

6to4

Automatic IPv6 tunnelling over IPv4

<http://www.braintrust.co.nz/presentations/20070809-nward-6to4-teredo.pdf>

6to4 notes

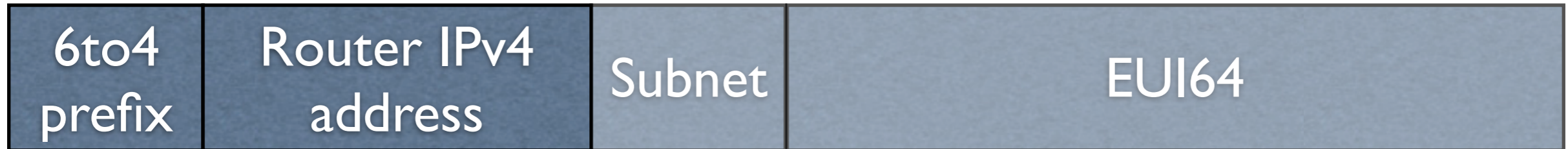
- RFC3056 and RFC3068
- Requires a single public IPv4 address, and gives a 48-bit prefix (65536 /64s - 1208925819614629174706176)
- Site border router (ie. NAT box, firewall, etc.) needs to support 6to4 - most IPv6 capable hardware does.
- Multipoint-to-multipoint, between border routers of different networks.
- Allows sites to run native IPv6 internally - no specific 6to4 support required on hosts - just normal IPv6.

6to4 notes

- 6to4 addresses in the form 2002:AABB:CCDD:: where AA.BB.CC.DD is the hex form of the public IPv4 address.
- Only 'nodes' exist, no servers to configure - just set it up on your border, and you're away.
- '6bone' connected 6to4 nodes can relay traffic for non-6bone connected 6to4 nodes, and their networks.
- Makes use of anycast for efficient routing - RFC3068 defines a public IPv4 anycast prefix for relays:
192.88.99.0/24 (use .1)

6to4 address format

← 16bit → ← 32bit →

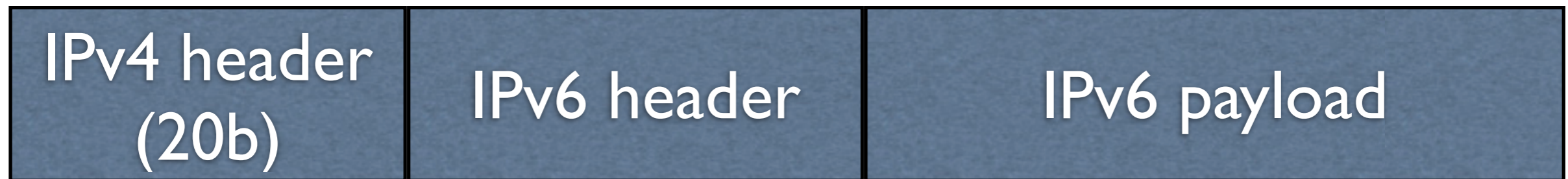


2002

- Subnetting can be done however you like, the above subnet/EUI64 is a common example.

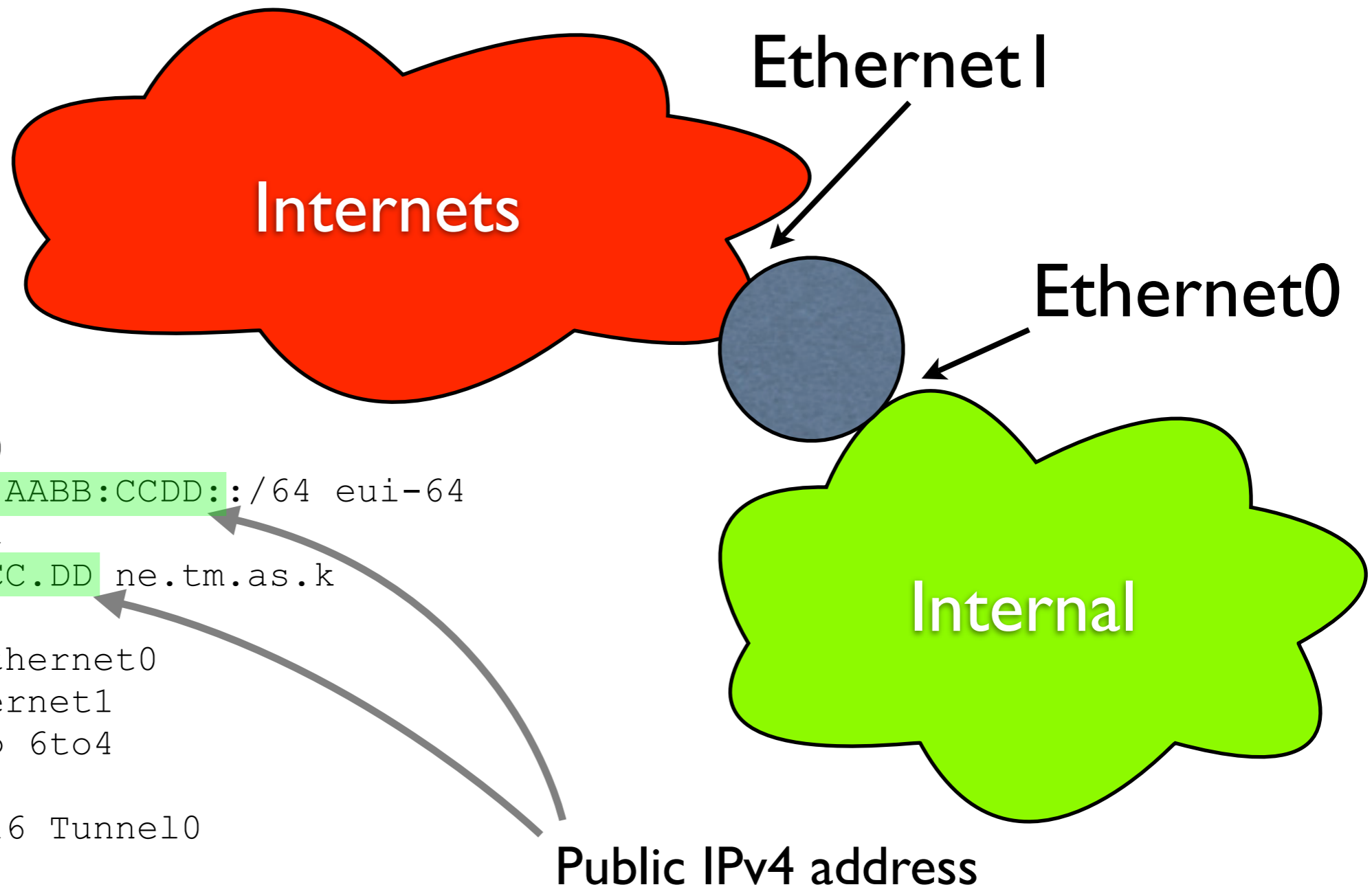
6to4 packet

Super simple - just wrap an IPv6 packet in an IPv4 header



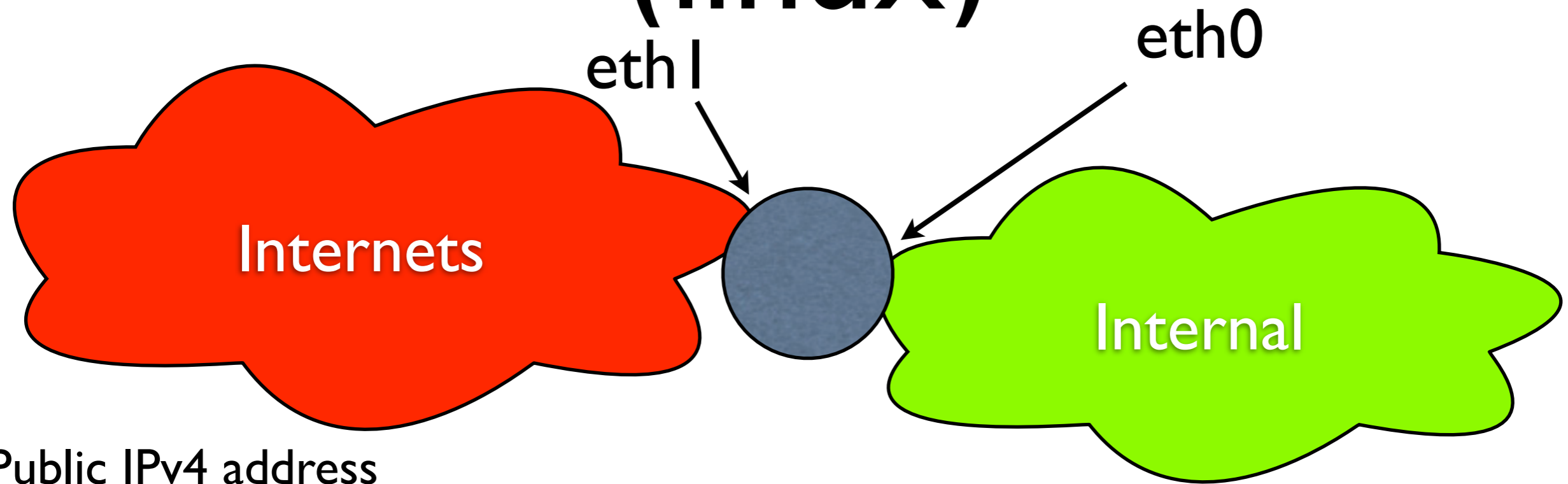
IP protocol 0x29 (ipv6)

