CS414 SP 2007 Assignment 5

Due April. 11 at 11:59pm Submit your assignment using CMS

1. Discuss whether clients in the following systems can obtain inconsistent or stale data from the file server and, if so, under what scenarios this could occur.
(a) AFS
(b) NFS

In AFS, writes to files are performed on local disks, and when the client closes the files, the writes are propagated to the server, which then issues callbacks on the various cached copies. During the time the file is written but not closed, the other clients could be accessing stale data. Also even after the file is closed, the other clients might access stale data if they had performed the open on the file before the updating client had closed it. It is only at the point of time when the next open is performed on a caching client that the server is contacted and the most recent data propagated from the server to the client. NFS uses a more ad-hoc consistency mechanism. Data is flushed from the clients to servers at periodic intervals and on file close operations. Also a client caching the file data checks for inconsistencies also at periodic intervals. Any updates made and flushed to the server during these intervals are not seen immediately on a client caching the file data.

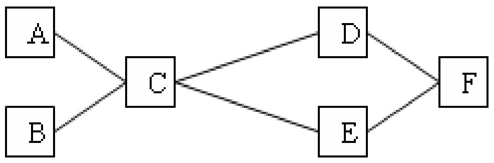
2.
(a) We described the algorithm for reliable flooding (used in Link-state routing) in class. Discuss what problem might raise if the field to store sequence number is small (say 6-bit). Do you think this is a concern in nowadays' Link-State routing? Why?

With a 6-bit field, sequence number can only go from 0 to 2^6-1=63. It means that router can only sent out 64 updates, then it will run out of available sequence number. This is known as the sequence number wrap-around issue. One solution is to use a circular sequence number space, that is, when reached 63, the router uses 0 as the next sequence number. In reality, the sequence number is 32 (64) bit long, wrap-around will not occur for many years. (that is, if the router is not malfunctioning)

(b) In IP, packets may be fragmented at an intermediate router, but are re-assembled only at the receiving end host. Why aren't packets re-assembled at the router adjacent to the fragmenting router?

First, different IP packets, to the same source, may take different paths. There may be no router that is in a position to re-assemble all fragments. Second, re-assembly requires are certain amount of processing and buffer space. This is often too much work for a router to do.

(c) Consider the following fish topology: Link A-C, C-D, D-F (all upper links) have bandwidth 1Gbps. Link B-C, C-E, E-F (all lower links) have bandwidth 100Mbps.



Suppose we know that A sends traffic to F at a rate of 1Gbps, and B sends traffic to F at a rate of 100Mbps. It would be nice to engineer the traffic so that A's traffic takes the path A-C-D-F, and B's traffic goes through B-C-E-F. This way we efficiently utilize all the links. Is this possible with IP routing? If so, describe how. If not, explain why.

In the strict sense, the answer is no. IP routing is destination-based (unlike virtual circuit). Flows from A to F and B to F all go through C. And for C, its routing algorithm will compute one route to F (even though 2 routes exist, the algorithm will pick one based on the metric), all the traffic arrive at C which going to F will be sent along this route, irrelevant with where the traffic came from. So it's not possible to let A's traffic go on one path, and B's go on the other one.

(Students who are familiar with networking might give the following two answer: both are correct
1. IP has an option called source routing. That is, the source, such as A or B, can specify the entire path for packet to traverse in the IP header. This way, A can specify its packet to go though A-C-D-F (similar for B). Though IP source routing is almost always disabled on the routers.
2. Equal-cost-multi-path (ECMP) is a comm practice to split traffic on multiple path if all paths have the same cost)

3. When DNS was originally designed, it was assumed that TTL (the cache lifetime) values would be on the order of many hours or even a few days. Later, companies like Akamai started setting the TTL values to very low values---a couple minutes or even less. (This is true only for the lowest level answers. In other words, for a DNS query on host10.akamai.com, the NS record for akamai.com would still have a high TTL, but the A record for host10.akamai.com would have a low TTL). At the time when this happened, some people thought that this would overload the DNS system. Considering the structure of DNS (hierarchy of servers, local DNS resolvers), do these small TTLs overload DNS? Why or why not?

Mostly no. Akamai's DNS servers will have a heavy load, but Akamai is free to deploy as many servers as it needs. This doesn't effect the rest of DNS. On the other hand, local resolvers will need to do more work because they will not cache answers as long. However, in practice this has not turned out to be a big problem.

4.
(a) TCP can perform poorly if there is a long end-to-end latency, and there are many losses in the network (say, 5%). If most of the losses happen at a certain link, and that link has a small latency, then performance gains can be had by implementing a retransmission scheme at the link layer. Would doing this violate the end-to-end principle? Why or why not?

If doing something in the "middle" improves performance, then it should definitely be done, and does not violate the end-to-end principle. The violation of end-to-end would happen if you decided to NOT do reliability at the ends because it is happening in the middle.

(b) TCP sends positive acknowledgments of received packets. Given that most packets arrive successfully, wouldn't it make more sense for TCP to send "negative acks" (that is, indicate which packets weren't received)? What is the problem with a NAK-based scheme.

The sender has to buffer packets until it knows that they have arrived at the receiver. If no packets were lost, the sender would have to buffer all packets.

