Sydney
Red Belly Snake
Red Belly Blockchain

Vincent Gramoli

(Special thanks to Tyler Crain, Mikel Larrea, Chris Natoli, Michel Raynal, Guillaume Vizier)
Roadmap

• Context: Blockchain
• Problem: Balance Attack
• Solution: Democratic BFT consensus
• Red Belly Blockchain
• Evaluation
What is a blockchain?
Blockchain [Nak’08]

- Chain of blocks
- Starts at a special block called the genesis block
- Each block contains the hash of its predecessor
- To create a block one must solve a crypto-puzzle
Participants can record transactions in blocks, to transfer digital assets (called coins) between participants.
Blockchain (con’t)

Secure: transactions are signed with private keys

\(\sigma_1(t_1)\)
\(\sigma_2(t_2)\)
\(\sigma_3(t_3)\)

P1
P2
P3

genesis
Blockchain (con’t)

Secure: transactions are signed with private keys
Disagreement \Rightarrow\text{ DAG}
Blockchain (con’t)
Blockchain (con't)

$\sigma_A(t_A)$: Alice gives all her coins to Bob

$\sigma_A(t_{A'})$: Alice gives all her coins to Carole
One branch is selected based on its length, the weight of its subtrees, its content...
Blockchain (con’t)

blockchain depth = \( i+k \)

We say that a transaction commits when it is in a decided block [NCA’16]
What's dangerous about it?
The Balance Attack \[\text{[DSN’17]}\]

Consider a communication graph $G=<V,E>$
Consider a communication graph \( G = \langle V, E \rangle \)

\[ G_1 = \langle V_1, E_1 \rangle \]

\[ G_2 = \langle V_2, E_2 \rangle \]

Let \( G_1 \) and \( G_2 \) be two subgraphs of \( G \) with the same mining power separated by \( E_3 \) such that \( E = E_1 \cup E_2 \cup E_3 \)
The Balance Attack (con’t)

Let an attacker issue $t_A$ in $G_1$ and delay links $E_3$

Then the two subgraphs end up with branches of similar lengths or weights $W_1$ and $W_2$
The Balance Attack (con’t)

The attacker has only to mine $|W_1-W_2|+1$ blocks in $G_2$

$G_1$

$E_3$

$G_2$

The weight of a branch in $G_2$ exceeds $G_1$’s branches
The Balance Attack (con’t)

When the attacker stops delaying the messages

$t_A$ is discarded, allowing double spending with $t_A’$
Analysis Balance Attack

Let $n$ be the number of hashes tested by a subgraph and let $p$ be the probability of solving a crypto-puzzle with a hash chosen u.a.r.

Let $\mu=np$ be the mean of the number of blocks mined on each subgraph.
Analysis Balance Attack

Let $\Delta = |W_1 - W_2|$ be the difference of the number of blocks mined in each subgraph.

The probability that $\Delta \geq 2\delta \mu$ is upper-bounded using Chernoff bounds [Motwani and P. Raghavan 1995] by

$$P[\Delta < 2\delta \mu] > 1 - 4 \exp(-\rho^2 \mu / (3(\rho-1)^2))$$

where $\mu$ is the number of blocks mined by correct and $\rho$ is the fraction of mining power owned by attacker.
Example with Banks

Consider an Ethereum/GHOST private chain with difficulty 30MH...
Example with Banks

Consider an Ethereum/GHOST private chain with difficulty 30MH and a mining power of 20MH/s.

~20MH/s
Example with Banks

Consider an Ethereum/GHOST private chain with difficulty 30MH and a mining power of 20MH/s.

Assume the attacker has 12% of the mining power, i.e., 2.4MH/s, so he can mine, in expectation, 97 blocks during 19 minutes 40 seconds...
Example with Banks

Consider an Ethereum/GHOST private chain with difficulty 30MH and a mining power of 20MH/s.

Assume the attacker has 12% of the mining power, i.e., 2.4MH/s, so he can mine, in expectation, 97 blocks during 19 minutes 40 seconds and that he can select E3 such that each subgraph has a mining power of 8.8MH/s.
Example with Banks

Consider an Ethereum/GHOST private chain with difficulty 30MH and a mining power of 20MH/s.

Assume the attacker has 12% of the mining power, i.e., 2.4MH/s, so he can mine, in expectation, 97 blocks during 19 minutes 40 seconds and that he can select E3 such that each subgraph has a mining power of 8.8MH/s.

