La Téléphonie IPv6

Simon Perreault

Viagénie

{mailto|sip}:simon.perreault@viagenie.ca

http://www.viagenie.ca

Présenté au AMUG à Montréal le 18 décembre 2008

Qui suis-je?

- Consultant en réseautique et VoIP chez Viagénie.
- Co-auteur du port de Asterisk à Ipv6 (voir http://www.asteriskv6.org)
- Auteur du port de FreeSWITCH à IPv6 (intégré depuis version 1.0.1)
- Auteur de Numb, un serveur STUN/TURN (voir http://numb.viagenie.ca)
- Participation à Astricon, Cluecon, SIPit, IETF, etc.
- Développé plusieurs applications VoIP sur mesure.

Cette présentation

- Cette présentation est un mélange de...
  - Contenu original en français
  - Extraits de présentations faites à Astricon 2006 et 2008 et Cluecon 2008 par Simon Perreault et Marc Blanchet
- Mille excuses pour les diapos en anglais!

Plan

VIAGÉNIE

- IPv6
- Why IPv6 and VoIP
- New API
- Asteriskv6
- FreeSWITCHv6
- Lessons Learned
- Conclusion

- NAT traversal si le temps le permet

IPv6?

- New version of IP:
  - fixes IPv4 issues
  - adds functionality
- Addresses:
  - 128 bits
  - written in hex with **:** as separator; method to compress the writing: all zeros = **::**
  - **2001:db8:1:1::1**
  - In URL: enclose with []: **sip:jdoe@[2001:db8:1:1::1]:5060**
  - Loopback is **::1**
  - Link(Subnet,vlan,...) mask is fixed: **/64**
  - Unique private address space: no collision of private networks

IPv6?

- Addresses (cont):
  - Scoped addressing: link scope, site scope. An enabled IPv6 stack has already an IPv6 address (link scope) on each interface, even if no IPv6 external connectivity.
  - Multiple addresses per interface: link-scope, global, [site,...]
  - No NAT.
- Mobility: keep connections up even when host changes IP address
- Autoconfiguration: Stateless address allocation **without DHCP server**. Routers announce the link prefix on the link. Hosts use their MAC address for the host part of the address
- Integrated IPsec
- Many more features

IPv6 Market

- IPv4 address depletion: < 25% of remaining address space. Predictions of exhaustion for 2009-2011.
- Asia
  - Japan: see http://www.v6pc.jp
  - China: through NGN. Olympics is important milestone.
- US government:
  - Mandating IPv6 for 2008 in all agencies
  - DoD is leading
- Providers (short list):
  - Teleglobe/VSNL, NTT, AT&T, GlobalCrossing,...
  - Comcast: can't address all the devices (100M+) with IPv4. Deploying IPv6. (DOCSIS 3.0 is IPv6-ready).

IPv6 Support

- Support on OS (stack and API):
  - Same (new) API everywhere!!! ;-)
  - Since: Linux 2.4, FreeBSD 4.X, MacOSX 10.2, Windows XP, Solaris 8, ...
- Opensource Apps: Apache 2.0+ (1.3 with a patch), Sendmail, Postfix, Open SSH, Xfree/Xorg, ...
  - Now Asterisk and FreeSWITCH... ;-)
- Support on network gear: Cisco, Juniper, Checkpoint, Quagga/Zebra, ...

Why IPv6 and VoIP?

- IPv6 and SIP
  - delivers direct end-2-end reachability between any host.
  - No NAT, no STUN, no TURN, no ICE, no MIDCOM, = no complexity, “just works”.
  - True end-2-end media path.
  - Much easier to deploy. A VoIP-IPv6 deployment in Japan found important cost reductions because of the ease of installation and support.
- To have an IPv6-enabled application, such as a PBX, need to convert to the new API.

New API

- New API for IPv6 [RFC3493, RFC3542]
  - Makes the application version independent. The stack chooses which IP version will be used for that connection.
  - A ported application becomes IP version unaware.
  - No change to socket(), bind(), listen(), accept(), connect(), recv(), send(), close()...
- Changes:
  - Struct **hostent** replaced by struct **addrinfo**
    - Addrinfo is a linked list of addresses
    - It contains everything needed to initialize a socket.

New API

VIAGÉNIE

- Changes:
  - sockaddr record
    - **sockaddr_in** : IPv4
    - **sockaddr_in6** : IPv6 only. Do not use.
    - **sockaddr_storage**: version independent for memory allocations.
    - **sockaddr *****: for casting
  - **gethostbyname()** replaced by **getaddrinfo()**
  - **gethostbyaddr(), inet_addr(), inet_ntoa()** replaced by **getnameinfo()**
- More considerations:
  - Parsing URLs: need to take care of the IPv6 syntax (i.e. [])
  - Parsing and storing IP addresses

VIAGÉNIE

Exemples de code

Établir une connexion TCP (vieux style IPv4 seulement)


```
int tcp_connect( const char* host, unsigned short port )
{
    struct hostent* h;
    struct sockaddr_in sin;
    int s;

    if ( !(h = gethostbyname(host)) )
        Return -1;

    sin.sin_family = AF_INET;
    sin.sin_port = htons(port);
    sin.sin_addr = (struct in_addr*)h->h_addr;

    if ( (s = socket(AF_INET, SOCK_STREAM, 0) < 0 )
        return -1;

    if ( !connect(s, (struct sockaddr*)sin, sizeof(sin)) ) {
        close(s);
        return -1;
    }

    return s;
}
```

gethostbyname() n'est pas réentrante

IPv4 hardcodé

Une seule adresse possible

Établir une connexion TCP (style *version-independent*)

VIAGÉNIE

```
int tcp_connect( const char* host, const char* port )
{
    struct addrinfo hints, *res, *iter;
    int s = -1;

    memset( &hints, 0, sizeof(hints) );
    hints.ai_socktype = SOCK_STREAM;
    if ( getaddrinfo(host, port, &hints, &res) != 0 )
        return -1;

    for ( iter = res; iter; iter = iter->ai_next ) {
        if ( (s = socket(iter->ai_family, iter->ai_socktype,
                         iter->ai_protocol)) < 0 )
            break;

        if ( connect(s, iter->ai_addr, iter->ai_addrlen) != 0 ) {
            close(s);
            s = -1;
        }
        else break;
    }

    freeaddrinfo(res);

    return s;
}
```