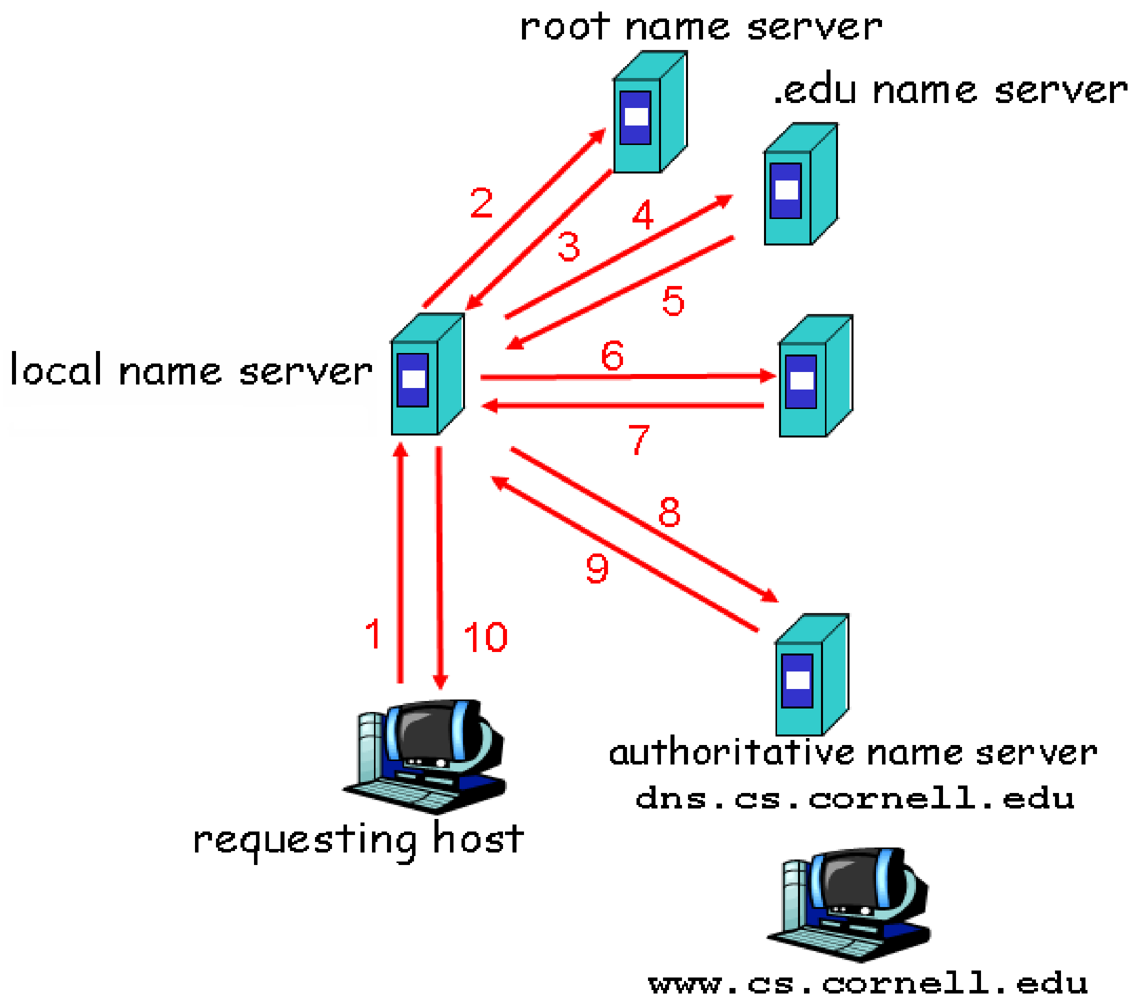
(c) In TCP slow start, at first the sending host will double the number of packets it sends every time it receives all acknowledgements for the previous packets it sent. In other words, the sending host will send one packet, receive the ACK for that packet, send two packets, receive the ACKs for those packets, send four packets, and so on. This exponential growth continues until there is a packet loss. (Subsequent to this loss, the sending host will increase the number of packets linearly, not exponentially.) Consider the scenario where a host H1 has two connections, one to a host HL on the same LAN, and one to a host HW thousands of miles away. The one-way latency to host HL is 1 microseconds, the one-way latency to host HW is 50 milliseconds and the round-trip latency is twice the one-way latency. Assuming no packet losses, and assuming near infinite bandwidth and processing speeds at all hosts, how long does it take to send the first 1023 packets to each host?

1023 packets can be sent in 10 packet-ACK "cycles". In the case of HL, each cycle takes 2 micro-seconds, so 1023 packets can be sent in 20 micro-seconds. In the case of HL, each cycle takes 100ms, so the total time is one second.

5. Lab questions, no unique answers
(a) The Unix utility **ping** can be used to find the RTT (Round-Trip-Time) to various Internet hosts. Read the man page for **ping**, and use it to find the RTT to www.google.com in California. Measure the RTT value at different times of day, and compare the results. What do you think accounts for the differences?

(b) The Unix utility **traceroute** can be used to find the sequence of routers through which a packet is routed. First, use this to find the path from your site to www.google.com. Paste the output below, and point out which of the routers are within your organization. Then try out a few more sites with **traceroute** and **ping**. How well does the number of hops correlate with the RTT times from **ping**?

(c) The Unix utility **nslookup** is a DNS-lookup tool. Read the man page for **nslookup**. Use **nslookup** to carry out manually a name lookup such as that in figure below, that is, carry out step 2 to 9. Note: you should do this in **nslookup** interactive mode, disable the recursive lookup feature, and specify that queries are for NS (Name Server) records rather than the usual A records. Paste your interactive **nslookup** session below (the command you typed and the output)


root name server
.edu name server
local name server
authoritative name server
dns.cs.cornell.edu
requesting host
www.cs.cornell.edu
1
2
3
4
5
6
7
8
9
10