6to4 configuration (cisco)



```
interface Ethernet0
  ipv6 address 2002:AABB:CCDD::/64 eui-64
interface Ethernet1
  ip address AA.BB.CC.DD ne.tm.as.k
interface Tunnel0
  ipv6 unnumbered Ethernet0
  tunnel source Ethernet1
  tunnel mode ipv6ip 6to4
!
ipv6 route 2002::/16 Tunnel0
```

6to4 configuration (linux)



Public IPv4 address

```
ip addr add AA.BB.CC.DD dev eth1
ip tunnel add tun6to4 mode sit remote any local AA.BB.CC.DD
ip link set dev tun6to4 mtu 1280 up
ip -6 addr add 2002:AABB:CCDD::/16 dev tun6to4
ip -6 route add ::/96 dev tun6to4 metric 1
ip -6 route add 2000::/3 via ::192.88.99.1 dev tun6to4 metric 1
```

6to4 configuration (relay, Linux/Cisco)

- Same as the prev two slides - remember, all nodes are equal - no servers/clients.
- But, has native (or non 6to4) IPv6 connectivity.
- Advertise 2002::/16 to IPv6-land.
- Routers using your relay configure you as their 2000::/3 route, or
- Advertise 192.88.99.0/24, configure 192.88.99.1 on your relay router.

6to4 configuration (Linux notes)

- Many distro's have automatic tunnelling configurable in config files.
- Redhat (maybe Debian GNU/Linux can do it, I dunno):
- `/etc/sysconfig/network-scripts/ifcfg-eth0:`

```
DEVICE=eth1
ONBOOT=yes
BOOTPROTO=static
IPADDR=202.74.202.114
NETMASK=255.255.255.252
GATEWAY=202.74.202.113
HWADDR=00:10:5A:61:7B:2C
IPV6INIT=yes
IPV6TO4INIT=yes
IPV6TO4_ROUTING="eth0-:1::1/64"
IPV6_CONTROL_RADVD=yes
```

Magic to configure
internal addresses
based on our
external IPv4 address


Magic to control
rtadvd when
interface state
changes

6to4 -> 6to4 process

- Router receives native IPv6 packet on it's 'internal' interface:
src = 2002:ca14:6187::219:e3ff:fed3:6148
dst = 2002:ca4a:ca72::1
- Router does a FIB lookup for dst address, gets directly connected on 6to4 virtual interface.
- Router considers ca4a:ca72 to find IPv4 address to send data to: 202.74.202.114
IPv4 source address doesn't matter.
- Router encapsulates entire IPv6 packet in an IPv4 packet, and transmits.

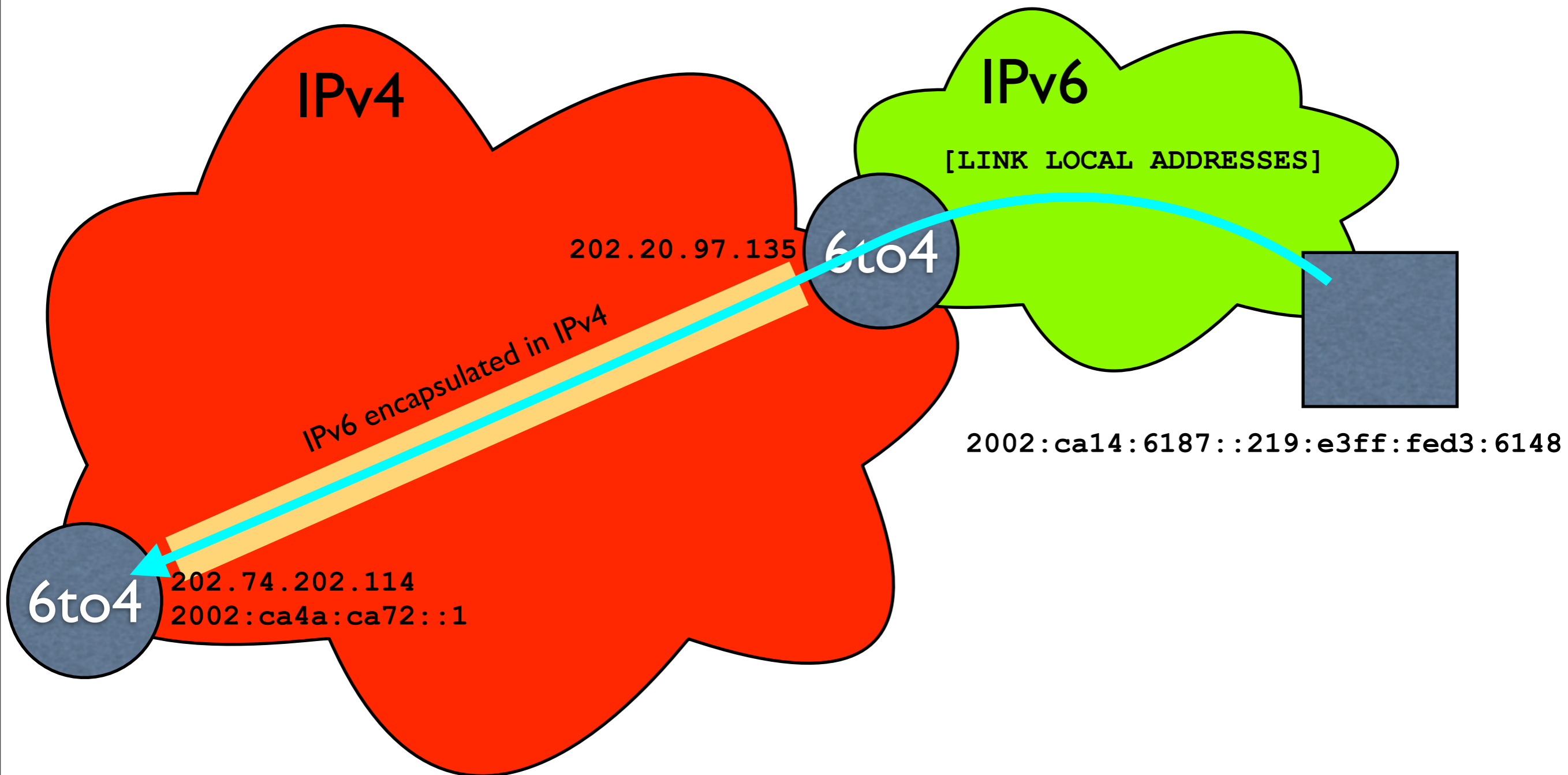
6to4 -> 6to4 process

```
Ethernet II, Src: Axxceler_0b:02:22 (00:c0:69:0b:02:22), Dst: 3com_61:7b:2c (00:10:5a:61:7b:2c)
Internet Protocol, Src: 202.20.97.135 (202.20.97.135), Dst: 202.74.202.114 (202.74.202.114)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 76
  Identification: 0x12c6 (4806)
  Flags: 0x00
  Fragment offset: 0
  Time to live: 24
  Protocol: IPv6 (0x29)
  Header checksum: 0xcf6a [correct]
  Source: 202.20.97.135 (202.20.97.135)
  Destination: 202.74.202.114 (202.74.202.114)
Internet Protocol Version 6
  Version: 6
  Traffic class: 0x00
  Flowlabel: 0x00000
  Payload length: 16
  Next header: ICMPv6 (0x3a)
  Hop limit: 63
  Source address: 2002:ca14:6187::219:e3ff:fed3:6148 (2002:ca14:6187::219:e3ff:fed3:6148)
  Destination address: 2002:ca4a:ca72::1 (2002:ca4a:ca72::1)
Internet Control Message Protocol v6
```



(magical hex->dotted-decimal conversion)

6to4 -> 6to4 diagram



6to4 -> native process

- Router decides to transmit a packet to a non-6to4 connected IPv6 address.
src = 2002:ca4a:ca72::1
dst = 2001:2f0:104:1:2e0:18ff:fea8:16f5
- Router does a FIB lookup for dst address, gets nexthop of 2002:c058:6301::, which is directly connected on 6to4 virtual interface.
- Router considers c058:6301 to find IPv4 address to send data to: 192.88.99.1
IPv4 source address doesn't matter.
- Router encapsulates entire IPv6 packet in an IPv4 packet, and transmits.