Itération sur plusieurs adresses possibles

Le protocole n'est pas hardcodé

La valeur retournée par getaddrinfo() nous appartient

Écouter sur un port TCP (vieux style IPv4 seulement)


```
int tcp_server( unsigned short port )
{
   int s;
   int true = 1;
   struct sockaddr_in sin;

   sin.sin_family = AF_INET;
   sin.sin.port = htons(port);
   sin.sin_addr = INADDR_ANY;

   if ( (s = socket(AF_INET, SOCK_STREAM, 0)) < 0
         || setsockopt(s, SOL_SOCKET, SO_REUSEADDR,
                       &true, sizeof(true)) != 0
         || bind(s, (struct sockaddr*)&sin, sizeof(sin)) != 0
         || listen(s, SOMAXCONN) != 0 ) {
      close(s);
      return -1;
   }

   return s;
}
```

IPv4 hardcodé

Un seul port possible

Écouter sur **des** ports TCP (style *version-independent*)


```
int tcp_server( const char* port, int* s )
{
    int num = 0;
    struct getaddrinfo hints, *res, *iter;

    memset( &hints, 0, sizeof(hints) );
    hints.ai_flags = AI_PASSIVE;
    hints.ai_socktype = SOCK_STREAM;
    if ( getaddrinfo(NULL, port, &hints, &res) != 0 )
        return -1;

    for ( iter = res; iter; iter = iter->ai_next ) {
        if ( (s[num] = socket(iter->ai_family, iter->ai_socktype,
                              iter->ai_protocol)) < 0
                || setsockopt(s[num], SOL_SOCKET, SO_REUSEADDR,
                              &true, sizeof(true)) != 0
                || bind(s[num], &iter->ai_addr, iter->ai_addrlen) != 0
                || listen(s[num], SOMAXCONN) != 0 )
            close(s[num]);
        else ++num;
    }

    freeaddrinfo(res);

    return num;
}
```

Array de sockets

Itération sur plusieurs adresses

On retourne le nombre de sockets

Afficher l'adresse du socket (vieux style IPv4 seulement)

IPv4 hardcodé

inet_ntoa() n'est pas réentrante

```
void print_address( int s )
{
   struct sockaddr_in sin;
   socklen_t len = sizeof(sin);

   if ( getpeername(s, (struct sockaddr*)&sin, &len)) != 0 )
      return;

   printf( "%s:%hu", inet_ntoa(sin.sin_addr),
                     ntohs(sin.sin_port) );
}
```

Afficher l'adresse du socket (style *version-independent*)


```
void print_address( int s )
{
   struct sockaddr_storage ss;
   socklen_t len = sizeof(ss);
   char host[NI_MAXHOST];
   char port[NI_MAXSERV];

   if ( getpeername(s, (struct sockaddr*)&ss, &len) != 0 )
      return;

   if ( getnameinfo((struct sockaddr*)&ss, len,
                host, sizeof(host), port, sizeof(port),
                NI_NUMERICHOST | NI_NUMERICSERV) != 0 )
      return;

   printf( "%s%s%s:%s",
         ss.ss_family == AF_INET6 ? "[" : "",
         host,
         ss.ss_family == AF_INET6 ? "]" : "",
         port );
}
```

Contient IPv4 ou IPv6 ou ...

Flags pour empêcher lookup DNS

Format spécial pour IPv6

Best Practices for API usage

- Use **sockaddr_storage** for storing sockaddrs.
- Use **sockaddr *** for pointer to sockaddrs
- Always pass and carry the sockaddr length (in a **socklen_t**) to be fully portable across OS platforms.
- After the **getaddrinfo()** call, go through the link list of addrinfo to connect.
- Parse addresses and URL to support both IPv4 and IPv6 addresses (with port numbers) syntax.
- Do not use IPv4-mapped addresses or old API calls (gethostbyname2(), getipnode*())

Asteriskv6

Challenges with IPv6 in Asterisk chan_sip

- Current architecture supports a single socket : 'sipsock'.
- The default source address is hardcoded to 0.0.0.0.
- The RTP socket is initialized from 'sipsock'
- Widespread use of sockaddr_in structures and short buffers (>256 bytes) to store hostnames and IP address strings.

- Many instances of similar code for parsing SIP url.

Design choices

- Use multiple sockets
  - Initial patch provides 1 socket per address family.
  - future work should include multiple sockets for each address family.
- Version independent when possible
  - Whenever possible, do not use sockaddr_in or sockaddr_in6 and never guess at the length of a sockaddr structure.
  - Only exception should be for setting socket options.

Code changes

- Replaced all use of sockaddr_in in data structures with sockaddr_storage.
- Associates a socklen_t element with each sockaddr_storage.
  - the socklen member is only initialized when a sockaddr_in of sockaddr_in6 structure is copied in the allocated memory... never when the memory is allocated.
- Use the new ast_vinetsock API

New ast_vinetsock API

- ast_netsock (netsock.h) is currently used in chan_iax, not in chan_sip.
- ast_netsock has link lists to manage multiple sockets.
- the ast_netsock API was augmented to support IPv6.
- New and modified functions are in the new ast_vinetsock namespace (defined in netsock.c): no collision with ast_netsock.
- 3 types of functions are defined in ast_vinetsock:
  - Address string parsing.
  - Address structure handling.
  - Socket management.

String parsing functions

- Parse host:port and address strings in a version independent way.
- Used for:
  - Parsing and validation of configuration files.
  - Parsing SIP header fields such as 'contact' and 'via'.
-
- Db store uses ':' between fields. ':' is used in IPv6 address. Enclosing IPv6 address in []. Impact for other db readers.

