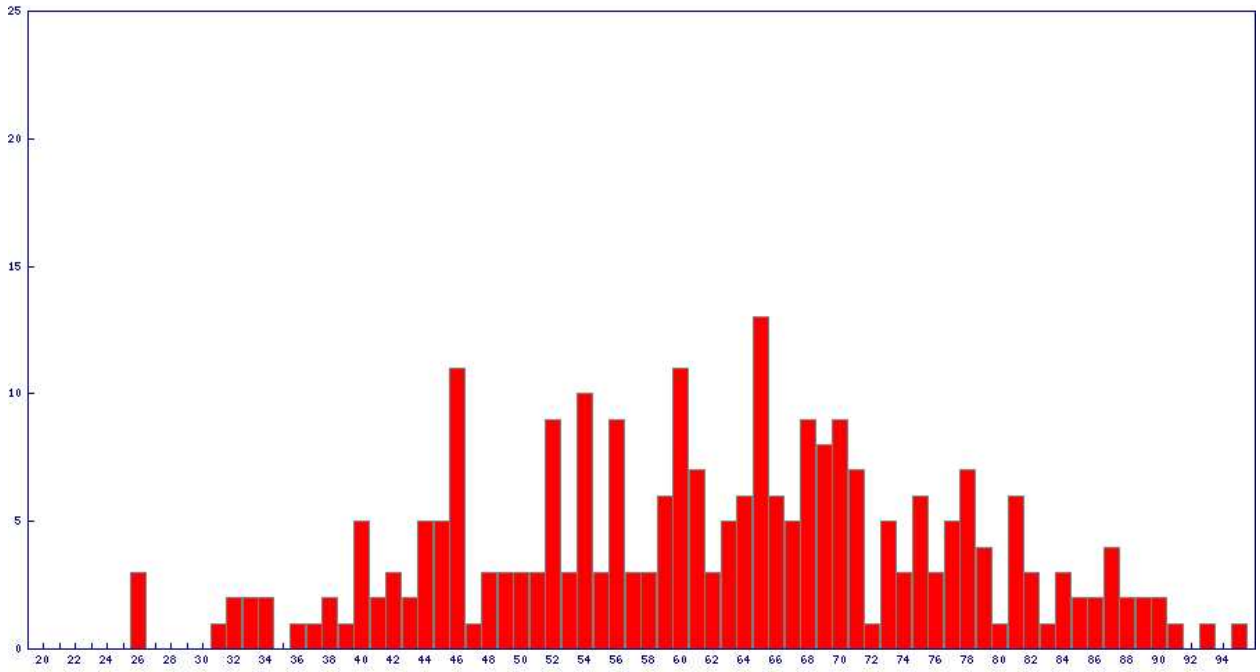
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2004

Quiz II Solutions



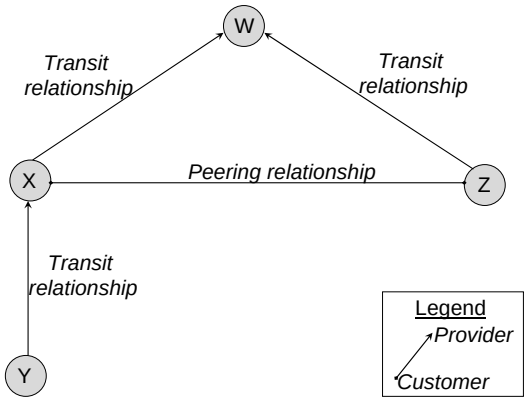
Average: 62
Standard deviation: 15

Name: Solutions

I Real-world systems

1. [8 points]: The figure below shows four interconnected autonomous systems (AS's) in the Internet, W, X, Y, and Z. For each AS i, let R_i denote the routes for destinations in AS i. If an AS i advertises a route for some destination to an AS j, then we say that "j hears the route from i." Each AS correctly implements the route advertisement and import rules described in the paper, "An Introduction to Wide-Area Internet Routing" (reading #9). Which of the following statements is true?

