Stacking it Up

Experimental Observations on the operation of Dual Stack Services

Geoff Huston

IETF-80
March 2011

End-to-End Service Measurements

- Examine IPv6 / IPv4 use from the perspective of a service delivery platform (web server)

- IPv6 is used by clients only when all the various IPv6 infrastructure components support IPv6, otherwise the client will fall back to IPv4 use
- Service metrics for IPv6 are reflective of end-to-end IPv6 capability

Methodology

Test every web client with 4 different retrieval tasks of a 1x1 pixel image:

- V6 only
- Dual-Stack
- V4 Only
- V6 Only Literal (no DNS)

Take just one test result for each unique source address per 24 hours

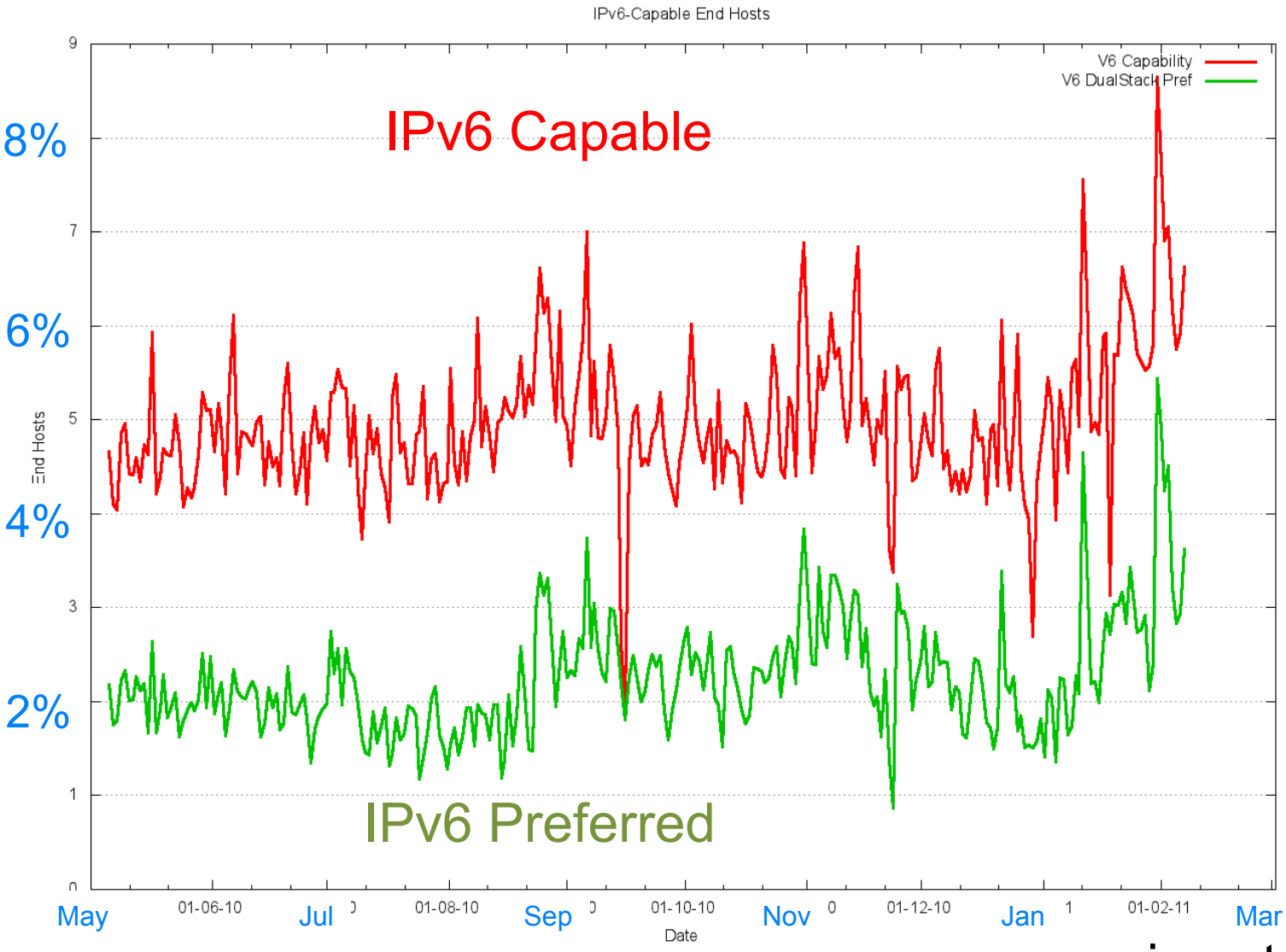
Look at retrieval rates, failure behaviour and transaction times

Use server packet dump and and web logs as the basis of the analysis

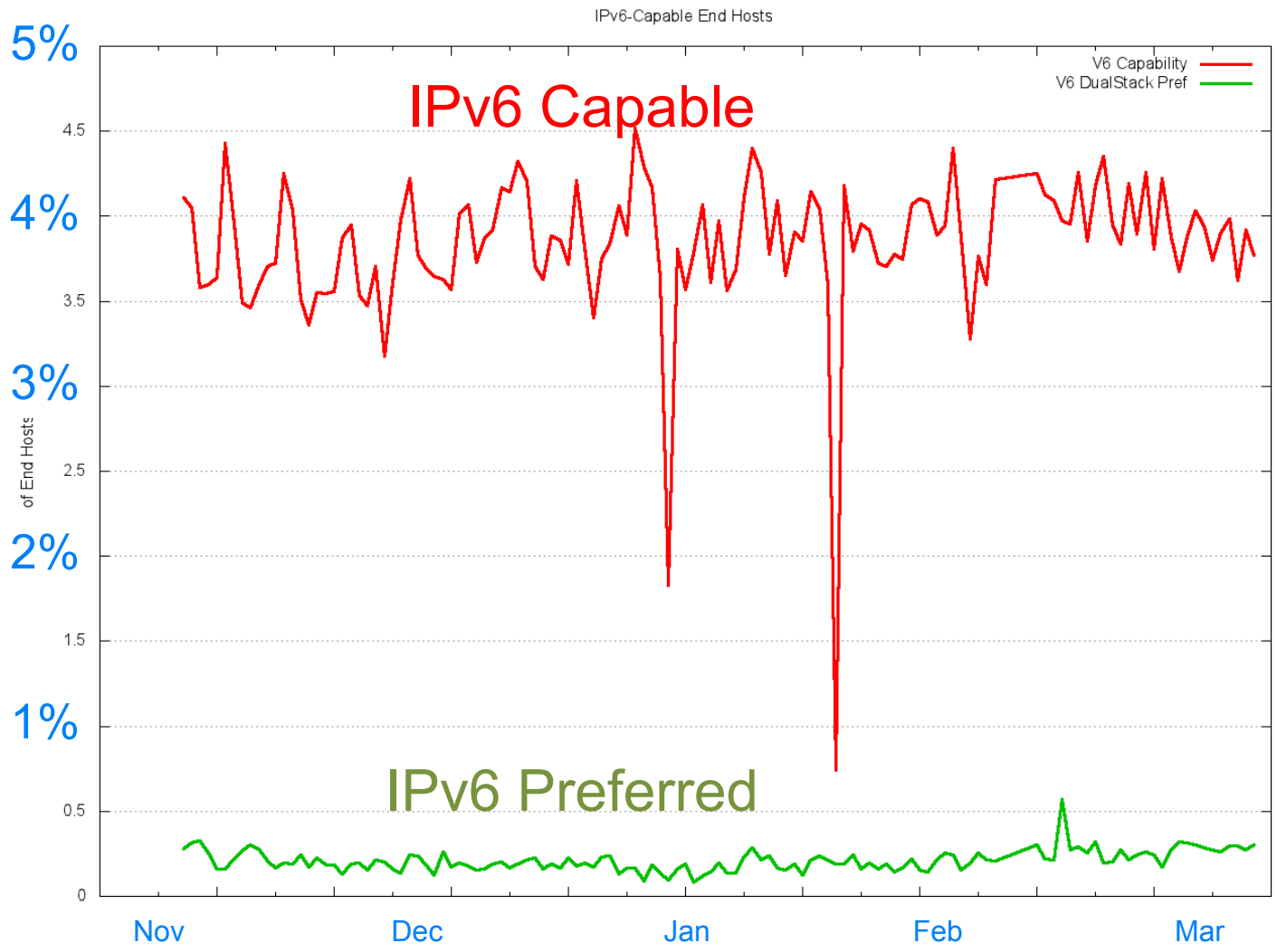
Access Combinations

V4	V6	Dual	Node Type
✔	✖	V4	V4-Only
✖	✔	V6	V6-Only
✔	✔	V6	V6-Preferred
✔	✔	V4	V6-Capable (V4-Preferred)
✔	✖	✖	Dual-Stack Loss

IPv6: "could" vs "will"



IPv6: "could" vs "will"



Site "C"

Where are we with IPv6?

