Principals and Practice of Cryptocurrencies

Cornell CS 5437, Spring 2016

Project
Project of Your Choice

• Simulate
  • Study the behavior of protocols
• Experiment
  • Measure properties of implementations
• Build
  • Add features to existing clients
Example:
Advanced Selfish Mining Simulation
Selfish Mining

Goal: Get more than fair share
How: Maintain secret blocks, publish judiciously

Intuition: Risk some work, others waste a lot
Known Results

• Selfish miner vs. single miner: threshold at 1/3
• Selfish miner vs. many small not well connected miners: threshold at 0
• Selfish miner vs. many small miners with fix: threshold at 1/4

What about other distributions of honest miner sizes?
Project Goal

Study selfish mining with various miner size distributions
Project Parts

• Understand the attack and known bounds
  • Read paper, reproduce math
• Formal analysis
  • Make approximations for new model
  • Analyze attack under new model
  • Instantiate results for different cases
  • Confirm edge cases with known results
• Simulation
  • Obtain pool sizes from measurements
  • Choose approximations
  • Design simulation
  • Confirm edge cases with known results
  • Confirm formal analysis
Example:
Propagate Pruned Blocks
Background

• Main chain is longest one
• Off-chain blocks are pruned
• They are not propagated
  • Both in theory and in the standard Bitcoin implementation, Bitcoin Core.

• Important data is lost – for system security and fairness analysis
Project Goal

Implement pruned blocks propagation as a patch to the standard Bitcoin client
Project Parts

• Understand current system behavior
  • Relevant data structures and their update
  • Communication protocol
• Implement changes
  • Change data structures
  • Change communication protocol
    • In a backward-compatible way
• QA
  • Extensive regression testing
  • Implications on security (mostly DoS)
  • Implications on performance
Example:
Simulate Protocols
Blockchain Protocols

Bitcoin – longest chain

GHOST – heaviest tree

Bitcoin-NG – longest chain, different block types
Simulate various protocols and compare their properties
Project Parts

• Study protocols and understand design choices
  • Bitcoin
  • GHOST
  • Bitcoin-NG
• Simulate
  • Make approximations
  • Construct **modular** simulation environment
  • Implement the protocols and measure
  • Confirm edge cases with known results
Example:
Client Profiling
Background

• A cryptocurrency client has many tasks
  • Send and receive state
  • Verify state
    • Transaction signatures are correct
    • Data structure correctness (e.g. block PoW)
  • Store state for crash tolerance

• Client speed is critical to system behavior
  • Limits bandwidth (transactions per second)
  • Limits propagation speed
    • Implications to security
    • Implications to performance
Project Goal

Evaluate client bottlenecks
Project Parts

• Locate potential bottlenecks
  • Storage
  • Communication
  • Processing
• Create experiment environment
  • Instrument code to measure
  • Create workloads (synthetic / real)
• Measure
  • Run experiments
  • Analyze results
  • Estimate speedup by replacing bottlenecks with artificial delays
Example:
Network Structure Simulation
Background

• Bitcoin uses a unique network topology construction
  • Gossip-based
  • Unstructured
  • Robust
• Best specification (I’m aware of) in Heilman et al.’s *Eclipse Attacks on Bitcoin’s Peer-to-Peer Network.*
Project Goal

Study Bitcoin’s network topology
Project Parts

• Understand the protocol
  • Standard operation
  • Edge cases (e.g., DNS bootstrapping)
  • Dynamics

• Simulate
  • Decide on approximations
  • Implement simulator and protocol
  • Study behavior in various scenarios
    • Network sizes
    • Latencies
    • Under attack
Other Examples

• Add a transaction script operation
  • Analyze security implications
  • Implement
  • Implement use cases
  • Add regression tests
• Change a cryptocurrency’s chain selection rule
• Tune cryptocurrency’s parameters (e.g., block size, frequency)
Logistics
Plan

- Teams of 4 (four, explicit permission in advance otherwise)
- Different leader per phase: coordinates and reports
- Delivery: 15min meeting, lead by leader, choose time in advance, all should be ready to answer questions

<table>
<thead>
<tr>
<th>Phase end</th>
<th>Task</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feb 8-12</td>
<td>Make plan</td>
<td>Topic, architecture, mission assignment, timed plan</td>
</tr>
<tr>
<td>Phase 2</td>
<td>Check point</td>
<td>Results, plan updates</td>
</tr>
<tr>
<td>Phase 3</td>
<td>Check point</td>
<td>Results, plan updates</td>
</tr>
<tr>
<td>May 9-11</td>
<td>Report</td>
<td>Report + meeting + class presentation</td>
</tr>
</tbody>
</table>
Make Plan

- Topic choice: confirm with me in advance by email
- Architecture: high-level but concrete design
- Timed plan:
  - **Flow diagram of tasks with nice little deliverables**

```
run client

issue standard txn

manually create special transaction

Issue 1000 standard txns

Issue 1000 special txns

Issue 1000 standard txns
```
Mission Assignment

- Parallelize
- Assign tasks by preference/specialty

- run client
- manually create special transaction
- issue standard txn
- Issue 1000 standard txns
- Issue 1000 special txns
Checkpoints

• Report on finished tasks
• Report on changes to plan (same level of detail)
  • Tasks that took longer than expected
  • New tasks you didn’t foresee
Final Report

• Paper report (1-2 pages, appendix if necessary)
  • Write it as you go
• Class presentation
  • 5-15 minutes, by phase leader
• Meeting
  • As in previous phases
Logistics

• Be thorough with planning
  A few days of programming can save you hours of planning
  Also – 50% of the project grade
• Coordinate frequently and efficiently
• Help the phase leader
  • Respect intermediate deadlines
• Use a distributed version control system (e.g. Git, Mercurial)
• Choose the right tools.
  (matlab/excel, matlab/python, python/java...)
• Ask for help when unsure: email, office hours.
Grade

• Total of 60%:
  • 50% planning
    • Decomposition to tasks
    • Work division and time planning
    • Group effort (talk to me)
  • 50% result
    • Code quality (structure, documentation)
    • Result (efficiency, analysis)
    • Report
• Factored per individual if necessary