Measuring Access Connectivity Characteristics with *Netalyzr*

Christian Kreibich (ICSI),
Nicholas Weaver (ICSI),
Boris Nechaev (HIIT/TKK),
and Vern Paxson (ICSI & UC Berkeley)

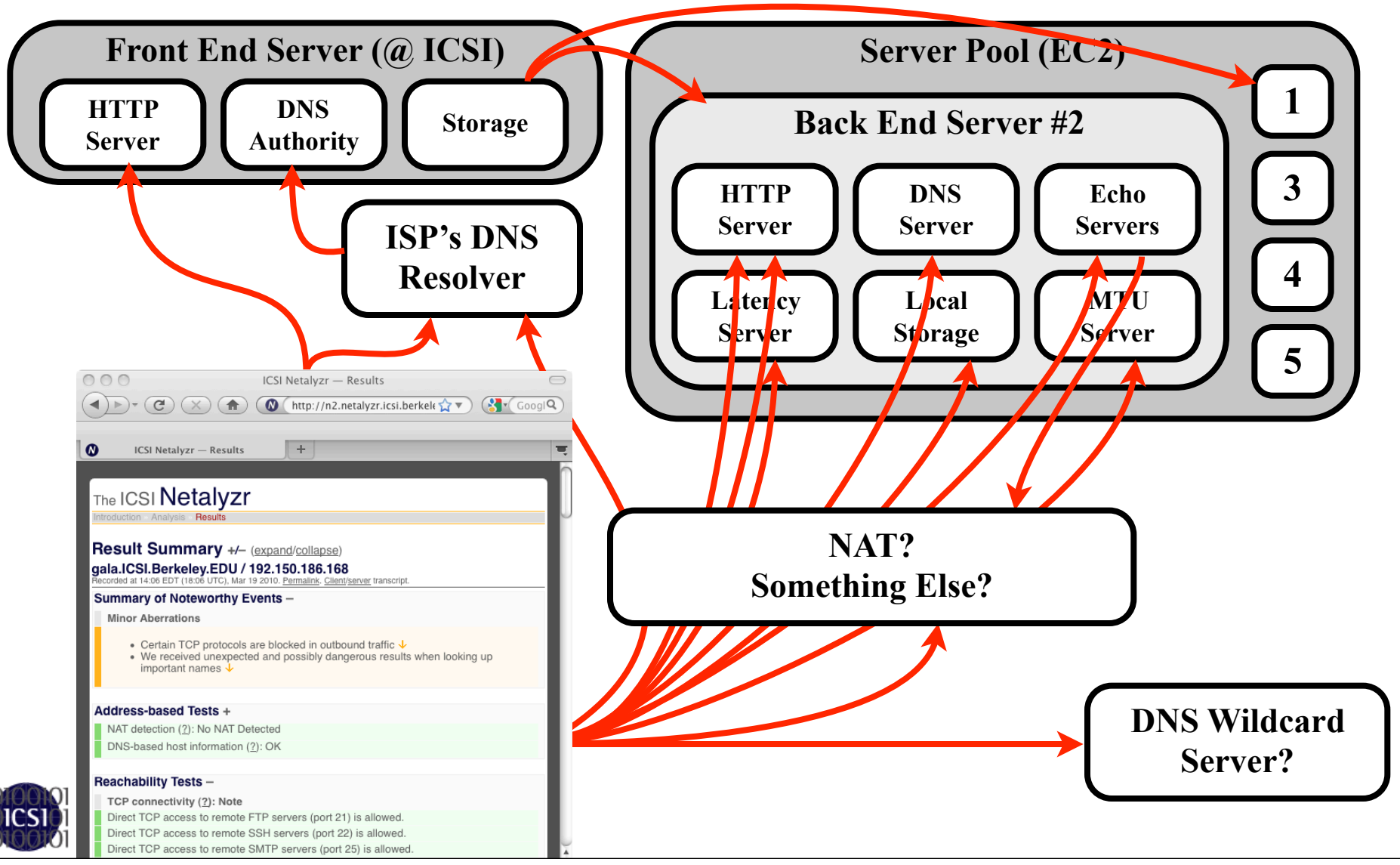
Network Transparency And Network Debugging

