CS 6014 Midterm Sample Exam

The exam is in-class, open-notes, and closed-computer. There is no limit on the size of your paper notes. The actual exam will have questions like the ones in this sample, but possibly with small variations in the style of the questions.

Name: ________________________

Question 1. (10 points) Here is a set of terms:

```
propagation        exponential backoff
transmission       address
congestion         flow control
port number        slow start
```

Use each term at most once to fill in the following blanks, and fill in blanks using only the terms above:

- TCP uses ________________________ as part of its management of ________________________.
- A ________________________ delay correlates with physical distance, while ________________________ delays can depend on the medium independent of the distance.
- A host can have both a TCP and UDP socket open for the same ________________________.

Expect a question with more terms on the actual exam.

Question 2. (5 points) Given these subnet designations:

Name	Prefix
east	`128.10.0.0/17`
west	`128.10.128.0/17`
north	`128.9.128.0/22`
south	`129.9.1.0/24`

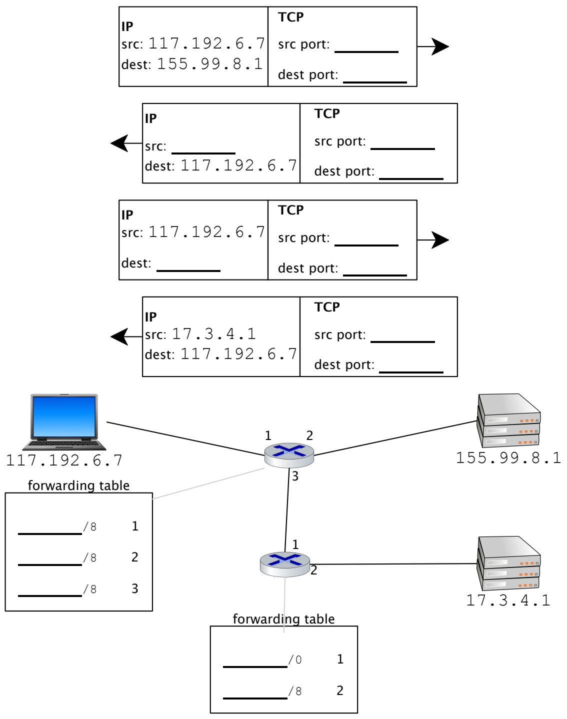
Which subnet do each of the following address belong to? Write "none" if it's none of the above.

- `128.5.10.128` is in ______________________
- `128.10.64.1` is in ______________________
- `128.10.254.1` is in ______________________
- `128.9.130.128` is in ______________________

Question 3. (10 points) Alice's laptop is connected to a network with two routers and two servers. She sends a query to one of the servers using TCP at port 123 and gets a reply in that TCP connection. She then sends a query to a second server at port 456, and again gets a response.

The picture on the following page represents Alice's network configuration and some packets that are sent in this scenario. Fill in blanks in the picture, using a letter abbreviation in each blank. You choose the abbreviation and record your choice in the abbreviation table. That is, each blank outside of the abbreviations table should have one letter, and the abbreviations table says what you mean by each letter. Note that, for some blanks, there is not a unique address or number that works. Also, you get to pick the letters for abbreviations (although "A" and "B" are suggested starting points). The content of some blanks determines the content content of other blanks, however. You **must** use the same abbreviation where addresses or numbers to fill in blanks must be the same, and you **must not** pick the same address/number or abbreviation for values that *could* be different.