(Circle ALL that apply)



A. Z will hear R_Y from both X and W.

YES. *Z will hear R_Y from X because when X agreed to a peering relationship it agreed to advertise routes to its own customers over the peering link. Z will also hear R_Y from W, because Z is paying W to receive routes and X is paying W to advertise routes to customers of X.*

B. W will hear R_Y from both X and Z.

NO. *W will hear R_Y from X because X is paying W to advertise routes to customers of X. But W will not hear about R_Y from Z, because Y is not a customer of Z and Z doesn't advertise routes it learns from its peering relationships.*

C. When X gets a packet destined for Z, it will send the packet on the peering link if the packet's source is in X, but will not forward it on the peering link if the packet's source is in Y.

NO. *X will forward, via the peering link, all packets that are destined for Z, independent of the source of those packets. The reason it agreed to install the peering link was so that it could forward packets destined for Z directly, rather than paying W to forward them.*

D. If the $W - X$ link fails, then W will not be able to use the path through Z and X to reach Y.

YES. *Z doesn't advertise to W the route to Y via its peering link to X because Z has no economic incentive to do so. As a result, even if the $W - X$ link fails, W won't realize that the $W - Z - X$ route exists.*

2. [8 points]: Ben Bitdiddle implements a NAT box according to the description in "The IP Network Address Translator (NAT)." Whenever Ben's NAT box sees a packet from a local address, it replaces the source address with one of the box's available global addresses and records the local address to global address mapping. For some reason, Ben is unable to properly use certain TCP-based client applications from behind the NAT box. Which of these reasons is a good explanation for the problem?

(Circle ALL that apply)

A. The client application might be sending its IP address in the payload for the server to process.

YES. *The NAT box needs to translate all IP addresses belonging to hosts behind it, and the failing clients are said to be behind it. But the NAT box must know where to find those IP addresses in the packet. Addresses in the end-to-end payload are likely to slip through the NAT box unnoticed.*

B. The server application might be sending its IP address in the payload for the client to process.

NO. *Since the NAT box translates only IP addresses belonging to hosts behind it, and it is the client that is behind the NAT box, server IP addresses don't need translation. So there is no problem with carrying them in end-to-end payloads.*

C. Ben's client is trying to communicate with a server that is behind the same NAT box, and the NAT does not know how to forward those packets.

NO. *Assuming that Ben's client application is not configured to use a proxy server that is outside the private network, the path between a client and server that are both behind the same NAT does not require address translation. If the NAT for some reason is in that path (for example, it is also acting as an ordinary forwarder) it will not look in its translation table for addresses within the private network. And in its role as an ordinary forwarder it should have a route to the server.*

D. Ben has forgotten to modify the Ethernet CRC sequence in the NAT after adjusting the IP source address, so packets are being dropped by the switch at the other end of Ben's NAT box.

NO. *The Ethernet CRC is in the link-layer header. The link layer doesn't calculate the CRC until the NAT box calls on it to send the packet, so the CRC calculation automatically includes any address translations the NAT box did. At the other end of the link, the link layer may receive a packet with a bad CRC and discard it, but the reason for the bad CRC can't be that the NAT box translated some network-layer address after calculation of the CRC.*

3. [8 points]: Which of the following points follows from Ken Thompson's arguments in "Reflections on Trusting Trust" (reading #12)?

(Circle all that apply)

A. It is difficult to discover malicious code in programs, even if you have access to the source code to the entire system, including the compiler.

YES. *The idea that malicious code may not appear in the source of a compiled program, or even in the source of the compiler, is the primary message of Thompson's paper.*

B. If a C compiler written by someone trustworthy, and all the other system programs are compiled with that compiler, then there can be no malicious behavior.

NO. *Having a trustworthy compiler may provide an assurance that the compiler doesn't add malicious behavior to a program, but the program may itself already be malicious.*

C. The C programming language is an evil language because it makes it easy to write Trojan horses.

NO. *Thompson says nothing about which languages or language features make it easy or hard. And, it is easy to write Trojan horses in most programming languages.*

D. It is possible to write a compiler for the C programming language in C.

YES. *Thompson's Trojan horse example shows that it is so.*

E. Self-reproducing programs are more important and worthy of a Turing award than UNIX.

NO. *Thompson says, "Dance with the one who brought you," which suggests otherwise. But we don't know what he really thinks!*

4. [8 points]: Which of the following points follows from Ross Anderson's arguments in "Why Cyptosystems Fail" (reading #13)?

