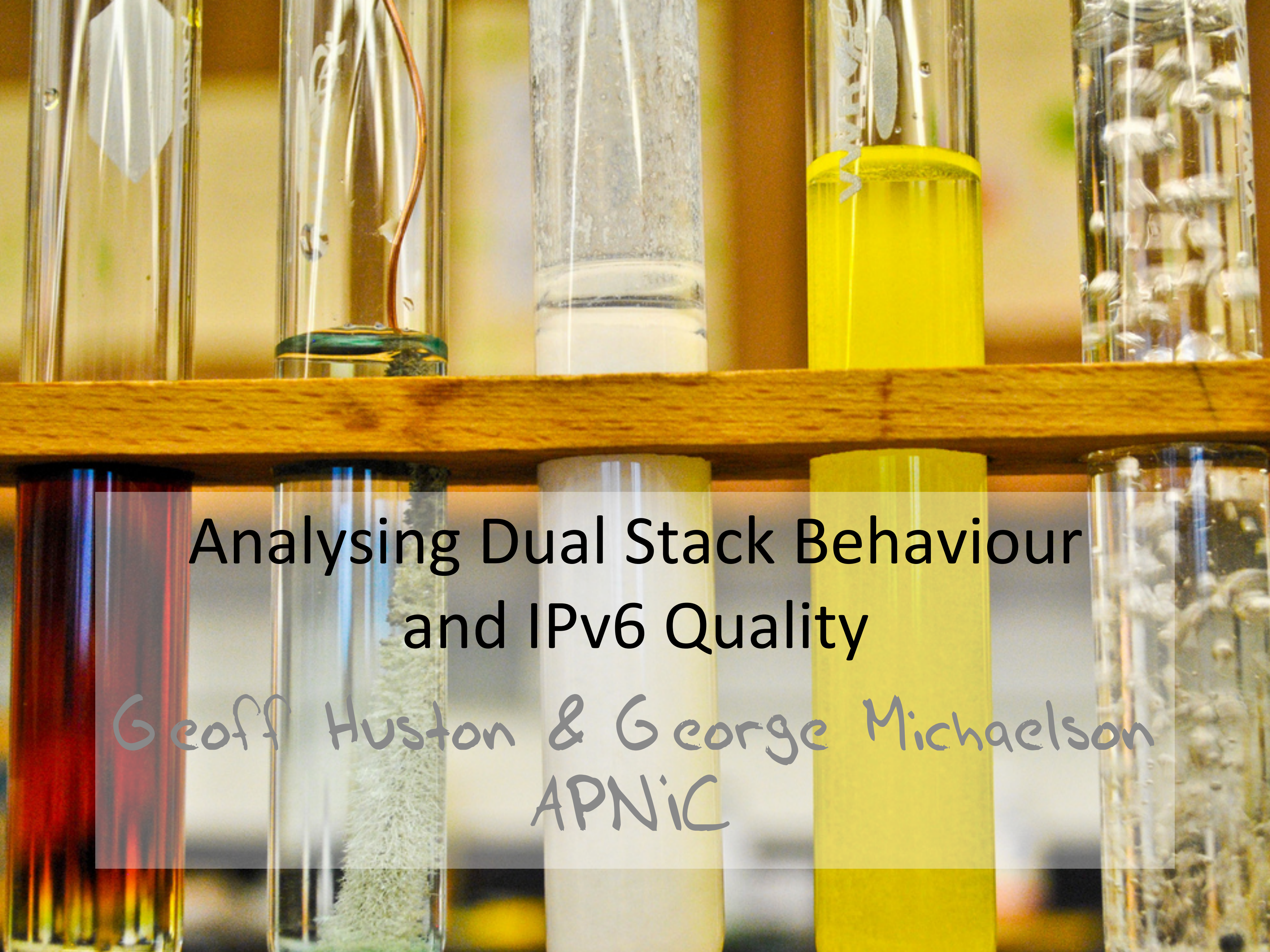


Analysing Dual Stack Behaviour and IPv6 Quality

Geoff Huston & George Michaelson
APNIC

What does a browser do in a dual stack environment?

Is this behaviour better – or worse – than comparable behaviour in a IPv4-ony world?

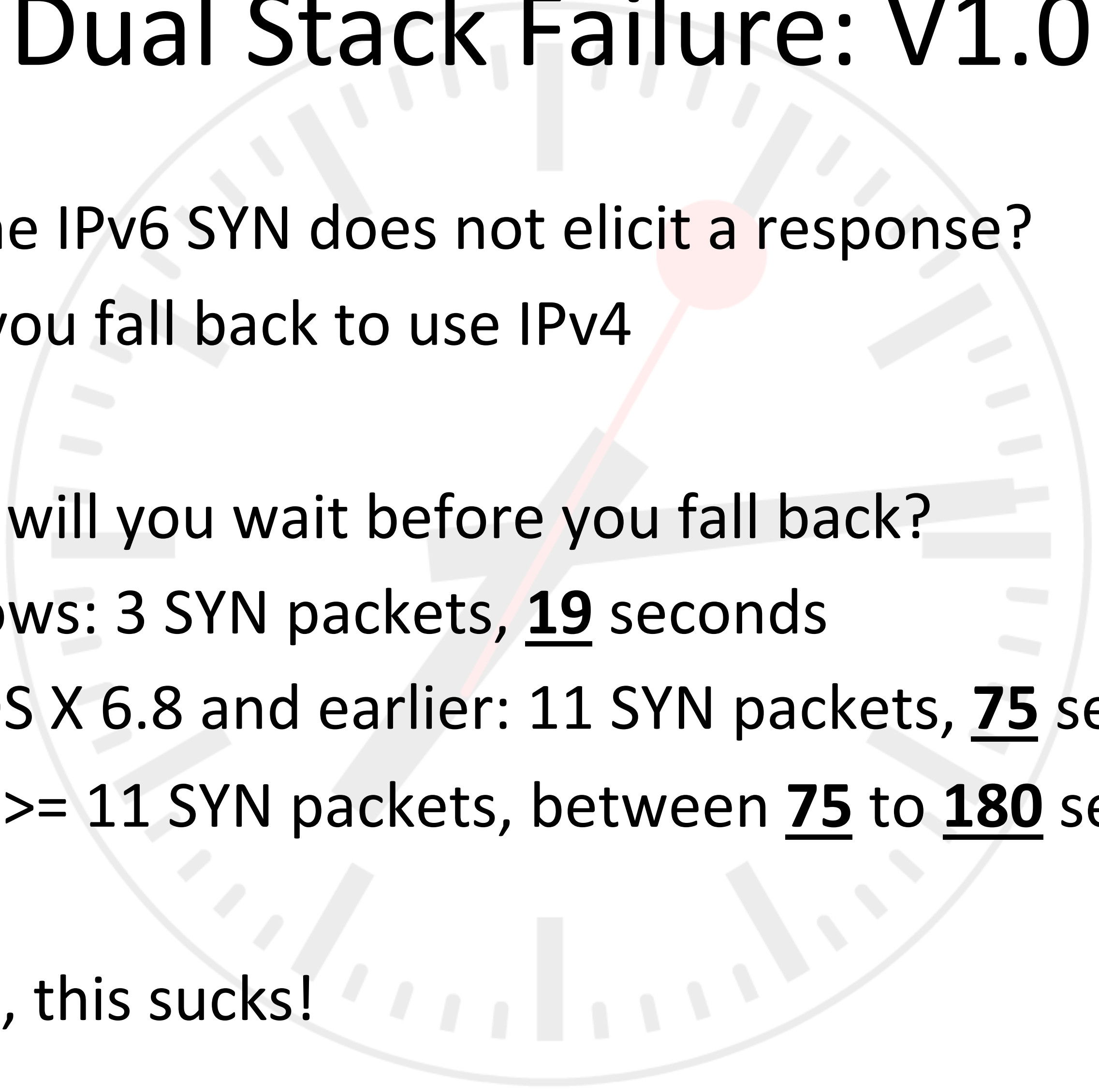
Dual Stack Behaviour: V1.0

Unconditional preference for IPv6 over IPv4

If the local client has an active IPv6 interface then:

- Perform two DNS queries: A and AAAA record queries
- Wait for both to complete
- If the AAAA query succeeds then initiate the browser connection using IPv6
- If there is no AAAA record then initiate the browser connection using IPv4

Dual Stack Failure: V1.0



What if the IPv6 SYN does not elicit a response?

- Then you fall back to use IPv4

How long will you wait before you fall back?

- Windows: 3 SYN packets, **<u>19</u>** seconds
- Mac OS X 6.8 and earlier: 11 SYN packets, **<u>75</u>** seconds
- Linux: >= 11 SYN packets, between **<u>75</u>** to **<u>180</u>** seconds

Obviously, this sucks!

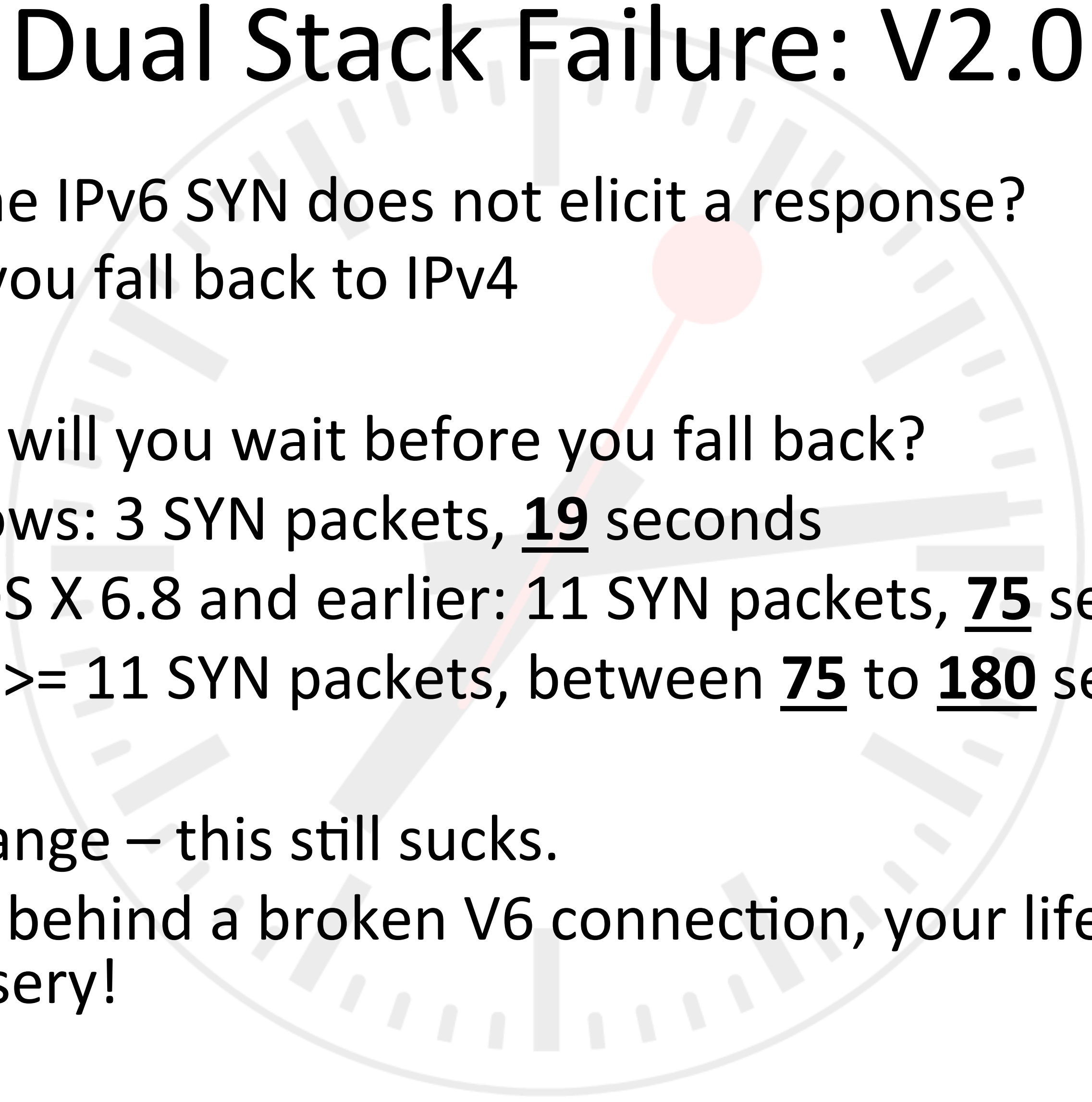
Dual Stack Behaviour: V2.0

Add Local Preference Rules:

1. unicast IPv6
2. unicast IPv4
3. 6to4 tunneled IPv6
4. Teredo IPv6

The effect of this preference table is that if the local IPv6 interface is an auto-tunneled interface than it will only be used when there is no local unicast IPv6 interface and the remote site is IPv6-only

Dual Stack Failure: V2.0



What if the IPv6 SYN does not elicit a response?
- Then you fall back to IPv4

How long will you wait before you fall back?
- Windows: 3 SYN packets, **<u>19</u>** seconds
- Mac OS X 6.8 and earlier: 11 SYN packets, **<u>75</u>** seconds
- Linux: >= 11 SYN packets, between **<u>75</u>** to **<u>180</u>** seconds

i.e. no change – this still sucks.

