RepuCoin: Reputation-based Byzantine Consensus

Jeremie Decouchant,
Joint work with Jiangshan Yu, David Kozhaya, Paulo Esteves-Veríssimo
CritiX, SnT
jeremie.decouchant@uni.lu
RepuCoin: addressing the 51% attack
RepuCoin: addressing the 51% attack

RepuCoin: Reputation-Based Byzantine Consensus

1. Introduction

A 51% attack is the most recorded Bitcoin's attack. According to the 2013 Bitcoin price information, the most recorded Bitcoin's attack was by a 51% attack. According to the 2013 Bitcoin price information, the most recorded Bitcoin's attack was by an attacker that controlled more than 51% of the mining power. In a 51% attack, an attacker can control the network's consensus by controlling more than 51% of the mining power. In a 51% attack, an attacker can control the network's consensus by controlling more than 51% of the mining power. In a 51% attack, an attacker can control the network's consensus by controlling more than 51% of the mining power.
1. Miners gain reputation by contributing to the blockchain

2. Only top reputed miners can vote through a BFT protocol (e.g., PBFT)

3. Mis-behaved miners will be punished, and they lose reputation

4. Leaders are randomly selected from top reputed miners to propose transactions
RepuCoin – Increased attack resilience

• For an adversary to build enough reputation takes time

<table>
<thead>
<tr>
<th>Joining time\ Target</th>
<th>1 week</th>
<th>1 month</th>
<th>3 months</th>
<th>6 months</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 month</td>
<td>infeasible</td>
<td>45%</td>
<td>30%</td>
<td>27%</td>
</tr>
<tr>
<td>3 months</td>
<td>infeasible</td>
<td>90%</td>
<td>45%</td>
<td>33%</td>
</tr>
<tr>
<td>6 months</td>
<td>infeasible</td>
<td></td>
<td>infeasible</td>
<td>68%</td>
</tr>
<tr>
<td>9 months</td>
<td>infeasible</td>
<td></td>
<td>infeasible</td>
<td>90%</td>
</tr>
<tr>
<td>12 months</td>
<td>infeasible</td>
<td></td>
<td>infeasible</td>
<td></td>
</tr>
<tr>
<td>18 months</td>
<td>infeasible</td>
<td></td>
<td>infeasible</td>
<td></td>
</tr>
<tr>
<td>20 months</td>
<td>infeasible</td>
<td></td>
<td>infeasible</td>
<td></td>
</tr>
</tbody>
</table>
RepuCoin – Increased attack resilience

• For an adversary to build enough reputation takes time

Breaking the liveness property:

<table>
<thead>
<tr>
<th>Joining time\ Target</th>
<th>1 week</th>
<th>1 month</th>
<th>3 months</th>
<th>6 months</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 month</td>
<td>infeasible</td>
<td>45%</td>
<td>30%</td>
<td>27%</td>
</tr>
<tr>
<td>3 months</td>
<td>infeasible</td>
<td>90%</td>
<td>45%</td>
<td>33%</td>
</tr>
<tr>
<td>6 months</td>
<td>infeasible</td>
<td>infeasible</td>
<td>68%</td>
<td>45%</td>
</tr>
<tr>
<td>9 months</td>
<td>infeasible</td>
<td>infeasible</td>
<td>90%</td>
<td>54%</td>
</tr>
<tr>
<td>12 months</td>
<td>infeasible</td>
<td>infeasible</td>
<td>infeasible</td>
<td>68%</td>
</tr>
<tr>
<td>18 months</td>
<td>infeasible</td>
<td>infeasible</td>
<td>infeasible</td>
<td>91%</td>
</tr>
<tr>
<td>20 months</td>
<td>infeasible</td>
<td>infeasible</td>
<td>infeasible</td>
<td>infeasible</td>
</tr>
</tbody>
</table>
A block was discovered in the network

The block is added to the blockchain

Byzantine Consensus

Block ordering
Implementation: BFT-SMaRt (Java)

https://github.com/bft-smart/library

3f+1 Miners

Consensus reached when:
- 2f+1 miners agree on a block
- They represent more than two 3rd of the group reputation

Byzantine Consensus
Performance evaluation

Measure
- Latency
- Throughput

Depending on
- Consensus group size
- Block size

Settings (in the code)
- Limit bandwidth
- Impose network latency
1. Find a set of machines on a single cluster
2. Create an interactive job and connect to it

   $ oarsub –I –l nodes=13,walltime=1:0:0
   $ oarsub –C 12345

3. Edit BFT-SMаRt’s config files (machines to use and port)
   $ cat $OAR_NODEFILE

4. Bash: script to run the throughput/latency benchmark
   – Kill any java application on the machines
   – Launch replicas
   – Launch clients

   $ oarsh –f ${ip_addr} “cd bftsmart.repucoin; ./runscript > /dev/null 2>&1 &” &

5. Python: collect the results (output file) and plot
Best practices

• Search for the right code basis
  – Your life will be much easier

• Automate everything
  – You always think you won’t need to repeat the experiments: wrong!
  – The initial additional work is quickly amortized

• Latency vs. throughput experiments are tricky
  – The throughput should increase with the load up to a certain point, where the latency starts increasing
  – But too many requests make the applications crash (message queues)
  – Find the right number of clients
Lessons learned

• Estimate the time it takes for your experiment and double it
  – Plan ahead

• It is difficult to be on a completely controlled environment
  – Change the machines → Change your performance
  – Are there a lot of jobs ongoing?

• Performance is sometimes difficult to understand
  – Example: The performance with 8MB blocks is lower than with 4MB
  – I spent a day repeating the experiments and got the same result: I still don’t explain it