CS 686: Special Topics in Big Data

Proof-of-Work

Lecture 15

Today's Agenda

- Proof-of-work systems
- Hashcash
- Bitcoin

Today's Agenda

- **Proof-of-work systems**
- Hashcash
- Bitcoin

Proof-of-Work

- Proof-of-work (**POW**) systems help prevent DDoS attacks and other types of spamming
- Also useful in cryptocurrencies
- Give up some of your time (or computational power) to legitimize an action/object

Shell Money

- Sea shells were used for thousands of years as legal tender
- It takes time to collect shells, carve them, etc.
  - In some cases, the shells were woven into fabric/leather
  - The currency itself reflected the time it took to be made, and therefore determined its value
- Different groups used different shells/designs
  - Only carry value because we say so

CAPTCHA

- **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part
- CAPTCHAs are basically proof-of-work systems for humans
- So in other words, POW is an annoying, time consuming task for your computer to do just in the interest of proving it's not spamming/DDoSing
  - Luckily computers don't get annoyed as easily as we do…

Pricing Functions

- POW systems use **pricing functions** to give the computer a workout
- A pricing function **f** has the following requirements:
  - **f** is moderately easy to compute
  - **f** is not amenable to amortization: given **L** values, $m_1 \ldots m_L$, the *amortized* cost of computing $f(m_1) \ldots f(m_L)$ is comparable to computing $f(m_i)$ for any $1 <= i <= L$
  - given **x** and **y**, it is easy to determine if **y** = **f**(**x**)

Dwork C., Naor M. *Pricing via Processing or Combatting Junk Mail*.

Hash Inversions

- A common pricing function is having the computer perform **hash inversions**
  - What was the **input** that produced this hash code?
- Hash inversions are tough to compute (assuming a cryptographic hash function)
  - After all, they're designed to be **one way** functions
  - Any time we map an infinite set of inputs to a finite set of numbers (hash space), this is feasible, but still tough.

An Example (1/2)

- Let's say our mission is to find a hash with four leading zeros
- Start out with what we want to send:
  - "Hello World!"
- We also need to append a **nonce**
  - Number used only once
  - We increase this with each hash attempt
  - This will change our output hash each iteration

An Example (2/2)

- This approach allows us to eventually find our matching hash, but has a weakness
  - We can precompute the hashes and re-use them later
- We also need some type of identifier for this particular transaction
  - Maybe a centralized service hands out transaction IDs
  - We could use the current time, as long as we can assume clocks are reasonably synced up

Pseudocode – Pricing Function

```
while True:
    nonce = nonce + 1
    string = message + str(nonce)
    hash = sha256(string)
    if prefix(hash) == '0000':
        # Send message with hash
        break
```

Pseudocode - Verification

```
if sha256(msg.payload) == msg.hash:
    # Valid… Whew! That was tough!
    # (You could also verify the
    # transaction id or timestamp here)
```

Varying the Difficulty

- To change the difficulty, we'll just adjust the number of zeros we want
- Unfortunately, the difficulty won't increase linearly
- Approaches:
  - Perform a bitwise comparison rather than string (allows more precision)
  - Have the sender perform multiple inversions (maybe message1 + another nonce)

Proof-of-Work Variants

1. Challenge-response
2. Solution verification

Challenge-Response

- Interactive link between sender and receiver
- Receiver chooses the challenge
  - Can adapt the challenge based on its own load or remote capabilities
- After completing the challenge, the service can be accessed by the sender