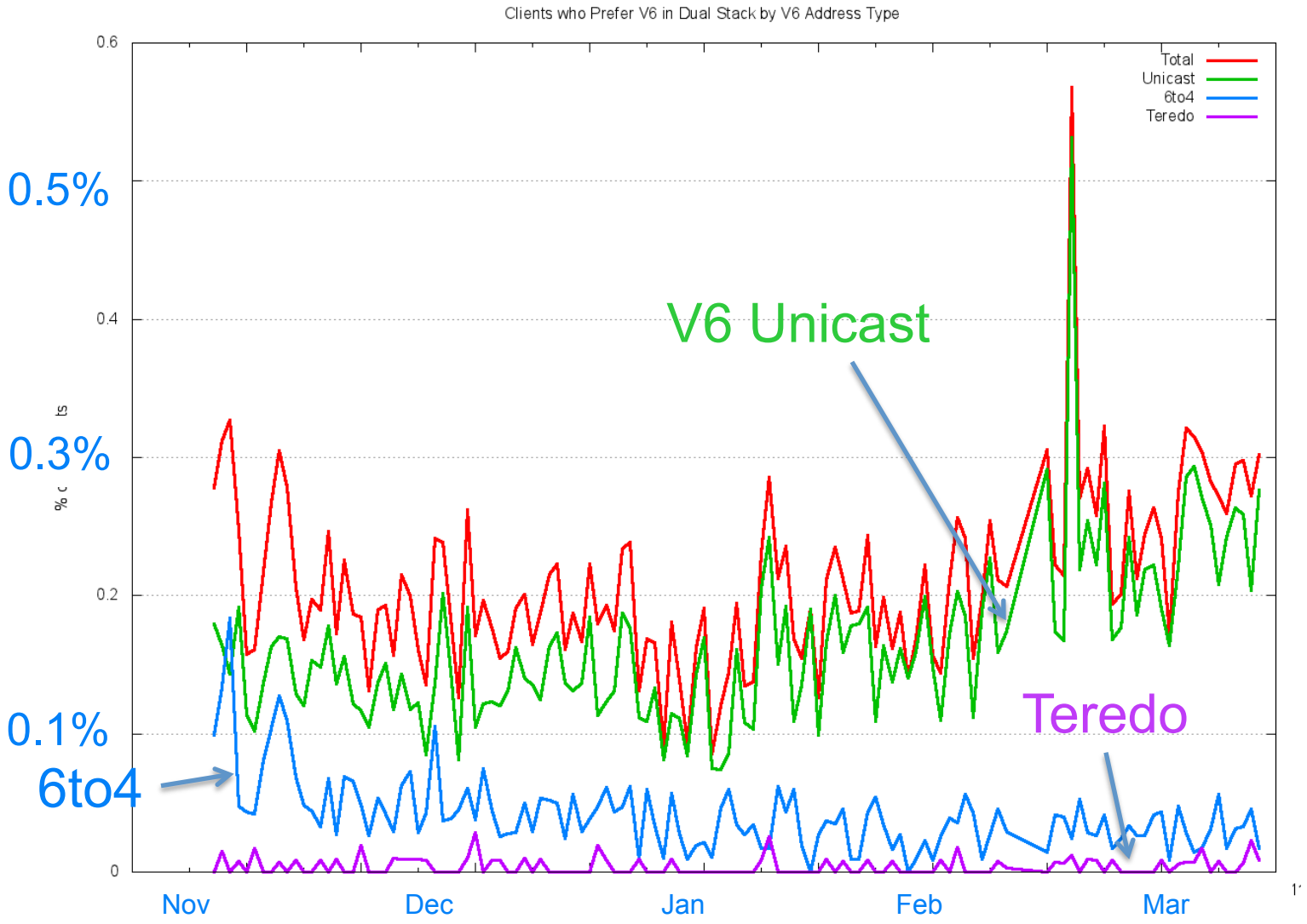
The ‘size’ of the IPv6 deployment in terms of end-to-end host IPv6 preference is around 0.2% of the total number of Internet end hosts at present

However, a further 4% of hosts can use IPv6, even though they prefer IPv4 in dual stack mode, using auto-tunnel access

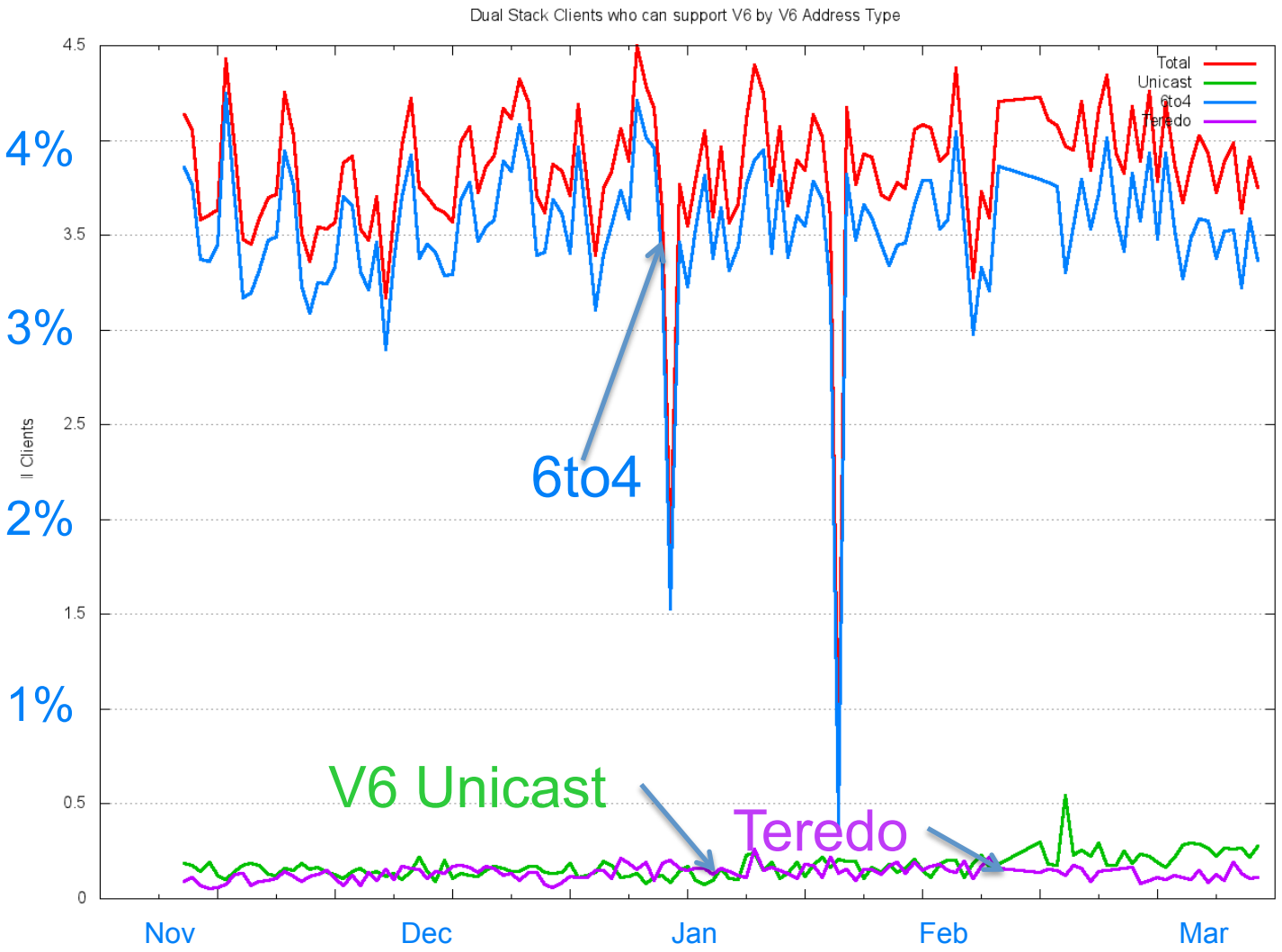
Why is there so much “hidden” IPv6 capability?

Why is the number of client hosts who are *capable* of performing an end-to-end IPv6 object retrieval 20 times greater than the number of client hosts who *prefer* to use IPv6 in a dual stack context?

Dual-Stack V6 Preferred by Address Type



Dual-Stack V4 Preferred by Address Type



Native vs Tunnels

- Most hosts with unicast IPv6 generally prefer V6 in a dual stack scenario
- Hosts with 6to4 auto-tunnel capability