Address structure handling functions

- Initialize sockaddr structures from strings.
- Extract data from sockaddr structures.
- Build host:port and address strings from sockaddr structures.
- Used for:
  - Selecting a source address.
  - Printing addresses and host:port strings to logs and console.
  - Building SIP/SDP fields from address structures.

Socket management functions

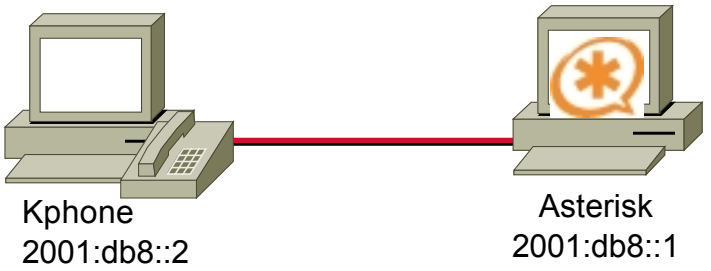
- Initialize sockets through ast_vinetsock structures.
- Set socket options.
- Bind on sockets and register callback functions.
- Used for:
  - Initializing IP listeners

Modifications to sip.conf

- 'bindaddr' now supports the address:port syntax such as:
  - 10.1.1.1
  - 10.1.1.1:5060
  - [2001:db8::1]
  - [2001:db8::1]:5060
- If no 'bindaddr' is specified for an address family, the wildcard is used (0.0.0.0 and [::]).
- 'host' contains only the address, therefore no brackets.
- 'bindport' is still supported for backward compatibility.

'Hello World' demo

- Uses Kphone as IPv6 SIP UA.
- Register to Asterisk.
- Make a call to play the 'Hello world' sound file.

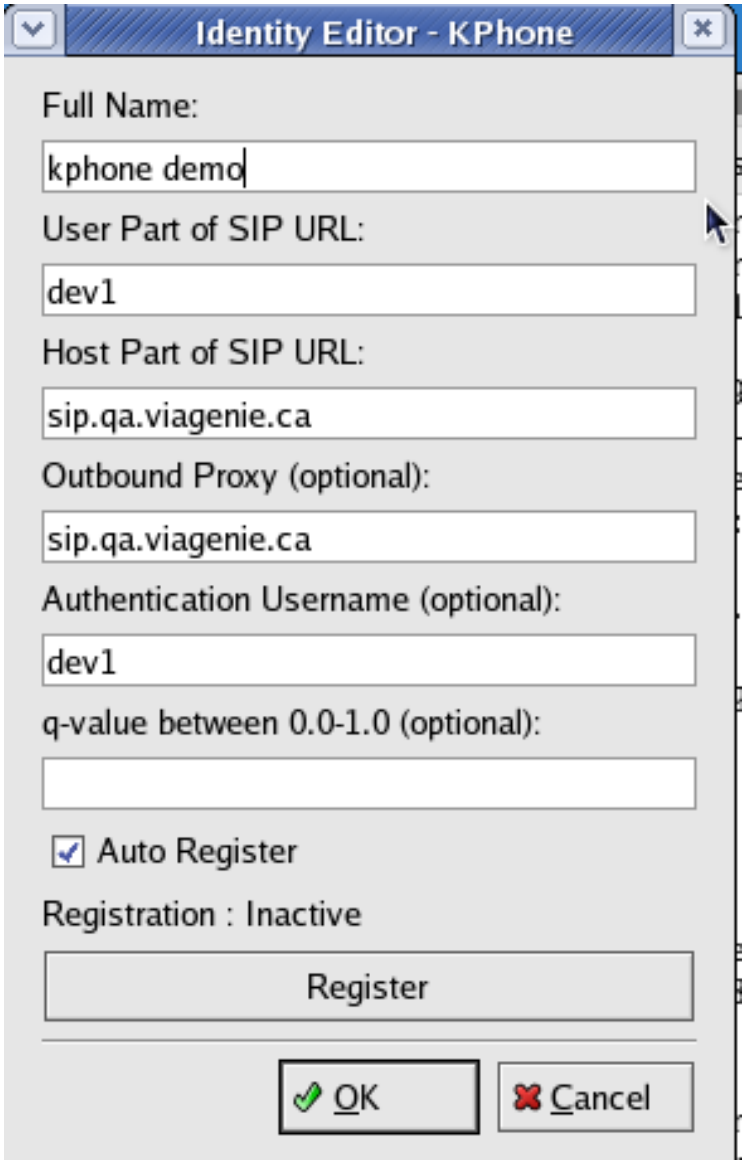


'Hello World' demo (cont.)

```
[general]
context=internal
bindaddr=[2001:db8::1]
allow=ulaw

[dev1]
type=friend
host=dynamic
context=internal
disallow=all
allow=ulaw

[dev2]
type=friend
host=dynamic
context=internal
disallow=all
allow=ulaw
```



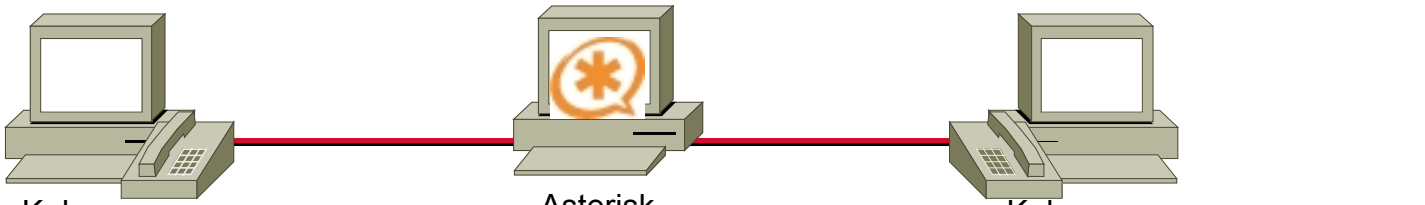
'Hello World' demo (cont.)

```
Reliably Transmitting (no NAT) to [2001:db8::2]:5060:
SIP/2.0 200 OK
Via: SIP/2.0/UDP [2001:db8::2];received=2001:db8::2
From: "Fred" <sip:dev1@sip.qa.viagenie.ca>;tag=61617230
To: <sip:2@sip.qa.viagenie.ca>;tag=as15d09daf
Call-ID: 3366000123
CSeq: 3245 INVITE
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY
Supported: replaces
Contact: <sip:2@[2001:db8::1]>
Content-Type: application/sdp
Content-Length: 168

v=0
o=root 1406 1406 IN IP6 2001:db8::1
s=session
c=IN IP6 2001:db8::1
t=0 0
m=audio 10610 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=silenceSupp:off - - - -
a=sendrecv
```

Bidirectional call demo

- 2 Kphone IPv6 SIP User Agents register to an Asterisk server.
- Establish a SIP call between the two user agents through an extension on Asterisk.