6to4 -> native process

Ethernet II, Src: 3com_61:7b:2c (00:10:5a:61:7b:2c), Dst: Axxceler_0b:02:22 (00:c0:69:0b:02:22)

Internet Protocol, Src: 202.74.202.114 (202.74.202.114), Dst: 192.88.99.1 (192.88.99.1)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

Total Length: 124

Identification: 0x0000 (0)

Flags: 0x04 (Don't Fragment)

Fragment offset: 0

Time to live: 64

Protocol: IPv6 (0x29)

Header checksum: 0x8242 [correct]

Source: 202.74.202.114 (202.74.202.114)

Destination: 192.88.99.1 (192.88.99.1)

Internet Protocol Version 6

Version: 6

Traffic class: 0x00

Flowlabel: 0x00000

Payload length: 64

Next header: ICMPv6 (0x3a)

Hop limit: 64

Source address: 2002:ca4a:ca72::1 (2002:ca4a:ca72::1)

Destination address: 2001:2f0:104:1:2e0:18ff:fea8:16f5 (2001:2f0:104:1:2e0:18ff:fea8:16f5)

Internet Control Message Protocol v6

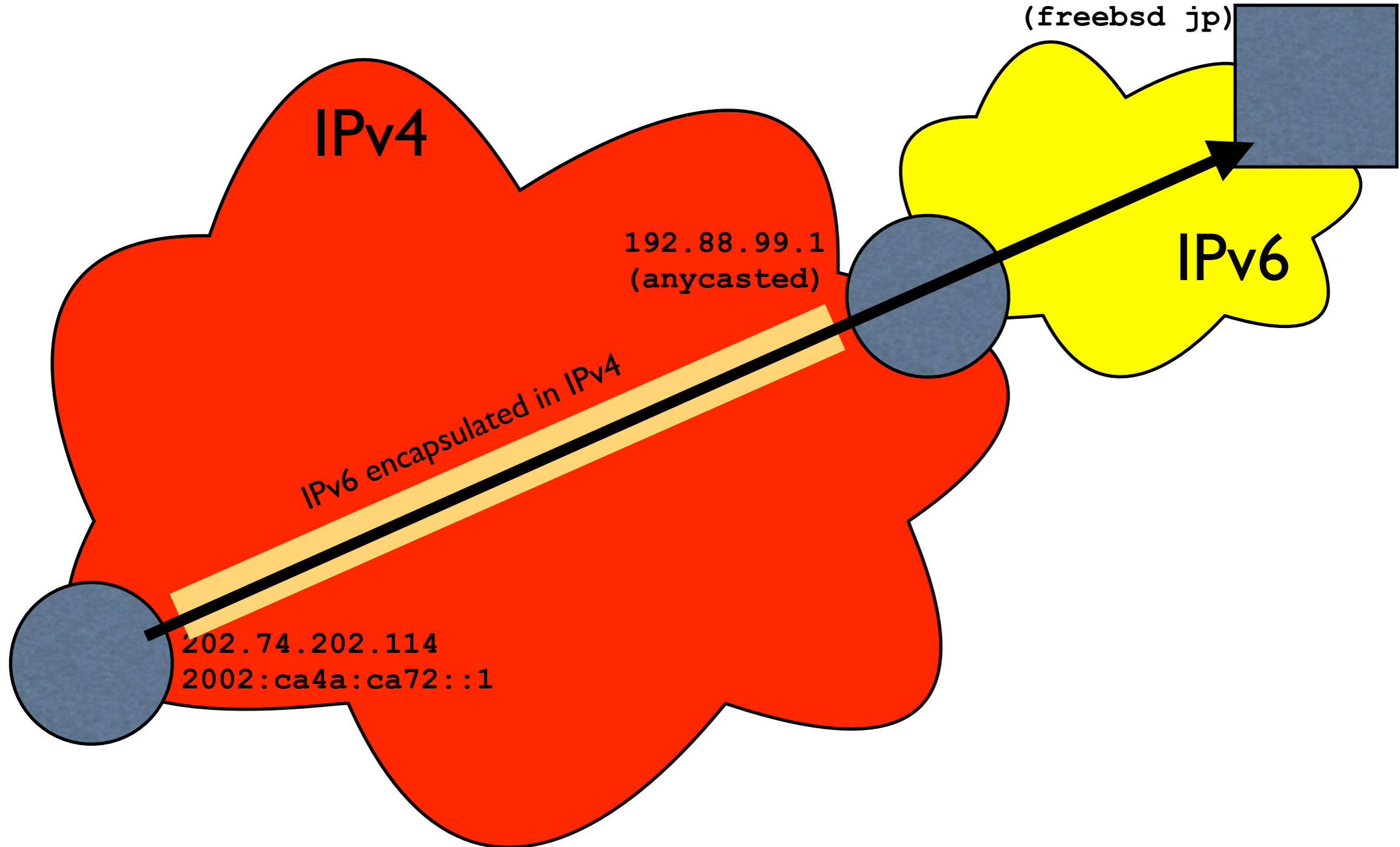
(magical hex->dotted-decimal conversion)

2000::/3 via 2002:c058:6301::
2002::/16 via 6to4

6to4 -> native diagram

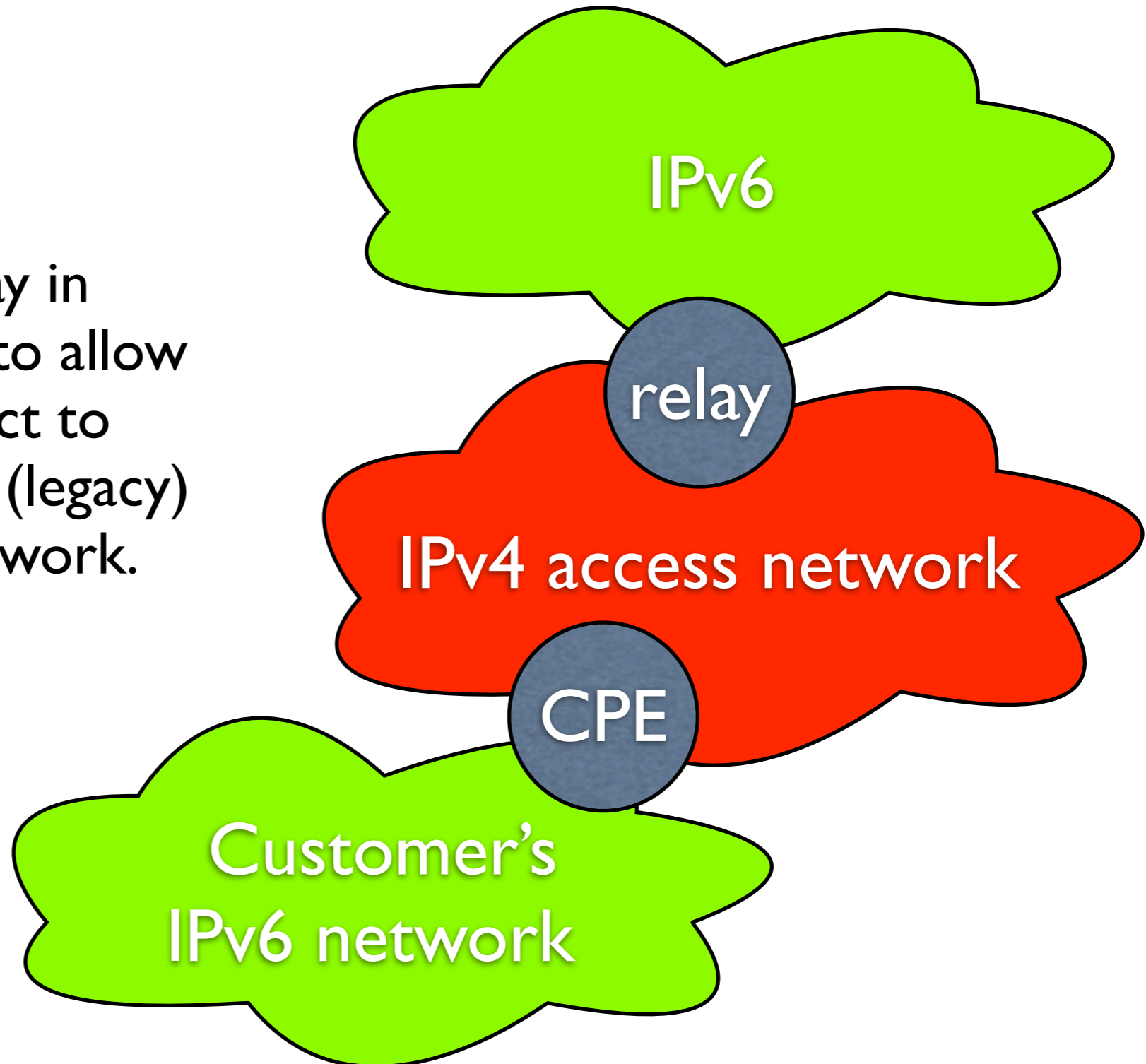
2001:2f0:104:1:2e0:18ff:fea8:16f5

(freebsd jp)



