DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

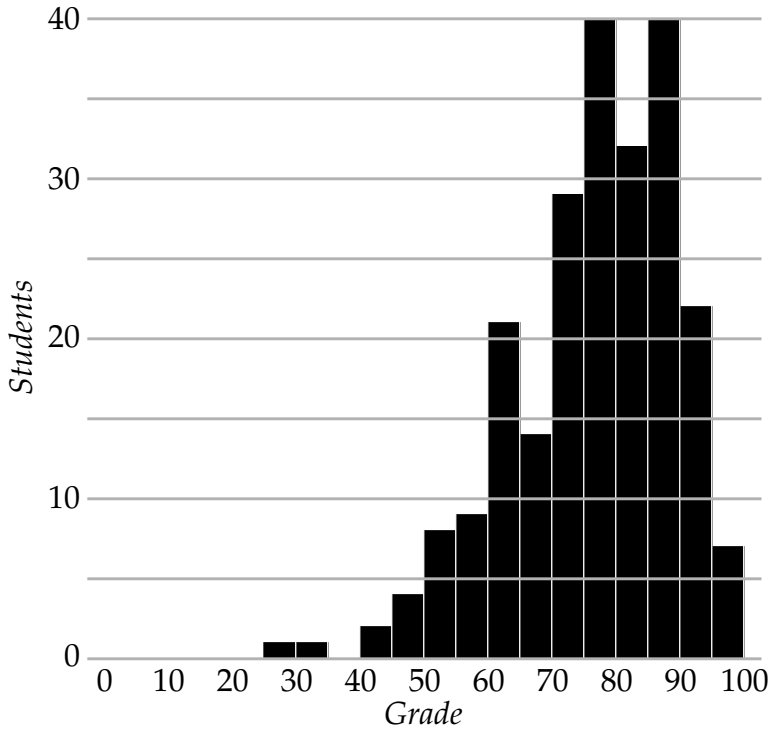
6.033—Computer System Engineering Handout 28—April 23, 1996

Quiz 2 Solutions

These solutions contain four problems; we have written our answers in the space provided.

One note: Our solutions are **not brief.** We have aimed for clarity and completeness (listing almost every possible answer) rather than brevity. Your answers may have been, and should have been, much shorter. We expect only one answer from you, while you should expect all the correct answers from us.

Grade distribution



Name: *Mrs. B. O. T. Envelope-Calculation*

1. Ben Bitdiddle has been thinking about RPC. He remembers that one of the problems with RPC is the difficulty of passing pointers: since pointers are really just addresses, if the server dereferences a client pointer, it'll get some value from *its* address space, not what the client intended. Ben hopes to get some insight by applying his new-found 6.033 naming knowledge to the problem.

1a. *(15 points)* Match up each concept on the left with the corresponding naming concept on the right. You won't use some of the naming concepts.

Here is one consistent interpretation:

_(6)_ address space	(1) implicit context
_(7)_ pointer	(2) object
_(2)_ memory value	(3) multiple contexts
_(4)_ pair of address space and pointer	(4) closure
_(1)_ dereferencing a pointer in the current address space	(5) search path
	(6) context
	(7) name

Alternatively, an address space may be interpreted as a closure. To see this, consider a pointer as a name, and the page table structure as a context. Then the address space may act as a mechanism binding names to their context: a closure.

Inspired by his naming insights, Ben decides to pass closures instead of bare pointers in his RPC system. Louis Reasoner, excited by Ben's discovery, decides to change *all* end-to-end protocols used on MIT's networks along the same lines.

1b. *(5 points)* Argue for or against Louis's changes, using 6.033 arguments.

We dislike Louis's changes. Not all applications need to pass pointers, so requiring all end-to-end protocols to support this service will incur needless performance penalties. Instead, only those applications that require safe pointer passing should provide it as part of their end-to-end protocol. This is an example of an end-to-end argument.

2. In a mobile computing environment, locating a user in the face of user movement is an important problem. Consider a system in which each user is assigned a permanent home location, which can be found simply by looking the user up in a name service.

