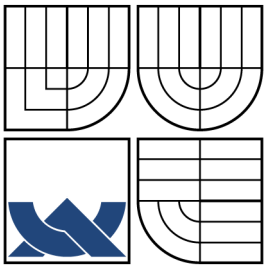
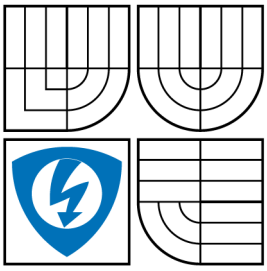


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

ARCHITEKTURA PRO GLOBÁLNÍ DISTRIBUOVANOU SIP SÍŤ S VYUŽITÍM IPv4 ANYCASTU

AN ARCHITECTURE FOR GLOBAL DISTRIBUTED SIP NETWORK USING IPv4 ANYCAST

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. LADISLAV ANDĚL
AUTHOR

VEDOUCÍ PRÁCE — ING. PETR KOVÁŘ
SUPERVISOR

BRNO 2008

ABSTRAKT

Tato diplomová práce se zabývá metodami pro výběr nejbližší RTP proxy k VoIP klientům s použitím IP anycastu. RTP proxy servery jsou umístěny v síti Internetu a přeposílají RTP data pro VoIP klienty za síťovými překladači adres(NAT). Bez zeměpisně rozmístěných RTP proxy serverů a metod pro nalezení nejbližšího RTP proxy serveru by došlo ke zbytečnému poklesu kvality přenosu médialních dat a velkému zpoždení. Tento dokument navrhuje 4 metody a jejich porovnání s podrobnějšími rozbory metod s využitím DNS resolvování a přímo SIP protokolu. Tento dokument také obsahuje měření chování IP anycastu v porovnání mezi metrikami směrování a metrikami časovými. Nakonec dokumentu je také uvedena implemetace na SIP Express Router platformě.

KLÍČOVÁ SLOVA

SIP, Anycast, RTP proxy, IP směrování, NAT

ABSTRACT

This thesis is about using IP anycast-based methods for locating RTP proxy servers close to VoIP clients. The RTP proxy servers are hosts on the public Internet that relay RTP media between VoIP clients in a way that accomplishes traversal over Network Address Translators (NATs). Without geographically-dispersed RTP proxy servers and methods to find one in client's proximity, voice latency may be unbearably long and dramatically reduce perceived voice quality. This document proposes four methods their comparison with further design of DNS-based and SIP-based methods. It includes IP anycast measurements that provides an overview of IP anycast behaviour in terms of routing metrics and latency metrics. It also includes implementation on SIP Express Router platform.

KEYWORDS

SIP, Anycast, RTP proxy, IP Routing, NAT

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Architektura pro globální distribuovanou SIP síť s využitím IPv4 anycastu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

First of all I would like to express my appreciation and thanks to my advisor Dipl.-Ing. Jiří Kuthan for his guidance and support, as well as many inspiring discussions and constructive criticism. He deserves recognition for his tremendous contribution to this work.

My special thanks go to Ing. Petr Kovář, my mentor at the Brno University of Technology, for all the valuable discussions, support, and understanding.

My gratitude goes to my entire family, especially my parents, who always supported and encouraged my studies over the years.

GLOSSARY

Affinity

Tendency of subsequent packets of a "connection" to be delivered to the same target.

Anycast

Anycast is a network technique which allows a client to access the nearest host of a group of hosts that provide the same service.

Autonomous System(AS)

Autonomous System(AS) is a set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets within the AS, and using an exterior gateway protocol(IBGP) to route packets to other ASs.

ICE (Interactive Connectivity Establishment)

A Methodology for Network Address Translator (NAT) Traversal for Multimedia Session Establishment Protocols.

NAT

Network Address Translation

Proximity

Ability to find close-by members of the anycast group.

RTP (Real-time Transport Protocol)

RTP is designed to provide end-to-end network transport functions for applications transmitting real-time data, such as audio, video or simulation data over multicast or unicast network services.

RTT (Round-Trip Time)

SDP (Session Description Protocol)

SDP is intended for describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation.

SIP (Session Initiation Protocol)

IETF standard for session initiation in multi-purpose communication systems.

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

STUN is a lightweight protocol that allows applications to discover the presence and types of NATs and firewalls between them and the public Internet.

TURN (Traversal Using Relay NAT)

TURN is a protocol that allows for an element behind a NAT or firewall to receive incoming data over TCP or UDP.

UAC (User Agent Client)

A User Agent Client is a logical entity that creates a new request.

UAS (User Agent Server)

A User Agent Server is a logical entity that generates a response to a SIP request. The response accepts, rejects, or redirects the request.

UA (User Agent)

A User Agent acts as both a User Agent Client and User Agent Server. It is an end device in a SIP network. They originate SIP transactions turning to dialogs and media sessions. Alternatively, a user agent can be a gateway to another network, such as a Public Switched Telephone Network (PSTN) gateway.

URI (Uniform Resource Identifier)

VoIP (Voice over IP)

The transmission of voice over data networks that use the Internet Protocol (IP).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

This thesis is about using IP anycast-based methods for locating RTP proxy servers close to VoIP clients. The RTP proxy servers are hosts on the public Internet that relay RTP media between VoIP clients in a way that accomplishes traversal over Network Address Translators (NATs). Without geographically-dispersed RTP proxy servers and methods to find one in client's proximity, voice latency may be unbearably long and dramatically reduce perceived voice quality. We are focusing on using IP anycast to find a reasonably close RTP proxy. IP anycast is Internet's capability to route IP packets from a source to one of multiple possible destinations. The destinations share the same IP address block and are advertised using a routing protocol. The choice of destination is made by routers using available routing tables. IP anycast is not a stable environment for stateful protocols(notably TCP), especially "long lived" sessions. However, recent studies and measurements[5] showed that the use of IP anycast may be even deployed with stateful services when deployed carefully and there has already been an existing deployment such as CacheFly[16]. The major use of IP anycast today is by DNS root-servers since requests sent to DNS servers are on query/reply basis, in other words it is a stateless service that do not suffer from routing instabilities.

As IPv4 address space suffers from shortage, NATs have significantly delayed this shortage but caused that almost anyone on the world is behind NAT. SIP (Session Initiation Protocol) as one of VoIP protocols and its deployment is not able to deal with SIP clients behind NAT on itself. There were introduced technologies handling this issue such as STUN, TURN, ICE or in the worst case an RTP proxy – a server through which RTP packets are relayed. If such an RTP proxy is too far away from both SIP clients, resulting latency is going to impair perceived voice quality dramatically. It is thus important for global SIP deployments, to have a network of geographically dispersed RTP proxy servers and actually use those that are close to the clients.

The focus of this work is anycast-based mechanisms for discovering an RTP proxy in SIP client's proximity. The mechanism shall satisfy the following criteria: it shall be easy to integrate with state-of-the-art SIP clients and servers, allow for fail-over on a geographically dispersed basis and be resilient against routing instabilities despite use of anycast. In particular dealing with routing instabilities is important as IP anycast tends to be sensitive to those. Delivery of subsequent packets to different anycast destinations can cause broken transactions on transport or application level if stateful.

The rest of this work is structured as follows. In chapter 1, readers are introduced

to background technologies: SIP and IP anycast. In chapter 2, readers get familiarized with related work done in the field. In chapter 3 we are reviewing several architectural options and their trade-offs. The options of our choice and their implementation details are explained in chapter 4. Finally chapter 5 provides conclusion and notes about future work.

1 BACKGROUND TECHNOLOGIES

1.1 SIP

The protocol is used for creating, modifying, and terminating sessions with one or more participants. By sessions we understand a set of senders and receivers that communicate and the state kept in those senders and receivers during the communication. Examples of a session can include Internet telephone calls, distribution of multimedia, multimedia conferences etc. For more about SIP see 2.1.

SIP on itself is not capable to handle SIP clients behind NATs. There were introduced supporting technologies handling this issue such as STUN, TURN, ICE or in the worst case simply RTP proxy.

1.1.1 Supporting Technologies Dealing with NAT

STUN

Simple Traversal of User Datagram Protocol (UDP) through Network Address Translators (NAT) or STUN is defined in RFC 3489 [30]. It provides a lightweight protocol that allows User Agents to probe and discover the type of NAT that exist between the User Agent and the STUN server on the public network. It also provides details of the external IP address/port combination used by the NAT device to represent the NATed UA on the public facing side of a NAT. On learning of such an external representation, a UA can use accordingly as the connection address in SDP to provide NAT traversal. STUN only works with Full Cone, Restricted Cone and Port Restricted Cone type NATs. STUN does not work with Symmetric NATs as the technique used to probe for the external IP address/port representation.

If SIP User Agents discover that it can traverse the NAT using STUN then it will do so and such UA will look like a UA with the public IP to SIP proxy – special treatment or use of RTP proxy is not necessary.

TURN

As mentioned above, STUN protocol does not work for UDP traversal through a Symmetric NAT. Traversal Using Relay NAT (TURN) provides the solution for UDP and TCP traversal of symmetric NAT. TURN is very similar to STUN in both syntax and operation. It provides an external address at a TURN server that will act as a relay and guarantee traffic will reach the associated private address. The full details of the TURN specification are defined in [26]. A TURN service will almost always provide media traffic to a SIP User Agent but it is recommended that

this method only be used as a last resort and not as a general technique for NAT traversal. This is because using TURN has high performance costs when relaying media traffic and can lead to unwanted latency.

ICE

A lot of NAT traversal techniques have been introduced, but none of them works universally or are applicable to all real world scenarios. These techniques make use of Connection Oriented Media, STUN, TURN, ALG and so on. All the techniques have been collected into one single document, which is called ICE. ICE (Interactive Connectivity Establishment) is a methodology for traversing NAT, but it is not a new protocol. It is a collection of all previously mentioned attempts to traverse NAT which work universally. The methodology is quite complex and requires mutual cooperation of all SIP entities involved in the communication. Refer to [27] for more details about ICE.

RTP Proxy Servers

RTP proxy servers are intermediate servers for media sessions established by SIP proxies. RTP proxy server is used as a last resort when no NAT technique is able with helping to traverse the media over NAT. Mostly, when symmetric NATs are involved. Using RTP proxy servers are similar to TURN servers but the difference is that SIP clients have TURN support and can ask for relaying RTP stream over TURN server on its own whilst RTP proxy servers are controlled from SIP proxy. SIP proxies forcibly rewrites SDP bodies in SIP messages enforcing NAT compatible symmetric packet flows.

RTP proxy servers introduce few drawbacks:

- adds extra hop for media that cause higher voice latency in between SIP clients depending on geographic location of RTP proxy
- may reduce perceived voice quality depending on network capacity and traffic

RTP proxy servers may be co-located with SIP proxies or may run separately from SIP proxies managed using an RTP control protocol. If they run remotely there may be introduced some delay issues while establishing a call that may cause timing out of some transaction timers.

1.1.2 Reference Network Organization

A typical state-of-the-art SIP deployment is organized in a cluster consisting of the SIP servers, frequently referred to as home proxies (HPs), and load balancers (LBs).

The load balancers distribute SIP traffic over the home proxies and also manage their availability. Either the load balancers or home proxy servers implement NAT traversal. Such SIP clusters appear as a simple SIP proxy to outside network. The home proxy servers share in some way database with SIP-related data.

In our architecture, we additionally put "fronting-elements" in front of existing SIP clusters(see Figure 1.1). The task of these fronting elements is discovery of the nearest RTP proxy using anycast and the RTP proxy functionality itself. A key design objective is to be able to put this distributed auto-discovered RTP network in front of existing SIP clusters without need for additional support in the cluster or client. As a side-effect the fronting-elements must also take over the NAT traversal role – this must be always done by the element closest to the clients unless it is fully transparent. We are leaving the more detailed definition of the fronting element to Chapter 3, in which we actually describe the NAT traversal details and several different designs of RTP proxy discovery.

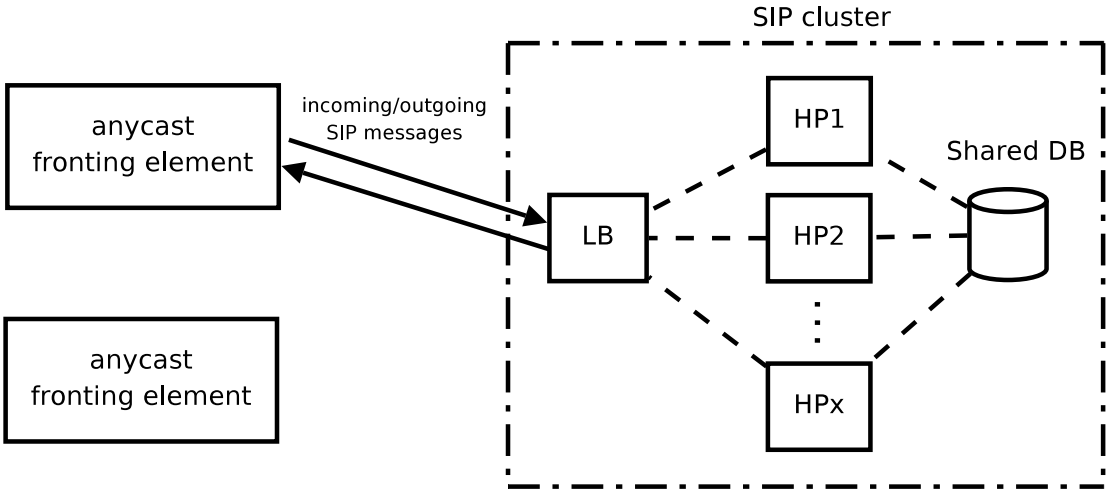


Fig. 1.1: SIP cluster

We have deployed this network architecture with fronting elements located in Prague and Berlin. As part of getting those available to the Internet with propagated anycast IP address we measured the convergence time(time to take over the service when one anycast node fail). For more details about convergence time see Section 1.2.3.

1.2 Anycast

Anycast is a network technique which allows a client to access the nearest host of a group of hosts that provide the same service. The nearest host is defined according

to the routing system's measure of distance. Usually, those hosts in the anycast group are replicas, able to provide the same service. To take an advantage of an anycast, servers are distributed topologically and geographically across the Internet. An anycast deployment solely depends on the network, routers and routing protocols. More detailed description of anycast is in Chapter 2.4.

We are examining several different uses of anycast for sake of discovery of RTP proxy. Anycast does not provide the best proximity in terms of latency but at least eliminate the worst case scenarios. It helps to select an RTP proxy as close as possible to one of SIP clients in session. A particular problem to deal with is anycast's sensitivity to routing instability. This problem is addressed in detail in Chapter 3.

1.2.1 Pros and Cons of Anycast

General pros and cons of anycast for selection an RTP proxy include:

Pros:

- locality/latency improvements by reducing network distance between client and RTP proxy servers (at least eliminating the worst case)
- high availability - provides a service without outages
- reduce list of geographically dispersed servers to a single distributed anycast address

Cons:

- IP anycast wastes the address space(the longest IP prefix is /24), even though one IP address used for running a service(see Section 2.4.3). This is because of BGP policy and route propagation in the Internet.
- Anycast may break connection affinity (Ballani et al[5] measured that this issue is quite negligible)
- IP anycast does not always offer the nearest anycast server (latency-based proximity).
- BGP sometimes converges slowly(when a service became unavailable it may make the service unreachable for even minutes[17]).
- Not suitable for "long lived" sessions if not handled carefully (keeping the TCP context)

1.2.2 Non-anycast Server Selection Alternatives

Another techniques for server selection have been developed. Such as virtual coordinate systems, on-demand probing overlays and some kinds of application-layer anycasts(see Section 2.4.7). However, these techniques need co-operation on application layer. The advantage of IP anycast is that it can be transparently handled on IP layer.

1.2.3 Convergence Measurements

Methodology

We wanted to find out how fast the convergence of our network setup is so we used planet-lab.org, a global research network, to measure latency of ICMP echo replies from anycast nodes in Prague and Berlin. We measured convergence time when one of BGP daemon stopped propagation of its route. The convergence time is expected not to be very high since Prague and Berlin are not so far from each other. Out of these measurements we are also able to derive the time of the service unavailability in case there is running an application server.

On planet-lab hosts were installed two scripts. One of them measured ICMP echo replies in 8 seconds interval and the other one was collecting results from traceroute in 2 minutes interval to see what anycast node is used for a particular planet-lab host.

Results

1. Table of ICMP packet destination to anycast nodes measured from 142 planet-lab hosts.

Anycast node	**No. of hosts routed to the node**
Prague	103
Berlin	39
$\sum$	142

Tab. 1.1: Destinations of planet-lab host ICMP requests

2. Convergence time of ICMP packets originally destined to Prague and then re-routed to Berlin (see Figure 1.2)

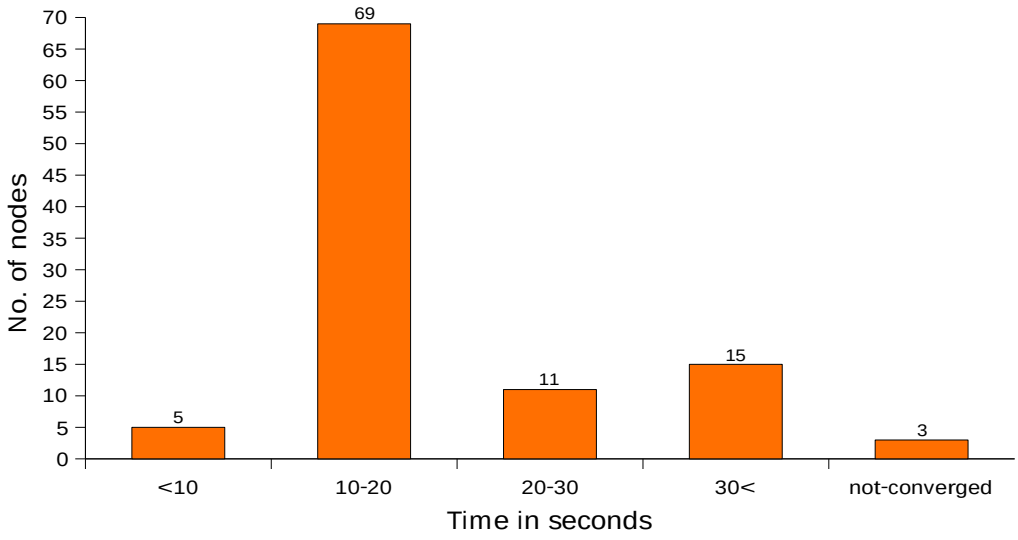


Fig. 1.2: Route convergence time to Berlin's anycast node

Not-converged: 3 planet-lab hosts could not reach new destination. Interestingly, all 3 planet-lab hosts are situated in Italy.

3. Convergence time of ICMP packets originally destined to Berlin and then rerouted to Prague (see Figure 1.3)

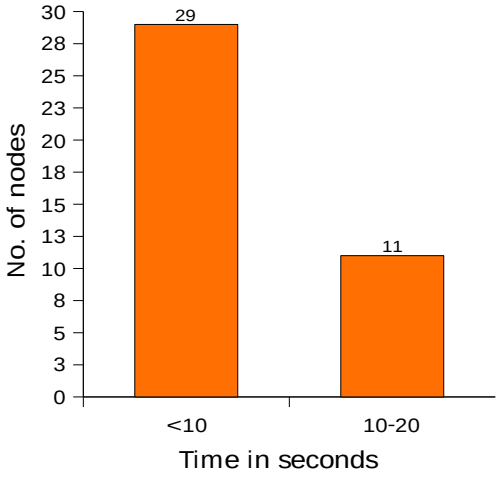


Fig. 1.3: Route convergence time to Prague's anycast node

Conclusion about Measurements

Most of planet-lab hosts directed ICMP packets to Prague node. It shows that in routing path prospective Prague anycast node is situated within an ISP(Internet Service Provider) on a back-bone that is more accesible in terms of routing metrics from the Internet. This does not always provide the best geographic location proximity to a given anycast node though. In our case to find the nearest RTP proxy, selected anycast node may not be the best to SIP client's location. ICMP packets from a lot of planet-lab hosts originally destined to Berlin did not register any service outage meaning that the convergence time was very fast for them.

1.2.4 Latency Measurements

Planet-lab network provides many nodes all around the world giving us a good image of routing in the Internet. The measuring of latency is ICMP echo based which give us a knowledge if our design proposals provide a good proximity for SIP clients which is our main goal.

We measured following destinations(IP addresses):

- anycast/Prague
- anycast/Berlin
- unicast/Prague
- unicast/Berlin

Methodology

From each planet-lab host we measured latency of ICMP echo replies of each IP address and also "tracerouted" to see what anycast box was "selected" by the particular planet-lab node. For better statistical results we collected 100 echo replies for each IP address destination where each planet-lab host produced RTT stats with min/avg/max values. Presented stats are for min value. For the average value there were nearly the same results (differs just in one planet-lab node).

Stats

We were interested if anycast nodes give the best proximity for SIP clients in comparison to unicast IP address destinations. The stats show:

- if anycast gave the best proximity by matching the unicast shortest-ping and anycast-ping

- time difference of anycast RTT against unicast RTT to the same place
- time difference of anycast RTT against shortest-ping destination
- difference of the same as both above in per-centage

The full results are included in Appendix A.

Does anycast ping destination match to shortest unicast ping?

ICMP packets from **146** planet-lab nodes were routed to Prague anycast node and from **49** planet-lab nodes packets were routed to Berlin anycast node. **32.8%(64 nodes)** matched and **67.2%(131 nodes)** did not match the measured latency of selected anycast destination and unicast ping to the same destination. Measured **out of total 195** planet-lab nodes. As can be seen using anycast does not give the best proximity. This is caused by close geographic location of Prague and Berlin and not so much different routing path for packets from planet-lab nodes. Figure 1.4 shows the results in graph.

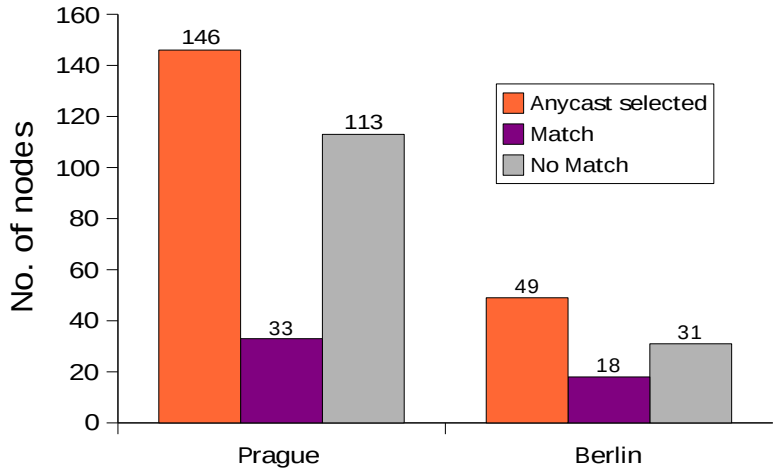


Fig. 1.4: Anycast ping compared to unicast shortest ping destinations

The gap between latency of the shortest unicast ping and anycast ping (for not matched hosts)

Following graph (Figure 1.5) shows that anycast in our scenario does not really give the best proximity. The conclusion about the graph is that anycast nodes should not be very close to each other to give better results.

