias attribute.
2. Routed Mode to allow direct control of RAS messages with very low level of bandwidth utilization. This mode allows number translation and dynamic call control for gateways that do not support the canMapAlias attribute.

3. Proxy Mode to allow full RAS and RTP data transfer for gateways behind NAT or gateways that want to keep their identity. This bandwidth-intensive mode fully controls the RAS and Q.932 data streams and supports number translation and dynamic call control.

Figure 5.22 depicts, for illustrative purposes, a more complete enterprise SIP VoIP application to illustrate how the various elements interplay (ETH Zurich's PolyPhone environment) [LOR200501].

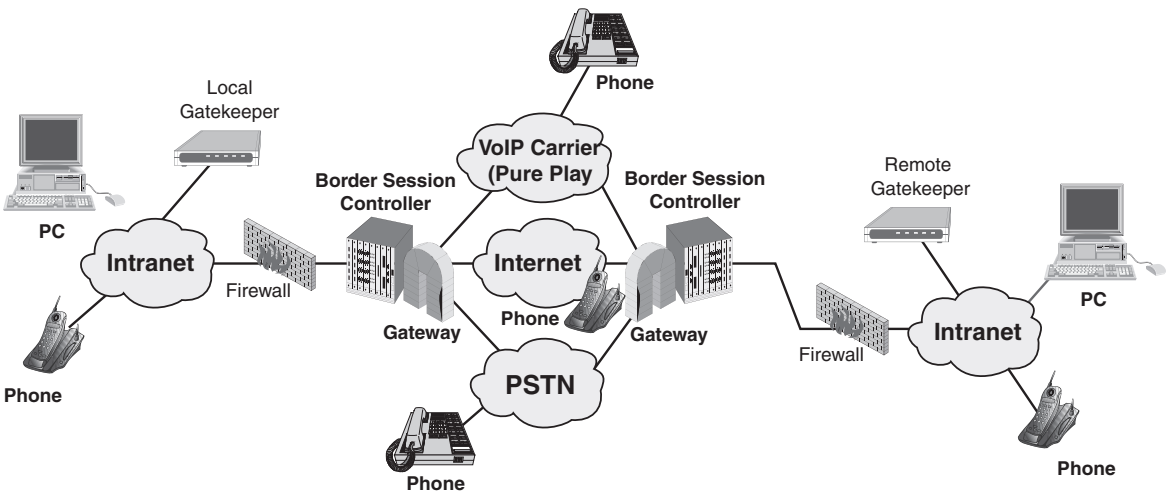
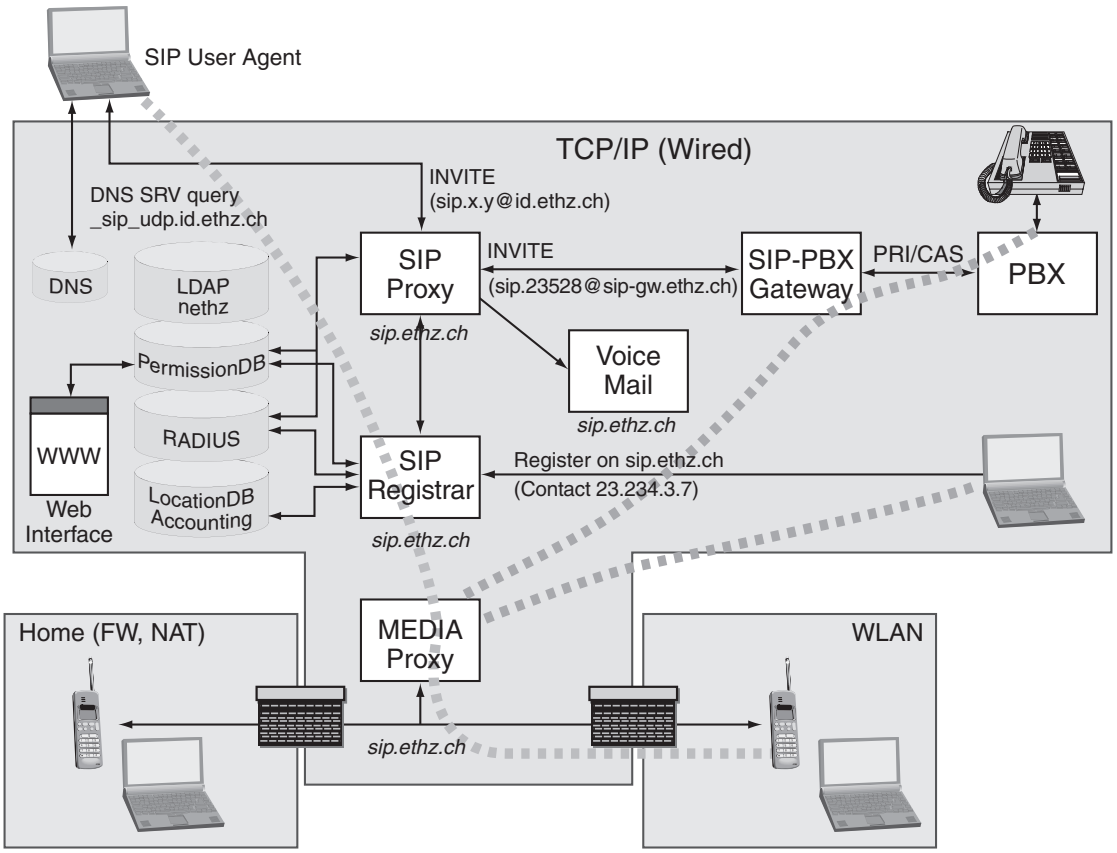