Bidirection call demo (cont.)

```
Reliably Transmitting (no NAT) to [2001:db8::3]:5060:
INVITE sip:lefebvre@[2001:db8::3];transport=udp SIP/2.0
Via: SIP/2.0/UDP [2001:db8::1]:5060;branch=z9hG4bK1dc90af0;rport
From: "Fred" <sip:dev1@[2001:db8::1]>;tag=as3038e677
To: <sip:lefebvre@[2001:db8::3];transport=udp>
Contact: <sip:dev1@[2001:db8::1]>
Call-ID: 5351c608290f3c9d03ab0e346ed44a80@2001:db8::1
CSeq: 102 INVITE
User-Agent: Asterisk PBX
Max-Forwards: 70
Date: Wed, 18 Oct 2006 19:38:06 GMT
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY
Supported: replaces
Content-Type: application/sdp
Content-Length: 224

v=0
o=root 1406 1406 IN IP6 2001:db8::2
s=session
c=IN IP6 2001:db8::2
t=0 0
m=audio 32770 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=silenceSupp:off - - - -
a=sendrecv
```

Impacts

- Files touched:
  - netsock.c/.h
  - chan_sip
  - rtp.c
  - Few others
- Some numbers:
  - ~25% of functions were changed/touched
  - ~ thousand lines changed/touched.
  - “Everywhere” in chan_sip, because: networking, logging (printing addresses) and sip url parsing.

FreeSWITCHv6

FreeSWITCHv6

- FreeSWITCH is IPv6-enabled since 1.0.1
- And there was much rejoicing...

FreeSWITCHv6

- SIP stack is Sofia-SIP, and is IPv6-enabled.
- Needed work:
  - mod_sofia glue
    - Uses address as string for registrar key. (Good!)
    - Some IPv4-specific URI building logic.
    - Some IPv4-specific SDP building logic.
  - Core: $${local_ip_v6} now contains useful data.
  - RTP:
    - Used a single port for input and output. Couldn't transcode network protocols.
    - Now opens a second port of other family when needed.

FreeSWITCHv6 (2)

- ACLs
  - Was completely IPv4-specific.
  - Redesigned for IPv4 and IPv6.
  - New in IPv6: scope ID must match.
  - Potential for optimization with SSE2 (anyone interested?)
  - Not contributed yet, needs more testing.

Lessons Learned

Use Addresses Sparingly

- Call connect() or bind(), then discard the address.
- Anti-pattern:
  - Have a host name resolving function return an address.
  - Later, use that address.
- Better:
  - Have a host name resolving function return **a list** of addresses.
  - Later, use these addresses.
- Best:
  - Combine the connecting/binding with the resolving.

Prepare for Multiplicity

- With version-independent programming, addresses are never encountered alone.
- Binding to **localhost** binds an IPv4 socket to **0.0.0.0** and an IPv6 socket to **::** (depends on OS).
- Hosts often have A as well as AAAA records. Try all of them when calling connect().
  - Let user choose sorting preference for IPv4 or IPv6.
- SDP offers contain many addresses. Use them all.

Banish Old APIs

- You should never use these:
  - inet_addr(), inet_aton(), inet_ntoa()
  - inet_pton(), inet_ntop()
  - gethostbyname(), gethostbyaddr()
- Not even these: (at least not for addresses)
  - htonl(), htons(), ntohl(), ntohs()
- All you need is:
  - **getaddrinfo()** (string to address)
  - **getnameinfo()** (address to string)

An Address is Atomic

- Do not separate address components.
  - Anti-pattern:

```
if ( sa->sa_family == AF_INET ) {
    addr = ((sockaddr_in*)sa)->sin_addr.s_addr;
    port = ((sockaddr_in*)sa)->sin_port;
} else if ( sa->sa_family == AF_INET6 ) {
[...]
snprintf( uri, sizeof(uri), “sip:%s@%s:%hu”,
    user, host, port );
```

  - Why it is bad:
    - Repeated logic for brackets in URL.
    - Not version-independent.
    - What about IPv6 scope ID?

An Address is Atomic (2)

- Better:

```
enum {
     URI_NUMERIC_HOST = 1,
     URI_NUMERIC_PORT = 2,
     URI_IGNORE_SCOPE = 4,
     […]
};

int build_uri( char *uri, size_t size,
    const char *user,
    const sockaddr *sa, socklen_t salen,
    int flags );
```

Eliminate Timeouts

- Many users already have an IPv6 address that is not reachable globally. (Local router, zombie Teredo, etc.)
- When connecting to results of getaddrinfo() sequentially, IPv6 connections will timeout.
- Reordering results so that IPv4 is tried first is a bad idea because the reverse may also be true.
- **Solution**: connect in parallel. (harder to implement)
- Even worse: DNS servers may timeout when queried for AAAA records. Cannot use getaddrinfo().
- **Solution**: single-family getaddrinfo() calls in parallel.

Eliminate Timeouts (2/2)

- Combine the two previous solutions within a single API for resolving and connecting.

```
int fd = resolve_connect( “example.com”, “80” );
```

- Use worker threads for resolving and connecting in parallel. (Better: a single thread with nonblocking sockets and a DNS resolving library.)
- Connect to each address as soon as it is received. Do not wait for all address families to finish resolving.
- Cancel other connections once one succeeds.
- Disadvantage: this wastes packets. May be significant in some cases (e.g. lots of short connections).

