Midterm Project Report

Tim Contois
Jake Downs
Josh Pikovsky

Project: Evaluating the Composition and Performance of the IPv6 Address Space with Traceroutes and Web Requests

Our Proposal

As described in the previous deliverable, our term project will be focused on examining and evaluating the IPv6 address space, and its various attributes, such as page load times and throughput. The objective of our work is to gain a better understanding of how popular websites and autonomous systems are adopting IPv6, and whether this change will affect the current Internet experience in any meaningful way for clients and providers. To accomplish this goal, our initial plan was to write a script that, given a list of Alexa top sites, would access the targets and produce statistics that could give us a better sense of how many sites support IPv6, and how the employment of this protocol affects access.

Our Progress

To start with, we have acquired a list of the top Alexa sites across fifteen different countries. As described in the proposal, our methodology is to access the top sites, collect objects (ex. images, etc), clean them for easy access, and then attempt to request these objects with both IPv4 and IPv6. Information about the connection, like request times, would be collected, and used to make conclusions about the IPv6 space, and how its implementation affects Internet users.

There are six primary scripts, which essentially take in raw data in the form of the top site URLs, and then work collaboratively to build an output JSON with the relevant statistics. The outline of our code base is as follows:

parser.py: this script takes in a list of websites and outputs a JSON that contains a list of aggregated website names from every country.

associator.py: this script takes the previous script's JSON as input, and visits every domain name listed. For each domain, it scrapes the site for objects (images, etc) that we can use for future requests. A JSON of domains and associated objects is outputted.

cleaner.py: this script takes in the JSON output by associator.py, and "cleans" up the objects that were found and associated with every domain. To make future requests easy, the script goes through each domain, and finds and builds an easily requestable object. The script labels the best (least likely to cause errors) object, and labels it as the "preferred" key. This association of domain and preferred object is then output into another JSON.

analyzer.py: this script runs the experiment. It takes in the JSON generated by cleaner.py, and executes a manual DNS lookup, and then times the requests. The DNS lookup provides information about IPv6 support, and then the timing experiment will use those statistics to decide whether or not to even attempt an IPv6 socket connection. A list of the request timings is then output as the final JSON.

runner.py: this script is a wrapper that allows us to run all the scripts sequentially with appropriately labelled output.

grapher.py: this script takes in a JSON of domain names with associated information about the attributes of the IPv4 and IPv6 requests, like the time and size of each. Using this information, graphs and visualizations are generated (discussed further in "Current Results").

By using these scripts, we are able to compile run a simplified version of our experiment. The input of websites is taken in, objects are found and formatted, and then those objects are requested with both IPv4 and IPv6 connections, and the resulting speeds recorded.

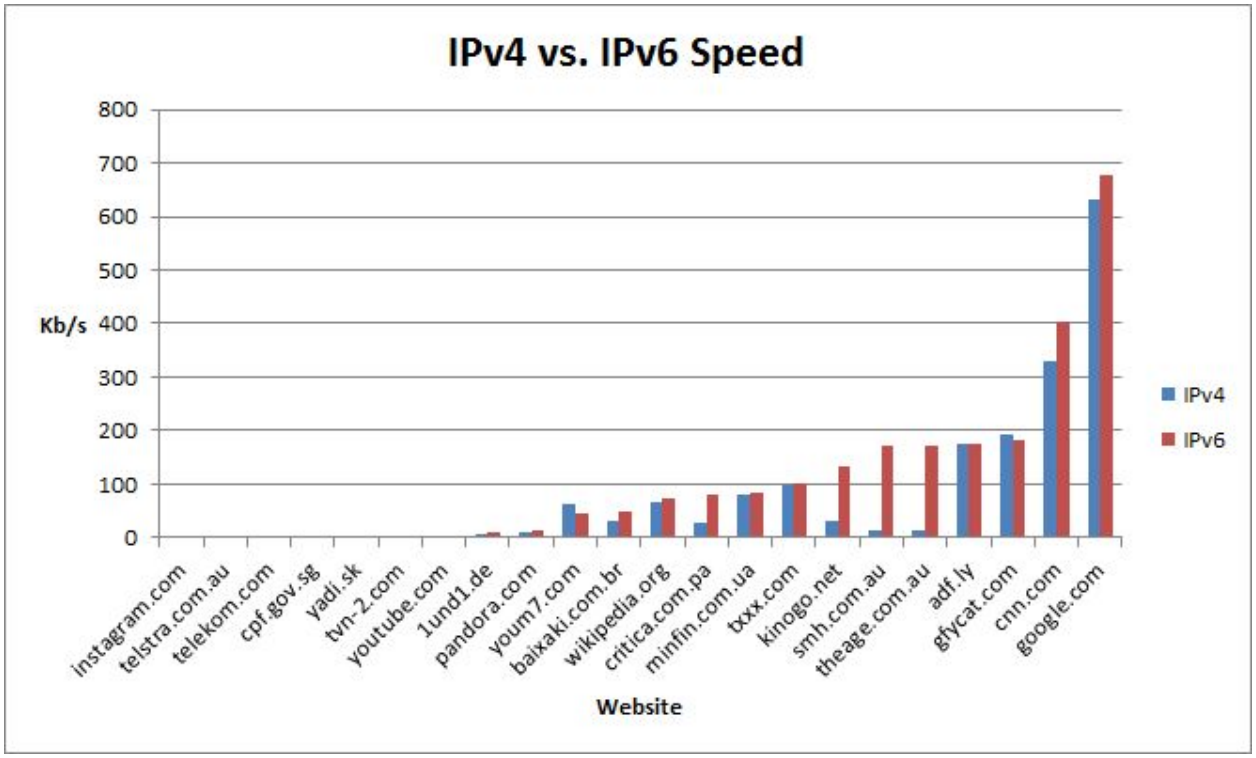
We ran into some issues with running our script with IPv6 connections. It was unclear for some time whether the mistake was on our end, or whether those sites just didn't support IPv6. We made some fixes to our code, but were still receiving address info errors. Eventually we diagnosed the problem as requests for bad objects, and were then able to connect via IPv6.

