May 2013
George Michaelson
Geoff Huston

APNIC Labs IPv6 Measurement System

For some years now at APNIC Labs we've been conducting a measurement exercise intended to measure the extent to which IPv6 is being deployed in the Internet. This is not a measurement of IPv6 traffic volumes, nor of IPv6 routes, nor of IPv6-capable servers. This is a measurement of the Ipv6 capabilities of devices connected to the Internet, and is intended to answer the question: what proportion of devices on the Internet are capable of supporting an IPv6 connection?

We've often been asked about our measurement methodology, and this article is intended to describe in some detail how we perform this measurement.

General Approach

Using Flash and JavaScript, clients' web browsers are inducted into a measurement of their capabilities to use IPv6, based on the scripted fetch of a set of 'invisible' 1x1 pixel images. Each test is intended to isolate a particular capability of the client, in so far as if the client successfully fetches the object associated with the test then the client is considered to be capable in that aspect.

We have a set of five basic properties that are available in the test: IPv4-only, IPv6-only, Dual Stack, Dual Stack with unresponsive IPv6 and Dual Stack with unresponsive IPv4.

We are interested in the behaviour of the DNS transport as well as in the behaviour of the HTTP transport.

The URL of each of the images is constructed using labels that describe the object's transport properties in DNS resolution and the HTTP transport for the web fetch. For example, a URL of http://xxx.r6.td.labs.apnic.net/1x1.png describes a web object that is only accessible using IPv6, ('r6') and the domain name itself is served by authoritative name servers that can respond to DNS resolver queries made over both IPv4 and IPv6 ('td'), while a URL of http://xxx.rd.t6.labs.apnic.net/1x1.png describes a dual stack web object ('rd') whose domain name is only served by authoritative name servers that are reachable only on IPv6 ('t6'). The complete name structure of the various tests is provided in the following table:

Behaviour	Value
DNS	t
HTTP	r
IPv4-only	4
IPv6-only	6
Dual Stack	d
Dual Stack, unresponsive IPv6	x
Dual Stack, unresponsive IPv4	z

In order to ensure that each client is forced to perform both the DNS lookups and the web object fetches from the experiment's servers, and not use locally cached values, we make use of dynamic name generation and wildcard DNS capabilities to generate a unique string as part of the object's name. This unique string is used in the DNS part of the URL, and is also used as an argument to the resource name part of the URL. Each client is served with a unique name value, and all the tests presented to the client share the same name value so that at the server end we can match the operations that were performed in the context of each test instance. As these domain name components map to a wildcard in the DNS zone, it does not increase the complexity or time taken to perform DNS resolution. The components of this unique string value include the time of day (seconds since 1 January 1970 00:00 UTC), a random number, and experiment version information.

The time taken by the client to fetch each URL is recorded by the client-side script.

The set of URLs concludes with a "result" URL. This URL is triggered either when all the other URLs have been loaded, or when a local timer expires. The fetch of this "result" URL includes, as arguments to the GET command, the results (and individual timer values) of the fetch operations of all the other URLs. The default value of this timer for result generation is 10 seconds.

As well as getting the client to perform self-timing of the experiment, we also direct all traffic associated with the experiment (the authoritative DNS name servers that will receive the DNS queries and the web servers that will receive the HTTP fetches) to a server that is logging all traffic. We perform logging at the DNS, HTTP and packet level. These logs provide server-side information on the nature of that clients capabilities such as the client-resolver relationship, apparent RTT in DNS and web fetch, IPv4 and IPv6 capability, MTU, and TCP connection failure rates.

Client-side Code

We use two forms of encoding of this method: Flash and JavaScript. They are used in different experiment contexts.

Flash permits embedding of the measurement in advertising channels using flash media for image ads. This channel delivers large volumes of unique clients who can be targeted by keyword, or economy, or exclude specific IP ranges.

There are a number of weaknesses of Flash, most notably being that Flash code is not loaded on some popular mobile platforms, including Apple's mobile platforms. It's also been observed that the Flash engine does not appear to perform consistent client-side timer measurements, probably due to a more complex internal object scheduler within the Flash engine. We have also observed that the Flash engine does not preserve fetch order, so that the order of objects to fetch generated by the Flash action script is not necessarily the order in which the Flash engine will perform the fetches. The most common permutation is that the Flash engine reverses the object fetch order as it retrieves the set of objects.

JavaScript permits embedding of the measurement in specific host websites. There are two variants of this script. One is where the JavaScript is directly inserted into the host web page, and the other form is as a user-defined code extension to Google's Analytics code. In the latter case the web administrator can use the Analytics reports to view the IPv6 capabilities of the site's visitors in addition to the other Analytics reports. The website does not itself have to be IPv6 enabled: the tests cause the client to interact with our experiment servers and the IPv6 capability is measured between he client and these servers. In this case there is no control over who performs the test: the test is performed by all end clients who visit the site where the JavaScript is embedded.

JavaScript appears to be more widely supported than Flash. However, because JavaScript uses code embedded in web sites, the number and diversity of clients being testing in this manner depends on the visitor profile of the hosting web. Many web sites have a large volume of repeat clients, so the tested client population of the JavaScript test appears to record a particular profile of capability (for example, we have observed an anomalously high proportion of IPv6 capability in the clients who use APNIC's `whois` web service). The JavaScript code can also be configured via cookies not to re-sample a particular client within a certain period (the cookie has a default retry value of 24 hours), in order to counter, to some extent, measurement bias generated by repeat visitors to the site.

The original versions of the test code explicitly enumerated the individual URL tests to be executed. There is a more recent variant of both the Flash and JavaScript codes that includes a runtime configuration server. In this variant of the test code, the client will initially perform a fetch from a configuration server. This server will return the set of URLs to be used for the test. This allows the parameters of the test to be varied in the fly without having to reload the JavaScript that was embedded in the web page, or without re-submitting the ad with the embedded Flash script.

Server Configuration

We use three servers for this experiment, One is located in Australia, one in Germany and one in the United States. One server is a Linux-based host, while the other two use FreeBSD as their host OS.

The servers use Apache for the web server, Bind for the DNS server, and tcpdump for the packet capture. They are also configured with a local Teredo server, and a local 6to4 relay.

Where possible, we use 16 different addresses in both IPv4 and IPv6.

When running these tests on a highly visited web page, or using a high volume ad campaign, we have noted that there can be relatively large peak demands for web fetches on our web servers. The experiment's webservers need sufficient capacity to handle hundreds of queries per second, which means using a system configuration that has thousands of pre-forked http daemons and kernel configuration support for thousands of open/active TCP sessions. This also requires servers with large memory configuration. Sufficient disk is required to ensure tcpdump and server logs can be held for a continuous cycle of experiments.

Post processing is currently performed in a central log archive, to integrate all sources of experiment data into a collated experiment log, which is then post processed on a daily basis.

DNS configuration

The DNS part of the experiment configuration depends on the 'wildcard' DNS record. All zones which serve terminal fully qualified domain names have a wildcard record which maps any name under that domain to the IPv4 or IPv6 address for the head server.

For the current experiments in both DNS and IPv6 capability, 4 distinct subdomains of DNS are registered under a single prefix:

`f.labs.apnic.net`	Experiments in Asia/Oceania
`g.labs.apnic.net`	Experiments in the Americas
`h.labs.apnic.net`	Experiments in Europe/Middle-East/Africa
`i.labs.apnic.net`	Testing, future expansion

The master server generates an experiment set for the client based on a basic geo-location mapping of the client's address to a geographic region. This is done as a rudimentary load balancing exercise, and, more importantly, to minimize the round trip time between the server and the client, and thereby avoid, to some extent, retransmits and timeouts at the client side while performing the experiment. The mapping of address to region is intentionally quite coarse, and some traffic inevitably goes to a distant head server, but this does not appear to have had a significant impact on the measurement outcomes.