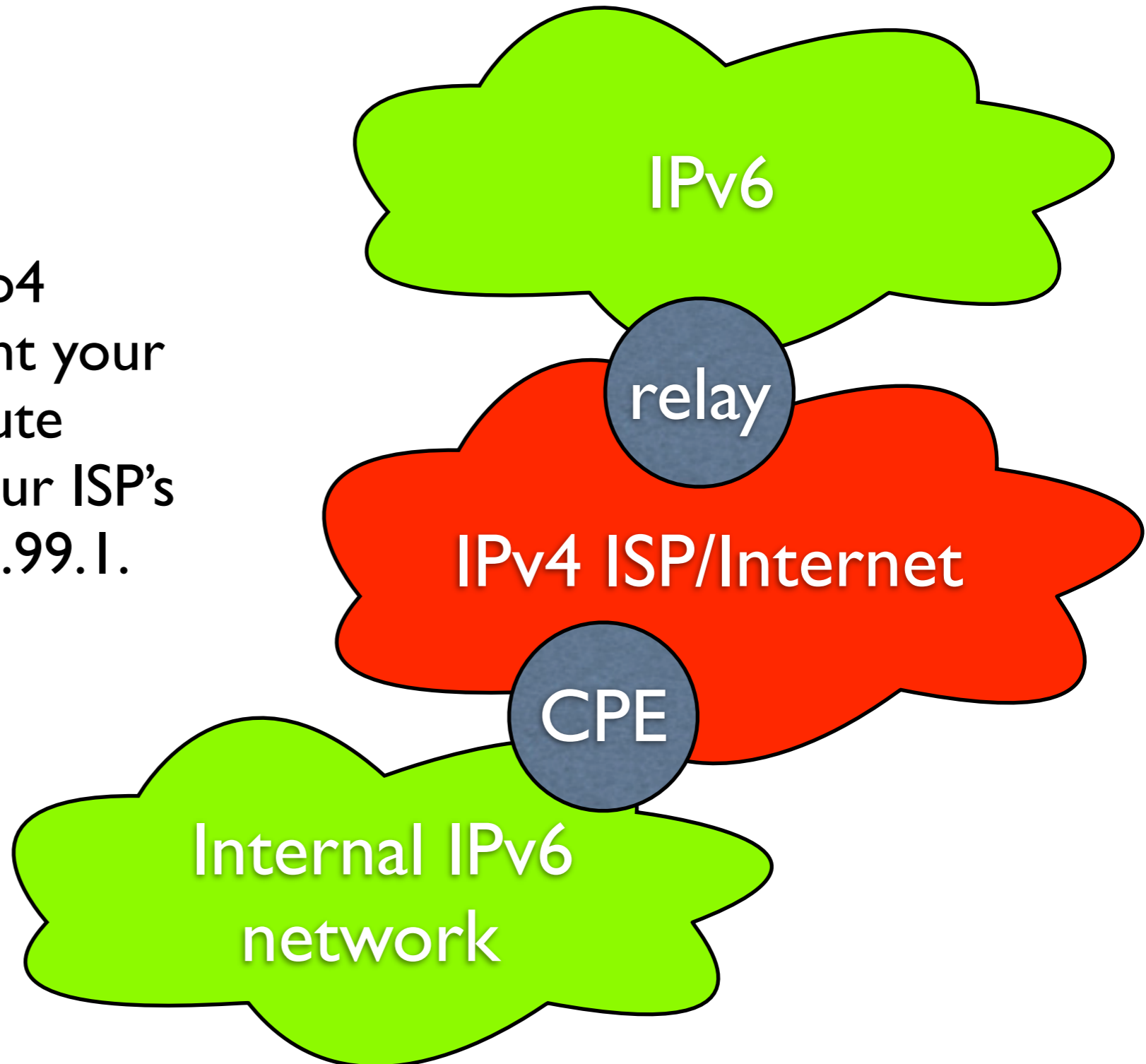
Deploying 6to4

- ISPs:
 - Run a 6to4 relay in your network, to allow users to connect to IPv6 over your (legacy) IPv4 access network.



Deploying 6to4

- End sites:
- Configure a 6to4 router, and point your default IPv6 route (2000::/3) at your ISP's relay, or 192.88.99.1.



Teredo

Automatic IPv6 tunnelling through IPv4 NAT

<http://www.braintrust.co.nz/presentations/20070809-nward-6to4-teredo.pdf>

Teredo notes

- IETF protocol, by Christian Huitema (MS), RFC4380.
- Gives hosts behind NAT a single (::/128) IPv6 address in 2001::/16.
- Encapsulates IPv6 in UDP.
- Fairly complicated setup mechanism - but can get around most NAT problems.
- No router support is required - only host software.

Teredo notes

- Clients talk to servers to set up addresses, detect NAT, etc.
- Relays do the heavy lifting.
- Servers used to help set up sessions with hosts behind restricted NAT - cone NAT goes direct.
- Servers and relays are stateless - the only stuff that requires state is either client-side, or encoded in the IPv6 address.
- Uses FancyTricks to locate the best Teredo relay for each IPv6 endpoint.

Teredo notes

- Several different node types:
 - Server
 - Relay
 - Host-specific relay
 - Client
- Origin Identifiers sometimes included in encapsulation to indicate IPv4 addresses and ports.
- Bubble packets (just IPv6 header) sometimes sent. Periodically to maintain NAT state mappings, and also for discovery processes.