- How do you know what the network actually is?
  - Network ***Transparency***: What does the network really do to the data?
- What is not working?
  - Network ***Debugging***: Is there something wrong that needs to be fixed
- We desired a comprehensive tool for multiple roles
  - An easy to use network survey for everyone
    - Over 110,000 executions to date
  - A detailed diagnostic and debugging tool for experts
- Thus we built ***Netalyzr***, a network debugging and diagnostic tool which runs in the web browser
  - Just two mouseclicks

Key Insights Behind *Netalyzr*

- Java applets can perform a lot of activity by default:
  - Can use arbitrary TCP and (usually) UDP connections to the server hosting the applet
  - Can lookup arbitrary DNS A (address) records, but the result can ***only*** resolve to the hosting server's IP or generate a security exception
- Java applets can do even more when "trusted" (the signature is accepted by the user)
  - Bypasses same origin for both DNS and connectivity
- Javascript can do other things
  - Load third party images and validate success
  - Examine DOM to see if its in an iframe
- And our servers can do whatever it wants
  - Any services, including deliberate protocol violations
  - Raw packet examination

Netalyzr's Architecture



Netalyzr's Test Suite

- Network Address Translation:
  - Is there a NAT?
  - Is the NAT a DNS proxy?
    - Is it an open DNS proxy?
  - How are ports renumbered?
- Network Link Properties:
  - Network latency and bandwidth
  - Network buffering
  - Path MTU discovery and potential path MTU problems
- Port Filtering:
  - What major TCP and UDP ports have ***outbound*** port filters?
  - What major TCP and UDP ports have ***protocol aware*** behavior?
    - A network device which enforces protocol semantics

Netalyzr's Test Suite Continued

- HTTP tests
  - Is there an HTTP proxy or cache?
    - If so, does it operate correctly?
  - Are various filetypes modified or blocked in the network?
  - Is the test run from within an unauthorized iFrame?
- DNS tests
  - DNS server identification
  - Support for EDNS, glue policy, IPv6
  - DNS port randomization
  - DNS transport issues
  - Lookups of popular names
  - DNS wildcarding of invalid names
- Misc items
  - IPv6 support
  - Clock drift

Significant Usage

- Released in public beta during the summer of 2009
- Non-beta (and enhancements) January 2010
- Over 110,000 unique sessions to date
  - Results are through June 2010
- Some significant biases
  - Comcast is significantly overrepresented with 11%
    - Due to initial slashdotting's article context
  - OpenDNS is significantly overrepresented with 12%
    - Suggests overall a significant "geek" bias

NAT Detection

- NAT detection is relatively straightforward
  - A TCP connection in Java can (usually) obtain the link local IP address and local TCP port number
  - The remote server returns the IP address and port number used to contact the host
- Uses this to discover NAT properties
  - Presence, port # rewriting
  - As expected, 90% of sessions are behind a NAT
- Also probes the NAT for DNS proxies
  - 67% of NATs showed a DNS proxy, which matches expectation
    - Can't tell whether this is the DNS settings returned to the client
  - 4.4% of sessions accept and fully process an external DNS request
    - We heard reports of this being rather common: a source of reflectors and DNS probes

Detecting Protocol-Aware Network Devices

- The port filtering tests (except for HTTP and UDP DNS) connect to our custom echo server with simply returns the IP and SRC port
- Observed behavior can deduce network policy
  - If the response is received as expected:
    No filtering on this port
  - If the IP address has changed:
    This port or system routes through a proxy or multiple IP addresses, or changed its IP address during the test
  - If the connection fails:
    This port is blocked somewhere in the network
  - If the connection succeeds but different data is returned:
    This port passes through a declared proxy
  - If the connection succeeds but no data is returned:
    The request or response was blocked by a network device that is probably protocol aware