The parent domains are provisioned to have identical sub-domains, which characterize DNS transport by the listed NS delegations. A domain which is only delegated to IPv6 DNS servers cannot be successfully resolved by a DNS resolver which does not have access to IPv6 transport. Consequently the client should not be told the experiment's IP address. A DNS resolver which is dual stacked may be fetched over either IPv4 DNS transport, or IPv6 DNS transport.

```
$TTL 1d
$ORIGIN        .

f.labs.apnic.net        IN      SOA     dns4f.labs.apnic.net. ggm.apnic.net.  (
                                2013051101      ; Serial
                                3600            ; Refresh
                                900             ; Retry
                                3600000         ; Expire
                                3600 )          ; Minimum
        IN      NS      dns4f.labs.apnic.net.
        IN      A       203.133.248.22
        IN      AAAA    2401:2000:6660::22

; sub delegations which feed off f.labs, served  by NS *in* f.labs
$ORIGIN f.labs.apnic.net.

dns0                    A       203.133.248.22
dns                     A       203.133.248.22
                        AAAA    2401:2000:6660::22
;
; dual stack
dnsd                    A       203.133.248.23
                        AAAA    2401:2000:6660::23
dnsrd                   A       203.133.248.24
                        AAAA    2401:2000:6660::24
;
; 4 only
dns4                    A       203.133.248.25
dnsr4                   A       203.133.248.26
;
; 6 only
dns6                    AAAA    2401:2000:6660::27
dnsr6                   AAAA    2401:2000:6660::28
;
; reachable on 4, not on 6
dnsx                    A       203.133.248.29
                        AAAA    6000:666::29
dnsrx                   A       203.133.248.30
                        AAAA    6000:666::30
;
; reachable on 6, not on 4
dnsz                    A       7.0.0.31
                        AAAA    2401:2000:6660::31
dnsrz                   A       7.0.0.32
                        AAAA    2401:2000:6660::32
;

dnssec                  NS      dns0
dualstack               NS      dnsrd
ipv4bad                 NS      dnsrd
ipv4only                NS      dnsrd
ipv6bad                 NS      dnsrd
ipv6badbadbad           NS      dnsrd
ipv6only                NS      dnsrd
t4                      NS      dns4
t6                      NS      dns6
td                      NS      dnsd
tx                      NS      dnsx
tz                      NS      dnsz
v6trans                 NS      dns6

www                     A       203.133.248.18
                        AAAA    2401:2000:6660::18

*.results               IN      A 203.133.248.18
results                 IN      A 203.133.248.18
```

As shown in the DNS zone file above, f.labs.apnic.net has 5 subdomains:

> t4.f.labs.apnic.net DNS NS is on IPv4 only
> t6.f.labs.apnic.net DNS NS is on IPv6 only
> td.f.labs.apnic.net DNS NS is dual-stacked
> tx.f.labs.apnic.net DNS NS is dual-stacked, but IPv6 is unreachable
> tz.f.labs.apnic.net DNS NS is dual-stacked, but IPv4 is unreachable

Separately, results.g.labs.apnic.net and *.results.g.labs.apnic.net (a wildcard) are defined as an IPv4 A record.

Within each subdomain(t4, t6,td, tx and tz) a further family of subdomains are defined. For example, the following is the zone file for t4.f.labs.apnic.net:

```
$TTL 1d
$ORIGIN         .

t4.f.labs.apnic.net     IN      SOA     t4.f.labs.apnic.net. ggm.apnic.net. (
        2013051101      ; Serial
        3h              ; Refresh
        1h              ; Retry
        1w              ; Expire
        3h )            ; Neg. cache TTL

        A       203.133.248.25
        AAAA    2401:2000:6660::25

;
;       name servers
;
        NS      dns4.f.labs.apnic.net.
;
;       zone contents
;
$ORIGIN t4.f.labs.apnic.net.
;
v4      A       203.133.248.25
v6      AAAA    2401:2000:6660::25
;
rd      NS      dnsr4.f.labs.apnic.net.
r4      NS      dnsr4.f.labs.apnic.net.
r6      NS      dnsr4.f.labs.apnic.net.
rx      NS      dnsr4.f.labs.apnic.net.
rz      NS      dnsr4.f.labs.apnic.net.
```

This zone has 5 sub-delegations (rd, r4, r6, rx and rz) each of which is defined to have a single name server.

`r4.t4.g.labs.apnic.net`	IPv4 NS, resources are reachable on IPv4 only
`r6.t4.g.labs.apnic.net`	IPv4 NS, resources are reachable on IPv6 only
`rd.t4.g.labs.apnic.net`	IPv4 NS, resources are reachable on dual-stack
`rx.t4.g.labs.apnic.net`	IPv4 NS, resources define IPv4/IPv6 but IPv6 is unreachable
`rz.t4.g.labs.apnic.net`	IPv4 NS, resources define IPv4/IPv6 but IPv4 is unreachable

For example, the r4.t4.f.labs.apnic.net subdomain uses the following zone file:

```
$TTL 1d
$ORIGIN         .

r4.t4.f.labs.apnic.net  IN      SOA     r4.t4.f.labs.apnic.net. ggm.apnic.net. (
        2013051101      ; Serial
        3h              ; Refresh
        1h              ; Retry
        1w              ; Expire
        3h )            ; Neg. cache TTL

        A       203.133.248.26
;
;       name servers
;
        NS      dnsr4.f.labs.apnic.net.
;
;       zone contents
;
$ORIGIN r4.t4.f.labs.apnic.net.
v4 5  IN  A       203.133.248.26
;
; wildcard
;
*  5  IN  A       203.133.248.26
```

Therefore from this delegation chain, an experiment configuration server can request a client to fetch an experiment such as:

`http://t10000.u8738132781.s1367808039.i333.v6024.r4.t4.f.labs.apnic.net/1x1.png`

The t10000.u8738132781.s1367808039.i333.v6024 part is all matched by the wildcard, under the r4.t4.f.labs.apnic.net domain.

```
$ dig +short a t10000.u8738132781.s1367808039.i333.v6024.r4.t4.f.labs.apnic.net.
203.133.248.26
$ dig +short aaaa t10000.u8738132781.s1367808039.i333.v6024.r4.t4.f.labs.apnic.net.
(no answer)
$ dig +short r4.t4.f.labs.apnic.net. IN NS
dnsr4.f.labs.apnic.net
$ dig +short dns4.f.labs.apnic.net. IN A
203.133.248.25
$ dig +short dns4.f.labs.apnic.net. IN AAAA
(no answer)
```

With 5 t* subdomains and 5 r* subdomains a total of 25 domains have to be populated, each slightly different, respecting the NS and A/AAAA combinations which have to apply to that experiment.

We operate the experiment's servers with 11 discrete BIND processes, each listening to a different IPv4 and IPv6 address. One server is used for the parent domains. One sever is used for `t4`, one for `t6`, one for `td`, one for `tx` and one for `tz`. One server is used for all subdomains that include the `r4` forms (`r4.t4, r4.t6, r4.td, r4.tx, r4.tz`). One is used for all r6 forms, one for rd, one for rx and one for rz. This separation of parent and child in the DNS servers ensures the integrity of the IP behaviours in the DNS, as within this structure of authoritative server separation, the authoritative name server for the parent is unable to answer questions that can be resolved by the authoritative name server for the child.

Web server and client code

The Apache webserver needs to be configured to accept all local IP bindings and use 'virtual server' configuration to service them. In the simple configuration model we use the ability of the Apache httpd 2.2 to define a *default* virtual server, which captures all otherwise un-defined instances. This framework is suitable to be the handler for all incoming 1x1.png requests.

Apache configuration

```
<VirtualHost *>
    DocumentRoot /usr/local/www/data/labs/docs
    ServerName g.labs.apnic.net
    ServerAlias g.labs.apnic.net
    ServerAlias *.g.labs.apnic.net
    ScriptAlias /cgi-bin/ "/usr/local/www/data/labs/cgi/"
    <IfModule mod_headers.c>
        <FilesMatch "\.(js|css|xml|gz)$">
            Header append Vary Accept-Encoding
        </FilesMatch>
    </IfModule>
    Header set Access-Control-Allow-Origin "*"
    <Directory "/usr/local/www/data/labs/docs">
      AllowOverride all
      Options Indexes FollowSymLinks Includes ExecCGI
      XBitHack on
      AddHandler cgi-script ipv6-test
      AddHandler cgi-script .cgi
      AddHandler cgi-script .py
      AddHandler cgi-script serveflashconfig
    </Directory>
</VirtualHost>
```

We use the 'prefork' model of Apache configuration, with a high peak load httpd, and an initial small start set, which quickly grows.

```
<IfModule mpm_prefork_module>
    StartServers          5
    MinSpareServers       5
    MaxSpareServers      10
    ServerLimit        3000
    MaxClients         2000
    MaxRequestsPerChild   0
    ListenBackLog      2000
</IfModule>
```

.HTACCESS file configuration

For .htaccess, we define a few extra behaviours to force the expiry on served images to be in the past, and ensure we can serve compressed data (this helps reduce the load time of the .js significantly)

```
<IfModule headers_module>
  #enable compression on only text for now
  AddOutputFilterByType DEFLATE text/html text/plain text/xml text/javascript
application/javascript
  # enable expirations
  ExpiresActive On
  ExpiresDefault "access plus 23 hours"
  header set Cache-Control "no-cache"
  header set Expires "Mon, 26 Jul 1997 05:00:00 GMT"
  <FilesMatch "\.(js|css|xml|gz)$">
    Header append Vary Accept-Encoding
  </FilesMatch>
</IfModule>
```

Log configuration

```
LogFormat "%v %h %t \"%r\" %>s %b \"%{Referer}i\"
         \"%{User-Agent}i\" %D %M %{%s}t %{SERVER_NAME}e %{Host}i" combined
```

To ensure that the logfile includes the specific virtual server called by the client, the ${Host}i field captures the Host: HTTP value from the client initial query.

The 1x1.png is served with a ?= list of arguments which also record the specific runtime fetch, in another field. The combination of this ensures that within the logfile we can correlate the specific experiment, and returned results with DNS and tcpdump logs.

Runtime Configuration

The runtime configuration is achieved by a CGI handler, which is called by the flash logic, or embedded in the <script>….</script> for the JavaScript instance.