(Circle ALL that apply)

A. Cryptographic protocols are the weakest link in practical systems where security is important.

NO. *Anderson's examples of failures suggest strongly that the problem is neither in the cryptographic math nor in the cryptographic protocols, it is in the environment surrounding their application.*

B. Getting the cryptography right is important, but not sufficient to achieve security.

YES. *The main point of the paper is that successful attacks on systems that use cryptography are almost never direct attacks on the cryptographic algorithms and protocols, they nearly always exploit blunders in the way the cryptography is applied.*

C. Insider attacks are a significant security problem in many practical systems.

YES. *Anderson gives several examples of insider attacks.*

D. Secure systems should not be designed like safety critical systems.

YES AND NO. *Anderson identifies two design strategies for safety critical systems, one used in railroad signalling systems (design the engineer out, to minimize mistakes) and one used in air transport safety systems (design the pilot in, to fix things the designer didn't think of). He goes on to suggest that only the second approach should be considered for secure systems.*

Name: Solutions

II The OttoNet

Inspired by the recent political success of his Austrian compatriot, "Arnie," in Caleeforneea, Otto Pilot decides to emigrate to Boston. After several months, he finds the local accent impenetrable, and the local politics extremely murky, but what really irks him are the traffic nightmares and long driving delays in the area.

After some research, he concludes that the traffic problems can be alleviated if cars were able to discover up-to-date information about traffic conditions at any specified location, and use this information as input to software that can dynamically suggest good paths to use to go from one place to another. He jettisons his fledgling political career to start a company whose modest goal is to solve Boston's traffic problems.

After talking to car manufacturers, Otto determines the following:

1. All cars have an on-board computer on which he can install his software. All cars have a variety of sensors that can be processed in the car to provide traffic status, including current traffic speed, traffic density, evidence of accidents, construction delays, etc.

2. It is easy to equip a car with a Global Positioning System (GPS) receiver (in fact, an increasing number of cars already have one built-in). With GPS, software in the car can determine the car's location in a well-known coordinate system (assume that the location information is sufficiently precise for the purposes of this quiz).

3. Each car's computer can be networked using an inexpensive 10 Megabits/second radio. You may assume that each radio has a spherical range, R, of 250 meters; *i.e.*, assume that a radio transmission from a car has a non-zero probability of directly reaching any other car within 250 meters, and no chance of directly reaching any car outside that range.

Otto sets out to design the *OttoNet*, a network system to provide traffic status information to applications. OttoNet is an *ad hoc* wireless network formed by cars communicating with each other using cheap radios, cooperatively forwarding packets for one another.

Each car in OttoNet has a client application and a server application running on its computer. OttoNet provides two functions that run on every car, which the client and server applications can use (read these specifications carefully!):

1. QUERY(*location*): When the client application running on a car calls QUERY(*location*), OttoNet delivers a *query* packet to at least one car within distance R (the radio range) of the specified location, according to a best-effort contract. The query packet is 1,000 bits in size.

2. RESPOND(*status_info, query_pkt*): When the server application running on a car receives a query packet, it processes the query and calls RESPOND(*status_info, query_packet*). RESPOND() causes a *response* packet to be delivered to the client that performed the query, according to a best-effort contract. The response packet summarizes local traffic information (*status_info*) collected from the car's sensors and is 10,000 bits in size.

For both query and response packets, the cars will forward the packet cooperatively in best-effort fashion toward the desired destination location or car. Cars may move arbitrarily, alternating between motion and rest. The maximum speed of a car is 30 meters/second (108 kilometers/hour or 67.5 miles/hour).

Name: Solutions

5. [8 points]: Which of the following properties is true of the OttoNet, *as described thus far*?
(Circle ALL that apply)

A. Because the OttoNet is "best-effort," it will attempt to deliver query and response packets between client and server cars, but packets may be lost and may arrive out-of-order.

YES. *Follows from the definition of a "best-effort" network.*

B. Because the OttoNet is "best-effort," it will ensure that as long as there is some uncongested path between the client and server cars, query and response packets will be successfully delivered between them.

NO. *Even when there's no congestion, packets could be lost. For example, noise could corrupt packets, routes could fail, a car's computer could fail, etc.*

C. Because the OttoNet is "best-effort," it makes no guarantees on the delay encountered by a query or response packet before it reaches the intended destination.

