IPv6, the Internet, and Google

what, why, how, huh?

Erik Kline
IPv6 Software Engineer

Talk TOC

Google

- IPv6 Background
- Motivation
  - Why bother with IPv6?
- IPv6 at Google
  - What have we done, exactly?
- IPv6 and the 3 Tier Architecture
  - Simple strategies for deploying IPv6
- Software Issues
  - IPv4-isms
  - IP-Agnosticism
- Final Thoughts

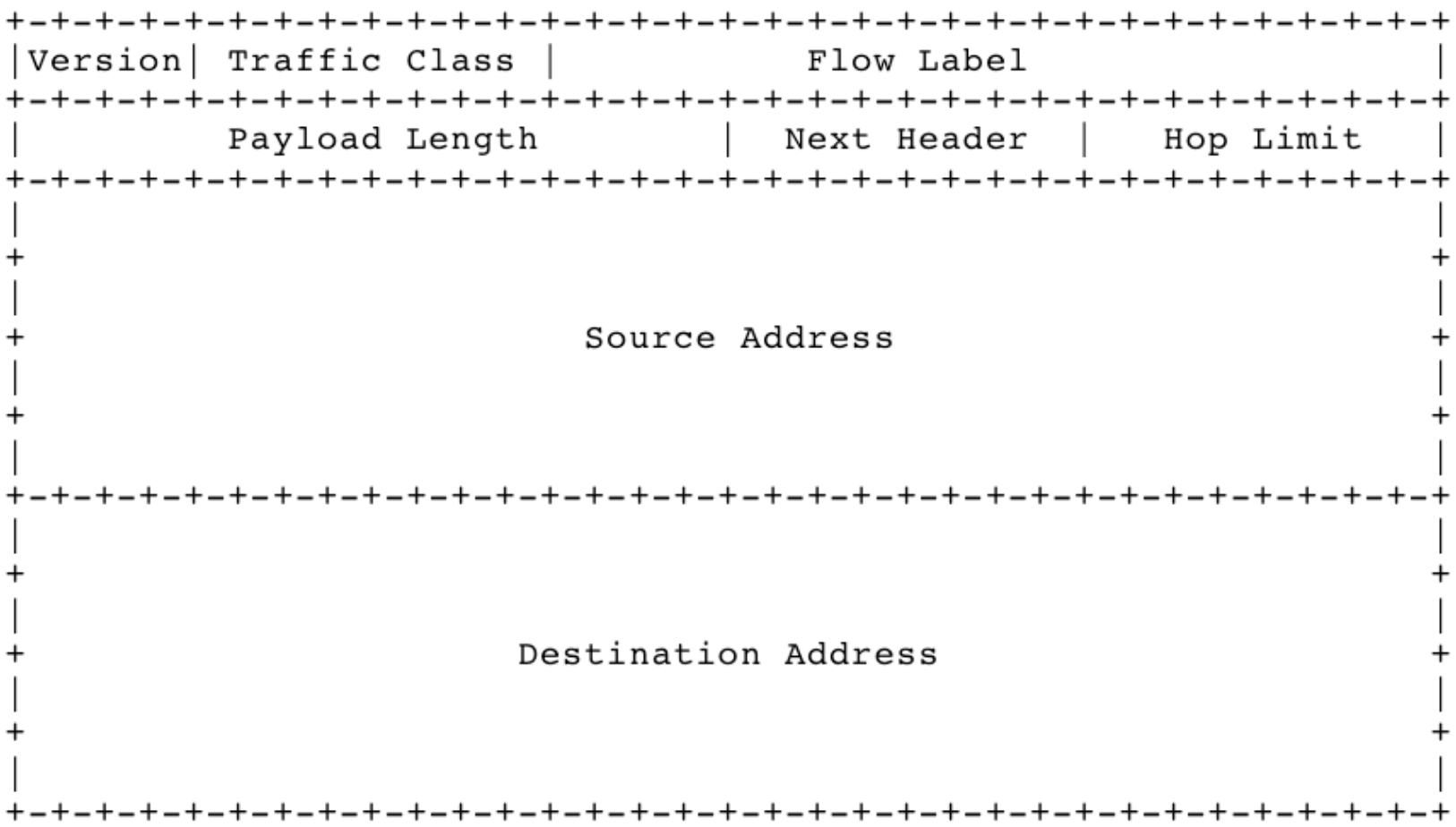
Google

not just IPv4 with bigger addresses

IPv6 Background

The Obligatory IPv6 Header Slide

Google



Version 4-bit Internet Protocol version number = 6.

http://tools.ietf.org/html/rfc2460

Differences: Some Obvious (beside 2^{128})

Google

Hop Limit versus TTL

- same effective meaning
- link-only messages have HL=255 (kinda like an unsigned -1)