If you are behind a broken V6 connection, your life is still abject misery!

Dual Stack Behaviour: V2.5 Windows Vista and 7

While Vista and 7 has IPv6 on by default, if the system is behind a NAT the IPv6 interface is a auto-configured as a Teredo auto-tunnel interface

The modified behaviour is that these systems will not even query the DNS for a AAAA record if the only local IPv6 interface is a Teredo interface

- i.e. the Teredo interface is only used when there is no precursor DNS lookup (e.g. use of IPv6 address literal form of URL)

Dual Stack Behaviour: V2.5

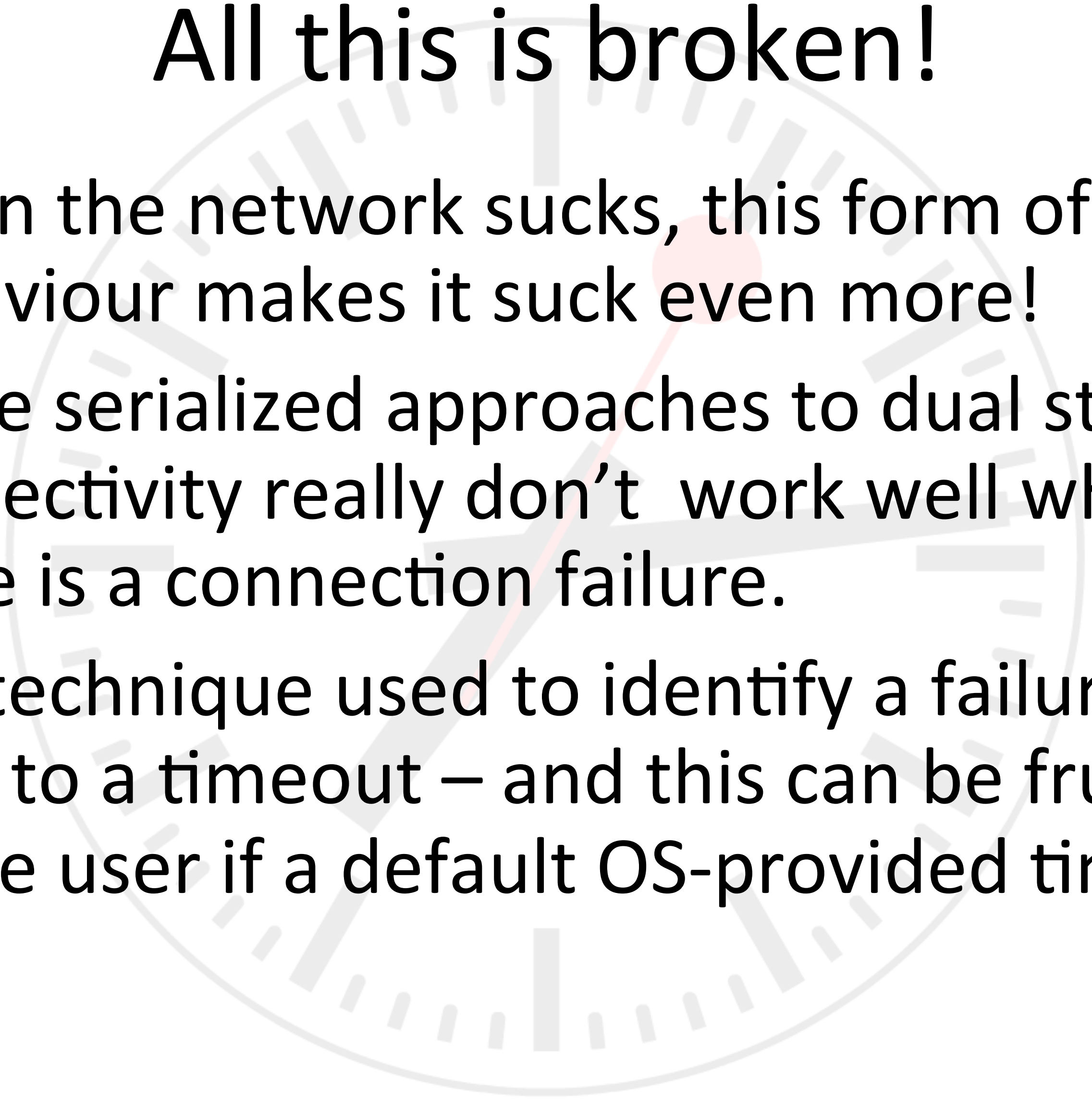
Add Local Preference Rules:

1. unicast IPv6
2. unicast IPv4
3. 6to4 tunneled IPv6

~~4. Teredo IPv6~~

The effect of this is that if the Windows box is behind a NAT and does not have a unicast V6 connection then it shows IPv4-only behaviours

All this is broken!



- When the network sucks, this form of browser behaviour makes it suck even more!
- These serialized approaches to dual stack connectivity really don’t work well when there is a connection failure.
- The technique used to identify a failure falls back to a timeout – and this can be frustrating to the user if a default OS-provided timeout is used

We need better failures!

We need better failures!

- Altering the local preference rules may alter the chances of encountering a failure, but does not alter the poor method of determining when you have failed

The fine print: The real problem here is that the assumption behind the TCP connection code in most operating systems was that there was no fallback – you either connected to a given address or you report failure. To provide a behaviour that was robust under adverse network conditions the OS connection code is incredibly persistent (up to 3 minutes In the case of Linux default). But to use this same code in the circumstance where you have alternate connection possibilities is just testing the user's patience. So we need to rethink this and use a connection strategy that tests all possibilities in a far shorter elapsed time.

How to conduct a two horse race...

Start with one horse

How to conduct a two horse race...

Start with one horse

if it dies on way then send off the other horse!

How to conduct a two horse race...

Or...

You can fire off both horses
at once and go with whichever is fastest...

Dual Stack Behaviour: V3.0
Safari and Mac OSX 10.7 and later

Moderately Happy Eyeballs:

- Determine the preference between IPv4 and IPv6 by maintaining a running performance metric of per-protocol average RTT to each cached destination address
- When DNS queries return both A and AAAA records initiate a connection using the protocol with the lowest current average RTT

Dual Stack Failure: V3.0 Safari and Mac OSX 10.7 and later

- If the connection is not established *within the RTT estimate time interval* then fire off a connection attempt in the other protocol
  - i.e. use a very aggressive timeout to trigger protocol fallback

Dual Stack Failure: V3.0
Safari and Mac OSX 10.7 and later

- ~~If the connection is not established *within the RTT estimate time interval* then fire off a connection attempt in the other protocol~~
  - i.e. use a very aggressive timeout to trigger protocol fallback

Dual Stack Failure: V3.0
Safari and Mac OSX 10.7 and later

- ~~If the connection is not established *within the RTT estimate time interval* then fire off a connection attempt in the other protocol~~
- Only when you have tried ALL the addresses in the first protocol family, then flip over to the other protocol

Dual Stack Failure: V3.0 Safari and Mac OSX 10.7 and later

- If the connection is not established
estimate time interval
attempt in
- esses in the
over to the other

Multi-addressing a critical service point in dual stack situations can make it look worse to clients, not better!

Dual Stack Behaviour: V3.1 Chrome Browser

Happy***ish*** Eyeballs:

- Fire off the A and AAAA DNS queries in parallel
- It's a DNS race: Initiate a TCP connection with the first DNS response
- If the TCP connection fails to complete in 300ms then start up a second connection on the other protocol

Dual Stack Behaviour: V3.2 Firefox and Fast Failover

Happ***ier*** Eyeballs:

- Fire off the A and AAAA DNS Queries
- Initiate a TCP connection as soon as the DNS response is received
- It's a SYN race: Use the first connection to complete the SYN-ACK handshake for data retrieval
- Close off the other connection

The bigger picture...

	Firefox	Firefox fast-fail	Chrome	Opera	Safari	Explorer
MAC OS X 10.7.2	8.0.1 75s IPv6	8.0.1 0ms SYN+ACK	6.9.912.41 300ms DNS	11.52 75s IPv6	5.1.1 270ms RTT	
Windows 7	8.0.1 21s IPv6	8.0.1 0ms SYN+ACK	.0.874.121 300ms DNS	11.52 21s IPv6	5.1.1 21s IPv6	9.0.8112 21s IPv6
Windows XP	8.0.1 21s IPv6	8.0.1 0ms SYN+ACK	.0.874.121 300ms DNS	11.52 21ds IPv6	5.1.1 21s IPv6	9.0.8112 21s IPv6
Linux 2.6.40-3.0	8.0.1 96s IPv6	8.0.1 0ms SYN+ACK		11.60 bets 189s IPv6		
iOS 5.0.1					? 720ms RTT	

Protocol Preference Setting

Failover Timer Values

http://www.potaroo.net/ispcol/2011-12/esotropia.html

The bigger picture...

	Firefox	Firefox fast-fail	Chrome	Opera	Safari	Explorer
MAC OS X 10.7.2	8.0.1 75s IPv6	8.0.1 0ms SYN+ACK	6.9.912.41 300ms DNS	11.52 75s IPv6	5.1.1 270ms RTT	
Windows 7	8.0.1 21s IPv6	8.0.1 0ms SYN+ACK	.0.874.121 300ms DNS	11.52 21s IPv6	5.1.1 21s IPv6	9.0.8112 21s IPv6
Windows XP	8.0.1 21s IPv6	8.0.1 0ms SYN+ACK	.0.874.121 300ms DNS	11.52 21ds IPv6	5.1.1 21s IPv6	9.0.8112 21s IPv6
Linux 2.6.40-3.0	8.0.1 96s IPv6	8.0.1 0ms SYN+ACK		11.60 bets 189s IPv6		
iOS 5.0.1					? 720ms RTT	

http://www.potaroo.net/ispcol/2011-12/esotropia.html

Why?

- Why add all this parallel complexity to browser behaviour?
- What was wrong with the initial concept of "prefer IPv6 if you can, use IPv4 otherwise"?
- Is there really any difference in performance between IPv6 connections?
- Lets see...

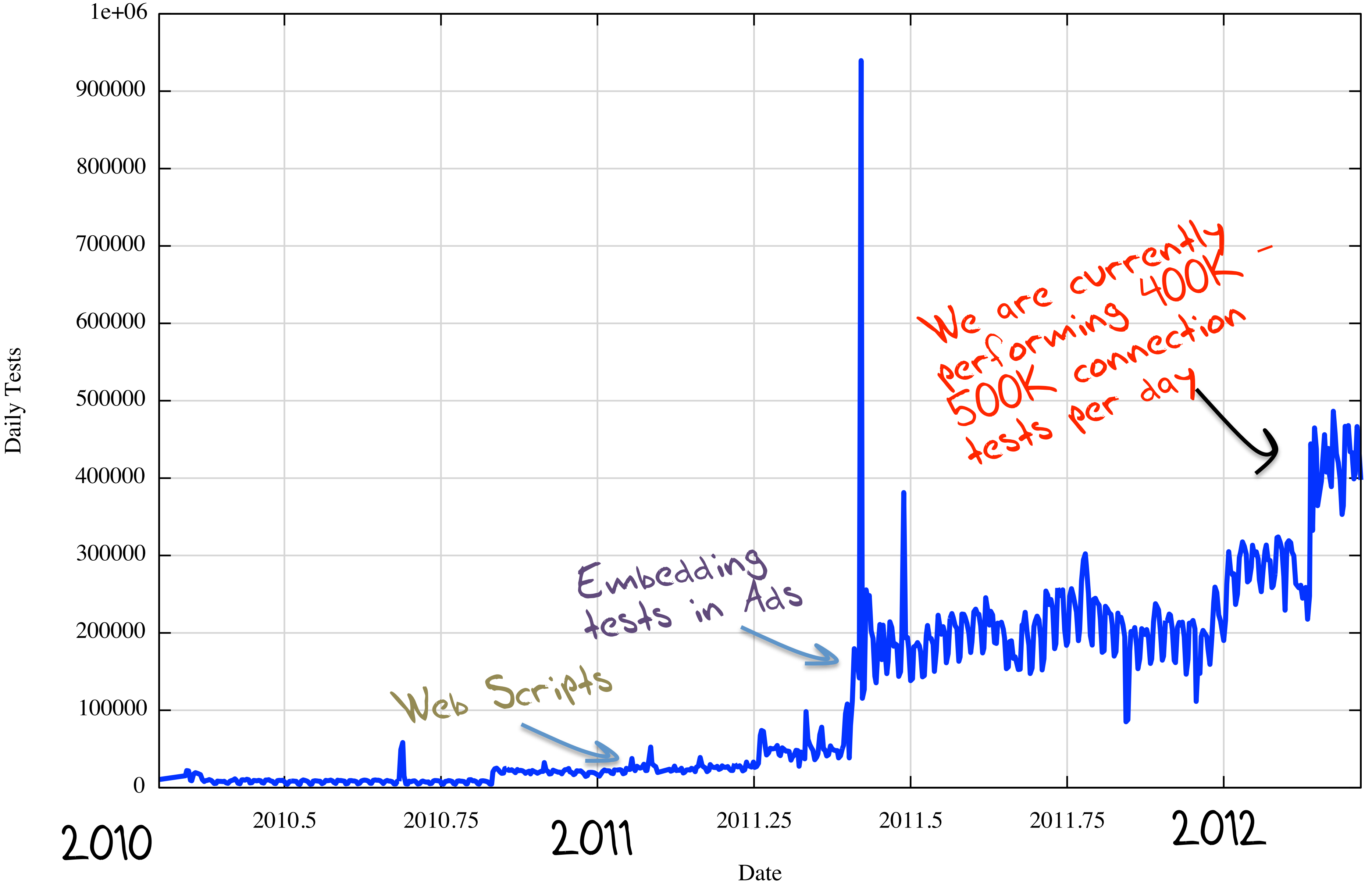
Measuring Dual Stack Quality

Enlist a large set of dual stack clients to connect to an instrumented server using both IPv4 and IPv6

