Reliability, distributed consensus & bitcoin

COSC312 / COSC412

Learning objectives

- Consider **reliability** as a key part of computer security
- Encourage you to **always design for failure**
- Appreciate how **decentralised consensus** helps security aspects such as **reliability** & **non-repudiation**
- Gain an initial view of **blockchain approaches** and how they support bitcoin, and other emerging **decentralised autonomous systems**

Securing valid results on fallible machines

- Digital devices suffer **(non-malicious) failures**
  - RAM corruption errors—c.f., ECC memory
  - Storage media may **fade or malfunction**
    - Beware cheap writable optical media or flash storage
    - SSD devices fail very differently from magnetic hard drives…
- Also may have vulnerability to **critical software failures**:
  - filesystem bugs
  - compression library bugs
  - system use contrary to supported operation

One solution: rerun your computations

- If you can estimate the **probability of failures**, you can determine how many trials of a computation you need to **achieve a given level of confidence** in the result
    - Excessive system failures may become overshadowed by other concerns anyway…
- Of course multiple trials need not be run in serial:
    - can structure **repeatability within a software service**
    - cloud computing provides convenient **elasticity for parallelism**

Aside: machines designed to fail frequently

- Computers have **adjustable reliability**
  - Can trade off against speed, power consumption, *etc*.
  - Consider the practice of **overclocking CPUs**:
    - may need to apply CPU voltage adjustments;
    - may affect reliability of computation—possibly catastrophically!

- Computer participates in a group repeating results?
  - Can purposefully **design such a computer to be less reliable**
  - May end up with a net saving in this resource trade-off

Distributed consensus—trustworthy results

- Common in more than just storage systems, *e.g.*:
  - Primary/primary relational **database server replication**
  - NoSQL: *e.g.*, use of gossip protocols and eventual consistency
  - Network infrastructure such as **routers with hot spares**

- Systems now exist that just handle **consensus gathering**
  - *e.g.*, Apache ZooKeeper, etcd offer distributed synchronisation
    - **Apache ZooKeeper** used in other systems: Hadoop, HBase, …
    - **etcd** used as main configuration database in Kubernetes

Apache ZooKeeper & etcd

- Essentially multi-server, **key-value database** systems
  - However, emphasis is on **correctness and synchronisation**
  - **ZooKeeper** introduced to Hadoop to address complex failures: coordinates & manages scheduling of map-reduce tasks
  - **etcd**, e.g., facilitates updating clusters without breaking them

- Key property: **facilitates atomic broadcast**
  - Under atomic broadcast all correct processes in a distributed system receive the **same sequence of events, or all abort**

Distributed consensus algorithms

- Fischer Lynch Paterson impossibility result (1985):
  - Consistency protocols pick 2 of: safety, liveness, fault tolerance
- **Paxos**: fault tolerant consensus over distributed nodes
  - Used widely, including within Apache ZooKeeper
- **Raft**: alternative to Paxos, used by etcd
  - Raft algorithm easier to understand and implement than Paxos
    - Sub-problem 1: leader election
    - Sub-problem 2: log (*i.e.*, data) replication by leader to followers
- **EPaxos**: more complex and efficient than Paxos

Add in potentially malicious parties

- ZooKeeper, etcd are used when **we trust all servers**: *e.g.*, they are owned by one organisation
- When **malicious parties** may be participating, the consensus set size must grow
  - Need a majority of votes from the **assumed-benign** server set
- Could we choose **not to control the server set**?
  - Enter **permissionless blockchains**, *e.g.*, bitcoin
  - Safety presumed if 50% of nodes are benign (isn't quite right!)

Different types of fault tolerance

- **Crash fault tolerance (CFT)**
  - Crash faults are what the name suggests: a node disappears
  - CFT usually tolerates $\frac{N}{2}$ failures within N distributed nodes
    - So a majority of nodes must agree
- **Byzantine fault tolerance (BFT)**
  - Byzantine faults include nodes acting maliciously
    - Malicious node may be actively trying to break a given protocol
  - BFT usually tolerates $\frac{N}{3}$ failures within N distributed nodes
- Raft & Paxos are only CFT; variants of Paxos are BFT

Warm up exercise: build a cryptocurrency

- How do we make a **cryptocurrency ‘coin’**?
- How do we identify **coin owners**?
- How can we protect the system from **forgery**?
- How do we record ownership and **transfer of ownership**?