   Alyssa P. Protocol-Hacker is designing an end-to-end protocol for locating these mobile hosts. Her first protocol is very simple: every time a user moves, store a forwarding pointer at the old host, pointing to the new host. This creates a chain of forwarding pointers with the permanent home location at the beginning and the mobile host at the end. Messages meant for the mobile host are sent to the home location, which forwards them along the chain until they reach the mobile host itself. (The chain is truncated when a mobile host returns to a previously visited location.)

   Alyssa notices that performance generally gets worse each time she moves her mobile host! "This must be because of the long chains of forwarding pointers," she muses.

   Help Alyssa design a more efficient protocol which avoids long chains. Assume that the end-to-end protocol you design is built on top of a network protocol which can reorder messages, but doesn't lose or duplicate them.

2a. *(8 points)* Alyssa's first try works like this. Each time a mobile host moves, it sends a message to its home location indicating its new location. The home location maintains a pointer to the new location. With this protocol, there are no chains at all. Nodes other than the home node do not maintain forwarding information.

   When this protocol is implemented, Alyssa notices that messages regularly get lost when she moves from one location to another. Explain briefly why or give an example.

   If a message is in transit to mobile host A at the time when A moves from location X to location Y, that message will arrive at X and find that A has left. Since there is no information at X indicating where A went, the message will be lost. Similarly, any message that the home node forwards after A has moved but before A's rerouting message gets to home will go to location X and be lost. All messages that home forwards to A from about one network transit time before the move until about one network transit time after the move will probably be lost.

   A more disastrous scenario, though less likely and therefore probably not the cause of messages "regularly" getting lost, happens when a mobile host A moves from X to Y, and then on to a new location Z. The rerouting message that A sends from Y may be delayed in transit long enough that the rerouting message it sends from Z gets home first. In that case, when the rerouting message sent from Y finally arrives, home will forward all future messages to the wrong place; A will be completely cut off, at least until its next move. The briefer the visit of A to Y, the greater the chance of this scenario happenening.

2b. *(8 points)* Alyssa is disappointed with her first attempt, and decides to start from scratch. In her new scheme, no forwarding pointers are maintained anywhere, not even at the home node. Say a message destined for a mobile host A arrives at a node N. If N can directly communicate with A, then N sends the message to A, and we're done. Otherwise, N broadcasts a search request for A to all the other fixed nodes in the network. If A is near a different fixed node N', then N' responds to the search request. On receiving this response, N forwards the message for A to N'.

Will messages get lost with this protocol, even if A moves before the message gets to N'? Briefly explain your answer.

No and Yes *(a typical 6.033 answer!).*

No, messages will not get lost: *Suppose mobile host A moves away from N' just after N' responds to the search request N sent. N will forward the original message to N', but when it arrives, N' will follow the standard procedure: If it is in contact with A it will pass the message along to A; however, since it is not, it* broadcasts another search request. *Assuming A is always in contact with some node, the original message will thus follow A around. If A ever rests long enough for the message to catch up with it, it will then be delivered.*

Yes, messages will get lost: *Search broadcasts have a transit time, and any given broadcast doesn't get to all nodes at exactly the same time. Suppose node N broadcasts a search request, and while the broadcast is in transit, A moves from node X to node Y. The following time sequence creates a problem:*

1. *N broadcasts "Where is A?";*
2. *Y receives broadcast, ignores it becase A is at X;*
3. *A moves from X to Y;*
4. *X receives broadcast, ignores it because A is now at Y;*
5. *N concludes from the lack of response that A has retired from active service and trashes the original message.*

And the following briefer example of the above scenario, in which Y is actually N, can lose messages even if broadcasts are delivered everywhere simultaneously. (We assume that mobile hosts don't actively notify locations when they arrive; instead, the locations try to locate them when messages for them are received.)

1. *N broadcasts "Where is A?";*
2. *A moves from X to N;*
3. *X receives broadcast, ignores it because A is now at N;*
4. *N concludes from the lack of response that A has retired from active service and trashes the original message.*

2c. *(8 points)* Unfortunately the network doesn't support broadcast messages efficiently, so Alyssa goes back to the keyboard and tries again. Her third protocol works like this. Each time a mobile host moves, say from N to N', a forwarding pointer is stored at N pointing to N'. Every so often, the mobile host sends a message to its permanent home node with its current location. Then, the home node propagates a message down the forwarding chain, asking the intermediate nodes to delete their forwarding state. Can Alyssa ever lose messages with this protocol? Explain briefly. (Hint: think about the properties of the underlying network.)

