

CSE 552 HW1

Jeffrey Hightower

March 13, 2002

1 Problem 1: Clock Synchronization

Figure 1 shows a distribution of round-trip time delays between the local hosts `saba.cs.washington.edu` and `cedar.cs.washington.edu`. Figure 2 shows a similar distribution of the round trip times from `saba.cs.washington.edu` across the wide-area network to my home DSL gateway machine. The minimum RTTs for cedar and home are 323 μ sec and 68.4 msec respectively.

1.1 Clock Drift

Using the information available in the `/etc/adjtime` file, I calculated the hardware clock drift for these machines. Table 1 summarizes my results. Note that `/etc/adjtime` was unavailable for `saba` so this its values are estimated.

Using the equation $e_{min} = 3 * \rho * min$ from the paper it is possible to conclude that `saba.cs` could synchronize with `cedar.cs` to $3.566 * 10^{-7}$ seconds and with my home DSL gateway to $5.25 * 10^{-5}$ seconds.

1.2 GPS

Using GPS (other out of band broadcast synchronization such as LORAN navigation or radio station WWV) to synchronize clocks simply forces the clock drift ρ to a much smaller value then is possible with conventional independent hardware clocks. This decrease in ρ is due to two factors. First, each of the 27 GPS satellites contains four cesium/rubidium atomic clocks which are locally

Host Name	Clock Drift	
	sec/day (from <code>/etc/adjtime</code>)	parts per sec (ρ)
<code>saba.cs</code>	-22.1 (est)	$-2.558 * 10^{-4}$ (est)
<code>cedar.cs</code>	-31.8	$-3.680 * 10^{-4}$
home DSL	-12.4	$-1.435 * 10^{-4}$

Table 1: Hardware clock drift rates for the machines used in my round-trip timing tests.

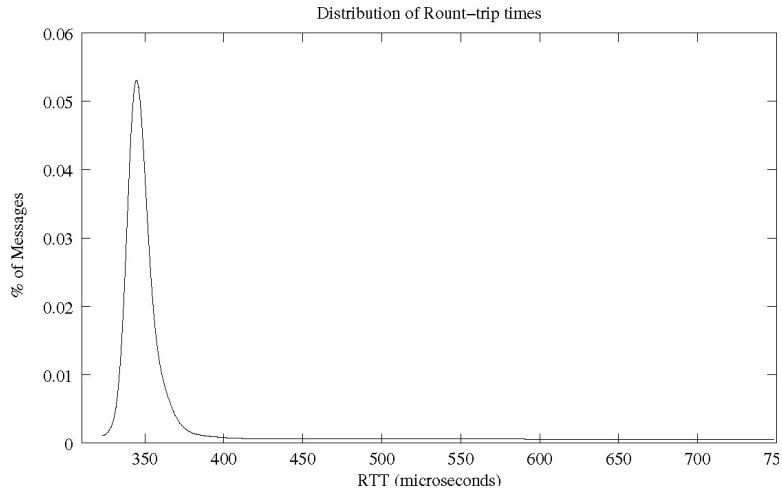


Figure 1: Distribution of round-trip times from `saba.cs.washington.edu` to `cedar.cs.washington.edu` for 1866 packets with no packet loss.

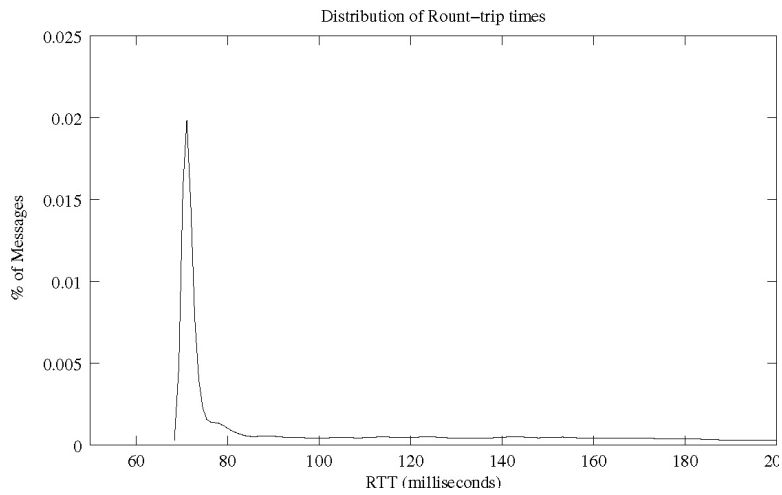


Figure 2: Distribution of round-trip times from `saba.cs.washington.edu` to my home DSL gateway machine for 3718 packets and a 3% packet loss.

averaged and synchronized daily to the more accurate atomic clocks at US Naval Observatory by US Air Force GPS ground control to maintain a time accuracy of 1 part in 10^{13} seconds – more accurate than most system hardware clocks [1]. Second, due to the out-of-band reception of timing data, the third ρ factor, relative drift between process P and Q 's clocks, is significantly reduced. However, coordinating distributed processes whose clock is set from GPS time still must contend with min , the minimum round trip delay across their unreliable communication channel. Therefore clock synchronization in distributed system coordination will continue to be an important issue.