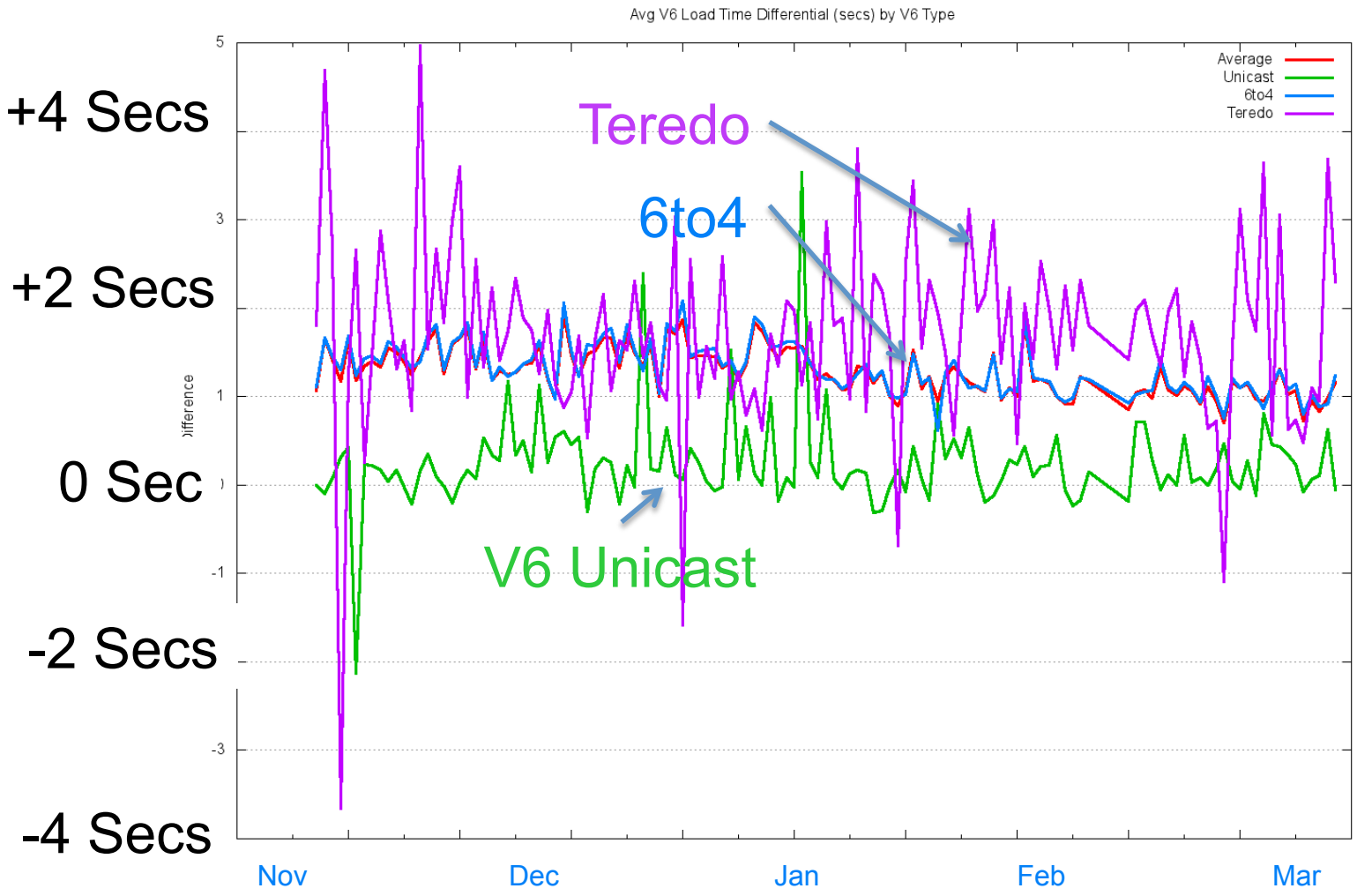
  stack scenario

Native vs Tunnels

- Older versions of dual stack software in hosts preferred IPv6 over IPv4 in all situations, including auto-tunnels
  - This resulted in very slow and erratic performance when accessing some dual stack servers due to the local IPv6 failure timers
    - For example, Windows XP takes 20 seconds to recover a connection if a 6to4 connection is not functioning correctly
- Recent OS releases have de-pref'ed auto-tunneled IPv6 below that of IPv4

Performance Observations

Performance and Tunnels



Performance and Tunnels

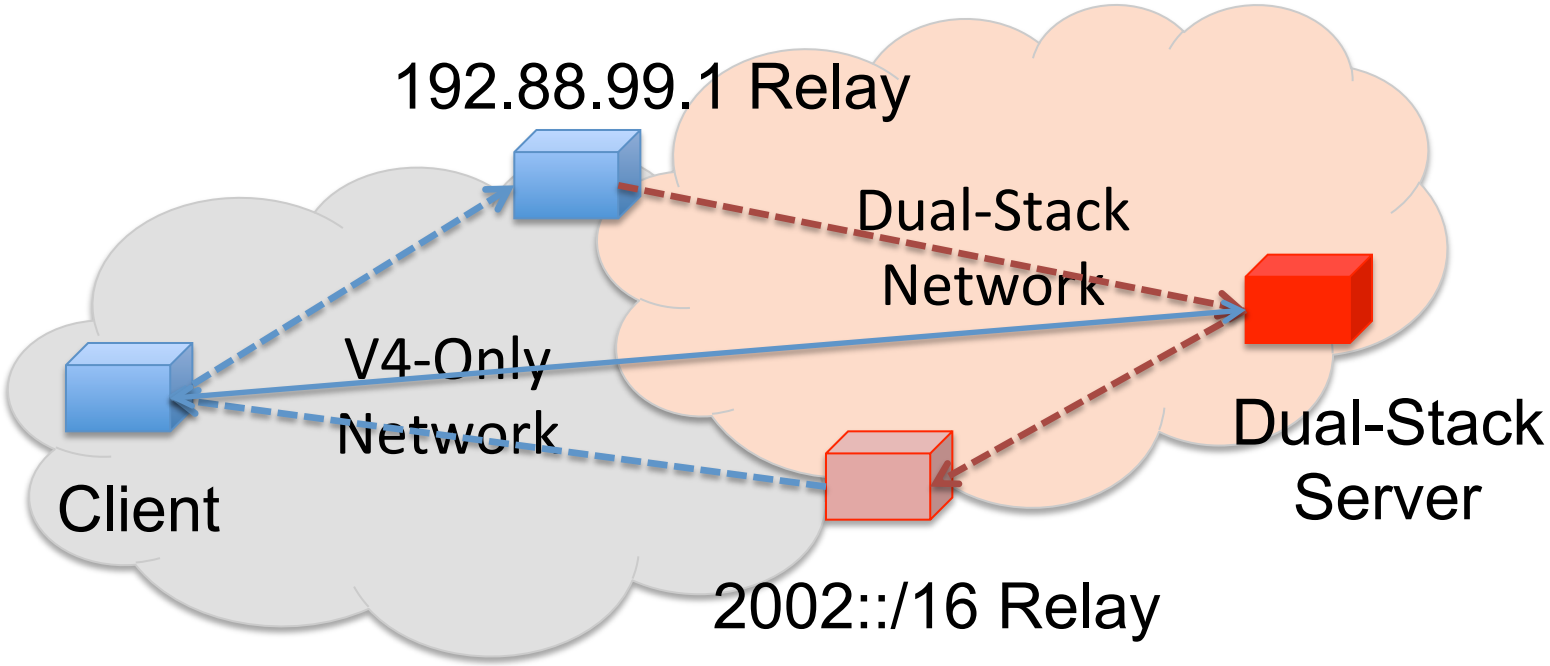
- Unicast IPv6 performance is on average equivalent to IPv4 performance for web object retrieval
- Auto-tunnel performance is on average considerably worse
  - Teredo is highly variable with **1 – 3 seconds** of additional delay per retrieval
  - 6to4 is more consistent with an average **1.2 seconds** additional delay per retrieval

Performance and Tunnels

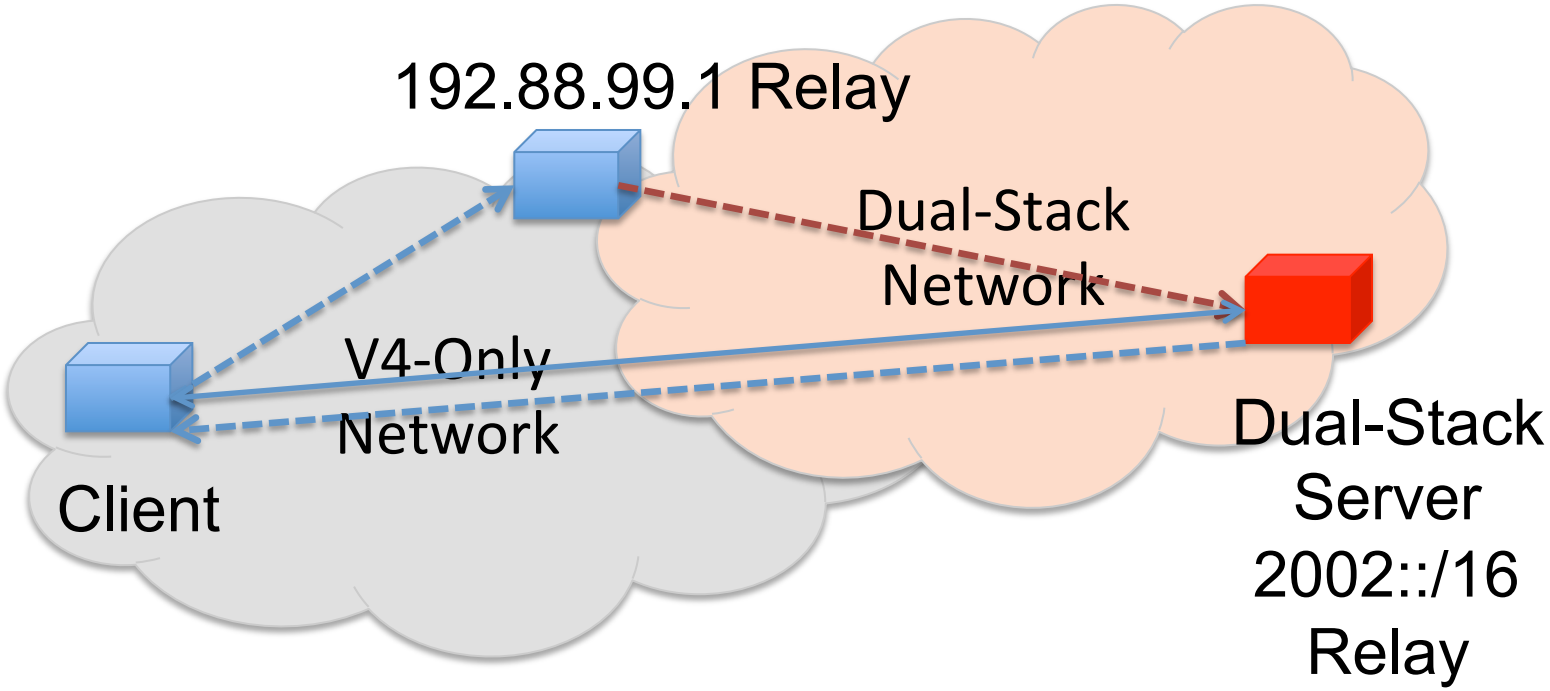
Two causes of incremental delay:

- Tunnel setup time
  - Stateful Teredo tunnels require initial packet exchanges to set the tunnel up (min 1 x RTT)
- Tunnelling can extend the RTT delay
  - addition of tunnel relays between the source and destination
  - This is exacerbated when the forward and reverse paths are asymmteric

6to4 Packet Path

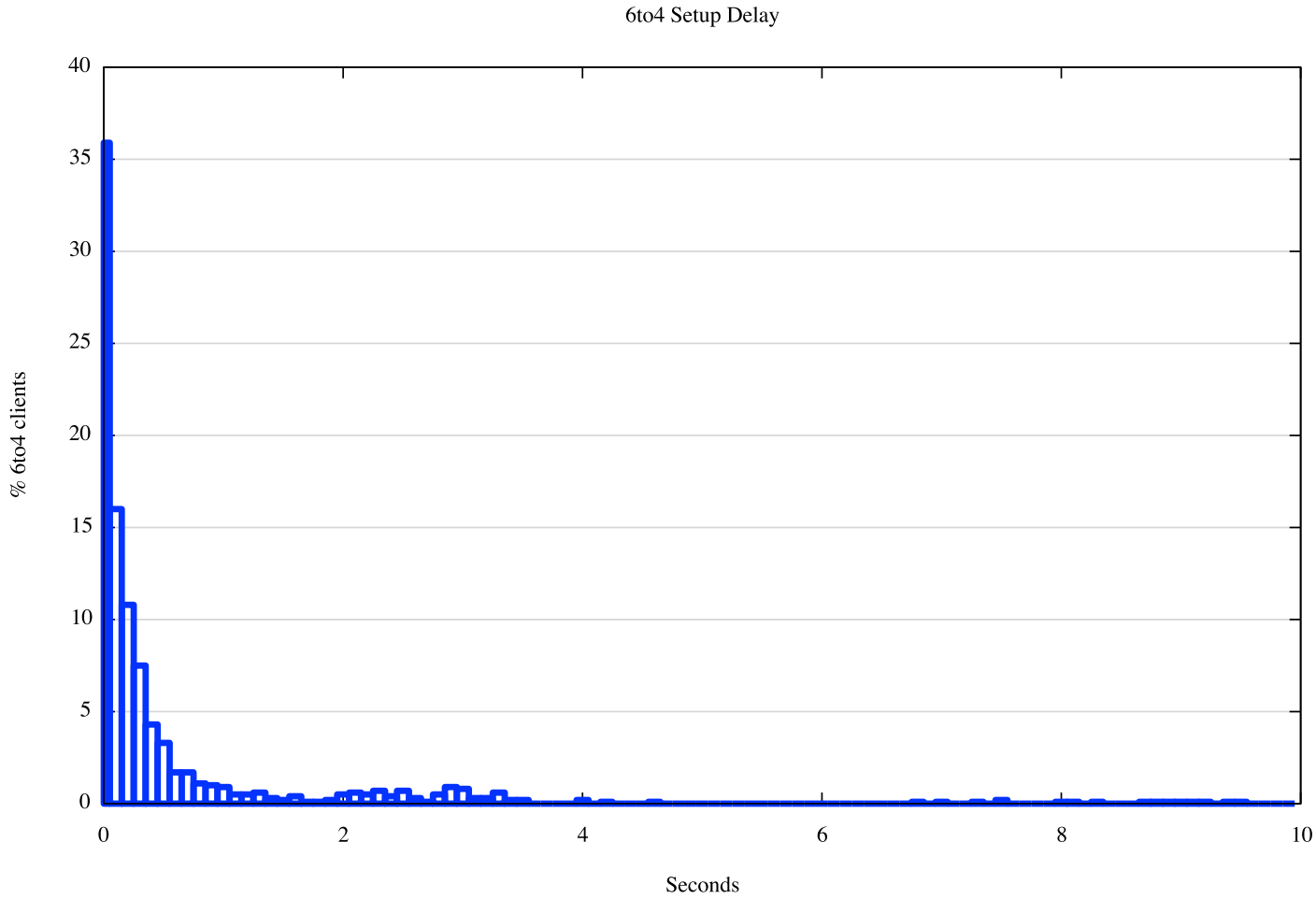


Partial Mitigation of 6to4 Packet Path



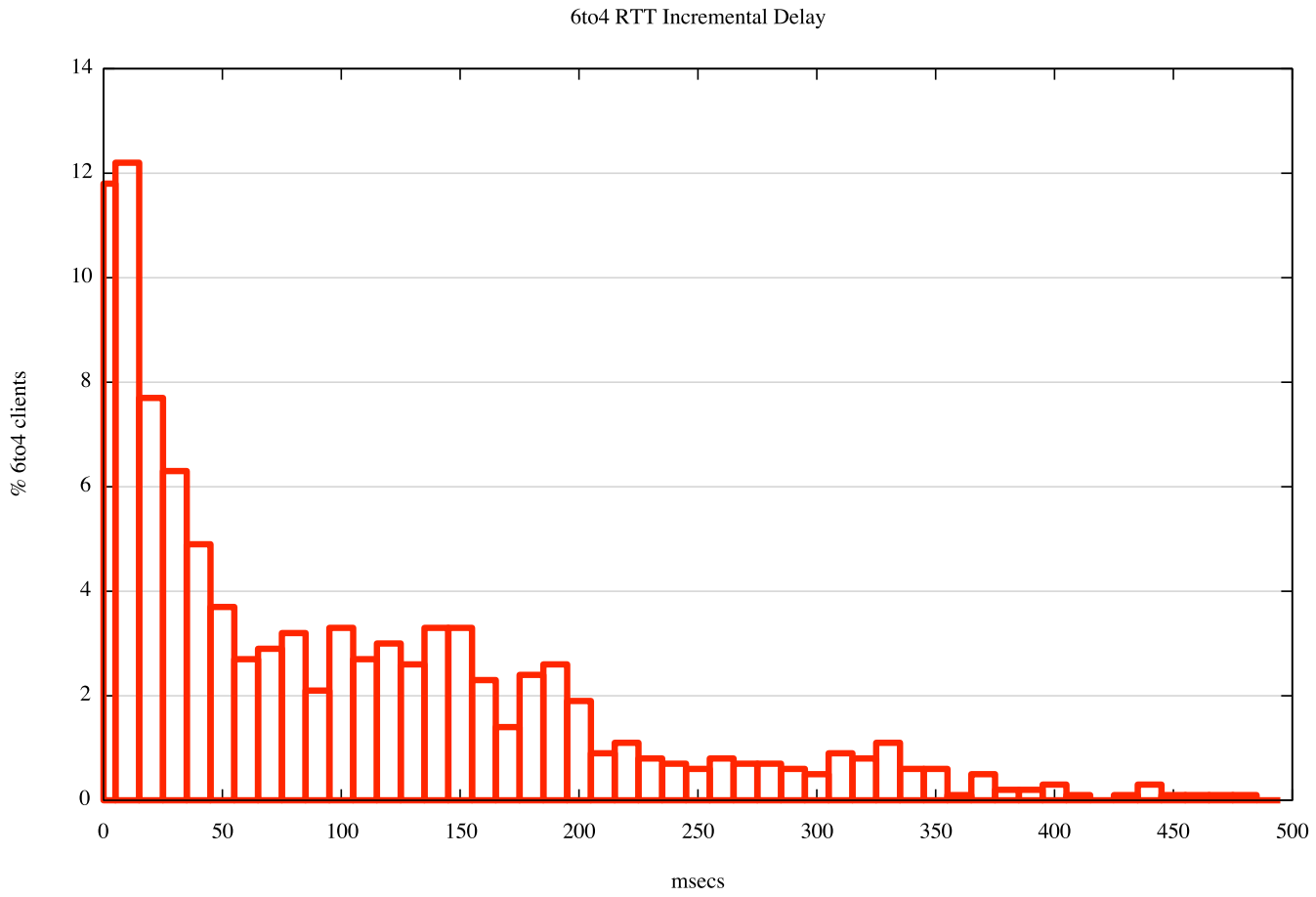
6to4 Performance

Setup Time



6to4 Performance

Tunnel RTT Cost



6to4 Relative Performance

6to4 adds an average of 1.2 seconds to the object retrieval time

- note this is one-way (as the server has a local 6to4 relay for the response traffic, so the 6to4 response path is the same as the V4 path)
- that's a very long transit time if this is just added transit time
- There may be a congestion load delay added in here
- But the level of 6to4 traffic is very low, so congestion overload is unlikely

Teredo vs 6to4

What we see:

- 4% of hosts use 6to4 (native V4, auto-tunnel)
- 0.1% of hosts use Teredo (NAT V4, auto-tunnel)

But why so little Teredo?