There are lots of ways to lose messages with this protocol. Here are three of them (numbered for future reference).

P1. Since the underlying network can reorder messages, the "delete forwarding pointers" message may overtake an ordinary message already in transit along the forwarding chain. Then, when the ordinary message gets to its next forwarding point it will find that the forwarding information has already been deleted.

P2. If mobile host A creates a long forwarding chain, sends an "I'm over here at node X" message home, then immediately moves to a location Y that is earlier in the chain and then on to somewhere else, the "delete forwarding pointers" message will come along and delete the current pointer in Y, leaving A unable to receive anything.

P3. A version of the disastrous scenario of part 2a also applies. A moves from X to Y, sends a message home, then moves to Z and sends another message home. The second message home gets there first, causing the chain out to Y (including the forwarding pointer from Y to Z) to be deleted. Then the first message gets home, telling home to send everything to Y.

2d. *(8 points)* What additional steps can the home node take to ensure that the scheme in question c never loses messages? Describe briefly.

Here are a number of different solutions.

S1. Since the latest "I'm here" message from mobile host A to its home short-circuits any forwarding pointers in other nodes, the home node could just omit sending the message that deletes old forwarding pointers. That would solve all three problems of part 2c, which is good enough. However, it would leave dead forwarding information lying around, using up space, with no way to know when it can be reclaimed (call that problem P4).

S2. If the home node knew the maximum transit time of the network, it could delay the sending of the "delete forwarding pointers" message for that length of time. This approach would take care of problem P1 and problem P4 and, with a clever home node, might help reduce the likelihood of problem P3. Unfortunately, it will increase the likelihood of problem P2.

S3. The home node could include sequence numbers in all forwarded packets. Then, when it gets a rerouting message from a mobile host, it can immediately begin forwarding new packets directly to the new location, and it can also send a message to the mobile host telling it the sequence number of the last packet sent via the old route, and asking for an

explicit acknowledge when all packets up to the sequence number have arrived. (Note that a timestamp isn't good enough here; we need a way of being sure that ***all*** *intervening packets have been delivered.) When that acknowledgement returns to home, send the "delete forwarding pointers" message down the old chain. Problems P1 and P4 are solved. Problems P2 and P3 persist.*

S4. Instead of sending a "delete forwarding pointers" message, the home node could send a "change forwarding pointers" message, alerting everyone along the old forwarding chain to adjust their forwarding pointer to A's new location. That would eliminate problems P1 and P2, but problems P3 and P4 persist.

S5. The home node could include its own address in every forwarded message, and all other forwarders could follow a new rule: if a message is undeliverable as originally addressed, send it back to the home node. The home node can then rescue and reroute any messages that would have been lost under the other methods. In addition, when it gets a rerouting message from a mobile node, it can immediately send a "delete forwarding information" request down the old route, confident that any dallying packets will be returned to it for rerouting. This approach takes care of problems P1 and P4, and some instances of P2. It also means that when problem P3 (or the bad instances of P2) arise, any packets that the home tries to forward to the mobile will loop indefinitely between home and the last place home thought the mobile was connected; if the mobile ever checks in again, these packets will eventually be delivered. Whether this is a satisfactory state of affairs is debatable.

S6. The mobile host could count the number of rerouting messages it sends home, sending a sequence number with each message. These sequence numbers impose an order on rerouting messages, so problem P3 is solved. The last sequence number the mobile sent is attached to each forwarding pointer it creates. When the home node receives a rerouting message with sequence number S, where S is greater than the last sequence number home saw from that mobile host, it sends out a message "delete forwarding information with sequence number smaller than S." This solves problem P2. Combine this method with solution S5 (undeliverable messages are sent back to the home node) and we have solved all four problems, P1 through P4!

