

Team 1: 3 of 5

Pros:

- Simple and clear message formatting.
- Easy to understand, step by step descriptions of the protocols.

Cons:

- There is no discussion of how a server reconnects to former children following a failure.
- The communication details are specified but a lot of the policy issues are missing.

Team 2: 4 of 5

Pros:

- Good detailed explanation of interactions.
- Covered all topics well.

Cons:

- There are some good suggestions in there that are presented more as requirements. They are getting very specific about exactly what each server will do and not do. For example, the random number generation of neighbors. They tell me I have to do this, but then don't tell me what the range of the random numbers are.
- Does the message format mean serialized Java String or a character array (would it then be null terminated)?

Team 3: 3 of 5

Pros:

- Brief but thorough.
- Precise definitions of messages.

Cons:

- You say, "this project employs the serialization of objects over UDP, the class structure for each message type is embedded with the abbreviations." Is not the point of Java's object serialization that we don't have to do this and the object is converted to/from a byte array for transmission? As this statement stands, you're not using serialization properly. How about defining a packet object that contains all of these things and have new servers that log on as neighbors for the byte-code describing the latest packet class if you want this route and not embed class contents in a message as plain-text? Otherwise, write to the lowest common denominator and have a message format that is plain-text and the parsing of the message is well defined and forget about forcing serialization on the project.

Team 4: 4 of 5

Pros:

- Good combination of explanation along with precise message descriptions.
- Seemed to cover all the bases

Cons:

- They list the message name and message structure, but never show how the message name is attached to the message structure. This is a huge failing in the specification. They also did not address how the messages are to be transmitted, which Dr. Newman specifically brought up.

Team 5: 4 of 5

Pros:

- Good explanations. Good mix of protocol specifics and addressing some policy issues.

Cons

- Lacked discussion of name collision.

Team 7: 2 of 5

Pros:

- Covered all parts of the specification

Cons:

- I think there needs to be more of a unified message format
- No statement of type of transmission

Team 8: 1 of 5

Pros:

- Attempted a complete mathematical unambiguous FSM representation of the system, even though they were unable to finish.

Cons:

- Put too much emphasis on the diagram but left out the substance of how states are transitioned other than some text, some transitions have no condition associated with them.
- If you're going to go through the trouble of making state diagrams, please make a full state transition matrix so there is no ambiguity in how the conditions affect the state transition. What if you receive some message that doesn't have an exit action for your current state???

Team 9: 1 of 5

Pros:

- The second submission was succinct

Cons:

- The team has us sending multiple objects for one message in a “packet.” Even though all the objects are serializable, there is no notion of how we are supposed to send all of these objects at once. Do we create a packet object that implements Serializable and contains these elements? Do we place them one after another in serialized byte form directly? Looking to the previous submission did nothing to clarify these issues.

Team 10: 3 of 5

Pros:

- Had something to cover every aspect of the spec.

Cons:

- No notion of delimiters in their description.
- Does “string” mean serialized Java String or a character array (would it then be null terminated)?

Team 11: 2 of 5

Pros:

- Had a protocol with delimited messages explicitly for a set of action that could successfully implement the behaviors from the spec.

Cons:

- Used terms neighbor and alternate server
- Why is “ReadyToJoin messages are collected from different users” collecting from users? Are not the servers joining?
- Does the message format mean serialized Java String or a character array (would it then be null terminated)?
- Not sure what a “Files” is from the “Connecting to a server” section. If you’re not willing to spend the time to proof-read, did you really spend the time to thoroughly think through a protocol?

Team12: 3 of 5

Pros:

- Tree-like server architecture minimizes the number of hops to distant servers.
- Message formats are clear and easy to understand and implement

Cons:

- We find that maintenance of this tree structure in the event of server failures and recoveries is very complicated and since the tree structure isn't really necessary, this would just be a pain to implement.

Team13: 2 of 5

Pros:

- With the exception of server recovery, it gets the job done

Cons:

- A little convoluted, with network functionality excessively intermingling with message format specification, so may be difficult to understand when attempting to implement
- Does not follow spec as discussed in class: Server recovery protocols communicate with neighboring servers rather than directly to former clients

Chaudhuri: 2 of 5

Pros:

- Appears to be a complete set of commands

Cons:

- Gives a whole lot of message formats, but doesn't really say how they work. While others have too much detail, this one definitely has not enough, and one is left wondering what the different commands actually do

Reza: 4 of 5

Pros:

- Nice, simple, rational protocols that make sense. Will be fairly easy to implement.

Cons:

- Says servers try to connect with other servers with fewer numbers of neighbors, but gives no protocol or message format for determining this. The given protocol seems to assume that a server already knows what servers it wants to connect to.