CSE 452/CSE M552
Problem Set #1
Due: 5pm, May 7, 2013

1. In class, we described the Domain Name System (DNS) as using timeout-based cache validation, and replicated servers. Updates get applied to a primary server, which are then propagated in the background to the other replicas.

   a) Give an example to illustrate why DNS is not serializable.

   b) Briefly justify why the Domain Name System (DNS) is eventually consistent.

2. Facebook uses a three tier system for implementing its website. An array of front-end servers interacts with web clients (each client is hashed into exactly one front-end server); these front-end servers gather the information needed to render the client web page from an array of cache servers and a separate array of storage servers. Hashing is used to locate which cache and storage server might have a particular object (e.g., a friend list, or set of postings). The number of front-end servers, cache servers, and storage servers is not identical (the numbers are chosen to balance the workload), so in general, all front-ends talk to all cache servers and all storage servers.

   The cache servers (called memcache servers) are managed as a "lookaside" cache. When rendering an object on a page, the front-end first sends a message to the relevant memcache server; if the data is not available, the front-end (not the cache) then retrieves the data from the relevant storage server. The front-end then stores the fetched data into the memcache server. On update, the front-end invalidates the cached copy (if any) and updates the storage server.

   a) What semantics (serializable, eventual, inconsistent) would occur if the front-end first invalidates the cache, and then updates the storage server? Briefly explain.

   b) What semantics would occur if the front-end updates the storage server and then invalidates the cache? Briefly explain.

   c) What semantics would occur if the front-end invalidates the cache, updates the storage server and then re-invalidates the cache? Briefly explain.

   d) An employee at Facebook suggests adding a write-token to the memcache server. When a front-end wants to change a value, it sends a message to memcache to atomically invalidate the entry and set the write-token; subsequent accesses to the server stall. The front-end releases the write-token when the data is updated at the server, allowing stalled accesses to proceed. What semantics would occur in this algorithm? Briefly explain.