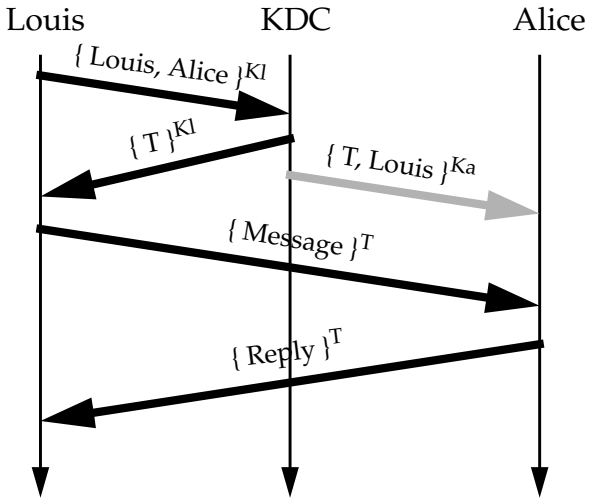
Note that none of the suggested solutions involve adding an acknowledge/timeout/ retransmit scheme to the forwarders. That approach would have the side effect of producing duplicates in a network that is certified never to produce duplicates, so it would require modifying the end-to-end protocol implementations to suppress these duplicates.

3. Louis B. Unreasonable has decided that the NSA has gotten their hands on PGP and probably made it weak or insecure. He therefore wants to write his own safer and more secure encryption system. Before doing so, he first wants to understand the most popular schemes of today, RSA and DES. RSA is a public-key system where encryption and decryption each take 1,000μs per block (a block is 0.1Kbyte or less). DES is a private-key system with a single key; encrypting or decrypting with DES only costs 10μs per block. (Only consider passive attackers for this problem.)

3a. *(6 points)* How long does it take to encrypt a file that is 1,000Kbytes in size, assuming you are using RSA? How long does it take to encrypt that same file using DES?

RSA: 1,000Kbytes / 0.1Kbytes/block × 1,000μs/block = 10 seconds.
DES: 1,000Kbytes / 0.1Kbytes/block × 10μs/block = 0.1 seconds.

3b. *(6 points)* Louis was planning on using public-key encryption, but the costs you calculated for him have changed his mind. Now, his system will have central key distribution centers (KDCs) using only private-key encryption. Here's how the protocol works when Louis sends a message to Alice. (Louis's private key is Kl; Alice's private key is Ka; and the temporary private key the KDC sets up for them is T.)



How long will it take for this exchange to complete, that is, from the time Louis starts encrypting the first message to the KDC to the time Louis is done decrypting Alice's reply? *Only consider encryption times, decryption times, and network latency.* Assume the network latency is 2,000μs per message. Assume also that each message fits in one block.

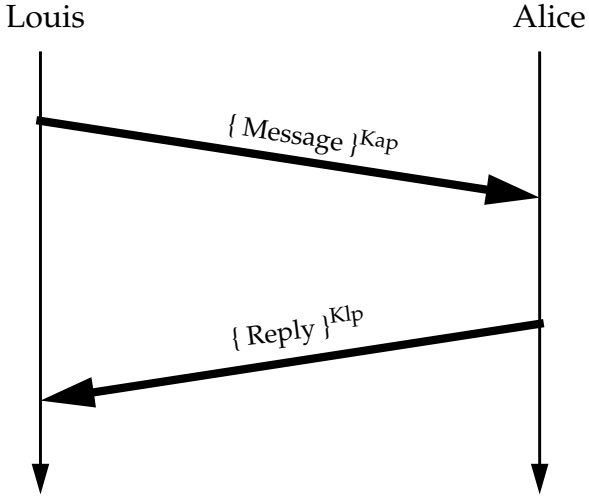
It takes 8080μs.

The critical message path consists of the black messages on the diagram above. The message we've greyed out, the KDC's response to Alice { T, Louis }Ka, is ***not*** *part of the critical path. As the diagram suggests, the KDC sends the temporary key to Louis before it sends the temporary key to Alice; Louis can begin working on his message before Alice receives hers.*

