Contributed article for *IETF Journal*

Submitted by Cisco

Final Draft – January 4, 2007

Interactive Connectivity Establishment: NAT Traversal for the Session Initiation Protocol

Jonathan Rosenberg

Cisco

The Session Initiation Protocol has seen widespread usage on the Internet for voice over IP. It sets up, manages, and tears down billions of minutes of calls each year, a number that continues to rise. However, deployment of SIP has not been without its challenges. Perhaps most significant among those challenges is traversal through NAT and through firewall devices, which have become commonplace on the Internet and within private IP networks. To date, this problem has been solved through proprietary and expensive techniques that have had a negative impact on security and interoperability. The IETF has responded by developing a new specification, called Interactive Connectivity Establishment. ICE is a form of peer-to-peer NAT traversal that works as an extension to SIP. In this article, we’ll review the NAT traversal problem, touch on alternative solutions, and briefly overview how ICE works.

# Introduction

Work on the Session Initiation Protocol first began in the IETF in the mid-1990s. It was initially targeted at supporting invitations to large-scale multicast conferences on the mbone (Multicast Backbone), but quickly found its primary application for the signaling of point-to-point voice over IP. It was published as RFC 2543 [[[1]](#endnote-1)] in 1999 and revised in June 2002 as RFC 3261 [[[2]](#endnote-2)], one of the longest RFCs ever to be produced by the IETF, but also one of the most successful.

SIP has seen widespread usage and deployment in both the public Internet and private IP networks. Billions of minutes of VoIP calls each year are managed by SIP. It is used in small enterprise PBX systems, consumer VoIP services such as Vonage and SunRocket, telephony backbone networks, and enterprise collaboration services. There are hundreds of independent implementations, dozens of open source codebases, and even a magazine dedicated to the technology. By all metrics, SIP has been a success.

However, its success has not come without difficulties. Perhaps most significant among them has been the proliferation of Network Address Translation and firewall devices. SIP was designed before those devices became commonplace, and consequently it does not operate successfully through NAT as originally specified. As NAT and firewalls proliferated, the market responded by adding several proprietary components and techniques to VoIP networks. These include application layer gateways embedded within NAT and firewall devices, and externalized ALGs known as Session Border Controllers (SBCs). Though they provided a path for the growth of VoIP on the Internet, they brought a host of problems with them, and a standardized solution was required.

The IETF responded to this need by the creation of a new specification that augments SIP with robust and low-cost NAT traversal. This specification, Interactive Connectivity Establishment [[[3]](#endnote-3)], was produced by the mmusic working group in the newly formed Real-time Applications and Infrastructure (RAI) area. ICE is in its final stages of specification and should be complete in early 2007.

# What Is the Problem, Anyway?

NAT operates by rewriting the IP addresses in the IP headers as packets pass from one interface to the other. When a packet is sent from the “inside” of the NAT toward the “outside,” the source IP address and port are rewritten from the address space on the inside (usually private IP address space) into the address space on the outside. Similarly, packets from the outside to the inside have the destination address and port rewritten from the address space on the outside to the one on the inside. Typically, NAT will rewrite the addresses by maintaining a table of bindings that map each internal IP address and port to an external IP address and port. A binding is dynamically created when the first packet from a particular internal IP address and port arrives at the NAT. This process is shown pictorially in Figure 1.



Figure : NAT Operation

This kind of translation works just fine for many protocols. HTTP, POP, and SMTP, for example, work fine through such devices. Things break down for protocols that carry IP addresses and ports in the payload of the packet itself – an area not touched by the NAT. Protocols such as SIP, whose job is to establish multimedia sessions between hosts on the Internet, fundamentally require IP addresses and ports in their payload. For these protocols, the NAT completely breaks their operation.