The URL: http://drongo.rand.apnic.net/measureipv6idseq.cgi?advertID=6024 shows as its output:

```
rd.td   http://t10000.u7643426681.s1369874228.i333.v6024.rd.td.f.labs.apnic.net/1x1.png?t10000.
        u7643426681.s1369874228.i333.v6024.rd.td

r4.td   http://t10000.u7643426681.s1369874228.i333.v6024.r4.td.f.labs.apnic.net/1x1.png?t10000.
        u7643426681.s1369874228.i333.v6024.r4.td

r6.td   http://t10000.u7643426681.s1369874228.i333.v6024.r6.td.f.labs.apnic.net/1x1.png?t10000.
        u7643426681.s1369874228.i333.v6024.r6.td

v6lit   http://[2401:2000:6660::f103]/1x1.png?t10000.u7643426681.s1369874228.i333.v6024.v6lit

results http://results.f.labs.apnic.net/1x1.png?t10000.u7643426681.s1369874228.i333.v6024&r=
```

This represents a URL to use for each of four tests, and URL to use to pass the results of the four tests back to the server.

The head-end has used the Apache REMOTE_ADDR environment variable to select which head-end server to use, based on the parent /8 block. This provides a crude granularity to the responsible RIR, mapping ARIN and LacNIC to node G, AfriNIC and RIPE to node H, and APNIC to node F.

The AdvertID= variable permits the head-end CGI handler to select different experiment criteria, so we can use the same control logic to run DNSSEC and IPv6 experiments, or vary the experiment behaviour slightly for a subset of clients.

This call is embedded directly in the flash experiment (see code in appendix).

JavaScript

Two forms of JavaScript are used. One is a pre-defined `.js` file which can be included along with configuration in a website, and then subsequently hand-tuned by the website manager to perform variants of the experiment, and send results to google analytics.

This is documented at: http://labs.apnic.net/tracker.shtml

And http://labs.apnic.net/script.shtml

```
<script type='text/javascript'>
  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'XX-YYYYYYYY-Z']);   // your google analytics tracking account ID
  _gaq.push(['_setDomainName', '.my.dom.ain']);  // your domain being tracked

  (function() {
    var ga = document.createElement('script');
    ga.type = 'text/javascript';
    ga.async = true;
    ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') +
                  '.google-analytics.com/ga.js';
    var s = document.getElementsByTagName('script')[0];
    s.parentNode.insertBefore(ga, s);
    })();

// enable the APNIC IPProtoTest and feed google analytics events
if ('http:' == document.location.protocol) {
   var ipproto_user = '968060';
   (function() {
    var iga = document.createElement('script'); iga.type = 'text/javascript'; iga.async = true;
    iga.src = 'http://labs.apnic.net/ipprototest.js';
    var is = document.getElementsByTagName('script')[0];
    is.parentNode.insertBefore(iga, is);
    })();  }
</script>
```

The test logic itself is loaded from http://labs.apnic.net/ipprototest.js

This version of the code can be configured to run different experiments by tuning variables passed in the ipproto_opts {…} structure.

An Alternative JavaScript system can be embedded in the website markup as follows:

```
<html>
 <head>
  <title>pack my box with six dozen liquor jugs</title>
 </head>
 <body>
  <h2>it works!</h2>

  <script>
      var ipproto_opts = {
       'docookies'        : false,
      };
  </script>

  <script type='text/javascript'
         src='http://drongo.rand.apnic.net/measureipv6js.cgi?advertID=1212'></script>
 </body>
</html>
```

This shows a small embedded instance of head-end configuration, which sends a download of the complete .JS to the client, which then executes this JavaScript. This variant of the code uses the head-end server to ‘minimize’ the JavaScript to the specific test set required for this experiment, and is therefore not a fully general case.

The generalized JavaScript code, and this specific code, is included as an example in the appendix.

Appendix 1: Flash Code

The Flash Code has been created using HaXe as a development language, which is converted by a HaXe compiler (ML) into actionscript compatible with flash version 8.

The sourcecode is as follows:

Template.hx

```
import flash.net.SharedObject;

import flash.display.MovieClip;
import flash.text.TextField;
import flash.text.TextFieldAutoSize;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.utils.Timer;
import flash.Lib;

class Prober extends MovieClip {
    var begun    : Int;
    var tests    : Array<String>;
    var results  : Hash<Int>;
    var timeout  : Timer;
    var baseUrl  : String;
    var argsUrl  : String;
    var testUrl  : String;
    var complete : Bool;
    var debug    : Bool;

    static function bind(target, handler, args:Array<Dynamic>)
    {
        return function(Dynamic):Void { return Reflect.callMethod(target, handler, args); }
    }

    public function new () {
        super();

        debug = ##DEBUG##;      // false
        complete = false;

        if (debug) trace("Starting up");

        tests = ##TESTSET##;  //["rd.td", "r4.td", "r6.td"];
        results = new Hash<Int>();

        // Construct base URL refs
        var now = Std.int(Date.now().getTime());
        var clientId = "##CLIENTID##";  // 007
        var scriptId = "##SCRIPTID##";  // f001
        var testHost = "##BASEURL##";   // labs.apnic.net
        argsUrl = "t10000.u" + now + '.s' + now + ".i" + clientId + ".v";
        argsUrl += scriptId + ".";
        baseUrl = "http://" + argsUrl;
        testUrl = "." + testHost + "/1x1.png?" + argsUrl;

        // Nothing in initialiser
        begun = flash.Lib.getTimer();

        // Construct URL loaders
        for (test in tests) {
            var url = baseUrl + test + testUrl + test;
            if (debug) trace("Requesting " + url);
            var req:URLRequest = new URLRequest(url);
            var link = new URLLoader();
            link.load(req);

            link.addEventListener(flash.events.Event.COMPLETE,
                    bind(this, probeSucceededHandler, [test]));
        }

        if (##V6LIT##) {
            var test = "v6lit";
            tests.push(test);
            var url = "http://[##V6HOST##]/1x1.png?" + argsUrl + test;
            if (debug) trace("Requesting " + url);
            var req:URLRequest = new URLRequest(url);
            var link = new URLLoader();
            link.load(req);

            link.addEventListener(flash.events.Event.COMPLETE,
                    bind(this, probeSucceededHandler, [test]));
        }

        timeout = new Timer(##TIMEOUT##, 1);            // 10000
        timeout.addEventListener(flash.events.TimerEvent.TIMER_COMPLETE,
                probesTimedOut);
        timeout.start();
        if (debug) trace("Started up successfully");
    }

    function probeSucceededHandler(test:String) {
        var elapsed = flash.Lib.getTimer() - begun;
        results.set(test, elapsed);

        if (debug) trace("Completed test " + test);

        // If done...
        if (Lambda.count(results) == Lambda.count(tests)) {
            timeout.stop();
            reportFinalResults();
        }
    }

    function probesTimedOut(e:Dynamic) {
        if (debug) trace("Timed out");
        reportFinalResults();
    }

    function reportFinalResults() {
        if (complete) return;
        var times = "";
        for (test in tests) {
            var result = results.get(test);
            times = times + "z" + StringTools.replace(test, '.', '') + "-";
            times = times + result + ".";
        }
        var url = baseUrl + times + "results" + testUrl + times;
        if (debug) trace("Sending results: " + url);
        var req:URLRequest = new URLRequest(url);
        var link = new URLLoader();
        link.load(req);
        complete = true;
    }

    static function main() {
        var prober = new Prober();
        flash.Lib.current.addChild(prober);
    }
}

class Main {
    static function main() {
        var i = flash.Lib.attach("picture");
        flash.Lib.current.addChild(i);
        if (##BORDER## > 0) {           // 10
            var border:flash.display.Shape = new flash.display.Shape();
            flash.Lib.current.addChild(border);
            border.graphics.lineStyle(##BORDER##, 0x##COLOR##);
            border.graphics.drawRect(0, 0, flash.Lib.current.stage.stageWidth,
                    flash.Lib.current.stage.stageHeight);
        }
        if (##USECOOKIE##) {            // false
            var cdb = SharedObject.getLocal("v6test", "/");
            var oneDayOld = Date.now().getTime() / 1000 - ##RATELIMIT##;
            if (cdb.data.lastRun == null || cdb.data.lastRun < oneDayOld) {
                var s = new Prober();
                flash.Lib.current.addChild(s);
                cdb.setProperty("lastRun", oneDayOld + ##RATELIMIT##);
                cdb.flush(1000);
            }
        } else {
            var s = new Prober();
            flash.Lib.current.addChild(s);
        }
        var b = new flash.display.SimpleButton();
        b.hitTestState = i;
        b.enabled = true;
```

```
            b.useHandCursor = true;
            b.addEventListener(flash.events.MouseEvent.MOUSE_UP, function(e) {
                var url = flash.Lib.current.root.loaderInfo.parameters.##PARAM##;
                var req = new flash.net.URLRequest(url);
                flash.Lib.getURL(req, "_blank");
            });
            flash.Lib.current.addChild(b);
            flash.Lib.current.stop();
        }
    }
```

