# Final Examination
# ECE419F: Distributed Systems, Fall 2001

Instructor: Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

*Notes:*

1. Make sure that you have all **14 pages** (9 parts) before you begin.
2. You have **150 minutes** to complete this examination.
3. This is a closed-book closed-notes (Type A) examination.
4. You are allowed to use a basic non-programmable (Type 2) calculator.
5. You are expected to uphold the highest degree of academic integrity.
6. Use the back pages if your answer does not fit in the space provided, and for general scratch area.
7. **Do not turn to the next page until starting time is announced.**

| Part number | Maximum score | Your score |
|---|---|---|
| 1 | 10 | |
| 2 | 20 | |
| 3 | 10 | |
| 4 | 10 | |
| 5 | 10 | |
| 6 | 10 | |
| 7 | 10 | |
| 8 | 10 | |
| 9 | 10 | |
| total | 100 | |

**Your Last Name:**

**Your First Name:**

**Your Student Number:**

## Part 1 - *Multiple Choices Questions* (10 Points)

Please select one and only one choice as an answer for each question.

1. (2 Points) Which of the following type of failure is **not** one of the *benign failures*?

   (A) send and receive omission failures

   (B) fail-stop failures

   (C) crash failures

   (D) general omission failures

   (E) None of the above

2. (2 Points) Which of the following statements is **not** true with respect to the clients in the *Network File System* (NFS)?

   (A) Clients cache individual files and directories in their main memory

   (B) Clients cache a timestamp indicating when the file was last modified on the server

   (C) Cached copies on the clients are only synchronized with the server when the files are opened or closed

   (D) After a validation check, cached blocks are assumed valid for a finite interval of time

   (E) None of the above

3. (2 Points) Which of the following is one of the features of the *remote reference module* in the generic model of middleware platforms?

   (A) It is responsible for maintaining a table containing mappings from names to remote object references

   (B) It is responsible for translating between local and remote object references

   (C) It is responsible for translating between remote object references and their external representations

   (D) It is responsible for generating the client proxies and server skeletons from the interface definition language

   (E) It is responsible for implementing the request-reply protocol between the client and the server

4. (2 Points) Which of the following statements is **not** a property of the *Domain Name System* (DNS)?

   (A) DNS is designed to resolve domain names into IP addresses of *mail hosts*

   (B) Any DNS server may cache data from any other servers, in order to increase performance

   (C) DNS is able to support recursive or iterative navigation in domain names

   (D) A time-to-live value is associated with each cached entry in order to assist propagating changes

(E) None of the above

5. (2 Points) Which of the following is **not** one of the measures that the CSMA/CD protocol uses to deal with the possibility of packet collisions?

   (A) The network interface hardware in each station listens for the presence of a signal; the station waits till there is no signal is present, and then begins to transmit packets

   (B) When a station is transmitting a packet through its output port, it also listens on its input port, to make sure that the two signals are identical

   (C) If there is a collision, the station transmits a jamming signal

   (D) When a collision occurs, the station chooses to wait a back-off time period before retransmitting

   (E) The back-off time period used by a station is doubled whenever there are further collisions, and reset to its initial value when the packet is transmitted successfully

**- You may use the space below as scratch area -**

## Part 2 - *Short Answer Questions* (20 Points)

Answer the following questions with a short but clear answer. Lengthy answers may not warrant extra credits, and you will have less time for later problems. **Please write legibly.**

1. (7 Points) When we discuss issues in fault-tolerant broadcasts, we define the term *causal order* as follows. If the broadcast of a message $m$ *causally precedes* the broadcast of a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$. However, this definition relies on the notion of *causal precedence*. Please present a formal definition of causal precedence when this notion is used *in the context* of defining causal order in fault-tolerant broadcasts.

2. (7 Points) There are four kinds of actions that may be performed by a transaction: *updates*, *reads*, *commits* and *aborts*. With respect to the *update-in-place* method used in abort recovery, what should be implemented for these actions?

3. (6 Points) In the Andrew File System, Vice servers and Venus clients use a mechanism involving the *callback promise* to maintain cache consistency. A callback promise is a token with two states: *valid* or *canceled*. Assume that Venus has already held a callback promise for a certain file that it requests to open. With respect to the callback promise, what actions should be performed on Venus to open the file? Assume further that after opening the file, Venus has modified the content of the file and wishes to close it. On closing, what actions should be performed on Vice?

## Part 3 - *Wireless Local Area Networks* (10 Points)

One of the problems that should be considered in the design of Medium Access Control (MAC) protocols in wireless local area networks is referred to as the "**exposed station**" problem. Answer the following questions.

1. Part 3A (5 Points): Briefly illustrate the problem, using a simple scenario with three stations, $A$, $B$ and $C$.

2. Part 3B (5 Points): One of the proposals suggested to solve the "**exposed station**" problem is to use Request-to-Send (RTS) and Clear-to-Send (CTS) packets. In your simple scenario above with three stations, how does such a solution work?

## Part 4 - *Failure Models* (10 Points)

In order to illustrate the effects of failures on the design of protocols in a distributed system, we may use the following example. Two processes, $A$ and $B$, communicate by sending and receiving messages on a bidirectional channel. Neither process can fail. However, the channel can experience transient failures, resulting in the loss of a subset of the messages that have been sent. We wish to devise a protocol where either of two actions $\alpha$ and $\beta$ are possible, and either (i) both processes take the same action; or (ii) neither takes both actions. In this course, we claim that this problem has no solution, i.e., such a protocol is impossible to devise. Please prove the correctness of this claim.