YES. *A best-effort network makes no guarantees on packet delay.*

D. An OttoNet client may receive multiple responses to a query, even if no packet retransmissions occur in the system.

YES. *An OttoNet client makes a query to a location, and the query could be delivered to the server application of more than one car at that location. Each of those servers will send a response to the client.*

Otto develops the following packet format for OttoNet (all fields except `payload` are part of the packet header):

```
structure packet {
    GPS dst_loc;        // intended destination location
    int_128 dst_id;     // car's 128-bit unique ID picked at random
    GPS src_loc;        // location of car where packet originated
    int_128 src_id;     // unique ID of car where packet originated
    int hop_limit;      // number of hops remaining (initialized to 100)
    int type;           // query or response
    int size;           // size of packet
    char *payload;      // query request string or response status info
};

structure packet pkt; // pkt is an instance of ''structure packet''
```

Each car has a 128-bit unique ID, picked entirely at random. Each car's current location is given by its GPS coordinates. If the sender application does not know the intended receiver's unique ID, it sets the `dst_id` field to 0 (no valid car has an ID of 0).

The function FORWARD(`pkt`) runs in each car, and is called whenever a packet arrives from the network or when a packet needs to be sent by the application. FORWARD() maintains a table of the cars within radio range (R) and their locations (using broadcasts every second to determine the locations of neighboring cars), and implements the following steps:

F1. If the car's ID is `pkt.dst_id`, then deliver to application (using `pkt.type` to identify whether the packet should be delivered to the client or server application), and stop forwarding the packet.

F2. If the car is within R of `pkt.dst_loc` and `pkt.type` is "query", then deliver to server application, and forward to any one neighbor that is even closer to `dst_loc`.

F3. *Geographic forwarding step:* If neither F1 nor F2 is applicable, then among the cars that are closer to `pkt.dst_loc`, forward the packet to some car that is closer in distance to `pkt.dst_loc`. If no such car exists, drop the packet.

The OttoNet's QUERY(*location*) and RESPOND(*status_info, query_packet*) functions have the following pseudocode:

procedure QUERY$(location)$
$pkt.dst\_loc = location$;
$pkt.dst\_id = \mathbf{X}$; //see question **6.**
$pkt.src\_loc = my\_gps$;
$pkt.src\_id = my\_id$;
$pkt.payload =$ "What's the traffic status near you?";
$send(pkt)$

procedure RESPOND$(status_info, query_pkt)$
$pkt.dst\_loc = query\_pkt.src\_loc$;
$pkt.dst\_id = \mathbf{Y}$; //see question **6.**
$pkt.src\_loc = my\_gps$;
$pkt.src\_id = my\_id$;
$pkt.payload =$ "My traffic status is: " + $status\_info$ // "+" *concatenates strings*
$send(pkt)$;

6. [4 points]: Give suitable values for **X** and **Y** in the pseudo-code above, such that the pseudo-code conforms to the specification of QUERY() and RESPOND() given earlier.

(Fill in the blanks)

X = 0

X $= 0$ *is the best answer; when the client does not care or know which server to use, setting 0 in the* `dst_id` *field causes the packet to be delivered to all the cars at the location specified in* `dst_loc`*, in best-effort fashion.*

Y = `query_pkt.src_id`

Y $=$ `query_pkt.src_id` *is the right answer; the response should go only to the client application that initiated the query.*

Name: Solutions

7. [6 points]: What kinds of names are the ID and the GPS location used in the OttoNet packets?
(Circle ALL that apply)

A. The ID and GPS location are both pure names because either of them uniquely identifies a car.

NO. *The GPS location is not a pure name, it is an address. That's because it is overloaded with location information that helps the OttoNet find a suitable car to process a query. On the response packets, the GPS location helps locate a car with a specific ID in the OttoNet.*

B. On response packets, the destination car's ID (`pkt.dst_id`) is a pure name; the destination car's GPS location is an address.

YES. *The destination car's ID on a response packet is non-zero, and is not overloaded. It is a pure name. As explained earlier, the GPS location is an address.*

C. On response packets, the destination car's ID and its GPS location are both addresses.

NO. *The ID is not an address, it is a pure name.*

8. [8 points]: Otto outsources the implementation of the OttoNet according to these ideas and finds that there are times when a QUERY() gets no response packet, and times when a receiver receives packets that are corrupted. Which of the following mechanisms is an example of an application of an *end-to-end* technique to cope with these problems?