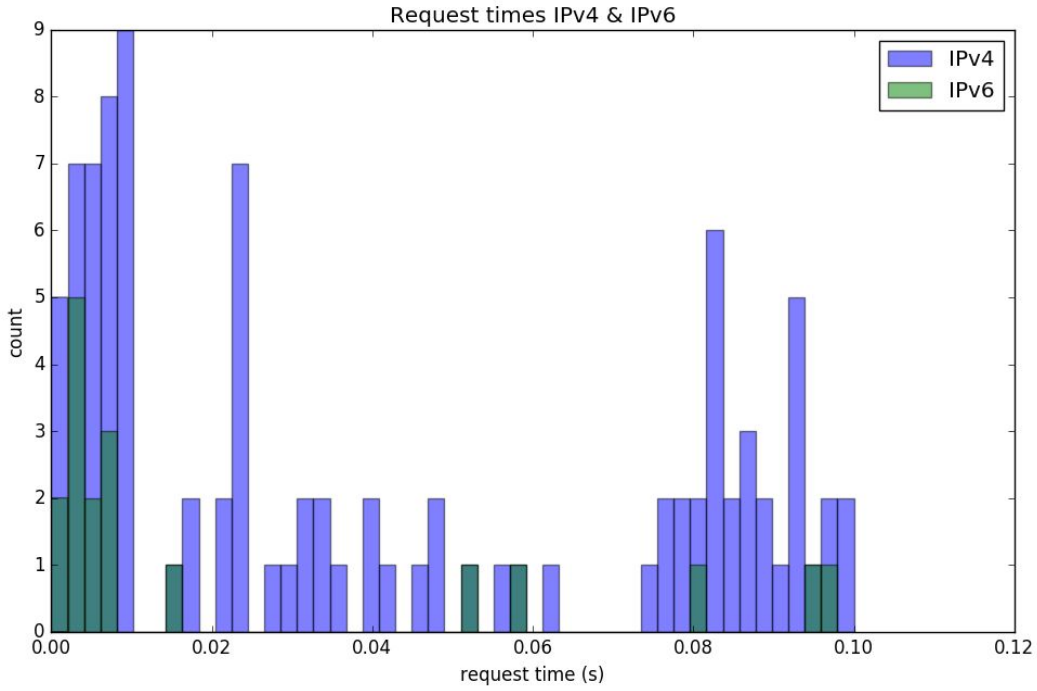
Current Results

After running the first four scripts (up through analyzer.py), the result is a JSON with information about the domains visited, the objects requested, and the request times for both IPv4 and IPv6. Websites that were not easily accessed are collected as well.
A simplified example of the format of our resultant JSONs is as follows:

```
{ "valid": {                          //valid domains stored here
        {"domain":                    //entries for each domain
                [objects]:{}          //objects stored for each domain
                "results": {          //statistics for IPv4 and IPv6
                        "Ipv4":{},
                        "Ipv6":{}
                }
},
 "exception": {}, "zeros":{}, "progress":{}}
```

Essentially, the JSON stores all the domains, their respective objects (with various attributes like the preference of the object and their IPv6 support, not included in chart above), and the request times for both IPv4 and IPv6. We have begun investigating the best way to visualize the statistics compiled so far, but for preliminary results we have prepared a histogram showing the distribution of request times for both protocol





From the list of top sites, we found 64 of 422 (15.1%) of sites examined supported IPv6.

Next Steps

We have several steps we can take to further our work. One possible step is to increase the amount of bytes we're receiving, since we are not currently receiving the full object being requested. Increasing the received bytes from 1024 to the size of the full object could alter the results. We could also add additional statistics to better our understanding of the IPv6 address space. This could include compensating for requests that are in other countries, which may be slowed down inherently by the distance.