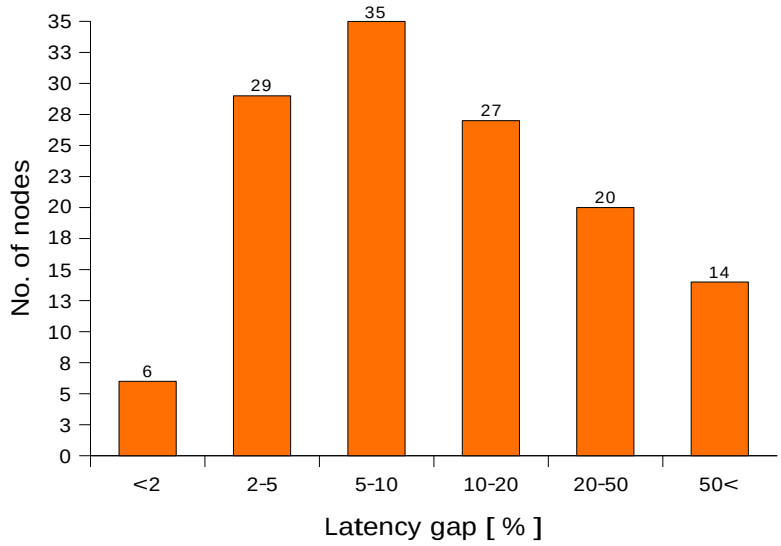


Fig. 1.5: The gap between latency of the shortest unicast ping and anycast ping

Unicast and anycast routes are different even for the same physical destination

- 46 planet-lab hosts measured that anycast RTT is lower than unicast RTT.
- 44 planet-lab hosts measured that anycast RTT is higher than unicast RTT.

2 SIP AND ANYCAST IN DETAIL

2.1 SIP

SIP (Session Initiation Protocol) is application-layer control protocol which has been developed and designed within the IETF(Internet Engineering Task Force). The protocol has been designed with easy implementation, good scalability, and flexibility in mind.

The specification is available in form of several RFCs and the most important one is RFC3261 [24] which contains the core protocol specification.The protocol is used for creating, modifying, and terminating sessions with one or more participants. By sessions we understand a set of senders and receivers that communicate and the state kept in those senders and receivers during the communication. Examples of a session can include Internet telephone calls, distribution of multimedia, multimedia conferences etc.

However, SIP is the subject of numerous specifications that have been produced by the IETF. It can be difficult to locate the right document, or even to determine the set of Request for Comments (RFC) about SIP. There is a specification covering completely SIP at <https://datatracker.ietf.org/drafts/draft-ietf-sip-hitchhikers-guide/>. This specification serves as a guide to the SIP RFC series. It lists the specifications under the SIP umbrella, briefly summarizes each, and groups them into categories.

2.1.1 Protocol Structure

Communication using SIP (often called as signalling) includes series of messages. SIP messages can be transported independently by the network usually over TCP, UDP or TLS. They are text based and the syntax and header fields are quite similar to HTTP. Each message consist of "first line", message header, and message body. The first line identifies type of the message. There are two types of messages - requests and responses. Requests are usually used to initiate some action or inform the recipient with something (connection info etc.). Responses are used to confirm that a request was received and processed and contain the status of the processing.

Following is a typical SIP request:

```
INVITE sip:admin@iptel.org SIP/2.0
Via:SIP/2.0/UDP 192.168.1.101:5060;rport;branch=z9hG4bK9FF9B
From: natuser <sip:nateduser@iptel.org>;tag=223549693
To: <sip:admin@iptel.org>
Contact: <sip:nateduser@192.168.1.101:5060>
```

```
Call-ID: 032BC0C9-C29E-4F23-9558-CDA469FFE75C@192.168.1.101
CSeq: 7111 INVITE
Max-Forwards: 70
Content-Type: application/sdp
User-Agent: X-Lite release 1103m
Content-Length: 241

v=0
o=nateduser 4955765 4955765 IN IP4 192.168.1.101
s=X-Lite
c=IN IP4 192.168.1.101
t=0 0
m=audio 8000 RTP/AVP 0 8 97 101
a=rtpmap:0 pcmu/8000
a=rtpmap:8 pcma/8000
a=rtpmap:97 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

The first line says it is an INVITE message, which is used to establish a session. The URI(Uniform Resource Identifier) on the first line `-- sip:admin@iptel.org` is called Request URI and contains URI of intended next hop of the message. In this case it will be host `iptel.org`.

A SIP request can contain one or more Via header fields which are used to record path of the request. They are later used to route SIP responses exactly the same way. This INVITE message contains just one Via header field which was created by the user agent that sent the request. From the Via field we can tell that the user agent is running on host 192.168.1.101 and port 5060. Branch parameter of Via header fields contains a transaction identifier.

From and To header fields identify initiator (caller) and recipient (callee) of the invitation. From header field contains a tag parameter, which serves as a dialog identifier.

Contact header field contains IP address and port where the sender is awaiting further requests sent by callee.

Call-ID header field is a dialog identifier and its purpose is to identify messages belonging to the same call. Such messages have the same Call-ID identifier. CSeq is used to maintain order of requests. Because requests can be sent over an unreliable transport that can re-order messages, a sequence number must be present in the messages so that recipient can identify retransmissions and out of order requests. The Max-Forwards serves to limit the number of hops a request can transit on the way to its destination and protects from possible loops. Other header fields are self-explanatory.

Message header is delimited from message body by an empty line. Message body of the INVITE request contains a description of the media type accepted by the

sender and encoded in SDP(Session Description Protocol).

2.1.2 SIP Requests

Above is described how an INVITE request looks like and mentioned that the request is used for invitation a callee to a session.

Other important requests are:

- ACK - This message acknowledges receipt of a final response to INVITE.
- BYE - Bye messages are used to tear down multimedia sessions.
- CANCEL - Cancel is used for cancelling not yet fully established session.
- REGISTER - Purpose of REGISTER request is to let registrar know of current user's location.

2.1.3 SIP Responses

When a user agent or proxy server receives a request it send a reply. Each request must be replied except ACK requests which trigger no replies.

A typical reply looks like this:

```
SIP/2.0 200 Ok
Via: SIP/2.0/UDP 192.168.1.50;branch=z9hG4bK9fbd.a095ba92.0
Via: SIP/2.0/UDP 10.0.10.3:5060;received=192.168.1.100;rport=5060
;branch=z9hG4bKB8D1CE9011C544AB90EA794B9C56D16E
From: natuser <sip:nateduser@iptel.org>;tag=223549693
To: <sip:admin@iptel.org>;tag=3280384206
Contact: <sip:admin@192.168.1.101:5060>
Record-Route: <sip:192.168.1.50;ftag=223549693;lr=on>
Call-ID: 032BC0C9-C29E-4F23-9558-CDA469FFE75C@10.0.10.3
CSeq: 7112 INVITE
Content-Type: application/sdp
Server: X-Lite release 1103m
Content-Length: 0
```

As can be seen, responses are very similar to requests, except for the first line. The first line of response contains protocol version (SIP/2.0), response code, and reason phrase.

The reply code is an integer number from 100 to 699 and indicates type of the replies. There are 6 classes of replies.

2.1.4 User Agent

User Agent(UA) is an Internet end-point that use SIP to find another end-point where negotiating session characteristics between each other. User Agents usually, but not necessarily, reside on a user's computer in form of an application - this is currently the most widely used approach, but user agents can be also cellular phones, PSTN gateways, PDAs, automated IVR systems and so on.

User Agents are often referred to as User Agent Server (UAS) and User Agent Client (UAC). UAS and UAC are logical entities only, each user agent contains a UAC and UAS. UAC is the part of the user agent that sends requests and receives responses. UAS is the part of the user agent that receives requests and sends responses.

2.1.5 SIP Proxy

SIP Proxy servers are very important entities in the SIP infrastructure. They perform routing of a session invitations according to callee's current location, authentication, accounting and many other important functions.

2.1.6 SIP Registrar

The registrar is a special SIP entity that receives registrations from User Agents, extracts information about their current location (IP address, port and username in this case) and stores the information into location database.

2.1.7 Record Routing

All requests sent within a dialog are by default sent directly from one User Agent to the other. Only requests outside a dialog traverse SIP proxies. This approach makes SIP network more scalable because only a small number of SIP messages hit the proxies.

There are certain situations in which a SIP proxy need to stay on the path of all further messages. For example, proxies controlling NAT devices or proxies doing accounting need to stay on the path of BYE requests.

Mechanism by which a proxy can inform user agents that it wishes to stay on the path of all further messages is called *record routing*. Such a proxy would insert `Record-Route` header field into SIP messages which contains address of the proxy. Messages sent within a dialog will then traverse all SIP proxies that put a `Record-Route` header field into the message.

The recipient of the request receives a set of `Record-Route` header fields in the message. It must mirror all the `Record-Route` header fields into responses because the originator of the request also needs to know the set of proxies.

2.2 RTP

RTP (Real-time Transport Protocol) defines a standardized packet format for delivering audio and video over the Internet. This protocol can be used for media-on-demand or for interactive services such as Internet telephony. It goes along with the RTP Control Protocol (RTCP) and it's built on top of the UDP[28].

When used with SIP signalling, parameters for RTP stream are negotiated through SDP documents. This way clients decide what media format will be figuring in a session.

RTP streams and NATs

There can be a number of RTP streams in a session. In the case where is a session between two User Agents, there are two RTP streams, one in each direction (sending and receiving RTP packets). If one of the User Agents involved in the session is with private IP address, that stream from the public UA towards the NAT will not be allowed to reach the UA on the inside of the NAT. Therefore the UA with public IP must send the packets to the source IP address and port of packets coming from the UA behind NAT.

Following Figure 2.1 shows both way direction RTP streams and also points out required feature for successful NAT traversal - symmetric RTP. In short a User Agent receives and sends packets on the same port. Currently, most of UAs has been supporting this feature as default.

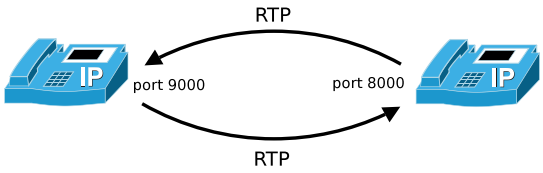


Fig. 2.1: RTP streams - User Agents receive and send packets on the same port(symmetric RTP)

2.2.1 SDP Documents

SDP(Session Description Protocol) is intended for describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session invitation. Within these SDP documents a SIP User Agent usually sends its IP address and port where RTP stream can be received. SDP document also includes a set of supported media codecs by the User Agent.

SDP document generated by a User Agent include lines `"c="` and `"m="` with IP address and port where media can be received.

For Example:

```
c=IN IP4 147.229.213.156
m=audio 8000 RTP/AVP 0 8 97 101
```

2.3 NAT Traversal using a SIP Proxy with an RTP Proxy

This section describes what must be changed at SIP proxy to make sure that SIP messages get delivered back to UA behind NAT and UAs establish two-directional media session. The key idea is making all traffic symmetric which is known to accommodate most of available NATs. Figure 2.2 shows possible scenario where one SIP client is behind NAT and RTP proxy is used for relaying media.

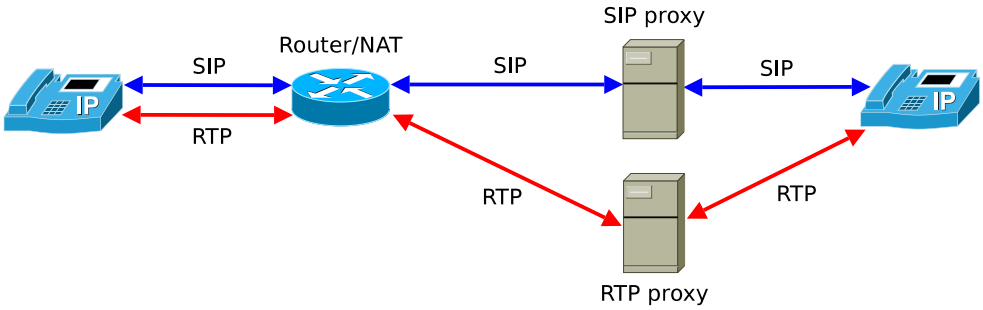


Fig. 2.2: NAT traversal

2.3.1 SIP Requests

Via Header

When a SIP proxy receives a request (e.g. REGISTER, INVITE), it examines the topmost Via header field value. If this Via header field value contains an ”rport” parameter with no value, it must set the value of the parameter to the source port

of the REGISTER request. This is analogous to the way in which a SIP proxy will insert the "received" parameter into the topmost Via header field value which is source IP address where a request came from.

For example:

IP address/port in Via header field received by SIP proxy.

```
Via:SIP/2.0/UDP 192.168.1.101:5060;rport;branch=z9hG4bK9FF9B128
```

SIP proxy add source IP address/port to received and rport parameters.

```
Via:SIP/2.0/UDP 192.168.1.101:5060;rport=1024;received=1.2.3.4
;branch=z9hG4bK9FF9B128
```

Contact Header

To keep a UAC routable from public network SIP proxy overwrites Contact header IP address/port to source IP address/port and saves it in user location databases for subsequent requests.

Contact IP address is used for forming new SIP requests by SIP User Agents.

For example:

Original Contact address.

```
Contact: <sip:admin@192.168.1.1:5060>
```

Overwritten by SIP proxy to routable contact IP address/port from public network and saved in location database.

```
Contact: <sip:admin@1.2.3.4:1024>
```

2.3.2 SIP Responses

Via Header

SIP response created with "rport" and "received" parameters in the Via header and sent to IP address/port where IP address is "received" parameter and port is "rport" parameter.

For UASs the response must be sent from the same address and port that the request was received on in order to traverse symmetric NATs. This is also called symmetric signalling extension. Refer to RFC3581[25] for more information.

Contact Header

Contact header field is appended to the response, which will contain the current location of the UA. It is the same approach as it is done in SIP requests.
Note: Contact header is appended to 2xx and 3xx responses only.

2.3.3 SDP and NATs

If SIP proxy detects that SIP request (e.g. INVITE) is received from UA behind NAT, media must be relayed. This is achieved by forcing the RTP media to traverse an RTP proxy. The SIP proxy server must then replace UA's private IP address in SDP payload with IP address of the RTP proxy. (Note that communication of the SIP proxy server with the RTP proxy is out of scope of this section.)

The example shows particular lines in SDP payload:

```
o=nateduser 4955765 4955765 IN IP4 192.168.1.101
```

This line describes the sender of this SDP message and its IP address. This address has to be changed to RTP proxy server's IP address.

```
c=IN IP4 192.168.1.101
```

This line indicates the IP address where the UA will be ready to receive RTP packets. It also has to be changed by SIP proxy to RTP proxy address.