(Circle ALL that apply)

A. Upon not receiving a response for a QUERY(), retry the QUERY() from the client after a timeout.

YES. *Retrying a* QUERY() *after a timeout is an end-to-end loss recovery technique.*

B. If FORWARD() fails to deliver a packet because no neighboring car is closer to the destination, store the packet at that car and deliver it to a closer neighboring car a little while later.

NO. *Although this approach might improve performance and reduce the number of losses observed by a client or server, it is not an end-to-end technique.*

C. Implement a checksum in the client and server applications to verify if a packet has been corrupted.

YES. *An end-to-end checksum is a great example of an end-to-end error detection technique.*

D. Run distinct TCP connections between each pair of cars along the path between a client and server to ensure reliable end-to-end packet delivery.

NO. *Running a reliable transport protocol like TCP between cars is like running TCP connections between IP routers. There is no end-to-end mechanism present in such an approach!*

9. [8 points]: Suppose Otto decides to retry queries that don't receive a response. The speed of the radio in each car is 10 Megabits/second, and the response and request sizes are 10,000 bits and 1,000 bits respectively. The car's computer is involved in both processing the packet, which takes 0.1 μs per bit, and in transmitting it out on the radio (*i.e.,* there's no pipelining of packet processing and transmission). Assume that each car's radio can transmit and receive packets at the same time.

Suppose that the maximum queue size is 4 packets in each car, the maximum radio range for a single hop is 250 meters, and that the maximum possible number of hops in OttoNet is 100. Ignore media access protocol delays. Assume that the server application takes negligible time to process a request and generate a response to be sent.

What is the smallest "safe" timeout setting that ensures that the retry of a query will happen *only* when the original query or response packet is guaranteed not to still be in transit in the network? *Answer this question in the space provided below, showing your work. Be neat and legible!*

SOLUTION: *Any timeout setting that is larger than the largest possible round-trip time between client and server is "safe." The round-trip time in the OttoNet is the sum of the per-hop transmission and processing times, the per-hop queueing delays, and the speed-of-light propagation time of the request and response packets.*

For each request, the total transmission + processing time is

$$100 \times (1000/10 \cdot 10^6) + 10^{-7} \cdot 10^3 = 20 \text{ milliseconds}$$

For each response, the total transmission + processing time is

$$100 \times (10000/10 \cdot 10^6) + 10^{-7} \cdot 10^4 = 200 \text{ milliseconds}$$

For a request or response packet to be allowed to enter a queue whose maximum size is four, there can be no more than three packets already in the queue. The worst-case queueing delay is when every queue has three unprocessed packets in it at the instant that the packet whose delay is being estimated arrives, and they are all response packets. Because no car pipelines transmission and processing, the worst-case queueing delay at each hop is $3\times$ *the processing + transmission delay of a response packet, or* 6 *milliseconds. The total round-trip queueing delay is therefore equal to* $2 \cdot 100 \cdot 6$ *milliseconds =* 1200 *milliseconds.*

Compared to these numbers, the propagation time is negligible, because the maximum distance is $2 \times 100 \times 250 = 50000$ *meters, and radio (light) travels such distances in less than 1 millisecond.*