Let us select $\delta = 0.136$ for the difference $\Delta$ to be lower than 97, which happens with probability $P[\Delta<97] > 53\%$
Unforkable Blockchain
We say that a transaction commits when it is in a decided block [NCA’16]
Model

- **Distributed system**: $n$ processes $\Leftarrow$ but additional processes can issue transactions and read the blockchain

- **Partially synchronous**: the upper-bounds on the delay of messages and computation is not known $\Leftarrow$ Internet can be congested

- **Byzantine failures**: there can only be $t < n/3$ arbitrary failures, all other processes are correct $\Leftarrow$ Attackers have incentives to try stealing

- **Point-to-point reliable channel**: a message sent by a correct process is eventually received by a correct process $\Leftarrow$ Public key encryptions can implement it
Byzantine Consensus?

Each correct process invokes propose(v) with its value v and decides the returned value such that:

1. **Agreement:** no two correct processes decide differently

2. **Termination:** every correct process decides

3. **Validity?**
Relation with Blockchain?

The values are the blocks and they can be invalid (not well-formed, containing badly-signed tx...)

- **Valid block** proposed by a correct process: Y
- **Valid block** proposed by a Byzantine process: Y
- **Invalid block** proposed by correct a process: N
- **Invalid block** proposed by a Byzantine process: N
Ignoring Byzantine Proposals

No values proposed only by Byzantine processes can be decided. [CJ06, JACM15, TPDS16]

Intr. Tolerant Cons. [CJ06, JACM15, TPDS16]

We would like to decide blocks from Byzantine processes
Deciding Invalid Blocks

If all processes are correct and propose $v$ then they decide $v$. [JACM88, Lyn96, MA06, CJ03]

We do not want to decide anything...
Deciding Invalid Blocks

The decided value that is validated has to be proposed [CRY01]

We would like to accept more values if possible
Deciding $n\cdot t$ Values

One has to decide $n\cdot t$ proposed values

[\textit{JACM80, PODC94}]

\begin{itemize}
  \item $Y_1$
  \item $N_1$
  \item $Y_2$
  \item $N_3$
  \item $Y_4$
  \item $N_2$
  \item $Y_5$
\end{itemize}

But we don't want to decide invalid blocks even from correct
Blockchain Consensus [AlgoTel’17]

Provided an application-specific valid() predicate, each correct process invokes propose(v) and decides the returned value such that:

1. Agreement: no two correct processes decide differently
2. Termination: every correct process decides
3. Validity: the decided value satisfies the predicate \text{valid}()
Deciding Valid Proposed Values

A decided value has to be valid() [AlgoTel17]

Blockchain Consensus [AlgoTel17]

We can decide all valid proposed values at once!
Democratic BFT
Binary Byzantine Consensus

Each correct process invokes propose(v) with its value \( v \in \{0,1\} \) and decides the returned value such that:

1. Agreement: no two correct processes decide differently

2. Termination: every correct process decides

3. Validity: if all correct processes propose the same value, then no other value can be decided
Safe DBFT \[CGLR'17\]

est: local current estimate of the value to decide

\(r\): local round number, initially 0

bin-values[]: array of binary values for all rounds

b, auxiliary binary value

values\(_i\), auxiliary set of values

The algorithm uses two message types

- EST\(r()\) used at round \(r\) to BV-bcast est.
- AUX\(r()\) used at round \(r\) to broadcast bin-values\([r]\)
Safe DBFT (con’t)

BV-broadcast [JACM’15]

• **BV-obligation**: if $t+1$ correct BV-bcast $v$, then $v$ is eventually added to the set bin-values of all correct processes

• **BV-justification**: if $pi$ is correct and $v$ in bin-values$_i$, then $v$ was BV-bcast by a correct process

• **BV-uniformity**: if $v$ is added to bin-values$_i$ of correct $p_i$, then eventually $v$ will be in bin-values$_j$ for all correct $p_j$

• **BV-termination**: eventually bin-values of correct $p_i$ is not empty
Safe DBFT (con’t)

est ← v
r ← 0
while (true) {
    r ← r+1
    BV-bcast EST[r](est)
    wait until (bin-values[r] ≠ ∅)
    bcast AUX[r](bin-values[r])
    wait until (messages AUX[r](bval_{p1}), ..., AUX[r](bval_{p(n-t)}) received
    from (n-t) distinct processes p(x), 1≤x≤n-t, and their content is such that:
    ∃ a non-empty set values_i such that (i) values_i in bin-values[r]; and (ii) values_i = U b_val_x

b ← r mod 2
if (values_i = {v}) {
    est ← v
    if (v=b) then decide(v) if not done yet
} else {
    est ← b
}
Safe DBFT (con’t)

est ← v
r ← 0
while (true) {
    r ← r+1
    BV-bcast EST[r](est)
    wait until (bin-values[r] ≠ ∅)
    bcast AUX[r](bin-values[r])
    wait until (messages AUX[r](bval_{p_1}), ..., AUX[r](bval_{p_{n-t}}) received
        from (n-t) distinct processes p(x), 1 ≤ x ≤ n-t, and their content is such that:
        ∃ a non-empty set values_i such that (i) values_i in bin-values[r]_i and (ii) values_i = U b_val_x
    b ← r mod 2
    if (values_i = {v}) {
        est ← v
        if (v=b) then decide(v) if not done yet
    } else {
        est ← b
    }
}
Safe DBFT (con’t)

Lemma 1: If at the beginning of a round, all correct processes have the same estimate, they never change their estimate thereafter.