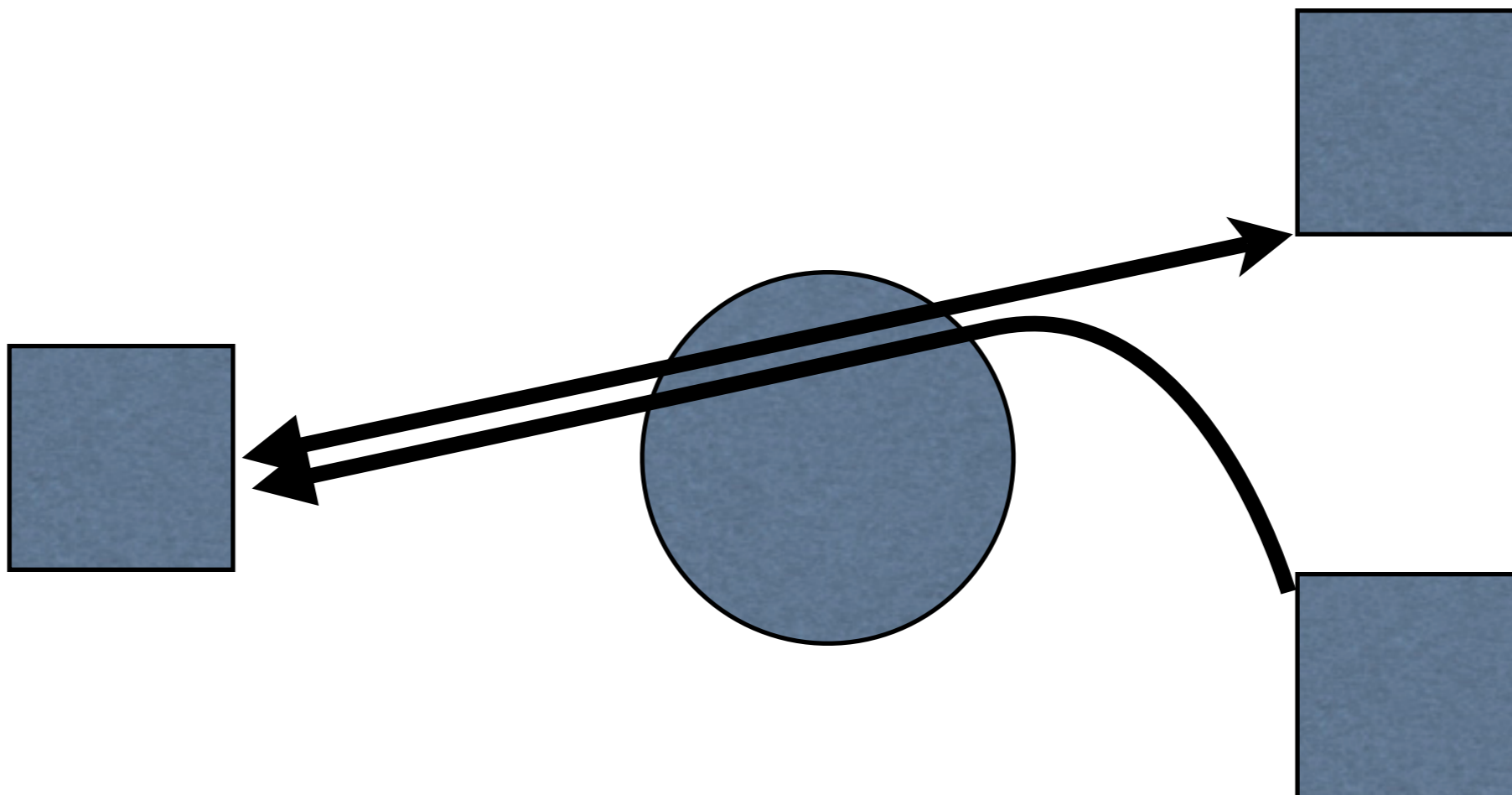
Teredo address format



- Client IPv4 address and port are XORed with 0xFFFFFFFF and 0xFFFF respectively, so NATs don't try to swap them.
- Flags show type of NAT, and scope.
 - Bit 1 - Cone NAT if set to 1
 - Bits 6-7 - Scope, always set to 00

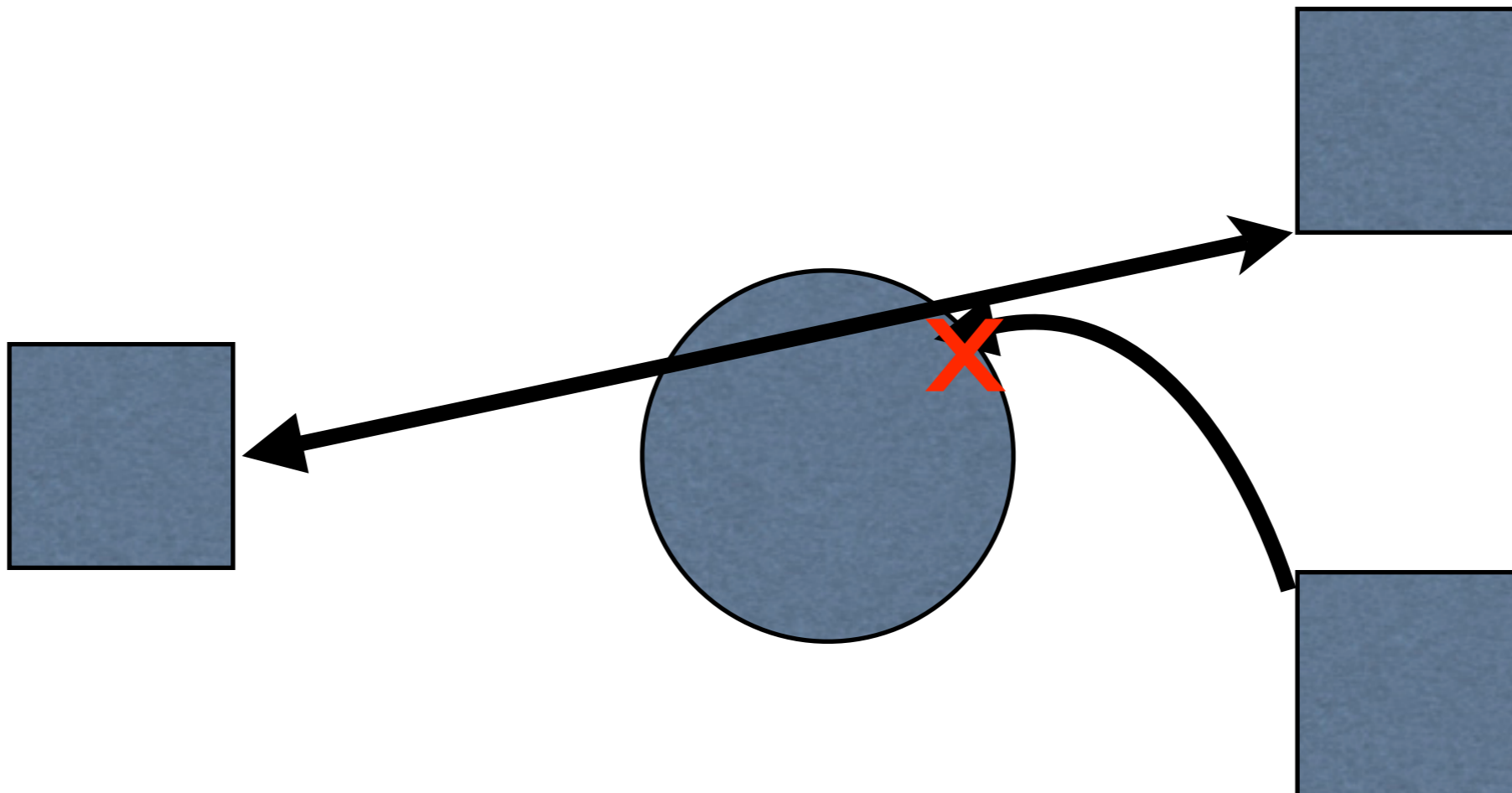
Cone NAT

- Outgoing packet sets up a dynamic mapping in NAT router.
- NAT router allows incoming connections to that mapping's port from ANY external host.



