15-446 Distributed Systems
Homework 1

Due: Beginning of Class, Feb, 12. 2009

February 11, 2009

A Ethernet exponential backoff

1. This problem illustrates possible danger of incorporating randomization in design.

   Let A and B be two stations attempting to transmit on an Ethernet. Each has a steady queue of frames ready to send; A's frames will be numbered A_1, A_2 and so on, and B's similarly. Let T = 51.2 us be the exponential backoff base unit. Suppose A and B simultaneously attempt to send frame 1, collide, and happen to choose backoff times of 0 x T and 1 x T, respectively. As a result, A transmits A_1 while B waits. At the end of this transmission, B will attempt to retransmit B_1 while A will attempt to transmit A_2. These first attempts will collide, but now A backs off for either 0 x T or 1 x T, while B bakcs off for time equal to one of 0 x T, ..., 3 X T.

   a) Give the probability that A wins this second backoff race immediately after his first collision.

   b) Suppose A wins this second backoff race. A transmits A_3 and when it is finished, A and B collide again as A tries to transmit A_4 and B tries once more to transmit B_1. Give the probability that A wins this third backoff race immediately after the first collision.

   c) Give a reasonable lower bound for the probability that A wins all the remaining backoff races.

   d) What then happens to the frame B_1?

   This scenario is known as the Ethernet capture effect.

B DNS

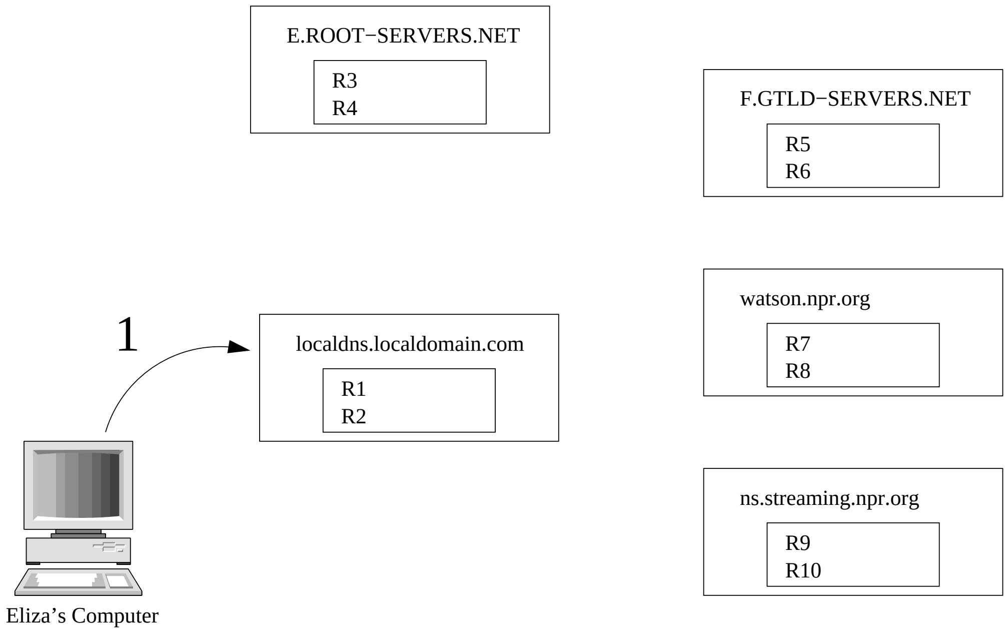
Elisa wants to listen to the National Public Radio news over the Internet. She starts her favorite audio player and points it to ra1.streaming.npr.org. The audio player calls gethostbyname() with the given name to obtain the IP address of the server. As a result of the gethostbyname() call, the local resolver in Elisa's machine contacts the local DNS server to translate the host name into an IP address. The local DNS server performs an iterative lookup. The table below contains the DNS distributed database. A row corresponds to a DNS record. The records are grouped by DNS server.

Record #	Name	TTL (sec)	IN	Type	Value
		`localdns.localdomain.com`			
R1	.	262542	IN	NS	`E.ROOT-SERVERS.NET.`
R2	`E.ROOT-SERVERS.NET.`	348942	IN	A	192.203.230.10
		`E.ROOT-SERVERS.NET`			
R3	org.	172800	IN	NS	`F.GTLD-SERVERS.NET`
R4	`F.GTLD-SERVERS.NET`	172800	IN	A	192.35.51.30
		`F.GTLD-SERVERS.NET`			
R5	`npr.org`	172800	IN	NS	`watson.npr.org.`
R6	`watson.npr.org.`	172800	IN	A	205.153.37.175
		`watson.npr.org`			
R7	`streaming.npr.org.`	172800	IN	NS	`ns.streaming.npr.org.`
R8	`ns.streaming.npr.org`	172800	IN	A	205.153.36.175
		`ns.streaming.npr.org`			
R9	`audio.streaming.npr.org.`	172800	IN	CNAME	`ra1.streaming.npr.org.`
R10	`ra1.streaming.npr.org.`	10	IN	A	205.153.36.175

2. In the figure below, draw arrows to indicate the sequence of queries and responses exchanged among the different machines. Label each arrow with a sequence number, and fill in the table below to indicate the following information:

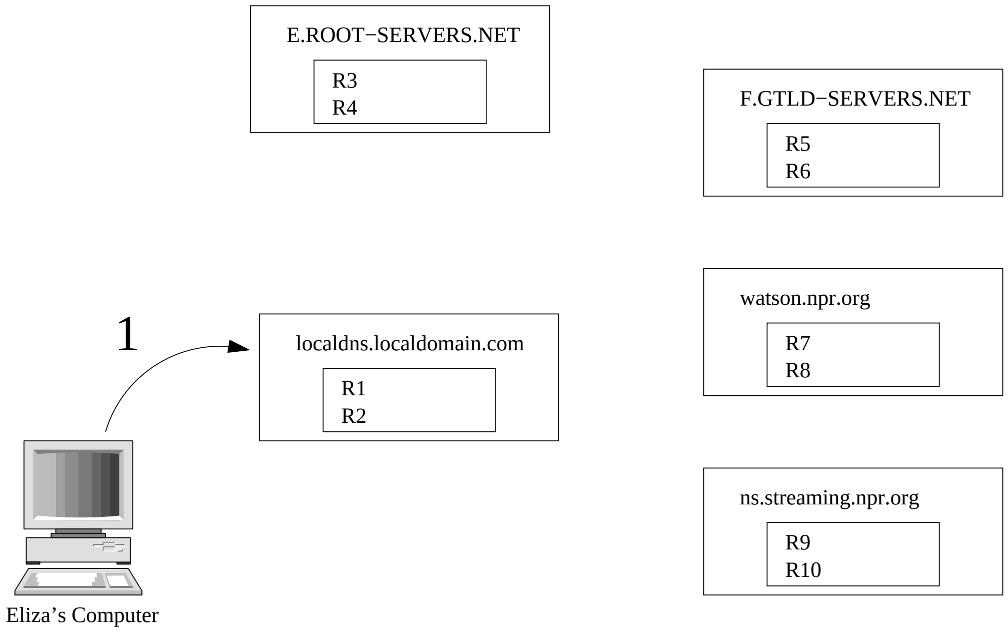
   - Sequence number indicating the ordering of the message exchanges.
   - Message Type: use Q for Query or R for Response.
   - Data: For queries use the value of the question data. For responses, specify the record ID(s) returned, if any, from the first column in Figure 1, e.g., R1, R2 ....
   - You may use abbreviations for host names, e.g. "ra1" rather than ra1.streaming.npr.org.

   The figure already contains an arrow indicating the first message from the local resolver to the local DNS server. The sequence number is 1 (first message), type = Q (query) and the data is the host name the application wants to resolve (ra1.streaming.npr.org). To make your sequence as simple as possible, assume the server includes both the A and NS records when applicable, so include both of them in the corresponding message.



Seq	Type	Data
1	Q	ra1.streaming.npr.org(A)

3. Eliza repeats her query two minutes later. Show what happens for this subsequent query.



Seq	Type	Data
1	Q	ra1.streaming.npr.org(A)

C RPC details

4. a) Below is the summarized steps of an RPC from a call to return. Order them in sequential order of execution.

   - The client stub builds a message and calls the local operating system.
   - The remote OS gives the message to the server stub.
   - The stub unpacks the result and returns to the client.
   - The server's OS sends the message to the client's OS.
   - The client's OS gives the message to the client stub.
   - The client procedure calls the client stub in the normal way.
   - The client's OS sens the message to the remote OS.
   - The server stub unpacks the parameters and calls the server.
   - The server does the work and returns the result to the stub.
   - The server stub packs it in a message and calls its local OS.

   b) Time outs in RPC.
   TCP uses timeout estimate (RTO) based on round-trip time. Can we use similair mechanism for RPC timeouts? Why or why not? Is there anything you would have to consider?

D Time synchronization with NTP

5. An NTP server B receives server A's message at 16:34:23.480 bearing a timestamp 16:34:13:430 and replies to it. A receives the message at 16:34:15:725, bearing B's timestamp 16:34:25.7. Estimate the offset between B and A and the accuracy of the estimate.

   [This is based on problem 10.7 from CDK, 3rd edition]

E TCP Timeout Estimation: Now and Then

6. Suppose that TCP is consistently observing RTTs of 1.0 second, with a mean deviation of 0.1 second. Suddenly, the RTT jumps to 5.0 seconds with no deviation. Compare the behavior of the original RTO estimator with the Jacobson/Karels algorithm:

   (a) How many timeouts are encountered with each algorithm?

   (b) What is the largest value of RTO calcuated? (How does the RTO change over time?)

   Assume that α, the srtt EWMA parameter, $= \frac{1}{8}$, and that γ, the sdev EWMA parameter, $= \frac{1}{4}$.

   [Hint]
   Recall that the original RTO estimator used an EWMA with constant 0.9:

$$\begin{aligned} \alpha &= 0.9 \\ R &= \alpha R + (1-\alpha)M \end{aligned}$$

   and computing the RTO as βR, with $\beta = 2$. R is Estimated RTT and M is the SampleRTT.

   For the Jacobson/Karels, we assume a gain of 0.875 for the RTT and 0.75 for the error term, as suggested in the paper.

EstimatedRTT = rgain * EstimatedRTT + (1-rgain) * SampleRTT
DEV = DEV * errgain + |SampleRTT-EstimatedRTT| * (1-errgain)
RTO = EstimatedRTT + 4* DEV