```
6.033 Spring 1999
Design project 1
Design space and design issues, version of 4/4/1999, 00:15.

The following notes compiled by Jerry Saltzer explore the design space
for the first 6.033 design project.  To a large extent they consist of
the union of most of the ideas mentioned in the papers for the two
sections that he graded, plus several other suggestions made by other
members of the teaching staff.  Note that not all of these ideas are
equally good.  Occasionally there is a comment about the value of an
idea, but for the most part those comments don't attempt anything
resembling a comprehensive evaluation.

1.  Definition of transparency
    -  User doesn't have to do anything special, but can detect
       that something is going on
    -  User can't tell he was redirected.  (Seems unnecessary)

2.  How dynamic does redirection need to be?
    -  once assigned a replica, may use it for multiple visits
    -  each time the home URL is issued, replica choice is reviewed
    -  for each use of any URL in the site, replica choice is reviewed
    This choice interacts with caching and with bookmark behavior and also
    with whether or not the server maintains changeable state.

3.  Design targets:  What scale does your design intend to encompass?
    Explain the design center, the design minimum, and the design
    maximum, particularly as to number of servers.

4.  Metric used to choose a server
    -  randomization (distributes load, ignores network service quality)
    -  geographical location (weak proxy for network proximity)
    -  network topology from static maps compared with client IP addr
    -  server properties
        - server up/down
        - server capacity
        - measured server response time to a small HTTP request
        - server port bandwidth
        - server port unused bandwidth
        - current server load
           -  thread queue length (current or averaged)
           -  cpu idle time (over how long?)
           -  number of open TCP connections (current or averaged?)
        - current server disk channel utilization (how measured?)
        - current server remaining capacity (how measured?)
    -  network properties measured at instant of connection request
        - minimum latency between client and server
        - latency as a proxy for congestion
          - ICMP ping latency from server to client
          - TCP probe latency from server to client
        - average of multiple probes from server to client
        - server to client hop count (traceroute, may be ponderous)
        - from conversation with a nearby router
           -  hop count (not a significant factor in performance)
```

```
           (e.g., Cisco Distributed Director uses BGP info)
           -  congestion/quality of service info
        - download rate of first HTTP connection
           - redirect only if below threshold.
           - first page from central server, each graphic on the
             page from a different server, after page is delivered
             the servers get together to decide who did best and
             central redirects the *next* request from that client.
    -  history of observed network properties to the region
       near the client
         -  region defined by network topology
         -  region defined by domain name
         -  region defined by IP address hierarchy
            (e.g., cache indexed by class-B or class-C subnet address)
         -  observed properties
            -  ICMP ping latency
            -  TCP probe latency
            -  packet loss rate
    -  political (border crossing rules, etc.)
    -  cost (some ISP's charge servers by gigabyte of network
       usage.)

5.  Level of centralization/decentralization
    -  all requests go to central site, which dispatches
    -  initial request goes to any server (e.g., with DNS round-robin),
       which asks all the others if they are closer.

6.  Method of assessing network latency caused by congestion
    -  ping by each replica server to client.  Requires high-level
       message from central server to each replica server, and
       decision of how long to wait for response.  (longest response
       time may be from the server closest to a distant client, but
       waiting for it delays everyone.)
    -  Use topology information to ask only promising servers to ping
       the client.  (avoids flooding client with pings.)
    -  ping by central site of each client, with loose source route
       via a router near each server (also ping server to subtract.  Not
       widely implemented, so it probably doesn't work in practice.)
    -  add a ping forwarding protocol to ACME servers: central site sends
       echo-him request to server, server sends echo request to client.
    -  ask a nearby router for RIP info

7.  How are multiple metrics combined
    -  linear weighting (how do you choose weights?  How do you linearly
       combine RTT in milliseconds and server load as a fraction?)
    -  assign first server that responds with load below a chosen
       threshold (e.g. 50%) and RTT below a chosen threshold (e.g. 50 ms.)
    -  use metrics in priority order, going lower only to break ties.
       e.g.,
        1/  server not overloaded
        2/  RTT (tie if within, e.g., 20%)
        3/  server connection speed
        4/  random or round robin

8.  When assessment leading to choice of server is made
    -  after request from client (adds delays)
       -  may post a temporary page saying, "one moment please, while
          we identify the best site to serve you" and playing the
          Acme theme song.
    -  while central site is handling the first several requests from
       the client.  (requires a cache of recently-assigned clients.)
```

- in advance, using a proxy for the client (what proxy?)
- load assessment can be done at a different time from network latency assessment. (e.g., immediately assign client to a lightly-loaded server, then reassign if another server later proves it has significantly less congestion on its path to the client.