URLTemplate.hx

```
    import flash.net.SharedObject;

    import flash.display.MovieClip;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.events.Event;
    import flash.utils.Timer;
    import flash.Lib;

    class Prober extends MovieClip {
        var begun     : Int;
        var tests     : Array<String>;
        var results   : Hash<Int>;
        var resultUrl : String;
        var timeout   : Timer;
        var complete  : Bool;
        var debug     : Bool;

        static function bind(target, handler, args:Array<Dynamic>)
        {
            return function(event:Event):Void {
                args.unshift(event);
                return Reflect.callMethod(target, handler, args);
            }
        }

        static function DJBHash(str:String):UInt
        {
            var hash:UInt=5381;
            var len:UInt = str.length;

            for (i in 0...len) {
                hash=((hash << 5) + hash) + cast(str.charCodeAt(i), UInt);
            }

            return hash;
        }

        // url: the clickable link from the ad network
        public function new (url:String) {
            super();

            debug = ##DEBUG##;      // false
            complete = false;

            if (debug) trace("Starting up");

            results = new Hash<Int>();

            // Get a hash of the URL parameter, this is quite random
            var hash = DJBHash(url);

            // Get the list of tests
            var listurl = "##LISTURL##&hash=" + hash;
            if ((hash & 1) == 0) listurl = "##LISTURL2##&hash=" + hash;
            if (debug) trace("Requesting " + listurl);
            var req:URLRequest = new URLRequest(listurl);
            var link = new URLLoader();
            link.addEventListener(Event.COMPLETE,
                    bind(this, urlListFetched, [hash]));
            link.load(req);
        }

        function urlListFetched(event:Event, hash:String) {
            var loader:URLLoader = event.target;
            try {
```

```
            var lines = StringTools.rtrim(loader.data).split("\n");
            tests = new Array<String>();
            timeout = new Timer(##TIMEOUT##, 1);            // 10000
            timeout.addEventListener(flash.events.TimerEvent.TIMER_COMPLETE,
                    probesTimedOut);
            timeout.start();
            begun = flash.Lib.getTimer();
            for (test in lines) {
                if (debug) trace("Added test: " + test);
                var bits:Array<String> = test.split("\t");
                var testname:String = bits[0];
                var testurl:String = bits[1];
                if (testname == 'results') {
                    resultUrl = testurl;
                } else {
                    tests.push(testname);
                    if (debug) trace("Requesting " + testurl);
                    var req:URLRequest = new URLRequest(testurl);
                    var link = new URLLoader();
                    link.addEventListener(flash.events.Event.COMPLETE,
                        bind(this, probeSucceededHandler, [testname]));
                    link.load(req);
                }
            }
        } catch (e:Dynamic) {
            if (##DEBUG##) trace(Std.string(e));
        }
    }

    function probeSucceededHandler(event:Event, test:String) {
        var elapsed = flash.Lib.getTimer() - begun;
        results.set(test, elapsed);

        if (debug) trace("Completed test " + test);

        // If done...
        if (Lambda.count(results) == Lambda.count(tests)) {
            timeout.stop();
            reportFinalResults();
        }
    }

    function probesTimedOut(e:Dynamic) {
        if (debug) trace("Timed out");
        reportFinalResults();
    }

    function reportFinalResults() {
        if (complete) return;
        var times = "";
        for (test in tests) {
            var result = results.get(test);
            times = times + "z" + StringTools.replace(test, '.', '') + "-";
            times = times + result + ".";
        }
        var url = resultUrl + times;
        if (debug) trace("Sending results: " + url);
        var req:URLRequest = new URLRequest(url);
        var link = new URLLoader();
        link.load(req);
        complete = true;
    }
}

class URLMain {
    static function main() {
    try {
        var url = flash.Lib.current.root.loaderInfo.parameters.##PARAM##;

        var i = flash.Lib.attach("picture");
        flash.Lib.current.addChild(i);
        if (##BORDER## > 0) {           // 10
            var border:flash.display.Shape = new flash.display.Shape();
            flash.Lib.current.addChild(border);
            border.graphics.lineStyle(##BORDER## * 2, 0x##COLOR##);
            border.graphics.drawRect(0, 0, flash.Lib.current.stage.stageWidth,
                    flash.Lib.current.stage.stageHeight);
        }
        if (##USECOOKIE##) {            // false
            var cdb = SharedObject.getLocal("v6test", "/");
            var oneDayOld = Date.now().getTime() / 1000 - ##RATELIMIT##;
            if (cdb.data.lastRun == null || cdb.data.lastRun < oneDayOld) {
                var s = new Prober(url);
                flash.Lib.current.addChild(s);
                cdb.setProperty("lastRun", oneDayOld + ##RATELIMIT##);
                cdb.flush(1000);
            }
        } else {
            var s = new Prober(url);
            flash.Lib.current.addChild(s);
        }
        var b = new flash.display.SimpleButton();
        b.hitTestState = i;
        b.enabled = true;
        b.useHandCursor = true;
        b.addEventListener(flash.events.MouseEvent.MOUSE_UP, function(e) {
            var req = new flash.net.URLRequest(url);
            flash.Lib.getURL(req, "_blank");
        });
        flash.Lib.current.addChild(b);
        flash.Lib.current.stop();
    } catch (e:Dynamic) {
        if (##DEBUG##) {
            trace(Std.string(e));
        }
    }
    }
}
```

This is processed through the compiler via a web-hosted engine which uses `swfmill` and `haxe` as follows:

```
#!/usr/local/bin/python

import cgi, os, sys
import cgitb
import math, string

from PIL import Image
from subprocess import Popen, PIPE, STDOUT

sizes= ["728x90",
      "468x60",
      "200x200",
      "120x600",
      "160x600",
      "250x250",
      "300x250",
      "336x280" ]

MAXSIZ=1024*35

def template_process(fin, fout, keyvals):
    for line in fin:
        for sub in keyvals:
            line = line.replace(sub, keyvals[sub])
        fout.write(line)

def fail(reason):
    print "Content-Type: text/html"
    print "Status: 400 Bad Data"
    print
    print "<!doctype html>"
    print "<html><head><title>SWF build failed</title></head>"
    print "<body><h1>SWF build failed</h1>"
    print "<p>"
    print cgi.escape(reason)
    print "<p><a href='http://labs.apnic.net/v6ad/urlad.cgi'>Try Again</a>"
    print "</p></body></html>"
    os.unlink(fn)
    sys.exit()


cgitb.enable()

form = cgi.FieldStorage()

# Get filename here.
fileitem = form['user_file']
advertID = form['advertID'].value
```

```
# Test if the file was uploaded
if fileitem.filename:
   # strip leading path from file name to avoid
   # directory traversal attacks
   filename = os.path.basename(fileitem.filename)
   fn = '/tmp/' + filename
   fh = open(fn, 'wb')
   fh.write(fileitem.file.read())
   fh.close()

siz=os.path.getsize(fn)
if siz > MAXSIZ:
    fail("Image too big: 35kb limit (%dkb)" % (siz/1024))

try:
    im = Image.open(fn)
except IOError:
    fail("Bad image")

wid,hei = im.size
widhei=str(wid)+'x'+str(hei)

if widhei not in sizes:

    fail("Image not correct size: %s not in %s" % (widhei, str(sizes)))


# retained in case we do static tests at some stage but currently useless
#tests = [t for t in ("rd.td", "r4.td", "r6.td", "rd.t6")
#          if form.getvalue(t, "off") == "on"]

template_process(
    open("/var/www/v6ad/URLTemplate.hx", "r"),
    open("/var/www/v6ad/out/URLMain.hx", "w"),
    {
        '##DEBUG##': "false",
        '##LISTURL##': "http://results.h.labs.apnic.net/measureipv6id.cgi?advertID="+advertID,
        '##LISTURL2##':
"http://results.g.labs.apnic.net/measureipv6id.cgi?advertID="+advertID,
        '##TIMEOUT##': "10000",
        '##BORDER##': "0",
        '##COLOR##': "ffffff",
      "##SCRIPTID##": str(advertID),
        '##USECOOKIE##': "false",
        '##RATELIMIT##': str(24 * 3600),
        '##PARAM##': "clickTAG"
    }
)

template_process(
    open("/var/www/v6ad/Template.swfml", "r"),
    open("/var/www/v6ad/out/imagelib.swfml", "w"),
    { '##WIDTH##': str(wid), '##HEIGHT##': str(hei), '##IMAGE##': fn }
)

# Construct imagelib.swf
working = "/var/www/v6ad/out"
swfmill = ("/usr/local/bin/swfmill", "simple", "imagelib.swfml", "imagelib.swf")
proc = Popen(swfmill, stdout=PIPE, stderr=STDOUT, cwd=working)
output, err = proc.communicate()
if proc.poll():
    fail("Building SWF image library failed: " + output)

# Construct v6test.swf
haxe = ("/usr/local/bin/haxe", "-swf-version", "10", "-swf-header",
    "%d:%d:12:FFFFFF" % (wid, hei), "-swf9", "urltest.swf", "-main", "URLMain",
    "-swf-lib", "imagelib.swf")
proc = Popen(haxe, stdout=PIPE, stderr=STDOUT, cwd=working)
output, err = proc.communicate()
if proc.poll():
    fail("Building final output SWF failed: " + output)

# make the test harness
template_process(
    open("/var/www/v6ad/TemplateHarness.html", "r"),
    open("/var/www/v6ad/out/index.html", "w"),
    { '##WIDTH##': str(wid), '##HEIGHT##': str(hei), '##IMAGE##': "urltest.swf" }
)

# clean up the tempfile
os.unlink(fn)
```

```
swf = open("/var/www/v6ad/out/urltest.swf", "rb")
swfbytes = swf.read()
swfsize = len(swfbytes)

print "Content-Type: application/octet-stream"
print "Content-Length:", swfsize
print "Content-Disposition: attachment; filename=urltest.swf"
print
print swfbytes
```

Appendix 2: the JavaScript.