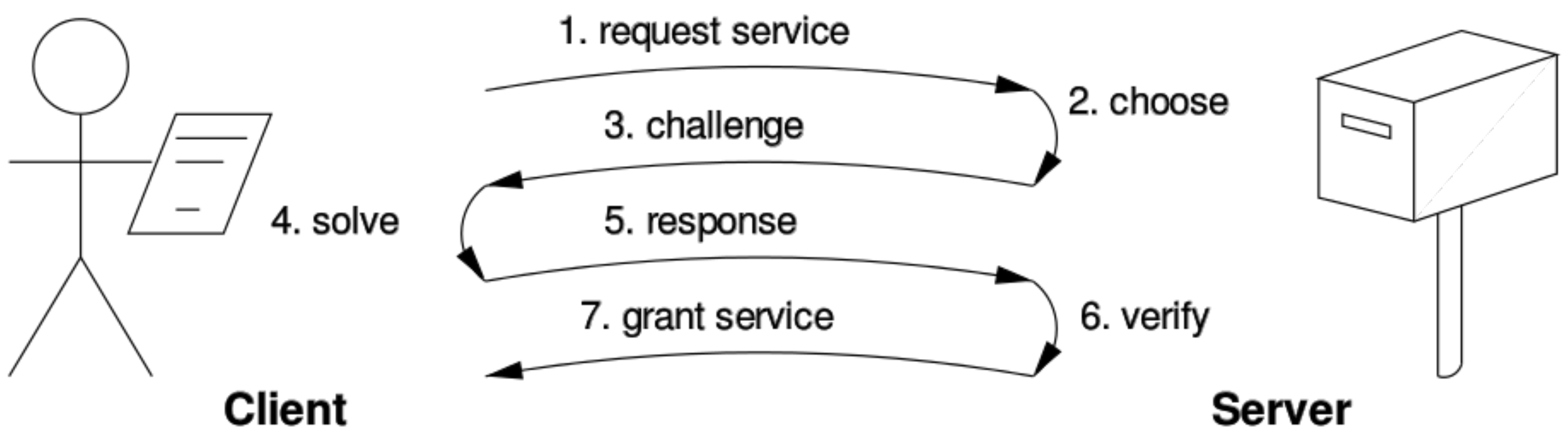

Figure Credit: Fabien Coelho

Solution Verification

- In this case, we already know what the challenge is in advance
  - Compute it locally and send the message
- Receiver needs to verify the message and then process it
  - Better for one-time communication

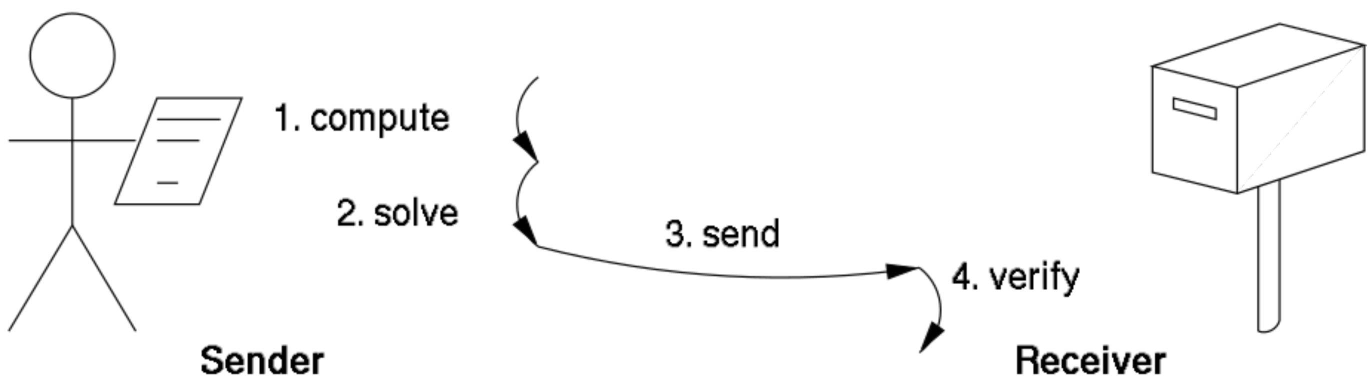

Figure Credit: Fabien Coelho

Other Challenges: I/O

- Hardware improvements in this space have traditionally been less rapid
  - Helps us avoid reconfiguring the difficulty frequently
- This could be a memory-intensive algorithm
- Another approach is requesting tokens from remote servers
  - Here the work is "tough" because of latencies rather than the computation

Today's Agenda

- Proof-of-work systems
- **Hashcash**
- Bitcoin

Hashcash

- Adds a new header to emails in an effort to reduce DDoS/spam
  - Proposed by Dwork and Naor
- You can install the **hashcash** command line utility:

```
$ hashcash -m 'mmalensek@usfca.edu’
$ hashcash stamp:
1:20:170927:mmalensek@usfca.edu::ZeNi+DkIeFrH3aUl:000
00000000000000000000000000000000000000000009nfO
```

- On the receiving end, all that has to be done is verify the SHA-1 hash of the header

Header Fields

- `1:20:170927:mmalensek@usfca.edu::ZeNi+DkIeFrH3aU1:00000000000000000000000000000000000000000009nf0`

- *ver*: Hashcash version
- *bits*: Number of zero bits
- *date*: The time that the message was sent: YYMMDD
- *resource*: Resource data string being transmitted
- *ext*: Extension (currently ignored)
- *rand*: String of random characters
- *counter*: Nonce