- Equip a number of web sites with a javascript module that poses a number of image-blot retrieval tests
- Extended this using Flash to embed the same tests in a Google Image Ad*

* Thank you to Google, RIPE NCC & ISC for your assistance to conduct this experiment!

Test Volume – Number of unique tests performed per day



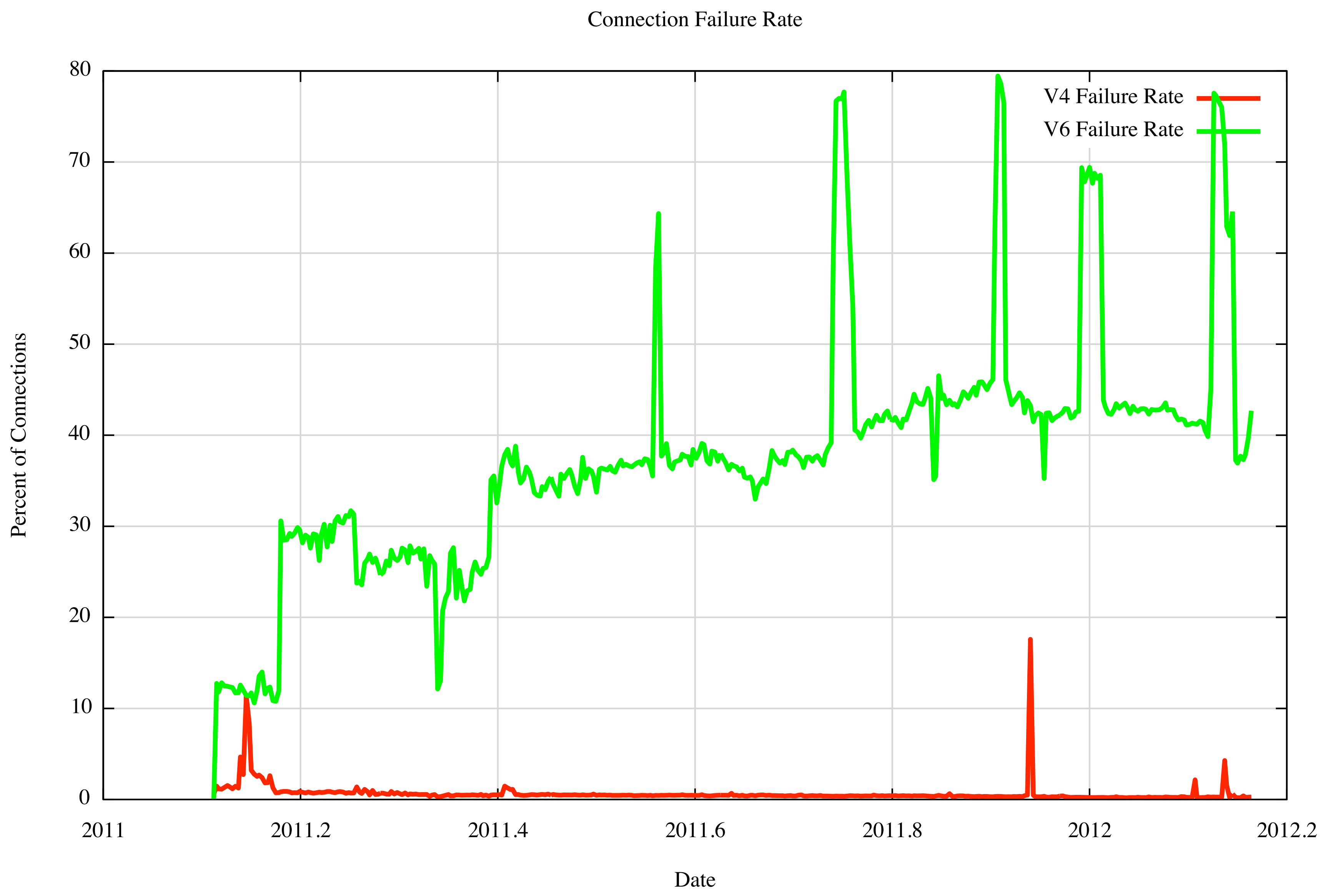
Measuring Dual Stack Quality

Enlist a large set of dual stack clients to connect to an instrumented server using both IPv4 and IPv6

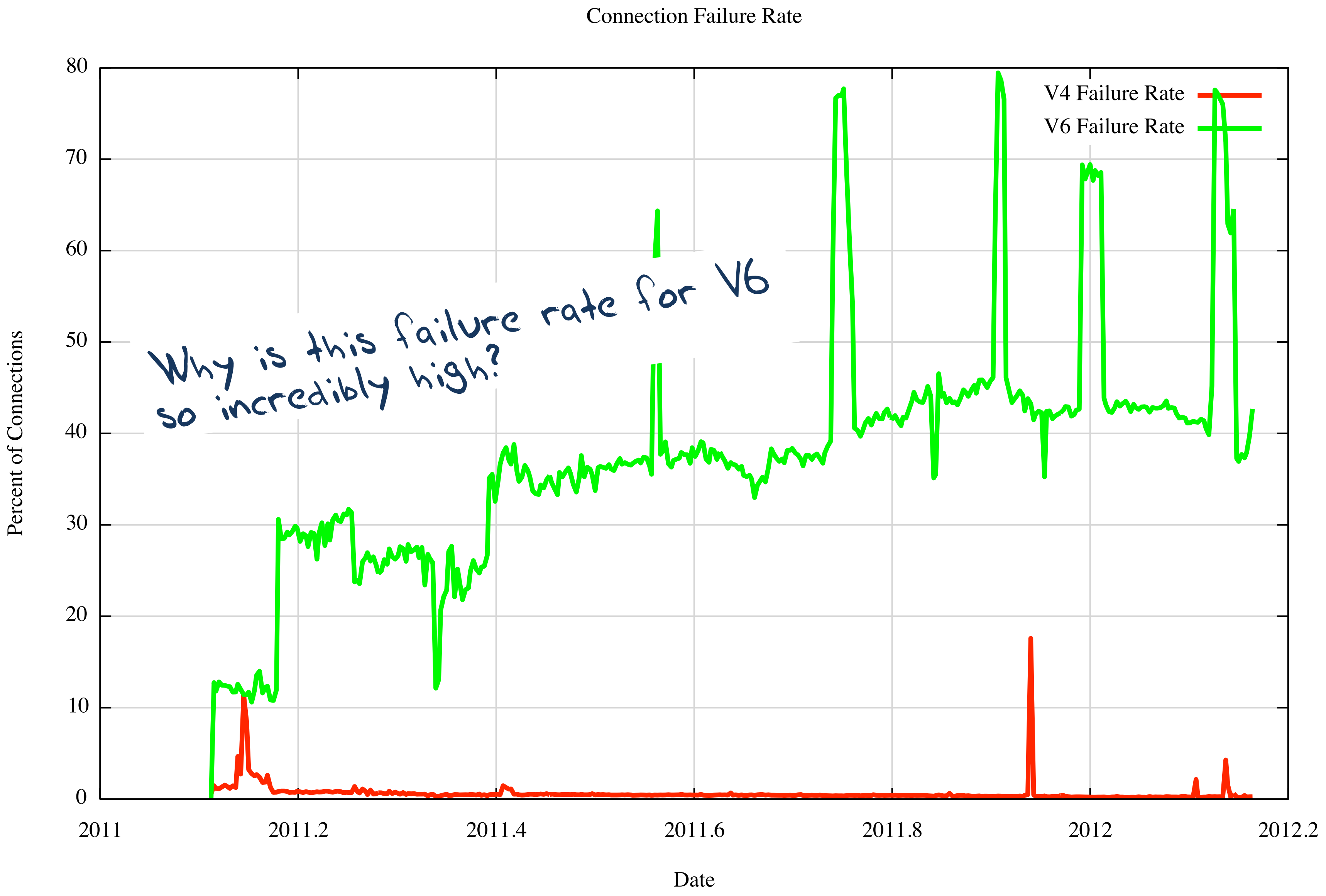
- For each successful connection couplet gather the pair of RTT measurements on the SYN-ACK exchanges
- Gather connection failure statistics (where a "failure" is defined as a received SYN, but no followup ACK)