```
m=audio 8000 RTP/AVP 0 8 97 101
```

Eventually, UA's listening port number advertised in `"m="` line must be replaced with RTP proxy server's.

2.4 Anycast

2.4.1 Network-layer(IP) anycast

IP Anycast is a network technique which allows a client to access the nearest host of a group of hosts that share the same anycast IP address, where the nearest host is defined according to the routing system's measure of distance. It is also referred as one-to-any communication where "any" means one host of the anycast group. Usually, those hosts in the anycast group are replicas, able to provide the same service. To take an advantage of an anycast, servers are distributed topologically and geographically across the Internet. An IP anycast deployment solely depends on the network, routers and routing protocols. The scale of anycast deployment within the routing system can vary from a small network handled by Interior Gateway Protocol(IGP)[3], to Border Gateway Protocol(BGP)[2], handling requests from the global Internet. Figure 2.3 shows the basic idea of a network-layer(IP) anycast deployment.

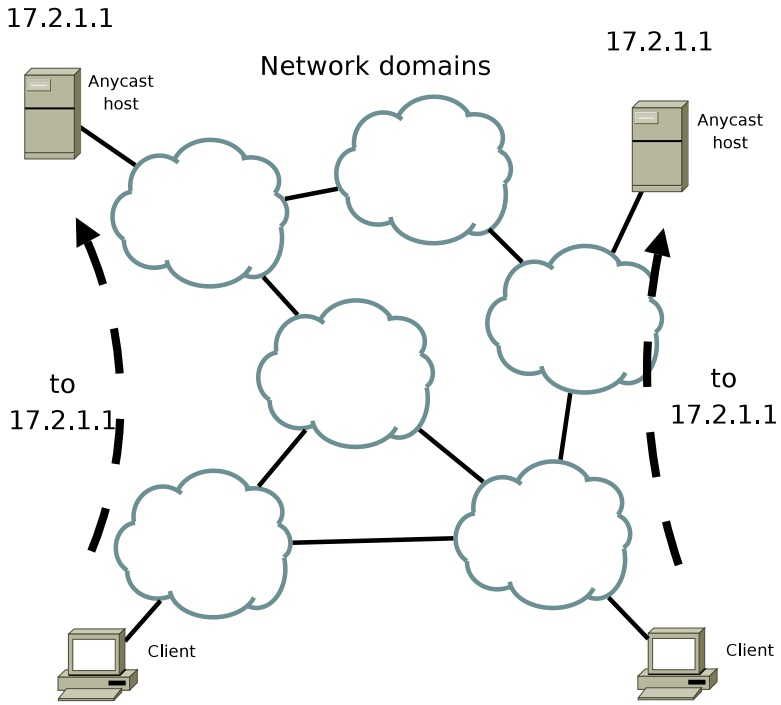


Fig. 2.3: IP anycast mechanism

Patridge et al[19] originally proposed the idea of anycast at the network-layer(IP). They defined that anycast is a stateless best effort delivery of an anycast datagram to at least one host, and preferably only one host. In RFC4786[1] J. Abley and K. Lindqvist cover the best current practices of using IP anycast or Kevin Miller[18] very well summarize deploying of IP anycast.

Ballani et al.[4] states that today deployment of IP anycast is quite limited to just query/reply services such as for DNS root servers[2], primarily to spread the load as a defence against DoS attacks. On the other hand Ballani et. al.[5] performed some measurements regarding proximity[1] and affinity[2] and states that IP anycast is also a good candidate for using other services based on TCP or applications with long-lived sessions(2.4.4). They found that IP anycast itself in global deployment provides good affinity. The measurement states that 93.75% of the source-destination pairs never changed(probability of selecting the same anycast node). In other words, the probability that a two minute(or one hour) connection would experience a change is roughly 1 in 13000 (or 1 in 450).

There has also been an existing deployment such as CacheFly[16] that uses anycast for their stateful service.

Routing Schemes

To make it clear here is just a short overview of communication ways in a network.

- **Unicast** - the process of sending a packet from one host to an individual host.
- **Broadcast** - the process of sending a packet from one host to all hosts in the network
- **Multicast** - the process of sending a packet from one host to a selected group of hosts
- **Anaycast** - the process of sending a packet from one host to an individual(nearest) host out of group of hosts

2.4.2 Common IP Anycast Deployments

AS 112 project

The anycasted AS112 servers are used to draw in reverse DNS queries to and for the link local address space (RFC1918 addresses – 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16). In other words they use anycasted sink-hole servers.

DNS Root Servers

Wide-scale deployment of DNS root servers. Anycasting of six of the thirteen root-servers C, F, I, J, K and M root. It takes advantage of simple query/reply behaviour.

[1] ability to find close-by members of the anycast group.

[2] tendency of subsequent packets of a "connection" to be delivered to the same target.

On local scale, IP anycast is used by operators to simplify and improve local DNS server availability.

IPv4-to-IPv6 relays

6to4 routers involve connecting v6 networks across v4 infrastructure. Anycast provides an easy way for end sites to locate relays into the native IPv6 world by using globally known IPv4 anycast prefix for 6to4 routers.

Rendezvous Discovery for IP Multicast

IP multicast packets are routed to shared multicast Rendezvous points using IP anycast address.

2.4.3 Routing Consideration

Addressing in IP Anycast

IP anycast address is an IP address which identifies a group of nodes(servers). This address is then assigned to each anycast node. IP anycast address must also be chosen from IP address space(prefix) that corresponding routes will be allowed to propagate within given routing system[1]. The length of prefix must be sufficiently short that it will not be discarded by commonly-deployed import policies in BGP speaking routers.

For an IPv4 numbering and deployment across the Internet the IP address is given by an address space where the minimum RIR(Regional Internet Registry) allocation size is 24 bits. It means that reachability of a service with anycast address would be in /24 subnet (24-bit prefix) for example 112.54.8.0/24. The disadvantage is that it uses the address space inefficiently.

An anycasted service deployed within a private network[22] can use locally-unused address and that address might be reached by 32-bit host route. This also apply for deploying anycast within area under single administration such as an autonomous system. The anycast service is within IGP has no inherent restrictions on the length of prefix as stated in [1].

In IPv6 network IP anycast addresses are not scoped differently from unicast addresses. However, IPv6 Anycast is beyond the scope of this document.

Route Advertising

Members of an anycast group have to indicate to the routers that they wish to receive anycast packets. One approach is to have the anycast host run a routing protocol and be able to advertise its anycast address to other routers in a network. Section 2.4.5 describe network configuration for intra-domain and inter-domain routing.

Service Management

Although each anycast host is intended to be reached by a particular community of clients via anycast address, there is also a requirement to be able to reach individual hosts in a predictable fashion for the purposes of systems administration, and so that service performance can be monitored. For this reason each host has a unique, unicast management IP address associated with it.

2.4.4 UDP, TCP transports and Anycast

It is important to remember that routing in the Internet is stateless. An anycast network has no obligation to deliver two successive packets sent to the same anycast host. This might happen when a client is topologically in the middle of two anycast hosts with equal-cost paths.

UDP

Since UDP transport is connectionless and anycasting is a stateless service, UDP can treat anycast addresses like regular IP addresses. A UDP datagram sent to an anycast address is just like a unicast UDP datagram from the perspective of UDP and its application.

Some services have very short transaction times, and may even be carried out using a single packet request and a single packet reply (e.g. DNS transactions over UDP transport). Here is no problem with Anycast.

Some services have long transaction times and need to exchange more datagram in between client and anycast host. This problem is discussed in Ballani's et al. paper[5] as connection affinity and concludes that packet delivery to different host is negligible.

TCP

TCP's use of anycasting is less straightforward because TCP is stateful. It is hard to envision how one would maintain TCP state with an anycast server when two

successive TCP segments sent to the anycast server might be delivered to completely different hosts.

Engel et al[12] propose a solution for this problem. This proposal is based on minor modification of TCP/IP stack at the host part where the anycast service is running. It does not require any modifications to routers and routing protocols. These modifications are limited to changes at the IP layer of the recipient of the TCP connection, making this scheme suitable to a client/server environment. Especially, it focuses on TCP transport protocol stateful connections since they tend to cause problems in anycast routing as described in [19]. The basic idea is to pin the end-host to which the first packet of the flow has been sent. The author states that it is very similar to route pinning in the context of QoS routing. The pinning is done by inserting a loose source route option in all subsequent packets from the same TCP flow.

2.4.5 Network Configuration

To deploy an anycast service there are two ways to set the anycast hosts up. Using either intra-domain routing or inter-domain routing configurations.

Intra-domain configuration

If the anycasted service is entirely within one routing domain(AS) or multiple intra-domain locations(more ASs but bound with an IGP protocol), only intra-domain consideration is needed. Routers need to be configured to deliver traffic to anycast servers either with static routes on first-hop router as shown in figure 2.4 or setting up dynamic routing by running a routing daemon on anycast hosts using for instance Zebra/Quagga. See figure 2.5. Static routes provides simple configuration but does not respond to server failure quickly. On the other hand it provides the ability to relocate servers without outage. Whilst in dynamic routes the anycast host is route originator and when the host is down the route is automatically withdrawn from routing system. Intra-domain anycast approach is described in [3].

Inter-domain Configuration

Setting up inter-domain routing for anycast is more difficult because this configuration needs its own AS, ISP independent IP prefix (see 2.4.3 and able to advertise the service anycast supernet. In this prospect Intra-domain routing must be correctly configured such as anycast servers can be IBGP peered and can use IGP redistribution. It must be able to withdraw routes when service is unavailable. Some deployments distinguish “global” nodes from “local” nodes[2] where global nodes

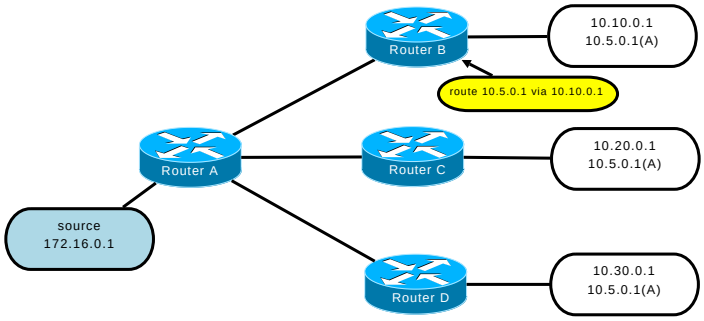


Fig. 2.4: Static IGP routes

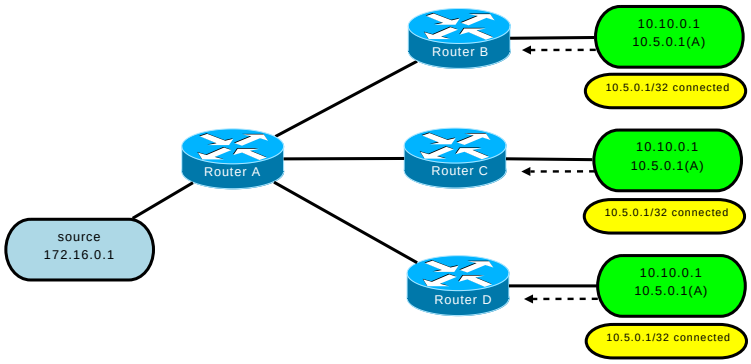


Fig. 2.5: Dynamic IGP routes

are announced to Internet routing system without restriction and local nodes add "no-export" BGP community attribute (2.5 to limit the clients that will use the node. For instance F-root DNS servers are using this approach. See section 2.4.5 for more details. Figure 2.6 shows advertisement of anycast address to the upstream ISP.

Global Nodes

In conjunction with an anycast service distribution across the global Internet, Global Nodes provides service to clients anywhere in the network. To be able to reach the service globally, BGP routers propagate reachability information, without restriction 2.5, by advertising routes covering the anycast service addresses for global transit to one or more ISPs.

More than one Global Node can exist for a single service which is commonly used (see Section 2.4.2 for reasons of redundancy and load-balancing.

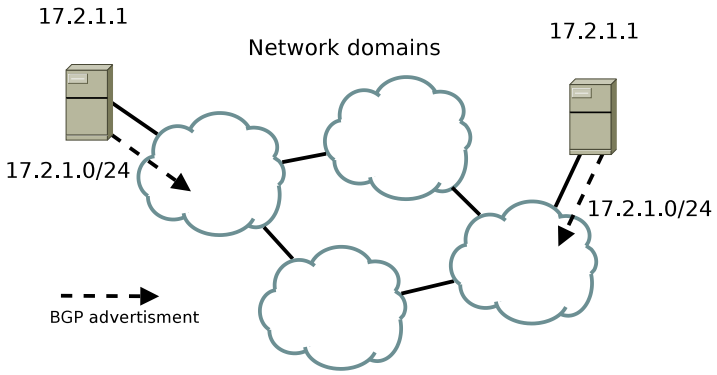


Fig. 2.6: BGP prefix advertisement

Local Nodes

On the other hand, it is sometimes desirable to deploy an anycast node which only provides services to a local catchment of autonomous systems, and which is purposely not available to the entire Internet. These nodes are referred to as Local Nodes. For instance a Local Node may be appropriate in regions with good internal connectivity but unreliable, congested or expensive access to the rest of the Internet.

Local Nodes advertise covering routes for anycast service addresses in a restricted way of propagation. This might be done using BGP community attribute such as **no_export** (covered more in Section 2.5.1) or **nopeer**[15] or by arranging with peers to apply a conventional "peering" import policy instead of a "transit" import policy, or some suitable combination of measures.

2.4.6 IP Anycast and its Characteristics

Ballani et al. [5] focused on measurements of IP Anycast despite previous studies did not report clear measurements and conclusions on IP Anycast performance such as failover, load distribution, proximity and affinity. Their measurements were accomplished on four existing IP Anycast deployments including two anycasted DNS root-servers and their own small scale IP Anycast service where they could test failure scenarios. The purpose of this study is to provide information on suitability of IP Anycast for stateful services.

Ballani's measurements states that current deployments such as J-root servers, does not offer good latency-based proximity. They found that approximately 40% of measured clients are directed to a root-server that is more than 100 msec farther away from the closest server and concluded that inter-domain routing metrics have an even more severe impact on the selection of paths to anycast destinations. The proposal is to ensure that an ISP, that provides transit to an anycast server, has

global presence and is (geographically) well covered by such servers improves the latency-based proximity offered by the anycast deployment. Basically, those measurements were compared between latencies of the unicast address and anycast address of tested server probed by clients. The unicast address is also usually used as management and monitoring access to servers.

IP Anycast is also affected by delayed routing convergence therefore clients using anycast service may experience slow failover. A failover may be caused by outage of anycasted service or BGP stability issues (see section 2.5.3. The already mentioned proposal addresses this by reducing the scope of routing convergence that follows a server failure and therefore may ensure fast failover for clients. The study shows a slow failover when anycast servers run in different ISP networks.

Their study concludes that IP Anycast offers good affinity to all clients with the exception of a small fraction that explicitly load balance traffic across multiple upstream ISPs. That means IP Anycast does not interact poorly with inter-domain routing and therefore should not significantly impact stateful services.

They also measured and load balanced servers by AS_PATH prepending which resulted in allowing for coarse-grained control over the distribution of client load across the deployment. AS_PATH prepending performs BGP speaker(on the anycast server) by adding its AS number more times in AS_PATH attribute which makes the anycasted server farther from clients.

2.4.7 Application-layer Anycast

Application-layer anycast is based on server or application metrics, such as available capacity, measured RTT(Round-trip time), number of active connections. However, application-layer anycast depends on an external entity that probes the location of clients, monitors the location and the status of servers in anycast group. Usually, it does not involve any change in clients but involves an overlay on existing routing infrastructure. Ballani et al[5] describes it as follows. One way of providing application-layer anycast is mapping high-level names, such as a DNS name, into one server of anycast group, returning the selected server's IP address to the client. Such an approach offer a number of advantages over IP anycast: it is easier to deploy, offers fine-grained control over the load on the servers and can provide very fast failover to clients. These advantages have led to the widespread adoption of application-layer anycast as a service discovery possibility. For example, commercial CDNs(Content Delivery Networks) use DNS-based redirection (in combination with URL-rewriting) to direct clients to an appropriate server.

Since application-layer anycast brings a lot of advantages it is not useful for

all protocols/applications. The fact that IP anycast operates at the network layer implies that it is only form of anycast that can be used by low-level protocols for example the use of anycast in IPv4-to-IPv6 relays. Ballani et al states that operating at the network layer gives IP Anycast a "ground level" resilience not easily achieved by application-layer anycast – for example, using DNS-based redirection to achieve resilience across a group of web servers requires first that the DNS servers themselves be available. It is this that makes IP Anycast particularly well suited for replicating critical infrastructures such as the DNS.

There are several projects using application-layer anycast approach such as OASIS(Anycast for Any Service)[13], Cisco DistributedDirector[10] and Application Layer Anycasting[7]. The up-to-date project and used by several services is OASIS which is further described in following section 2.4.7.

OASIS:Anycast for Any Service

Global anycast faces several requirements.

- must be fast and accurate
- must minimize probing to reduce risk of abuse complaints
- must scale to many services and provide high availability
- must integrate seamlessly with unmodified client applications

OASIS(Overlay-based Anycast Service InfraStructure)[13], a global distributed anycast system, addresses these challenges which allows legacy clients to find nearby or unloaded replicas for distributed services. Two main features distinguish OASIS from prior systems. First, OASIS allows multiple application service to share the anycast service. Second, OASIS avoids on-demand probing when clients initiate requests. This is because OASIS maintains locality information (an application independent way) by mapping portions of the Internet in advance(based on IP prefixes) to the geographic coordinates of the nearest known landmark.

OASIS, a shared locality-aware server selection infrastructure, allows a service to register a list of servers for later optimal selection which is also the primary approach. However, selection also bases on liveness and load of a individual server in a distributed service. OASIS can, for example, be used for locating IP anycast proxies[6], or it can select distributed SMTP servers in large email services.

Before introducing OASIS some other techniques have been used so far such as virtual coordinate systems (e.g. Vivaldi) and on-demand probing overlays. While on-demand probing potentially offers greater accuracy, it has several drawbacks. First,

probing adds latency and second, performing several probes to clients might trigger intrusion-detection alerts, resulting in abuse complaints.

OASIS eliminates on-demand probing(when clients make anycast requests) by probing (in OASIS -> clients direction) in the background. OASIS uses techniques which practically measure the entire Internet in advance. By leveraging the locality of the IP prefixes[14], OASIS probes only each prefix, not each client. In practice, IP prefixes from BGP dumps are used as starting point. OASIS is implemented at each service replica and thus delegates measurements to them. Service replica is a one copy of serving server in a distributed network.

To share OASIS across services and to make background probing feasible, OASIS requires stable network coordinates [3] for maintaining locality information. However, virtual coordinates tend to drift over time so instead, OASIS stores the geographic coordinates of the replica closest to each prefix it maps.

OASIS is publicly deployed on PlanetLab(http://www.planet-lab.org/) and has already been adopted by a number of services such as CoralCDN, OCALA, OpenDHT and more. The full list can be found at OASIS project page http://oasis.coralcdn.org/.

As a service selection algorithm it uses a DNS redirector that performs server selection upon hostname lookups, thus supporting a wide range of unmodified client applications(almost every network application firstly does DNS lookup before proceeding with other tasks). However, OASIS also provide HTTP and RPC interface for locality estimation.

System and design overview

The OASIS architecture combines reliable core nodes that implement anycast with a larger number of replicas belonging to different services that assist in network measurement. Firstly, every replica knows its geographic coordinates before any network measurement. Then, OASIS estimates the geographic coordinates of every netblock on the Internet(OASIS as a shared infrastructure spread measurement costs over many hosts). OASIS re-probe every physical location quite infrequently since IP prefixes rarely change[23].

The system consists of a network core nodes that help clients select appropriate replicas of various services as shown in Figure 2.7. All services employ the same core nodes. Replicas also run OASIS-specific code, both to report their own load and liveness information to the core, and to assist the core with network measurements. Clients need not to run any special code to use OASIS, because the core nodes

[3]Network coordinates provide a scalable way to estimate latencies among large numbers of hosts

provide DNS or HTTP based redirection. For example, an OASIS nameserver calls its core node with client resolver's IP address and a service name extracted from the requested domain name (e.g. *coralcdn.nyuld.net* indicates service *coralcdn*).

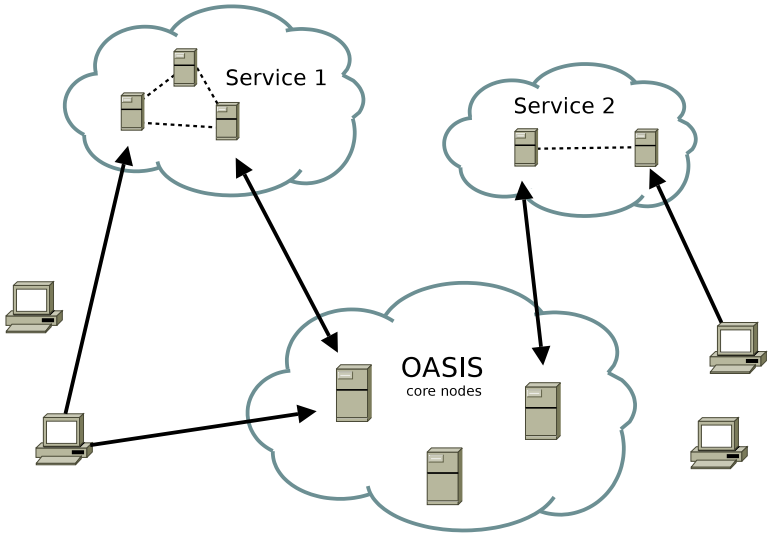


Fig. 2.7: OASIS system overview

2.5 BGP - Border Gateway Protocol

The Border Gateway Protocol (BGP), specifically BGP-4, is defined in RFC 4271[21]. In this section are some citations taken from [11] and [33]. BGP provides loop-free inter-domain routing between autonomous systems. An autonomous system(AS) is a set of routers that operate under the same administration and routing policy. BGP is often used within the networks of Internet service providers (ISP). BGP is an exterior routing protocol(EGP) which use a path-vector routing protocol.

Routers that belong to the same AS and exchange BGP updates are said to be running internal BGP (IBGP), and routers that belong to different ASs and exchange BGP updates are said to be running external BGP (EBGP). Figure 2.8 shows a network that demonstrates the difference between EBGP and IBGP.

Note that this section covers just necessary information about BGP to understand anycast and is explained on a cisco router. The rest is beyond the scope of this document.

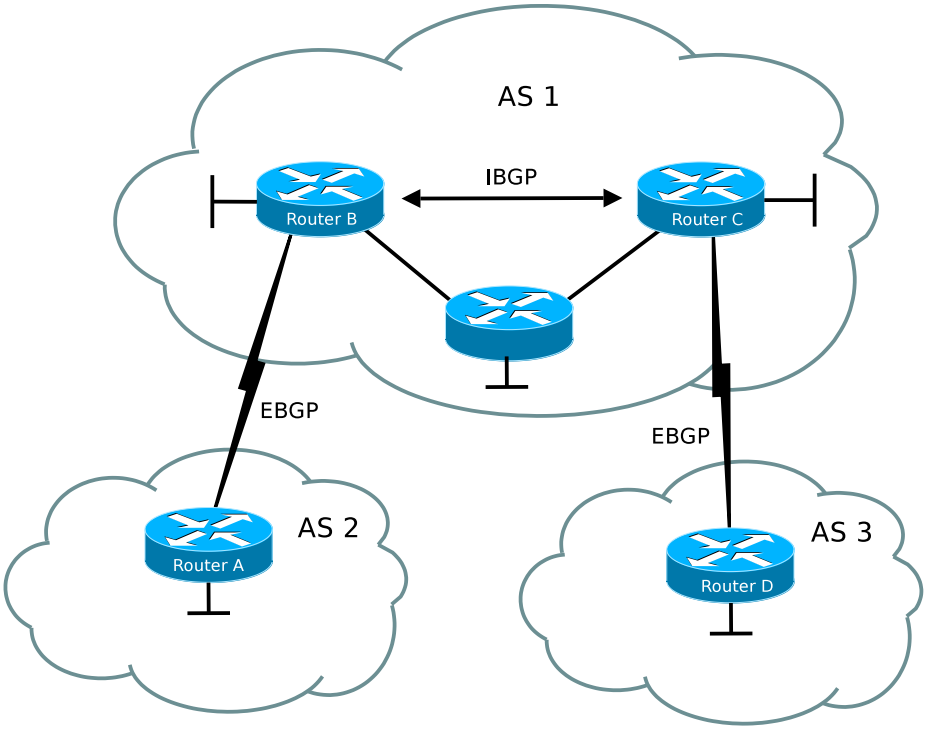


Fig. 2.8: EBGP, IBGP, and Multiple ASs

Before the routing system exchanges information with an external AS, BGP ensures that networks within the AS are reachable. This is done by a combination of internal BGP peering among routers within the AS and by redistributing BGP routing information to Interior Gateway Protocols (IGP) that run within the AS,

such as Open Shortest Path First (OSPF), Intermediate System-to-Intermediate System (IS-IS) and Routing Information Protocol (RIP).

BGP uses the Transmission Control Protocol (TCP) as its transport protocol (specifically port 179). Any two routers that have opened a TCP connection to each other for the purpose of exchanging routing information are known as peers or neighbours. In figure 2.8, routers A and B are BGP peers, as are routers B and C, and routers C and D. The routing information consists of a series of AS numbers that describe the full path to the destination network. BGP uses this information to construct a loop-free map of ASs. Note that within an AS, BGP peers do not have to be directly connected.

BGP peers initially exchange their full BGP routing tables when the TCP connection between peers is first established. When changes to the routing table are detected, the BGP routers send to their peers only those routes that have changed. BGP routers do not send periodic routing updates, and BGP routing updates advertise only the optimal path to a destination network.

2.5.1 BGP Attributes

Routes learned via BGP have associated properties that are used to determine the best route to a destination when multiple paths exist to a particular destination. These properties are referred to as BGP attributes, and an understanding of how BGP attributes influence route selection is required for the design of robust networks. This section describes the attributes that BGP uses in the route selection process:

- Weight
- Local Preference(LOCAL_PREF)
- Multi-exit discriminator (MULTI_EXIT_DISC)
- ORIGIN
- AS_PATH
- NEXT_HOP
- Community

Weight Attribute

Weight is a Cisco-defined attribute that is local to a router. The weight attribute is not advertised to peering routers. If the router learns about more than one route to the same destination, the route with the highest weight will be preferred.

Local Preference

Local Preference(LOCAL_PREF) shall be included in all UPDATE messages that a given BGP speaker sends to other internal peers. A BGP speaker shall calculate the degree of preference for each external route based on the locally-configured policy, and include the degree of preference when advertising a route to its internal peers. The higher degree of preference must be preferred. A BGP speaker uses the degree of preference learned via LOCAL_PREF in its Decision Process.