9. Forwarding mechanism
   - DNS
     - standard DNS, client chooses from multiple name records (see client-side designs, below)
     - Acme DNS servers choose, set TTL to zero to avoid caching
       (N.B., works well for randomization, round-robin, or even server load-based distribution, but it is very difficult to base the decision on client latency...
       - the DNS server doesn't know who the real client is, because many DNS requests are forwarded by intermediate recursive DNS servers.
       - client timeouts may require fast action, and probably mean that there will be duplicate requests from the same client via different paths.
       - most companies don't run their own DNS servers.)
   - HTTP redirect
     - via host name (requires another DNS lookup)
     - put IP address in URL (avoids second DNS lookup)
     - permanent versus temporary redirect
       (affects bookmarks, partial URL's and perhaps POST)
   - DNS for first cut distribution not based on latency, use HTTP redirect for improvement if initial server latency is too high.
   - Base element redirect
     - Central site responds with initial page, but inserts a BASE element to do the redirection
     - all internal links are partial URL's
   - link insertion redirect: Central site responds with the initial page, but has dynamically constructed it so that all links--even the graphics on the initial page--are to a better replica.
   - IP Tunnel: Central site maintains an open TCP connection to every replica, and immediately tunnels the initial request to the preferred replica
     - replica thinks it is at the central IP address and tunnels its response back through the central site and thence to the client. All requests go through central site
     - replica tunnels initial response back (or tries to IP-spoof its way back directly.) The page returned by the replica contains links with absolute URL's that name the replica site, so future requests will go directly to the replica.

10. Client-side designs
    - what the upgraded client does
      - DNS sends list of servers, client chooses best.
      - Web server sends list of servers, client chooses best
      - client pings each server
      - client makes end-to-end request (e.g., HTTP HEAD) to include server load in RTT measurement.
      - send all probes in parallel, use first server that responds.
    - method of upgrading the client
      - initial server downloads a Java applet

- wait for next release of Internet Explorer and NetScape

11. In addition to the above, fairly central, issues, a good paper should give some answer to each of the more subtle questions in the design project handout:

- how hard is it to add a replica?
- what if everyone used your design?
- where is the bottleneck in your design?
- how do you decide when and where to add replicas?
- can anyone (e.g., AOL) create a replica?

12. Response to failure, e.g., central site crashes, replica crashes.

- (not evaluated, since we haven't come to this topic yet.)

13. How well does the proposed latency-measuring algorithm deal with the following case:

```
                                         x
   Acme central---------------------------x-----------server 1
             \                            x          /
              \---------------server 2----x--client--/
                                          x
                                         bad
                                      congestion
```

The challenging problem here is that server 1 will measure a short delay to the client, but its report may take a very long time to get back to Acme central--maybe long enough to exceed Acme Central's timeout. Server 2 will measure a long delay to the client, but its report will get back to Acme Central very quickly. A short timeout may result in server 2 getting the job, while A long timeout will slow things down for the case where the client is on the near side of the congestion. It appears that a better way to identify this case is for Acme Central to also separately ping the client, and use some small multiple of that RTT as its timeout. (Note that client-based models handle this problem very nicely, as does the first-server-below-threshold method.)

14. Some naming issues that come up, even though this topic arises after the design project was due.

- What is in mirror site hyperlinks that point within the site?
  - HREF = "http://www.acme.com/filename"
  - HREF = "http://mirror1.acme.com/filename"
  - HREF = "filename"
- How does the previous decision interact with decision to use
  - 301 permanently moved
  - 302 temporarily moved
- How do those two decisions interact with requirement to get user confirmation on forwarded POST operations?
- How do those two decisions interact with bookmarks and, more generally, the intended dynamics of redirection?

15. Subtleties

- Include a note and a link in the entity-body field of the initial response, for old browsers that don't know about HTTP redirection.
- central site should occasionally ping the servers and keep a running average of the response time, to use in determining the timeout when making future inquiries about load, etc.
- Decision of where more server capacity is needed is muddled by

the efforts of the system to distribute the load to the least-loaded servers.

16. Abstract of one good design; there are many others that are equally good, depending on the desiderata chosen:

Up to ten primary web servers have the same DNS name, with DNS round robin used to choose among them. When an HTTP request arrives at any server it pings the client and it sends a request to each of the other nine servers (each server must also have a distinct DNS name) asking them to ping the client and report back the RTT and their own processor utilization, averaged over the last 2 minutes. The first server that reports an RTT under 50 ms and an average load under 50% gets the job, with an HTTP 301 permanently moved response, using the server name. If no server is that good, redirect to the best available server, with (RTT/100 ms.) and (1/(1-load)) equally weighted. If more than ten servers are needed, additional servers are clustered around one of the primary servers and a local dispatcher distributes the load. Each primary web server maintains a cache of recently served clients, and if a request comes in that hits in the cache, it omits server polling and serves the client immediately unless its own utilization is running above a high threshold, say 95%. To keep clients returning to the designated site, all within-site URL's are partial rather than absolute.