Sending a Message

- The sender performs the hash inversion and prepares the header
  - This takes a little CPU time, but shouldn't be noticeable
- Adds the header to the email message
- Performs the send operation as usual

Receiving a Message

- On the receiving side, all we need to do is compute the SHA-1 hash of the entire Hashcash header
- Then we check:
  - That the correct number of leading zeroes is present
  - The provided date is valid
- This takes an imperceptible amount of time

Why Hashcash Works

- Even heavy email users only send a few hundred emails per day
- Spammers want to send millions
  - This is going to cost a lot of CPU time
- Additionally, sending an email with no header or an incorrect header will incur steep penalties
  - Too many incorrect headers? Ban the IP
- Best of all, we don’t have to start paying for email

Why it Doesn't Work (1/2)

- Back in 1992 when Hashcash was invented, we didn't have such a huge variety of computing hardware
  - Smartphones, tablets, refrigerators, etc.
  - This makes coming up with the right difficulty for the challenge… difficult.
- The power of computing hardware isn't distributed uniformly across the Earth
- Hash inversions are amenable to parallelism and custom hardware

Why it Doesn't Work (2/2)

- Spammers could adopt similar hardware to that of Bitcoin miners
  - GPUs, ASICs
  - Depends on cost vs. benefit
  - Related: cloud instances. Computing is **so** cheap!
- Since email is decentralized, you can't force everyone to use this new standard
  - Would actually be easier nowadays (get Google and Microsoft on board, and you're just about done)

Today's Agenda

- Proof-of-work systems
- Hashcash
- **Bitcoin**

Bitcoin Value, Sep 27, 2017

As of Now

- 1 BTC = 4,090.75 USD
- 273,158 transactions per day
- ~17m bitcoins in circulation
- See: http://blockchain.info

Blockchain

- The Bitcoin **blockchain** is a decentralized database of Bitcoin transactions
- Each block in the chain includes the hash of the previous block
- Starts with the **genesis block**
- When a transaction occurs, it is added to the current block and will be verified by miners

Blocks

- A **block** is a list of transactions with some metadata
- Magic number (4 bytes) = 0xD9B4BEF9
- Block size (4 bytes)
- Block header
- Transaction counter
- Transaction data

Block Headers

- Version
- Hash of the previous block
  - This makes tampering with the chain difficult
- Current hash of the transactions in the block
- Timestamp (last update)
- Difficulty
- Nonce

Agreement

- As we've seen, we don't always agree in our distributed systems
- Bitcoin allows **forks** off of the current block
- Whichever fork is acknowledged and used by the most participants becomes the "true" path
  - Longest path wins
- Transactions that went to a "failed" fork are added back to the "true" blockchain

Reaching an Agreement

- Provisions are in place to ensure transactions are dealt with in a reasonable amount of time
  - Target: 10 minutes
- Every 2,016 blocks the system automatically adjusts its difficulty to hit the 10-minute target
- From 2014 - 2015 the average number of nonces tried before a new block could be created increased from 16 quintillion to 200 quintillion

Mining Bitcoin

- Bitcoin uses the Hashcash algorithm for a different purpose: **mining** coins
- "Mining" means verifying a block of transactions
  - Finding the nonce (aka **solution**)
- Miners, who are the basis of transaction verification, are paid in new bitcoins and transaction fees
  - The reward of new bitcoins is halved every 210,000 blocks (~4 years)
  - Monetary supply limited to 21m bitcoins

Verification

- In bitcoin, the **difficulty** of the challenge is varied to keep the network chugging along
- Once all 21m bitcoins are created, miners will be rewarded for verification via transaction fees only
- What is the cost vs. benefit of mining these coins?
  - Electricity vs. the size of the reward
- Lots of companies now build power-efficient hardware specifically for mining

Pooled Mining

- As difficulty goes up, the chances of a single miner verifying a block goes down
- To combat this, **pools** of miners formed
- Pools divide up the work (nonces) among participants
  - Rewarded with a **share** of new bitcoins based on how much work was done
  - Less wasted effort, but less reward

Moral Issues

- We are consuming massive amounts of fossil fuels to produce fake money
  - Production is only hard because we make it so
- Mining hardware gets bought up and then discarded once we move to harder hash inversions
- Some *Useful Proof-of-Work* systems try to do beneficial work
  - Finding prime numbers (**Primecoin**)
  - Protein folding (**Curecoin**)