Multi-exit discriminator

The Multi-exit discriminator(MULTI_EXIT_DISC) is intended to be used on external (inter-AS) links to discriminate among multiple exit or entry points to the same neighbouring AS.

ORIGIN

ORIGIN specifies the origin of the routing update. When BGP has multiple routes, it uses the ORIGIN as one factor in determining the preferred route. It specifies one of the following origins:

- **IGP** — The route is interior to the originating AS. This value is set when the network router configuration command is used to inject the route into BGP.
- **EGP** — The route is learned via the Exterior Border Gateway Protocol (EBGP).
- **Incomplete** — The origin of the route is unknown or learned in some other way. An origin of incomplete occurs when a route is redistributed into BGP.

AS_PATH

This attribute identifies the autonomous systems through which routing information carried in BGP UPDATE message has passed.

NEXT_HOP

The NEXT_HOP is an attribute that defines the IP address of the router that should be used as the next hop to the destinations listed in the UPDATE message.

Community Attribute

Community attribute is an extension of BGP-4 protocol[8]. Community attribute provides a way of grouping destinations, called communities, to which routing decisions (such as acceptance, preference, and redistribution) can be applied. Predefined community attributes are:

- **no-export** — Do not advertise this route to EBGP peers.
- **no-advertise** — Do not advertise this route to any peer.
- **internet** — Advertise this route to the Internet community; all routers in the network belong to it.

Figure 2.9 shows the no-export community. AS 1 advertises 172.16.1.0 to AS 2 with the community attribute **no-export**. AS 2 will propagate the route throughout AS 2 but will not send this route to AS 3 or any other external AS. This way is configured a Local Node in IP anycast as described in section 2.4.5.

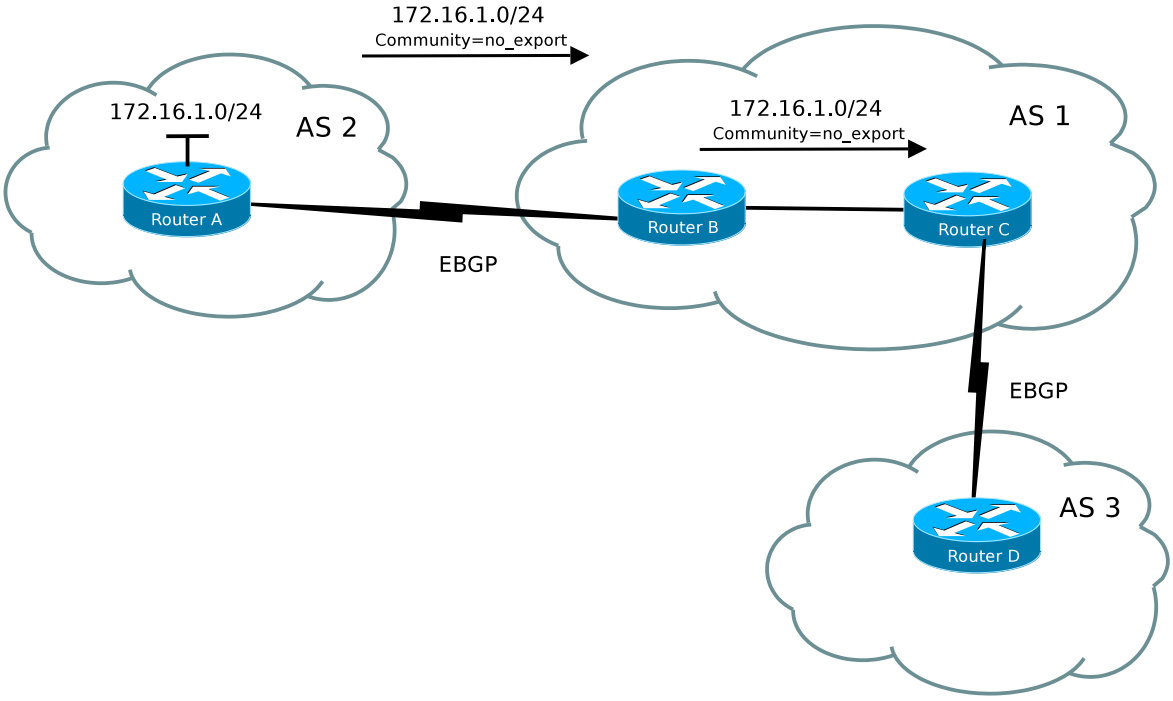


Fig. 2.9: BGP no-export Community Attribute

2.5.2 BGP Path Selection

BGP could possibly receive multiple advertisements for the same route from multiple sources. BGP selects only one path as the best path. When the path is selected,

BGP puts the selected path in the IP routing table and propagates the path to its peers. BGP uses the following criteria, in the order presented, to select a path for a destination:

- If the path specifies a next hop that is inaccessible, drop the update.
- Prefer the path with the largest weight.
- If the weights are the same, prefer the path with the largest local preference.
- If the local preferences are the same, prefer the path that was originated by BGP running on this router.
- If no route was originated, prefer the route that has the shortest AS_path.
- If all paths have the same AS_path length, prefer the path with the lowest origin type (where IGP is lower than EGP, and EGP is lower than incomplete).
- If the origin codes are the same, prefer the path with the lowest MED attribute.
- If the paths have the same MED, prefer the external path over the internal path.
- If the paths are still the same, prefer the path through the closest IGP neighbour.
- Prefer the path with the lowest IP address, as specified by the BGP router ID.

2.5.3 BGP routing stability

BGP routing changes happen for a variety of reasons[23]. The exchange of update messages depends on having an active BGP session between a pair of routers. Device failures or reconfiguration may trigger the closing of the BGP session, forcing each router to withdraw the routes learned from its neighbour. After re-establishing the session, the routers exchange their routing information again. Each router applies local policies to select the best route for each prefix and to decide whether to advertise this route to the neighbour. Changes in these policies can trigger new advertisements. A group of ASs may have conflicting policies that lead to repeated advertising and withdrawing of routes. In addition, intra-domain routing or topology changes may cause some routers to select new BGP routes and advertise them to neighbouring ASs. BGP routing changes can cause performance problems. A single event, such as a link failure, can trigger a long sequence of updates as the routers explore alternate paths. During this convergence period, the packets headed

toward the destination prefix may be caught in forwarding loops. Exchanging and processing the update messages also consumes bandwidth and CPU resources on the BGP speaking routers in the network. In addition, the new advertisements from neighbouring ASs may change the paths that traffic takes through the network. This can cause congestion on certain links in the AS. Frequent changes in the advertisements from other domains make it difficult for operators to engineer the flow of traffic through an AS. For example, a BGP routing change may cause traffic to a particular destination prefix to leave the AS through a different egress point. If BGP routing changes affect a large portion of the traffic, past information about BGP updates would not be a good basis for future operations decisions.

3 SOLUTION SPACE

In this chapter, we are reviewing several anycast-based methods for discovery of an RTP proxy server. The key objective of all the method is to avoid use of RTP proxy servers that are too distant from a call party. In the first section we set several evaluation criteria. In the next section we suggest four different methods to solve the proximity problem. We conclude with a comparison of all the methods based on the criteria set in previous sections.

3.1 Evaluation Criteria

Easy of Integration

This means what must be done about to make a proposed method working and how difficult is to deploy it.

"Proxy" Effect

A "middlebox" between client and anycast server that leads to the middlebox being used for discovery instead of the client.

SIP Interoperability

It is a behaviour and cooperation of all SIP entities in the SIP communication. If one of the SIP entity does not support a required feature it is not possible to use the feature. An anycast solution must be as SIP interoperable as possible and should not break any standards and policy issues.

Resilience against Routing Instabilities

It is important to keep the system resilient against changes in routing. That means that the fronting elements must be as stateless as possible. I.e., the elements shall minimize its transport-layer and application-layer context to either stateless or at least short-lived transactions.

Integration Overhead

If RTP proxies are not co-located with SIP proxies then they must be controlled by SIP proxy servers remotely. This introduces additional concerns: latency and security. This particularly applies to methods that concentrate SIP servers in a single place, from which multiple geographically-dispersed RTP proxy servers are controlled. RTP control is a specific source of complexity.

Failure-reactiveness

Time for a SIP client to switch over if a fronting-element fails.

3.1.1 Network Constraints

When designing the "fronting element", the following constraints have to be kept in mind:

- For the anycast proximity service to take effect, the anycast service(which is not the RTP proxy itself!) must be co-located with the RTP proxy.
- Scope shall be easily extended to one-way RTP servers such as SEMS(SIP Express Media Server).
- Anycast can unlikely deliver the best proximity but importantly it avoids reliably the worst-case.
- NAT traversal has to be accomplished by fronting element's SIP proxy unless the fronting-element is completely SIP-unaware. (see the IP-tunnel-based method later)

3.2 Anycast-based Methods for Finding the Closest RTP Servers

We are suggesting several proxy-discover anycast-based methods, that differ in how they are integrated in the whole system.

- Anycasting geographically spread DNS servers
- Anycasting SIP Proxy Servers
- Anycasting SIP tunnels
- Anycast "bootstrap" redirect service

In the following sections will be described upsides and downsides of mentioned methods in architectural detail and made comparison among them.

3.3 Anycasting Geographically Spread DNS Servers

The DNS-based method, used also similar way in OASIS[13] features application-independence and high resilience against routing instabilities. It relies on geographically dispersed DNS servers responsible for the serving domain. The IP address returned by the DNS server is used to associate client with that particular region and to find the appropriate RTP proxy. In this scenario, the RTP proxy and SIP proxy(fronting element) are co-located with this DNS server. The RTP proxy and SIP proxy servers listen on unicast address, whereas the DNS server uses anycast. Clients before sending SIP messages perform a DNS lookup which is usually handled by provider's DNS resolver. DNS resolver recursively finds the closest anycast DNS server which returns unicast IP address and used for sending SIP messages. After SIP client sends SIP messages to resolved unicast IP address and anycast IP address does not play any other role(until DNS re-lookup given by TTL parameter). The path of these SIP messages must be remembered for use by subsequent SIP traffic. Otherwise, it could hit a different fronting element without appropriate TCP context, or with a different IP address that would not be accepted by symmetric NATs.

The scenario and the call flow are shown in 3.1 and 3.2.

3.3.1 Call Flows

This call flow shows how fronting elements(SIP proxies) are involved in the SIP traffic. For simplicity there are no SIP user authorizations included.

DNS Lookup

1. Before UA1 sends any request it performs a DNS lookup. The lookup is done usually by provider's DNS resolver that recursively finds the closest anycast DNS server.

2. This anycast DNS server returns a unicast IP address particular for the region. Actually, this unicast IP address is associated with SIP fronting element and RTP proxy where is also residing the DNS server.

REGISTER

UA1 sends REGISTER message to unicast IP address of the nearest SIP proxy #1(fronting element). The SIP proxy #1 changes the Contact header field to

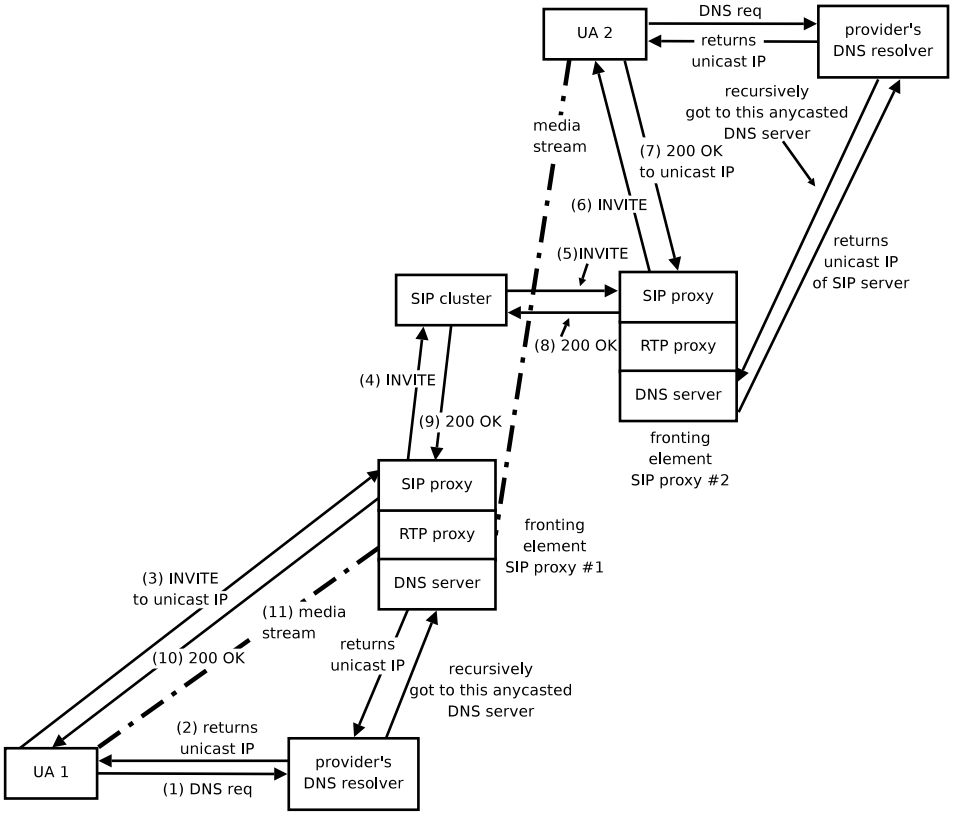


Fig. 3.1: Anycasting DNS servers

remember the path of the message. SIP proxy #1 sends the request to SIP cluster which stores the Contact of UA1 in location DB. For details about path processing see Section 4.3 - *remember the path*.

INVITE

3. UA1 wants to setup a call with UA2. UA1 sends INVITE to unicast IP address of the SIP proxy #1.

4. SIP proxy #1 changes the Contact header - *remember the path*(see 4.3) and sends it to SIP cluster. The message is also record routed to stay in the same path for BYE requests. If the message came from behind it does NAT traversal procedure as described in Section 2.3 and mark the message that the NAT traversal was done here. RTP binding is allocated and SDP changed.

5. SIP cluster look up Contact of UA2 and replace the Request URI and send to the host part of the Request URI which is SIP proxy #2.

6. As SIP proxy #2 receives INVITE it parses the Request URI, uses the information stored in it and sends the INVITE to UA2.

7. UA2 replies 200 OK towards SIP proxy #2 based on top most Via header of INVITE.

8. SIP proxy #2 changes the Contact header in the 200 OK reply(*remember the path*) and sends it to SIP cluster.

9. SIP cluster forwards it to SIP proxy #1.

10. SIP proxy #1 changes the Contact header of 200 OK and sends it to UA1.

11. A media flows between UA1 and UA2 based on INVITEs and 200OKs SDP bodies.

BYE

12. Similarly as with INVITE the processing is done with BYE.

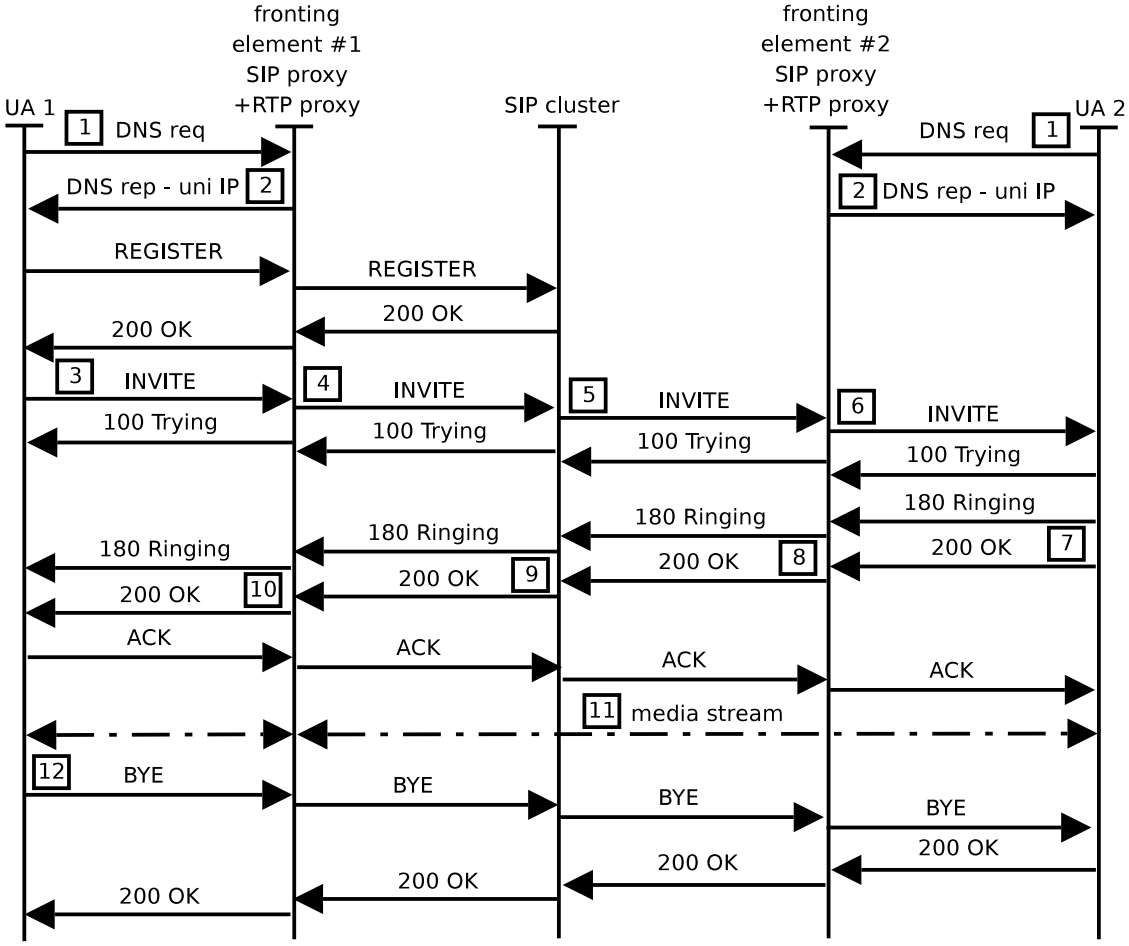


Fig. 3.2: Anycasting DNS servers - call flow

3.4 Anycasting SIP Proxy Servers

In this alternative, it is the SIP proxy server in the fronting element that listen on anycast address for discovery purposes. RTP proxies are co-located with each anycast SIP proxy. The SIP proxy does NAT traversal handling and also Contact mangling to remember the path for future requests. The SIP proxy shall stay as stateless as possible to guarantee minimum impact of routing instabilities. This is however not entirely possibly. On the transport layer, use of TCP breaks this requirement. On the SIP layer, the proxy can be stateless. Importantly, all the anycast SIP proxy servers must produce the same transaction id (branch Via parameter) otherwise down-stream SIP cluster will not match requests belonging to the same transactions during routing instabilities.

3.4.1 Call Flows

REGISTER

1. UAs register with SIP cluster through their closest SIP proxy server. Once a SIP proxy receives REGISTER message it fixes Contact header (as described in Path processing section 4.3) by adding unicast IP address of the proxy to the host field of SIP URI and encoding source IP address of received message into the Contact header.

2. The SIP proxy forwards the message to SIP cluster where it saves the fixed Contact in location DB and returns a response 200 OK to the UA1 back through the SIP proxy server.

INVITE

3. UA1 sends INVITE to SIP proxy anycast address. As the message is received the SIP proxy #1 fixes the Contact header(Path processing). If the request comes from behind NAT the SIP proxy mangles SIP message. RTP binding is allocated and SDP changed as described in section NAT traversal 2.3. The request is also record routed and forwarded to SIP cluster but from unicast address of that SIP proxy.

4. Once the INVITE is received at SIP cluster it looks up contact of UA2 in location DB, replace the Request URI and forwards the message to unicast IP address of SIP proxy #2.

5. As SIP proxy #2 receives INVITE it strips down the Request URI and forwards the message to UA2 from anycast IP address.

6. UA2 replies with 200 OK and sends the message back to SIP proxy #2 . If the reply comes from UA behind NAT then applies appropriate NAT mangling to this reply.

7. SIP cluster forwards message based on top most Via header to SIP proxy #1 and finally SIP proxy #1 towards UA1 from anycast address as source IP address. SIP proxy #1 might apply any NAT traversal mangling if the UA2 was marked as behind NAT.

BYE

8. BYE requests are processed similar way as INVITEs. As earlier was introduced record routing this transaction will go through the same path as it was record-routed by INVITE transaction. (UA1->proxy-1->SIP cluster->proxy-2->UA2)

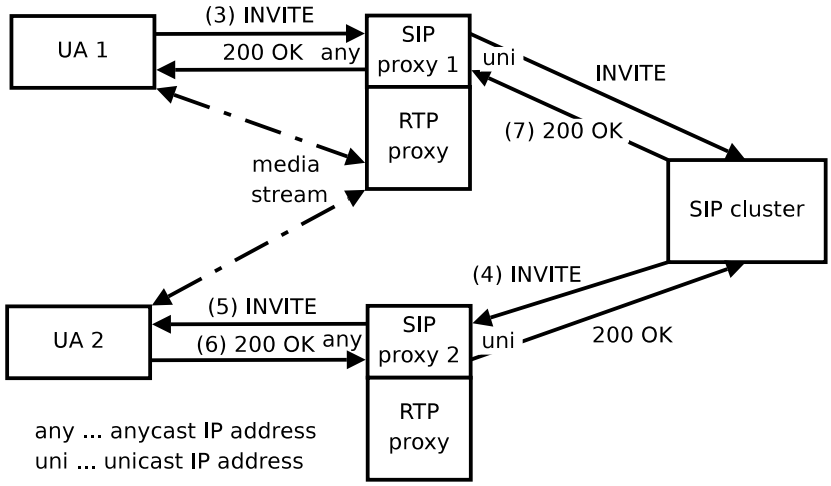


Fig. 3.3: Anycasting SIP proxy scenario

3.4.2 TCP Persistent Connection Issue

The key problematic part of anycasting SIP proxy servers is routing instability issues. In case a SIP client uses TCP transport for sending SIP messages it needs to create a TCP connection with a SIP proxy server. The connection must be kept persistent because of reachability of this client. In case a re-routing occurs the connection is lost and TCP ACK is not able to reach the SIP proxy. Figure 3.5 shows the impact.

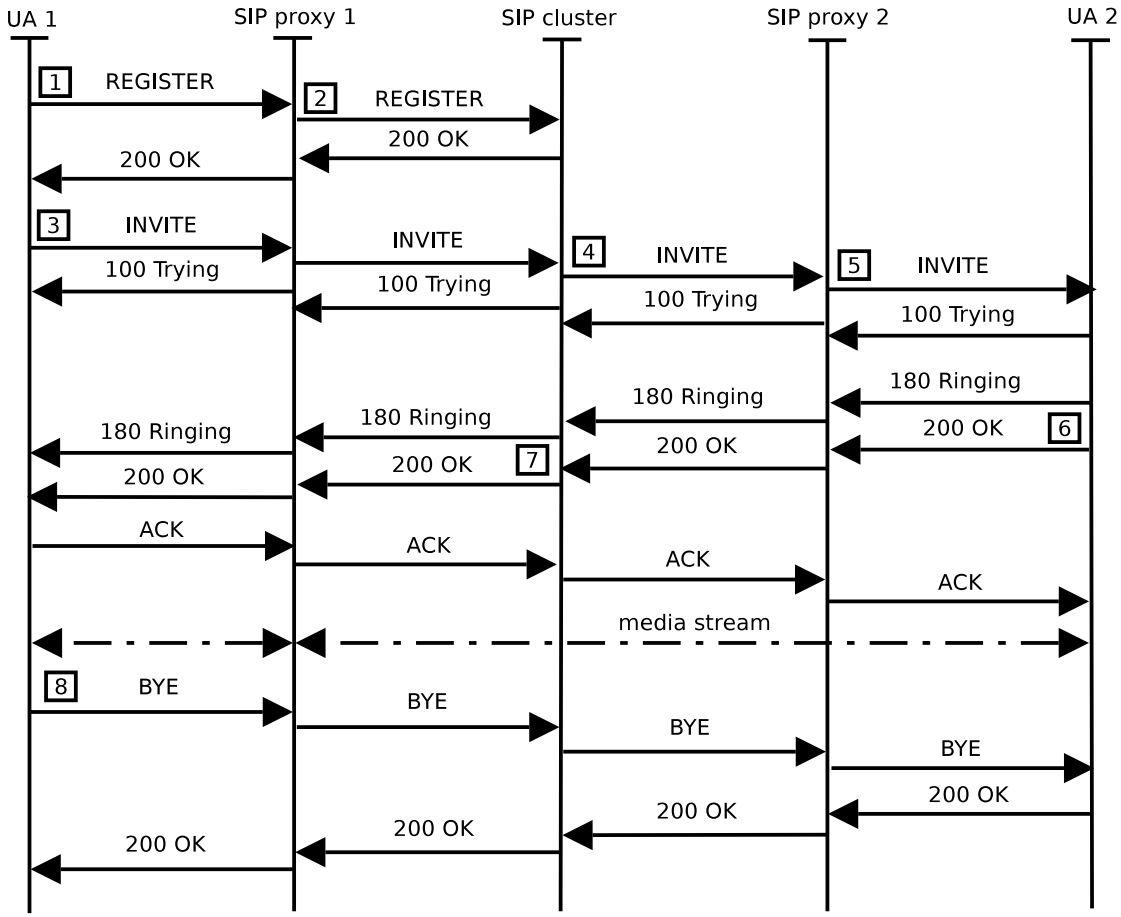


Fig. 3.4: Anycasting SIP proxy call flow

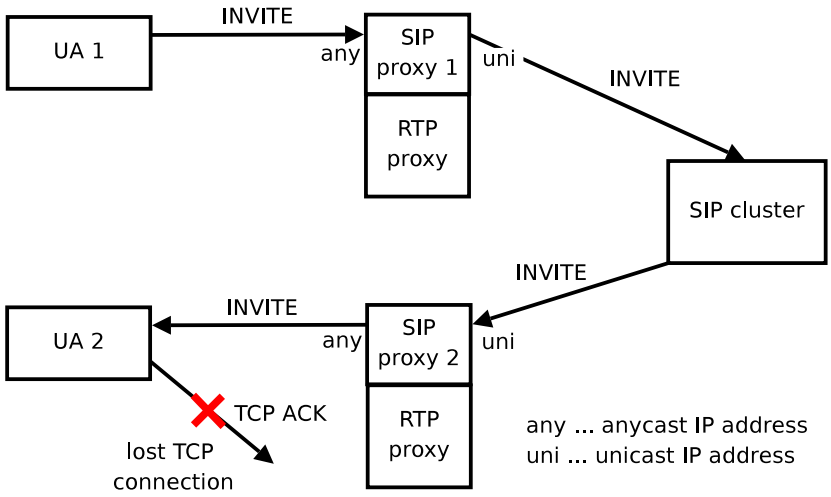


Fig. 3.5: Broken TCP persistency

3.5 Anycasting SIP Tunnels

This concept is based on IP tunnels. There will be more tunnel entrances at geographically dispersed anycast nodes with RTP proxies listening on unicast IP address. SIP messages are tunneled through a tunnel to SIP proxy where the tunnel is terminated which produces IP packets as if they came directly from a UA. SIP proxy runs on anycast address too. Packets before entering the tunnel are somehow marked(ToS field) with the ID of RTP proxy running at this end-point and encapsulated to this tunnel. Packets get de-capsulated at the end of the tunnel and the mark of is used for matching against RTP proxy list with their unicast IP addresses and used for further NAT traversal processing.

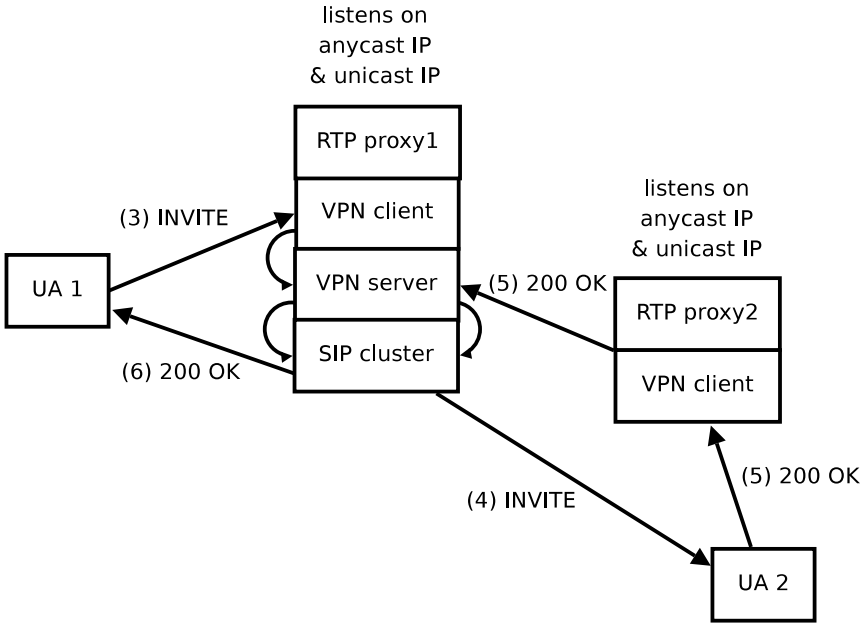


Fig. 3.6: Anycasting SIP tunnels

3.5.1 Call Flows

REGISTER

1. UA1 sends REGISTER to anycast IP address that gets forwarded to SIP cluster via tunnel.

2. SIP server replies with 200 OK directly to UA1.

INVITE

3. UA1 sends INVITE to anycast IP address. A VPN tunnel is listening on this address that forwards the INVITE to the SIP cluster. The tunneled packets are marked with an ID of RTP proxy running on that node. At the end of tunnel (VPN server) the INVITE gets de-capsulated and delivered to SIP cluster listening also at anycast IP address. Packet marking is used in the cluster to identify which RTP proxy to control.

4. SIP cluster record-routes the INVITE and sends it directly to UA2.

5. UA2 replies with 200 OK. The message can go through different anycast tunnel but always gets delivered to the same SIP cluster where the call was initiated.

6. SIP cluster matches the transaction and replace IP addresses in SDP body with unicast IP address of RTP proxy #1 as marked at the beginning of transaction and sends to UA1 directly.

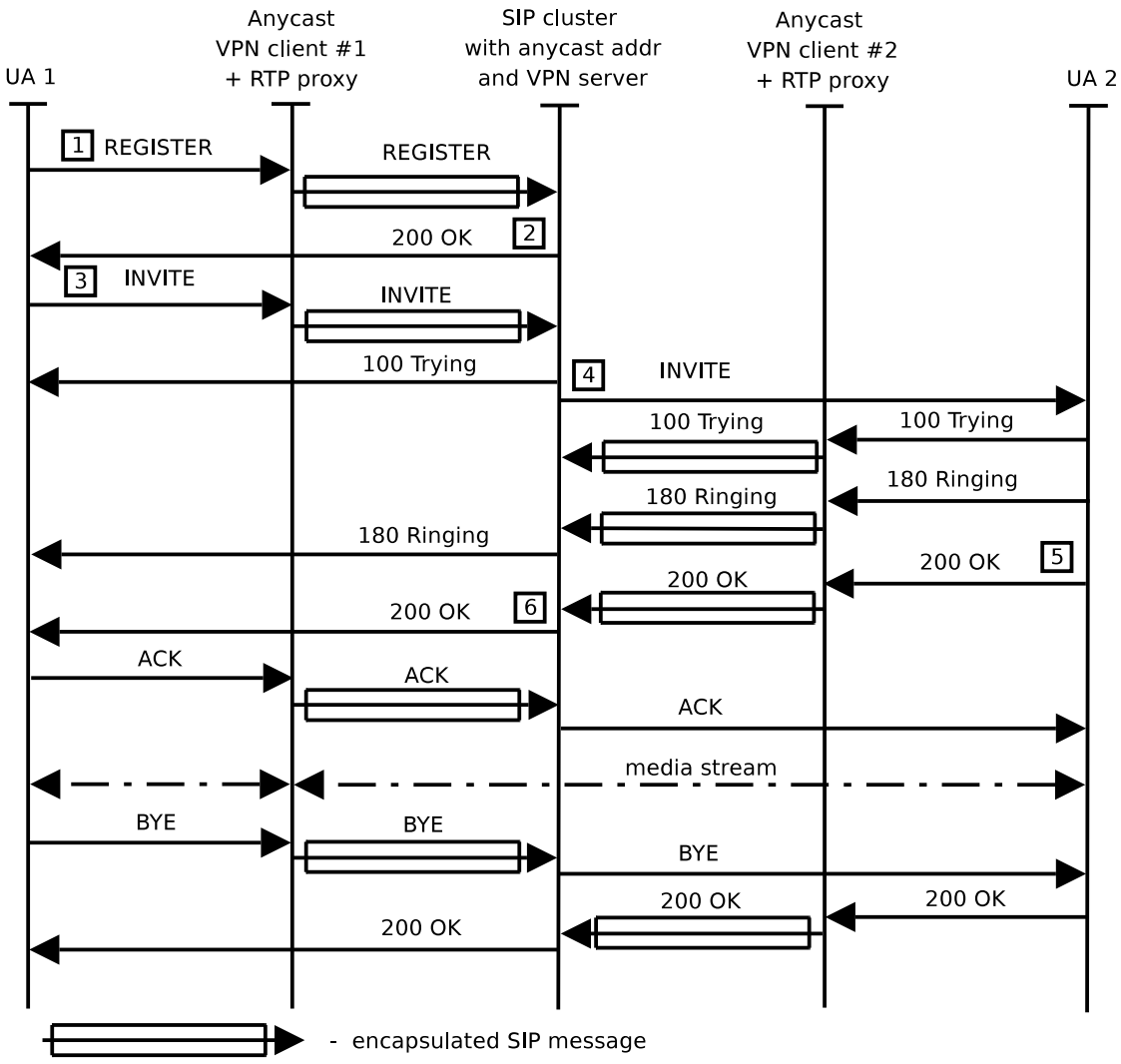


Fig. 3.7: Anycasting SIP tunnels - call flow

3.6 Anycast "bootstrap" Redirect Service

This concept is based on selecting RTP proxy during SIP redirection. The SIP cluster redirects an initial INVITE to anycast SIP proxy with co-located RTP proxy. This proxy redirects the INVITE back to the cluster with location information in it. The SIP cluster uses the location information to steer the proper RTP proxy and passes the request on. Note that RTP proxies are controlled remotely from SIP cluster that causes additional complexity and call setup delay.

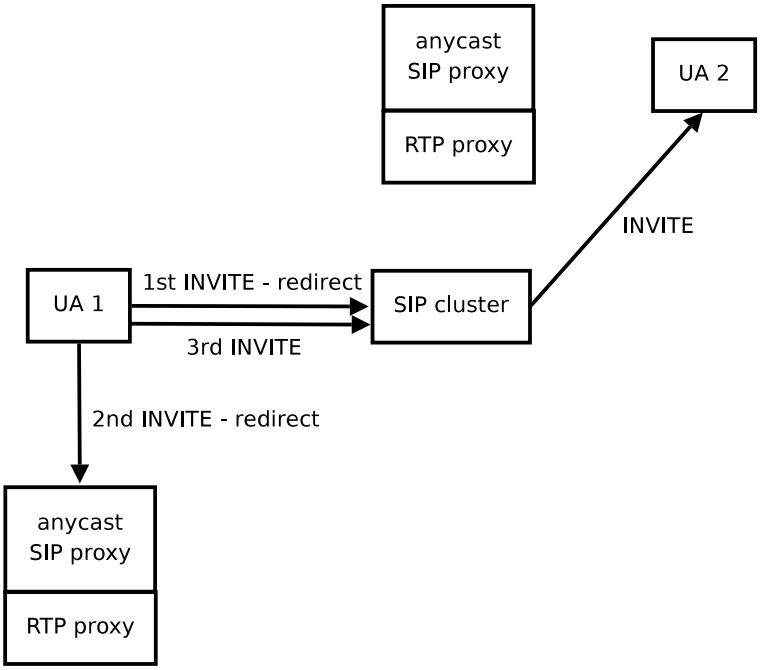


Fig. 3.8: Anycast "bootstrap" redirect service

3.6.1 Call Flows

REGISTER

There is no change to common SIP setup with REGISTER requests.

1. UA1 sends REGISTER to unicast IP address of SIP cluster where the UA1s Contact is saved in location DB. In case the UA is behind NAT the SIP cluster also save that the UA is behind NAT.

INVITE

2. UA1 sends INVITE to SIP cluster that checks if UA1 is behind NAT. If so, then it checks if combinations Client-IP, RTPproxy IP is in cache(location DB

in memory), if the cache is empty then SIP cluster redirects to Anycast SIP proxy.

3. Anycast SIP proxy receives INVITE and redirects it back to SIP cluster. The redirection URI in Contact HF stores a URI parameter with unicast IP address of co-located RTP proxy.

4. UA1 sends INVITE to SIP cluster with parameter including IP address of RTPproxy as URI param in Contact, SIP cluster parse the parameter and store in local cache Client-IP, RTPproxy IP.

5. SIP cluster use the IP address in Contact for selecting RTP proxy and subsequently used for relaying media. The message is record-routed to stay in path for BYEs and forwards it to UA2.

6. UA2 replies with 200 OK, the message reaches the SIP cluster and the INVITE transaction is matched with the IP address of RTP proxy used when the call was initiated.

BYE

7. UA2 sends BYE towards SIP cluster that tests if the UA is behind NAT. It should unforce RTP proxy based on earlier selection RTP proxy.

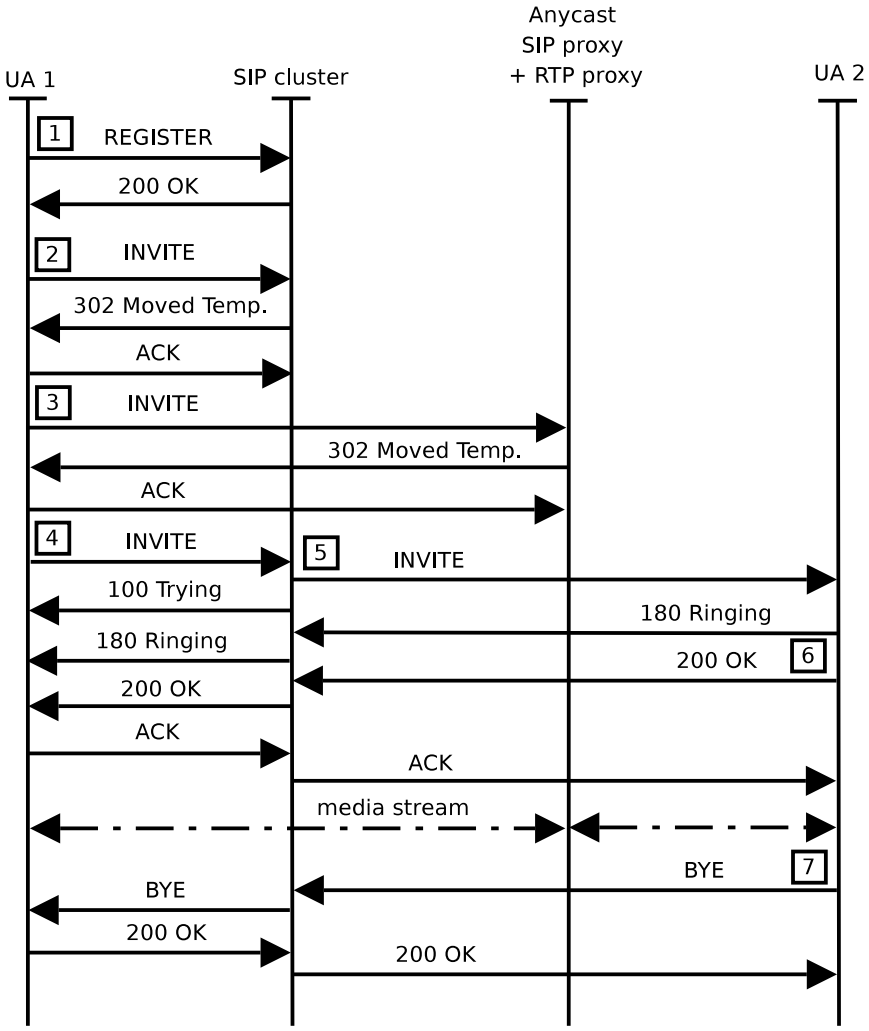


Fig. 3.9: Anycast “bootstrap” redirect service call flow

3.7 Evaluation of Methods

Anycasting DNS Servers

Upside: This method is simple to integrate at the system level and SIP messages are simply sent to unicast IP address returned from DNS lookup. It is resilient against routing instabilities as the anycast traffic is limited to a short-lived UDP-based DNS transaction. RTP proxy servers are co-located with SIP proxy servers and no remote control is needed.

Downside: Failure reactiveness is low for practical reasons. DNS resolvers in SIP clients and DNS proxy servers are known to cache DNS information for quite long time. If an anycast site fails and stops advertising its route, poor DNS clients will keep using an unavailable IP address. Also, the proximity measurement may be impaired if a client uses a DNS resolver that is not located in its proximity.

Anycasting SIP Proxy Servers

Upside: RTP proxy servers are co-located with SIP proxy servers and no remote control is needed.

Downside: Most sensitive against routing instabilities 3.1.

Anycast "bootstrap" Redirect Server

Upside: Easy to integrate at SIP level only. Resilient against routing instability.

Downside: INVITE-redirect brings too high uncertainty due to possible interoperability and policy issues – to many SIP clients are known not to support redirection due to poor implementation or for policy reasons (3xx to +1-900- ban). Call setup latency increases. Controlling remotely RTP proxies that cause additional complexity and call setup delay.

Anycast SIP Tunnels

Upside: Resilient against routing stability issues. No need to do anything at SIP level.

Downside: Dealing with remote RTP proxy servers.

3.8 Summary and Comparison of Methods

Method	Anycasting DNS servers	Anycasting SIP proxy servers	Anycast "bootstrap" redirect server	Anycast SIP tunnels
Easy of integration	Path processing and NAT handling	Path processing and NAT handling	Remote RTP proxy control	Remote RTP proxy control and Diffserv processing
"proxy" effect	measuring DNS resolvers	measuring the outbound SIP proxy	measuring the outbound SIP proxy	measuring the outbound SIP proxy
Anticipated interop level	no problems (anycast only on DNS)	no problems (all managed on server side)	UA needs to have enabled and functional redirect support, policy issues	
Resilience against routing instabilities	good, routing changes have no impact on DNS processing.	problematic, using anycast IP addresses for SIP signalling	good – boot-strap transaction is short-lived	good – it does not matter which IP tunnel is used
Failure reactiveness	can be low with mis-implemented DNS clients and DNS proxy servers	depends on how fast BGP re-routing is	depends on how fast BGP re-routing is	depends on how fast BGP re-routing is

Tab. 3.1: Comparison of methods using IP anycast to find the nearest RTP proxy

3.9 Conclusion about Methods

We have chosen the DNS-based method and SIP-based method for further observations. The primary reason is they are simple to deploy. They do not require sophisticated integration (as would be the case with tunneling, marking and remote RTP control) and are not going to suffer from interoperability issues (as the SIP bootstrap method would). The key remaining concerns are low failure reactiveness for the DNS-based method and low resilience against routing instabilities for the SIP-based method. In the long-term, it may be beneficial to include the tunnel-based method in future observations. Overcoming the integration effort can be rewarded by both good failure reactiveness and resilience against routing instabilities.

4 DESIGN OF THE FRONTING ELEMENT

4.1 DNS-based Fronting Element

Each fronting element consists of DNS server, SIP proxy and RTP proxy. The DNS server listens on anycast IP address for DNS queries and in response returns unicast IP address of co-located SIP proxy and RTP proxy. The returned unicast IP address is always from the closest DNS server in routing metrics(the way anycast works). This way SIP client forwards SIP messages to returned unicast IP address where the RTP proxy is co-located.

The SIP proxy should remain as stateless as possible. For TCP-based traffic and traffic from behind NAT it must remain stateful however. This SIP proxy does NAT traversal and uses co-located RTP proxy if necessary. If SIP traffic is TCP-based the SIP proxy must use the same TCP connection initiated by SIP client's REGISTER request. We talk about TCP context which is described in Section 4.2.2 and which is more significant for SIP-based anycast proxy. We need to remember the path through this SIP proxy as described in Section 4.3. The same applies for traffic coming from behind NAT. Also, if the SIP proxy remains stateful the SIP traffic must be record routed.

4.1.1 SIP call flow in detail for DNS-based method

Firstly, SIP client performs DNS lookup. It receives a reply from DNS resolver with unicast IP address of the closest fronting SIP proxy. For simplicity there are no SIP authorizations and auxiliary replies included. SIP messages include just important header fields for explanation the process. This call flow shows how path processing is done(for more details see Section 4.3). Figure 4.1 shows possible scenario which is followed by detailed SIP message description.

1. The UA1 constructs an INVITE message and sends it to the unicast IP address of fronting SIP proxy. The SIP client is behind NAT as can be seen private IP addresses appearing in the message.