laptop ↔ servers packets

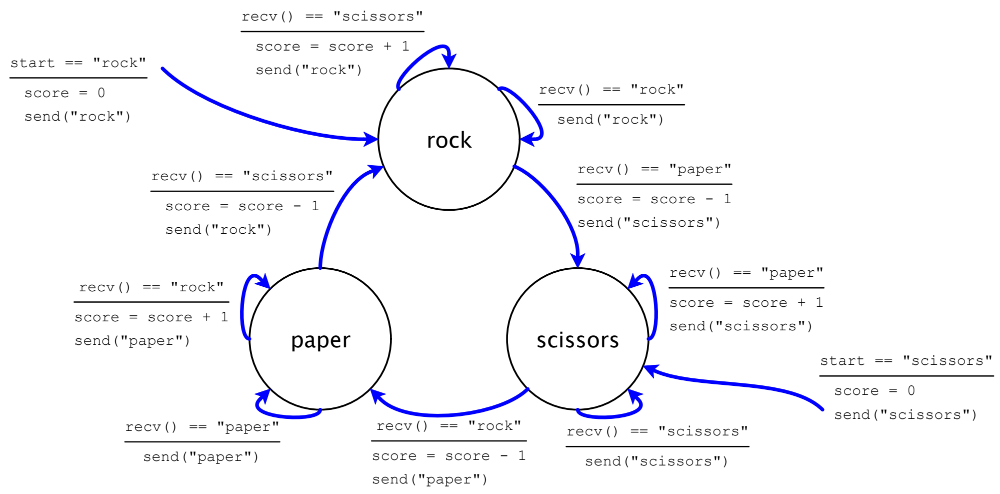


Your abbreviations

A =

B =

Question 4. (10 points) Two machines M1 and M2 both use this state machine:



Note that the state machine has two entry points, depending on the value of `start`. Assume that M1 and M2 start at the same time, plus the following:

- `send` takes 1 second.
- `recv` is an instantaneous event that fires when a message was sent by the other machine in the past second.

Scenario 1: M1 and M2 both run with `start = "rock"`.

Value of `score` on M1 after 5 seconds: ________________________

Value of `score` on M2 after 5 seconds: ________________________

Scenario 2: M1 runs with `start = "rock"` and M2 runs with `start = "scissors"`.

Value of `score` on M1 after 5 seconds: ________________________

Value of `score` on M2 after 5 seconds: ________________________

To enable the possiblity of partial credit, include a brief rationale for your answers ni the margins.

Question 5. (5 points) Fill in the two blanks below.

A TCP connection receives four packets in a row, without yet sending any packets back in between:

- sequence number = 100, acknowledgement number = 5, data is 5 bytes
- sequence number = 105, acknowledgement number = 5, data is 5 bytes
- sequence number = 115, acknowledgement number = 5, data is 5 bytes

From these packets, the local connection infers that the other end has received all of the data that it sent previously, and the connection is ready to send back 7 bytes. When it sends that packet, the sequence number should be ________________________. The acknowledgement number should be ________________________.

Following questions rely on an idealized `MSDcryptonet` library providing random-number generation, a one-way function, an inverse-pair generator, and operations on N-bit integers of type `Int` (for some predefined N, but we don't specify it here).

Imagine that this library is provided in a C-like, Java-like, or Python-like setting where you also have lists or arrays (treat the type `List` as an array if you find that more convenient) with 0-based indexing, plain `int`s, `for` loops, and so on. If you prefer to imagine a Python-like language, treat the types as comments.

- `Int random()`

  Returns a random N-bit number.

- `Int one_way(Int a, Int b)`

  Takes two N-bit integers and returns another N-bit integer, where there's no feasible way to take the result and the one of the arguments to find the other argument (i.e., any other argument that would produce the same result). Furthermore, `one_way` is associatve: `one_way(one_way(a, b), c)` produces the same result as `one_way(a, one_way(b, c))` for any `a`, `b`, and `c`.

- `List inverse_pair()`

  Returns a random pair of N-bit integers `a` and `b` (in a two-element list) such that both `one_way(one_way(c, a), b)` and `one_way(one_way(c, b), a)` is `c` for all `c`. These pairs are so rare that you'd never find them by random search, but somehow also plentiful enough that every random pair is unique.

- `Int plus(Int a, Int b)`

  Returns the sum of two N-bit numbers modulo 2^N.