## Part 5 - *Security* (10 Points)

Before Alice sends a message to Bob, she wishes to first digitally sign the message, and then encrypt it for better security. For both signing and encrypting the message, she only considers using asymmetric cryptographic algorithms. In addition, she wishes to use secure digest functions to improve efficiency. Assuming that the complexity of digital certificates are not involved, explain the steps that are needed to be performed on *both Alice and Bob* to complete the delivery of the message.

## Part 6 - *Reliable Broadcasts* (10 Points)

In an asynchronous distributed system, if we assume that:

(1) Processes in the system may only suffer from benign failures (those who may fail are said to be *faulty*, otherwise they are *correct*);

(2) Links between processes are reliable;

(3) The point-to-point inter-process communication primitives (*send* and *recv*) are available from the lower layers that satisfy the following properties:

(3A) *Validity:* if $p$ sends $m$ to $q$, and both $p$ and $q$ are correct, then $q$ eventually delivers $m$;

(3B) *Integrity:* For any message $m$, $q$ receives $m$ at most once from $p$, and only if $p$ has previously sent $m$ to $q$.

Under these assumptions, we consider implementing *reliable broadcast* using *send* and *recv*. Answer the following three questions.

1. Part 6A (3 Points): How do we define the *agreement*, *validity* and *integrity* properties of reliable broadcasts?

2. Part 6B (3 Points): Before any implementations of reliable broadcast are possible, we need to make *one further assumption*. What is this critical assumption, without which reliable broadcasts can not be implemented even in synchronous distributed systems?

3. Part 6C (4 Points): One of the implementations of reliable broadcasts introduced in this course is referred to as the *Diffusion Algorithm.* In pseudocode, describe this algorithm in details.

## Part 7 - *Concurrency Control* (10 Points)

Assume that we have two transactions in a distributed system, $T_1$ and $T_2$. We use the notation $w_k(x)$ to indicate a *write* operation in $T_k$ on object $x$, and $r_k(y)$ to indicate a *read* operation in $T_k$ on object $y$. $c_k$ denotes the *commit* operation in $T_k$, and $a_k$ denotes the *abort* operation in $T_k$. We have three objects, $x$, $y$ and $z$. Consider the following schedules:

Schedule $S_1$: $r_2(x)w_1(x)w_2(x)w_2(y)w_1(x)r_1(y)c_2c_1$
Schedule $S_2$: $r_2(x)r_1(x)w_1(y)w_1(x)r_2(z)w_2(z)c_1c_2$
Schedule $S_3$: $r_1(x)r_2(x)r_1(y)r_2(z)w_1(y)a_1w_2(y)w_2(z)c_2$

1. Part 7A (3 Points): Is schedule $S_1$ *conflict serializable*? If so, please give the serial schedule that $S_1$ is conflict equivalent to. Otherwise, please show your reason.

2. Part 7B (3 Points): Will schedule $S_2$ be generated by the execution of the *two-phase locking* mechanism? If so, explain the detailed steps of acquiring, promoting and releasing locks. Otherwise, justify your answer.

3. Part 7C (4 Points): Will schedule $S_3$ be generated by the execution of the *strict two-phase locking* mechanism? If so, explain the detailed steps of acquiring, promoting and releasing locks. Otherwise, justify your answer.

## Part 8 - *Transaction Recovery* (10 Points)

In this course, we have introduced one of the proposals to ensure transactions are recovered appropriately after a system crash that erases contents in the main memory. This measure enforces the *undo rule* and the *redo rule* when the transaction commits or progresses normally, and introduces two phases — a *redo phase* and an *undo phase* — to recover transactions after a crash. Answer the following questions.

1. Part 8A (5 Points): What are the details of the *redo rule* and the *undo rule* when the transaction commits or progresses normally?

2. Part 8B (5 Points): A researcher, Alice, claims that such proposal that involves a redo phase and an undo phase is **incorrect**. She gives the following example. Consider a transaction $T$: $w(x)w(y)w(z)$ *[commit]* . . . *[crash]*. Assume that at the time of crash, all new values of $x, y$ and $z$ are lost since none of them have been written to secondary storage. Since the crash occurs after the transaction commits, we need to reinstall the new values of all objects. We first execute the *redo phase*. Using redo records, we are able to re-write new values of $x, y$ and $z$. However, during the later *undo phase*, we start to use the undo records to revert all objects to their original values! Obviously, the protocol does not work correctly. Based on what you have learned, is Alice correct? Justify your answer.

## Part 9 - *Global States* (10 Points)

We consider using a *monitor* process $p_0$ to construct *consistent observations* of the distributed computation. $p_0$ achieves this by using a *reactive* architecture, where $p_0$ will assume a passive role. Whenever processes execute an event, they notify $p_0$ by sending it a *notification message*. $p_0$ then constructs an *observation* of the underlying distributed computation as the *sequence of events* corresponding to the order in which the notification messages arrive. Assume that the processes never fail, the communication channel is reliable, but messages may be received out-of-order. The observation is determined to be *consistent* only when the events within such an observation are *causally ordered*. In this general context, answer the following questions.

1. Part 9A (3 Points): Due to the lack of a global real-time clock, we have learned that we can devise a simple clocking mechanism referred to as the *logical clock* to replace the global real-time clock, as the logical clock satisfies the *Clock Condition*. What is the *Clock Condition*? What is the formal definition of the *Logical Clock*?

2. Part 9B (3 Points): We have learned that one of the major advantages of using *vector clocks* over *logical clocks* to construct consistent observations on $p_0$ is that the vector clock supplies the *gap detection property* in certain cases, where the logical clock does not. This is critical to guarantee *liveness* in an asynchronous distributed system. What is the gap detection property?

3. Part 9C (4 Points): Why does the vector clock supply the *gap detection property*?