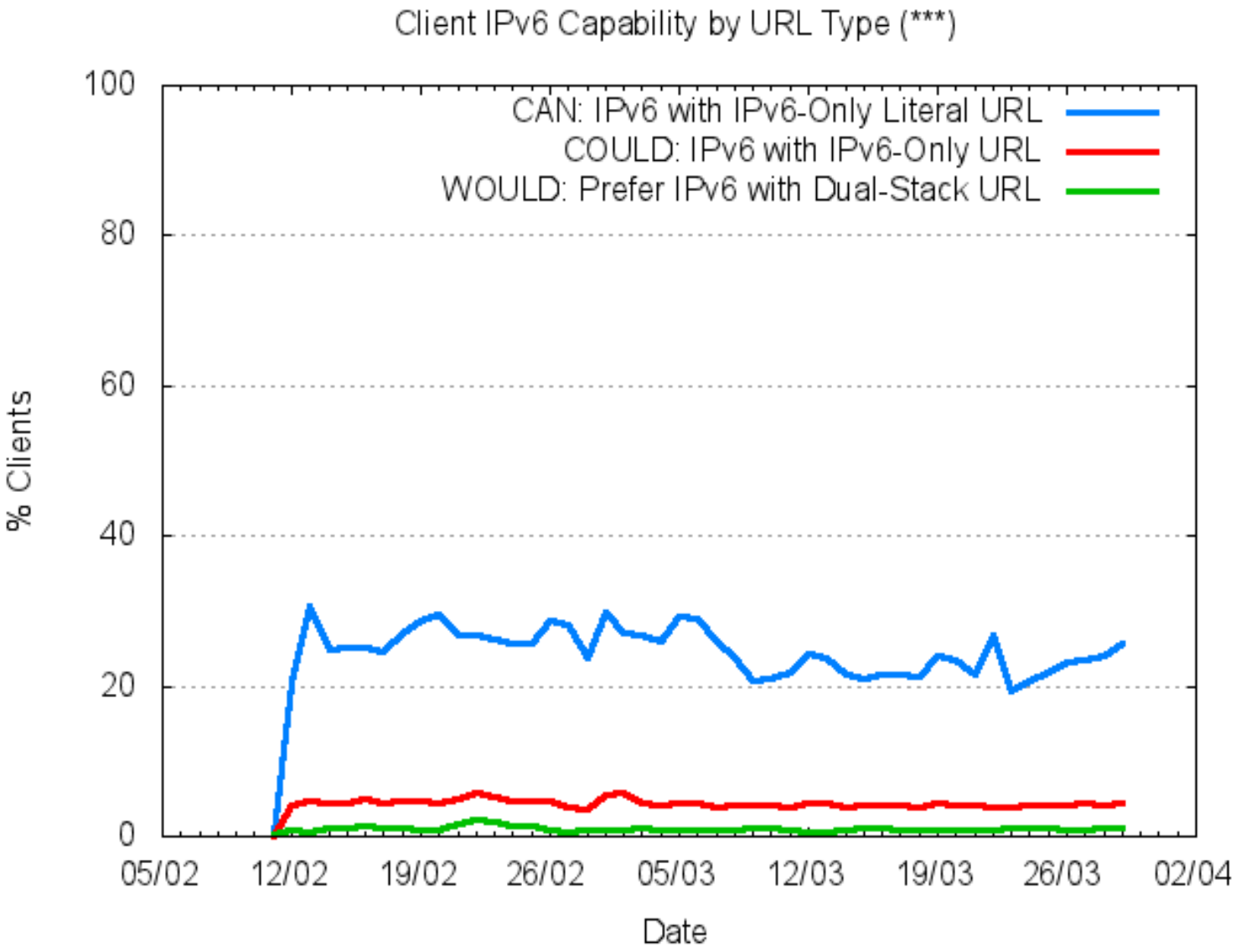
- Windows Vista and Windows 7 gethostbyname() will not query for a AAAA record if the only local IPv6 interface is Teredo
- Can we expose latent Teredo capability?

Exposing Teredo

Use an IPv6 literal as the object URL:

http://[2401:2000:6660::f003]/1x1.png

Exposing Teredo



Exposing Teredo

Use an IPv6 literal as the object URL:

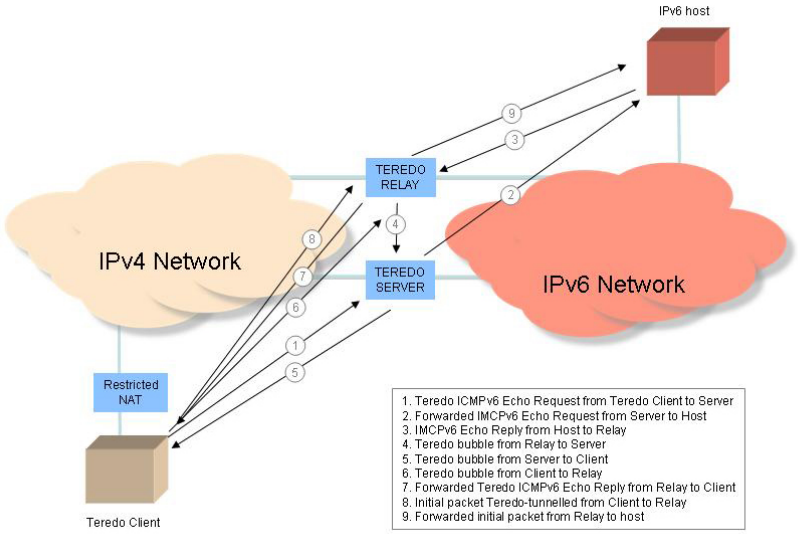
http://[2401:2000:6660::f003]/1x1.png

- In the context of the experimental setup it was observed that ~30% of the client base successfully fetched this IPv6 URL using Teredo!

- Conversely, 70% of the clients did not manage a successful object retrieval of this URL

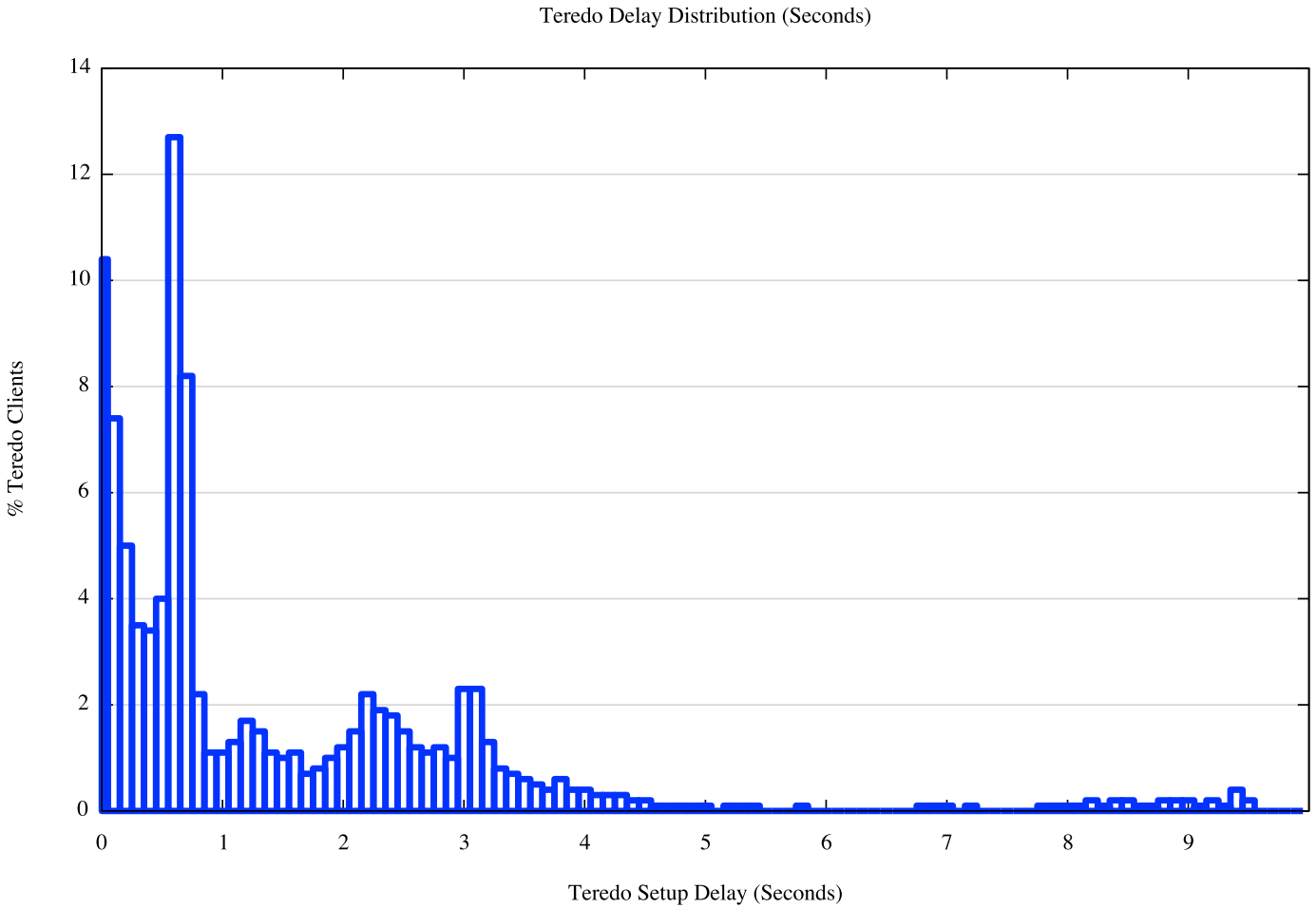
Performance and Tunnels

Teredo adds a further performance penalty in the form of state setup between the Teredo relay and the client