Connection Failure

Measuring Failure

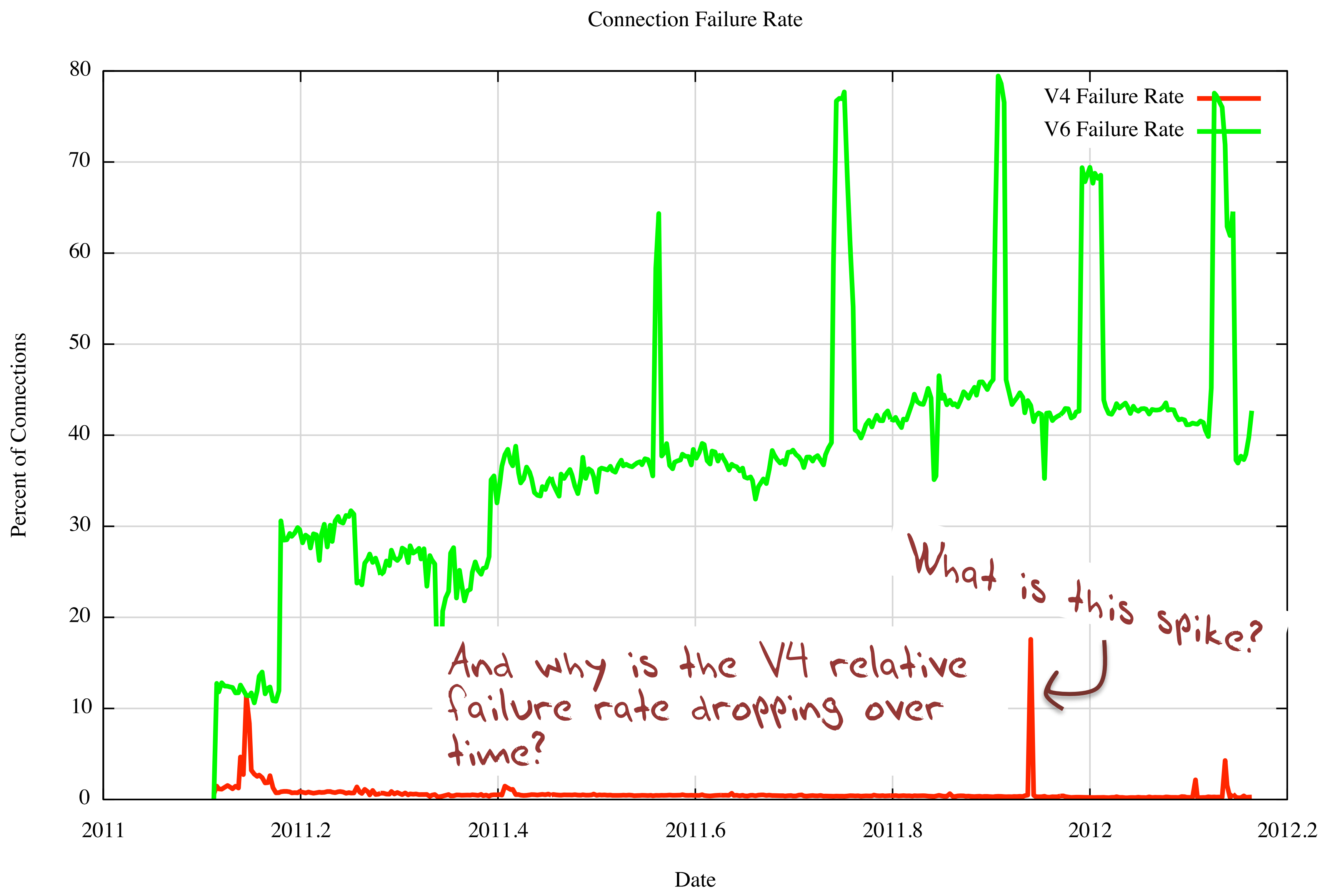


Relative Connection Failure Rates

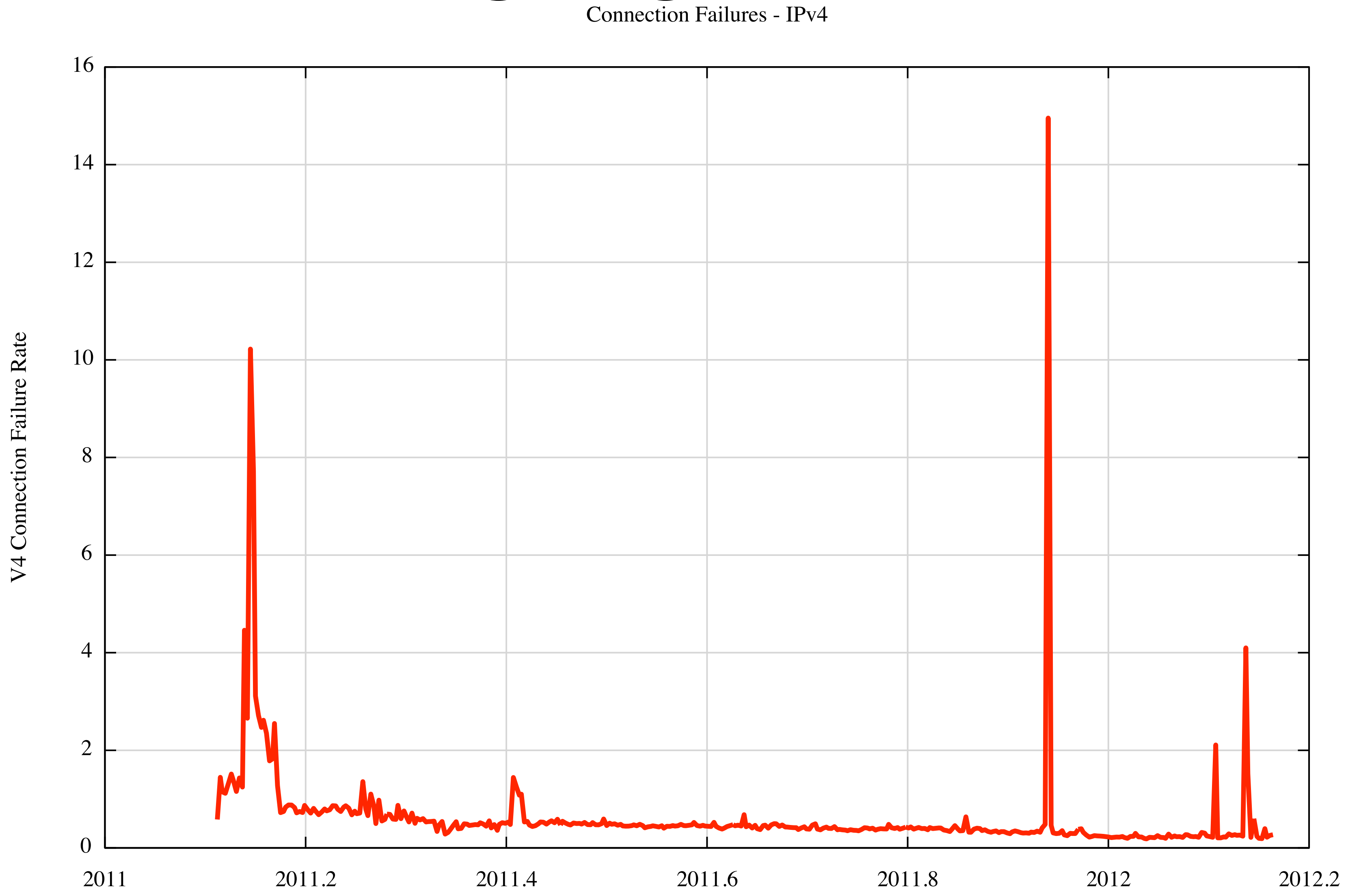


Why is this failure rate for V6 so incredibly high?

Relative Connection Failure Rates



What is going on with IPv4?

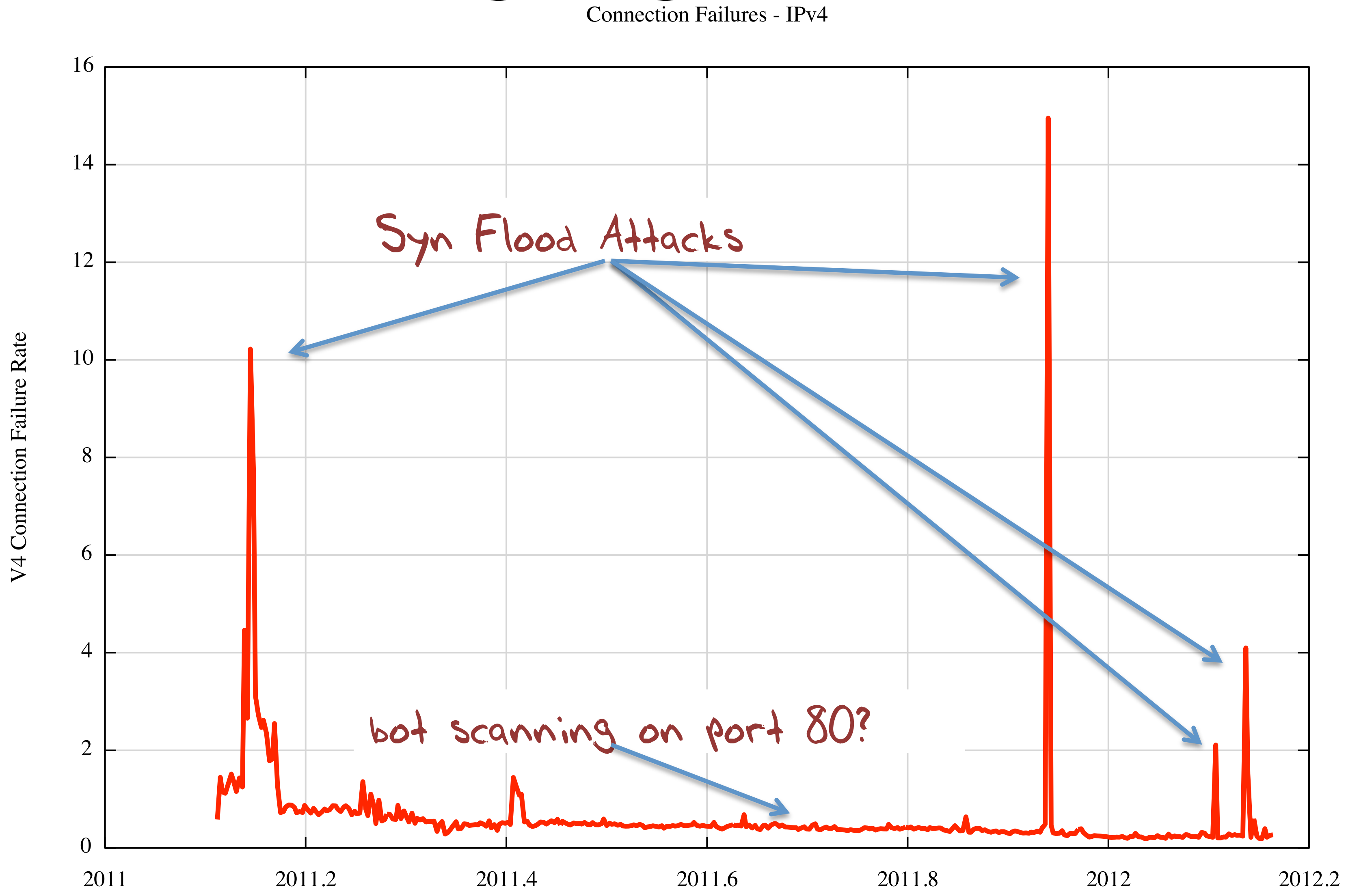


What is going on with IPv4?

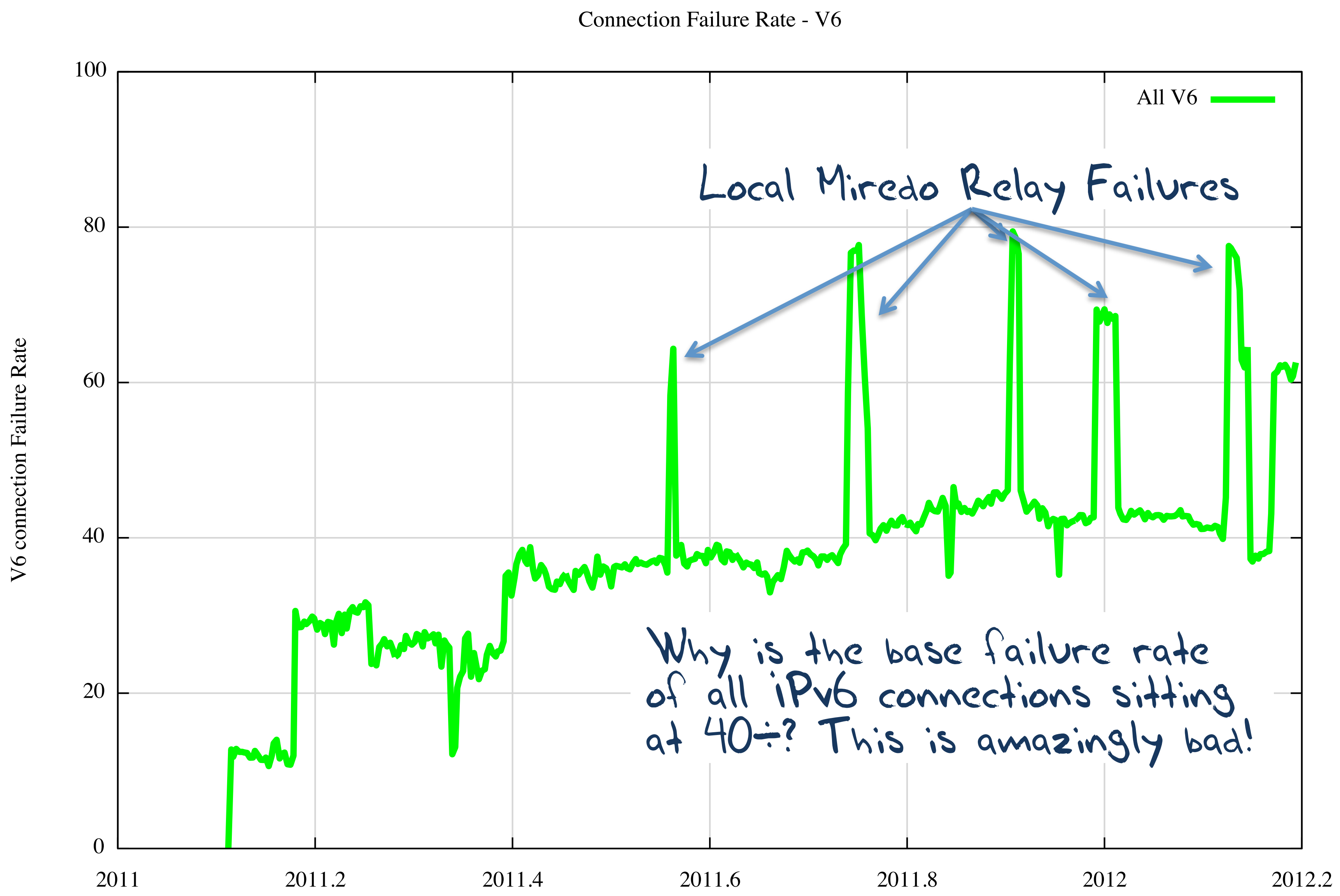
The failure rate for V4 decreases as the volume of experiments increases – which implies that the number of "naked SYNs" being sent to the servers is not related to the number of tests being performed.