```
Direction        147.229.214.225:5090() -> 213.192.59.77:5060()

INVITE sip:UA2@siptel.org SIP/2.0
Via: SIP/2.0/UDP 192.168.1.100:5090;rport;branch=z9hG4bKdxrbqiko
To: <sip:UA2@siptel.org>
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
Contact: <sip:UA1@192.168.1.100:5090>
```

2. The fronting SIP proxy receives INVITE and checks if the message came from behind NAT. Basing on this check the SIP proxy creates new Contact SIP URI

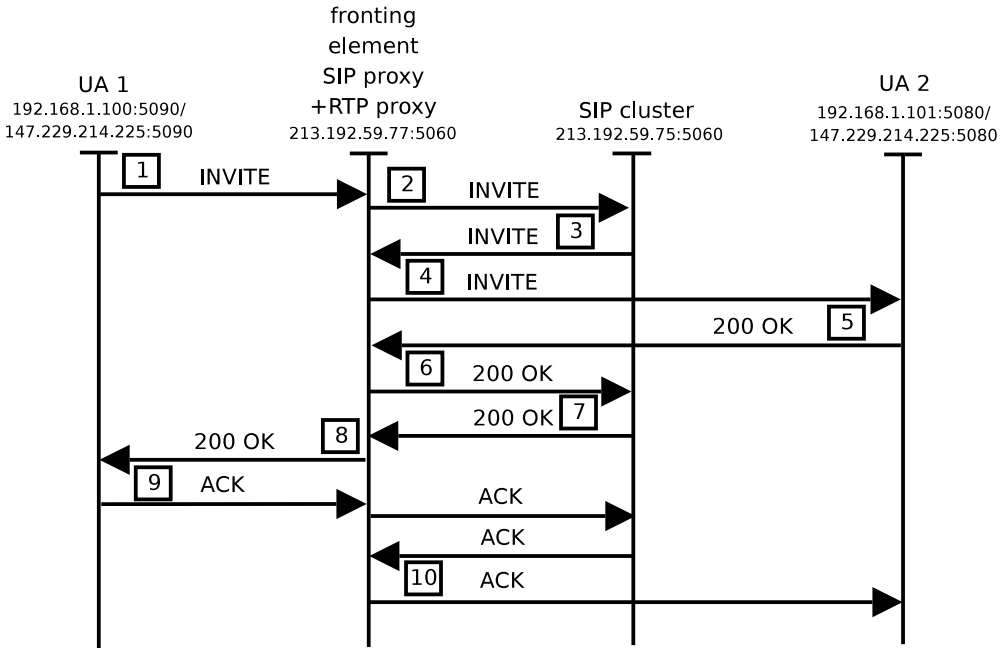


Fig. 4.1: DNS-based scenario - referential call flow

where in the username part includes, except the username, a mark that the SIP client is behind NAT, source IP address and port of the packet, private IP address and port of the SIP client. Host part includes the unicast IP address of this fronting element. It is the address where replies will be expected. Once this Contact header is mangled, then SIP proxy record route, marks the message that RTP proxy used at this server the message and sends it to SIP cluster.

```
Direction        213.192.59.77:5060() -> 213.192.59.75:5060()

INVITE sip:UA2@siptel.org SIP/2.0
Record-Route: <sip:213.192.59.77;lr=on>
Via: SIP/2.0/UDP 213.192.59.77;branch=z9hG4bK93ad.f50d8a83.0
Via: SIP/2.0/UDP 192.168.1.100:5090;received=147.229.214.225;
    rport=5090;branch=z9hG4bKdxrbqiko
To: <sip:UA2@siptel.org>
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
Contact: <sip:_NAT_*UA1**192.168.1.100*5090**147.229.214.225*5090*@
    213.192.59.77>
P-Behind-NAT: Yes
P-RTP-Proxy: YES
```

3. Once the SIP cluster receives INVITE it looks up the contact of UA2 in location database. This contact inserts in Request URI header. SIP cluster also finds that fronting SIP proxy already applied RTP proxy which says that no other RTP proxy should be used. INVITE is sent to IP address stated in Request URI. In this case it is the same fronting SIP proxy as for UA1.

```
Direction       213.192.59.75:5060() -> 213.192.59.77:5060()

INVITE sip:_NAT_*UA2**192.168.1.101*5080**147.229.214.225*5080*@
213.192.59.77 SIP/2.0
Record-Route: <sip:213.192.59.75;lr=on>
Record-Route: <sip:213.192.59.77;lr=on>
Via: SIP/2.0/UDP 213.192.59.75;branch=z9hG4bK63ad.1a75a823.0
Via: SIP/2.0/UDP 213.192.59.77;rport=5060;branch=z9hG4bK63ad.0130cb05.0
Via: SIP/2.0/UDP 192.168.1.100:5090;received=147.229.214.225;
    rport=5090;branch=z9hG4bKpcbtblbi
To: <sip:UA2@siptel.org>
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
Contact: <sip:_NAT_*UA1**192.168.1.100*5090**147.229.214.225*5090*@
213.192.59.77>
P-Behind-NAT: Yes
P-RTP-Proxy: YES
```

4. As fronting SIP proxy receives the message it decodes the Request URI header. From decoded information uses the public IP address of UA2 and sends it there.

```
Direction       213.192.59.77:5060() -> 147.229.214.225:5080()

INVITE sip:UA2@192.168.1.101:5080 SIP/2.0
Record-Route: <sip:213.192.59.77;lr=on>
Record-Route: <sip:213.192.59.75;lr=on>
Record-Route: <sip:213.192.59.77;lr=on>
Via: SIP/2.0/UDP 213.192.59.77;branch=z9hG4bK63ad.1130cb05.0
Via: SIP/2.0/UDP 213.192.59.75;rport=5060;branch=z9hG4bK63ad.1a75a823.0
Via: SIP/2.0/UDP 213.192.59.77;rport=5060;branch=z9hG4bK63ad.0130cb05.0
Via: SIP/2.0/UDP 192.168.1.100:5090;received=147.229.214.225
     rport=5090;branch=z9hG4bKpcbtblbi
To: <sip:UA2@siptel.org>
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
Contact: <sip:_NAT_*UA1**192.168.1.100*5090**147.229.214.225*5090*@
213.192.59.77>
P-RTP-Proxy: YES
```

5. Once UA2 receives INVITE, the UA2 inserts its own Contact header to 200 OK reply and sends it back to fronting SIP proxy.

