One Protocol Good, Two Protocols?

Geoff Huston
APNIC
What does a TCP client do in a dual stack environment?

Is this behaviour better - or worse - than an IPv4-only TCP client?
Let’s start by looking at what happens in a single protocol world
Good TCP!

SYN

C <-- SYN + ACK --> S

ACK
Bad TCP!
Bad TCP

What if you don’t get back a SYN+ACK?

– Most TCP stack implementations will retry sending the original SYN packet
– And again
– And again
TCP SYN Attempts

Windows:
- wait 3 seconds, resend the SYN
- wait 6 seconds, resend the SYN
- wait 12 seconds, report connection failure

19 seconds, 3 SYN packets
TCP SYN Attempts

FreeBSD:
wait 1 second, resend the SYN
wait 1 second, resend the SYN
wait 1 second, resend the SYN
wait 1 second, resend the SYN
wait 2 seconds, resend the SYN
wait 4 seconds, resend the SYN
wait 8 seconds, resend the SYN
wait 16 seconds, resend the SYN
wait 32 seconds, resend the SYN
wait 8 seconds, report connection failure

75 seconds, 11 SYN packets

sysctl net.tcp.keepinit = 75000
TCP SYN Attempts

Linux:

wait 3 seconds, resend the SYN
wait 6 seconds, resend the SYN
wait 12 seconds, resend the SYN
wait 24 seconds, resend the SYN
wait 48 seconds, resend the SYN
wait 96 seconds, report connection failure

189 seconds, 6 SYN packets

sysctl net.ipv4.tcp_syn_retries = 5
Bad TCP

Why are all these implementations so slow to signal failure?

– These settings date back more than a decade
– They reflect a connection strategy where persistence in attempting to connect had few downsides
– There was no Plan B!
TCP in a Dual Stack Environment

What changes should we make to TCP-based applications in an environment where there IS a Plan B?

– What do we do now?
– Can we do better?
Dual Stack Behaviour: V1

IPv6 First:

Unconditional preference for IPv6 over IPv4
Dual Stack Behaviour: V1

IPv6 First:
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 Behaviour: V1

Why this unconditional preference for IPv6?

– The dual stack transition plan’s last phase is the turning off of IPv4 when all the network is IPv6 capable
– But if hosts still prefer to use IPv4 then this final phase will never complete
– The IPv6 preference is designed to maximize IPv6 use through the transition
Dual Stack Failure: V1

What if the IPv6 connection attempt does not elicit a response?
Then you fall back to use IPv4

How long will you wait before decide that this has failed and you need fall back?
As long as it takes for the Operating System’s TCP system to fail
- Windows: 3 SYN packets, 19 seconds
- Mac OS X 6.8 and earlier: 11 SYN packets, 75 seconds
- Linux: >= 11 SYN packets, between 75 to 180 seconds

Obviously, this sucks!
Dual Stack Behaviour: V2

Native IPv6 First:
Unconditional preference for native IPv6 over IPv4

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

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?
As long as it takes for the Operating System’s TCP system to fail
Windows: 3 SYN packets, 19 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: V3
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: V3

Native IPv6 First:
Unconditional preference for native IPv6 over IPv4
(and avoid Teredo)

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
Dual Stack Connection Model

Start Point

DNS

AAAA?

A?

Sync Point

TCP

Send SYN

TCP Failover

Send SYN
This is broken!

Parallel DNS, followed by Serial TCP:

• 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 the way then send off the other horse!
You can send off both horses at once and go with whichever is fastest...
Parallel Connection Model

Start Point

DNS

AAAA?

Send SYN

A?

Send SYN

TCP

Wait For First SYN/ACK
Parallel Connection Model

"Happy Eyeballs"

Start Point ➔ AAAA? ➔ Send SYN ➔ Send SYN/ACK ➔ TCP

Wait For First SYN/ACK
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:
  nettop -n -m route

• When DNS queries return both A and AAAA records initiate a connection using the protocol with the lowest current average RTT
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
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
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

  – What happens if there are multiple addresses for the name?
    • Then you try each address in turn, using the extended 75 second TCP timeout
Safari and Mac OSX 10.7 and later

- If the connection is not established within the Round Trip Time interval then fire off a connection attempt in the other protocol – i.e. use a very aggressive Homeout to trigger protocol fallback.