No header checksum

- defined pseudo-header checksum
- left for other layers to compute and verify

No fragment identifier nor offset

- fragments are now handled in a separate extension header
- in fact all “options” are now handled in extension headers

“Don’t Fragment” is implicit

- PathMTU Discovery is required

Differences: Some Not-So-Obvious

Google

Minimum MTU

- IPv4: 576
- IPv6: 1280

Different EtherType

- `0x0800` (IPv4) and `0x0806` (ARP)
- `0x86dd` (IPv6)

No broadcast address or messages

- multicast to different multicast addresses for different purposes
- `ff02::1  ip6-allnodes`
- `ff02::2  ip6-allrouters`

Differences: ICMPv6

Google

Neighbor Discovery and Solicitation

- “ARP” is now specified by Layer 3 messages, using the multicast implementation at Layer 2
- Duplicate Address Detection (DAD)
- NS to solicited node multicast address,
  `ff02::1:ff00:0/104` + lower 24 bits of L2 address

Router Advertisements and Solicitations

- nodes can find routers by themselves (RS)
- routers can communicate link properties to nodes (RA)
  MTU, prefix information, onlink determination, auto-configuration

Multicast Listener Discovery (MLDv2)

- IGMP for IPv6

Differences: Addressing

Google

Abbreviation

- 2001:0DB8:0000:0000:0000:0000:0000:0001 == 2001:db8::1

Scope

- node-local (`::1/128` for loopback, `ff01::/16` for multicast)
- link-local (`fe80::/10` for unicast, `ff02::/16` for multicast)
- global (`2000::/3` for unicast, `ff0e::/16` for multicast)

Stateless Address Auto-Configuration (SLAAC)

- 64-bit network prefix + 64-bit “IPv6 Interface Identifier”
- MAC/OUI48 (`00:1b:63:07:87:8a`)
  to EUI64 (`001b:63ff:fe07:878a`)
  to IPv6 Interface Identifier (`021b:63ff:fe07:878a`)
  prefixed with 64-bit network (`2001:db8:c001:cafe::/64`)
  == SLAAC address (`2001:db8:c001:cafe:21b:63ff:fe07:878a`)

ICMPv6 Router Advertisement example

Google

```
12:49:50.962721 IP6 (class 0xe0, hlim 255, next-header:
   ICMPv6 (58), length: 64) fe80::5:73ff:fea0:66 > ff02::1:
   [icmp6 sum ok] ICMP6, router advertisement, length 64
   hop limit 64, Flags [none], pref medium, router lifetime
   1800s, reachable time 0s, retrans time 0s

     source link-address option (1), length 8 (1):
   00:05:73:a0:00:66
       0x0000:  0005 73a0 0066

     mtu option (5), length 8 (1):  1500
       0x0000:  0000 0000 05dc

     prefix info option (3), length 32 (4):
   2001:db8:1001::/64, Flags [onlink, auto], valid time
   2592000s, pref. time 604800s
       0x0000:  40c0 0027 8d00 0009 3a80 0000 0000 2001
       0x0010:  0db8 1001 0000 0000 0000 0000 0000
```

When You Can't Have Dualstack

Google

Teredo

- IPv4 [UDP [IPv6 […]]]
- `2001:0:server_ipv4:flags:~port:~client_ipv4/128`

6to4

- IPv4 [IPv6 […]]
- `2002:public_ipv4::/48` and `192.88.99.1`

Static Tunnels

- to encapsulate in UDP or not to, that is the question
- relatively deterministic routing

