Name:

**CSE 291 (Distributed Systems)**
**Spring 2016 (Kesden)**
**Homework #1**

## 1. Networks

(a) Given a 100Mbps network with an optimum sliding-window size of 1024 bytes, what would be the optimal window size if the bitrate was increased to 1Gbps and the maximum allowable run of the cable was cut in half? Why?

*1024 bytes * (1Gbps = 1,000Mbps / 100Mbps) * ½ (run size) = 1024 bytes * 5 = 5120 bytes*

*Cutting the run in half cuts the number of bytes in half, reducing the needed window by half. Increasing the bit rate by 10x increases the number of bits in flight by 10x, increasing the required window size by 10x.*

(b) When transmitting at the link layer, we normally “frame” data. Why? For example, why do we not just stream bits?

*It would be nearly impossible to tell where to start and stop listening, to perform error detection and/or correction, to recover from lost or out of order data, etc. Frames provide a unit we can manage.*

(c) Consider global scale communication over the next 30 years. In which dimension, latency or bandwidth, will we mostly likely see the greatest improvement? Why?

*Bandwidth (Bits/second). Bandwidth can be improved by faster clocking and more parallelism. Latency is subject to the speed of light, which isn’t getting much faster. We can shake some more inefficiencies out – but not too many.*

## 2. Middleware/RPC/RMI

(a) Consider the implementation of an RPC system in a homogenous environment (same hardware, same OS, same language, same, same, same). Is it possible to implement a pass-by-reference (not necessarily pass-by-address) mechanism? If not, why not? If so, in what ways might it be best to relax the semantics of a typical local pass-by-reference situation? Why?

*Yup. This isn't any different than what you did for RMI in the project. Consider your "Remote Object References (RoRs)". But, without a shared memory, there will more of a problem maintaining consistency, etc.*

(b) Consider the implementation of an RPC system in a heterogeneous environment (different processor architecture, different OS, different programming language, different, different, different). How might the heterogeneity complicate the model? Please consider each of the following:

i) Simple primitives (consider endian-ness, width, etc

*Well, I practically gave away the answer on this one. One would need to agree on how values should be represented in transmission, any limits/bounds to enable compatibility, and take care of network←>local conversion*

ii) Complex data types, including structs and strings,

*Consider things like null termination, strings as tuples including size, the character set, mutability, values vs refernces, etc. And, in the case of OO languages, the ability to objectify simpler forms.*

iii) Higher-order language and library data structures, such as linked lists, maps, etc.

*Here we've got everything in (ii) above, as well as an obvious mess ranging from operations to the memory model.*

iv) Programming paradigms (function pointers, jump table, functors, etc)

*These, quite simply, might not even be possible to approximate in an automated way. I don't know, for example, how to turn a C jump table (functions and function pointers) into a Java equivalent in an automated way. What would need to be done? Well, beginning with translating the C code to Java, it gets hard.*

(c) Consider Java's RMI facility, which generates stubs at compile time. Could it, instead, generate the stubs at runtime? For example, could it disassemble a class file, or inspect an object's properties at runtime, rather than at compile time? If not, why not. If so, what would be the advantages and disadvantages of this model?

*Yup. You did much of this in your project. It is nice in that it is clean, and prevents any version problems. But, it does make more work at runtime, which can slow down things, reducing throughput.*

(d) Consider Java's RMI facility, which only plays nicely with classes that implement the *Serializable* or *Remote* interfaces. Would it be possible to implement an RMI facility in Java that worked for all classes? For example, by using a combination of the class file, as well as reflection and other Java mechanisms to decompose, serialize, and reconstitute instances by brute force? If so, please explain any necessary limitations. If not, please example why not.

*Not really. We can't know how to serialize something without knowing a lot about what it is. And, that is something only the programmer knows. How does one serialize, for example, a database? And, does that even make sense? Do we copy the handle to the server? The caches associated with that handle? Or the database, deeply? And, how?*

## 3. Distributed Concurrency Control

In class we discussed enforcing mutual exclusion, among other ways, via a central server, majority voting, and token ring.

(a) Which of these systems requires the fewest messages under heavy contention? How many messages are required per request?

*Token Ring. One per request.*

(b) Which of these systems requires the most messages under heavy contention? Why?

*Voting. It requires the same number of messages under any level of contention (neglecting RELINQUISH). But, it is bad.*

(c) Which of these systems is most robust to failure? Why?