For Protocol Designers

- Protocols that transport addresses are harder to implement in a version-independent way.
- SIP, RTSP, and SDP do transport addresses **very much**.
- Many ways to encode addresses make it hard:
  - By themselves (e.g. **c=IN IP6 2001:db8::1**)
  - With brackets and port (e.g. **Via: SIP/2.0/UDP [2001:db8::1]:5060**)
  - Implicitly as part of any URI (e.g. **From: <sip:jdoe@example.com>**)

VoIPv6 Deployment

IPv6 is not an IPv4 killer

- IPv6 is something that you **add** to a network.
- Goal: To provide new IPv6 services, not to replace old IPv4 services.

It starts with purchasing

- Networking and VoIP equipment investments may last for many years.
- Impending IPv4 address shortage.
- Therefore, make sure new equipment is IPv6-ready.

IPv4 - IPv6 Interoperability

- IPv4 and IPv6 UAs can communicate via a relay.
- Usually relay is a B2BUA (e.g. FreeSWITCH)
- Relaying media may cause unwanted load.
- Consider using a cross-protocol TURN server instead.
- A TURN server is designed for this task.
- Reliability and scalability provided by anycast + load balancing mechanism.

Conclusion

- Discussed:
  - Benefits of IPv6 and why open-source PBXes benefit from being IPv6-enabled.
  - How to port an application to IPv6
  - Changes to FreeSWITCH
  - Lessons learned
  - VoIPv6 deployment
- Try IPv6 now! http://freenet6.net

Questions?

Contact info:

Simon.Perreault@viagenie.ca

This presentation is available at http://www.viagenie.ca/publications/

References

- [RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.
- [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.
- IPv6 Network Programming, Junichiro itojun Hagino, Elsevier, 2004, ISBN 1555583180.
- Migrating to IPv6, Marc Blanchet, Wiley, 2006, ISBN 0-471-49892-0, http://www.ipv6book.ca

NAT and Firewall Traversal with STUN / TURN / ICE

Simon Perreault

Viagénie

{mailto|sip}:Simon.Perreault@viagenie.ca

http://www.viagenie.ca

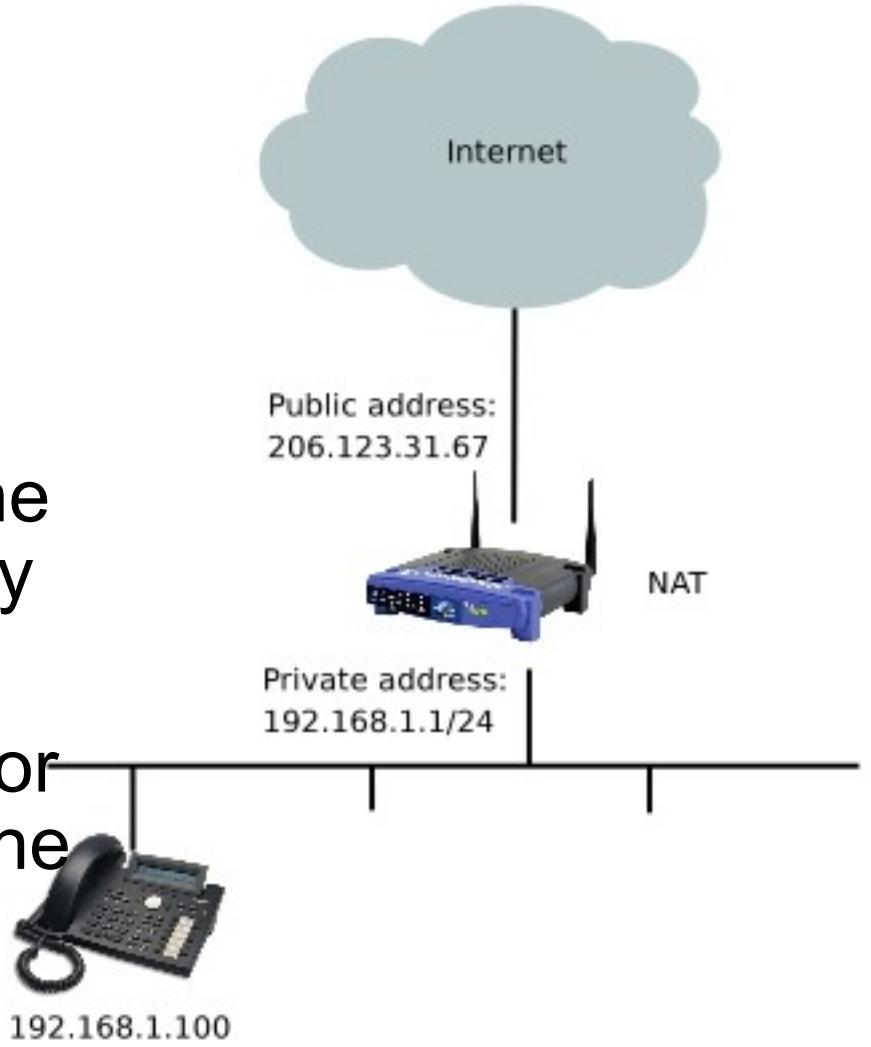
Plan

- The problem of NAT and firewalls in VoIP
- How STUN, TURN, and ICE solve it
- Asterisk specifics
- Wireshark traces

The Problem of NAT and Firewalls in VoIP

VIAGĒNIE

- Network address translators (NATs) are common devices that “hide” private networks behind public IP addresses.
- Connections can be initiated from the private network to the Internet, but not the other way around.
- Having separate addresses for signaling and media makes the situation worse.



Server-Reflexive Address

- A NAT device works by associating a public address and port with a private destination address and port.