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

What happens if there are multiple addresses for the name?
- Then you try each address in turn, using the extended 75 second TCP timeout
Happyish 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

Yes, 300ms is arbitrary. But assuming that a fast DNS response equates to a fast data path RTT is equally arbitrary!
Firefox and Fast Failover

Happier 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-ACK race: Use the first connection to complete the SYN-ACK handshake for data retrieval
• Close off the other connection

This makes a little more sense – now the data path RTT has some influence over protocol selection, and the user connection will proceed with the protocol that completes the connection in the least time
## The bigger picture...

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

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?
- Let's 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, ISOC RIPE NCC & ISC for your assistance to conduct this experiment!
Test Volume – Number of unique tests performed per day

We are currently performing 800K – 900K unique IPv6 tests 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

– Gather connection failure statistics (where a “failure” is defined as a received SYN, but no followup ACK)

– For each successful connection couplet gather the pair of RTT measurements on the SYN-ACK exchanges
Connection Failure

Outbound SYN
Connection Failure

Outbound SYN

Busted SYN ACK
Return path
Measuring Failure

Connection Failure Rate

IPv6 Connection Failure Rate
IPv4 Connection Failure Rate
Measuring Failure

Why is this failure rate for V6 so incredibly high?

What are these v4 failure spikes?
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 passing across the addresses of the test servers.
What is going on with IPv4?

Syn Flood Attacks

bot scanning on port 80
What about IPv6?

Local Miredo Relay Failures

Why is the base failure rate of all IPv6 connections sitting at 30% - 40%?

This is amazingly bad!
V6 Failure Rate by Address Type

V6 Failed Connections

Date

% Connections

6to4 IPv6 Failure Rate
Teredo IPv6 Failure Rate
Unicast IPv6 Failure Rate
V6 Failure Rate by Address Type

V6 Failed Connections

Teredo IPv6 Failure Rate
Teredo IPv6 Failure Rate
Unicast IPv6 Failure Rate

6 to 4
Unicast

% Connect

Date

03/11 05/11 07/11 09/11 11/11 01/12 03/12 05/12 07/12 09/12
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

V6 Teredo Failed Connections (*)

- Teredo IPv6 Failure Rate
- Teredo IPv6 SYN Failure Rate
- Teredo IPv6 ICMP Failure Rate

ICMP Exchange fails to complete

ICMP completed, but SYN Exchange fails to complete

Date
It’s NAT Traversal Failure

• Teredo failure is around 35% 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
  – The ICMP failure rate comes from the limited number of UDP NAT traversal models used by the Teredo handshake protocol vs the variance of UDP NAT traversal models used in networks
  – The SYN failure rate is a result of the Teredo protocol making incorrect assumptions about the NAT’s behaviour
Working with Failure

A 35% connection failure is unworkable is *almost* all circumstances

But one particular application can thrive in this environment, and makes use of Teredo addresses – Bit Torrent

  – The massive redundancy of the data set across multiple sources reduces the sensitivity of individual session failures
  – Not many DPI interceptors are sensitive to Teredo’s V6 in V4 UDP encap
  – Microsoft continues to ship active Teredo in its Windows platform
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
Failing 6to4

V6 6to4 Failed Connections

6to4 IPv6 Failure Rate

% Connections

Date

03/11 05/11 07/11 09/11 11/11 01/12 03/12 05/12 07/12 09/12
Failing 6to4

V6 6to4 Failed Connections

Local relay failures

“Base” failure rate
6to4 Failure Rate

6to4 Connection Failure

- IN-BUILT Relay (ASIA)
- NON-ADJACENT Relay (US)
- ADJACENT Relay (EUR)

Graph showing the failure rate of 6to4 connections from November 2011 to March 2012 for different regions: US, ASIA, and EU.
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 more prevalent in corporate environments than home environments)

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% connection failure rate for 6to4 is unacceptably high!
V6 Unicast Failures

January – August 2012:
  962,737 successful V6 connecting endpoints
  22,923 failures
    That’s a connection failure rate of 2.3%!