Figure 5.21: Gateway/session border controller example.



SIP Sessions

From the SDP specification (RFC 2327): "A multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A multimedia conference is an example of a multimedia session." (A session as defined for SDP can comprise one or more RTP sessions.) As defined, a callee can be invited several times, by different calls, to the same session. If SDP is used, a session is defined by the concatenation of the SDP user name, session id, network type, address type, and address elements in the origin field. SIP supports stateless and stateful connections. A stateless proxy establishes the connection and then "gets out of the way." A stateful proxy stores all signaling events for the duration of the call (some SIP proxy servers deposit cookies in the IP phone/terminal as a method of providing state information).

SIP Proxy

An intermediate device that receives SIP requests from a client and then initiates requests on the client's behalf. The SIP proxy server provides similar functionality to a gatekeeper in an H.323 environment or a softswitch in an MGCP/MEGACO environment.

SIP Registrar

The default SER registrar where all active SIP clients are registered.

PeerPoint

A third party Border Gateway Controller is integrated to support NAT/Firewalled user agents and hide internal topology of SIP environment (Proxies, Gateways, Servers).

(Figure 5.22 continued on next page)

LDAP nethz

Stores usernames, passwords, E-Mail (primary and aliases) and internal phone numbers.

Lightweight Directory Access Protocol

An emerging software protocol for enabling anyone to locate organizations, individuals, and other resources such as files and devices in a network, whether on the Internet or on a corporate intranet. LDAP is a "lightweight" (smaller amount of code) version of DAP (Directory Access Protocol), which is part of X.500, a standard for directory services in a network.

PermissionDB

Stores phonenumbers, settings and permissions for users. In the near future, the PermissionDB will be integrated into the LDAP infrastructure.

Web-Interface

Instead of serweb, this implementation uses a custom website with interface to the SER proxy. In the future, the web interface may be integrated in the web interface of the existing LDAP services (n.ethz.ch). The web interface also gives useful information about this project and monitors the status of the environment components.

Radius

The following operations are authorized using the existing RADIUS server infrastructure:

- *Registration of SIP users (REGISTER)*
- *Establishing calls using an n.ethz.ch digest header (INVITE)*

Location DB, Accounting

The default SER Location DB. Accounting is only used for statistical purposes.

DNS

Domain Name Server of ethz.ch (any former e-mail could be resolved to a SIP account or an internal PSTN phone number.)

Gateways

Existing PSTN infrastructure has been integrated in the environment. Authorized SIP users can reach every internal and external phone.

Voice Mail

A voice mail-box system is available to the user.

TCP/IP (Wired)

The wired TCP/IP network of organization. Directly addressable IP numbers are used (no firewalls or number translation).

HOME (FW, NAT)

Infrastructure used by employees when at home or en route.

WLAN

The Wireless LAN infrastructure of the organization. For public users the WLAN allows only access to selected IP addresses inside the organization (e.g., the home page www.ethz.ch). Other addresses can only be reached after VPN validation. Since this is not currently possible with WLAN SIP phones, exceptional access to the SIP Server/ MEDIA Proxy has been granted. Phone registered on the SIP server will be able to establish connections to any other SIP phone on the Internet.

Figure 5.22: Example of an institutional SIP environment.