6.5840: Byzantine Fault Tolerance

Lecture 21

Last week: security

Lecture 19: Fork Consistency (SUNDR)

Lecture 20: Decentralized payments (Bitcoin)

Last week: security

Lecture 19: Fork Consistency (SUNDR)

Lecture 20: Decentralized payments (Bitcoin)

- Can we replicate state in an open system?
- Solved problem thought to be impossible!

Last week: security

Lecture 19: Fork Consistency (SUNDR)

Lecture 20: Decentralized payments (Bitcoin)

- Can we replicate state in an open system?
- Solved problem thought to be impossible!

But, limitations…

- Throughput: ~1K pay/min Latency: ~1hr (6-block depth)
- No linearizability guarantee!

Idea: RSM? (Raft)

But Raft has a linearizability guarantee…

- Network problems: wait to recover

Can we implement a RSM with malicious replicas?

- Throughput: ~1K pay/min
- Latency: ~1hr (6-block depth)
- No linearizability guarantee!

Can we implement a RSM with <u>malicious</u> replicas?

"*Byzantine*"

(Lamport, Shostak, Pease '82)

Practical Byzantine Fault Tolerance (Castro + Liskov ’99)

- “Academic problem”
- Ancestor of many of today’s cryptocurrency protocols

Can we implement a RSM with <u>malicious</u> replicas?

“*Byzantine*”

(Lamport, Shostak, Pease ’82)

Aside: About me

- Class project: Implement PBFT
  (6.5840 final project)

- Job: Implement BFT protocol at company
  (Algorand, Inc.)

- PhD project: Implement PBFT without bugs
  (Formal verification (c.f. Ironfleet, 6.5120)
  w/ Frans Kaashoek + Nickolai Zeldovich)

PBFT solves harder problem than Raft

Similar idea: RSM, but with malicious replicas

- Leaders ≈ Primaries
- Terms ≈ Views
- Timeouts

PBFT solves harder problem than Raft

Similar idea: RSM, but with malicious replicas

- Leaders ≈ Primaries
- Terms ≈ Views
- Timeouts

Additional ingredients needed

1. Authenticity of messages
2. More honest nodes
3. Leadership “fairness”

Assumptions

- Network
  - Attacker can reorder messages
  - Attacker can delay messages for *limited* time (denial of service)
- Nodes
  - Attacker controls f server replicas
    - Detail: client honest in paper
  - Cryptography protects messages of honest machines

Aside: Digital signatures

(see 6.5610 or 6.5620 for details)

- *KeyGen*(randomness) → (public key, private key)
- *Sign*(private key, message) → signature
- *Verify*(public key, message, signature) → ok?

Properties:

- **Correctness**: signature from non-faulty node ⇒ *Verify* ok
- **Security**: *Verify* ok ⇒ signature from non-faulty node

Assumptions

- Network

  Realistic?

  - Attacker can reorder messages
  - Attacker can delay messages for limited time (denial of service)

- Nodes
  - Attacker controls f server replicas
    - Detail: client honest in paper
  - Cryptography protects messages of honest machines

Let’s build a PBFT

Start with one client

One-client protocol

(inspired by Baudet, Danezis, Sonnino ’20)

Cannot tolerate >=N/3 faults! (i.e., N >= 3f+1)

Quorum: any Q where |Q| >= 2f+1

- **Safety**: Every Q has an honest majority (therefore unique)
- **Liveness**: |Q| <= N - f (no dependency on faulty replicas)

Multiple clients

Need to elect a *primary*

Problem: what if the primary is faulty?

Faulty primaries

These make PBFT expensive

Recovery example

Faulty primaries impose requirements

Add new all-to-all *prepare* round

Recovery messages must be justified: signature *stapling*

Multiple views

Like Raft: need primary with sufficiently-fresh view

(NULL ~ view 0)

Unlike Raft: (liveness) can't just elect any eligible candidate!

⇒ "term limits": rotating primary

⇒ timeouts: *View-change* must be all-to-all as well

(exponential)

Extending from op to RSM

- (Like Raft): Primary pipelines many client requests
  - Low- and high-water mark prevent sequence # exhaustion
  - Each view is a new “log”
- Checkpoints allow log compaction (c.f. Raft)
  - *Commit* also all-to-all
- Clients get f+1 replies
  - Sufficient: contains at least 1 honest replica

Optimizations

- Hash of values (c.f. Bitcoin)
- Tentative replies
- Read-only operations
- MACs vs. signatures
- Network NACKs

PBFT: RSM, securely

- Redundant state redundantly replicated for redundancy
    - Never lose evidence of *commit*
- Cryptography limits attacker's influence
    - Under some assumptions!
- Design limits *any node's* influence
    - A good idea for robust systems, digital or not

PBFT: RSM, securely

- Redundant state redundantly replicated for redundancy
  - Never lose evidence of *commit*
- Cryptography limits attacker's influence
  - Under some assumptions!
- Design limits *any node's* influence
  - A good idea for robust systems, digital or not

Questions?