

Tutorial 6

Fault tolerance

1. A firewall is a mission-critical aspect of many organisations' computer infrastructure. It is proposed that a replicated firewall built of 3 computers be designed, to provide a fault-tolerant solution.
 - a. Discuss the extent to which each of the following apply to this example:
 - i. Triple modular redundancy.
 - ii. Single thread using non-blocking system calls.
 - iii. Availability, reliability, safety and maintainability.
 - iv. Byzantine failures.

In all cases, consider the likely nature of failures in this specific application, and include consideration of other levels of fault tolerance, e.g., in the network.

- b. Discuss advantages and disadvantages of a distributed solution over buying a purpose-design fault-tolerant computer to implement a highly-dependable firewall.
2. Explain why Byzantine failures require more replicated resources to deal with than silent failures.
3. Work through the Byzantine generals problem as in the lecture slides, but add a second traitorous general (anyone will do, but let's pick on general 2).
 - a. What happens?
 - b. What would happen if the traitorous generals were more consistent in their fibbing?
 - c. Does this problem correspond with any real situation?
4. Explain why network-related problems make it hard to implement transparent RPC.
5. Idempotence is a desirable property of network transactions in a distributed system.
 - a. Explain why requiring idempotence, in general, is not useful.
 - b. Explain workarounds of the problem of dealing with nonidempotent transactions.
 - c. Is handing in an assignment through an electronic submission system idempotent? Explain.
6. Explain why orphans are a difficult problem in the face of client crashes.
7. Multicasting in general has problems scaling.
 - a. Are the problems harder for distributed systems than for general networking? Explain.
 - b. A group of distributed processes needs to stop every now and then to exchange information. The following mechanism is proposed:
 - processes involved in a cooperative computation are members of a multicast group, and periodically multicast state information to the rest of the group

- when one process determines that the group has reached a checkpoint at which they should exchange more information, it sends a message to a process not already in the group, requesting that it join the multicast group
 - this new process joins the group, and waits for a message from all the other processes, indicating that they have sent their information to the other processes
 - when it has seen information from all the other processes, it leaves the group
- i. Discuss the role of *view changes* in this solution.
 - ii. How robust is this solution? Consider ways it could go wrong, and possible remedies.
8. Can you think of any situation (ordinary networking or computing, or distributed computing) where you have encountered *forward recovery*? Explain your example.