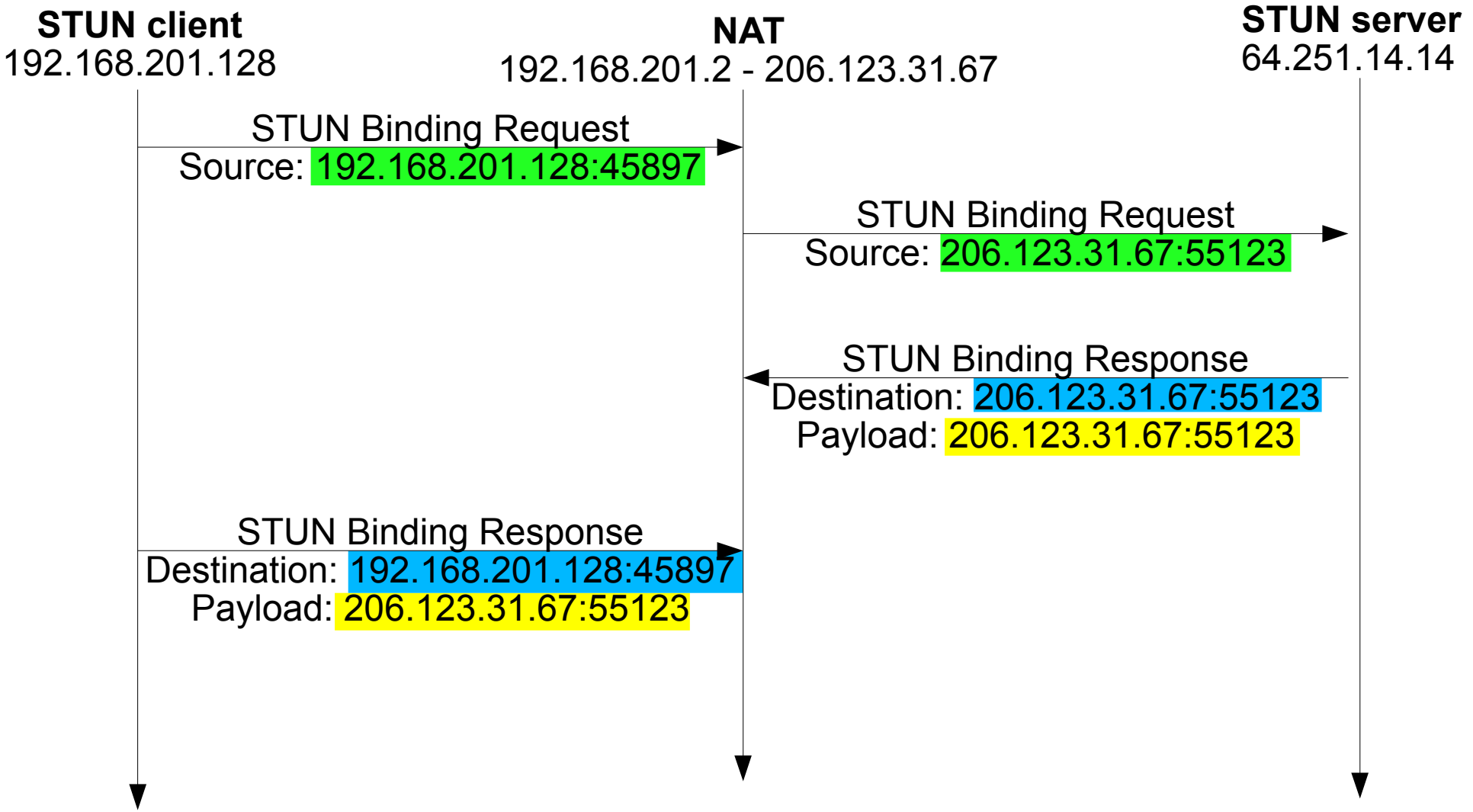
Public		**Private**
206.123.31.67 : 55123	↔	192.168.1.2 : 5060

- Valid for duration of flow
  - Meaning of “flow” for UDP?
  - Must be kept alive.
- Useful to discover this address.

STUN

- Session Traversal Utilities for NAT (STUN): simple protocol for discovering the server-reflexive address.
  - Client: Where do you see me at?
  - Server: I see you at 206.123.31.67:55123.
- A STUN server is located in the public Internet or in an ISP's network when offered as a service.
  - Double NATs pose an interesting problem...

STUN Flow Diagram



STUN

- It turns out that some NAT devices try to be clever by inspecting the payloads and changing all references to the server-reflexive address into the private address.
- STUN2 obfuscates the address by XORing it with a known value.
- TCP and UDP are supported over IPv4 and IPv6.

Server-Reflexive Address

- A client who knows its server-reflexive address could use it in place of its private address in the SIP headers.
  - Not the intended usage. See *sip-outbound* IETF draft.
- Intended usage: RTP ports.
- RTP port ⇒ NAT binding ⇒ STUN request