6rd, ISATAP

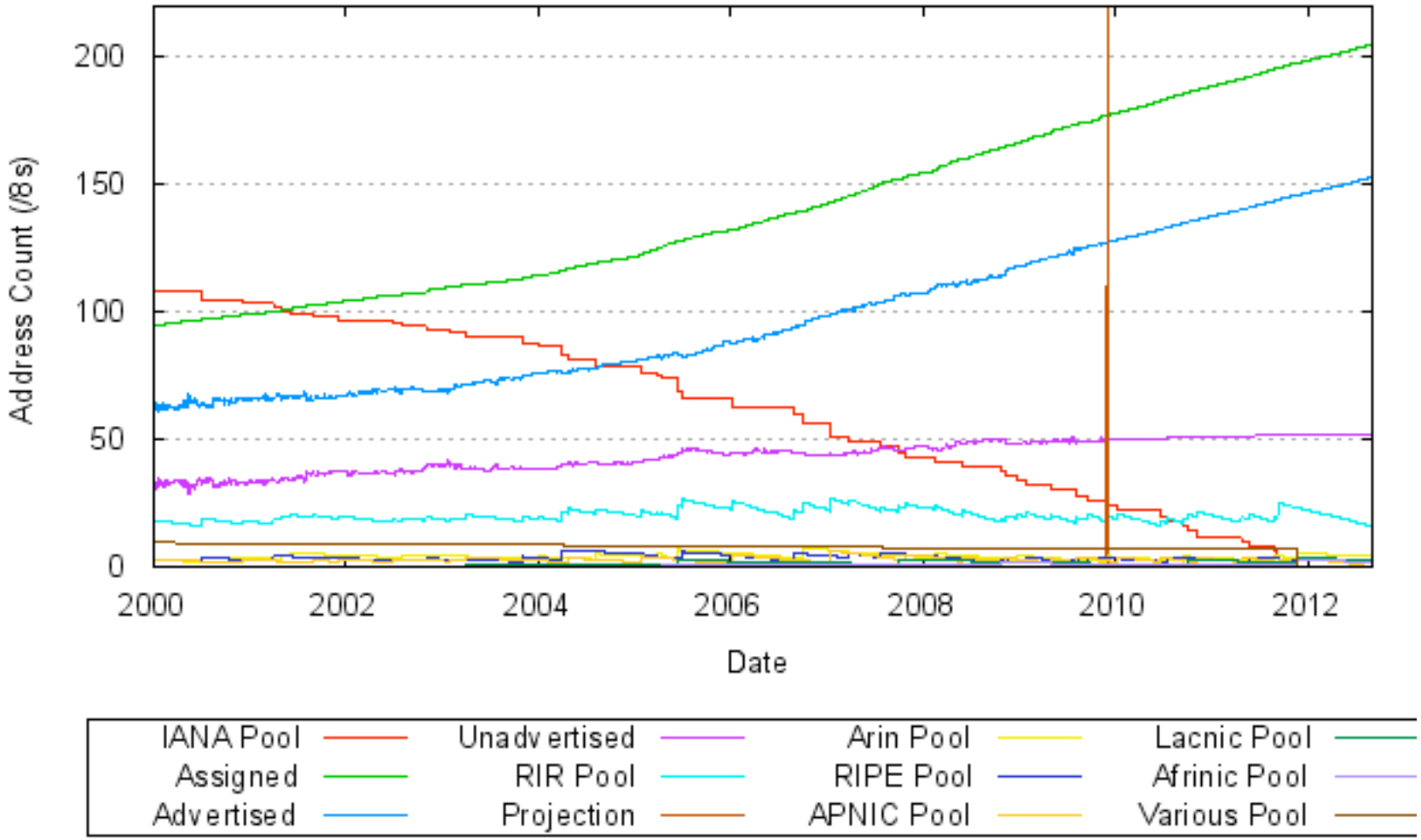
- provider- or company-specific IPv6 prefix
- like 6to4 but can be used with RFC1918 IPv4 addresses

Google

not living in a van down by the river

MOTIVATION

Address Consumption Model (Geoff Huston)



http://www.potaroo.net/tools/ipv4/

Cost of IPv4

Google

Buying IPv4 addresses will be expensive

Carrier-grade (or Large Scale) NAT may be expensive

- Lots of session state memory
- Session logging for legal reasons
- Bandwidth

Being behind a NAT is hard to manage

- Can't fix problems without NAT operator's help
  - VPN, VoIP, video streaming, gaming, P2P
  - think of hassle on certain hotel networks
- Expensive in operator time, support costs

Network complexity creates operation / support costs

Address semantics

Google

With carrier-grade NAT, users share IPv4 addresses

Less accurate geolocation

- legal requirements
- contractual obligations, e.g. content licensing for streaming

Abuse identification / blocking

- if an IPv4 address is spamming/hacking/… who is responsible?
- if we block it, do we take out 100 users?

Carrier-grade NAT requires updating all logs and procedures to record the source port as well

What about non-UDP/TCP traffic?

Opportunities with IPv6

Google

There are a growing number of IPv6-only deployments where IPv4 addressing is truly limited

- free.fr set-top boxes
- Comcast set-top boxes
- NTT's IPTV and VoIP over IPv6

Users still want end-to-end

- Skype
- Bittorent
- “Back to my Mac/PC”

The killer application of IPv6 is the survival of the open Internet as we have known it

So why IPv6 at Google?

Google

When the day comes that users only have IPv6, Google needs to be there for them

Serve current users better over IPv6

- IPv6 can have lower latency and packet loss
  - we have user reports to prove it
- AJAX applications break behind excessive NAT
  - connections exhaust public IPv4 TCP port space
- growing number of IPv6-only client deployments
  - set-top boxes, mobile, …

Not a question of *if*, but when

- so why not now!