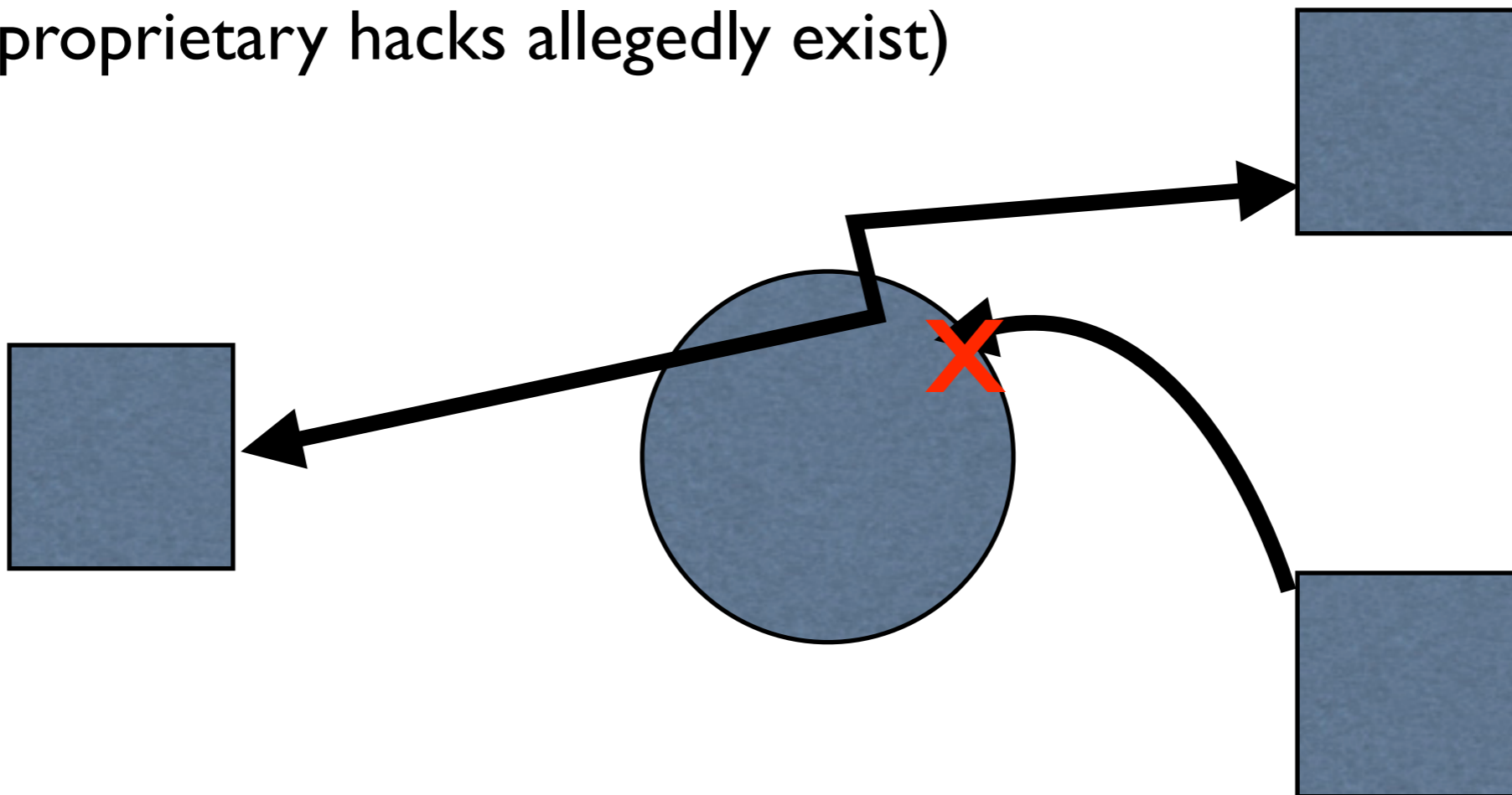
Restricted NAT

- Outgoing packet sets up a dynamic mapping in NAT router.
- NAT router allows incoming connections to that mapping's port from **ONLY** the original external host.



Symmetric NAT

- Like restricted NAT, but rewrites the source port of outgoing packets - new source port for every connection.
- Does not work with standard Teredo! (though some proprietary hacks allegedly exist)



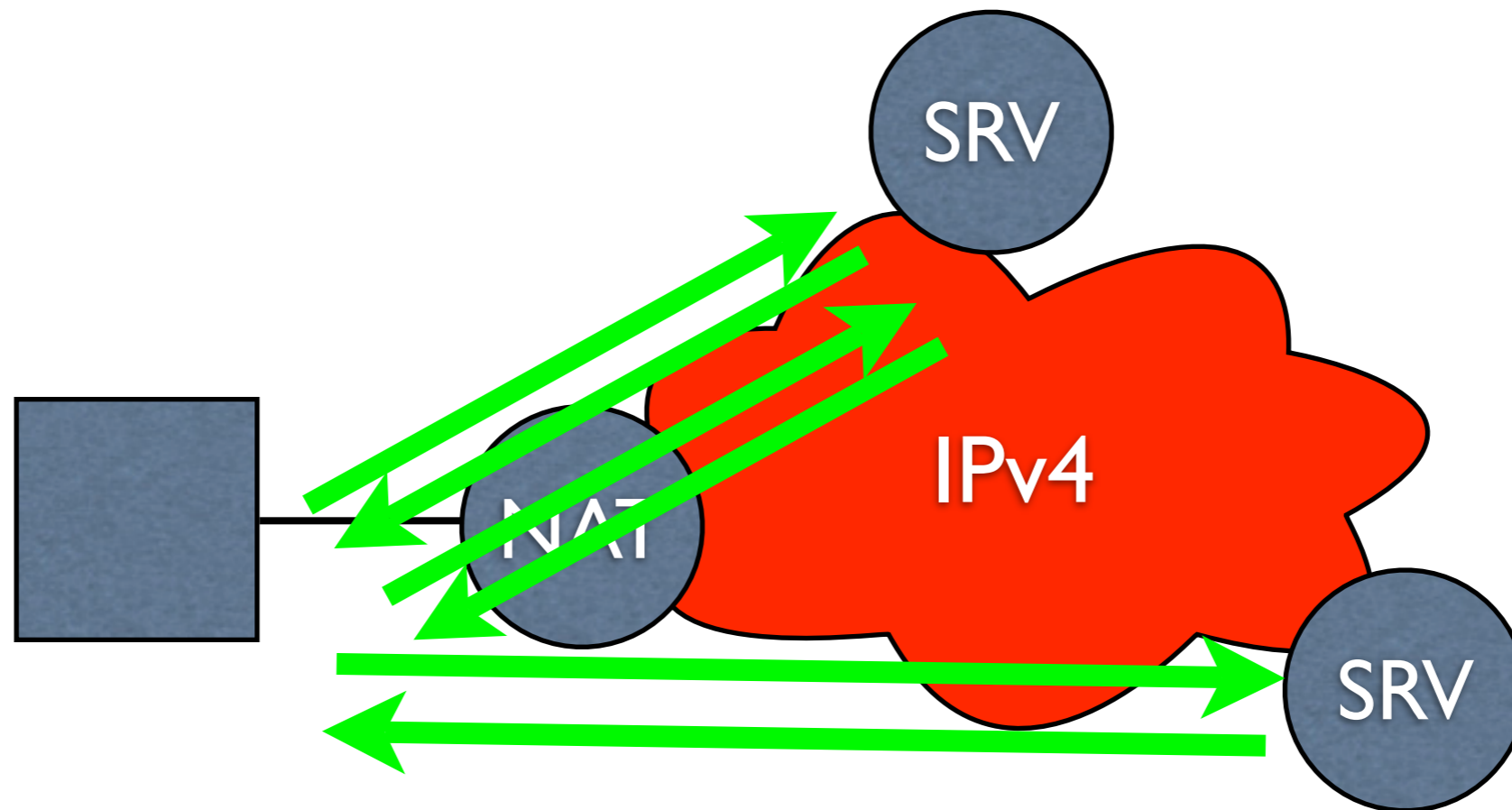
Teredo setup

Cone NAT

1. Client sends RS to server, with cone flag set.
2. Server responds with RA from a different IPv4 address (to detect cone NAT), with origin indicator details from IPv4.
3. Client sends RS to server without cone flag set.
4. Server responds with RA from same IPv4 address, with origin indicator details from IPv4.
5. Client sends RS to a second server without cone flag set.
6. Server responds with RA from same IP address, with origin indicator details from IPv4.
7. Client compares origin indicators to detect NAT type and uses this and the external IPv4 from the origin indicator to set a globally reachable Teredo address.

Teredo setup

Cone NAT



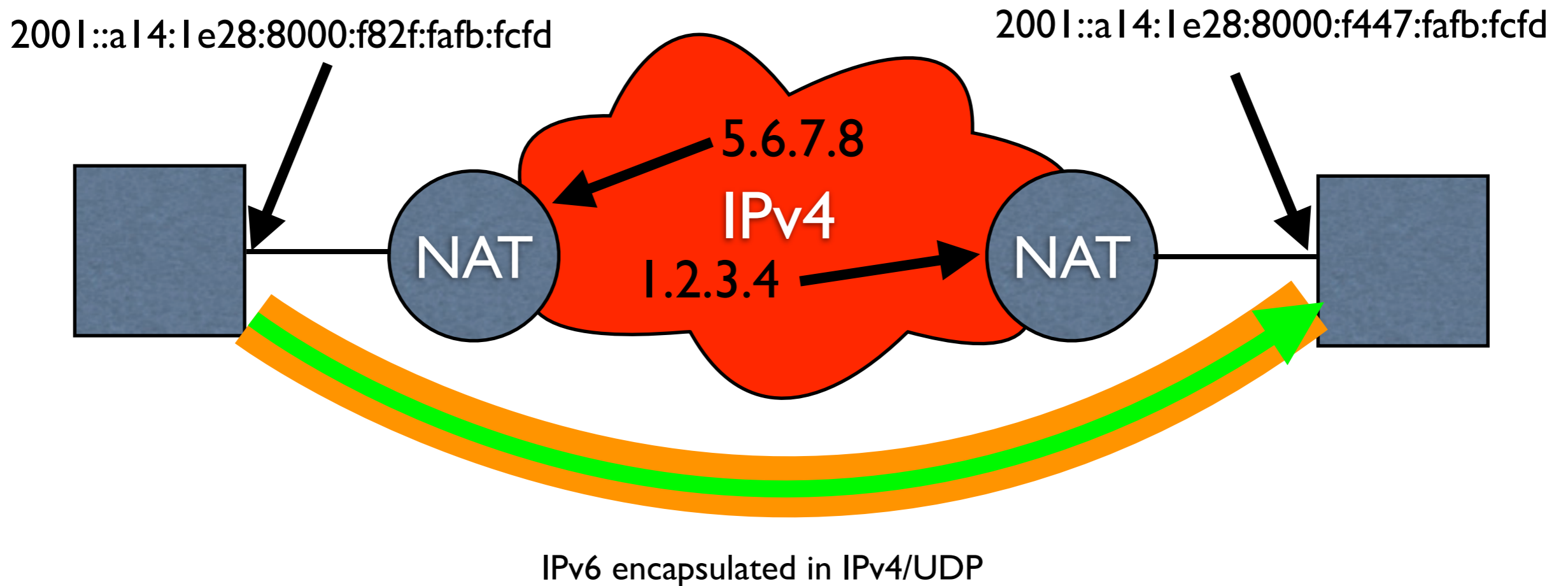
Teredo process

Teredo -> Teredo (cone)