- `Int times(Int a, Int b)`

  Returns the product of two N-bit numbers modulo 2^N.

- `Int xor(Int a, Int b)`

  Returns the xor of two N-bit numbers.

- `Int one()`

  Returns 1 as an N-bit integer.

- `void broadcast(String from, String to, Int message)`

  Broadcast an N-bit integer message that purports to be sent by `from` and that is intented for `to`.

- `Int receive(String from, String to)`

  Receives an N-bit integer message that purports to be sent by `from` and that is intented for `to`—but there's no guarantee who actually sent the message, and anyone can receive it.

The actual exam will refer to this same idealized library.

Question 6. (5 points) The following functions use the `MSDcryptonet` library to implement a block cipher as applied to a list of plaintext values. It's best charactered as using which common **mode of operation**?

```
List encrypt(List plaintexts, Int key) {
  List ciphertexts = new List();
  ciphertexts[0] = random()
  for (int i = 0; i < plaintexts.length; i++) {
     ciphertexts[i+1] = xor(plaintexts[i],
                            one_way(key, ciphertexts[i]));
  }
  return ciphertexts;
}

List decrypt(List ciphertexts, Int key) {
  List plaintexts = new List();
  for (int i = 0; i < ciphertexts.length-1; i++) {
     plaintexts[i] = xor(ciphertexts[i+1],
                         one_way(key, ciphertexts[i-1]));
  }
  return ciphertexts;
}
```

Mode of operation: ______________________

Question 7. (10 points) Alice and Bob want to play a guessing game where Bob guesses a number and Alice answers "yes" or "no." The code below shows part of an attempt to implement this game using `MSDcryptonet`. Assume that key pairs are generated by Alice and Bob using `inverse_pair` and that Alice's and Bob's public keys have been reliably communicated to each other. Bob will always guess a different number, but Alice and Bob do not want anyone else to see a guessed number or know whether it was right.

```
// Run on Alice's machine, and uses an external `a_ok` function
void a_check(Int a_priv_key, Int a_pub_key, Int b_pub_key) {
  Int ct_msg = receive("bob", "alice");
  Int msg = one_way(ct_msg, b_pub_key);

  Int ans = one();
  if (!a_ok(msg))
    ans = xor(one(), one()); // => zero

  broadcast("alice", "bob", one_way(ans, a_priv_key))
}

// Run on Bob's machine
bool b_try(Int msg, Int b_priv_key, Int b_pub_key, Int a_pub_key) {
  Int ct_msg = one_way(msg, b_priv_key);
  broadcast("bob", "alice", ct_msg);

  Int ct_ans = receive("alice", "bob");
  Int ans = one_way(ct_msg, a_pub_key);

  if (ans == one())
    return true;
  else if (pt_ans == xor(one(), one()))
    return false;
  else
    error("answer was not from Alice");
}
```

Unfortunately, despite the use of keys to sign messages, this implementation of the game is susceptible to interference by third parties after a little while. Specifically, Bob may start to receive messages that he incorrectly judges as being from Alice, and so he may get the wrong reply without reporting a "answer was not from Alice" error.

Part 1: Describe an attack to interfere with Alice and Bob's game.

Part 2: Annotate the code above with changes to repair the problem so that your attack can no longer intefere without detection. That is, Bob can check whether a reply for a guess is really from Alice.