A simple example can help illustrate. Consider Alice, who wishes to place a call to Bob. This is done in SIP by sending a SIP INVITE message. The INVITE message contains Alice’s IP address and port where she expects to receive media packets. When Bob receives the message and answers the call, he sends his media packets to that IP address and port. This allows the latency-sensitive multimedia traffic to make its way directly from Bob to Alice. If Alice is behind a NAT, her INVITE message will contain a private address. As the SIP message passes through the NAT, the NAT will rewrite the source IP address of the SIP packet but will not touch its contents. When the message arrives for Bob, the address indicated within its payload will, in most cases, not be reachable by him. Consequently, media traffic will not flow.

# The Market Responds

The market quickly responded to this problem with several solutions. The two most common are the Application Layer Gateway and the Session Border Controller.

An ALG is an application layer component whose functionality is resident in the NAT itself. The NAT inspects SIP packets as they transit the NAT. Instead of just ignoring the content of the packets, as a normal NAT does, the ALG translates the IP addresses within the body of the SIP message, matching them with the translated source IP address. In some regards, this is the obvious solution to the problem. The NAT is the element that broke SIP, so it should fix it. It is completely transparent to the SIP clients and servers.

SIP ALGs have found usage primarily in enterprise environments. However, they are far from an ideal solution. Because the ALG needs to inspect and modify the SIP packets, many of SIP’s security mechanisms, such as SIP over TLS (SIPS) and SIP Identity [[[4]](#endnote-4)] break when used with an ALG. Indeed, these security mechanisms need to be disabled in order for the ALG to operate. The reason for this is simple:The ALG operates like “a man in the middle,” and its modification of SIP packets cannot be differentiated from a man-in-the-middle attack.

ALGs also make it extremely difficult to introduce extensions to SIP. The ALG needs to be SIP-aware, and must be programmed with all SIP functions that might affect NAT traversal. Since the ALG is part of the router itself, this results in SIP functionality being built into the network. Adding an extension to SIP that interacts with NAT traversal requires support from every single NAT that might possibly see SIP messages. In essence, the Internet itself must be upgraded as well. This is contrary to the very notion of IP, which separates the network from the applications that run on top of it.

Finally, ALGs have been a proven source of problems in implementation and interoperability. They frequently implement only subsets of the required functionality, breaking more complex cases. When problems do occur, diagnosing them is nearly impossible, since the ALG is invisible to the rest of the SIP network.

Instead of relying on ALGs, most SIP networks have made use of a close cousin of the ALG, the Session Border Controller. The SBC does many of the same things an ALG does: It receives SIP packets and rewrites those portions of the message that contain IP address information. However, whereas an ALG is transparent and modifies packets as they pass through the NAT, the SBC looks to the outside world like a SIP proxy and is the direct target for SIP requests. Because it is not a transparent intermediary, it does not break SIP security mechanisms meant to operate between SIP elements, such as SIP over TLS. However, since the SBC does still modify SIP packets, it does break other SIP security techniques, such as SIP Identity.

Unlike ALGs, which require every NAT device in the network to be upgraded, a VoIP provider can simply add an SBC to its network without changing the SIP clients, SIP servers, or NAT devices in the rest of the network. This makes SBCs relatively easy to deploy, which is the primary reason for their success in the market. However, SBCs share many of the problems of ALGs, including breaking SIP security mechanisms and making it difficult to introduce SIP extensions. The latter deficiency is particularly problematic, since one of the key strengths of SIP’s design, and one of the reasons for its success in the market, has been this flexibility and adaptability. SBCs make SIP networks much more rigid.

# The IETF to the Rescue

The IETF was aware of the difficulties in working SIP through NAT since the very beginning and has made numerous attempts to solve them.

The first attempt was called midcom (Middlebox Communications) [[[5]](#endnote-5)]. Midcom allows a SIP proxy server to communicate with NAT or a firewall to ask it for explicit translation and pinhole services. However, the proxy is still required to modify the SIP message, resulting in many of the same problems that SBCs had. Worse still, midcom works only in a rigid set of topologies where the proxy server knows the location of the NATs and firewalls, and has a strong trust relationship with them. This limited its applicability, and consequently midcom has seen limited usage.