Port Filtering Results

- A few surprises:
  - Local POP proxies surprisingly common (often on the host itself)
    - 7% reject our protocol violation, and another 6% of sessions captured a proxy's banner
  - Many NATS include FTP proxies
    - 20% show FTP interference
  - SIP-aware network devices surprisingly common as well
    - 5% reject our protocol violation
  - Less outbound SMTP filtering than we expected
    - 25% blocked, 8% reject the protocol violation
      - Suggests that many ISPs are using dynamic blocking of spam-bots
- Expected Results:
  - Port 443 is almost completely unmolested (2% showed blocking, .3% rejected the protocol violation)
  - Windows Port blocking very common
  - Slammer blocks still common

Measuring DNS filtering

- Applet sends several probe requests over UDP port 53 to the back-end server for both legitimate and illegitimate requests
- Filtering surprisingly common:
  - 11% of sessions reject “non DNS” over DNS
    - Open Question: How will such devices react to DNS extensions and unknown RRTYPEs?
      How flexible is DNS?
  - DNS proxies are rare however, only 1.2%
- Direct checks using EDNS (Extended DNS) records of various sizes
  - 1.3% fail the small test (network can’t handle EDNS)
  - 4.5% fail the medium test (additional cause: network assumes DNS <= 512B)
  - 14% fail the large test (additional cause: fragmentation issues)
  - Significant problem for DNSSEC validation on the client

DNS Server Tests

- DNS glue policy
  - How do DNS resolver react to additional records in replies
    - Special names in our DNS server return different values when fetched directly
  - Do DNS servers request DNSSEC records
  - Can fingerprint DNS resolvers
    - 32% of sessions show BIND's default policy
- Actual DNS MTU
  - Many (~10%) of DNS resolvers which ***advertise*** the ability to receive large responses can't actually receive fragmented traffic!
    - Will be a potential problem with DNSSEC, as DNSSEC-enabled replies may exceed 1500B

DNS Wildcarding

- Disturbing relatively recent trend:
  - Instead of NXDOMAIN errors, return a “helpful” address of a web server instead
- Three ways to do it:
  - Bad: Wildcard anything that is www.*.com and related
    - Comcast, Verizon
  - Even worse: Wildcard everything
    - Charter, Qwest
  - Even worser: Also wildcard SERVFAIL
    - OpenDNS
- 28% show wildcarding
  - Excluding Comcast and OpenDNS: 21% show wildcarding
- You can’t trust NXDOMAINs to be NXDOMAINs anymore!

DNS Man-in-the-Middle

- The applet looks up a large number (~70) names on the client, returning the results to the server
  - The server then performs reverse lookups to validate
- Three major strains of maliciousness discovered, beyond using DNS for blocking and NXDOMAIN wildcarding
  - Annoying: OpenDNS
    - OpenDNS acts as a Man-in-the-Middle for **`www.google.com`**, redirecting all traffic through a proxy they control
      - It is disclosed, but they don't talk about it much
  - Really questionable: Wide Open West and a few other ISPs
    - Acts as a Man-in-the-Middle for **`www.google.com`**, redirecting traffic through a custom proxy
      - Proxy when given bad input refers to phishing-warning-site.com, a parked domain
  - Downright criminal: Malicious DNS resolvers
    - Malcode sets users to point to a malicious resolver
    - Redirects **`windowsupdate.microsoft.com`** to a google IP address
    - ***May*** redirect **`ad.doubleclick.net`** to serve adds for products such as "ViMax Male Enchancement"
- All these problems are due to the recursive resolver itself:
  - DNSSEC validation ***must*** be on the end client...
    But as we saw earlier, 14% of the clients measured would have problems with this!