The default javascript, ipprototest.js, is as follows:

```
// GPLv3
// $Id: ipprototest.js 27701 2011-01-26 15:21:50Z eaben $
// 2011 Hacked on by Byron Ellacott, APNIC
// 2011 Hacked on by George Michaelson, APNIC
// Written by Emile Aben, RIPE NCC
// Code inspired by Sander Steffann's IPv6test at http://v6test.max.nl/
(function() {
   var __ipprototest;

   IPProtoTest = function (opts) {
      if ( this instanceof IPProtoTest ) {
         this._version = '10i';
         this._done = false;
         this.userId          = '';
         this.timeout         = 10000; // 10 seconds
         this.noCheckInterval = 86400000; // 1 day
         this.domainSuffix    = 'labs.apnic.net';
         this.testSet         = ['r4.td','rd.td','r6.td'];
         this.testSetLen      = this.testSet.length; // shortcut
       this.randomize       = false;     // disable shuffling
       this.dotunnels       = false;     // disable tunnel testing
       this.dov6literal     = true;// disable tunnel testing
       this.dov6dns         = true;// disable tunnel testing
       this.docookies       = true;// enable cookie time testing
         this.sampling        = 1; // sampling frequency. 1 == disable
                             // eg 2 == 50% 20 == 5% 100 == 1%

         this._testsComplete = 0;
         this._now = new Date(); // keep track of init-time
         this._testTime = this._now.getTime();
         this._cookieExpire = new Date(this._testTime + this.noCheckInterval );
         this._testId = Math.floor(Math.random()*Math.pow(2,31));

         this._result = {};
         // sets this._cookie_last_run etc. vars
         // and set results from previous run (if available)
         this.parseCookies();
         //TODO compare the testset to set tested in cookies?

         // override defaults
         if ( opts instanceof Object) {
            for ( prop in opts ) {
               //safeguard is now done in the IPProtoTest(opt) call at the end of this file
               this[prop] = opts[prop];
            }

          if ( this.dov6dns ) {
            this.testSet.push('rd.t6');
          }
          if ( this.dov6literal ) {
            this.testSet.push('v6lit');
          }
          if ( this.dotunnels ) {
            this.testSet.push('v6stf');
            this.testSet.push('v6ter');
          }
          this.testSetLen = this.testSet.length;
          if ( this.randomize ) { this.shuffle( this.testSet ); }
         }

         // make object accessible for callbacks
         __ipprototest = this;

         // determine if tests need to be done
         if ( !this.docookies || !this._cookie_last_run ) {
            if ( this.sampling > 1 && Math.random() > 1/this.sampling ) {
               // not running test, but setting the cookie
               if (this.docookies) this.setCookie('__ipprototest_last_run',this._testTime);
               return this;
            }  else {
               this.startTest();
            }
         }
         return this;
      } else
         return new IPProtoTest(opts);
   };
   // public functions

   IPProtoTest.prototype.doGAQ=function() {
      var _gaq = window._gaq || []; // assuming google default
      if (this.GAQ instanceof Object) { // .. but allow override
         _gaq = this.GAQ;
      }
      //TODO document this
      var ipv4 = this._result.r4td ? 'yes' : 'no';
      var ipv6 = this._result.r6td ? 'yes' : 'no';
      var dual = this._result.rdtd ? 'yes' : 'no';
      if (this.dov6dns) {
       var v6dns = this._result.rdt6 ? 'yes' : 'no';
      }
      if (this.dov6literal) {
       var v6lit = this._result.v6lit ? 'yes' : 'no';
      }

      if (this.dotunnels) {
       var v6stf = this._result.v6stf ? 'yes' : 'no';
       var v6ter = this._result.v6ter ? 'yes' : 'no';
      }

      var summary = ((ipv4  == 'yes' ?  1 : 0) +
                    (ipv6  == 'yes' ?  2 : 0) +
                    (dual  == 'yes' ?  4 : 0));
      if (this.dov6dns) {
       summary += ((v6dns == 'yes' ?  8 : 0));
      }
      if (this.dov6literal) {
       summary += ((v6lit == 'yes' ? 16 : 0));
      }

      if (this.dotunnels) {
       summary += ((v6stf == 'yes' ? 32 : 0) +
                    (v6ter == 'yes' ? 64 : 0));
      }

      // Normalize v4val to 0 if the v4 test fails.
      // Normalize all other test times to relative to v4, if v4 worked
      // If a test fails, set to zero.
      // If v4 failed, set all tests to zero.
      // This ensures zero cases, and no v4 do not contribute to avg times
      var v4val = this._result.r4td ? this._result.r4td : 0;
      var v6val = this._result.r6td ? (this._result.r4td ? (this._result.r6td - v4val) : 0) :
0;
      var duval = this._result.rdtd ? (this._result.r4td ? (this._result.rdtd - v4val) : 0) :
0;
      if (this.dov6dns) {
       var dnval = this._result.v6dns ? (this._result.r4td ? (this._result.v6dns - v4val) : 0)
: 0;
      }
      if (this.dov6literal) {
       var lival = this._result.v6lit ? (this._result.r4td ? (this._result.v6lit - v4val) : 0)
: 0;
      }

      if (this.dotunnels) {
       var stval = this._result.v6stf ? (this._result.r4td ? (this._result.v6stf - v4val) : 0)
: 0;
       var teval = this._result.v6ter ? (this._result.r4td ? (this._result.v6ter - v4val) : 0)
: 0;
      }

      _gaq.push(['_trackEvent', 'ipprototest', 'ipv4:' + ipv4, 'Tested', 0]);
      _gaq.push(['_trackEvent', 'ipprototest', 'ipv6:' + ipv6, 'Tested', v6val]);
      _gaq.push(['_trackEvent', 'ipprototest', 'dual:' + dual, 'Tested', duval]);
      if (this.dov6dns) {
       _gaq.push(['_trackEvent', 'ipprototest', 'v6lit:' + v6lit, 'Tested', lival]);
      }
      if (this.dov6literal) {
       _gaq.push(['_trackEvent', 'ipprototest', 'v6dns:' + v6dns, 'Tested', dnval]);
      }

      if (this.dotunnels) {
       _gaq.push(['_trackEvent', 'ipprototest', 'v6stf:' + v6stf, 'Tested', stval]);
       _gaq.push(['_trackEvent', 'ipprototest', 'v6ter:' + v6ter, 'Tested', teval]);
      }
      // push summary line, this users testset.
      _gaq.push(['_trackEvent', 'ipprototest', 'summary:' + summary]);
   };


   IPProtoTest.prototype.onFinishInit=function() {
         if ( this.GAQ ) { // do Google analytics
            this.doGAQ();
         }
   };

   IPProtoTest.prototype.finishTest=function() {
      // cancel the timeout
      clearTimeout( __ipprototest._timeoutEvent );
      if (! __ipprototest._done ) {
         // report back results
         var pfx = __ipprototest.getTestPfx();
         for (var t_idx=0; t_idx < __ipprototest.testSetLen; t_idx++) {
            var test = __ipprototest.testSet[t_idx];
            test = test.replace(/\./g,'');
            pfx += [
               'z',
               test ,
               '-' ,
               __ipprototest._result[ test ] ?
                  __ipprototest._result[ test ] :
                  'null',
               '.'
            ].join('');
         }
         var imgURL = [
            'http://',
            pfx,
            'results.',
            __ipprototest.domainSuffix,
            '/1x1.png?',
            pfx
         ].join('');
         var req=document.createElement('img');
         req.src = imgURL; // loads it

         if(__ipprototest.docookies)          __ipprototest.setCookie('__ipprototest_last_run',
__ipprototest._testTime);
         // do all the stuff that needs to be done when
         // results are in
         __ipprototest.onFinishInit();
         __ipprototest._done = true;

        // if there is a callback function, invoke it
        if         (         __ipprototest.callback         instanceof         Function         )
__ipprototest.callback(__ipprototest._result);
      }
   };

   IPProtoTest.prototype.getTestPfx = function() {
      return [
         't', this.timeout, '.',
         'u', this._testTime , '.',
         's', this._testId , '.',
         'i', this.userId , '.',
         'v', this._version , '.'
      ].join('');
   };

   IPProtoTest.prototype.startTest = function() {
         var testPfx = this.getTestPfx();
         var testPath='/1x1.png?'+testPfx;
         for(var i=0;i<this.testSetLen;i++) {
            var subId = this.testSet[i];
          this._result[subId.replace(/\./g,'')] = false;
          if (subId == 'v6lit') {
             this.httpFetchImg(
               'http://[2401:2000:6660::f003]'+testPath+subId,
               '__ipprototest_'+subId.replace(/\./g,'') );
          } else if (subId == 'v6stf') {
             this.httpFetchImg(
               'http://[2401:2000:6660::f00a]'+testPath+subId,
               '__ipprototest_'+subId.replace(/\./g,'') );
          } else if (subId == 'v6ter') {
             this.httpFetchImg(
               'http://[2401:2000:6660::f00b]'+testPath+subId,
               '__ipprototest_'+subId.replace(/\./g,'') );
          } else {
             this.httpFetchImg(
               'http://'+testPfx+subId+'.'+this.domainSuffix+testPath+subId,
               '__ipprototest_'+subId.replace(/\./g,'') );
          }
         }
         this._timeoutEvent = setTimeout( this.finishTest, this.timeout );
   };

   // Fisher-Yates Shuffle
   IPProtoTest.prototype.shuffle=function( list ) {
      var i = list.length;
      if ( ! i ) return false;
      while ( --i ) {
         var j = Math.floor( Math.random() * ( i + 1 ) );
         var tmp_i = list[i];
         var tmp_j = list[j];
         list[i] = tmp_j;
         list[j] = tmp_i;
      }
      return true;
   };

   // cookie stuff
   IPProtoTest.prototype.setCookie=function(key,value) {
      document.cookie = key+'='+value+';expires='+this._cookieExpire.toUTCString()+';path=/';
   };

   IPProtoTest.prototype.parseCookies=function() {
      var _all = document.cookie.split(';');
      for(var i=0, len=_all.length;i<len;i++) {
         var _tmp =_all[i].split('=');
         var ckName=_tmp[0].replace(/^\s+|\s+$/g,'');
         var res_match;
         if(ckName=='__ipprototest_last_run'){
            this._cookie_last_run = parseInt(unescape(_tmp[1].replace(/^\s+|\s+$/g,'')),10);
         }
      }
   };

   function __ipprototest_onload(el) {
      // note to self : this = the img element, not the proto
      this._duration = new Date().getTime() - this._start;
      __ipprototest._testsComplete++;
      var res_match = this.name.match(/^__ipprototest_(\w+)$/);
      if ( res_match[1] ) { // codes like 'rdtd'
         __ipprototest._result[res_match[1]] = this._duration;
      }
      if (__ipprototest._testsComplete >= __ipprototest.testSetLen) {
         __ipprototest.finishTest();
      }
   }

   // inspired by gih_ajax-function by Geoff Huston, George Michaelson, Byron Ellacott (APNIC)
   IPProtoTest.prototype.httpFetchImg=function( url, name ) {
      var req=document.createElement('img');
      req.name = name;
      req._start = new Date().getTime();
      req.onload = __ipprototest_onload;
      req.src = url;
   };
})();

// initialize variables and structs to avoid null ref bugs
var ipproto_user = ipproto_user || 'anon';
var ipproto_opts = ipproto_opts || {};

// if we have been correctly initialized for google analytics setup by the user...

// the set of externally defined variables we will accept
// userID takes the ipproto_user variable by default,
//  or the value passed in array externally if provided
// defaults shown below.
var opts = {
     'docookies'  : true,           // test based on noCheckInterval in cookie
     'dov6dns'          : true,           // test v6 dns to a dual-stack URL
     'dov6literal'      : true,           // test a v6 literal URL
     'dotunnels'  : false,    // test for 6to4 and teredo tunnels
     'randomize'  : false,    // by default, sorted test order
     'noCheckInterval'  : 86400000, // interval to test on, if docoookies is true
     'sampling'         : 1,        // 1/sampling eg sampling=4 1/4th tested
     'userId'           : ipproto_user, // what to log in the collector website as
     'callback'        : function (results){},      // prototype function to receive callback
of results
     'GAQ'       : true      // hook into google analytics
};

if ( ipproto_opts instanceof Object ) {
      // for the list we know, if its in the externally set object, map it in.
      for ( prop in opts ) {
            if (prop in ipproto_opts) { opts[prop] = ipproto_opts[prop]; }
      }
}

var ippt = IPProtoTest(opts);
```