The next specification that was produced was Simple Traversal of UDP through NAT [[[6]](#endnote-6)]. With STUN, the SIP client generates a STUN request to a STUN server on the public Internet. This request causes the NAT to allocate a binding to the client. The STUN server sends a response to the client and, within its body, returns the source IP address and port of the request as seen by the STUN server. The client then uses this IP address and port in its SIP messages. STUN has the benefit of being extremely lightweight and scalable. It avoids all of the security pitfalls of SBCs and ALGs. However, it does not work through certain types of NAT, and it fails in topologies where both caller and called party happen to be behind the same NAT. This limits its applicability.

To broaden the applicability, a companion protocol, called Traversal Using Relay NAT [[[7]](#endnote-7)] was developed. As with STUN, a client sends a request to a TURN server prior to making a call. The TURN server returns an IP address and port to the client that it can use as the destination for media. The client includes this IP address and port in its signaling messages. However, the IP address and port provided by the TURN server are those of the TURN server itself, which acts as a relay, forwarding packets to and from the client. In essence, the TURN server is like a VPN server, but running at the UDP layer rather than IP.

Though TURN works in more cases than STUN does, TURN is expensive, since it requires the provider to relay media for every SIP call. This also increases voice latency. What was needed was a technology that somehow combined the benefits of STUN and TURN without their drawbacks.

# ICE Is Nice

ICE was first submitted as an individual draft in February 2003 and was adopted as a deliverable of the IETF mmusic working group in October 2003. Having gained increased interest over the years, ICE is finally near completion after two rewrites and several redesigns.

ICE provides NAT and firewall traversal capabilities for any type of session-oriented protocol, though it has been designed to work with SIP and its companion protocol, the Session Description Protocol (SDP). ICE makes use of STUN and TURN and provides a unifying framework around them. ICE is extremely robust, providing traversal under even the most complex topologies. It is also optimal, in that it will make use of intermediate relays (the TURN server) only when nothing else works. ICE also supports TCP media sessions, such as those used for shared whiteboards or application sharing.

Even though ICE has not yet reached RFC status, there are already several large-scale deployments supporting hundreds of thousands of users. There are implementations in several softphone clients.

The essential idea of ICE is relatively straightforward. Rather than pick just STUN or just TURN for a particular call, a client will obtain IP addresses and ports using both techniques. It includes both addresses, in addition to ports allocated from local interfaces, into the SIP call-setup messages. Each of these is called a *candidate*, and it represents a potential point of communications for the agent. When the SIP call-setup request arrives, the called party does the same thing, including numerous addresses in the SIP response. At that point, the agents begin a process of connectivity checks. These are STUN messages sent from one agent to the other, probing to find a particular pair of addresses that work. Once a pair is found, the probes cease, and media can begin to flow.

The detailed operation of ICE can be broken into seven steps: gathering, prioritizing, encoding, offering and answering, pairing, checking, and completing.

## Step 1: Gathering

Prior to making a call, the caller begins gathering IP addresses and ports that are each a potential candidate for communications. The first such candidate is gathered from interfaces on the host. If the host is multihomed, the agent gathers a candidate from each interface. Candidates from interfaces on the host (including virtual interfaces) are called *host candidates*. Next, the agent contacts a STUN server from each host interface. The result will be a set of *server-reflexive candidates*. These are IP addresses that route to the outermost NAT between the agent and the STUN server, which is typically on the public Internet. Finally, the agent obtains *relayed candidates* from TURN servers. These IP addresses and ports reside on the relay servers. As an optimization, the TURN protocol allows a client to learn its relayed and server-reflexive candidates at the same time.