Answers

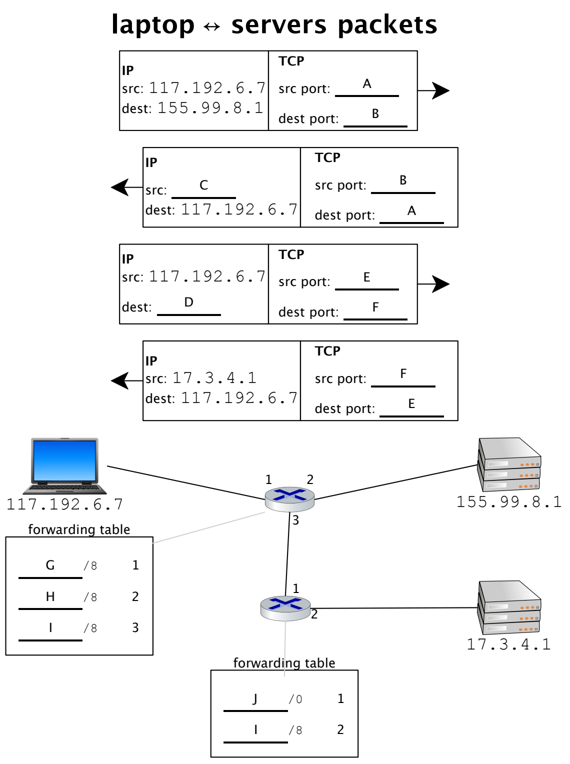
1.

- TCP uses *exponential backoff* as part of its management of *congestion*.
- A *propagation* delay correlates with physical distance, while *transmission* delays can depend on the medium independent of the distance.
- A host can have both a TCP and UDP socket open for the same *port number*.

2.

- `128.5.10.128` is in *none*
- `128.10.64.1` is in *east*
- `128.10.254.1` is in *west*
- `128.9.130.128` is in *north*

3.



Your abbreviations

A = 7765 // or any distinct ephemeral port
B = 123
C = 155.99.8.1
D = 117.192.6.7
E = 9976 // or any distinct ephemeral port
F = 456
G = 117.0.0.0
H = 155.0.0.0
I = 17.0.0.0
J = 0.0.0.0

4.

Scenario 1:

Value of `score` on M1 after 5 seconds: *0*

Value of `score` on M2 after 5 seconds: *0*

Both machines stay in the rock state and keep sending `"rock"` without changing `score`.

Scenario 2:

Value of `score` on M1 after 5 seconds: *-1*

Value of `score` on M2 after 5 seconds: *1*

Time	M1	M2
0 seconds	rock state	paper state
	recvs `"paper"`	recvs `"rock"`
1 seconds	`score` is -1	`score` is 1
	scissors state	paper state
	recvs `"paper"`	recvs `"scissors"`
2 seconds	`score` is 0	`score` is 0
	scissors state	rock state
	recvs `"rock"`	recvs `"scissors"`
3 seconds	`score` is -1	`score` is 1
	paper state	rock state
	recvs `"rock"`	recvs `"paper"`
4 seconds	`score` is 0	`score` is 0
	scissors state	rock state
	recvs `"rock"`	recvs `"scissors"`
5 seconds	`score` is -1	`score` is 1

5.

When it sends that packet, the sequence number should be *12*. The acknowledgement number should be *110*.

6.

Mode of operation: *output feedback (OFB)*

7.

Part 1: Others who see Alice's broadcasts can save and resend her messages that mean 1 or 0, even though they couldn't have produced those messages before seeing any of Alice's replies. After collecting the two different answers, anyone on the network could send either one with `broadcast("alice", "bob", ....)`.

Part 2: Alice needs to send a different reply for every guess, and Bob needs a way to make sure a reply is for his specific guess. Both of those can be achieved by having Alice reply with the guess, but flipping the low bit if the guess is wrong. Since Bob guesses with a different number every time, others cannot construct a suitable reply or interfere by sending an old reply.

This part of Alice's code:

```
Int ans = one();
if (!a_ok(msg))
  ans = xor(one(), one()); // => zero
```

can be changed to

```
Int ans = msg;
if (!a_ok(msg))
  ans = xor(msg, one()); // flip bit
```

And this part of Bobs code:

```
if (ans == one())
  return true;
else if (ans == xor(one(), one()))
  return false;
```

can be

```
if (ans == pt_msg)
  return true;
else if (ans == xor(pt_msg, one()))
  return false;
```