Reduced JavaScript

The reduced JavaScript supplied by the head-end configuration engine is as follows (this example shows a specific end-user configuration embedded inline.)

This version has a slightly different configuration and runtime invocation model, because it is embedded as a function call directly in the markup rather than as a text/javascript reference to a specific .js.

The head-end configuration engine for this, is also running other experiments using a file-backed mechanism which specifies the URLs for experiments from back-end configuration master files.
The code which emits this .js is as follows:

```
#!/usr/bin/env python2.6

import os
import sys
import getopt
import random
import re
import time
import cgi
import cgitb
import socket
import string

_debug = 0              # by default not debugging
_verbose = 0        # we run silent but deep...
ifn=""
sep=None            # null str == LWSP separation

eights= {
'0':'f',  '1':'f',  '2':'h',  '3':'g',  '4':'g',  '5':'h',  '6':'g',  '7':'g',  '8':'g',  '9':'g',  '10':'f',
'11':'g', '12':'g',
'13':'g',  '14':'f',  '15':'g',  '16':'g',  '17':'g',  '18':'g',  '19':'g',  '20':'g',  '21':'g',  '22':'g',
'23':'g', '24':'g',
'25':'h',  '26':'g',  '27':'f',  '28':'g',  '29':'g',  '30':'g',  '31':'h',  '32':'g',  '33':'g',  '34':'g',
'35':'g', '36':'f',
'37':'h',  '38':'g',  '39':'f',  '40':'g',  '41':'h',  '42':'f',  '43':'f',  '44':'g',  '45':'g',  '46':'h',
'47':'g', '48':'g',
'49':'f',  '50':'g',  '51':'h',  '52':'g',  '53':'g',  '54':'g',  '55':'g',  '56':'g',  '57':'g',  '58':'f',
'59':'f', '60':'f',
'61':'f',  '62':'h',  '63':'g',  '64':'g',  '65':'g',  '66':'g',  '67':'g',  '68':'g',  '69':'g',  '70':'g',
'71':'g', '72':'g',
'73':'g',  '74':'g',  '75':'g',  '76':'g',  '77':'h',  '78':'h',  '79':'h',  '80':'h',  '81':'h',  '82':'h',
'83':'h', '84':'h',
'85':'h',  '86':'h',  '87':'h',  '88':'h',  '89':'h',  '90':'h',  '91':'h',  '92':'h',  '93':'h',  '94':'h',
'95':'h', '96':'g',
'97':'g', '98':'g', '99':'g', '100':'g', '101':'f', '102':'h', '103':'f', '104':'g', '105':'h', '106':'f',
'107':'g',
'108':'g',  '109':'h',  '110':'f',  '111':'f',  '112':'f',  '113':'f',  '114':'f',  '115':'f',  '116':'f',
'117':'f', '118':'f',
'119':'f',  '120':'f',  '121':'f',  '122':'f',  '123':'f',  '124':'f',  '125':'f',  '126':'f',  '127':'f',
'128':'g', '129':'g',
'130':'g',  '131':'g',  '132':'g',  '133':'f',  '134':'g',  '135':'g',  '136':'g',  '137':'g',  '138':'g',
'139':'g', '140':'g',
'141':'h',  '142':'g',  '143':'g',  '144':'g',  '145':'h',  '146':'g',  '147':'g',  '148':'g',  '149':'g',
'150':'f', '151':'h',
'152':'g',  '153':'f',  '154':'h',  '155':'g',  '156':'g',  '157':'g',  '158':'g',  '159':'g',  '160':'g',
'161':'g', '162':'g',
'163':'f',  '164':'g',  '165':'g',  '166':'g',  '167':'g',  '168':'g',  '169':'g',  '170':'g',  '171':'f',
'172':'g', '173':'g',
'174':'g',  '175':'f',  '176':'h',  '177':'g',  '178':'h',  '179':'g',  '180':'f',  '181':'g',  '182':'f',
'183':'f', '184':'g',
'185':'h',  '186':'g',  '187':'g',  '188':'h',  '189':'g',  '190':'g',  '191':'g',  '192':'g',  '193':'h',
'194':'h', '195':'h',
'196':'h',  '197':'h',  '198':'g',  '199':'g',  '200':'g',  '201':'g',  '202':'f',  '203':'f',  '204':'g',
'205':'g', '206':'g',
'207':'g',  '208':'g',  '209':'g',  '210':'f',  '211':'f',  '212':'h',  '213':'h',  '214':'g',  '215':'g',
'216':'g', '217':'h',
'218':'f',  '219':'f',  '220':'f',  '221':'f',  '222':'f',  '223':'f',  '224':'f',  '225':'f',  '226':'f',
'227':'f', '228':'f',
'229':'f',  '230':'f',  '231':'f',  '232':'f',  '233':'f',  '234':'f',  '235':'f',  '236':'f',  '237':'f',
'238':'f', '239':'f',
'240':'f',  '241':'f',  '242':'f',  '243':'f',  '244':'f',  '245':'f',  '246':'f',  '247':'f',  '248':'f',
'249':'f', '250':'f',
'251':'f', '252':'f', '253':'f', '254':'f', '255':'f',
}

varnames=[
       "VERSION",
       "TIMEOUT",
       "RANDOMIZE",
       "IDENTITY",
       "STEM",
       "RESULTS",
       "STUB",
       "DOMAIN",
        ]

vars=dict()
tests=[]

randf=0

cpath="/usr/local/www/data/labs/confid" # standard path on labs clone hosts

# ${cpath}/${cstem}.${rnd} gets us rnd out of some m/n selection
#cpath="."          # for now, local files
cstem="serveflashconfig.txt"

# the .js we are going to invoke is inline here. it could be done as an external file.

def printIPP(ippOpts):

       ippStrTemplate = '''

// GPLv3
// $Id: ipprototest.js 27701 2011-01-26 15:21:50Z eaben $
// 2013 Hacked on by George Michaelson, APNIC to parameterize it across its call from a head-end configure
engine
// 2011 Hacked on by Byron Ellacott, APNIC
// 2011 Hacked on by George Michaelson, APNIC
// Written by Emile Aben, RIPE NCC
// Code inspired by Sander Steffann's IPv6test at http://v6test.max.nl/
(function() {

   // nothing is local. this probably now exists in global namespace but its
   // in the candidate set of 'names list likely to collide'
   var __ipprototest;

   var __ggmdebug__ = 1

   doIPProtoTest = function (opts) {

      if ( this instanceof doIPProtoTest ) {

         this._version        = '$ippversion';
         this._done           = false;
         this.userId          = '$ippuserid';
         this._testId         = '$ipptestid';
         this.timeout         = $ipptimeout;
         this.noCheckInterval = $ippnocheck;
         this.domainSuffix    = '$ippdomain';
         this.testSet         = $ipptestset;
         this.testSetLen      = this.testSet.length; // shortcut
        this.tests        = $ipptests;
        this.docookies       = $ippcookies;
         this.sampling        = $ippsample;    // sampling frequency. 1 == disable
                                          // eg 2 == 50% 20 == 5% 100 == 1%

         this._testsComplete = 0;
         this._now = new Date(); // keep track of init-time
         this._testTime = this._now.getTime();
         this._cookieExpire = new Date(this._testTime + this.noCheckInterval );

         this._result = {};
         // sets this._cookie_last_run etc. vars
         // and set results from previous run (if available)
         this.parseCookies();

         //TODO compare the testset to set tested in cookies?

         // override defaults
         if ( opts instanceof Object) {
            for ( prop in opts ) {
               //safeguard is now done in the doIPProtoTest(opt) call at the end of this file
               this[prop] = opts[prop];
            }
         }
         // make object accessible for callbacks
         __ipprototest = this;

         // determine if tests need to be done
         if ( !this.docookies || !this._cookie_last_run ) {
            if ( this.sampling > 1 && Math.random() > 1/this.sampling ) {
               // not running test, but setting the cookie
               if (this.docookies) this.setCookie('__ipprototest_last_run',this._testTime);
               return this;
            }  else {
               this.startTest();
            }
         }
         return this;
      } else
         return new doIPProtoTest(opts);
   };
   // public functions


   doIPProtoTest.prototype.getTestPfx = function() {
      return [
         't', this.timeout, '.',
         'u', this._testTime , '.',
         's', this._testId , '.',
         'i', this.userId , '.',
         'v', this._version , '.'
      ].join('');
   };

   doIPProtoTest.prototype.finishTest=function() {
      // cancel the timeout
      clearTimeout( __ipprototest._timeoutEvent );