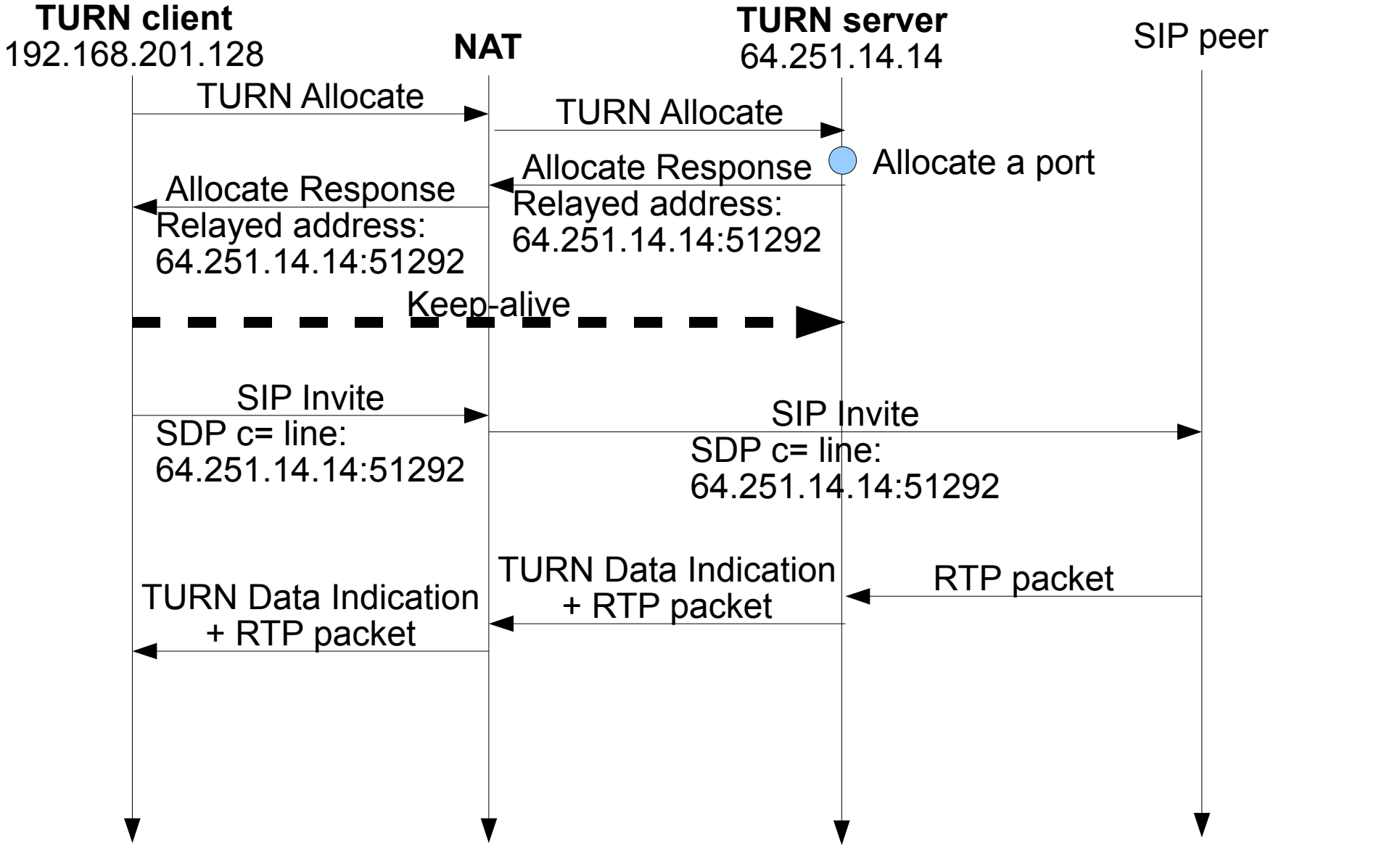
Symmetric NATs

- Some NAT devices only allow packets from the remote peer to reach the NATed peer.
  - Address dependent
  - Port dependent
  - Both
  - Implication: knowing server-reflexive address is useless.
- These NAT devices are called *symmetric NATs*.
  - Often “enterprise” NATs ⟹ many devices.
  - Significant presence, must be worked around.

TURN

- Makes devices behind symmetric NATs reachable.
  - Device initiates and maintains connection to relay.
- Traversal Using Relays around NAT (TURN)
  - Protocol between NATed device and relay.
  - Built on top of STUN.
- TURN server is located outside the NAT.
  - On the public Internet
  - or in an ISP's network when offered as a service by the ISP.

TURN Flow Diagram



Relayed Address

- The address allocated by the TURN server is called the *relayed address*.
  - TURN server communicates it to TURN client.
  - TURN client communicates it to SIP peer.
- The TURN client may use it in the SIP headers.
- Intended usage: RTP ports.
- RTP port ⇒ NAT binding ⇒ TURN allocation

- TURN **guarantees** communication in all NAT cases unless there is an explicit firewall policy to prohibit its use.

Disadvantages of TURN

- TURN server is in forwarding path.
  - Requires a lot of bandwidth.
  - Server must remain available for the whole duration of the allocation.
  - Triangle routing results in longer path.
- Encapsulation.
  - Lowers MTU (not so much a problem for VoIP packets).
  - Additional headers consume a bit more bandwidth.
  - Firewall must inspect payload to discover real sender.
- Allocation must be kept alive.

Disadvantages of TURN

- ICMP not relayed.
  - No path MTU discovery.
- TTL not properly decremented.
  - Possibility of loops.
- DiffServ (DS) field not relayed.
- As of now only IPv4 and UDP.

Mitigating Mechanisms

- Availability and scalability provided by anycast.
  - Only used for discovery, server must remain up for the duration of the allocation.
- Channel mechanism for minimizing header size.
  - 4 bytes only.
- Permission mechanism enforced by TURN server.
  - Only peers previously contacted by client may send data to relayed address.
  - Firewall may “trust” the TURN server, no payload inspection.
- Keep TURN server close to NAT device.
  - Offered as a service by ISPs.

IPv4 and IPv6 Interoperability

- TURN will also be used to relay packets between IPv4 and IPv6.
- Alleviates load from the B2BUA.
  - Designed for relaying performance.
  - Anycast ensures scalability and reliability.
- TURNv6 draft still in progress.

Numb

- Numb is a STUN and TURN server developed by Viagénie.
  - Supports IPv4 and IPv6 in mixed scenarios.
  - Supports anycast.
- Free access at http://numb.viagenie.ca
- To install it in your own network, contact us: info@viagenie.ca

Connectivity Establishment

- Many addresses may be available:
  - Host addresses.
  - Server-reflexive address.
  - Relayed address.
  - Each in IPv4 and IPv6 flavour!
  - Each in UDP and TCP flavour!
- Which one to choose?
- Need for an automatic *connectivity establishment* mechanism.