```
Direction       147.229.214.225:5080() -> 213.192.59.77:5060()

SIP/2.0 200 OK
Via: SIP/2.0/UDP 213.192.59.77;branch=z9hG4bK63ad.1130cb05.0
Via: SIP/2.0/UDP 213.192.59.75;rport=5060;branch=z9hG4bK63ad.1a75a823.0
Via: SIP/2.0/UDP 213.192.59.77;rport=5060;branch=z9hG4bK63ad.0130cb05.0
Via: SIP/2.0/UDP 192.168.1.100:5090;received=147.229.214.225;
    rport=5090;branch=z9hG4bKpcbtblbi
Record-Route: <sip:213.192.59.77;lr=on>,<sip:213.192.59.75;lr=on>,
    <sip:213.192.59.77;lr=on>
To: <sip:UA2@siptel.org>;tag=shfxe
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
Contact: <sip:UA2@192.168.1.101:5080>
```

6. Fronting SIP proxy encode the Contact header the same way as for INVITE above and sends it to SIP cluster.

```
Direction       213.192.59.77:5060() -> 213.192.59.75:5060()

SIP/2.0 200 OK
Via: SIP/2.0/UDP 213.192.59.75;rport=5060;branch=z9hG4bK63ad.1a75a823.0
Via: SIP/2.0/UDP 213.192.59.77;rport=5060;branch=z9hG4bK63ad.0130cb05.0
Via: SIP/2.0/UDP 192.168.1.100:5090;received=147.229.214.225;
    rport=5090;branch=z9hG4bKpcbtblbi
Record-Route: <sip:213.192.59.77;lr=on>,<sip:213.192.59.75;lr=on>,
    <sip:213.192.59.77;lr=on>
To: <sip:UA2@siptel.org>;tag=shfxe
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
Contact: <sip:_NAT_*UA2**192.168.1.101*5080**147.229.214.225*5080*@
    213.192.59.77>
```

7. SIP cluster simply forwards the message back to fronting SIP proxy as based on Via header.

```
Direction      213.192.59.75:5060() -> 213.192.59.77:5060()

SIP/2.0 200 OK
Via: SIP/2.0/UDP 213.192.59.77;rport=5060;branch=z9hG4bK63ad.0130cb05.0
Via: SIP/2.0/UDP 192.168.1.100:5090;received=147.229.214.225;
    rport=5090;branch=z9hG4bKpcbtblbi
Record-Route: <sip:213.192.59.77;lr=on>,<sip:213.192.59.75;lr=on>,
    <sip:213.192.59.77;lr=on>
To: <sip:UA2@siptel.org>;tag=shfxe
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
Contact: <sip:_NAT_*UA2**192.168.1.101*5080**147.229.214.225*5080*@
    213.192.59.77>
```

8. Fronting SIP proxy forwards the reply to UA1.

```
Direction       213.192.59.77:5060() -> 147.229.214.225:5090()

SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.100:5090;received=147.229.214.225;
    rport=5090;branch=z9hG4bKpcbtblbi
Record-Route: <sip:213.192.59.77;lr=on>,<sip:213.192.59.75;lr=on>,
    <sip:213.192.59.77;lr=on>
To: <sip:UA2@siptel.org>;tag=shfxe
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
Contact: <sip:_NAT_*UA2**192.168.1.101*5080**147.229.214.225*5080*@
    213.192.59.77>
P-RTP-Proxy: YES
```

9. UA1 sends ACK towards UA2 through all SIP proxies stated in Route header. In Request URI is the full path UA2 SIP URI.

```
Direction       147.229.214.225:5090() -> 213.192.59.77:5060()

ACK sip:_NAT_*UA2**192.168.1.101*5080**147.229.214.225*5080*@
    213.192.59.77 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.100:5090;rport;branch=z9hG4bKhyynxpop
Route: <sip:213.192.59.77;lr=on>,<sip:213.192.59.75;lr=on>,
    <sip:213.192.59.77;lr=on>
To: <sip:UA2@siptel.org>;tag=shfxe
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
```

10. The SIP cluster is skipped and this ACK is produced by fronting SIP proxy. It shows that the Request URI is decoded and from decoded information uses the public IP address and port of UA2 and sends it there.

```
Direction       213.192.59.77:5060() -> 147.229.214.225:5080()

ACK sip:UA2@192.168.1.101:5080 SIP/2.0
Record-Route: <sip:213.192.59.75;lr=on>
Via: SIP/2.0/UDP 213.192.59.77;branch=0
Via: SIP/2.0/UDP 213.192.59.75;rport=5060;branch=0
Via: SIP/2.0/UDP 213.192.59.77;rport=5060;branch=0
Via: SIP/2.0/UDP 192.168.1.100:5090;received=147.229.214.225;
    rport=5090;branch=z9hG4bKhyynxpop
To: <sip:UA2@siptel.org>;tag=shfxe
From: "UA1 siptel" <sip:UA1@siptel.org>;tag=etazs
```

4.1.2 Technical Issues with DNS-based Method

DNS-based method is resilient against re-routing which is a feature we need. However, the best SIP proxy selection in SIP client proximity is dependent on periodic DNS lookups. Firstly, it depends on ISP's DNS resolver which usually caches DNS records and the TTL value is usually not low enough for refreshing the unicast IP address of the closest SIP proxy in case of BGP re-routing. Secondly, DNS clients are frequently mis-implemented and do DNS lookup just at boot time. Figure 4.2 shows the process.

This method results in possibly not the best SIP proxy selection but at least it avoids selecting very far SIP proxies. In case of failure, the service might be unavailable until new DNS lookup which might take some time.

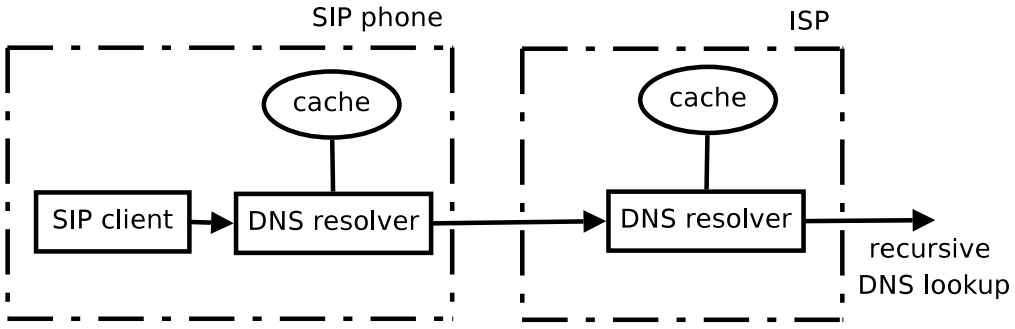


Fig. 4.2: DNS lookup

4.2 SIP-based Fronting Element

Here is described the SIP-based method with SIP proxy listening on anycast IP address as part of fronting element. The path processing(see 4.3) is the same as for

DNS-based method but the fronting SIP proxy deals with two IP addresses(anycast and unicast IP address). This method is more error-prone against routing instability issues and SIP messages must be handled carefully. For the best result, the fronting element must be *SIP-wise* as stateless as possible to guarantee minimum impact of routing instabilities. The proxy remains stateful TCP-wise but it may and actually should remain a SIP-wise stateless machine(UDP-based traffic less affected). For this purpose a path through this proxy must be remembered because each SIP client might register through different anycast SIP proxy. This way we also loose transparency of the SIP traffic because we deal with multiple anycast SIP proxies.

4.2.1 INVITE and CANCEL/ACK

The CANCEL request, as the name implies, is used for cancelling a previous request sent by a client. A CANCEL request should be only used for pending calls as stated in RFC3261 [24] section 9.1. Further, if a CANCEL request is sent it is part of the INVITE transaction and Via header of this CANCEL is matched 4.2.1 against INVITEs top most Via header at the first SIP proxy which is in our case the fronting SIP proxy. In case of routing instability, cancelling pending calls might cause following issue: CANCEL is forwarded through a different fronting SIP proxy than the initial INVITE and is not recognized as related by the downstream SIP cluster. Then the SIP cluster replies “481 Transaction leg does not exist”. In this case for UDP based SIP traffic the branch parameter should be generated statelessly as described in 4.2.1. However, we can not match this transaction. Sent-by value is used as part of the matching process because there could be accidental or malicious duplication of branch parameters from different clients. The reason is that the transaction id is formed by branch and sent-by, where sent-by is different for both fronting-elements (if it was identical using anycast, we would loose guarantee that replies to requests will get back to the same SIP proxy). This is a failure scenario where we can not do nothing about.

Matching Requests to Server Transactions

When a request is received from the network by the stateful proxy, it is matched to an existing transaction. This is accomplished in the following manner.

The branch parameter in the topmost Via header field of the request is examined. If it is present and begins with the magic cookie ”z9hG4bK”, the request was generated by a client transaction compliant to this specification. Therefore, the branch parameter will be unique across all transactions sent by that client. The request matches a transaction if:

1. the branch parameter in the request is equal to the one in the top Via header field of the request that created the transaction, and

2. the sent-by value in the top Via of the request is equal to the one in the request that created the transaction, and

3. the method of the request matches the one that created the transaction, except for ACK, where the method of the request that created the transaction is INVITE.

This matching rule applies to both INVITE and non-INVITE transactions alike. All above is taken from RFC3261, Section 17.2.3. See more in [24].

Stateless Generating Branch Parameter

In case of re-routing in time window between sending INVITE and CANCEL/ACK we need to be able to deliver CANCEL/ACK without breaking the SIP transaction. Fronting elements must generate branch parameter the same way for all messages to match transactions stateless anycast environment. For instance by inserting a fixed string in branch parameter of the Via header inserted by this fronting element.

4.2.2 Technical Issues with SIP-based Fronting Element

TCP context

RFC3261[24] says: "For reliable transports, the response is normally sent on the connection on which the request was received. Therefore, the client transport MUST be prepared to receive the response on the same connection used to send the request."

The first time, the TCP connection is opened by REGISTERing of a SIP client. For the subsequent SIP traffic the SIP client and SIP proxy must use the same TCP connection. If a routing instability occurs this connection will be lost. This means that SIP proxy should remain SIP-wise as stateless as possible that we can deal with UDP traffic and not break the consistency of SIP dialogs.

SIP client may open a new connection with a different anycast SIP proxy in case of re-routing. In case of opening a new connection in the middle of dialog, SIP messages will get lost because transactions would not match.

TCP call flow - REGISTER

1. First UA1 open a new TCP connection with fronting SIP proxy.

2. Then the UA1 sends REGISTER over TCP connection to fronting SIP proxy which forwards it through UDP transport to SIP cluster.

3. SIP cluster sends reply back through fronting SIP proxy. This proxy forwards the reply through existing TCP connection.

Once a SIP client registers with SIP cluster the same TCP connection must be used for upcoming SIP transactions.

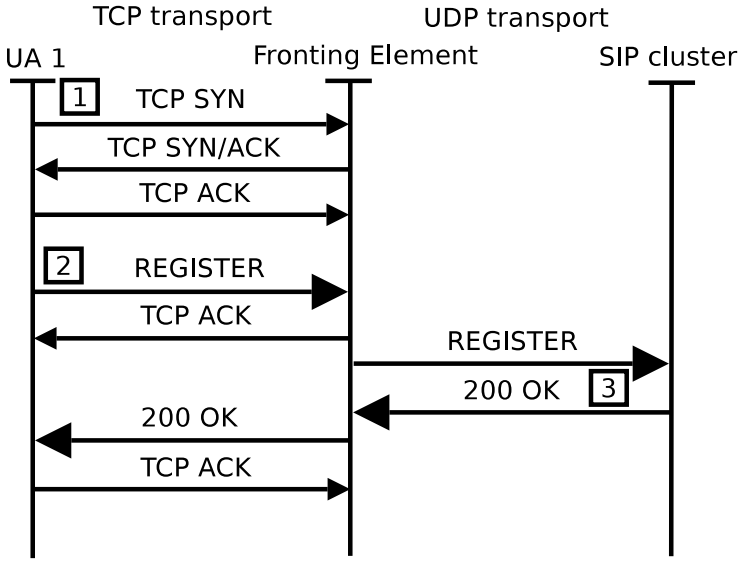


Fig. 4.3: REGISTERing over TCP

Transaction Issues

Another problem is that the failure window for the SIP protocol is fairly large. For example, if a client registers via anycasted SIP proxy, there may be up to one hour (default re-registration period in RFC3261) until an incoming message comes in. If in this period re-routing happens (which is not entirely unlikely since the time window is REALLY long), failures may occur. TCP connection will fail for sure, as the SIP proxy if it occurs to be stateful. The TCP issue may be improved by forcing keep-alives to detect issues early (which has to be done due to NATs anyhow), and relying clients to re-register. Even for UDP, there may be one minute between INVITE and CANCEL/ACK, still fairly long time window.

4.3 PATH Processing

This subsection describes ways how to remember path of SIP messages. We need to guarantee that incoming requests for a SIP client will go through the same SIP server

through which the SIP client registered. The reason is for SIP clients behind NAT and for clients using TCP transport for their communication. If we do not remember the path, replies may be destined to the client with different source IP address and port and the delivery fails. There are SIP standards for path remembering. However, the standards require support in end-devices, implementation of which is still quite rare. See the following subsections to find out how these extensions work. The last paragraph of this section "Proprietary path remembering" mentions an alternative solution which does not require compliant clients at the price of possible message integrity violation.

Path Extension(RFC3327)

This RFC3327 standard describes an extension for remembering the path of SIP requests. The Path extension header field allows accumulating and transmitting the list of proxies between UA1 and REGISTRAR. Intermediate nodes such as SIP proxy#1 (see Figure 4.4)may statefully retain Path information if needed by operational policy. This mechanism is in many ways similar to the operation of Record-Route in dialog-initiating requests. The routing established by the Path header field mechanism applies only to requests transiting or originating in the home domain. However, this approach must be implemented at both sides. SIP client and SIP proxy. For more details about Path extension see [31].

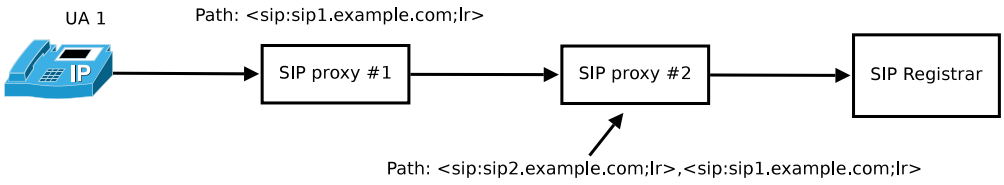


Fig. 4.4: Path extension header field

Service Route Extension

The "Service-Route" is a SIP extension header field (RFC 3608[32]), which can contain a route vector that will direct requests through a specific sequence of proxies. A registrar uses a Service-Route header field to inform a SIP client of a service route that, if used by the SIP client, will provide services from a proxy or set of proxies associated with that registrar. The Service-Route header field is included by a registrar in the response to a REGISTER request.

Then SIP clients include a Route header field in an initial request to force that request to visit and potentially be serviced by one or more proxies. Using such a

route (called a "service route" or "preloaded route") allows a SIP client to request services from a specific home proxy or network of proxies.

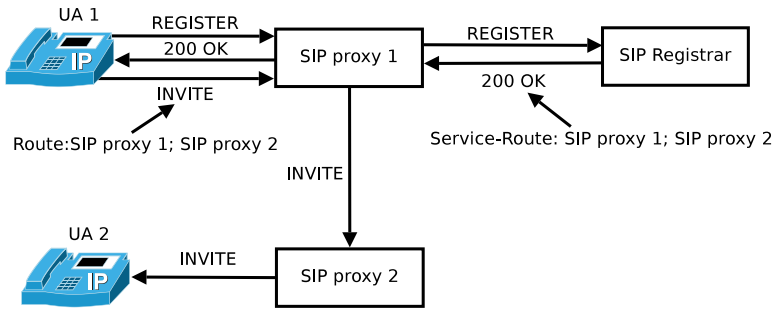


Fig. 4.5: Service-route header field(simplified)

This approach must also be implemented at both sides. SIP client and SIP registrar which is not we are looking for because of interoperability issues. In our anycast scenarios this would work as a kind of bootstrap solution(using firstly anycast address and then unicast address).

Proprietary Path Remembering

This proprietary solution of path remembering involves Contact mangling on fronting element. The new Contact header must include the information where the SIP message came from and the unicast IP address of this fronting element. All this information is encoded as new Contact SIP URI. Finally, the message is forwarded to SIP cluster 1.1.2. At SIP cluster this format of Contact is stored in user location DB. The following example and Figure 4.6 shows the way of path remembering. The downside of this approach is that modification of the Contact header-field conflicts with possible use of Message Integrity Check in RFC 4474[20]. Mostly, these MICs as described in the RFC 4474 are not implemented nowadays.

The format of Contact before REGISTER enters fronting element(see 3.3 for the scenario).

```
Contact: "Test" <sip:test@192.168.1.100:5060>;transport=udp"
```

The format of Contact when REGISTER is leaving fronting element where the address field of SIP URI is 213.192.59.76 the unicast IP address of fronting element, and the username field consists 147.229.214.225:5090 public IP address and port where the request came from and 192.168.1.100:5090 is private IP address and port of the SIP client behind NAT. The "_NAT_" mark is for requests that came from behind NAT. It is used for recognition once the Contact is decoded.

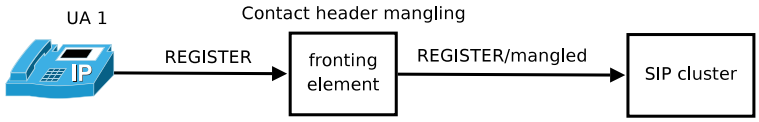


Fig. 4.6: Proprietary Contact mangling