- Can copy digital assets perfectly, so how can coins be single-use?

Distributed consensus needs within bitcoin

- To work, currencies need to track **who** has **what**
  - Normal currency uses TTPs such as mint, banks, *etc*.
- bitcoin has all validating nodes store the **whole ledger**
  - Distributed ledger is **sequence of blocks** of transactions
  - Collectively agreeing transaction order **avoids double-spending**
- A **wallet** is a hash of a public key a client generates
  - Own private key? Can prove your connection to transactions
    - … don’t actually need a representation of ₿ apart from ledger

Proof of work—validate Ƀ transactions

- Must **protect validation from Sybil attacks**, so:
  - Make it computationally costly to incorporate new transactions
    - move to **how much computing power you control**, not just the number of identities that you control (*i.e.*, the basis of Sybil attacks)
  - Make it **rewarding to incorporate new transactions**—more later
- Validator collects transactions into a block
  - **checks transactions internally** first—could be double spending
  - **forms Merkle tree** over transaction hashes (see later slides…)
  - to close off the block, it **applies proof of work** algorithm

bitcoin transaction validation

- Proof of work must be **easy to check**; **hard to compute**
  - In some ways like a hard-to-apply digital signature
- bitcoin: must **find a nonce** that when appended to the block of transactions+ gives a **hash value less than target**
  - SHA-256 hash function used, specifically
  - Target is dynamic: ensures blocks take ~10 minutes to compute, regardless of changes in net computational resources available
- Mid-August 2023: bitcoin blockchain is about 504.38 GB

+ and other things we are not talking about

Blockchain approaches predate bitcoin

- Blockchain because new block **includes hash of previous block**
  - Thus records' integrity checks are linked together sequentially

- Linked hashes widely used before bitcoin (2008), e.g.:
  - Git (2005) **chains hashes** to preserve integrity of whole history
    - (Git's rebase operation can be disruptive: hashes get changed)
  - Solaris ZFS (2006) forms **trees of hashes** to confirm the integrity of stored files and folders
    - Let's explore Merkle trees in more detail…

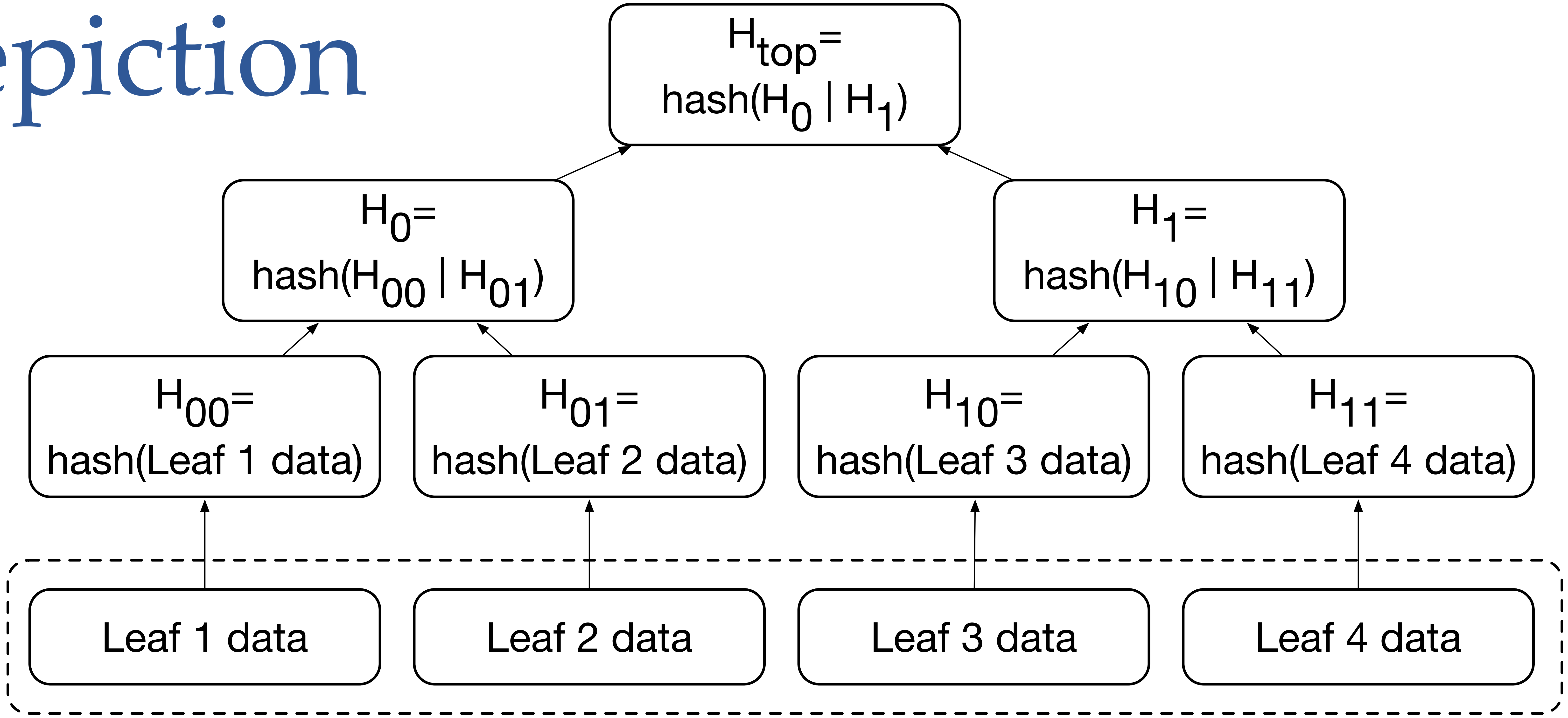
Merkle tree: efficient integrity checking