## Step 2: Prioritizing

Once the agent has gathered its candidates, it assigns each of them a priority value. Priorities are from 0 to 231 – 1, with larger numbers denoting higher priority. The priorities are computed by means of a formula that combines preferences for types of candidates (where the types are host, relayed, and server reflexive) along with preferences for each host interface. Typically, the lowest priority is given to the relayed candidates, since sending media through a relay is expensive and increases voice latency. When a host is multihomed, it typically prefers one interface to another for communications. For example, a VPN interface might be preferred to an Ethernet interface, in order to keep intracompany voice communications on a private enterprise network.

## Step 3: Encoding

With its candidates gathered and prioritized, the agent constructs its SIP INVITE request to set up the call. The body of the SIP request contains an SDP message that conveys the information needed for transmitting the media content of the call. This includes the types of media codecs, their parameters, and the IP addresses and ports to be used. ICE extends SDP by adding several new SDP attributes. The most important of these is the candidate attribute. For each media stream signaled in the SDP, there is a candidate attribute for each candidate that the agent has gathered. This attribute contains the IP address and port for that candidate and also contains the priority and type of the candidate (host, server reflexive or relayed). The SDP also contains credential information that is used to secure the STUN messaging, which will commence later.

## Step 4: Offering and Answering

Once the calling agent has constructed its SIP INVITE request with the SDP payload, it sends the request to the called party. The SIP network delivers the request to the called party. Assuming the called party also supports ICE, the called party holds off on ringing the phone. However, it performs the same gathering, prioritizing and encoding that the caller performed. The called party then generates a provisional SIP response. Such a response indicates to the caller that the request is being processed but processing has not been completed. The provisional response contains an SDP with the candidates that the called party has gathered. The SIP network delivers the provisional response to the caller.

## Step 5: Checking

At this point, the caller and called party have exchanged SDP messages. Each is therefore aware of the set of candidates for each media stream that will make up the call (there may be more than one media stream; in videophones, for example, there would be an audio and a video stream). In this next step, ICE performs the bulk of its work.

Each agent pairs each of its candidates with a candidate from its peer. The result is a list of candidate pairs. If each agent provided three candidates for a media stream, there would be a total of nine candidate pairs for that media stream. Each agent computes a priority for the candidate pair by combining the priority of each candidate in the pair. The objective for ICE is to determine a candidate pair for which media will successfully flow in each direction. If many candidate pairs work, the objective is to select the highest-priority pair. Since the priority of each candidate (and consequently the pair) is largest for those with fewest intermediate relays (whether they be a NAT or a TURN server), the highest-priority pair will also be the one that provides the most direct path for media traffic.

To verify that a candidate pair works, ICE makes use of a STUN transaction from each agent towards the other called a *connectivity check*. The STUN transaction uses the IP addresses and ports in the candidate pair – the same IP addresses and ports that will be used for the transmission of media. Considering again our example of Alice and Bob, Alice sends a STUN request from one of her candidates to one of Bob’s candidates. If the STUN request is received, Bob generates a response that reaches Alice. If Alice gets Bob’s response, she knows that she can send a packet from her candidate to Bob’s candidate and that Bob will be able to receive it. Since she got a response, she also knows that packets from Bob are able to reach her. Thus, the STUN transaction serves to verify bidirectional reachability for a candidate pair. If Bob performs his own transaction, he can verify bidirectional reachability as well (note that the receipt of a request from Alice is not sufficient; it doesn’t indicate to Bob whether his response reached Alice).

Since the STUN transactions are sent on the same IP addresses and ports that will eventually be used for media traffic, there is a need to demultiplex the STUN and media using something besides the port. STUN has several fields built into its headers that allow it to be demultiplexed from arbitrary application traffic. In an ideal world, the UDP and TCP port could be used to multiplex. However, NAT has effectively made the port numbers part of the IP layer, since they now are significant in the routing of IP datagrams.