1. Encryption: Louis encrypts { Louis, Alice } with Kl	*10μs*
(All encryption and decryption in this example use the private-key scheme DES. Also note that all messages fit in one block.)	
2. Latency: Louis sends { Louis, Alice }^{Kl} to KDC	*2000μs*
3. Decryption: KDC decrypts { Louis, Alice }^{Kl}	*10μs*
4. Encryption: KDC encrypts { T } with Kl	*10μs*
5. Latency: KDC sends { T }^{Kl} to Louis	*2000μs*
6. Decryption: Louis decrypts { T }^{Kl}	*10μs*
7. Encryption: Louis encrypts { Message } with T	*10μs*
8. Latency: Louis sends { Message }^{T} to Alice	*2000μs*
9. Decryption: Alice decrypts { Message }^{T}	*10μs*
10. Encryption: Alice encrypts { Reply } with T	*10μs*
11. Latency: Alice sends { Reply }^{T} to Louis	*2000μs*
12. Decryption: Louis decrypts { Reply }^{T}	*10μs*
Total	*8080μs*

Another acceptable answer was 8090μs: the KDC might encrypt T with both keys before sending either message. This would add a step 4.5 to the list: KDC encrypts { T, Louis } with Ka.

3c. *(6 points)* Alyssa P. Hacker thinks that Louis's reasoning is all wrong, particularly when messages and replies are small. She proposes a public-key encryption system; with her protocol, the same message exchange would look like this. (She assumes that Louis and Alice already know each others' public keys.)



How long will it take for this message exchange to complete with Alyssa's protocol (encryption times, decryption times, and network latencies only)? Again, the message and reply each fit in a block.

With this protocol, the message exchange takes 8000μs: less than Louis's optimized protocol, even though expensive public-key encryption is used!

1. Encryption: Louis encrypts { Message } with Kap	*1000*μs
(All encryption and decryption in this example uses the public-key scheme, RSA.)	
2. Latency: Louis sends { Message }^{Kap} to Alice	*2000*μs
3. Decryption: Alice decrypts { Message }^{Kap}	*1000*μs
4. Encryption: Alice encrypts { Reply } with Klp	*1000*μs
5. Latency: Alice sends { Reply }^{Klp} to Louis	*2000*μs
6. Decryption: Louis decrypts { Reply }^{Klp}	*1000*μs
Total	*8000*μs

3d. *(6 points)* What guarantee does Louis's protocol provide for Alice that Alyssa's protocol doesn't? Assume that Louis keeps T secret.

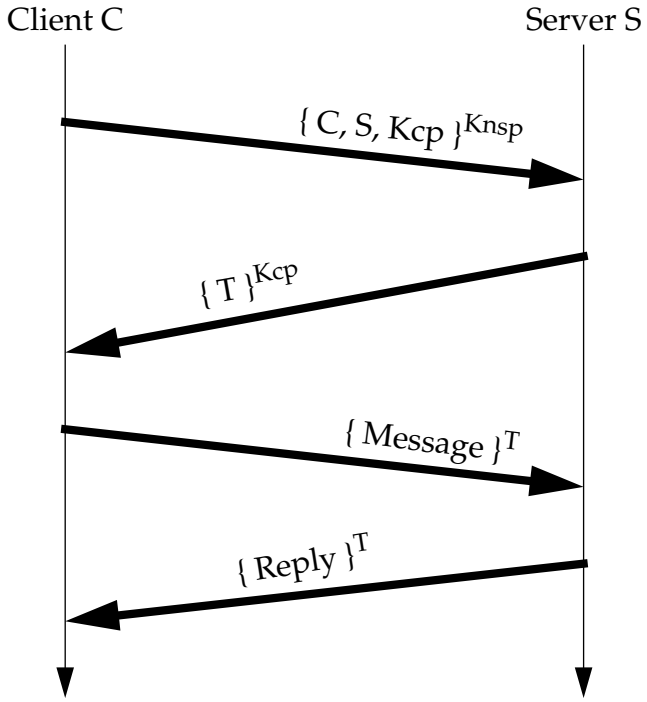
There are two good answers.

1. Louis's protocol provides Alice with ***authentication****: the message { Message }^{T} is guaranteed to come from Louis, since the KDC only sent T to Alice and Louis, and Louis didn't give it away. (We also assume the KDC is secure.) Since there are no active attackers, the message { Message }^{T} could not be a replay.*

2. Since there are no active attackers, authentication is not important! No one will ever pretend to be Louis anyway. Therefore, Louis's protocol provides ***no*** *guarantees whch Alyssa's doesn't. (It may be easier to determine the source of a message with Louis's protocol, since the KDC tells you explicitly, but this is not really a guarantee.)*

4. A startup company with an insanely high stock price, the Notsafe Communications Corporation, is designing a protocol for safe and private transactions over the Internet. Rather than rely on centralized security servers like Kerberos, the Notsafe designers are using public-key cryptography to implement their protocol. However, only the initial handshake uses public-key, since the protocol will be used for high-volume data transfers, where public-key cryptography is just too expensive. The handshake will set up a private key for communication between the parties, which then can continue for as much as a day or more.