Fragments and Path MTU Discovery

- Fragments are a ***big*** problem
  - Tested by sending or receiving a large UDP datagram
  - 8% of sessions can't send UDP fragments
  - 8% of sessions can't receive UDP fragments
  - Those who ***can*** send fragments may have an MTU hole:
    3% of sessions which can send 2000B fragments can ***not*** send a 1500B packet!
- The network is ***mostly*** but not all Ethernet (83% use the Ethernet MTU)
  - A significant amount still uses PPPoE (MTU 1492, 13%)
- ICMP reporting unreliable:
  - Only 61% of sessions where an ICMP "too big" should have been generated ***actually generated one***
- Conventional Wisdom is correct:
  - The Network has decreed that fragmentation doesn't work
  - Path MTU discovery must use fallbacks when ICMP isn't received
    - Linux bug: uses "Path MTU discovery" on UDP traffic, by setting the DF bit on UDP packets
      Creates a UDP Path MTU hole, as even when the ICMP is generated, it causes Java to raise an exception

What are your questions?

- What do you wish to know about the end-user connections that you don't already?
  - Netalyzr is not a static project, but undergoes continuous enhancements
- In particular, what are your IPv6 concerns?
  - Both for systems and for web browsers?

Conclusions...

- It ***worked!***
  - We discovered a lot about how the edge of the Internet really behaves
  - A small group can build a ***robust*** and ***comprehensive*** network measurement and diagnostic tool
    - You know you built your architecture right when your sysadmin asks you “so when is it going up on Slashdot” and your answer is “its been up for an hour”

Information About Other Tests

- Slides about other tests of potential interest

HTTP Proxies

- Does the web browser use a proxy?
  - The Java high level API routes requests through the web browser itself
  - Our web server adds an HTTP header which indicates client source address
- The applet then constructs requests using a direct TCP port 80 connection
  - To a web page which HTTP encodes the headers used in the request
  - Any change in the headers indicates a mandatory HTTP proxy
    - Including CaPitAlization changes: HTTP headers are case insensitive but many devices will transcode HTTP header capitalization
  - Also generates deliberately invalid requests
- 8.6% show some evidence of proxying
  - 90% of sessions with proxying are ***mandatory*** HTTP proxies
  - HTTP proxies are common enough

More HTTP tests

- Generates a deliberately invalid request
  - In-path proxies may reject as invalid, 4% of all sessions
- Fetches four test files using a direct HTTP connection
  - 1% failed to get the .exe, .7% failed to get the .mp3
  - 1.2% failed to get the .torrent
  - 10% failed to get the EICAR “test virus”
- Sends a request to our server, but with the host set to **`www.google.com`**
  - Checks for a vulnerability in in-path HTTP proxies which will instead redirect the request violating same-origin protections, very rare (except in New Zealand!)
- Fetches a common test image with different caching headers
  - Image alternates between two versions with the same size but different color maps
  - 5% show caching, but no major US ISP does caching
    - Caches happen, and when they do, they are often broken!
  - Transcoding ***very*** rare

JavaScript Tests

- Is the test run from within an iFrame?
  - Fetches a 0x0 “image” with the Javascript accessible cookies and whether this is the top-level iFrame encoded in the “fetched” image’s URL
    - Overall, very rare: mostly airports or similar hotspots
- Does the host have IPv6?
  - We don’t (currently) have any IPv6 servers for Netalyzr
  - In a hidden , the analysis page loads the logo from ipv6.google.com
    - Bind Javascript to success and failure
  - Success and failure report results back with another 0x0 image “fetch”: 4.5% of sessions were able to get the image
    - Could be cached from previous IPv6 access
    - IPv6 is slow to adopt, even amongst the technically savvy

Network Buffering

- These measurements are done using UDP, not TCP
  - Eliminates TCP performance artifacts
  - Wish to stress the network
- For 10 seconds
  - Send large UDP packets to our server
    - The server's responds to each packet with a small reply
  - Ramp up the sending rate with exponential doubling
    - For each packet received, send two more
  - Measure the bandwidth and additional latency for each packet during the last 5 seconds of this process
    - Detects both the bandwidth and estimates the capacity of the bottleneck packet buffer
  - Wait an additional 5 seconds for buffers to drain
- Then repeat for the downlink direction

Uplinks suffer from *chronic* overbuffering