Aside from residual IPv4 failures in the image fetch due to device resets, connection dropouts, etc, the bulk of the recorded failures here is probably attributable to bots doing address scanning on port 80

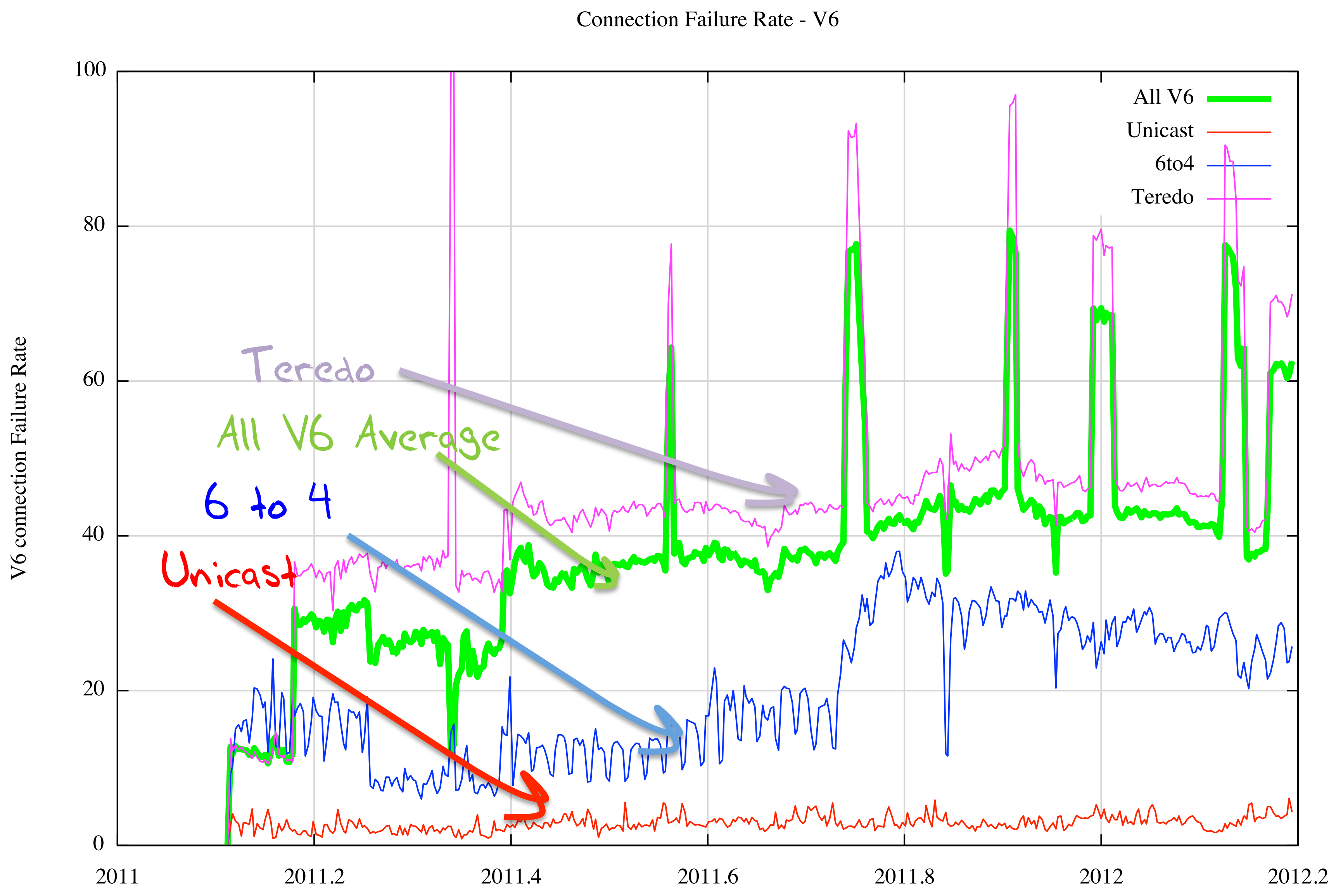
What is going on with IPv4?



What about IPv6?



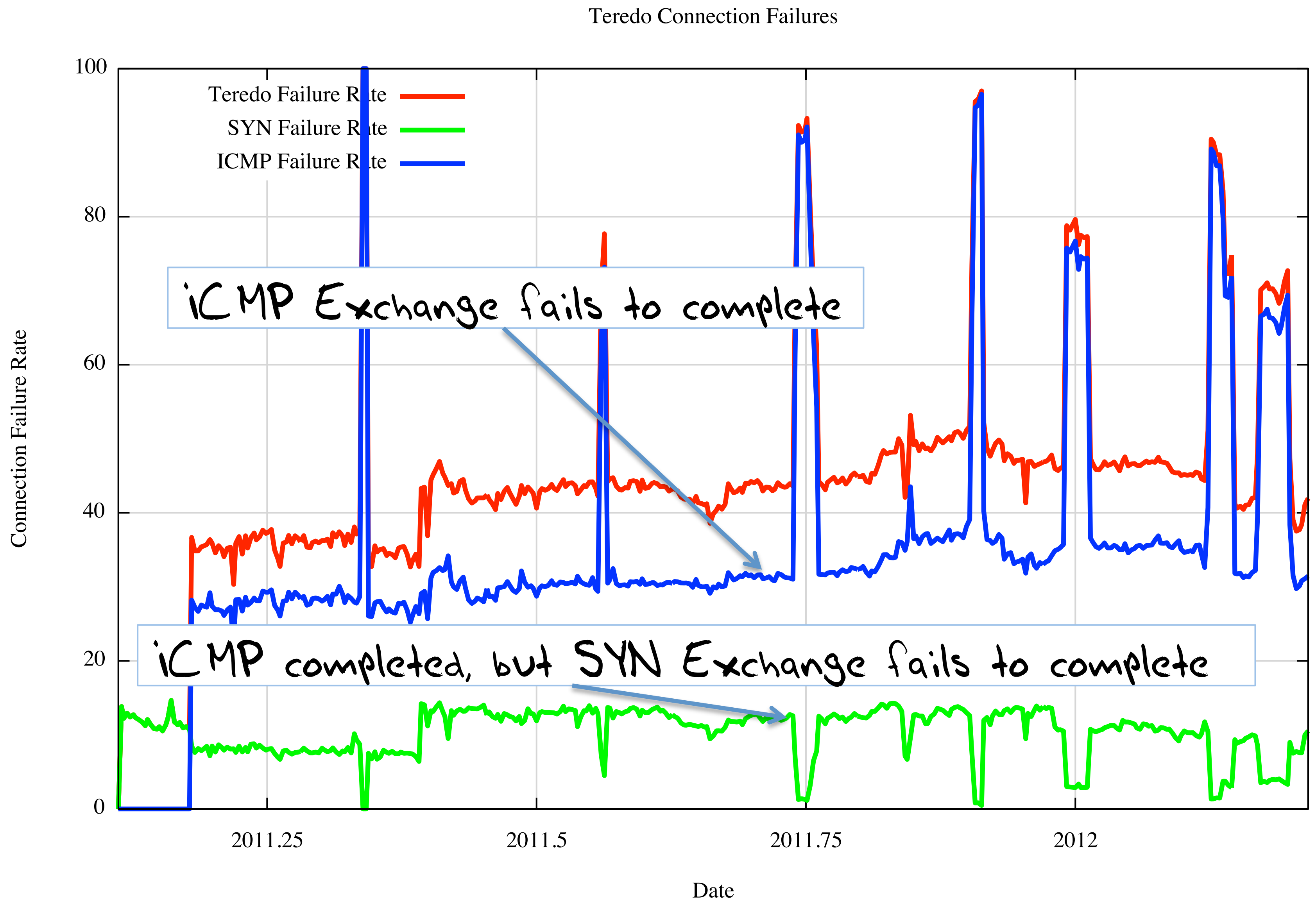
V6 Failure Rate by Address Type



Teredo Failures

- Teredo connections use a 2-step connection process:
  - An ICMP exchange to establish the form of local NAT behaviour (full cone, port restricted cone, ...) and to set up the symmetric path
  - A TCP 3-way handshake
- There are 2 failure modes:
  - ICMP seen, no SYN
  - ICMP seen, SYN seen, no ACK

Teredo Failure Rate



It’s NAT Traversal Failure

- Teredo failure is around 40% of all connection attempts
  - Obviously, this is unacceptably high!
  - This is unlikely to be local filtering effects given that Teredo presents to the local NAT as conventional IPv4 UDP packets
  - More likely is the failure of the Teredo protocol to correctly identify the behaviour mode of the local NAT device

Working with Failure

A 40% connection failure is unworkable is *almost* all circumstances

But one particular application can thrive in this environment, and makes use of Teredo addresses: Torrents

- Not many DPI interceptors are sensitive to V6 in V4 UDP encap
- The massive redundancy of the data set across multiple sources reduces the sensitivity of individual session failures

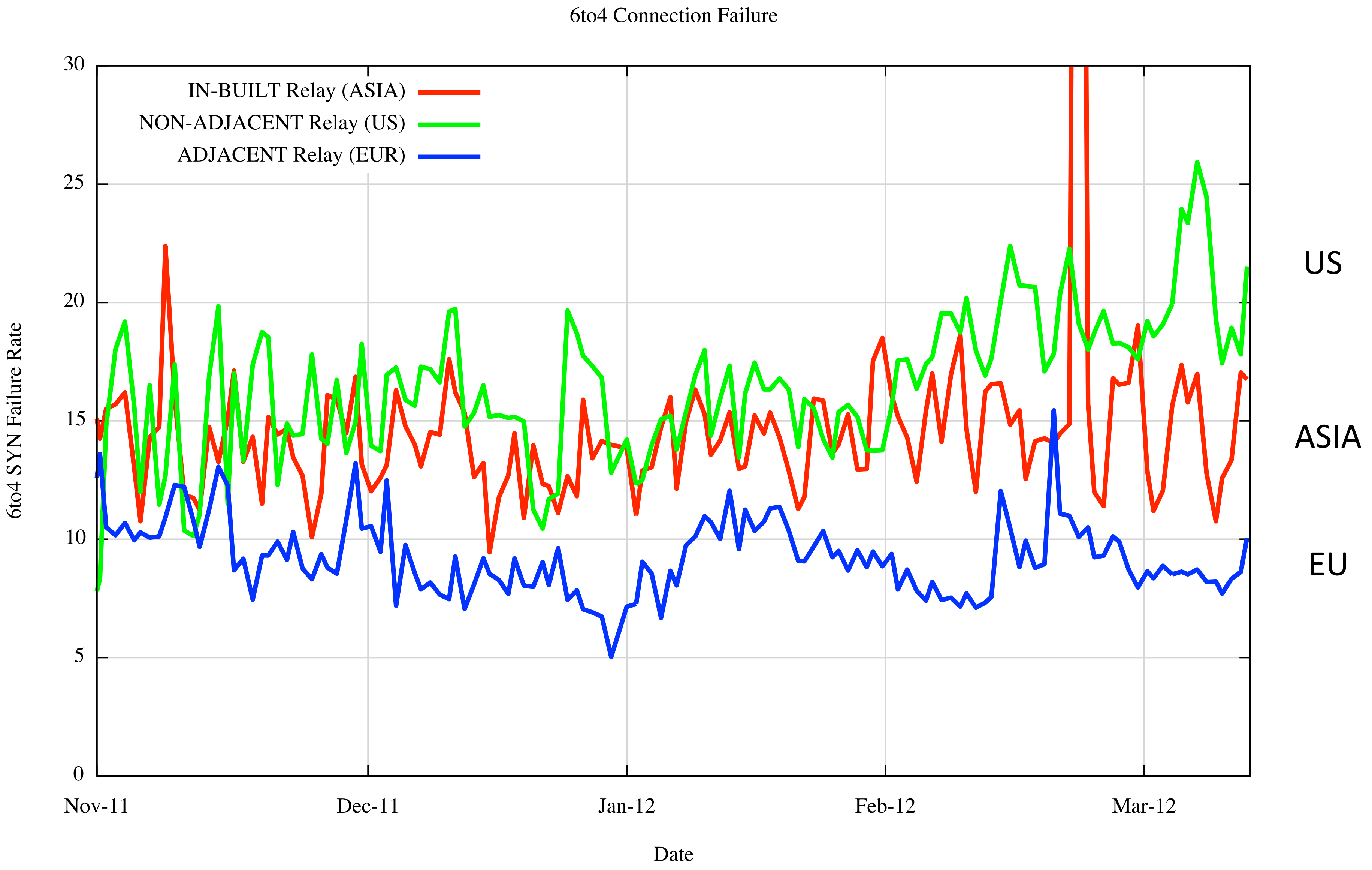
6to4 Auto-tunnelling

6to4 Auto-tunnelling technique

- Cannot operate through IPv4 NATs
- Relies on third party relays in BOTH directions
- Asymmetric traffic paths

- Some of the performance problems can be mitigated by placing the reverse 6to4 relay into the V6 service point

6to4 Failure Rate



6to4 Failure is Local Failure

6to4 failure appears to be related to two factors:

1. The client’s site has a protocol 41 firewall filter rule for incoming traffic (this is possibly more prevalent in AsiaPac than in Europe)
2. Load / delay / reliability issues in the server’s chosen outbound 6to4 relay (noted in the data gathered at the US server)

Even so, the 10% to 20% connection failure rate for 6to4 is unacceptably high!