   The Notsafe protocol has a well-known public key, K_{nsp}, which is used to establish connections. In addition, servers who want to talk the Notsafe protocol must use a Notsafe server program, which includes the secret key, K_{nss}. The protocol is graphically depicted below.



   In the first message, the client sends a handshake including C's public key, encrypted with K_{nsp}: $\{C, S, K_{cp}\}^{Knsp}$. The server decrypts this with K_{nss} and sends a response back to the client including a 32-bit private key T for the rest of the communication, encrypted with K_{cp}. Then, the client decrypts T with K_{cs} and sends messages and receives replies encrypted with T.

4a. *(2 points)* Notsafe's marketing department declares that "the risk of an intruder reading your sensitive data with Notsafe's products is less than one in one trillion." Mrs. Bach O. T. Envelope-Calculation immediately exclaims, "Poppycock!" Given the key that is being used, what might she have been thinking?

The private key T has only 32 bits, so there are exactly 2^{32} possible values for T. Therefore, an intruder making a random guess at T has a one in 2^{32} chance of guessing right.
$1 / 2^{32} \approx 1 / 4{\cdot}10^{9}$ is a much better chance than Notsafe's advertised risk, $1 / 10^{12}$!

4b. *(9 points)* List 3 fundamentally different problems with Notsafe's protocol.

*There are **many** things wrong with Notsafe's protocol. Valid answers included:*

1. Notsafe's secret key, Knss, is the same for all servers, and is stored in the server binary. An intruder can just purchase a server from Notsafe. She can then disassemble the binary to get Knss, or she could even feed the unmodified Notsafe binary messages she sniffed from the wire; it will dutifully decrypt them. Compromising Knss in this way breaks the entire protocol.

2. Replay attacks are possible (either on messages, or on the key T that is being sent encrypted).

3. There is no authentication being performed. C doesn't really know he is talking to S as all servers share the same public key. Also, since C's supposed public key Kcp is included in the first message (which anyone can spoof as Knsp is well-known), someone can pose as C by sending their own public key in place of Kcp. Because there is no authentication on either end, Lucifer could even sit in the middle, pretending to be the server to the client, and pretending to be the client to the server! This would allow Lucifer to read every message and reply sent, or even make alterations.

4. An active attacker can replace the encrypted session key $\{ T \}^{Kcp}$ with random data. When the client decrypts this, it will obtain something that looks like a temporary session key; however, C will not be able to communicate with S with this garbage key, causing denial of service.

4c. *(5 points)* List one possible problem that could arise from a very poor implementation of the Notsafe server (*not* a fundamental problem with the protocol itself).

Here is one problem.

T is a randomly generated session key. If the pseudorandom number generator is weak, T becomes predictable. (In the worst case, the same T may be used over and over. T may also be easy to guess in other ways; if the seed for the random number generator is based on the current time, for example, which is essentially the same everywhere. The real Netscape Communications Corporation had a protocol which broke for this very reason.)

4d. *(8 points)* Introduce two changes to the Notsafe protocol which improve its security and explain them briefly. *You are not expected to create a completely secure protocol*.

Again, many possible answers. Not including "buy a working protocol," some are:

1. Use a timestamp or nonces to avoid replay.

2. Make T much larger than 32 bits to make brute-force guessing harder.

3. Don't have all servers use the same secret key. Instead, let every server set up its own public/secret key pairs; clients then have to obtain the right public key via certifying authorities.

*4. Add authenticaion. This is not simple to implement. One possible method requires **everyone** to have a well-known public key; therefore, a separate distribution method is needed for the keys. The server no longer receives Kcp with the first message, and must instead get Kcp itself through secure channels. Also, nonces can be used to ensure each message is part of the authenticated chain, and not a replay.*