```
Contact: "Test" <sip:_NAT_*test**192.168.1.100*5090**147.229.214.225*5090*@
    213.192.59.76>
```

4.4 Implementation Details

For deploying our own anycast network we requested RIPE(Regional Internet Registry) for assigning IP address block 91.199.168.1/24 and AS(autonomous system) number - AS44592. We installed two anycast nodes, in Prague and Berlin. These nodes are fronting elements as mentioned in Chapter 4. They are forwarding all SIP traffic to iptel.org's SIP cluster. These nodes have assigned two unicast IP addresses and the shared anycast IP address. At each node is running DNS server(named) listening on anycast IP address and two instances of SER(SIP Express Router) for testing both DNS-based and SIP-based methods. One SER is configured to handle unicast-way SIP traffic which is bound with DNS server. In Appendix B.1 is SER configuration for DNS-based scenario. The latter listens on anycast and the other unicast IP address for incoming and outgoing SIP traffic(see Appendix B.2 for the configuration).

SIP Express Router

SER (SIP Express Router) is a high-performance, configurable, free SIP server. It can act as SIP registrar, proxy or redirect server. SER is a modular based and features for example an application-server interface, presence support, SMS gateway, RADIUS/syslog accounting and authorization, server status monitoring, etc. SER's configuration script is very powerful tool parsing SIP messages at low level. SER's configuration ability meets the needs of a whole range of scenarios including small-office use, enterprise PBX replacements and carrier services. SER is being developed by a team at iptel.org based in Prague and Berlin. The developer's page can be found at <http://iptel.org>.

DNS server setup

In a DNS registrar(not important which one) we registered our DNS servers ns.siptel.org with IP address 91.199.168.1 and ns3.siptel.org with IP address 91.199.168.3. At the

both fronting elements are running DNS server which replies to requests sent to these anycast IP addresses. DNS servers returns unicast IP address depending on location of DNS resolver. It always chooses the closest DNS server in routing metrics. This way SIP client forwards SIP messages to returned unicast IP address where is also co-located RTP proxy.

BGP daemon

Each anycast node runs BGP daemon propagating 91.199.168.0/24 route to the upstream Internet Service Provider(ISP). From the ISP the route propagates further to the Internet.

5 CONCLUSION

Our anycast fronting elements in Prague and Berlin were tested and measured on ICMP echo reply basis from planet-lab hosts. Because the anycast locations are very close to each other the route convergence time was quite short. Re-routing from Prague to Berlin took for most of planet-lab hosts between 10 and 20 seconds. Interestingly, the other way from Berlin to Prague it took less than 10 seconds for most of planet-lab hosts. In other terms this would be an outage of the service if one of the anycast nodes fails. We also compared latency between shortest unicast IP address destination against anycast IP address destination. It shows that anycast does not provide the best proximity for SIP clients in 131 cases measured out of 195 planet-lab hosts. The reason is that our anycast nodes are very close to each other and the routing path from planet-lab hosts is not very different. We also found out that unicast and anycast routes are different even for the same physical destination. We can conclude and proof that anycast metrics are not latency metrics, as verified in our measurements but at least would eliminate the worst case scenarios in global deployment.

We proposed four IP anycast-based methods for locating an RTP proxy close to SIP clients. We decided to choose for further observations the DNS-based method and the SIP-based method because they were easy to deploy. DNS-based method is resilient against re-routing, however due to frequently mis-implemented DNS clients and proxy servers it can fail to react to changes timely and is subject to possible proximity impairment. SIP-based method suffers from low resilience against routing instabilities. These issues are covered in design chapter making our fronting elements as stateless as possible and remaining stateful for TCP-based SIP traffic and SIP traffic from SIP clients behind NAT.

We implemented DNS-based and SIP-based method at our anycast fronting elements using SIP Express Router. Our configuration worked and provided good proximity at coarse scale but not so much on finer scale. We did not thoroughly test it because of problematic configuration for routing in instability scenarios and lack of time for complete measurements. There are scenarios we are not able to solve such as SIP-based method using TCP transport because of loosing connection in instable scenario. We also found out that even for UDP is not easy to get smooth switch over to different SIP proxy because of matching transaction ID at SIP cluster constructed from branch and sent-by parameter at fronting elements. Sent-by is always different because it is unicast IP address.

Each method we have been proposing has some drawbacks (see comparison in Section 3.8). Also we had too few anycast nodes to validate the really important

coarse scenarios finding that anycast provides a good proximity.

5.1 Future Work

Proposed designs of methods mentioned in this thesis should be thoroughly tested, tuned and measured. There are still some withstanding UDP and more complex TCP issues especially for anycast SIP proxy servers and its behaviour in routing instability scenarios we are facing to and need to solve.

We have not described the design of the IP-tunnels based method but it is a good candidate for dealing with routing instability issues. There is no need to do anything at SIP level but on the other side we must deal with remote RTP proxy servers. This introduces additional concerns: latency and security. We need to do further analysis of using the remote RTP proxy control protocol.

We need to do field measurements with an established global SIP user basis. This will provide us with better view of locating RTP proxy for each method once we will have more globally dispersed anycast nodes. Prague and Berlin locations are only good as a functionality test but do not deliver a significant latency improvement. We need to test how the SIP anycast setup will behave in failure scenarios such as stopping the BGP route propagation and seeing what happens as the SIP traffic converges to another anycast node.

Another work which needs to be done is a geo-failover. This means we need SIP service with good latency and availability at different locations on the world. But we need to solve two problems. One of them are geo-distributed RTP proxy servers and the other one are geo-distributed SIP proxy servers. Anycast SIP proxy servers have this feature build-in. The other methods will need further inspection and testing.

We need to test against some live populations and seeing the actual latency savings. Also we need to remeasure with more better dispersed anycast nodes.

BIBLIOGRAPHY

[1] Abley J., Lindqvist K., *Operation of Anycast Services*, RFC 4786, December 2006 <http://www.ietf.org/rfc/rfc4786.txt>

[2] Abley J., *Hierarchical Anycast for Global Service Distribution*, ISC Technical Note ISC-TN-2003-1, <http://www.isc.org/tn/isc-tn-2003-1.html>

[3] Abley J., *A Software Approach to Distributing Requests for DNS Service using GNU Zebra, ISC BIND 9 and FreeBSD*, ISC Technical Note ISC-TN-2004-1, March 2004, <http://www.isc.org/pubs/tn/isc-tn-2004-1.html>

[4] Ballani H., Francis P., *Towards a Deployable IP Anycast Service*, Proc. of First Workshop on Real, Large Distributed Systems (WORLDS'04) San Francisco, California, Dec 2004.

[5] Ballani H., Francis P., Ratnasamy S., *A Measurement-based Deployment Proposal for IP Anycast*, Proc. of Internet Measurement Conference(IMC'06) Rio de Janeiro, Brazil, Oct 2006.

[6] Ballani H., Francis P., *Towards a global IP anycast service*, SIGCOMM, 2005.

[7] Bhattacharjee S., Ammar M. H., Zegura Viren Shah E. W., Fei Z., *Application-Layer Anycasting*, INFOCOM (3), pages 1388-1396, 1997.

[8] Chandra R., Traina P., *BGP Communities Attribute*, RFC1997, August 1996, <http://www.ietf.org/rfc/rfc1997.txt>

[9] Chen, Lim, Katz, Overton, *On the stability of nework distance estimation*, SIGMETRICS Perf. Eval. Rev., 2002

[10] Cisco^{TM} Distributed Director, <http://www.cisco.com>

[11] Doyle J., Carroll J. D., *Routing TCP/IP, Volume II (CCIE Professional Development)*, Publisher: Cisco Press, April 2001, 976 pages, ISBN 978-1-57870-089-9

[12] Engel R., Peris V., Saha D., Basturk E., Haas R., *Using IP Anycast For Load Distribution And Server Location*, In Proc. Third Global Internet Mini-Conference, November 1998.

[13] Freedman M., Lakshminarayanan K., Mazieres D., *OASIS: Anycast for Any Service*, In NSDI, 2006.

[14] Freedman M. J., Vutukuru M.,Feamster N., Balakrishnan H., *Geographic locality of IP prefixes*, In IMC, October 2005.

[15] Huston, G., *NOPEER Community for Border Gateway Protocol (BGP) Route Scope Control*, RFC 3765, April 2004.

[16] Levine M., Lyon B., Underwood T., *Operational experience with TCP and Anycast - Cachefly*, NANOG37, June 2006, <http://www.nanog.org/mtg-0606/pdf/matt.levine.pdf>.

[17] Mao Z., Govindan R., Varghese G., Katz R., *"Route Flap Dampening exacerbates Internet routing convergence"*, SIGCOMM'02, June 2002.

[18] Miller K., *Deploying IP Anycast*, Carnegie Mellon Network Group, NANOG 29, October 2003 <http://www.net.cmu.edu/pres/anycast/anycast.pdf>

[19] Partridge C., Mendez T., Milliken W., *Host Anycasting Service*, RFC 1546, November 1993. <http://www.ietf.org/rfc/rfc1546.txt>

[20] Peterson J., Jennings C., *Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)*, August 2006, <http://www.ietf.org/rfc/rfc4474.txt>

[21] Rekhter Y., Li T., Hares S., *A Border Gateway Protocol 4 (BGP-4)*, RFC 4271, January 2006, <http://www.ietf.org/rfc/rfc4271.txt>

[22] Rekhter Y., Moskowitz B., Karrenberg D., Groot G. J., Lear E., *Address Allocation for Private Internets*,BCP 5, RFC 1918, February 1996, <http://www.ietf.org/rfc/rfc1918.txt>

[23] Rexford J., Wang J., Xiao Z., Zhang Y., *BGP Routing Stability of Popular Destinations*, In IMW, Nov. 2002.

[24] Rosenberg J., Schulzrinne H., Camarillo G., Johnston A., Peterson J., Sparks R., Handley M. and E. Schooler, *SIP: Session Initiation Protocol,* RFC 3261, June 2002. <http://www.ietf.org/rfc/rfc3261.txt>.

[25] Rosenberg J., and Schulzrinne H., *An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing,* August 2003 <http://www.ietf.org/rfc/rfc3581.txt>.

[26] Rosenberg J., Mahy R., and Huitema C., *Traversal Using Relay NAT (TURN)*, Sseptember 2005, <http://www.tools.ietf.org/html/draft-rosenberg-midcom-turn-08>.

[27] Rosenberg J., *Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Multimedia Session Establishment Protocols*, October 2007, <http://tools.ietf.org/html/draft-ietf-mmusic-ice-19>.

[28] Schulzrinne H., Casner S., Frederick R., Jacobson V., *RTP: A Transport Protocol for Real-Time Applications,* RFC 1889, January 1996. <http://www.ietf.org/rfc/rfc1889.txt>.

[29] Yu S., Zhou W. , Wu Y., *Research on Network Anycast*, Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02), page 154, October 2002.

[30] Weinberger J., Huitema C., and Mahy R., *STUN - Simple Traversal of User Datagram Protocol (UDP)Through Network Address Translators(NATs),* March 2003 <http://www.ietf.org/rfc/rfc3489.txt>.

[31] Willis D., Hoeneisen B., *Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts*, December 2002, <http://www.ietf.org/rfc/rfc3327.txt>

[32] Willis D., Hoeneisen B., *Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration*, October 2003, <http://www.ietf.org/rfc/rfc3608.txt>

[33] *Cisco documentation - Border Gateway Protocol (BGP)*, October 2006, <http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/index.htm>

LIST OF APPENDICIES

A ANYCAST MEASUREMENTS

A.1 Latency of ICMP replies of Prague and Berlin Anycast Nodes

Prague anycast node

Hint:
PRG - Prague, TXL - Berlin
A2Uloc - difference of anycast/prague to unicast/prague
A2Usho - difference of anycast to shortest unicast ping
Match - if the shortest ping matches the anycast selection

Any [ms]	PRG [ms]	TXL [ms]	A2Uloc [ms]	A2Uloc [%]	A2Usho [ms]	A2Usho [%]	Select anycast	Match	Host
21	21	19	0.0	0.2	1.3	6.6	Prague	No	146-179.surfsnel.dsl.internl.net
114	113	110	1.1	1.0	4.8	4.4	Prague	No	75-130-96-12.static.oxfr.ma.charter.com
122	122	116	-0.3	-0.2	5.6	4.8	Prague	No	bob.cc.vt.edu
186	208	178	-21.5	-12.0	8.2	4.6	Prague	No	deimos.cecalc.ula.ve
114	109	108	4.7	4.3	5.8	5.4	Prague	No	earth.cs.brown.edu
308	307	297	1.0	0.3	10.8	3.6	Prague	No	eve.ee.ntu.edu.tw
190	190	182	0.1	0.0	8.6	4.7	Prague	No	grouse.hpl.hp.com
150	150	131	-0.3	-0.2	19.3	14.7	Prague	No	kc-sce-plab1.umkc.edu
159	141	131	18.1	13.7	27.9	21.1	Prague	No	kupl1.ittc.ku.edu
114	113	113	0.9	0.8	1.1	1.0	Prague	No	lefthand.eecs.harvard.edu
41	38	41	3.7	8.9	3.7	9.7	Prague	Yes	lsirextpc01.epfl.ch
197	198	180	-0.3	-0.2	17.5	9.7	Prague	No	node1.lbnl.nodes.planet-lab.org
168	169	156	-0.8	-0.5	12.0	7.7	Prague	No	node1.planetlab.uprr.pr
50	50	42	-0.0	-0.1	7.9	18.8	Prague	No	onelab3.warsaw.rd.tp.pl
109	109	101	0.2	0.2	7.9	7.8	Prague	No	orbpl1.rutgers.edu
39	40	45	-1.2	-2.7	-1.2	-3.0	Prague	Yes	peeramide.irisa.fr
109	112	125	-3.3	-2.6	-3.3	-2.9	Prague	Yes	pepper.planetlab.cs.umd.edu
57	63	69	-6.3	-9.0	-6.3	-9.9	Prague	Yes	pl1.grid.kiae.ru
294	294	281	0.7	0.2	13.2	4.7	Prague	No	pl1-higashi.ics.es.osaka-u.ac.jp
54	44	38	10.0	26.1	16.6	43.3	Prague	No	pl-1.hip.fi
292	292	281	-0.6	-0.2	10.8	3.8	Prague	No	pl1.planetlab.ics.tut.ac.jp
325	325	323	-0.3	-0.1	1.9	0.6	Prague	No	pl1snu.koren21.net
140	140	174	0.0	0.0	0.0	0.0	Prague	Yes	pl1.ucs.indiana.edu
16	16	39	0.0	0.1	0.0	0.3	Prague	Yes	plab1-c703.uibk.ac.at
161	172	133	-11.4	-8.5	27.3	20.4	Prague	No	plab1.eece.ksu.edu
185	184	168	0.9	0.5	17.3	10.3	Prague	No	plab1.engr.sjsu.edu
12	12	35	0.1	0.2	0.1	0.6	Prague	Yes	plab1-itec.uni-klu.ac.at
118	117	103	1.0	1.0	15.3	14.8	Prague	No	plab1.nec-labs.com
21	21	31	0.1	0.2	0.1	0.3	Prague	Yes	planck227.test.ibbt.be
122	122	116	0.1	0.1	6.3	5.4	Prague	No	planet02.csc.ncsu.edu
296	297	286	-1.4	-0.5	10.3	3.6	Prague	No	planet0.jaist.ac.jp
179	180	167	-1.0	-0.6	12.2	7.3	Prague	No	planet1.berkeley.intel-research.net
40	40	30	0.0	0.0	10.4	34.6	Prague	No	planet1.colbud.hu
114	115	104	-0.9	-0.8	10.2	9.8	Prague	No	planet1.cs.rochester.edu
180	180	173	-0.1	-0.0	6.7	3.9	Prague	No	planet1.cs.ucsb.edu
111	112	101	-1.5	-1.5	10.1	10.0	Prague	No	planet1.ecse.rpi.edu
107	106	95	0.5	0.5	11.7	12.2	Prague	No	planet1.scs.cs.nyu.edu
123	123	113	-0.2	-0.2	10.4	9.2	Prague	No	planet2.pittsburgh.intel-research.net
128	127	141	0.6	0.4	0.6	0.5	Prague	Yes	planet.cc.gt.atl.ga.us
174	179	165	-5.4	-3.2	8.9	5.4	Prague	No	planetdev01.fm.intel.com
110	110	99	-0.0	-0.0	11.0	11.1	Prague	No	planetlab-01.bu.edu
39	31	40	8.1	19.9	8.1	25.7	Prague	Yes	planetlab01.cnds.unibe.ch
185	182	172	2.8	1.6	12.3	7.1	Prague	No	planetlab01.cs.washington.edu
171	170	156	1.6	1.0	15.6	10.0	Prague	No	planetlab-01.ece.uprm.edu
32	32	26	-0.3	-1.0	6.3	24.0	Prague	No	planetlab01.ethz.ch
119	119	114	-0.5	-0.4	4.9	4.3	Prague	No	planetlab01.sys.virginia.edu
181	180	176	0.5	0.3	4.9	2.8	Prague	No	planetlab-1a.ics.uci.edu
7	7	30	-0.3	-1.0	-0.3	-4.1	Prague	Yes	planetlab1.ani.univie.ac.at
165	165	155	-0.2	-0.1	10.3	6.6	Prague	No	planetlab1.arizona-gigapop.net
1	1	21	-0.3	-1.3	-0.3	-19.5	Prague	Yes	planetlab1.cesnet.cz
48	48	32	0.1	0.4	16.3	50.3	Prague	No	planetlab1.ci.pwr.wroc.pl
121	121	121	0.4	0.4	0.4	0.4	Prague	Yes	planetlab1.cis.upenn.edu
162	163	141	-0.8	-0.6	21.1	14.9	Prague	No	planetlab1.citadel.edu
132	129	117	2.7	2.3	14.9	12.7	Prague	No	planetlab-1.cmcl.cs.cmu.edu
111	112	106	-0.8	-0.8	4.6	4.3	Prague	No	planetlab1.cnds.jhu.edu
110	111	113	-0.2	-0.2	-0.2	-0.2	Prague	Yes	planetlab1.csail.mit.edu
159	160	151	-1.3	-0.9	7.6	5.0	Prague	No	planetlab1.cs.colorado.edu
106	106	95	0.0	0.0	11.6	12.2	Prague	No	planetlab1.cs.columbia.edu
119	122	109	-2.5	-2.3	10.7	9.8	Prague	No	planetlab1.cs.cornell.edu
143	143	137	-0.3	-0.2	6.0	4.4	Prague	No	planetlab1.cse.msu.edu
146	146	181	0.0	0.0	0.0	0.0	Prague	Yes	planetlab1.cse.nd.edu
131	134	122	-2.5	-2.1	9.7	8.0	Prague	No	planetlab-1.cse.ohio-state.edu
32	32	26	-0.1	-0.3	5.7	21.5	Prague	No	planetlab1.csg.uzh.ch
82	82	71	-0.0	-0.0	10.3	14.4	Prague	No	planetlab1.cslab.ece.ntua.gr
140	140	170	-0.0	-0.0	-0.0	-0.0	Prague	Yes	planetlab1.cs.purdue.edu

153	153	148	-0.5	-0.3	4.4	2.9	Prague	No	planetlab1.csres.utexas.edu
180	180	175	-0.4	-0.2	5.0	2.9	Prague	No	planetlab1.cs.ucla.edu
181	181	175	-0.5	-0.3	6.1	3.5	Prague	No	planet-lab1.cs.ucr.edu
132	132	136	-0.6	-0.4	-0.6	-0.4	Prague	Yes	planetlab1.cs.uiuc.edu
112	110	98	2.2	2.2	13.8	14.0	Prague	No	planetlab1.cs.umass.edu
189	188	207	1.4	0.7	1.4	0.8	Prague	Yes	planetlab1.cs.uoregon.edu
15	15	14	-0.0	-0.3	1.4	9.6	Prague	No	planetlab1.cs.vu.nl
140	138	132	1.6	1.2	8.2	6.2	Prague	No	planetlab1.dtc.umn.edu
185	185	188	-0.6	-0.3	-0.6	-0.3	Prague	Yes	planetlab1.eas.asu.edu
187	187	175	-0.6	-0.4	12.0	6.8	Prague	No	planetlab1.ece.ucdavis.edu
130	130	131	0.0	0.0	0.0	0.0	Prague	Yes	planetlab1.eecs.northwestern.edu
157	157	154	-0.6	-0.4	2.5	1.6	Prague	No	planetlab1.eecs.ucf.edu
139	138	133	1.7	1.3	6.8	5.1	Prague	No	planetlab1.eecs.umich.edu
187	187	206	0.1	0.1	0.1	0.1	Prague	Yes	planetlab1.een.orst.edu
3691	3656	2756	35.3	1.3	935.4	33.9	Prague	No	planetlab1.eurecom.fr
62	62	60	-0.0	-0.1	2.5	4.2	Prague	No	planetlab1.fct.ualg.pt
289	278	274	10.5	3.8	15.5	5.7	Prague	No	planetlab-1.fing.edu.uy
4	4	25	-0.1	-0.3	-0.1	-1.5	Prague	Yes	planetlab1.fit.vutbr.cz
185	185	153	0.0	0.0	31.7	20.6	Prague	No	planetlab1.flux.utah.edu
45	45	37	-0.4	-1.0	8.4	22.7	Prague	No	planetlab1.fri.uni-lj.si
44	44	32	-0.1	-0.2	11.9	36.7	Prague	No	planetlab1.hiit.fi
311	314	320	-2.9	-0.9	-2.9	-0.9	Prague	Yes	planetlab1.icu.ac.kr
44	49	28	-4.4	-15.4	16.1	56.1	Prague	No	planetlab1.ifi.uio.no
282	282	270	0.5	0.2	12.4	4.6	Prague	No	planetlab1.iii.u-tokyo.ac.jp
299	301	292	-2.1	-0.7	6.9	2.4	Prague	No	planetlab1.iitb.ac.in
35	35	23	-0.0	-0.2	12.6	54.5	Prague	No	planetlab-1.imperial.ac.uk
32	32	26	0.0	0.0	6.6	25.1	Prague	No	planetlab1.inf.ethz.ch
57	57	55	0.2	0.4	2.1	3.8	Prague	No	planetlab-1.iscte.pt
112	112	108	-0.8	-0.7	3.1	2.9	Prague	No	planetlab1.isi.jhu.edu
53	53	44	0.0	0.0	8.8	19.7	Prague	No	planetlab1.it.uc3m.es
43	38	26	4.6	17.0	16.6	61.7	Prague	No	planetlab-1.it.uu.se
45	45	51	0.1	0.2	0.1	0.2	Prague	Yes	planetlab1lannion.elibel.tm.fr
269	269	245	0.0	0.0	24.1	9.8	Prague	No	planetlab1.larc.usp.br
53	53	48	-0.1	-0.3	4.9	10.1	Prague	No	planetlab1.ls.fi.upm.es
34	45	30	-10.5	-35.0	4.7	15.6	Prague	No	planetlab-1.man.poznan.pl
185	186	174	-0.8	-0.5	11.7	6.7	Prague	No	planetlab1.millennium.berkeley.edu
358	357	335	1.2	0.3	23.3	7.0	Prague	No	planetlab1.netmedia.gist.ac.kr
35	35	22	-0.3	-1.2	12.8	56.0	Prague	No	planetlab1.nrl.dcs.qmul.ac.uk
320	309	296	10.8	3.7	24.2	8.2	Prague	No	planetlab1.ntu.nodes.planet-lab.org
107	107	95	0.4	0.4	12.2	12.8	Prague	No	planetlab1.poly.edu
357	299	209	58.0	27.8	148.7	71.1	Prague	No	planetlab1.pop-mg.rnp.br
191	191	201	-0.1	-0.1	-0.1	-0.1	Prague	Yes	planetlab1.postel.org
72	72	48	0.0	0.0	23.9	49.3	Prague	No	planetlab1.science.unitn.it
57	58	56	-0.1	-0.2	1.7	3.1	Prague	No	planetlab-1.tagus.ist.utl.pt
72	71	62	0.6	0.9	9.2	14.6	Prague	No	planetlab1.tlm.unavarra.es
40	40	30	-0.0	-0.0	10.7	35.2	Prague	No	planetlab1.tmit.bme.hu
34	34	43	0.1	0.2	0.1	0.3	Prague	Yes	planetlab-1.tssg.org
403	413	396	-10.4	-2.6	6.8	1.7	Prague	No	planetlab1.ucb-dsl.nodes.planet-lab.org
135	138	129	-2.5	-2.0	6.7	5.2	Prague	No	planetlab1.uc.edu
183	183	178	0.0	0.0	4.9	2.7	Prague	No	planetlab1.ucsd.edu
276	273	204	2.7	1.3	72.1	35.3	Prague	No	planet-lab1.ufabc.edu.br
39	39	40	-0.1	-0.2	-0.1	-0.2	Prague	Yes	planetlab1.unineuchatel.ch
154	165	152	-11.3	-7.4	2.0	1.3	Prague	No	planetlab-1.unk.edu
67	67	62	-0.0	-0.1	4.9	7.9	Prague	No	planetlab1.upc.es
151	151	141	0.3	0.2	9.7	6.8	Prague	No	planetlab1.uta.edu
139	140	137	-0.6	-0.4	2.6	1.9	Prague	No	planetlab-1.vuse.vanderbilt.edu
110	111	101	-0.5	-0.5	9.4	9.3	Prague	No	planetlab1.williams.edu
41	41	28	-0.0	-0.2	13.1	46.5	Prague	No	planetlab1.xeno.cl.cam.ac.uk
29	30	18	-0.3	-1.5	11.7	64.7	Prague	No	planetlab-2.amst.nodes.planet-lab.org
349	357	342	-7.4	-2.2	7.4	2.2	Prague	No	planetlab2.comp.nus.edu.sg
127	127	118	0.1	0.1	9.5	8.1	Prague	No	planetlab2.cs.pitt.edu
82	82	79	-0.2	-0.3	3.3	4.1	Prague	No	planetlab2.cs.uoi.gr
138	137	141	1.1	0.8	1.1	0.8	Prague	Yes	planetlab2.cs.wisc.edu
147	146	143	1.4	0.9	3.9	2.7	Prague	No	planetlab-2.ece.iastate.edu
333	329	260	4.6	1.8	73.4	28.2	Prague	No	planetlab2.pop-rs.rnp.br
129	128	119	1.1	0.9	10.3	8.6	Prague	No	planetlab-2.rml.ryerson.ca
165	167	162	-2.2	-1.4	3.3	2.1	Prague	No	planetlab2.utep.edu
48	38	32	10.2	31.8	16.3	50.6	Prague	No	planetlab3.mini.pw.edu.pl
57	57	49	0.0	0.0	7.9	16.0	Prague	No	planetlab3.piotrkow.rd.tp.pl
32	32	39	0.1	0.3	0.1	0.4	Prague	Yes	planetlab-europe-01.ipv6.lip6.fr
188	188	197	-0.2	-0.1	-0.2	-0.1	Prague	Yes	planetlabnode-1.docomolabs-usa.com
111	112	99	-0.9	-0.9	11.5	11.5	Prague	No	planetlabone.ccs.neu.edu
177	177	165	-0.1	-0.0	12.4	7.5	Prague	No	planetslug1.cse.ucsc.edu
787	821	531	-34.0	-6.4	256.1	48.2	Prague	No	plnode01.cs.mu.oz.au
295	294	284	1.1	0.4	10.7	3.8	Prague	No	pub1-s.ane.cmc.osaka-u.ac.jp
165	166	169	-0.6	-0.4	-0.6	-0.4	Prague	Yes	ricepl-1.cs.rice.edu
178	177	194	0.2	0.1	0.2	0.1	Prague	Yes	sanfrancisco.planetlab.pch.net
133	133	120	0.1	0.1	12.7	10.5	Prague	No	scratchy.cs.uga.edu
81	81	71	0.1	0.2	9.8	13.7	Prague	No	stella.planetlab.ntua.gr
40	41	46	-0.4	-0.9	-0.4	-1.0	Prague	Yes	sv01-h010.utt.fr
140	140	131	-0.2	-0.1	9.0	6.9	Prague	No	vn1.cs.wustl.edu

Berlin anycast node

Hint:
PRG - Prague, TXL - Berlin
A2Uloc - difference of anycast/berlin to unicast/berlin
A2Usho - difference of anycast to shortest unicast ping
Match - if the shortest ping matches the anycast selection

Any [ms]	PRG [ms]	TXL [ms]	A2Uloc [ms]	A2Uloc [%]	A2Usho [ms]	A2Usho [%]	Select anycast	Match	Host
26	22	26	0.1	0.4	4.3	19.5	Berlin	No	aladdin.planetlab.extranet.uni-passau.de
28	36	27	0.2	0.5	0.2	0.5	Berlin	Yes	chronos.disy.inf.uni-konstanz.de
165	184	165	0.6	0.4	0.6	0.4	Berlin	Yes	cs-planetlab1.cs.surrey.sfu.ca
18	24	18	0.1	0.7	0.1	0.7	Berlin	Yes	edi.tkn.tu-berlin.de
23	31	23	0.3	1.4	0.3	1.4	Berlin	Yes	freedom.ri.uni-tuebingen.de
19	27	19	-0.3	-1.7	-0.3	-1.7	Berlin	Yes	host1.planetlab.informatik.tu-darmstadt.de
21	29	20	0.5	2.4	0.5	2.4	Berlin	Yes	irabonnie.iralab.uni-karlsruhe.de
133	140	131	1.7	1.3	1.7	1.3	Berlin	Yes	mtuplanetlab1.cs.mtu.edu
119	130	119	-0.0	-0.0	-0.0	-0.0	Berlin	Yes	pl1.csl.utoronto.ca
141	129	145	-4.3	-2.9	11.6	8.9	Berlin	No	pl1.cs.utk.edu
64	55	64	-0.0	-0.0	9.7	17.6	Berlin	No	plab-1.sinp.msu.ru
18	15	18	0.0	0.1	2.9	18.8	Berlin	No	plab201.wiai.uni-bamberg.de
22	24	23	-0.3	-1.2	-0.3	-1.2	Berlin	Yes	plane-lab-pb1.uni-paderborn.de
18	24	18	0.0	0.1	0.0	0.1	Berlin	Yes	planet01.hhi.fraunhofer.de
18	20	18	0.0	0.1	0.0	0.1	Berlin	Yes	planet1.inf.tu-dresden.de
19	21	19	-0.2	-0.9	-0.2	-0.9	Berlin	Yes	planet1.l3s.uni-hannover.de
132	133	131	1.1	0.9	1.1	0.9	Berlin	Yes	planet1.ottawa.canet4.nodes.planet-lab.org
165	168	167	-2.6	-1.5	-2.6	-1.5	Berlin	Yes	planet1.scs.stanford.edu
20	15	20	0.2	0.8	5.9	39.4	Berlin	No	planet2.prakinf.tu-ilmenau.de
122	131	122	-0.0	-0.0	-0.0	-0.0	Berlin	Yes	planetlab01.erin.utoronto.ca
298	303	292	5.7	2.0	5.7	2.0	Berlin	Yes	planetlab-01.kyushu.jgn2.jp
21	12	22	-0.1	-0.5	9.3	73.9	Berlin	No	planetlab01.mpi-sws.mpg.de
279	293	279	-0.1	-0.0	-0.1	-0.0	Berlin	Yes	planetlab-01.naist.jp
276	287	276	-0.0	-0.0	-0.0	-0.0	Berlin	Yes	planetlab0.dojima.wide.ad.jp
269	280	269	-0.1	-0.0	-0.1	-0.0	Berlin	Yes	planetlab0.otemachi.wide.ad.jp
133	118	127	6.0	4.8	15.0	12.7	Berlin	No	planetlab1.cs.dartmouth.edu
145	134	139	6.0	4.3	11.3	8.4	Berlin	No	planetlab1.csee.usf.edu
110	118	111	-1.2	-1.1	-1.2	-1.1	Berlin	Yes	planetlab-1.cs.princeton.edu
165	168	170	-5.3	-3.1	-3.7	-2.2	Berlin	No	planetlab-1.cs.uh.edu
189	198	189	0.0	0.0	0.0	0.0	Berlin	Yes	planetlab1.eecs.wsu.edu
25	15	25	0.0	0.1	9.8	62.1	Berlin	No	planetlab1.exp-math.uni-essen.de
20	15	20	-0.1	-0.6	5.7	37.7	Berlin	No	planetlab1.fem.tu-ilmenau.de
19	24	19	0.0	0.2	0.0	0.2	Berlin	Yes	planetlab-1.fokus.fraunhofer.de
131	110	125	6.3	5.0	20.7	18.7	Berlin	No	planetlab1.georgetown.edu
21	22	21	0.2	0.8	0.2	0.8	Berlin	Yes	planetlab1.informatik.uni-goettingen.de
23	13	23	-0.1	-0.2	9.6	69.8	Berlin	No	planetlab1.informatik.uni-kl.de
23	14	23	0.1	0.4	9.4	67.2	Berlin	No	planetlab1.informatik.uni-wuerzburg.de
23	13	23	0.1	0.5	9.6	69.3	Berlin	No	planetlab1.itwm.fhg.de
26	16	26	0.0	0.2	9.4	56.4	Berlin	No	planetlab1.lkn.ei.tum.de
269	282	269	0.0	0.0	0.0	0.0	Berlin	Yes	planetlab1.sfc.wide.ad.jp
135	137	136	-1.0	-0.8	-1.0	-0.8	Berlin	Yes	planetlab1.win.trlabs.ca
19	25	4	15.2	324.4	15.2	324.4	Berlin	Yes	planetlab1.wiwi.hu-berlin.de
23	24	23	-0.1	-0.6	-0.1	-0.6	Berlin	Yes	planetlab2.eecs.jacobs-university.de
279	351	346	-67.7	-19.5	-67.7	-19.5	Berlin	Yes	planetlab2.ie.cuhk.edu.hk
18	12	18	0.1	0.3	5.8	45.2	Berlin	No	planetlab2.informatik.uni-erlangen.de
119	128	128	-8.8	-6.8	-8.5	-6.7	Berlin	No	planetlab2.mnlab.cti.depaul.edu
23	30	24	-0.8	-3.5	-0.8	-3.5	Berlin	Yes	planetvs1.informatik.uni-stuttgart.de
104	114	108	-4.0	-3.6	-4.0	-3.6	Berlin	Yes	plgmu2.ite.gmu.edu
123	128	117	5.9	5.0	5.9	5.0	Berlin	Yes	server1.planetlab.iit-tech.net

ICMP packets from 146 planet-lab nodes were routed to Prague anycast node
and from 49 planet-lab nodes packets were routed to Berlin anycast node.

32.8%(64 nodes) matched and 67.2%(131 nodes) did not match the measured latency of selected
anycast destination and unicast ping to the same destination.
Measured out of total 195 planet-lab nodes.

B SER CONFIGURATIONS

B.1 SER Config for Anycast DNS-based Method