IPv6 is good for the Internet, and we want to help

Google

engineering a chicken

IPv6 at Google

A Brief History

Google

14 March 2005	Register with ARIN — 2001:4860::/32
August 2007	Network architecture and software engineering begins (all 20% time)
5 December 2007	Mark Townsley publicly suggests Google serve AAAAs by IETF 73 (which Google was sponsoring)
11 January 2008	First production IPv6 router
29 January 2008	First public demonstration of Google services transparently accessible over IPv6
12 March 2008	Launch ipv6.google.com for IPv4-blackout hour at IETF 71
23 October 2008	First external trusted tester of Google over IPv6
16 November 2008	IETF 73 conference network receives AAAAs for Google services
7 January 2009	Official public availability of Google over IPv6
March 2009	IPv6 maptiles; 3x increase in IPv6 traffic
August 2009	IPv6 enabled in Android source

Corporate IPv6 Deployment

Google

Engineering-driven corporate IPv6 rollout

- engineers need to be able to access and debug the IPv6 versions of the web properties

Dualstacked thus far:

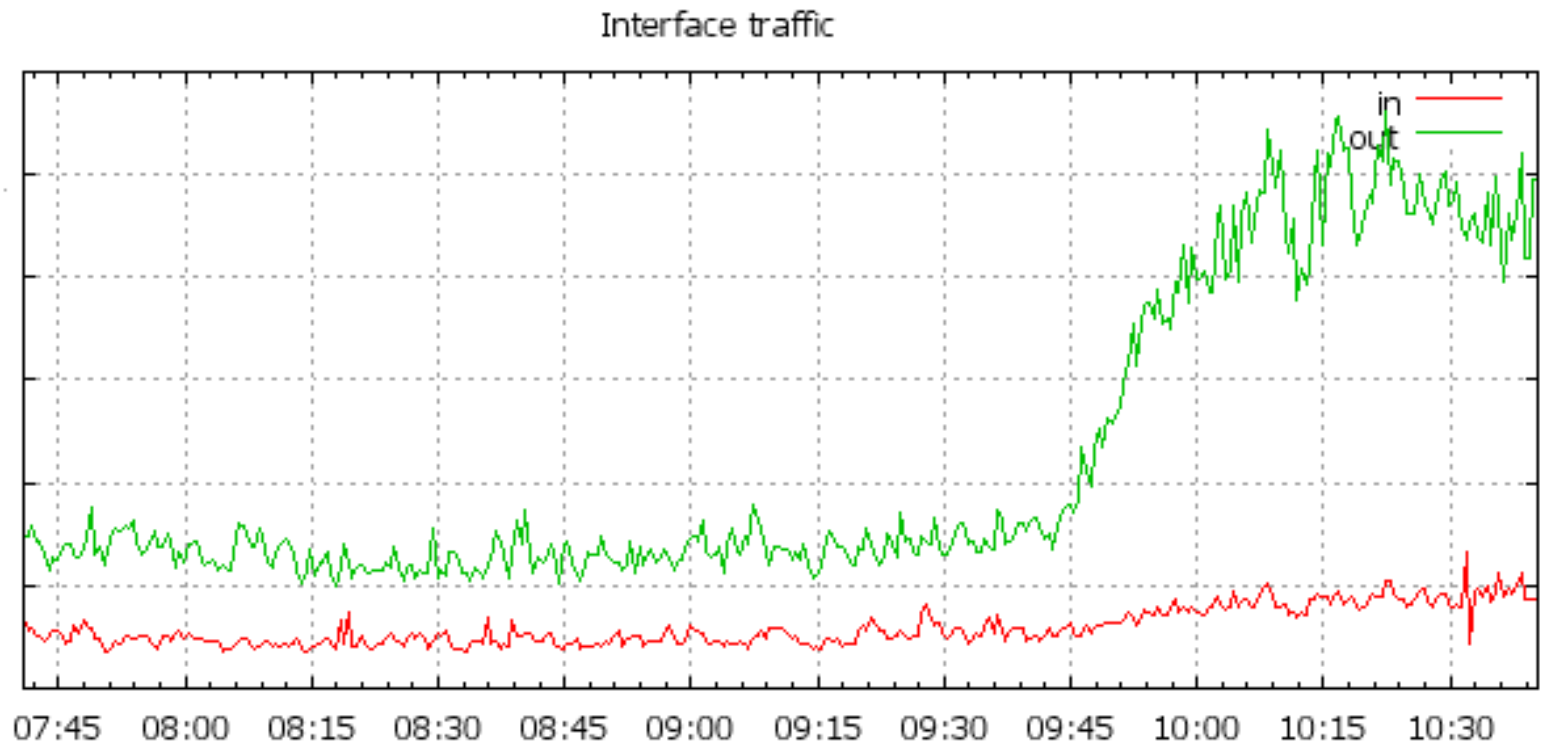
- all corporate POPs
- at least two (if not more) corporate datacenters
- most major Engineering offices worldwide
  - some engineering desktop VLANs
  - wifi networks