1. Client decides to send packet to another Teredo address (2001::/32).
2. Client considers flags embedded in destination Teredo address, and sees that it is a cone NAT.
3. Client puts an IPv6 packet in a UDP/IPv4 packet, destined to the address and port of the destination, as encoded in the Teredo address. Sends.
4. Destination NAT forwards packet on to destination host.
5. Destination host takes IPv6 packet out, and stuffs it in to its IPv6 stack.

Teredo process

Teredo -> Teredo (cone)



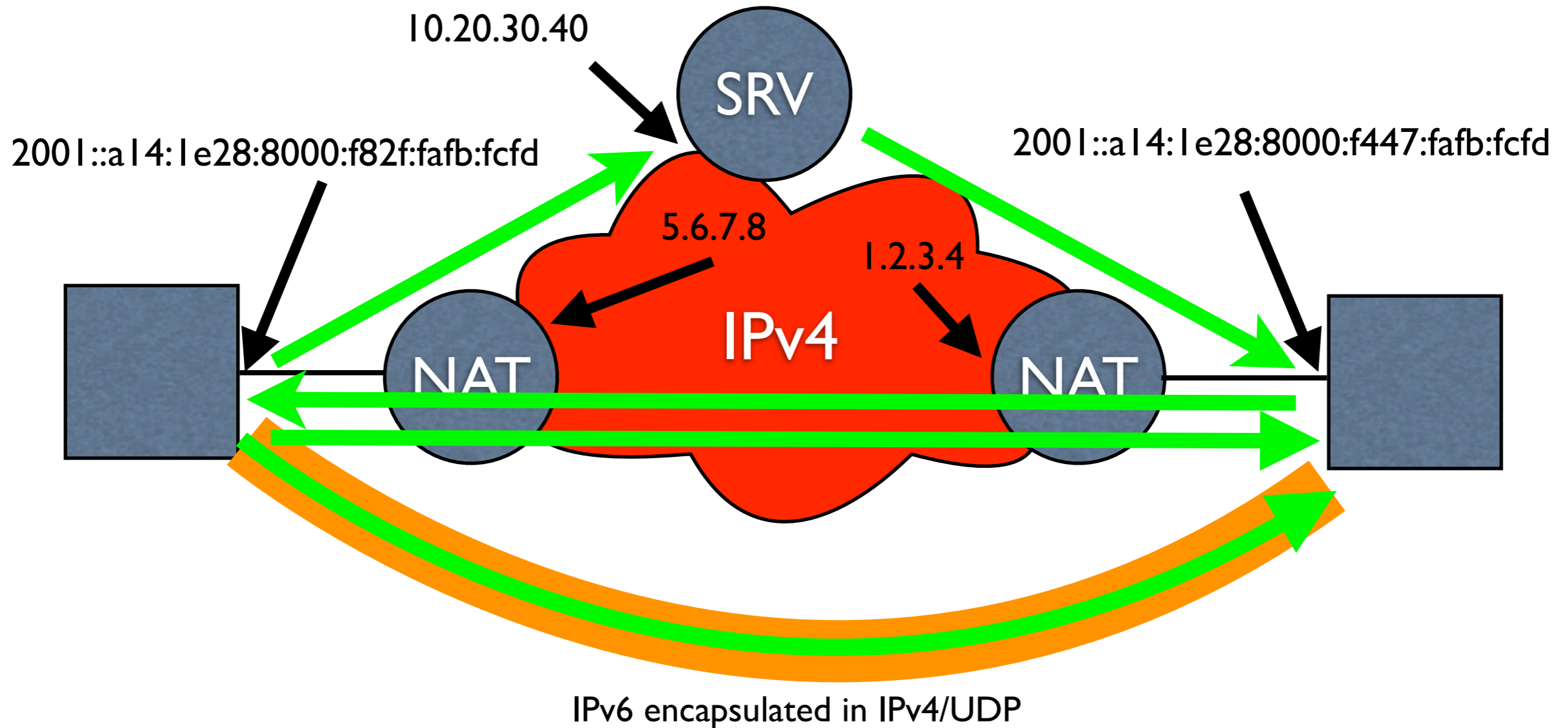
Teredo process

Teredo -> Teredo (restricted)

1. Client decides to send packet to another Teredo address (2001::/32).
2. Client considers flags embedded in destination Teredo address, and sees that it is a restricted NAT. It is blocked/dropped by the destination NAT (as intended!).
3. Client puts a bubble packet in a UDP/IPv4 packet, destined to the address of the destination's IPv4 address and port, as encoded in the Teredo address. Sends.
4. Client puts a bubble packet in a UDP/IPv4 packet, destined to the address of the destination's Teredo server, as encoded in the Teredo address. Sends.
5. Destination's server takes the bubble, and re-encapsulates it in a UDP/IPv4 packet destined for the destination's port and address as encoded in the destination's IPv6 address. Sends.
6. Destination host takes the bubble, and responds by encapsulating a new bubble packet in UDP/IPv4, destination address set to the IP address and port of the sender. Sends.
7. Sender's NAT forwards packet to sender. (Mapping is set up in step #2)
8. Sender receives bubble. Ports are now open!
9. Sender encapsulates the original IPv6 packet in UDP/IPv4, destination set to destination's IPv4 address and port.
10. Destination's NAT now has a mapping, so forwards to the destination host.
11. Destination host takes out the IPv6 packet, and stuffs it in to its IPv6 stack.

Teredo process

Teredo -> Teredo (restricted)



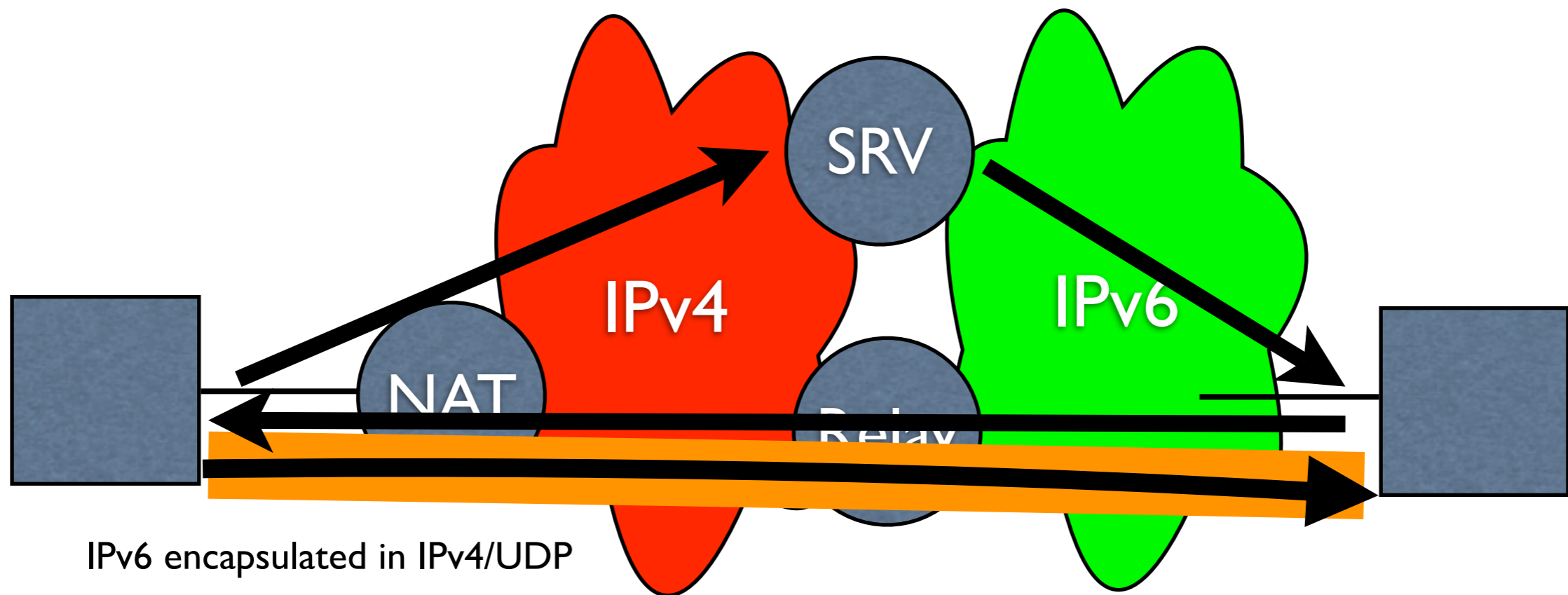
Teredo process

Teredo -> native (cone)