V6 Unicast Failures

January – March 2012:

110,761 successful V6 connecting endpoints
6,227 failures
That's a failure rate of 5.3%!

7 clients used fe80:: link local addresses
7 clients used fc00:/7 ULA source addresses
2 clients used fec0::/16 deprecated site local addresses
16 clients used 1f02:d9fc::/16
Nobody used 3ffe::/16 prefixes!

What about the other 6,195 clients?

Unicast IPv6 Failures

8 were using unallocated unicast V6 addresses

66 were using unadvertised unicast V6 addresses

6,116 were using V6 addresses drawn from conventional advertised V6 prefixes!

Local inbound filters appear to be a common problem in IPv6

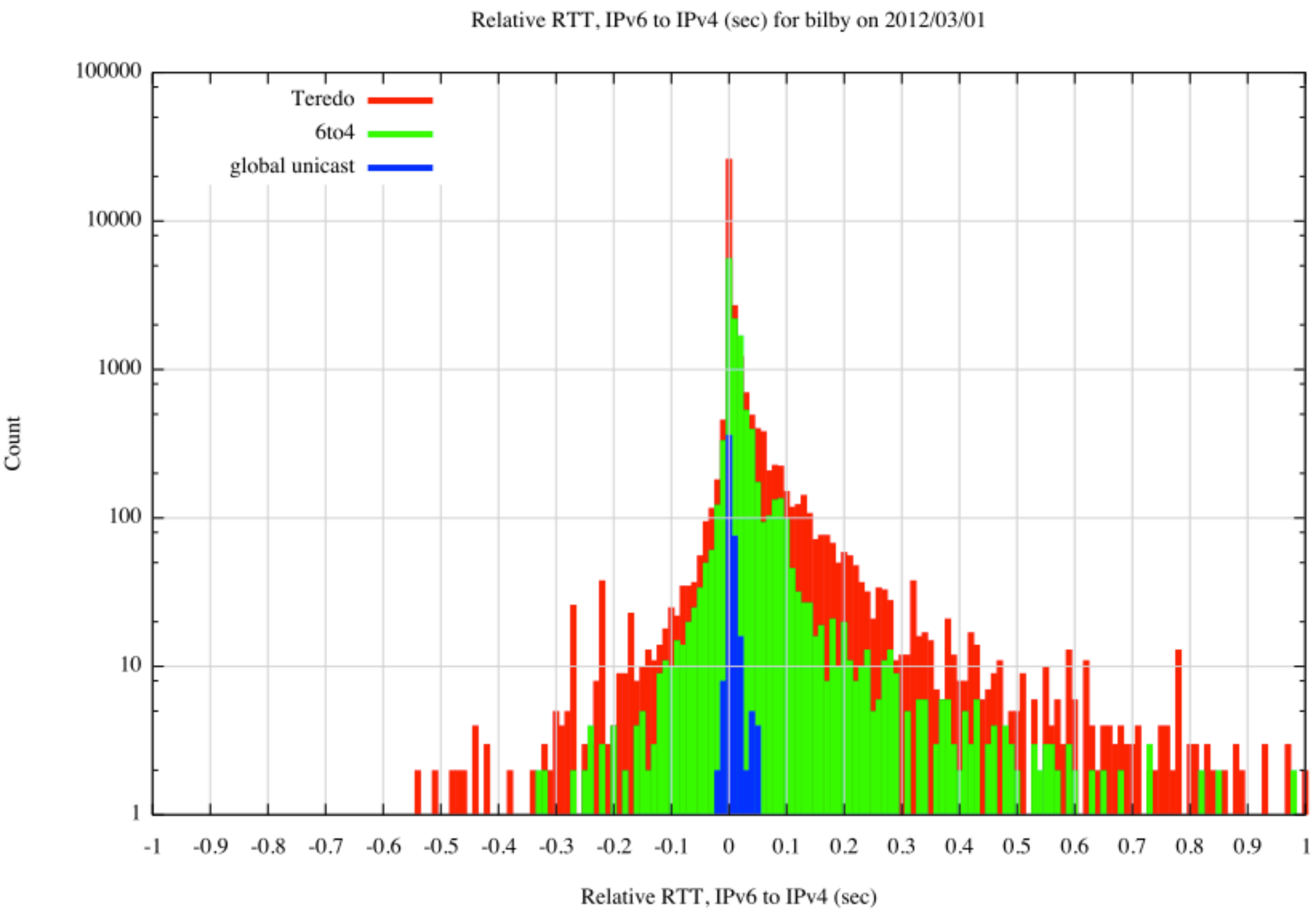
Where does V6 Fail?

Average - 5.3% of unicast V6 connections fail to complete
However, we saw wide variance across countries:

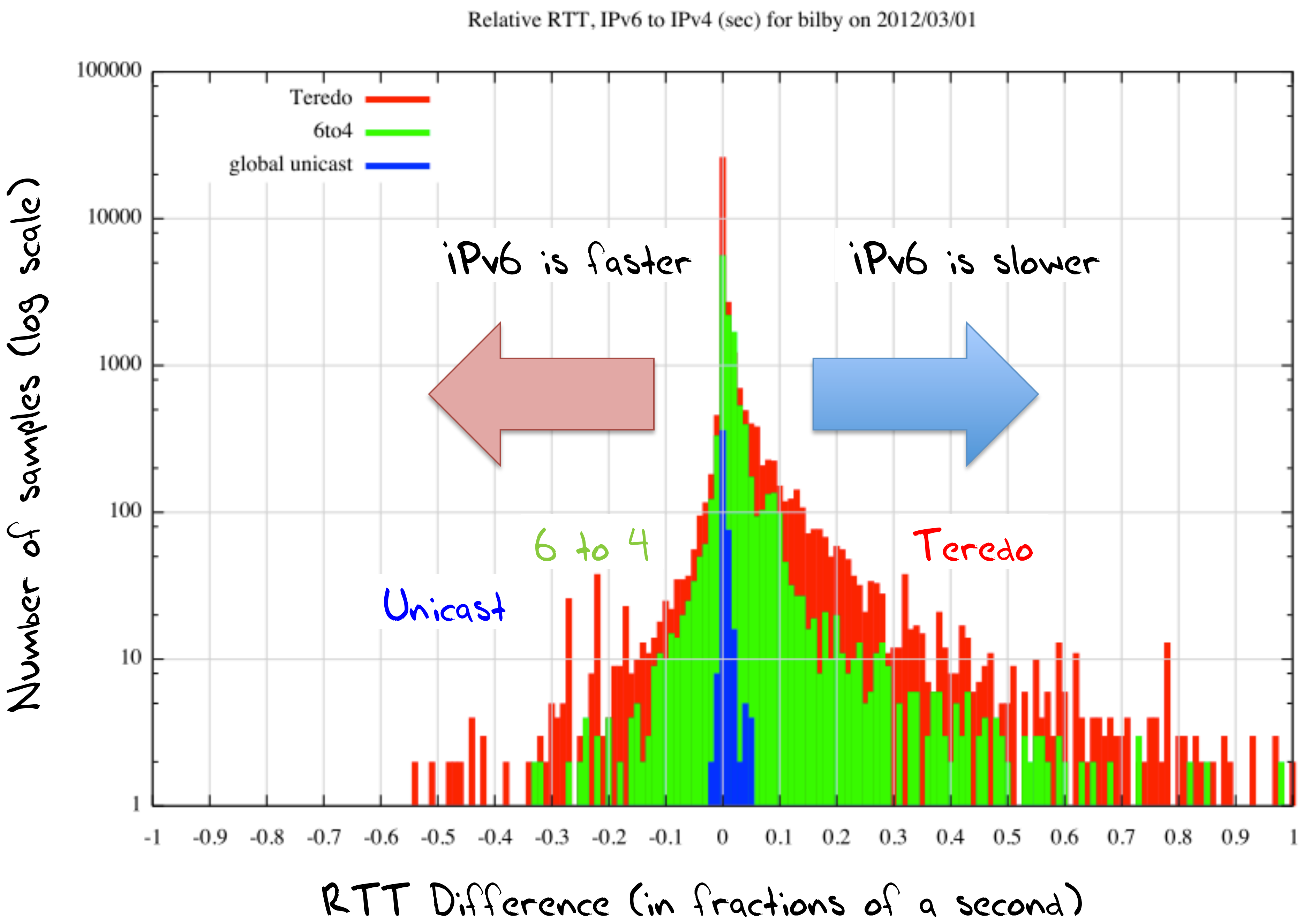
Highest:	Lowest:
Spain – 18%	Norway – 0.2%
Vietnam – 16%	Australia – 0.7%
Indonesia – 13%	France – 0.8%
Hong Kong – 10%	Russia – 1.7%
Canada – 10%	Italy – 2%
Sweden – 10%	China – 3%
Brazil – 6%	Germany – 3%
United States – 6%	Japan – 4%

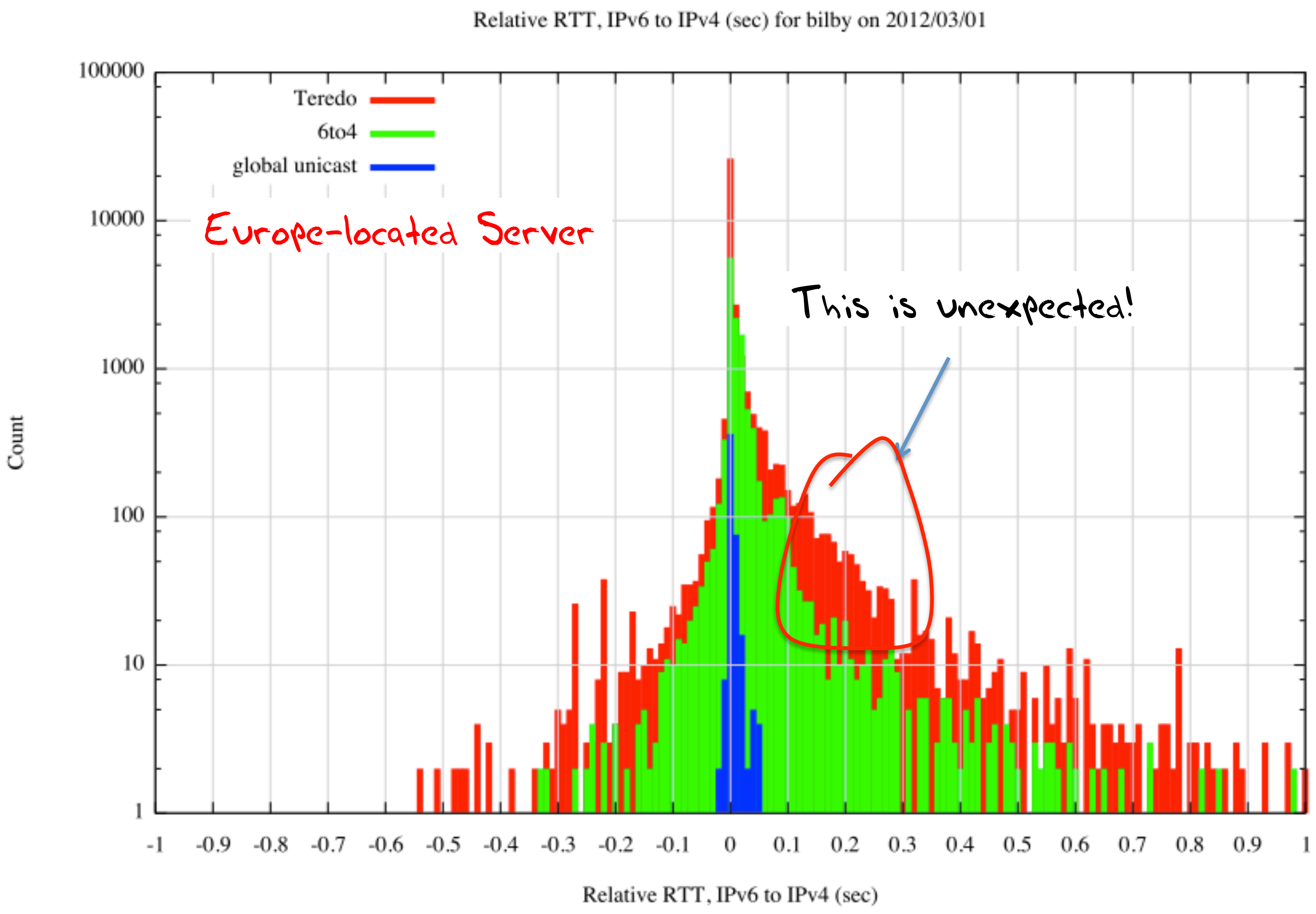
Measuring Dual Stack Quality

- For each successful connection couplet gather the pair of RTT measurements on the SYN-ACK exchanges
  - Use the server's web logs to associate a couplet of IPv4 and IPv6 addresses
  - Use the packet dumps to collect RTT information from the SYN-ACK Exchange