STUBL tunnel server for desktops in offices not yet dualstacked

- http://code.google.com/p/stubl

IPv6 Maptiles Traffic

Google



Data Gathering

Google

There was too little data about IPv6 among clients

- existing measurements mostly on a small scale and/or only indirectly related to client IPv6 availability (e.g. IPv6 traffic percentage, IPv6-enabled ASNs)

General worry that turning on IPv6 would reveal all sorts of brokenness

- poorly configured tunnels, suboptimal routing
- home routers dropping/mangling AAAA queries, firewalls blocking IPv6

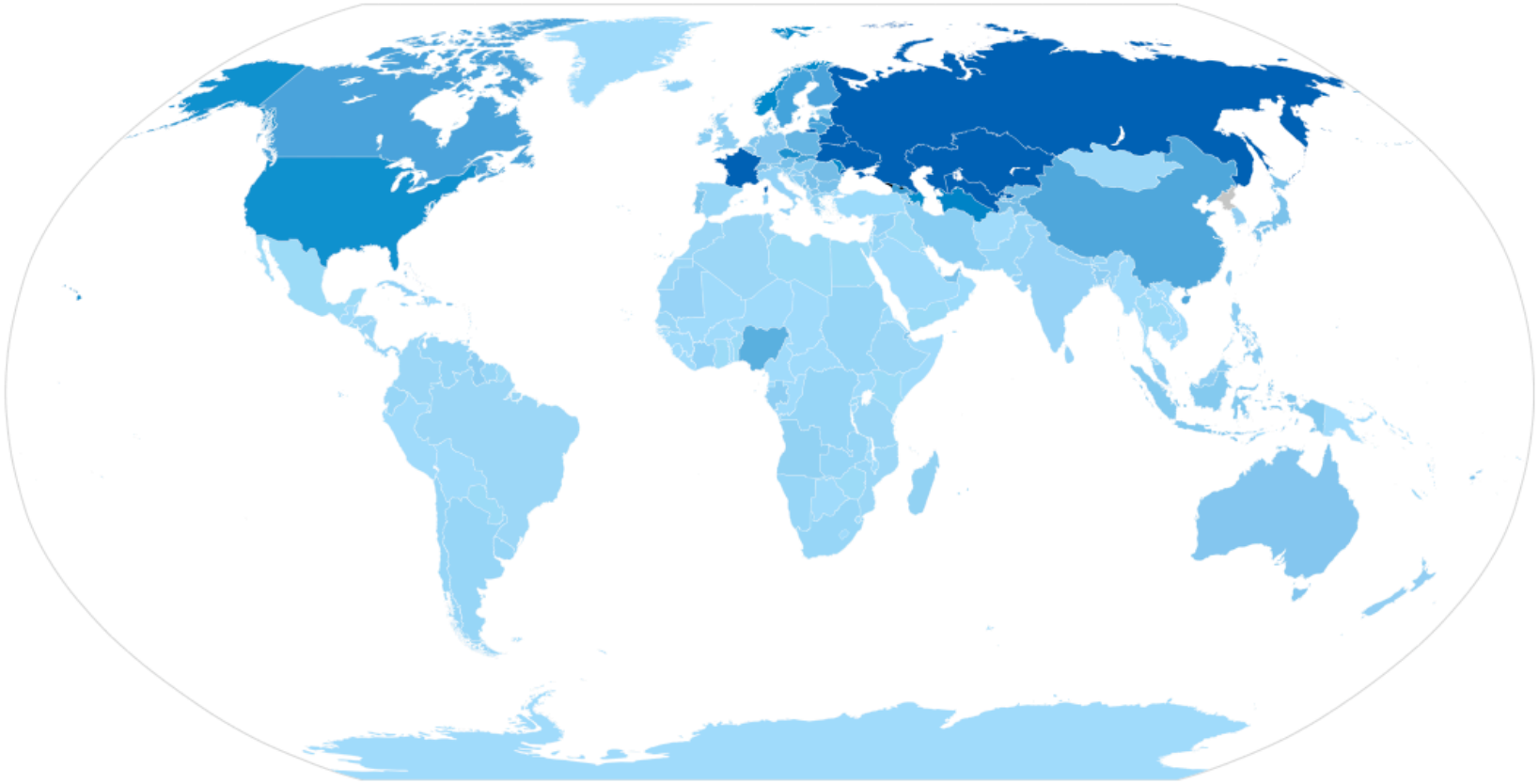
We need to figure out **how common** IPv6 is among our users, how prevalent **brokenness** is, and how we can best serve our IPv6 users

- What is the impact of adding an AAAA record to a website?