13 clients used fe80:: link local addresses
139 clients used fc00:/7 ULA source addresses
22 clients used fec0::/16 deprecated site local addresses
16 clients used 1f02:d9fc::/16
1 client used 1f01:7e87:12:10ca::/64
1 client used a 3ffe::/16 address
7 clients used :: IPv4 –mapped addresses (10/8, 192.168/16)
7 clients used ::ffff:<IPv4>-mapped addresses

What about the other 22,717 clients?
Unicast IPv6 Failures

38 were using unallocated unicast V6 addresses
150 were using unadvertised unicast V6 addresses
22,529 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 - 2.3% of unicast V6 connections fail to complete
However, we saw wide variance across countries:

<table>
<thead>
<tr>
<th>Highest:</th>
<th>Lowest:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pakistan - 35%</td>
<td>France – 0.3%</td>
</tr>
<tr>
<td>Hong Kong - 18%</td>
<td>UK – 0.3%</td>
</tr>
<tr>
<td>Canada - 12%</td>
<td>Germany – 0.9%</td>
</tr>
<tr>
<td>Vietnam – 12%</td>
<td>Norway – 0.9%</td>
</tr>
<tr>
<td>Romania – 10%</td>
<td>Australia – 0.9%</td>
</tr>
<tr>
<td>Indonesia – 10%</td>
<td>Japan - 1%</td>
</tr>
<tr>
<td>Taiwan – 10%</td>
<td>Greece – 1%</td>
</tr>
<tr>
<td>Malaysia – 7%</td>
<td>Italy – 1%</td>
</tr>
<tr>
<td>New Zealand – 7%</td>
<td>Finland – 1%</td>
</tr>
</tbody>
</table>
### The “Good” IPv6 AS’s

<table>
<thead>
<tr>
<th>AS</th>
<th>V6 connection</th>
<th>AS Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS38083</td>
<td>0.0%</td>
<td>AU CURTIN-UNI-AS-AP Curtin University</td>
</tr>
<tr>
<td>AS24226</td>
<td>0.1%</td>
<td>NZ CATALYST-IT-AS-AP Catalyst IT</td>
</tr>
<tr>
<td>AS1312</td>
<td>0.1%</td>
<td>US VA-TECH-AS - Virginia Polytechnic Institute and State Univ.</td>
</tr>
<tr>
<td>AS12552</td>
<td>0.1%</td>
<td>SE IPO-EU IP-Only Telecommunication Networks AB</td>
</tr>
<tr>
<td>AS31334</td>
<td>0.1%</td>
<td>DE KABELDEUTSCHLAND-AS Kabel Deutschland Vertrieb und Service GmbH</td>
</tr>
<tr>
<td>AS237</td>
<td>0.1%</td>
<td>US MERIT-AS-14 - Merit Network Inc.</td>
</tr>
<tr>
<td>AS55</td>
<td>0.2%</td>
<td>US UPENN-CIS - University of Pennsylvania</td>
</tr>
<tr>
<td>AS17727</td>
<td>0.2%</td>
<td>ID NAPINFO-AS-AP PT. NAP Info Lintas Nusa</td>
</tr>
<tr>
<td>AS21453</td>
<td>0.2%</td>
<td>RU FLEX-AS Flex Ltd</td>
</tr>
<tr>
<td>AS2516</td>
<td>0.2%</td>
<td>JP KDDI KDDI CORPORATION</td>
</tr>
<tr>
<td>AS6661</td>
<td>0.2%</td>
<td>LU EPT-LU Entreprise des P. et T. Luxembourg</td>
</tr>
<tr>
<td>AS2107</td>
<td>0.2%</td>
<td>SI ARNES-NET ARNES</td>
</tr>
<tr>
<td>AS12322</td>
<td>0.2%</td>
<td>FR PROXAD Free SAS</td>
</tr>
<tr>
<td>AS3676</td>
<td>0.2%</td>
<td>US UIOWA-AS - University of Iowa</td>
</tr>
<tr>
<td>AS4802</td>
<td>0.3%</td>
<td>AU ASN-IINET iiNet Limited</td>
</tr>
<tr>
<td>AS39326</td>
<td>0.3%</td>
<td>GB GOSCOMB-AS Goscomb Technologies Limited</td>
</tr>
<tr>
<td>AS53347</td>
<td>0.3%</td>
<td>US PREMIER-COMMUNICATIONS - Premier Communications</td>
</tr>
<tr>
<td>AS3333</td>
<td>0.3%</td>
<td>NL RIPE-NCC-AS Reseaux IP Europeens Network Coordination Centre (RIPE NCC)</td>
</tr>
<tr>
<td>AS22394</td>
<td>0.3%</td>
<td>US CELLCO - Cellco Partnership DBA Verizon Wireless</td>
</tr>
<tr>
<td>AS19782</td>
<td>0.3%</td>
<td>US INDIANAGIAGON - Indiana University</td>
</tr>
<tr>
<td>AS5661</td>
<td>0.3%</td>
<td>US USF - UNIVERSITY OF SOUTH FLORIDA</td>
</tr>
<tr>
<td>AS4608</td>
<td>0.3%</td>
<td>AU APNIC-AP Asia Pacific Network Information Centre</td>
</tr>
<tr>
<td>AS3582</td>
<td>0.3%</td>
<td>US UONET - University of Oregon</td>
</tr>
<tr>
<td>AS22548</td>
<td>0.3%</td>
<td>BR Comite Gestor da Internet no Brasil</td>
</tr>
<tr>
<td>AS8426</td>
<td>0.3%</td>
<td>ES CLARANET-AS ClaraNET LTD</td>
</tr>
<tr>
<td>AS2852</td>
<td>0.4%</td>
<td>CZ CESNET2 CESNET, z.s.p.o.</td>
</tr>
<tr>
<td>AS57</td>
<td>0.4%</td>
<td>US UMN-REI-UC - University of Minnesota</td>
</tr>
<tr>
<td>AS7018</td>
<td>0.4%</td>
<td>US ATT-INTERNET4 - AT&amp;T Services, Inc.</td>
</tr>
<tr>
<td>AS1103</td>
<td>0.4%</td>
<td>NL SURFNET-NL SURFnet, The Netherlands</td>
</tr>
<tr>
<td>AS55391</td>
<td>0.5%</td>
<td>JP MF-NATIVE6-E INTERNET MULTIFEED CO.</td>
</tr>
</tbody>
</table>
# The “Not So Good” IPv6 AS’s