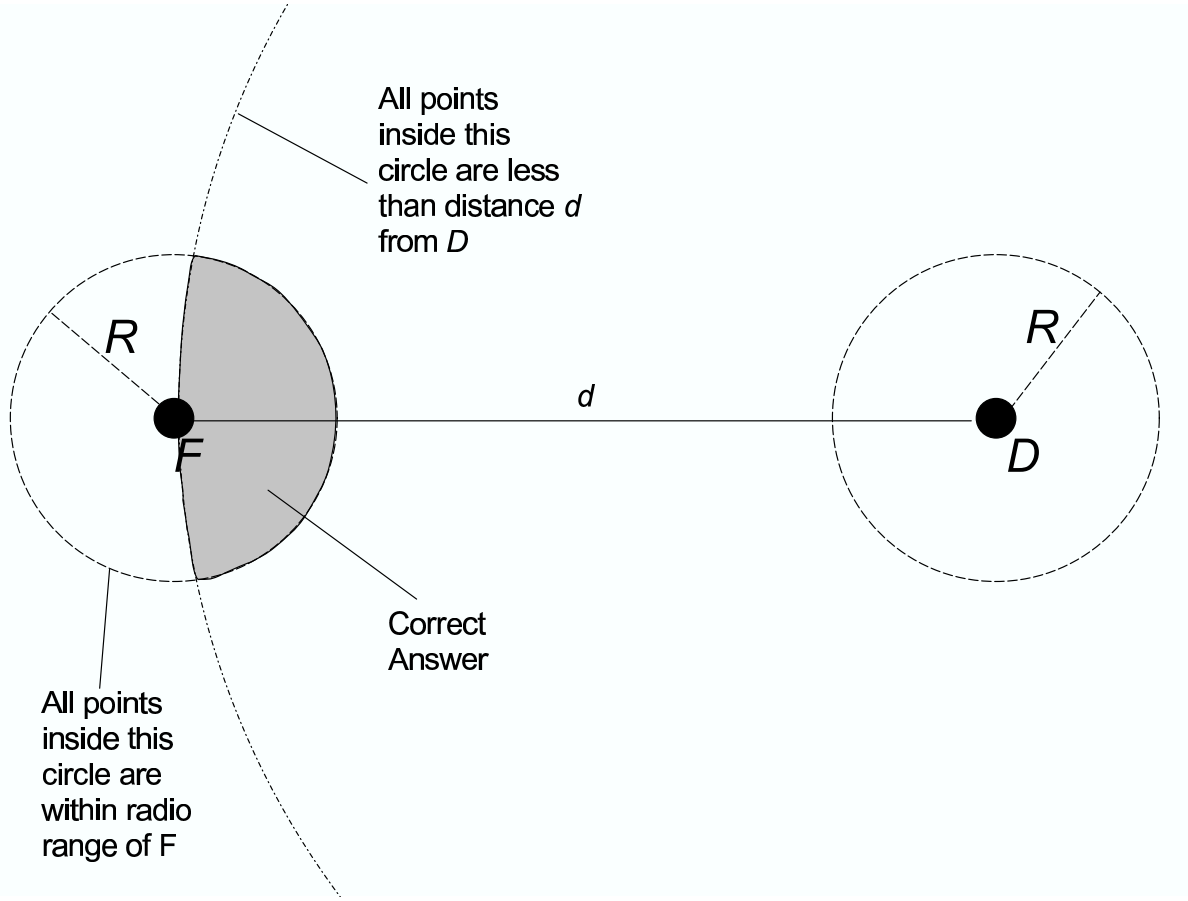
Hence, the maximum possible round-trip time is $20 + 200 + 1200$ *ms =* 1420 *milliseconds =* 1.42 *seconds.*

Name: Solutions

Otto now proceeds to investigate why FORWARD() sometimes has to drop a packet between a client and server, even though it appears that there is a sequence of nodes forming a path between them. The problem is that geographic forwarding does not always work, in that a car may have to drop a packet (rule F3) even though there is some path to the destination present in the network.

10. [8 points]: In the figure below, suppose the car at F is successfully able to forward a packet destined to location D using rule F3 via some neighbor, N. Assuming that neither F or N has moved, clearly mark the region in the figure where N must be located. (Assume a two-dimensional network.)

To be able to forward a packet using rule F3, two conditions must be true: first, the next hop, N, must be closer to the destination, D, than the source, F; second, N must be within F's communication radius. To identify points that are closer to D than F, let's label the distance between D and F as "d". Clearly, any point that is less than distance d from F will be within a circle of radius d centered at F. If we take the intersection of this circle with F's communication radius, we get the possible locations where N can be.



11. [8 points]: Otto decides to modify the client software to make "pipelined" QUERY() calls in quick succession, sending a query before it gets a response to an earlier one. The client now needs to match each response it receives with the corresponding query. Which of these statements is correct?

(Circle ALL that apply)

A. As long as no two pipelined queries are addressed to the same destination location (the `dst_loc` field in the OttoNet header), the client can correctly identify the specific query that caused any given response it receives.

NO. *The response to a query does not include the* `dst_loc` *that the sender used, so the* `dst_loc` *in the query packet cannot be used to differentiate replies. The* `src_loc` *in the response packet cannot be used either, since the car which responds to a query is not guaranteed to be at the* `dst_loc` *of the query packet. The client can't simply match a the* `reply.src_loc` *to the nearest* `query.dst_loc` *because the same server may reply to two different queries addressed to two different locations (if, for example, that server is the nearest car to both destination locations).*

B. Suppose the OttoNet packet header includes a nonce set by the client, and the server acknowledges the nonce in its response, and the client maintains state to match nonces to queries. This approach can always correctly match a response to a query, including when two pipelined queries are sent to the same destination location.