Javascript-based experiment in small fraction of search results

Observed Worldwide IPv6 Deployment

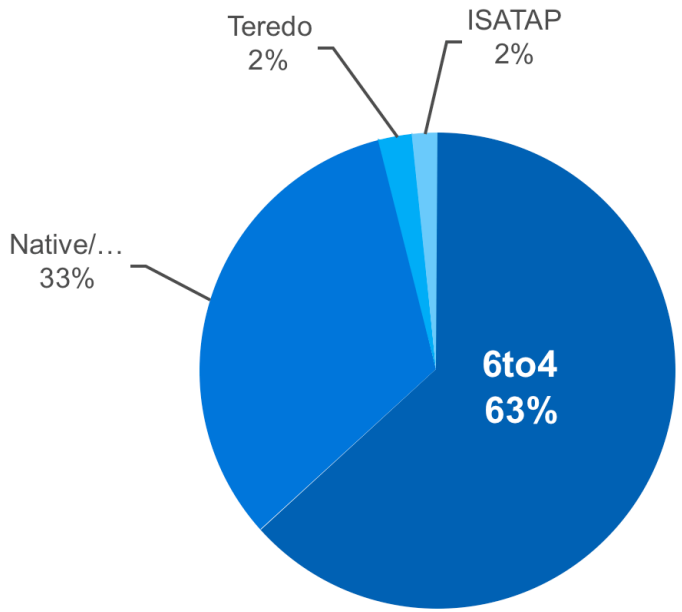
Google



Global data as of June 2009

Method of IPv6 Connectivity

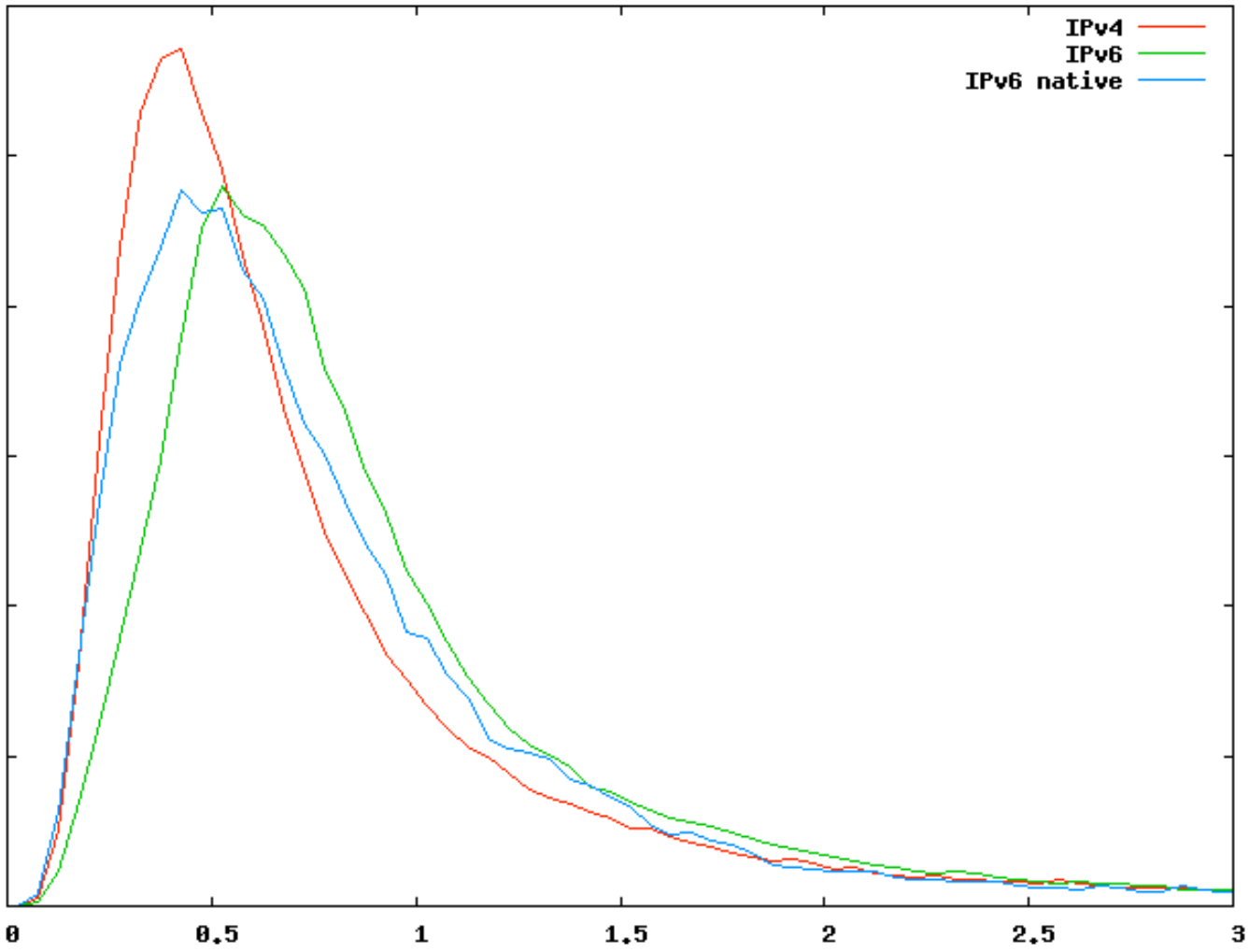
- Based on the IPv6 address we can infer how the user gets IPv6 access
  - Unfortunately, no good way of distinguishing native from tunnels (based on the address alone)
  - Vista with Teredo prefers IPv4 by default, so probably undercounted