<table>
<thead>
<tr>
<th>AS</th>
<th>V6 connection</th>
<th>AS Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS29113</td>
<td>12.5%</td>
<td>CZ SLOANE-AS UPC Ceska Republica, s.r.o.</td>
</tr>
<tr>
<td>AS1659</td>
<td>12.6%</td>
<td>TW ERX-TANET-ASN1 Tiawan Academic Network (TANet) Information Center</td>
</tr>
<tr>
<td>AS45230</td>
<td>12.6%</td>
<td>NZ UBERGROUP-AS-NZ UberGroup Limited</td>
</tr>
<tr>
<td>AS18119</td>
<td>12.8%</td>
<td>NZ ACSDATA-NZ ACSData</td>
</tr>
<tr>
<td>AS17451</td>
<td>13.6%</td>
<td>ID BIZNET-AS-AP BIZNET ISP</td>
</tr>
<tr>
<td>AS24173</td>
<td>13.8%</td>
<td>VN NETNAM-AS-AP Netnam Company</td>
</tr>
<tr>
<td>AS12271</td>
<td>15.1%</td>
<td>US SCRR-12271 - Road Runner HoldCo LLC</td>
</tr>
<tr>
<td>AS17709</td>
<td>16.8%</td>
<td>TW EBT Eastern Broadband Telecom Co., Ltd</td>
</tr>
<tr>
<td>AS11427</td>
<td>18.4%</td>
<td>US SCRR-11427 - Road Runner HoldCo LLC</td>
</tr>
<tr>
<td>AS2907</td>
<td>18.4%</td>
<td>JP SINET-AS Research Organization of Information and Systems, National Institute of Informatics</td>
</tr>
<tr>
<td>AS8591</td>
<td>19.2%</td>
<td>SI AMIS AMiS</td>
</tr>
<tr>
<td>AS812</td>
<td>19.6%</td>
<td>CA ROGERS-CABLE - Rogers Cable Communications Inc.</td>
</tr>
<tr>
<td>AS12046</td>
<td>19.8%</td>
<td>MT ASN-CSC-UOM University of Malta</td>
</tr>
<tr>
<td>AS3356</td>
<td>20.2%</td>
<td>US LEVEL3 Level 3 Communications</td>
</tr>
<tr>
<td>AS4725</td>
<td>20.9%</td>
<td>JP ODN SOFTBANK TELECOM Corp.</td>
</tr>
<tr>
<td>AS8970</td>
<td>21.6%</td>
<td>PL WASK WROCMAN-EDU educational part of WASK network, Wroclaw, Poland</td>
</tr>
<tr>
<td>AS17579</td>
<td>21.6%</td>
<td>KR KREONET2-AS-KR Korea Institute of Science and Technology Information</td>
</tr>
<tr>
<td>AS7539</td>
<td>22.3%</td>
<td>TW TANET2-TW TANet2, sponsored by NSC, TAIWAN</td>
</tr>
<tr>
<td>AS3262</td>
<td>22.9%</td>
<td>ES SARENET SAREnet, Spain</td>
</tr>
<tr>
<td>AS11537</td>
<td>25.3%</td>
<td>US ABILENE - Internet2</td>
</tr>
<tr>
<td>AS16880</td>
<td>32.5%</td>
<td>US TRENDMICRO Global IDC and Backbone of Trend Micro Inc.</td>
</tr>
<tr>
<td>AS9431</td>
<td>33.2%</td>
<td>NZ AKUNI-NZ The University of Auckland</td>
</tr>
<tr>
<td>AS4528</td>
<td>33.6%</td>
<td>HK HKU-AS-HK The University of Hong Kong</td>
</tr>
<tr>
<td>AS45809</td>
<td>34.7%</td>
<td>NZ NZRS-AS-AP ASN for .nz registry content</td>
</tr>
<tr>
<td>AS2576</td>
<td>42.0%</td>
<td>US DOT-AS - U. S. Department of Transportation</td>
</tr>
<tr>
<td>AS17996</td>
<td>42.3%</td>
<td>ID UIINET-ID-AP PT Global Prima Utama</td>
</tr>
<tr>
<td>AS3562</td>
<td>45.5%</td>
<td>PK SNLL-NET-AS - Sandia National Laboratories</td>
</tr>
<tr>
<td>AS24514</td>
<td>58.6%</td>
<td>MY MYREN-MY Malaysian Research &amp; Education Network</td>
</tr>
</tbody>
</table>
Comparing RTTs