2 Problem 2: Logical time stamps, distributed snapshots

2.1 Transitivity of Causal Concurrency

If \rightarrow is the “happens-before” relationship, let \rightleftharpoons be the causally concurrent relationship such that if $E \rightleftharpoons F$ then $E \not\rightarrow F$ and $F \not\rightarrow E$. Causal concurrency is not transitive as per this simple proof. Let E and G be events in the same process such that $E \rightarrow G$. Let F be an event in another process. No messages are exchanged. $E \rightleftharpoons F$ and $F \rightleftharpoons G$, but $E \not\rightleftharpoons G$ since $E \rightarrow G$ in the same process. QED.

2.2 Distributed Mail Snapshot

We assume that mail delivery operates according to the following four steps.

1. The PS assigns a unique message id m to the message.
2. The PS delivers all non-DL messages to the recipients by doing a request-reply message exchange with each user's mailbox server.
3. The PS expands all address in the message for DLs it manages and delivers all these to the appropriate servers also.
4. For other DLs, the message is sent along to the appropriate servers managing those DLs.

For simplicity we ignore the problem of eventually terminating delivery of recursive distribution lists. Recursion would need to be handled gracefully in a real mail system but for this problem, the delivery verification snapshot algorithm shown below will simply return NOT-DELIVERED. The verification algorithm is started when the user injects a single token T to any mail server q . T contains the ID of the message to verify delivery T_m and the address to send the result T_{addr} .

The requester of delivery verification must simply listen to T_{addr} and wait for responses from all mail servers. If any response comes back NOT-DELIVERED, then the message has not been placed in all destination mailboxes

Require: q receives a token T

- 1: **if** q has not recorded its state, $S_q = \emptyset$ **then**
- 2: **if** q is currently delivering the message with ID T_m **then**
- 3: $S_q \leftarrow$ NOT-DELIVERED
- 4: **else**
- 5: $S_q \leftarrow$ DELIVERED
- 6: **end if**
- 7: q sends T on all outgoing links
- 8: **else**
- 9: **if** q received delivery requests for message T_m since recording its state **then**
- 10: $S_q \leftarrow$ NOT-DELIVERED
- 11: **end if**
- 12: **end if**
- 13: **if** q has received T on all incoming channels **then**
- 14: send S_q to T_{addr}
- 15: **end if**

Algorithm 1: Mail delivery verification algorithm based on Chandy and Lamport

An alternative to distributed snapshot would be to build up a recursive callback verification tree for each message. For example, the posting server (PS) maintains state for all direct (non-DL) addresses in the message and marks them as delivered once each direct request succeeds. For each DL, the message is passed to a new server S which is then responsible for verifying delivery of direct addresses in that DL the same way the PS did for top-level direct addresses. Upon success, the recursion returns and S passes state back to PS that the DL is completely delivered.

Although this recursion algorithm has quite a bit of state and overhead, it is potentially “better” than distributed snapshot because it scales in the number of addresses (after complete DL expansion) in each message. Snapshot scales in the number of mail servers since verification tokens much touch every server in the network. On a network as large as the Internet where the number of mail servers far outsizes the average number of recipients per message, recursion would seem a better choice.

3 Problem 3: DNS spelunking

Table 2 summarizes the results for items (i), (ii), and (iii) of this problem. Recursive capability is determined by querying for a name for which that server is not authoritative and observing whether it returns the answer or refers the client to the root name servers. Negative caching capability is determined by sending the same bogus name multiple times in succession to a server. The bogus name must not be in the authoritative domain of the server.

Domain	Authoritative	Recursive	Neg. Caching
com	a.gtld-servers.net		
	b.gtld-servers.net		
	c.gtld-servers.net		
	d.gtld-servers.net		
	e.gtld-servers.net		
	f.gtld-servers.net		
	g.gtld-servers.net		
	h.gtld-servers.net		
	i.gtld-servers.net		
	j.gtld-servers.net		
	k.gtld-servers.net		
	l.gtld-servers.net		
	m.gtld-servers.net		
edu	a.root-servers.net		
	b.root-servers.net		
	c.root-servers.net		
	d.root-servers.net		
	e.root-servers.net		
	f.root-servers.net		
	g.root-servers.net		
	h.root-servers.net		
	i.root-servers.net		
ca	relay.cdnnet.ca		
	ns2.uunet.ca	✓	✓
	ns3.utoronto.ca	★	★
	rs0.netsol.com		
	merle.cira.ca	✓	✓
	clouso.risq.qc.ca	✓	✓
cs.washington.edu	lumpy.cs.washington.edu	✓	✓
	marge.cac.washington.edu		
	trout.cs.washington.edu	✓	✓
	ns.unet.umn.edu	✓	✓
	june.cs.washington.edu	✓	✓
	hanna.cac.washington.edu		
inktomi.com	ns1.inktomi.com	✓	✓
	ns2.inktomi.com	✓	✓
	ns3.inktomi.com	✓	✓
	ns4.inktomi.com	✓	✓

Table 2: Summary of DNS spelunking results. Entries marked with a ★ indicate that host was unreachable.