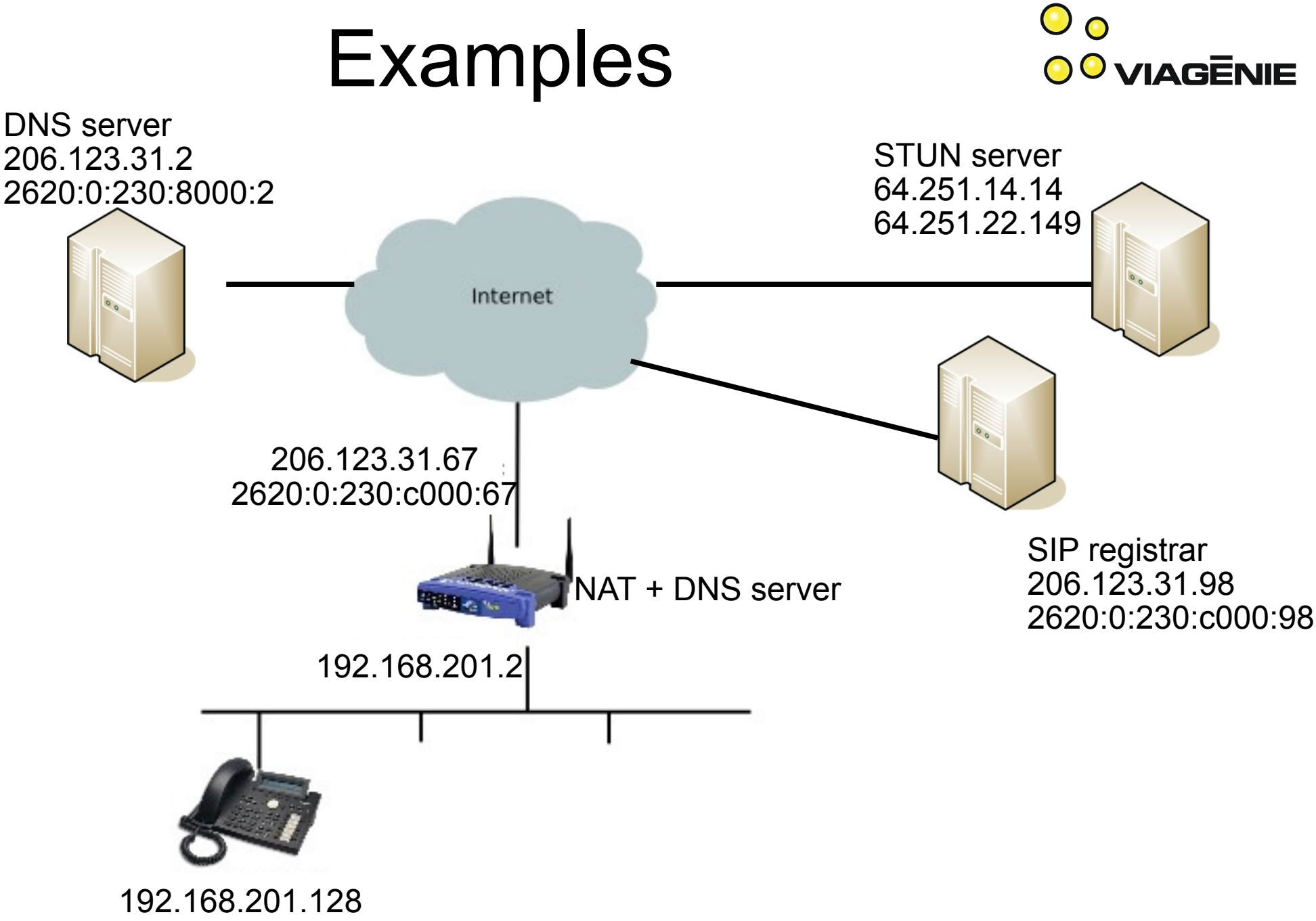
Interactive Connectivity Establishment (ICE)

- Conceptually simple.
  - Gather all *candidates* (using STUN/TURN).
  - Order them by priority.
  - Communicate them to the callee in the SDP.
  - Do connectivity checks.
  - Stop when connectivity is established.
- Gnarly details:
  - Keep candidates alive.
  - Agree on priority.
  - Reduce delays and limit packets.

Peer-Reflexive Address

- Remember: Server-reflexive address useless with symmetric NAT.
- Address as seen from peer (instead of STUN server) is *peer-reflexive address.*
  - Works even with a symmetric NAT.
    - ...but not two of them (TURN still necessary).
- During ICE connectivity checks, peer-reflexive candidates are gathered and prepended to check list.
- Information reuse between ICE instances.

Examples


DNS server
206.123.31.2
2620:0:230:8000:2
Internet
STUN server
64.251.14.14
64.251.22.149
206.123.31.67
2620:0:230:c000:67
NAT + DNS server
SIP registrar
206.123.31.98
2620:0:230:c000:98
192.168.201.2
192.168.201.128

Asterisk Specifics

- NAT traversal in 1.6 was greatly enhanced
  - Can define internal NATed network (*localnet*)
  - Can determine external address either...
    - directly (*externip*)
    - via dynamic DNS (*externhost*)
    - with a **STUN client** (*stunaddr*)
- RFC 3581 rport mechanism (*nat* = *yes*)
- Don't re-INVITE internal ↔ external calls (*canreinvite* = *nonat*)

Deployment

- ISPs are deploying STUN / TURN servers within their network.
- TURN a part of the IPv6 migration.
- SIP client vendors are implementing ICE.
- B2BUAs also should implement ICE.

Conclusion

- Discussed
  - The problem of NAT and firewalls in VoIP
  - How STUN, TURN, and ICE solve it
    - Obtaining a server reflexive address via STUN
    - Obtaining a relayed address via TURN
    - Telling the other party about these addresses via ICE
    - Making connectivity checks
    - Obtaining peer reflexive addresses
- STUN / TURN / ICE stack too thick? Use IPv6!

Questions?

Simon.Perreault@viagenie.ca

This presentation: http://www.viagenie.ca/publications/

STUN / TURN server: http://numb.viagenie.ca

References:

STUNv1 RFC: http://tools.ietf.org/html/rfc3489

STUNv2 draft: http://tools.ietf.org/html/draft-ietf-behave-rfc3489bis

TURN draft: http://tools.ietf.org/html/draft-ietf-behave-turn

ICE draft: http://tools.ietf.org/html/draft-ietf-mmusic-ice