- Some countries stand out
  - United States, Canada: 93% 6to4
  - France: 96.6% native (almost all free.fr)
  - China: 70% native, 24.75% ISATAP

Latency Estimates

Google



6to4 and Teredo latency penalty of ~ 50 msec

Google over IPv6

Google

Enables IPv6 access to Google for selected networks

IPv6 access to most Google web properties

- www, mail, calendar, docs, maps, …
- youtube is in the works (no estimate when though)
- goal is to offer AAAAs for everything

Requirements:

- Good IPv6 connectivity to Google
- Production-quality IPv6 network
- Commitment to fix problems that break Google for users

Individuals can receive Google over IPv6 via approved tunnel providers

- currently SixXS, Hurricane Electric, Gogo (formerly Hexago)

How It Works

Normally, if a DNS resolver requests an IPv6 address for a Google web site, it will not receive one…

…but a DNS resolver with Google over IPv6 will receive an IPv6 address, and its users will be able to connect to Google web sites using IPv6

Google

a day in the life of an IPv6 request

IPv6 and the 3 Tier Architecture

Principle of Least Effort

Google

IPv6 doesn't have to be perfect to get started, just get started

- start small and expand steadily
- design for least surprise

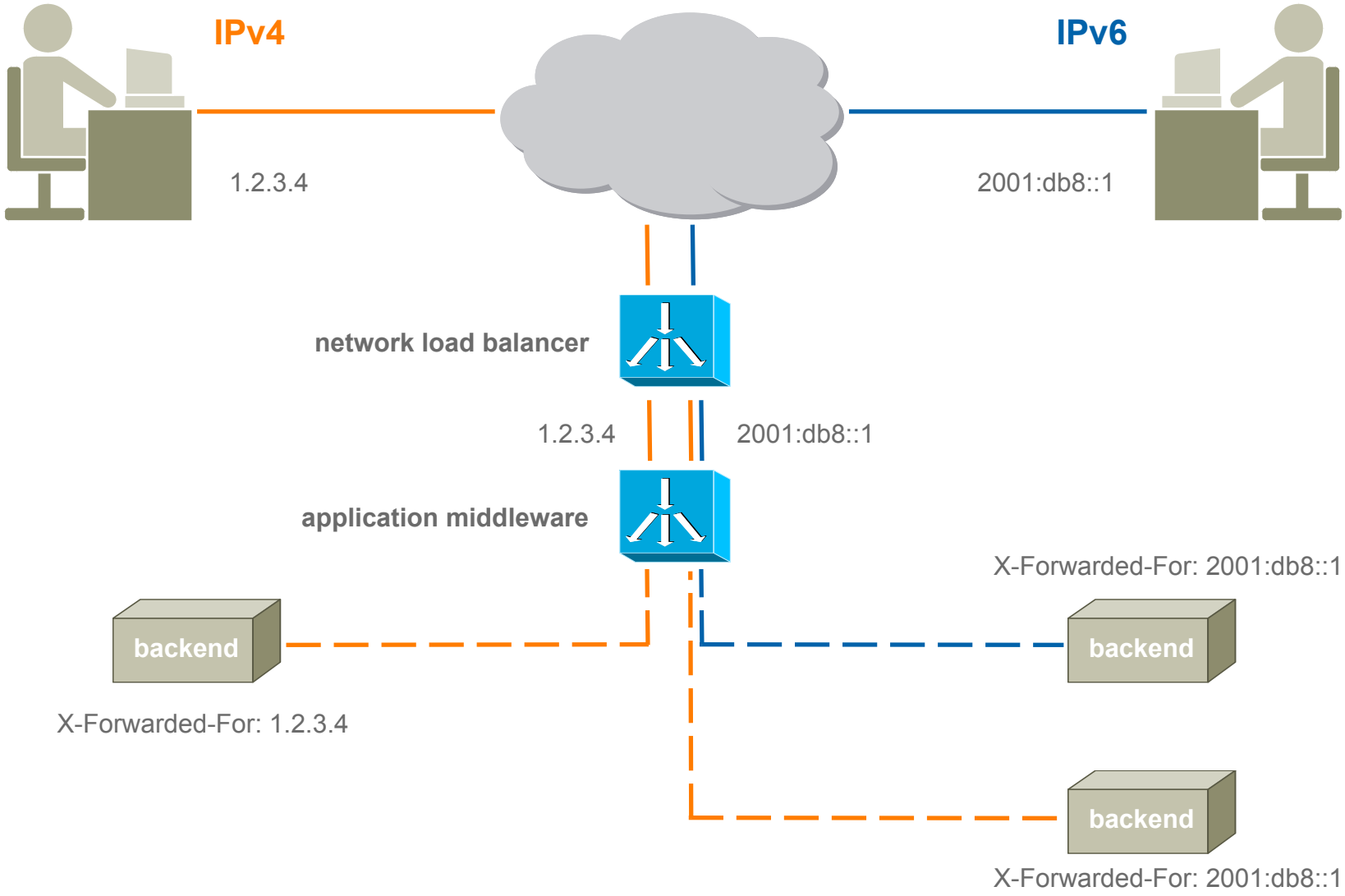
Distinguish among:

- what needs IPv6 native support
- what simply needs to be IPv6 "aware"
- what can comfortably remain IPv4-only

Consider the standard 3-tier architecture application stack:

- IPv6 transport to network load-balanced frontend
- IPv4 or IPv6 to the middle layer, but aware of client address for logging
- IPv4-only to database backends

Simple Strategies



What Worked For Us

Google

Tap enthusiasm

- started as 20% project
- great influx of contributors

Make it easy for contributors to get initial results

- a pilot network is not expensive
- once network is up, internal applications follow

Do it in stages

- IPv6 needn't be as capable as IPv4 on day one
- but it must be done properly
- if it's not production quality, it's no use to anyone

Fold it into your normal upgrade cycles

Google

oh no they didn't!

SOFTWARE IPV4-ISMS

Common IPv4-isms

Google

String literals

- `len(ip_string.split('.')) == 4`
- `re.match('\d+\.\d+\.\d+\.\d+', ip_string)`
- `(ip, port) = sockaddr_string.split(':')`

Assuming IP addresses are integers

- endianness (inconsistent use of `ntohl()`)
- "unspecified" state (`INADDR_NONE`, -1, or 255.255.255.255)

Socket addresses

- Storing `sockaddr_in`
- passing pointers which can be modified

Advanced IPv4-isms

Google

IPv4 netmasks versus strict CIDR syntax

- no more `0.0.255.255` (or worse)

Relationship to other data structures

- IP addresses in a Trie (for access lists, routing)
- map, hash_map

Sockets API

- `IP_TOS` vs `IPV6_TCLASS`
- IPv4 options like Strict Source routing, Record Route
- IPv4 vs IPv6 multicast

see RFC 3542 for Advance IPv6 Sockets API

Google

not including IPX

SOFTWARE IP-AGNOSTICISM

IPv6 Addresses

Google

Audit all string parsing

- beware IPv4 regular expression parsing
- watch for IPv6 URI syntax ("`[2001:db8::1]:443`")

Standardize on a canonical format for readability

- lowercase (MAC addresses are uppercase)
- inet_ntop() zero compression (when "`::`" is used)
- RFC to make formal recommendations

Address comparison

- compare binary structs
- compare "canonicalized" strings

Make code as IP-agnostic as possible

IPv6 Sockets API

Google

Choose a socket strategy

- IPv6 and IPv4 w/ IPv4 mapped addresses ("`::ffff:1.2.3.4`")
  - can be simpler (only need 1 file desriptor)
  - requires careful "normalization"
  - complicated for more advanced usage (DiffServ, Multicast)
- IPv6-only sockets via `IPV6_V6ONLY` sockopt
  - easier for more complex use cases (e.g. TOS/TCLASS)
  - requires 2 (or more) listening sockets

Use only newer API calls

- `inet_ntop, inet_pton`
- `getaddrinfo, getnameinfo`
- `struct sockaddr_storage`

Use proper abstraction via networking libraries

IPv6 and the DNS

Google

Check your resolver library behaviour

- Looking up an AAAA can return an A if no AAAA is found

Dual-stack your applications first

- `/etc/hosts` for testing (or separate DNS servers)
- exposing AAAAs is the control switch that enables IPv6 use (vis. the Google over IPv6 program[1])

Decide when to use IP string literals and when to rely on hostnames

Choose a hostname strategy

- same hostname for A and AAAA
- require different hostnames

[1] http://google.com/ipv6

Google

everyone still awake?

SUMMARY

Observations

Google

IPv6 is not rocket science

- start now
- refactor your network and code steadily

Choose an IPv6-readiness strategy that makes sense for you

Make your IPv6 network production *quality*

- don't worry about production *capacity*

Principle of least surprise

- design close to your existing IPv4 expectations
- but do appreciate the differences where they make sense

Think IP-agnostic and make judicious use of abstractions

Let us know when you're ready!

- http://google.com/ipv6

Thank You!

Google