3.1 Negative Caching Performance (iv)

I can detect no noticeable performance difference for negatively cached local names in the `cs.washington.edu` domain. Indeed, intuitively there should be no more work to return a negative as opposed to a positive response when querying for a local name using a local name server that is authoritative for the domain

3.2 Nameserver Attacks (v)

An effective denial of service attack would be to flood requests to a recursively enabled name server (perhaps from distributed clients) in order to computationally overload the server. These requests should consist of randomly generated bogus names where the mid-level and top-level components are rotated through a long list of valid domains (e.g. `randombogus.yahoo.com`, `randombogus2.fbi.gov`, ...). Since it is not authoritative for any of these names, the server would perform full queries for each name and negative caching would not help stave off the attack.

3.3 TLD Download vs. Query (vi)

If the number of DNS query-response bytes per client per day is more than the size of all second-level NS records, then it would be ostensibly more efficient for each client to download a single glob every other day.

Based on the facts I have found there are around 35,000 new domain registrations of per day. The Internet Software Consortium's domain survey says there are 125,888,197 unique names and 37,502,747 `.com` names. If we assume the percentage of existing `.com` names corresponds with the same percentage of daily registrations, we can estimate that around 10,426 `.com` registrations occur per day. Most likely this number is slightly low because `.com` likely receive more registration activity than other TLDs.

There are 125,888,197 hosts and 2,851,938 second level domains according to ISC. Assuming 128 byte NS records, this means that each complete glob of second level NS records downloaded by clients every two days is 365,048,064 bytes (348MB) in size. To estimate DNS traffic we use the following data from Jung et. al.:

1. 4 packets ($4 * 512 = 2048$ bytes) for each lookup success
2. 48 packets ($48 * 512 = 24,576$ bytes) for each lookup failure
3. 1200 clients making 4,160,954 requests over 7 days *rightarrow* clients average 495 DNS requests per day.

We can use the following derivations to estimate the number of bytes of DNS traffic.

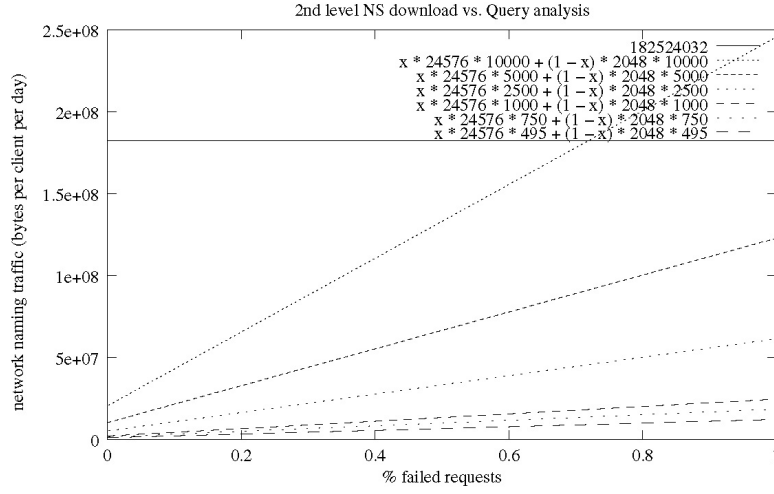


Figure 3: Effect of the percent of failed requests on the total number number of network bytes sent for naming traffic.

$$\frac{bytes}{day} = \rho \left(\frac{bytes}{req_{succ}} * \frac{req}{hosts} \right) + (1 - \rho) \left(\frac{bytes}{req_{fail}} * \frac{req}{hosts} \right) \quad (1)$$

Plotting these values on Figure 3, we can see that unless the number of requests per client per day is $\approx 10,000$ with a 70% failure rate, downloading a glob of all second-level names every two days is much less efficient. Furthermore, forcing all clients to download such a glob neglects to take into account the fact that not all clients have a sufficient bandwidth channel to make this second-level hoarding feasible.

References

- [1] Peter H. Dana. Global positioning system overview. Website, 2000. <http://www.colorado.edu/geography/gcraft/notes/gps/gps.html>.