- 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 vs. Relative RTT, IPv6 to IPv4 (sec)
Relative RTT, IPv6 to IPv4 (sec) for bilby on 2012/03/01

Number of samples (log scale)

RTT Difference (in fractions of a second)

IPv6 is faster

IPv6 is slower

Teredo

6to4

global unicast

Unicast

6 to 4
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!
This is unexpected!

Europe-located Server
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?
  – Because IPv4 is slower?
    • Is this related to the behaviour characteristics of some CPE based NATs and their handling of NAT bindings during a SYN exchange?
• 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
Use of local outbound 6to4 relay has reduced this skew.
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 ~2% 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 footing in a parallel connection environment, using 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 Homeout:

- 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
Everything is connected...

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

Browser and operating system behaviours have a direct impact on the scaling pressures for CGN deployment

So how can we help this story along?
How Should Browsers Behave?

– Fire off the A and AAAA DNS queries in parallel
– When the DNS returns an AAAA response fire off a V6 connection attempt immediately
– When the DNS returns a A response wait for a small amount of time, and if the V6 connection has not completed, then fire off a V4 connection attempt
  • Use a *reasonably aggressive* wait timer on the DNS to TCP gap
    E.g. Chrome with 300ms failover timer
    E.g. Safari + Mac OS X with RTT-derived timer
Parallel Connection Model

Start Point

DNS

AAAA?

Send SYN

TCP

A?

pause

Send SYN

Wait For First SYN/ACK
How Should Browsers Behave?

- Fire off the A and AAAA DNS queries in parallel
- When the DNS returns an AAAA response, connection attempt immediately
- When the DNS returns an A response, wait for a small amount of time, and if the V6 connection has not completed, then fire off a V4 connection attempt
  
  • Use a *reasonably aggressive* wait timer on the DNS to TCP gap
    
    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?