*Token ring can handle any number of host failures – without additional delays. The token just gets passed past the dead participant(s). The lost time takes less time than the critical section would, right?*

(d) The *voting protocol*, and the *voting district* protocol, are based upon participants reaching an agreement as to who can enter the critical section. What happens in the event of a tie that could otherwise risk deadlock?

*Some type of total ordered logical time stamp is used to enable a host to determine that it voted for a lower-priority request. It makes a RELINQUISH request to get its vote back. Since the system is otherwise deadlocked, its vote is available. In this way, votes will move to the single correct winner, breaking the tie.*

(e) Many coordinator election protocols have analogous mutual exclusion protocols and vice-versa. What is the most important similarity of the two problems? What is the most important difference? Focus your answer on the nature of the problem, itself, not the solution.

*They are similar in that one host is selected to be special. They are different in that mutual exclusion requires only that each participant know only whether or not it is special (has access to the critical resource). Coordinator election requires that each participant knows who is special (the coordinator). We could also guesstimate that coordinator election is required less often than access to a critical section (but this might not be true). We could also guesstimate that a coordinator election may be followed by some type of coordination.*

(f) In the event of a partitioning, techniques such as token ring can result in two or more distinct groups, each with its own coordinator. How can this be prevented, while enabling progress?

*One easy way is just to require a majority to form a new group of participants.*

## 4. Logical time

(a) One form of logical time is per-host sequence numbers, e.g. “5.1” is time 5 on host 1. Another form is Lamport’s logical time. What advantage does Lamport time offer? At what cost?

*Lamport time provides an ordering that respects the “happens before” relationship. This enables the ordering of intervals between a sender and receiver. It requires that timestamps be sent along with messages and can result in “missing time” as the timestamps do not necessarily advance monotonically.*

(b) In class, we discussed a simple form of vector time. What relationship(s) among timestamps can be discerned from this form of vector logical time, but not Lamport logical time?

*We discussed a simple vector timestamp that enabled us to detect "[Potential] Causality Violations".*

(c) Consider the amazing progress we've made in data communication networks over the years. Will it –ever-- be possible to sync physical clocks rapidly enough to use as the basis for correct synchronization of a global distributed system? Why or why not?

*Nope. That speed of light thing, again. With any significant distance, we lose too much work.*

## 5. Replication and Quorums

(a) Consider replication to multiple servers based upon a write-one, read-all static quorum with version numbering. What steps must be taken upon a write to ensure the correctness of the version number?

*The version number must be read from a read quorum before being advanced for the new version.*

(b) Consider Coda Version Vectors (CVVs). Give an example of two concurrent CVVs. What is indicated by concurrent CVVs? Please describe one situation in which this might occur.

*[2,2,1,1] and [1,1,2,2] or [1,0,1,0] and [0,1,0,1], etc.*

*Concurrent versions numbers indicate that writes happened to disjoint sets of servers. This might result from a partitioning.*

(c) Consider a situation with 5 replica servers employing a write-2/read-4 policy and version numbering. Please design and describe a locking scheme that ensures that version numbers remain correct, even in light of concurrent writes. You do not need to consider failure. But, you do need to describe all necessary communication and state, such as communication between or among servers and/or participants.

*One easy answer is to obtain a write-lock on a read quorum until after the version umber is read and the subsequent write is completed.*

(d) Please consider the special case of a mobile client using a read-one/write-all quorum. Please design an efficient locking protocol to ensure concurrency control. But, please ensure that your protocol permits the lock to be acquired at one replica and released by another.

*There are (at least) three key ideas here (among potentially many more). The first is that we want to be able to pick any server and use it as a proxy to request and/or release a lock on behalf of a client, so the client need not connect to all of the servers for reads. The second is that we want to lock objects, not whole servers. And, the third is that we might want to let writes post to one server and then atomic commit from there to minimize the time that reads are limited by writes-in-progress. It could take a while for a slow client to finish a write, which could lock an object for a while, forcing us to then implement a time-out mechanism. But, many answers are potentially very good here.*

(e) Coda Version Vectors (CVVs) are a form of logical time stamp, specifically a vector time stamp, which are used to aid in the management of replication. What could cause two CVVs be *concurrent* (and non-identical)? Why? Illustrate your answer with an example involving 2 servers and 2 clients.

*Sorry. This was an unintended duplicate. See 5(b).*