Relative RTT, IPv6 to IPv4 (sec) for bilby on 2012/03/01
Teredo
6to4
global unicast
Count
100000
10000
1000
100
10
1
-1
-0.9
-0.8
-0.7
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Relative RTT, IPv6 to IPv4 (sec)

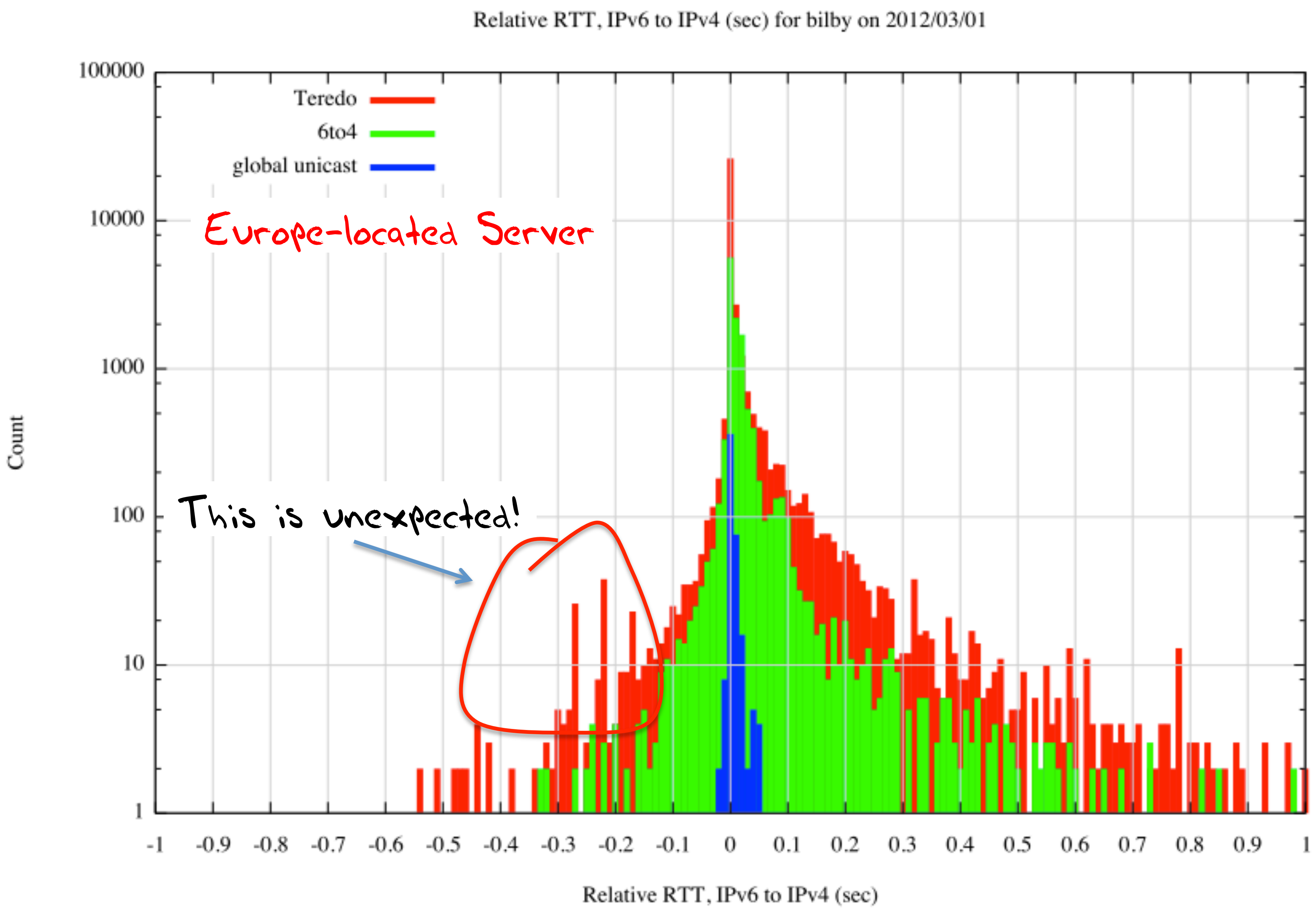

Relative RTT, IPv6 to IPv4 (sec) for bilby on 2012/03/01
Teredo
6to4
global unicast
iPv6 is faster
iPv6 is slower
6 to 4
Teredo
Unicast
Number of samples (log scale)
RTT Difference (in fractions of a second)
100000
10000
1000
100
10
1
-1
-0.9
-0.8
-0.7
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Relative RTT, IPv6 to IPv4 (sec) for bilby on 2012/03/01
Count
Relative RTT, IPv6 to IPv4 (sec)
Teredo
6to4
global unicast
100000
10000
1000
100
10
1
-1
-0.9
-0.8
-0.7
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Europe-located Server
This is unexpected!

Why is Teredo slower?

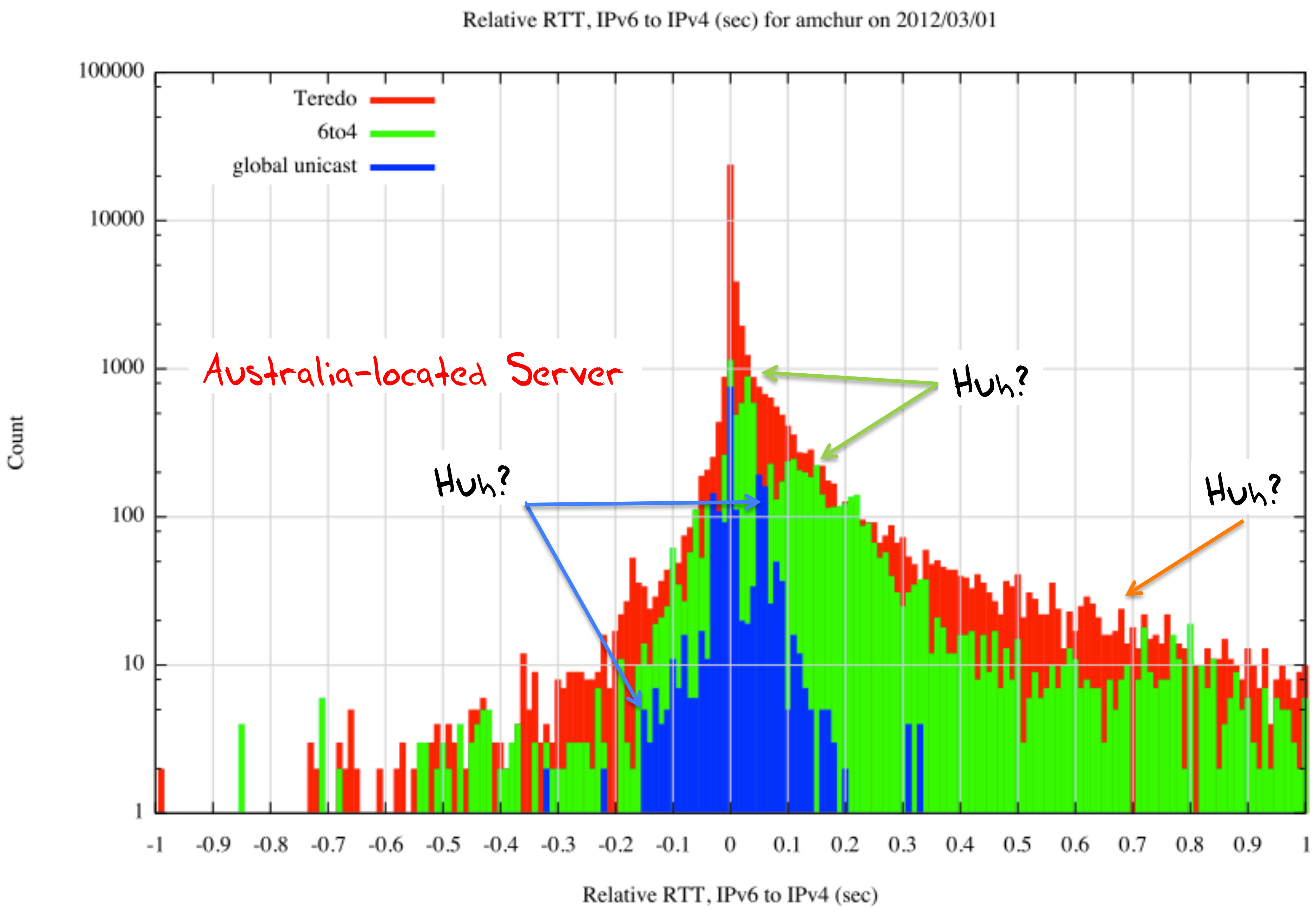
The technique used here is to measure the interval between the first received SYN and the first received ACK

- But something is happening with Teredo
  - we use inbuilt Teredo Relays, so the Teredo RTT should precisely match the IPv4 RTT
    - But we are measuring the initial SYN exchange
    - It appears that there are some major setup delays in Teredo that are occurring in the initial SYN ACK exchange
    - The performance of CPE based NATs has a massive tail of delay, woe and abject misery!

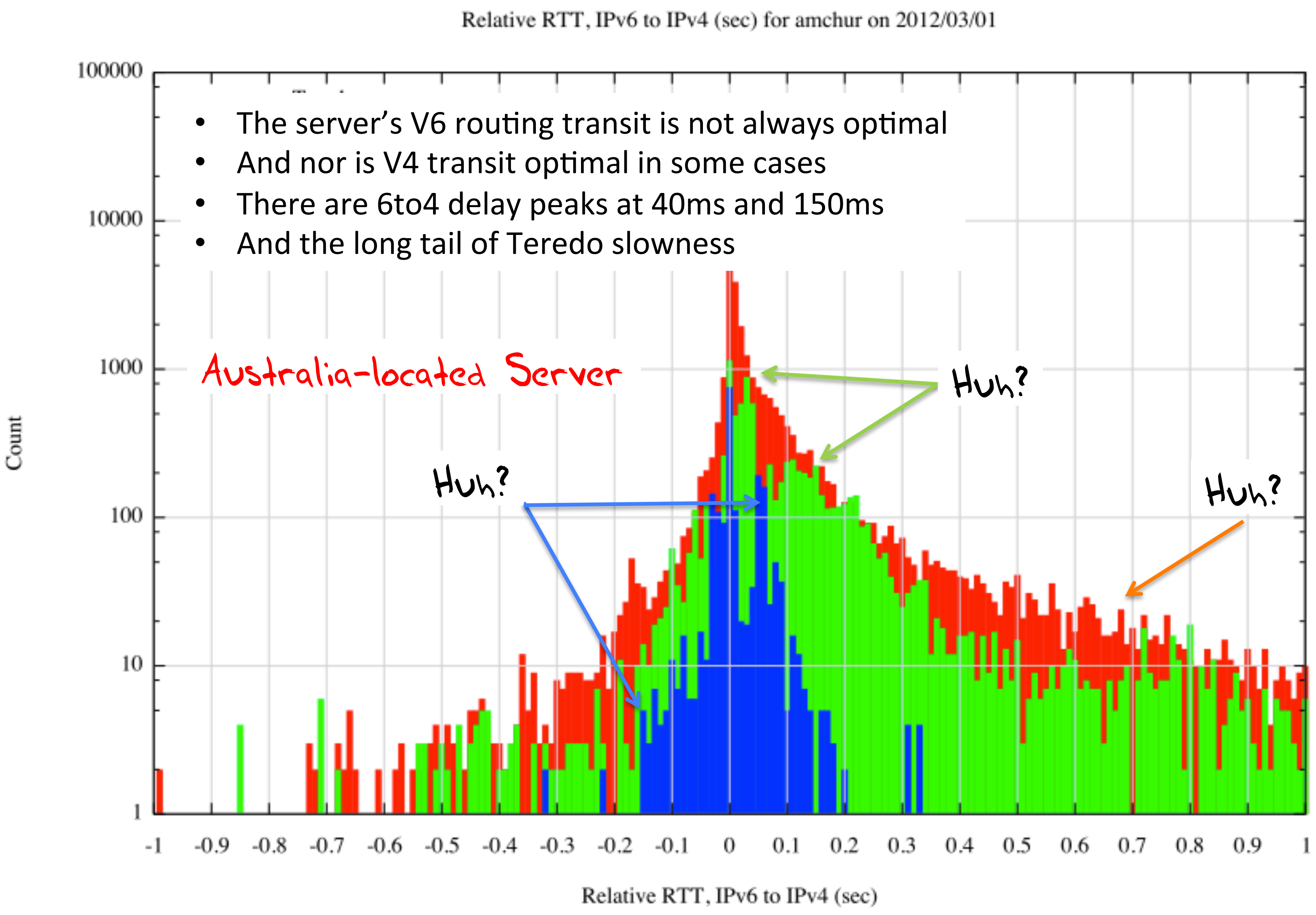

Relative RTT, IPv6 to IPv4 (sec) for bilby on 2012/03/01
Count
Relative RTT, IPv6 to IPv4 (sec)
Teredo
6to4
global unicast
100000
10000
1000
100
10
1
-1
-0.9
-0.8
-0.7
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Europe-located Server
This is unexpected!