      if (! __ipprototest._done ) {
         // report back results
         // var pfx = __ipprototest.getTestPfx();
         var pfx = ''
         for (var t_idx=0; t_idx < __ipprototest.testSetLen; t_idx++) {
            var test = __ipprototest.testSet[t_idx];
            test = test.replace(/\./g,'');
            pfx += [
               'z',
               test ,
               '-' ,
               __ipprototest._result[ test ] ?
                  __ipprototest._result[ test ] :
                  'null',
               '.'
            ].join('');
         }
         var imgURL = [
            '$ippresults',
            pfx
         ].join('');
         var req=document.createElement('img');
         req.src = imgURL; // loads it

         if(__ipprototest.docookies)                       __ipprototest.setCookie('__ipprototest_last_run',
__ipprototest._testTime);
         // do all the stuff that needs to be done when
         // results are in
         __ipprototest._done = true;

        // if there is a callback function, invoke it
        if ( __ipprototest.callback instanceof Function ) __ipprototest.callback(__ipprototest._result);
      }
   };

   doIPProtoTest.prototype.startTest = function() {
         for(var i=0;i<this.testSetLen;i++) {
            var subId = this.tests[i].exName;
           this._result[subId.replace(/\./g,'')] = false;
            this.httpFetchImg(
               this.tests[i].exUrl,
               '__ipprototest_'+subId.replace(/\./g,'') );
         }
         this._timeoutEvent = setTimeout( this.finishTest, this.timeout );
   };

   // cookie stuff
   doIPProtoTest.prototype.setCookie=function(key,value) {
      document.cookie = key+'='+value+';expires='+this._cookieExpire.toUTCString()+';path=/';
   };

   doIPProtoTest.prototype.parseCookies=function() {
      var _all = document.cookie.split(';');
      for(var i=0, len=_all.length;i<len;i++) {
         var _tmp =_all[i].split('=');
         var ckName=_tmp[0].replace(/^\s+|\s+$/g,'');
         var res_match;
         if(ckName=='__ipprototest_last_run'){
            this._cookie_last_run = parseInt(unescape(_tmp[1].replace(/^\s+|\s+$/g,'')),10);
         }
      }
   };

   function __ipprototest_onload(el) {
      // note to self : this = the img element, not the proto
      this._duration = new Date().getTime() - this._start;
      __ipprototest._testsComplete++;
      var res_match = this.name.match(/^__ipprototest_(\w+)$/);
      if ( res_match[1] ) { // codes like 'rdtd'
         __ipprototest._result[res_match[1]] = this._duration;
      }
      if (__ipprototest._testsComplete >= __ipprototest.testSetLen) {
         __ipprototest.finishTest();
      }
   }

   // inspired by gih_ajax-function by Geoff Huston, George Michaelson, Byron Ellacott (APNIC)
   doIPProtoTest.prototype.httpFetchImg=function( url, name ) {
      var req=document.createElement('img');
      req.name = name;
      req._start = new Date().getTime();
      req.onload = __ipprototest_onload;
      req.src = url;
   };
})();

// initialize variables and structs to avoid null ref bugs
var ipproto_user = ipproto_user || 'anon';
var ipproto_opts = ipproto_opts || {};

// the set of externally defined variables we will accept.
// users can preset these before calling us and we 'honour' them.

// userID takes the ipproto_user variable by default,
//  or the value passed in array externally if provided
// defaults shown below.
var opts = {
     'docookies'    : $ippcookies,             // test based on noCheckInterval in cookie
     'noCheckInterval'  : $ippnocheck,
     'sampling'           : $ippsample,       // 1/sampling eg sampling=4 1/4th tested
     'callback'           : function (results){}     // prototype function to receive callback of results
};

if ( ipproto_opts instanceof Object ) {
       // for the list we know, if its in the externally set object, map it in.
       for ( prop in opts ) {
              if (prop in ipproto_opts) { opts[prop] = ipproto_opts[prop]; }
       }
}

var ippt = doIPProtoTest(opts);
'''

       ippTemplate = string.Template(ippStrTemplate)

       print ippTemplate.safe_substitute(ippOpts)



# main is the go.. but we run in __main__
def main(argv):

        global _debug, _verbose, ifn, sep,randf

        # args processing.
        try:

                opts, args = getopt.getopt(argv, "hvdri:", ["help", "verbose", "debug", "random", "ip"])

        except getopt.GetoptError:
                usage()
                sys.exit(2)

        for opt, arg in opts:
                if opt in ("-h", "--help"):
                        usage()
                        sys.exit()
                elif opt in ("-d", "--debug"):
                        _debug += 1
                    if (_debug > 1):
                                sys.stderr.write("debug %d\n" % (_debug))

                elif opt in ("-v", "--verbose"):
                        _verbose += 1
                    if (_debug > 0):
                                sys.stderr.write("verbose\n")
                elif opt in ("-r", "--random"):
                        randf += 1
                    if (_debug > 1):
                                sys.stderr.write("random %d\n" % (randf))
                elif opt in ("-i", "--ip"):
                        os.environ['REMOTE_ADDR'] = arg
                    if (_debug > 1):
                                sys.stderr.write("REMOTE_ADDR %d\n" % (arg))
        cgitb.enable()

        form = cgi.FieldStorage()
        if randf:
              ifn=os.path.join(cpath, '.'.join((cstem, str(random.randint(0, 15)))))
        else:
              if os.environ.has_key('REMOTE_ADDR'):
                    host = os.environ['REMOTE_ADDR'].split('.')[0]
                    if ':' in host:
                          host = '202'
                    ifn=os.path.join(cpath, '.'.join((cstem, eights[host])))
              else:
                    ifn=os.path.join(cpath, '.'.join((cstem, 'f')))

        # to turn off 6022 behaviour, change this line to 9999. to turn on, change to 6022
        # override experiment 6022
        if form.has_key('advertID') and int(form['advertID'].value) == 2407:
              userid=str(int(random.random()*10000000000))
                 timeval=str(int(time.time()))

              # Create a UDS socket
              sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)

              # Connect the socket to the port where the server is listening
              server_address = '/tmp/ggm/uds_socket'

              try:
                  sock.connect(server_address)
              except socket.error, msg:
                  print >>sys.stderr, msg
                  seqn='00003'
                  bads='00004'

              try:
                  # Send data
                  seqn = sock.recv(1024)
                  bads = "%05x" % (int(seqn, 16)+1)

              finally:
                  sock.close()

              print '''Cache-Control: no-cache
Expires: Thu, 1 Jan 1970 00:00:00 UTC
Content-Type: text/plain

d
        http://d.t10000.u%s.s%s.i868.v6022.%s.z.dotnxdomain.net/1x1.png?d.t10000.u%s.s%s.i868.v6022.%s.z.dotn
xdomain.net
e
        http://e.t10000.u%s.s%s.i868.v6022.%s.z.dashnxdomain.net/1x1.png?e.t10000.u%s.s%s.i868.v6022.%s.z.das
hnxdomain.net
f
        http://f.t10000.u%s.s%s.i868.v6022.%s.z.dotnxdomain.net/1x1.png?f.t10000.u%s.s%s.i868.v6022.%s.z.dotn