- Consider a set of data blocks D_i, then:
  - A hash value is computed for each data block D_i
  - A tree is built, with **parent hash hashing hashes of its children**
  - The root hash will thus summarise all the data blocks

- Checking hash on particular D_i can be done cheaply
  - **Get trusted root hash**; other hashes can **come from anywhere**
  - Used within Bittorrent to check blocks retrieved build valid file
  - Also with ZFS, within bitcoin transaction blocks, *etc*.

Merkle tree depiction

- **Leaf data blocks** may be any size
- All **upper blocks** H_{xx} are fixed size
- Need good H_{top}
  - After that, H_{xx} **from untrusted sources** OK: still integrity-checked
- Secure implementation needs a few more details
  - e.g., H_{xx} blocks must not be able to be passed off as leaf data



Validators, mining, fees and the network

- Bitcoin **miners** are carrying out **validation of blocks**
- Two incentives for miners to solve block hash task:
  - **reward** of 6.25 bitcoin since May 2020; around NZ$216,000 (ish)
    - value halves periodically; was 50Ƀ in 2009!
    - by 2140 CE no further bitcoin increase
  - **ability to levy fees**—commercial competition applies

- Broadcast communication between miners uses a peer-to-peer protocol
  - avoids central infrastructure… and knowing the miner set (!)

Results from block validation

- Rate is **~10 minutes**, but this is probabilistic
  - *e.g.*, might guess an appropriate nonce first off (if really lucky)
- Automatically **helps serialisation**: variance in mining time is larger than the message broadcast time
  - Miners want to **publish results ASAP** so to receive payment
  - (Some potential attacks do involve holding back a solution.)
- Still **possible for multiple answers** to be broadcast, so…

Blockchain forks need to be resolved

- When **nodes hear multiple solutions** they keep them all
- Subsequent mining is only **done on your longest fork**
  - Extremely unlikely that parallel forks will continue for long
  - **Software bugs can cause long-lived forks**—have happened!
  - Probability distribution likely to clearly favour one branch
- Attacker with significant resources can try to keep fork alive, but cost, coordination and probability won't help
  - (Some attacks involve late revealing of privately mined forks.)

How/when is a transaction approved?

- Clearly the **transaction has to be recorded** in a block

- Two simple rules are applied:
  - Relevant block must be in the **longest fork of blockchain**
  - **Five or more blocks** must already follow it in the blockchain

- This causes a **transaction clearing delay** (in effect)
  - Consider possible attacks, *e.g.*, partitioning of network
  - Probably impractically difficult to effect

Conclusion

- Failures can threaten security by **affecting availability**
  - Hardware and software problems

- Efficient means exist to reach **decentralised consensus:**
  - **Merkle trees** for checking integrity
  - **Apache ZooKeeper**, and **etcd**, within a known set
  - **Proof-of-work within permissionless blockchain** such as bitcoin

- Discussed at high level **blockchain** & **how bitcoin works**