```
# ----------- global configuration parameters ------------------------

# adjust debug level, useful values are 0 (shut up) or >5 (very verbose)
debug=3
memdbg=100

check_via=no
dns=no
rev_dns=no

listen="213.192.59.77"

# ------------------ module loading ----------------------------------

loadmodule "sl"
loadmodule "rr"
loadmodule "maxfwd"
loadmodule "nathelper"
loadmodule "textops"
loadmodule "ctl"
loadmodule "uri"
loadmodule "tm"
loadmodule "mangler"

# optional listen addresses, if no one is specified,
# ctl will listen on unixs:/tmp/ser_ctl
modparam("ctl", "binrpc", "unix:/tmp/ser_ctl_unicast") # default
# unix sockets and fifo creation parameters
modparam("ctl", "mode", 0660)                       # permissions
#modparam("ctl", "group", "ser")

#: -- rr params --
#: add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)

#: don't add fromtags to RR, it helps keep the messages smaller
modparam("rr", "append_fromtag", 0)


#advanced options

dns_retr_time=1
dns_retr_no=1
dns_servers_no=1
dns_use_search_list=no

use_dns_cache=on
use_dns_failover=on

use_dst_blacklist=on
dst_blacklist_mem=10
dst_blacklist_expire=300
dst_blacklist_gc_interval=120

tcp_connection_lifetime=3600
tcp_max_connections=1024  # 1024 connections
tcp_send_timeout=5
tcp_connect_timeout=1
tcp_buf_write=1
tcp_fd_cache=1
tcp_conn_wq_max=65536
tcp_wq_max=10240000
tcp_delayed_ack=1
tcp_linger2=10
tcp_keepalive=yes
tcp_keepidle=30
tcp_keepintvl=5
tcp_keepcnt=4

flags F_NAT, F_FWD_IPTEL, F_RELAY_ANYCAST, F_ORIG_RTPPROXY,
                F_MANGLE_CONTACT_NONNAT_REPLY;

route{

        #: initial sanity checks -- messages with
        #: max_forwards==0, or excessively long requests
        if (!mf_process_maxfwd_header("10")) {
                sl_send_reply("483","Too Many Hops");
                break;
        };
```

```
        if (msg:len >= 2048 ) {
                sl_send_reply("513", "Message too big");
                break;
        };

        force_rport();
        force_tcp_alias();
/*      RR disabled, interferes badly with contact rewriting/uri fixing */
        #: we record-route all messages -- to make sure that
        #: subsequent messages will go through our proxy; that's
        #: particularly good if upstream and downstream entities
        #: use different transport protocol
        #: we don't record route REGISTERs, messages within
        #: a dialog (pointless).

        if (!method=="REGISTER" && !has_totag())
                if (proto==udp)
                        record_route_preset("213.192.59.77");
                else if (proto==tcp)
                        record_route_preset("213.192.59.77;transport=tcp");
                else{
                        sl_send_reply("500", "Unsupported protocol");
                        drop;
                }
                #record_route();

        #: P-Hint and P-Behind-NAT are used by us, don't allow it before
        remove_hf("P-Hint");
        remove_hf("P-Behind-NAT");

        # set source ip used for sending
        if (src_ip==213.192.59.75){
                # if it comes from iptel use anycast
                #force_send_socket(91.199.168.1);
                route(R_FIX_URI);    # fix possible mangled uris
        }else{
                # else use normal ip (it should be forwarded back to iptel)
                #force_send_socket(217.9.54.30);
                append_hf("P-iptel-fwd: yes\r\n");
                setflag(F_FWD_IPTEL);
                # if NAT, mangle contacts (we want subseq. messages to go
                # through us, so this should be applied both to normal contact
                # updating request and to REGISTERs)
                if (isflagset(F_NAT) || nat_uac_test("19")){
                        setflag(F_NAT);
                }
                # mangle all the contacts due to possible firewalls
                        if (!search('^(Contact|m)[ \t]*:.*sips?:[^>[:cntrl:]@]*_RCVD|NAT_'))
                        {
                                # only if not already fixed
                                if (isflagset(F_NAT)){
                                        encode_contact("_NAT_", "213.192.59.77");
                                        append_hf("P-Behind-NAT: Yes\r\n");
                                }else{
                                        encode_contact("_RCVD_", "213.192.59.77");
                                        append_hf("P-Contact-Mangled: Yes\r\n");
                                }
                                if (method=="REGISTER"){
                                        resetflag(F_NAT);
                                        t_on_reply("R_UNMANGLE_CONTACT_REPLY");
                                }
                        }
        }
/*  RR interferes with the contact rewriting/uri fixing
    (loose_route() sees a myself uri and thinks a strict router needs fixing..)
*/

        #: subsequent messages withing a dialog should take the
        #: path determined by record-routing
        if (loose_route()) {
                if (!has_totag()){
                        sl_send_reply("404", "Preloaded routes forbidden");
                        break;
                }
                resetflag(F_FWD_IPTEL); # obey rr
        }

        route(R_FWD);        # forward
}


onreply_route[R_RTPPROXY_REPLY]{
        if (status=~ "(183)|2[0-9][0-9]" ){
                if (!is_present_hf("P-RTP-Proxy")){
                        force_rtp_proxy("r");
                        append_hf("P-RTP-Proxy: YES\r\n");
                }
                route(R_MANGLE_CONTACT_REPLY);
```

```
        }
}


onreply_route[R_MANGLE_CONTACT_REPLY]{
        if (status=~ "(18[0-9])|2[0-9][0-9]" ){
                if ((src_ip!=213.192.59.75) &&
                   !search('^(Contact|m)[ \t]*:.*sips?:[^>[:cntrl:]@]*_(RCVD|NAT)_')){
                        if (isflagset(F_MANGLE_CONTACT_NONNAT_REPLY))
                                encode_contact("_RCVD_", "213.192.59.77");
                        else
                                encode_contact("_NAT_", "213.192.59.77");
                }
        }
}


onreply_route[R_UNMANGLE_CONTACT_REPLY]{
        decode_contact_header();
}


failure_route[R_RTPPROXY_FAILURE]{
        if (isflagset(F_ORIG_RTPPROXY))
                unforce_rtp_proxy();
}


route[R_FWD]{
        if (isflagset(F_NAT))
                route(R_RTPPROXY);
        else if (isflagset(F_MANGLE_CONTACT_NONNAT_REPLY))
                t_on_reply("R_MANGLE_CONTACT_REPLY");
        if (isflagset(F_RELAY_ANYCAST))
                route(R_RELAY_ANYCAST);
        else if (isflagset(F_FWD_IPTEL))
                route(R_FWD_IPTEL);
        else{
                #forward(uri:host, uri:port);
                t_relay();
        }
}


route[R_RELAY_ANYCAST]{
        t_relay();
}


route[R_FWD_IPTEL]
{
        t_relay("sip01.iptel.org", "5060");
}


route[R_FIX_URI]
{
        if (uri==myself){
                if (uri=~"_NAT_"){
                        setflag(F_NAT);
                        if (!decode_contact()){
                                sl_reply("500", "decode uri failed");
                                drop;
                        }
                        append_hf("P-uri-decoded: NAT\r\n");
                }else if (uri=~"_RCVD_"){
                        setflag(F_MANGLE_CONTACT_NONNAT_REPLY);
                        if (!decode_contact()){
                                sl_reply("500", "decode uri failed");
                                drop;
                        }
                        append_hf("P-uri-decoded: non-NAT\r\n");
                }else{
                        sl_reply("500", "local uris not allowed");
                        drop;
                }
                setflag(F_RELAY_ANYCAST);
        }else{
                append_hf("P-iptel-fwd: failover\r\n");
                setflag(F_FWD_IPTEL);
        }
}


route[R_RTPPROXY]{
        # don't RTP proxy if somebody already did it before us
        if (!is_present_hf("P-RTP-Proxy")){
                if (method=="BYE"||method=="CANCEL")
```

```
                unforce_rtp_proxy();
            else if (method=="INVITE"){
                force_rtp_proxy("r");
                append_hf("P-RTP-Proxy: YES\r\n");
                setflag(F_ORIG_RTPPROXY);
                t_on_failure("R_RTPPROXY_FAILURE");
                t_on_reply("R_RTPPROXY_REPLY");
            }
        }else{
            # if RTP PROXIED by someone else, we still want to
            # catch the reply to fix the contact
            if (method=="INVITE")
                t_on_reply("R_MANGLE_CONTACT_REPLY");
        }
}
```

B.2 SER Config for Anycast SIP-based Method

```
# ----------- global configuration parameters ------------------------

# adjust debug level, useful values are 0 (shut up) or >5 (very verbose)
debug=3
memdbg=100

check_via=no
dns=no
rev_dns=no

listen="91.199.168.1"
listen="213.192.59.76"

# ------------------ module loading ----------------------------------

loadmodule "sl"
loadmodule "rr"
loadmodule "maxfwd"
loadmodule "nathelper"
loadmodule "textops"
loadmodule "ctl"
loadmodule "uri"
loadmodule "tm"
loadmodule "mangler"

# optional listen addresses, if no one is specified,
# ctl will listen on unixs:/tmp/ser_ctl
modparam("ctl", "binrpc", "unix:/tmp/ser_ctl") # default
# unix sockets and fifo creation parameters
modparam("ctl", "mode", 0660)                     # permissions

#: -- rr params --
#: add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)

#: don't add fromtags to RR, it helps keep the messages smaller
modparam("rr", "append_fromtag", 0)

#: -- nathelper params --
#modparam("nathelper", "rtpproxy_disable", 1)


#advanced options

#mlock_pages=yes
#shm_force_alloc=yes
#real_time=7

dns_retr_time=1
dns_retr_no=1
dns_servers_no=1
dns_use_search_list=no

use_dns_cache=on
use_dns_failover=on

use_dst_blacklist=on
dst_blacklist_mem=10
dst_blacklist_expire=300
dst_blacklist_gc_interval=120

tcp_connection_lifetime=3600
tcp_max_connections=1024  # 1024 connections
tcp_send_timeout=5
tcp_connect_timeout=1
tcp_buf_write=1
tcp_fd_cache=1
tcp_conn_wq_max=65536
tcp_wq_max=10240000
tcp_delayed_ack=1
tcp_linger2=10
tcp_keepalive=yes
tcp_keepidle=30
tcp_keepintvl=5
tcp_keepcnt=4

syn_branch=0  # we want to generate the same branch for the same request at
              # different boxes, to account for anycast routing changes
                        # mid-transaction

flags F_NAT, F_FWD_IPTEL, F_RELAY_ANYCAST, F_ORIG_RTPPROXY,
                F_MANGLE_CONTACT_NONNAT_REPLY;

route{
```

```
        #: initial sanity checks -- messages with
        #: max_forwards==0, or excessively long requests
        if (!mf_process_maxfwd_header("10")) {
                sl_send_reply("483","Too Many Hops");
                break;
        };
        if (msg:len >= 2048 ) {
                sl_send_reply("513", "Message too big");
                break;
        };

        force_rport();
        force_tcp_alias();
/*      RR disabled, interferes badly with contac rewriting/uri fixing */
        #: we record-route all messages -- to make sure that
        #: subsequent messages will go through our proxy; that's
        #: particularly good if upstream and downstream entities
        #: use different transport protocol
        #: we don't record route REGISTERs, messages within
        #: a dialog (pointless).

        if (!method=="REGISTER" && !has_totag())
                if (proto==udp)
                        record_route_preset("213.192.59.76");
                else if (proto==tcp)
                        record_route_preset("213.192.59.76;transport=tcp");
                else{
                        sl_send_reply("500", "Unsupported protocol");
                        drop;
                }

        #: P-Hint and P-Behind-NAT are used by us, don't allow it before
        remove_hf("P-Hint");
        remove_hf("P-Behind-NAT");

        # set source ip used for sending
        if (src_ip==213.192.59.75){
                # if it comes from iptel use anycast
                force_send_socket(91.199.168.1);
                route(R_FIX_URI);    # fix possible mangled uris
        }else{
                # else use normal ip (it should be forwarded back to iptel)
                force_send_socket(213.192.59.76);
                append_hf("P-iptel-fwd: yes\r\n");
                setflag(F_FWD_IPTEL);
                # if NAT, mangle contacts (we want subseq. messages to go
                # through us, so this should be applied both to normal contact
                # updating request and to REGISTERs)
                if (isflagset(F_NAT) || nat_uac_test("19")){
                        setflag(F_NAT);
                }
                # mangle all the contacts due to possible firewalls
                        if (!search('^(Contact|m)[ \t]*:.*sips?:[^>[:cntrl:]@]*_RCVD|NAT_'))
                        {
                                # only if not already fixed
                                if (isflagset(F_NAT)){
                                        encode_contact("_NAT_", "213.192.59.76");
                                        append_hf("P-Behind-NAT: Yes\r\n");
                                }else{
                                        encode_contact("_RCVD_", "213.192.59.76");
                                        append_hf("P-Contact-Mangled: Yes\r\n");
                                }
                                if (method=="REGISTER"){
                                        resetflag(F_NAT);
                                        t_on_reply("R_UNMANGLE_CONTACT_REPLY");
                                }
                        }
        }
/*  RR interferes with the contact rewriting/uri fixing
    (loose_route() sees a myself uri and thinks a strict router needs fixing..)
*/

        #: subsequent messages withing a dialog should take the
        #: path determined by record-routing
        if (loose_route()) {
                if (!has_totag()){
                        sl_send_reply("404", "Preloaded routes forbidden");
                        break;
                }
                resetflag(F_FWD_IPTEL); # obey rr
        }

        route(R_FWD);        # forward
}


onreply_route[R_RTPPROXY_REPLY]{
        if (status=~ "(183)|2[0-9][0-9]" ){
```

```
                if (!is_present_hf("P-RTP-Proxy")){
                        force_rtp_proxy("r");
                        append_hf("P-RTP-Proxy: YES\r\n");
                }
                route(R_MANGLE_CONTACT_REPLY);
        }
}


onreply_route[R_MANGLE_CONTACT_REPLY]{
        if (status=~ "(18[0-9])|2[0-9][0-9]" ){
                if ((src_ip!=213.192.59.75) &&
                   !search('^(Contact|m)[ \t]*:.*sips?:[^>[:cntrl:]@]*_(RCVD|NAT)_')){
                        if (isflagset(F_MANGLE_CONTACT_NONNAT_REPLY))
                                encode_contact("_RCVD_", "213.192.59.76");
                        else
                                encode_contact("_NAT_", "213.192.59.76");
                }
        }
}


onreply_route[R_UNMANGLE_CONTACT_REPLY]{
        decode_contact_header();
}


failure_route[R_RTPPROXY_FAILURE]{
        if (isflagset(F_ORIG_RTPPROXY))
                unforce_rtp_proxy();
}


route[R_FWD]{
        if (isflagset(F_NAT))
                route(R_RTPPROXY);
        else if (isflagset(F_MANGLE_CONTACT_NONNAT_REPLY))
                t_on_reply("R_MANGLE_CONTACT_REPLY");
        if (isflagset(F_RELAY_ANYCAST))
                route(R_RELAY_ANYCAST);
        else if (isflagset(F_FWD_IPTEL))
                route(R_FWD_IPTEL);
        else{
                #forward(uri:host, uri:port);
                t_relay();
        }
}


route[R_RELAY_ANYCAST]
{
        # NOTE: force_send_socket doesn't work w/ stateless forward,
        #  (the forwarded replies will come from the forced socket
        #   instead of the original socket on which the msg was recvd)
        #forward(uri:host, uri:port);
        #force_send_socket(91.199.168.1);
        t_relay();
}


route[R_FWD_IPTEL]
{
        # see above NOTE
        #forward(sip01.iptel.org, 5060);
        #force_send_socket(213.192.59.76);
        t_relay("sip01.iptel.org", "5060");
}


route[R_FIX_URI]
{
        if (uri==myself){
                if (uri=~"_NAT_"){
                        setflag(F_NAT);
                        if (!decode_contact()){
                                sl_reply("500", "decode uri failed");
                                drop;
                        }
                        append_hf("P-uri-decoded: NAT\r\n");
                }else if (uri=~"_RCVD_"){
                        setflag(F_MANGLE_CONTACT_NONNAT_REPLY);
                        if (!decode_contact()){
                                sl_reply("500", "decode uri failed");
                                drop;
                        }
                        append_hf("P-uri-decoded: non-NAT\r\n");
                }else{
                        sl_reply("500", "local uris not allowed");
```

```
                        drop;
                }
                setflag(F_RELAY_ANYCAST);
        }else{
                append_hf("P-iptel-fwd: failover\r\n");
                setflag(F_FWD_IPTEL);
        }
}


route[R_RTPPROXY]{
        # don't RTP proxy if somebody already did it before us
        if (!is_present_hf("P-RTP-Proxy")){
                if (method=="BYE"||method=="CANCEL")
                        unforce_rtp_proxy();
                 else if (method=="INVITE"){
                        force_rtp_proxy("r");
                        append_hf("P-RTP-Proxy: YES\r\n");
                        setflag(F_ORIG_RTPPROXY);
                        t_on_failure("R_RTPPROXY_FAILURE");
                        t_on_reply("R_RTPPROXY_REPLY");
                 }
        }else{
                # if RTP PROXIED by someone else, we still want to
                # catch the reply to fix the contact
                if (method=="INVITE")
                        t_on_reply("R_MANGLE_CONTACT_REPLY");
        }
}
```