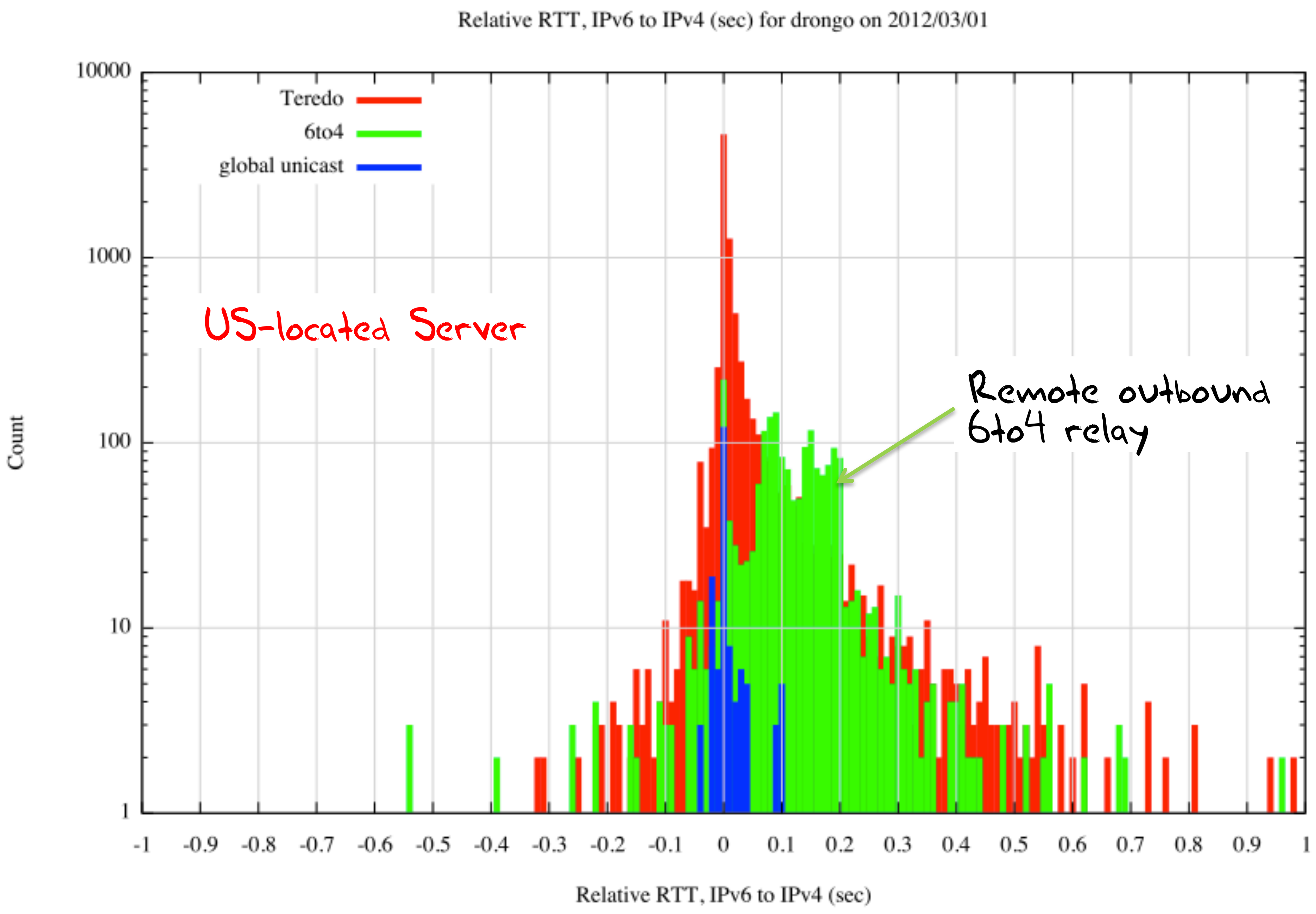
Why is V6 faster in some cases?

- We see some sessions that have faster V6 RTTs than their paired IPv4 counterpart
  - Because IPv6 is faster?
    - This is possible – there are some strange IPv4 paths out there
    - But why would a Teredo SYN exchange be faster than a native IPv4 SYN exchange?
  - Becuase IPv4 is slower?
    - Is this related to the behaviour characteristics of some CPE based NATs and their handling of NAT bindings during a a SYN exchange?

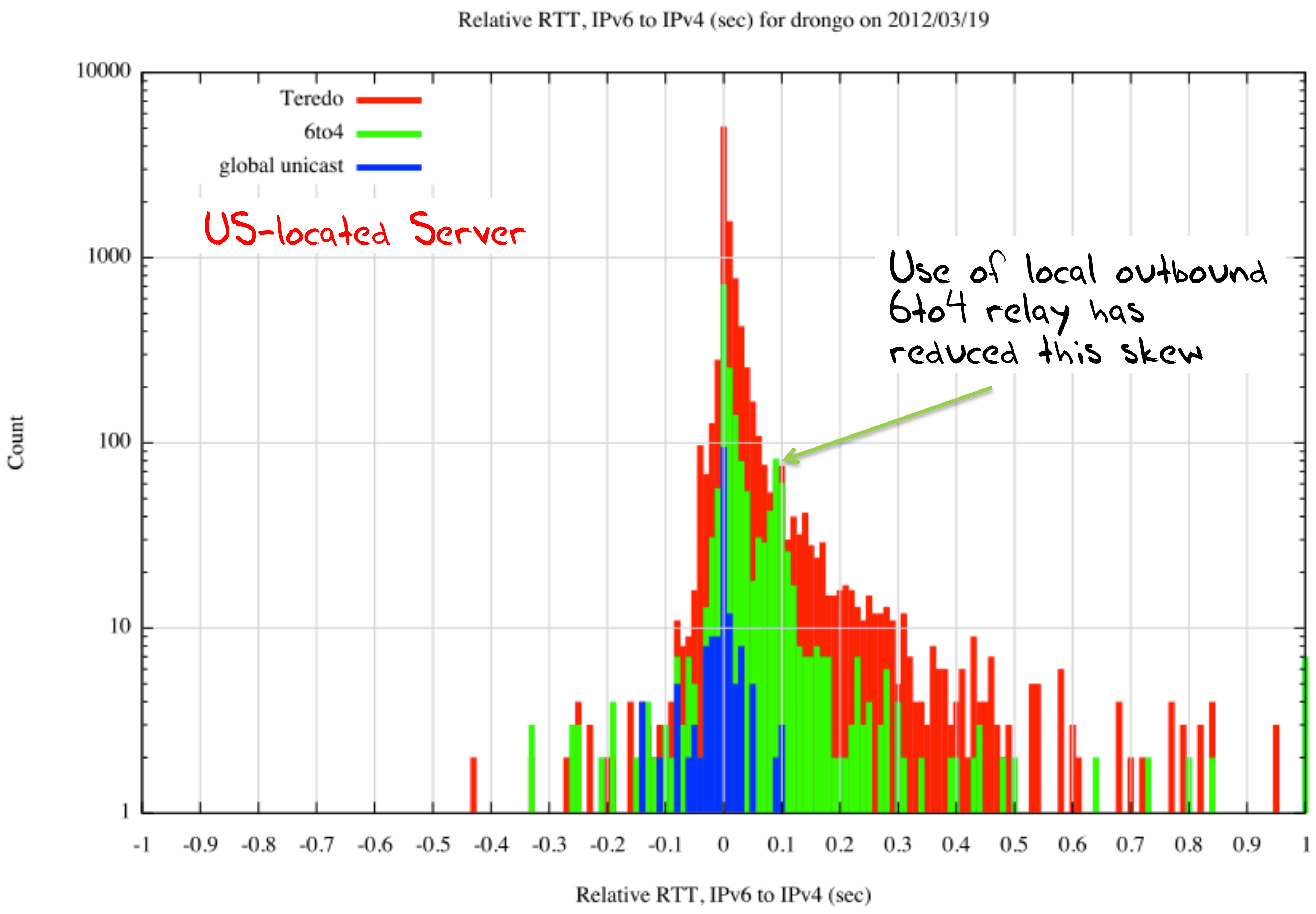

Relative RTT, IPv6 to IPv4 (sec) for amchur on 2012/03/01
Teredo
6to4
global unicast
Count
Relative RTT, IPv6 to IPv4 (sec)
100000
10000
1000
100
10
1
-1
-0.9
-0.8
-0.7
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Australia-located Server
Huh?
Huh?
Huh?

- The server's V6 routing transit is not always optimal
- And nor is V4 transit optimal in some cases
- There are 6to4 delay peaks at 40ms and 150ms
- And the long tail of Teredo slowness





Relative RTT, IPv6 to IPv4 (sec) for drongo on 2012/03/01
Teredo
6to4
global unicast
US-located Server
Remote outbound 6to4 relay
Count
10000
1000
100
10
1
-1
-0.9
-0.8
-0.7
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Relative RTT, IPv6 to IPv4 (sec)


Relative RTT, IPv6 to IPv4 (sec) for drongo on 2012/03/19
Teredo
6to4
global unicast
US-located Server
Use of local outbound 6to4 relay has reduced this skew
Count
10000
1000
100
10
1
-1
-0.9
-0.8
-0.7
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Relative RTT, IPv6 to IPv4 (sec)

Observations

- Is IPv6 as fast as IPv4?
  - If you are native in IPv6, then, yes!
  - The use of tunnels and overlays can make this worse in some cases, but, in general, V6 is as fast as V4

Observations

Is IPv6 as robust as IPv4?

- Sadly, No
- The base failure rate of V6 connection attempts at 5% of the total V6 unicast traffic volume is simply unacceptable as a service platform
- But its not in the core network. It appears that this is mainly self-inflicted with local edge firewall filter settings that trap V6 packets

How Should Browsers Behave?

One view is to place both protocols on equal footing in a parallel connection environment, using a "SYN-ACK race" with parallel DNS and TCP session establishment

- E.g. Firefox with fast retransmit

Or reduce the server load by using a "DNS race" and take whichever answers first, but prepare for failover using a very aggressive timeout

- E.g. Chrome with 300ms failover timer

Or use local heuristics to estimate which is faster and failover within 1 RTT interval

- E.g. Safari + Mac OS X >= 10.7

How Should Browsers Behave?

One view is to place both protocols on equal f
parallel connection environment. usi
parallel DNS and TCP sessi

– E.g. Firefox

ace" and take
pare for failover using a very

ome with 300ms failover timer

Or use local heuristics to estimate which is faster and failover within 1 RTT interval

– E.g. Safari + Mac OS X >= 10.7

None of these are that bad – they are all very fast. The trade off is slightly higher number of connections with the server against speed and robustness of the session

But it’s still not enough!

Many access providers see their immediate future as having to deploy IPv6 across their infrastructure, and at the same time field CGNs

But how \$big\$ does the CGN need to be?

> Generically, the CGN needs to be as big as the residual preference for using IPv4 in dual stack scenarios

So how can we help this story along?

How Should Browsers Behave?

- Fire off the DNS queries in parallel
- If the DNS returns AAAA and A records, fire off a V6 connection attempt first
- Use a *reasonably aggressive* fallback timer to trigger V4 connection

    E.g. Chrome with 300ms failover timer

    E.g. Safari + Mac OS X with RTT-derived timer

How Should Browsers Behave?

– Fire off the DNS queries in parallel

– If the DNS returns AAAA and off a V6 connection

onably aggressive fallback timer

E.g. Chrome with 300ms failover timer

E.g. Safari + Mac OS X with RTT-derived timer

“Biased, but still Pleasantly Amused Eyeballs”!

Thank You

Questions?

APNIC labs