xdomain.net
results
        http://results.t10000.u%s.s%s.i868.v6022.%s.x.rand.apnic.net/1x1.png?results.t10000.u%s.s%s.i767.v602
2.%s&r=
'''    %    (userid,timeval,seqn,    userid,timeval,seqn,    userid,timeval,seqn,    userid,timeval,seqn,
userid,timeval,bads, userid,timeval,bads, userid,timeval,seqn, userid,timeval,seqn)

        else:

              try:
                    ifp=open(ifn)
                    if (_debug > 1):
                          sys.stderr.write("open(%s)\n" % (ifn))
              except Exception, e:
                    sys.stderr.write("open(%s):%s\n" % (ifn, str(e)))
                            sys.exit(2)

              userid=int(random.random()*10000000000)
              timeval=int(time.time())
              for i in ifp:
                    i = i[:-1]
                    if (_debug > 3):
                          sys.stderr.write("read(%s)\n" % (i))
                    w = i.split(sep)
                    if (_debug > 2):
                          sys.stderr.write("w[%s]\n" % ','.join(w))

                    # deal with reserved variable-words. rest are tests.
                    if w[0] in varnames:
                          vars[w[0]] = w[1]
                    else:
                          tests.append(w)

              # overrides.

              # override from form if passed as arg. Otherwise, wire value if not in config
              if form.has_key('advertID'):
                    vars['VERSION'] = int(form['advertID'].value)
              elif not vars.has_key('VERSION'):
                    vars['VERSION'] = 1234
```

```
            # only override if not in config
            if not vars.has_key('TIMEOUT'):
                  vars['TIMEOUT'] = 10000

            # only override if not in config
            if not vars.has_key('DOMAIN'):
                  vars['DOMAIN'] = 'labs.apnic.net'

            if (_debug > 2):
                  sys.stderr.write("vars["+str(vars)+']\n')
                  sys.stderr.write("tests"+str(tests)+'\n')

            print '''Cache-Control: no-cache
Expires: Thu, 1 Jan 1970 00:00:00 UTC
Content-Type: text/javascript
'''
            # if we set RANDOMIZE, then shuffle
            if vars['RANDOMIZE'] == '1':
                  random.shuffle(tests)

            # apply prints to RESULTS and STUB
            for i in ['RESULTS', 'STUB']:
                  vars[i] = re.sub('%v', str(vars['VERSION']), vars[i])
                  vars[i] = re.sub('%i', str(vars['IDENTITY']), vars[i])
                  vars[i] = re.sub('%t', str(vars['TIMEOUT']), vars[i])
                  vars[i] = re.sub('%u', str(userid), vars[i])
                  vars[i] = re.sub('%s', str(timeval), vars[i])

            # apply vars to % printf() in tests and emit
            # have to apply %r per loop iteration because its
            # value depends on the test in question applied to STUB
            tarray=[]
            for t in tests:
                  exname = t[0]
                  # has no substitutions. its a literal. do the minimum
                  if '%' not in t[1]:
                        restub = re.sub('%r', t[0], vars['STUB'])
                        exURL = 'http://%s/%s' % (t[1], restub)
                  else:
                  # do the substitutions.
                        outs = re.sub('%v', str(vars['VERSION']), t[1])
                        outs = re.sub('%i', str(vars['IDENTITY']), outs)
                        outs = re.sub('%t', str(vars['TIMEOUT']), outs)
                        outs = re.sub('%u', str(userid), outs)
                        outs = re.sub('%s', str(timeval), outs)
                        restub = re.sub('%r', t[0], vars['STUB'])
                        # if its fully specified, don't append
                        if outs.endswith('.'):
                              exURL = 'http://%s/%s' % (outs[-1], restub)
                        else:
                              exURL = 'http://%s.%s/%s' % (outs, vars['STEM'], restub)
                  tarray.append("{'exName':'%s', 'exUrl':'%s'}" % (exname, exURL))

            # this loads and prints the javascript function, and its invocation.

            doIPPopts = {
                'ippversion' : vars['VERSION'],   # from the config data
                'ippuserid'  : vars['IDENTITY'],  # what to log in the collector website as
                'ipptestid'  : userid,            # the uXXXXXXXXX value
                'ipptimeout' : vars['TIMEOUT'],   # from the config data
                'ippnocheck' : 86400 * 1000 * 30, # interval to test on, if docoookies is true
              'ippdomain'  : vars['DOMAIN'],
                'ipptestset' : map(lambda x: x[0], tests), # from the list of tests, in order
                'ippcookies' : 1,                 # test based on noCheckInterval in cookie
                'ippsample'  : 1,                 # 1/sampling eg sampling=4 1/4th tested
                'ipptests'   : '['+','.join(tarray)+']',    # a string rep of [ { test: url }, { test: url
}, ... ]
                'ippresults' : ''.join(['http://',vars['RESULTS']]) # the results URL as encoded from the
config
              }

            # all in one template emission of the .js with these params, and its execution follows...
            printIPP(doIPPopts)



def usage():
        print 'Usage: ' + sys.argv[0] + ''' -h -v -d
        -h print this display
        -v verbose (print lines as classified)
        -d debug mode
        -r random mode (by default selects on /8)
      -i <ip>       set <ip> into set -x REMOTE_ADDR="<ip>"

      VERSION             numeric # eg 1001
      IDENTITY      numeric # eg 9999
      TIMEOUT             numeric # eg 10000 for 10sec
      RANDOMIZE     0 or 1 (to enable)
      STEM          dom.ain to apply to names
      STUB          /1x1.png? argument to URL
      RESULTS             dom.ain to send results to
      test-name     fmt-string
      :
      %v version from VERSION
      %i identity from IDENTITY
      %t timeout (millisec) from TIMEOUT
      %u userid from random()
      %s time in seconds since 1970
      '''


# the real main

if __name__ == "__main__":
        if (len(sys.argv) <= 0):
              usage()
              sys.exit(2)

        main(sys.argv[1:])
```

Appendix C. Collating the data

Data about each experiment has three sources of capture:

1. TCPDUMP of port 53 (dns) and port 80 (web) to the authoritative DNS server and webserver, which are co-resident on the same host, even if using different IP addresses. Additional captures can be made of teredo and 6to4 tunnel bindings, to detect tunnel specific behaviours.
2. DNS query logs
3. Web query logs.

All three sources should contain events which relate to a specific u*.s* instance, unique to one user.

The query logs in DNS provide the relationship of this experiment-id to a specific resolver, or set of resolvers conducting the DNS queries. The query logs additionally identify DNS flags (EDNS0, DNSSEC OK, Checking Disabled…) and transport (UDP or TCP, IPv4 or IPv6).

The query logs in Web provide the Clients IPv4 and IPv6 address, for the experiments and permits the specific IPv4 and IPv6 pair to be related to each other. The *.results. lines records for each experiment id the clients own sense of the completion times for the experiments or 'null' if not completed, which provides details on the client view of experiment behaviour.

Disclaimer