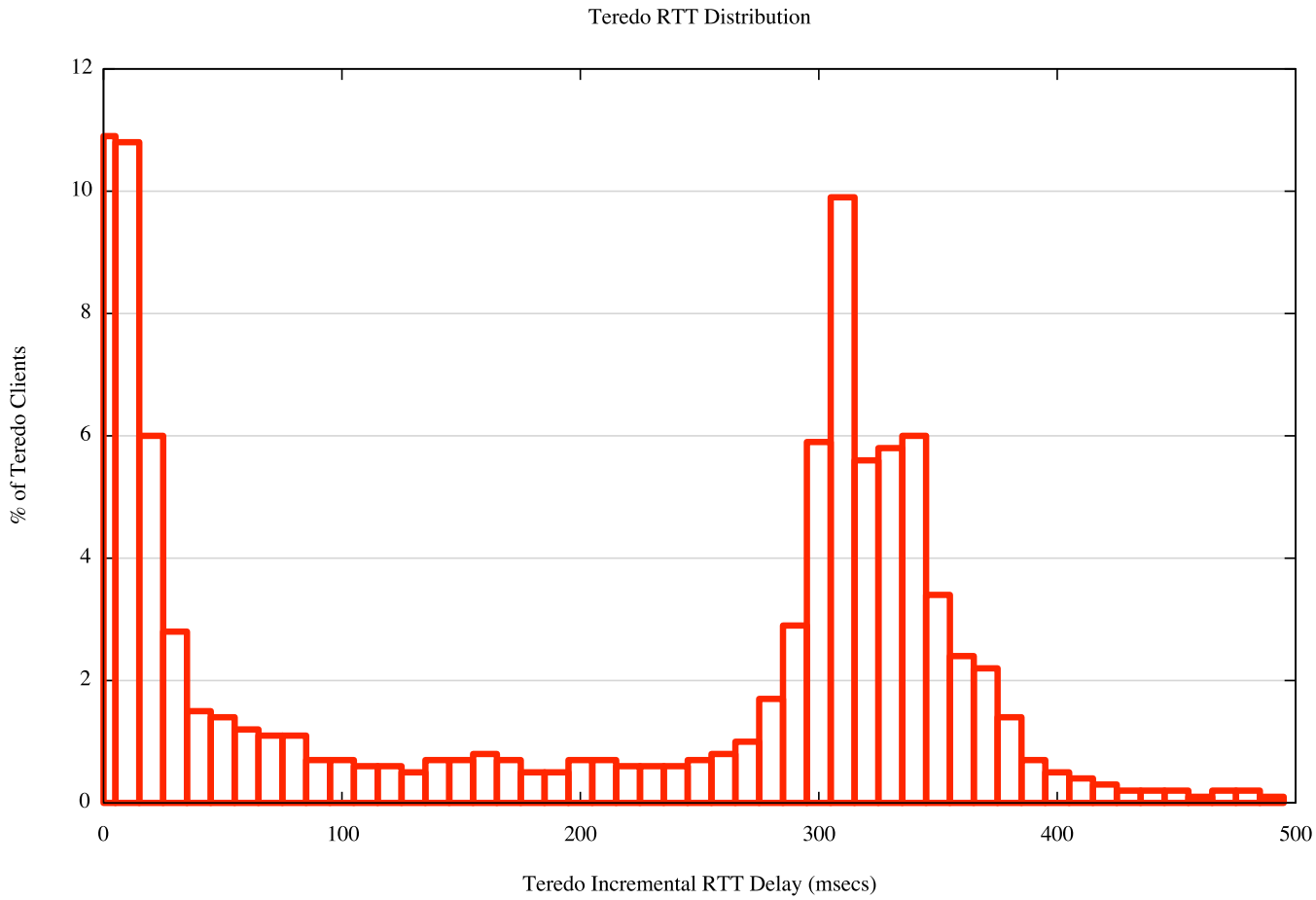
Teredo Performance

Tunnel Setup Time



Teredo Performance

Tunnel RTT Cost



Teredo Relative Performance

Teredo adds an average of 1 - 3 seconds to the object retrieval time

- Teredo setup takes between 0.6 second to 3 seconds
- Average RTT cost of Teredo is 300ms
- Object retrieval takes ~3 RTT intervals to complete
- Total time cost is some 2 seconds on average

IPv6 Performance

- Unicast IPv6 appears to be as fast as IPv4 for object retrieval
- Auto-tunnelling IPv6 attracts some performance overheads
  - these are strongly context dependent
  - widespread deployment of 6to4 relays and Teredo relays and servers would mitigate this, to some extent
  - Dual Stack servers may want to consider using local 6to4 relays to improve reverse path performance for auto-tunnelling clients

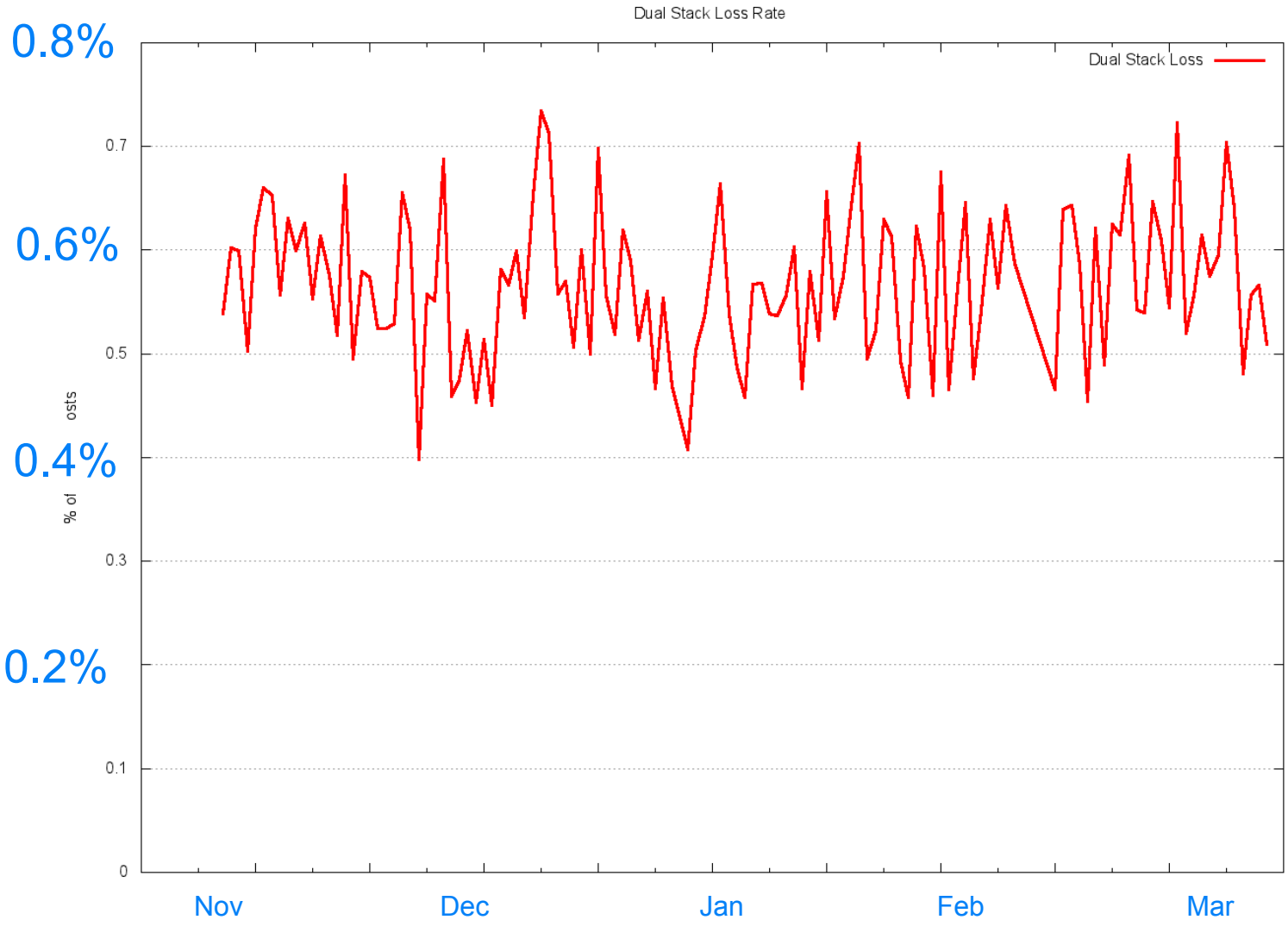
Failure Observations

Dual Stack Failure

How many clients retrieve the V4 only object but DON’T retrieve the Dual Stack objects?

i.e. how many clients exhibit “Dual Stack Failure”?