YES. *A nonce is a unique identifier that the client can use to differentiate responses.*

C. *Both* the client and the server need to set nonces that the other side acknowledges (*i.e.,* both sides need to implement the mechanism in choice B above), to ensure that a response can always be correctly matched to the corresponding query.

NO. *With choice B the client can already match responses to queries, so there is no need for additional mechanism such as having the server add a nonce to the reply.*

D. None of the above.

NO, *because C is correct.*

Name: Solutions

12. [8 points]: After running the OttoNet for a few days, Otto notices that network congestion occasionally causes a congestion collapse because too many packets are sent into the network, only to be dropped before reaching the eventual destination. These packets consume valuable resources. Which of the following techniques is likely to reduce the likelihood of a congestion collapse?

(Circle ALL that apply)

A. Increase the size of the queue in each car from 4 packets to 8 packets.

NO. *There are two possibilities for the timeout value. First, suppose that Ben used the answer to question 9 to set the timeout. Given a fixed timeout, lengthening queues would increase, not decrease, the chance of congestion collapse. The longer queues may cause clients to time out and resend their request packets, even though a response may already be on its way back.*

Second, suppose that Ben adjusted the timeout for the longer queues. Doubling queue lengths certainly doesn't prevent congestion collapse, because congestion collapse can occur with queues of any length. There is no a priori reason to believe that it is less likely with 8-packet queues than with 4-packet queues. Increasing the size of the queue to 8 packets might have a positive effect: some packets that would otherwise have been dropped might eventually reach their destination. However, it might also have a negative effect: packets that would otherwise have been dropped remain in the system and may cause congestion elsewhere.

B. Use exponential backoff in the timeout mechanism while retrying queries.

YES. *Exponential backoff reduces the injection rate of packets to a level that the network can tolerate.*

C. If a query is not answered within a timeout interval, multiplicatively reduce the maximum rate at which the client application sends OttoNet query packets.

YES. *If this question had said "current" rather than "maximum" rate, it would have exactly been exponential backoff. Reducing the maximum rate eventually produces the same end result.*

D. Use a flow control window at each receiver to prevent buffer overruns.

NO. *Flow control windows apply to streams of data. OttoNet requests are not streams, they are independent packets, each one of which may be delivered to a different server, so a flow control window is not applicable. Moreover, flow control is an end-to-end mechanism to ensure that a slow receiver's buffers don't get overwritten by a fast sender. But the problem states that the server and client processing are both infinitely fast, so adding flow control would not accomplish anything.*

13. [10 points]: The OttoNet is not a secure system. Otto has an idea—he observes that the 128-bit unique ID of a car can be set to be the public key of the car! He proposes the following protocol. On a query packet, sign the packet with the client car's private key. On a response packet, seal the packet with the client car's public key (that public key is in the query packet). To allow the response packets to be forwarded through the network, the server does not seal the destination location and ID fields of those packets. Assume that each car's private key is not compromised.

Which of these statements is true of this scheme?

(Circle ALL that apply)

A. A car that just forwards a query packet can read that packet's payload and verify it.

YES. *Since the packet is not sealed, anyone who handles the packet can read the payload. Since the packet is signed, and the corresponding public key (the ID of the client car) is inside the signed packet, anyone who handles the packet can verify that the source is as claimed and that the payload hasn't been modified since it left the source.*

B. The only car in the network that can unseal the response packet from a server is the car specified in the destination field.

NO. *The destination fields are not signed. A malicious node that is forwarding the packet could modify the destination field, or could modify the payload (say, by substituting one that is sealed to someone else).*

C. The client cannot always verify the message integrity of a response packet, even though it is sealed.

YES. *A malicious node in the network can fabricate and seal an arbitrary payload with the client car's public key (the destination), and use the result to replace the original payload.*

D. If every server at some queried location is honest and not compromised, the client can be sure that a sealed response it receives for a query actually contains the correct traffic status information.

NO. *See the answer to 13C. In addition, malicious nodes can also perform replay attacks or change the unsealed destination location and ID fields.*

E. None of the above.

NO, *because some of the above answers were true.*

End of Quiz II

Name: Solutions