Since the number of candidate pairs grows by the square of the number of candidates, performing the checks for each pair in parallel is problematic. Instead, ICE performs the checks sequentially. The candidate pairs are ordered by priority, and every 20 milliseconds, each agent generates a STUN transaction for the next pair in the list. In addition, when an agent receives a STUN request on a candidate pair, it generates a STUN transaction in the reverse direction immediately. This is called a triggered check, and it improves the responsiveness of ICE.

## Step 6: Completing

Once a check is completed, the agent knows it has found a pair that will work for media traffic. Since the checks are done in priority order, the first one to be completed will usually be the highest-priority pair that works. One of the agents, typically the caller, will generate a final check toward the other agent, confirming that the pair is the one selected. This allows for each agent to unambiguously communicate which pair will ultimately be used for media.

Once this final transaction has been sent, the called agent can now ring the phone. All of the processing so far – the gathering and all of the connectivity checks – take place prior to the called party’s phone even ringing. This means that ICE has the side effect of increasing call-setup delays. This is ICE’s primary drawback. However, the increase in delay tends to be proportional to the complexity of the situation. For a basic voice call between two endpoints on the public Internet, with no intervening NAT, ICE adds only a single round-trip time to the call setup, which is inconsequential. By avoiding ringing the phone until the ICE checks are completed, ICE can guarantee that when the called party does answer, media will successfully flow in each direction. ICE therefore eliminates “ghost rings” – cases where the phone rings but the users hear nothing when they answer the phone. Ghost rings are a common problem in VoIP and are almost always caused by NAT and firewall traversal problems.

Once the phone rings, the called party answers. This generates a SIP 200 OK final response, confirming acceptance of the call. When callers get a 200 OK, they send a SIP ACK. If ICE negotiation results in the selection of a candidate pair that differs from the default IP address and port carried in the SDP (the default is used for communicating with non-ICE endpoints), the caller performs a SIP re-INVITE to update the default. This is done for the benefit of intermediate SIP elements that are not ICE-aware, but that need to know where media is being sent.

# Concluding Remarks

ICE is one of the most important extensions produced to date for SIP. Indeed, it is considered one of the few “core” SIP extensions – those expected to be used by every SIP client for every SIP call [[[8]](#endnote-8)]. Though designed for SIP, ICE is applicable to any session-oriented protocol. Indeed, ICE is currently in deployment with a non-SIP protocol. Work is also in progress to apply it to the Real Time Streaming Protocol (RTSP), used for streaming media control.

ICE’s importance goes beyond just robust NAT traversal. ICE adds significant security to SIP overall, eliminating a key DoS attack (the voice hammer), which can be launched using SIP networks as an amplifier.

With all of these benefits, its not hard to see why ICE is likely to be a cornerstone of all SIP networks in the near future.

1. M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, “SIP: Session Initiation Protocol.” IETF RFC 2543, March 1999 [↑](#endnote-ref-1)
2. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, “SIP: Session Initiation Protocol.” RFC 3261, June 2002 [↑](#endnote-ref-2)
3. J. Rosenberg, “Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols,” IETF Internet Draft draft-ietf-mmusic-ice-12, October 2006 [↑](#endnote-ref-3)
4. J. Peterson, C. Jennings, “Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP),” IETF RFC 4474, August 2006 [↑](#endnote-ref-4)
5. P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, A. Rayhan, “Middlebox Communication Architecture and Framwork,” IETF RFC 3303, August 2002 [↑](#endnote-ref-5)
6. J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, “ STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs),” IETF RFC 3489, March 2003 [↑](#endnote-ref-6)
7. J. Rosenberg, R. Mahy, C. Huitema, “Obtaining Relay Addresses from Simple Traversal Underneath NAT (STUN),” IETF Internet Draft draft-ietf-behave-turn-02, October 2006 [↑](#endnote-ref-7)
8. J. Rosenberg, “A Hitchhiker’s Guide to the Session Initiation Protocol (SIP),” IETF Internet Draft draft-ietf-sip-hitchhikers-guide-01, October 2006 [↑](#endnote-ref-8)