Dual Stack Failure Rate



Dual Stack Failure

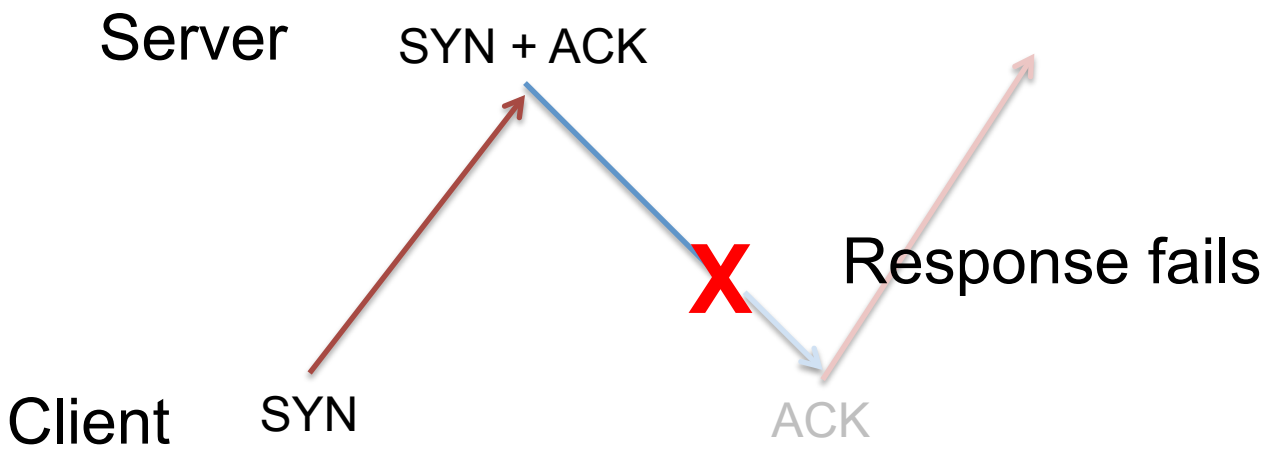
This 0.6% of clients is the rate of failure of IPv4 clients to retrieve a dual stack object

But this is not a reliable metric of underlying communication failure

- This is the rate of failure of the client to retrieve a dual stack object from within a javascript code object
- The client may:
  - Not execute the javascript at all
  - User reset of the retrieval before completion
  - In addition to the failure to fallback to IPv4 retrieval

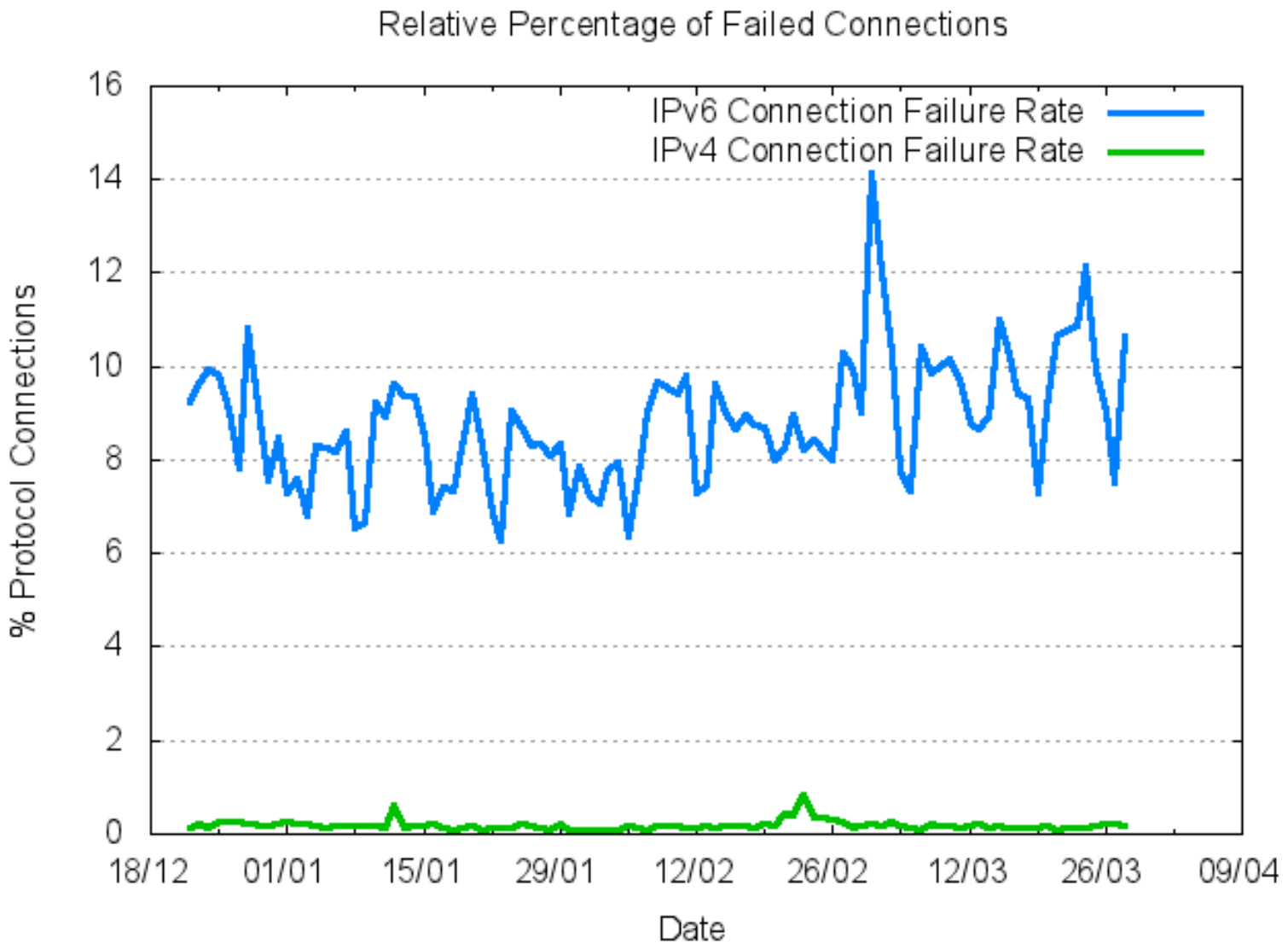
Connection Failure

To attempt to look more precisely for **some** instances of connection failure, lets looking for connections that fail after the initial TCP SYN

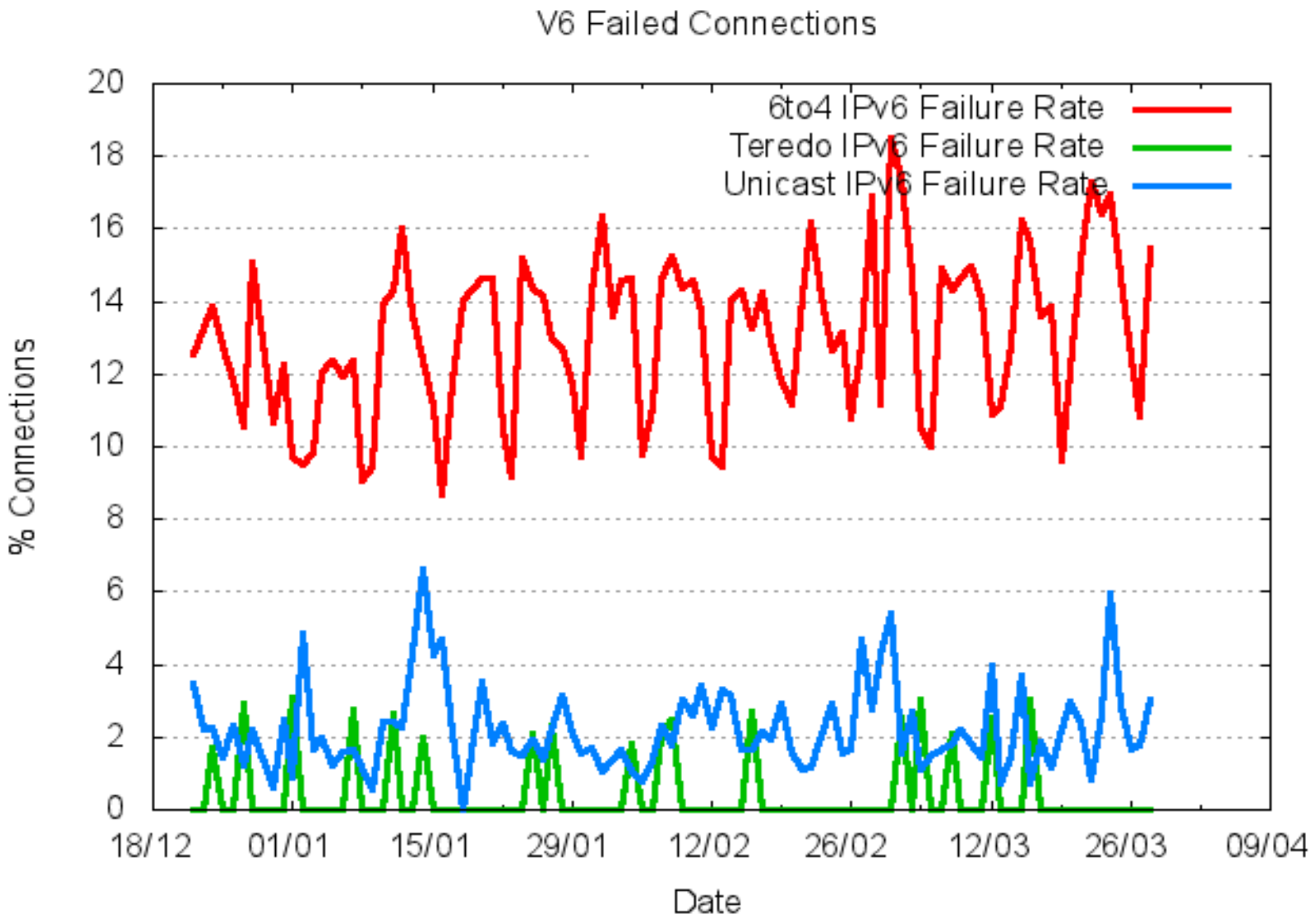


Note that this approach does not detect failure of the initial SYN packet, so the results are a lower bound of total connection failure rates

