SELECT TOPICS OF DISTRIBUTED SYSTEMS SECURITY
(DRAFT LECTURE NOTES)

I. Byzantine Faults

So far: **fail-stop failures**: node crashes, network breaks or partitions.
Need 2f+1 replicas to tolerate f simultaneous fail-stop failures.
Two fault tolerance protocols: Paxos and RAFT.

Byzantine faults: node fails arbitrarily
- may perform incorrect computation
- may give conflicting information to different parts of the system
- may collude with other failed nodes

Why: - software bugs, hardware failures, malicious attack

Today: Byzantine Fault Tolerance (BFT): Can we provide state machine replication for a service in the presence of Byzantine faults?

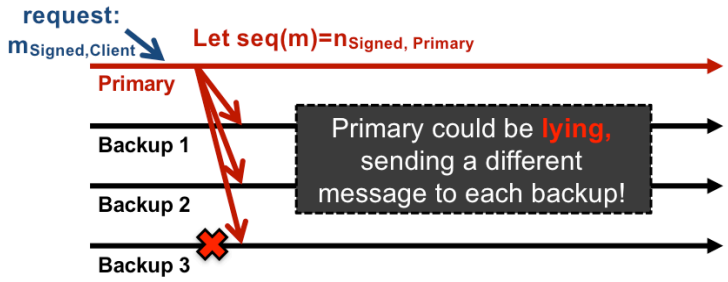
Traditional state machine replication (with Paxos)
- Requires 2f+1 = 3 replicas if f=1.
- Operations are totally ordered => correctness and consistency
- A two-phase protocol (last phase is just catching up partitioned nodes)
- Each phase waits for >=f+1 (i.e., 2 if f=1) of the nodes so you have overlapping quorums, i.e., at least one replica remembers a previous value.

(Multi-)Paxos for BFT? What could go wrong?
1. Can't rely on primary to assign seqno (he might assign the same seqno to different requests)
2. Can't just wait for f+1: the intersection of two majorities may be a lying node. E.g.:
    a. Suppose you have N0, N1, N2, and N1 is liar.
    b. N0, N1 form quorum for N0:1 and decide on value xyz. N2 doesn't hear.
    c. Then N2 proposes and forms quorum with N1. N1 accepts N2's proposal but doesn't send him value xyz on which he had already agreed to with N1. N1 doesn't hear about this. So N2 decides on abc and believes it has reached consensus with N1 on abc.
    d. At this point, N0 and N1 are decided but with different values. Contradition!
3. Other potential failures too: e.g., N1 could impersonate N0 and pretend that N0 answers something else than what he'd answer. A bad replica could also add new updates to the database that are not produced by any client. Etc. etc. etc.

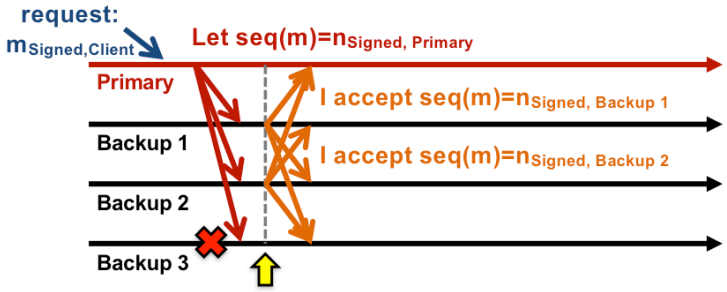
Practical Byzantine Fault Tolerant Protocol (PBFT, Castro & Liskov, OSDI 1999)
Key differences:
1. Messages are authenticated (signed) s.t. everyone knows who sent hem & can do the accounting of responses correctly. Also, signatures prevent repudiation.
2. Majority = 2f+1 (super-majority) now because you need to know that at least f+1 non-faulty nodes have responded. Intersection of any 2f+1 super-majorities formed at different times will result in *at least 1 non-faulty node, who can "remember"* any past agreed upon version. This is so you can handle both f faults and potential partitions.
3. Protocol works through broadcast b/c no individual node (including the leader) can be trusted.

Goal of the protocol is to have all non-faulty replicas eventually agree on: (1) set of requests and (2) order of these requests.

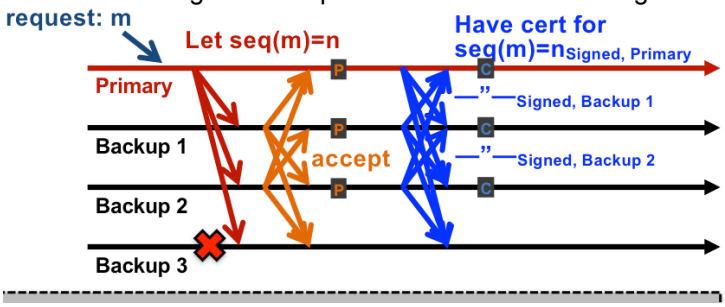
Phase 1: **Pre-prepare:** Primary assigns a sequence number to a client request and forwards it to all the replicas (this is the standard primary/secondary architecture).



Phase 2: **Prepare:** replicas exchange/disseminate request they got from the primary to each other. They need to exchange these messages because the leader cannot be trusted, so he could have sent n, m to one set of replicas and n, m' to another set of replicas. Replicas wait to receive 2f+1 confirmations for the same <n, m>. If they get this, they are "ready to commit."



Phase 3: Commit: replicas confirm to one another that they are ready to commit (i.e., they have confirmed the <n,m> mapping with enough other replicas). Replicas wait to receive "commit" confirmation from 2f+1 replicas. Each node that gets this replies to the client confirming that he's reached consensus for <n,m>.



Termination: Client waits for f+1 replicas to send him confirmation of commit before he continues operation.

For more information, refer to the practical byzantine fault tolerance paper (not necessary for final): http://pmg.csail.mit.edu/papers/osdi99.pdf.

II. Node Authentication, Key Management, and Other Topics

NOTE FOR 2019 DS1 EDITION: Notes are not yet available for broader topics of security in DS, including authentication, key management, distributed data protection, and other critical security topics. Therefore, NONE of these topics will be subject to the final quiz.

However, if you are interested in the topics I covered briefly in the make-up lecture, Fred Schneier has good references for Needham-Schroeder and Kerberos in his Systems Security course at Cornell:

- Needham Schroeder: http://www.cs.cornell.edu/courses/cs5430/2019sp/L24.new.html
- Kerberos: http://www.cs.cornell.edu/courses/cs5430/2019sp/L25.Rev.html

Also, a good, high-level primer on infrastructure security was published by Google:
https://cloud.google.com/security/infrastructure/design/

Acknowledgements

Pictures for BFT protocol are from Kyle Jamieson's Distributed Systems course:
https://www.cs.princeton.edu/courses/archive/fall16/cos418/docs/L9-bft.pdf