

## EECS 325/425 Project 2

**Due December 5, 2008**

In this project, you will get an experience conducting a real Internet measurement. The experiment involves testing how good the actual server selection mechanism is in a real content delivery network. Specifically, you will try to discover a number of CDN servers, then imitate a Web download from several hosts, and then compare the distance between those hosts and the servers the CDN has selected for download with the distance between the same hosts and the other CDN servers (which were not selected). As is often the case in real Internet experiments, you will rely on existing tools, and your rate of progress will some times depend on external factors. The tool you will be using is our own DipZoom, available at [dipzoom.case.edu](http://dipzoom.case.edu). The project involves the following steps:

- Lay the groundwork. Familiarize yourself with DipZoom and a measurement study of CDNs, which describes a mechanism for CDN server discovery (Su et al, “Drafting behind Akamai”, SIGCOMM’2006, available in the projects folder). Read the DipZoom papers available on the Web site ([dipzoom.case.edu](http://dipzoom.case.edu)). (Note: you will see multiple references for paying for measurements in DipZoom. The payment mechanism is not enabled.) Install a DipZoom measuring point (MP) and the client on your computer. Download the client library, which you will be using to implement your experiments.
- Discover as many CDN servers as you can, using the technique described in the “Drafting” paper. Hint: use nslookup measurements from as many measuring points as you can for a hostname of objects hosted by Akamai. Debug your measurements using your own MP before submitting measurement requests to other MPs.
- Use the same measuring points to measure the distance to the CDN server selected by Akamai for a given MP and compare it with the distance to other CDN servers from the same MP. Try to use the hop number (using traceroute measurements), the RTT (using ping measurements) and the web download bandwidth (using wget or curl measurements). Think of the best way to present your results. [One possibility: Let A be the server selected by Akamai, and B be the best server among other servers. For each MP and each other server B, compute the ratio of  $\text{metric}(A)/\text{metric}(B)$ . For RTT, the ratio below 1 indicates good server selection. Now, take an average of these ratios for a given MP (over all servers B) and plot a CDF graph of these average ratios. The CDF graph has the values of this ratio as the argument; for a given ratio value R, the value of the CDF function is the percentage of the MPs whose ratio is below R.] Hint: to force a download of an object from a particular CDN server, use its IP address instead of hostname in the object URL. For this to work, you need to specify an appropriate value of HTTP HOST header in the wget or curl request.
- **(EECS 425 only)** Apply some heuristics to divide all MPs into well-connected MPs (these are typically MPs running on PlanetLab nodes – a set of well-connected publicly available servers provided by universities around the world, which can be identified by 3-4 digit MPIDs) and residential MPs (typically connected via a DSL line; there will be a bunch of these running on your classmates’ laptops but there are also a small number elsewhere). Do your findings about the quality of Akamai’s server selection change if you consider these MP classes separately?
- **(EECS 425 only)** Pick a host that you can traceroute, e.g., [eecs.case.edu](http://eecs.case.edu). Discover the network paths leading from the MPs to this host. (Think of a heuristic you would use to deal with anomalies such as unresponsive hops in the traceroute, that is, those hops for which the traceroute outputs “\*”, or a situation where a hop is represented by a different router in some of the three probes.) Construct a multicast shortest path tree from the host to around 50 MPs using the network topology you discovered. While constructing the shortest path tree, you should run the Dijkstra’s algorithm and assume every link cost is 1. As a measurement result from this step, please report the total bandwidth consumed by multicasting a datagram over your distribution tree as opposed to unicasting the datagram separately to the 50 MP hosts from their corresponding CDN servers selected by Akamai for PC World. (What would be an appropriate metric for measuring the bandwidth consumption?). Note: we assume here that the multicast tree is constructed using a reverse shortest path technique since we are using traceroutes *to* rather than *from* the web site.

**Deliverables:** You should submit your work in electronic format. Your submission should include

- (1) All source code, including your source code for the Java programs implementing your measurements, the source code for post-processing the collected routes, the Java code for the multicast tree construction, and a README file to explain your implementation and the instructions to compile/run/test it.
- (2) The measurement data in the following form: (a) the MP IP address, hostname, country, state, and city (b) the CDN server selected for this MP by Akamai (c) the best server among non-selected servers and (d) the metric ratios according to the three metrics considered
- (3) **A report describing your results and the project management approach you used if you worked as a team. (Not applicable for individual projects)**

**Total points: 85 (EECS325)/120 (EECS425)**

**Hints and comments:**

- Please do not wait until the last minute. You will discover that conducting a realistic experiment takes a long time (may be days or weeks), may be frustrating, and does not always depend on you. So, try to at least progress to the point beyond external dependencies (collect all the measurement data).
- Because collecting measurements is a pain, you might consider splitting the entire experiment in steps and running steps separately. For example, the discovery step can be run as a separate experiment.
- There is a manual for the client library on the DipZoom web site. It is automatically generated and lists all methods of all classes including internal classes.