Connection Failure

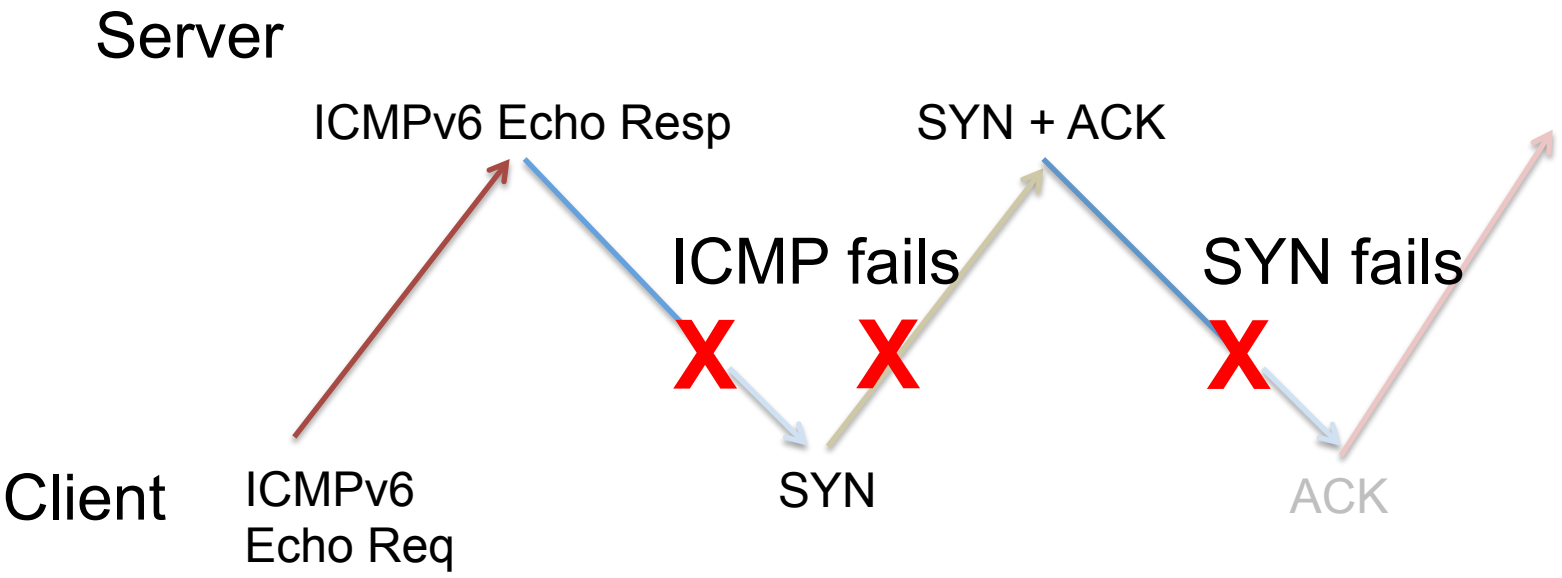


IPv6 Connection Failure



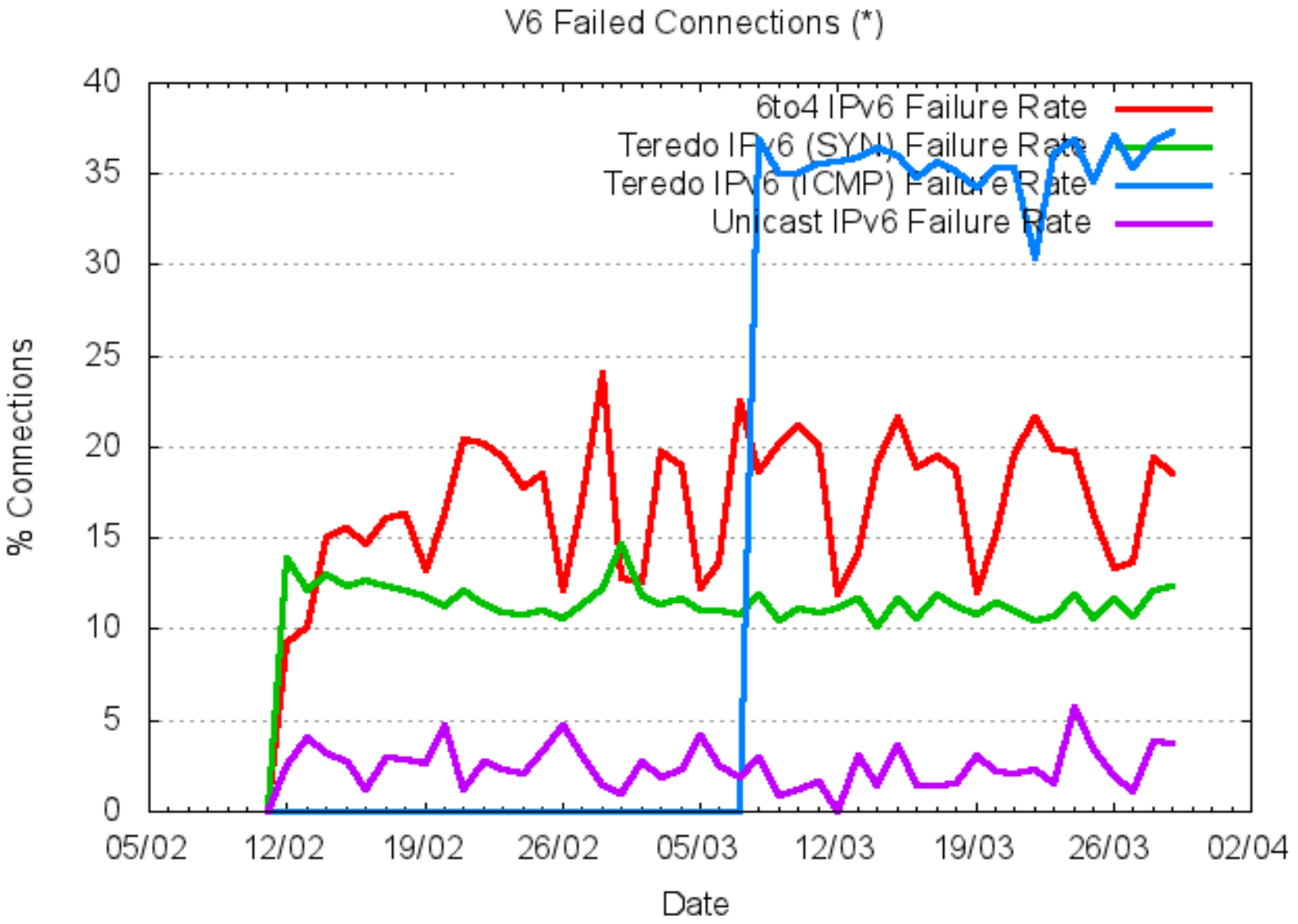
Teredo Connection Failure

Teredo uses an initial ICMPv6 exchange to assist in the Teredo Server / Relay state setup



Note that this approach does not detect failure of the initial ICMPv6 echo request , so the results are a lower bound of total connection failure rates

IPv6 Connection Failure using V6 Literal



IPv6 Connection Failure

- Some **12% - 20% of 6to4** connections fail!
  - This is a very high failure rate!
  - The failure is most likely a protocol 41 filter close to the client that prevents incoming 6to4 packets reaching the client
- Some **38% of Teredo** connections fail!
  - Again this is a very high failure rate
  - Local ICMP Filters + ???? SYNACK Filters
- Some **2%-5% of unicast** IPv6 connections fail!
  - This rate is better than auto-tunnels, but is still 20x the rate of IPv4 connection failure

Conclusions

What can we say about the performance and robustness of a Dual Stack network environment as a result of these observations?

For an Online Service…

Converting a service to operate as a Dual Stack service is a viable option in today's environment

But:

- a small fraction of existing clients will experience a much slower service
- a very small fraction of existing clients will fail to connect to the dual stack service at all

What about IPv6-Only Services?

Is an IPv6-only service a viable option today?

Not really.

- Only ~4% of the existing client base would successfully connect to an IPv6-only service
- End-host auto-tunnelling is **not** a solution
  - Auto-tunnelling appears to encounter many more performance and reliability problems than it solves in terms of IPv6 connectivity
  - Auto-tunnelling is **not** proving to be a useful mainstream transition tool for IPv6

Can I run these tests?

Yes …

http://labs.apnic.net