1. Client decides to send packet to a non-Teredo address.
2. Client sends an ICMPv6 echo request encapsulated in UDP/IPv4 to its Teredo server.
3. Teredo server forwards this ICMPv6 packet to its native IPv6 connectivity.
4. Destination responds with ICMPv6 echo response, routed to its best path to 2001:0000::/32
5. Relay notes that the cone NAT flag is set, and encapsulates ICMPv6 response packet in UDP/IPv4, destined for the client's port and address as encoded in the destination's IPv6 address.
6. Client's NAT forwards this packet to the client.
7. Client decapsulates, and takes note of the IPv4 source address and port of the encapsulated packet.
8. Client encapsulates intended traffic in UDP/IPv4, using the IPv4 destination address and port taken from step 7. Sends.

Teredo process

Teredo -> native (cone)



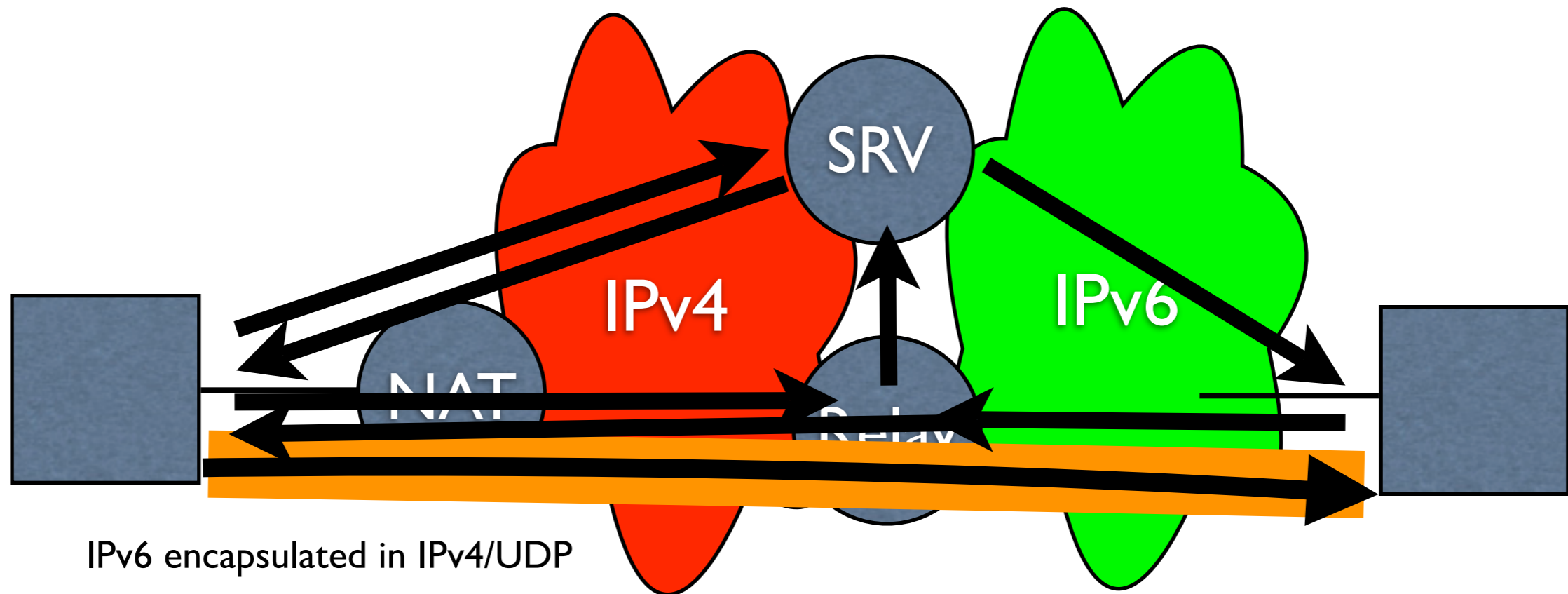
Teredo process

Teredo -> native (restricted)

1. Client decides to send packet to a non-Teredo address.
2. Client sends an ICMPv6 echo request encapsulated in UDP/IPv4 to its Teredo server.
3. Teredo server forwards this ICMPv6 packet to its native IPv6 connectivity.
4. Destination responds with ICMPv6 echo response, routed to its best path to 2001:0000::/32
5. Relay notes that the cone NAT flag is NOT set, and sends a bubble packet destined to the client's IPv6 address, and Teredo server's IPv4 address as taken from the client's IPv6 address.
6. Server forwards to client, using client's IPv4 address and port taken from client's IPv6 address, adding a Teredo bubble packet, indicating the relay's IPv4 address and UDP port.
7. Client's NAT allows this through, as NAT mapping exists for the Teredo server's IPv4 address and UDP port.
8. Client decapsulates the bubble, and notes the IPv4 address and UDP port in the origin indicator.
9. Client sends a bubble to the relay's IPv4 address and UDP port taken from the bubble in step 6.
10. Relay now knows that the NAT mapping is open, so encapsulates the ICMPv6 response packet in IPv4/UDP, using address and port taken from client's IPv6 address.
11. Client's NAT has a NAT mapping for the relay's IPv4 address and port, so forwards encapsulated ICMPv6 response to client.
12. Client decapsulates, and notes that the NAT mappings are working correctly.
13. Client encapsulates intended traffic in UDP/IPv4, using the IPv4 destination address and port of the best relay, taken from origin indicator in step 8. Sends.

Teredo process

Teredo -> native (restricted)



Teredo process

Native -> Teredo

- More or less the same as Teredo -> Native.
- But without the Teredo node forcing the native node to respond with ICMPv6 Echo Request.
- The native node just sends it's intended packet, and the relay/server do NAT traversal etc. if required.
- Sorry, no pictures.

Teredo configuration

- Miredo implementation for Linux/BSD
 - Client, Server, and Relay.
- Windows XP SP2 onwards.
 - Enabled by default in Vista - just try connect to a v6 host, and it will turn itself on.

Teredo Security

- Not intended for use in enterprise environments.
- Teredo has had a lot of criticism by enterprise security people.
- Really, Teredo doesn't do anything that you couldn't do before with an SSH (or similar) tunnel.
- Though it does put you on an unfiltered, globally reachable IPv6 address.
 - But in Windows, it won't turn on unless the Windows Firewall is enabled.

Teredo Security

- So, block Teredo ports at your border if you're a worried enterprise.
 - Or better, just don't open it up - you block everything already, right?
- Some privacy concerns, as servers are (so far) just run by vendors (MS + Miredo), as they require configuration on the client to change. This may change slightly.
 - Anycast
 - Overloaded DNS name

Teredo Deployment

- Content providers intending to deploy IPv6 should set up a relay.
- ISPs with IPv6 native, or 6to4 relays, should deploy a relay.
- In the future, you should deploy a server, but for now there's no good way to tell your customers to use your new server, so just use the defaults.

Transition protocols wrap-up (users)

- 6to4 - When your border router supports 6to4, and you want to run native IPv6 internally. IPv4 NAT is ok, but requires one public address on the router.
- Teredo - When your border router does not support 6to4, and/or you don't want native IPv6 internally. IPv4 NAT is good! Most NATs work fine.
- ISATAP for internal stuff, if you need IPv6 internally, but can't upgrade all your routers to native IPv6.

Transition protocol wrap-up (providers)

- Run relays. 6to4 and Teredo.
- If you're nice, anycast your 6to4's IPv6 and IPv4, and your Teredo relay's IPv6 prefixes to your peers.
 - You will pay for their traffic!
- Teredo relays go as close to native servers as possible.
- 6to4 relays go as close to both clients and servers as possible.
- Relays are important for both content providers and ISPs.