Proof sketch. Assume all correct have the same estimate $v$ at round $r$. BV-Obligation and BV-Justification $\Rightarrow$ bin_values[$r$] = \{v\}. Hence, at every correct we have values = \{v\}, so that est becomes $v$. 
Safe DBFT (con’t)

Lemma 2: Let $p_i$ and $p_j$ be correct. If their values are singletons, then they are the same.

Proof sketch. If a correct process has values $= \{v\}$ then it received $\text{AUX}[r]\{v\}$ from $n-t$ distinct processes and $t+1$ correct. For two correct processes to have $w$ and $v$ as this singleton, it would mean they received these distinct values from $n-t$ processes each. As $(n-t)+(t+1) > n$, one correct process must have sent the same value to both, and we have $v=w$. 
Theorem [Agreement]: No two correct processes decide different values.

Proof sketch. Let r be the 1st round where a correct process decides, hence values[r] = \{v=r \mod 2\}. If another correct process decides w in the same round, then v=w by Lemma 2. Let pj be a correct that does not decide in round r. By Lemma 2, values_j \neq \{w\} so values_j = \{0,1\}. Thus est_j = v. All correct have thus the same estimate in round r+1, and stick to it (Lemma 1).
Safe DBFT (con’t)

BC, binary consensus instance

mv-propose(v) { // similar to [PODC’94]
    RB-bcast VAL(v) // reliable broadcast [Bra’87]
    repeat if (∃ k : proposals[k]≠⊥ ∧ BC[k].binpropose() not invoked)
        BC[k].binpropose(1)
    until(∃ l: bin-decisions[l]=1)
    for each k such that BC[k].binpropose() not invoked:
        BC[k].binpropose(0)
    wait until (¬1≤x≤n bin-decisions[x]≠⊥)
    j = min{x : bin-decisions[j] = 1}
    wait until proposals[j] ≠⊥
    decide(∪∀j:bin-decisions[j]=1 proposals[j])
}

when val(v) is RB-delivered from pj do // reliable broadcast delivery
    if valid(v) then
        proposals_i[j] <- v;
        BV-deliver b-val[1](1) to BC[j]

when BC[k].binpropose() returns b do bin-decisions[k] <- b
Safe DBFT (con’t)

BC, binary consensus instance

\[
\text{mv-propose}(v) \{ \text{ // similar to [PODC’94]}
\]
\[
\text{RB-bcast } \text{VAL}(v) \text{ // reliable broadcast [Bra’87]}
\]
\[
\text{repeat if } (\exists \ k : \text{proposals}[k] \neq \bot \land \text{BC}[k].\text{binpropose()} \text{ not invoked})
\]
\[
\text{BC}[k].\text{binpropose}(1)
\]
\[
\text{until}(\exists \ l : \text{bin-decisions}[l]=1)
\]
\[
\text{for each } k \text{ such that BC}[k].\text{binpropose()} \text{ not invoked:}
\quad \text{BC}[k].\text{binpropose}(0)
\]
\[
\text{wait until } (\land \{x \leq n \mid \text{bin-decisions}[x] \neq \bot\})
\]
\[
\text{j} = \text{min}\{x : \text{bin-decisions}[j] = 1\}
\]
\[
\text{wait until } \text{proposals}[j] \neq \bot
\]
\[
\text{decide}\left(\bigcup_{j : \text{bin-decisions}[j]=1} \text{proposals}[j]\right)
\}
\]

when \text{val}(v) \text{ is RB-delivered from pj do} \text{ // reliable broadcast delivery}

if valid(v) then

\[
\text{proposals}_{i}[j] \leftarrow v;
\]
\[
\text{BV-deliver } b-\text{val}[1](1) \text{ to } \text{BC}[j]
\]

when \text{BC}[k].\text{binpropose()} \text{ returns } b \text{ do bin-decisions}[k] \leftarrow b

Spawn multiple binary cons. instances

Use binary decisions as a bitmask
Red Belly Blockchain
The Red Belly Blockchain

All nodes communicate through TCP + SSL

Certificates are given in blocks
The Red Belly Blockchain

The genesis block also contains a list of n participants.
A tx is committed if t+1 participants say so.
The Red Belly Blockchain

The $n$ nodes running the consensus...

are regularly changed: $n$, $n'$, $n''$... but $t'<n'/3$, $t''<n''/3$...
O The Red Belly Blockchain

And encoded in the next block

if the correct $n$ participants agree (consensus)
The Red Belly Blockchain

Signature verification uses ECDSA and is **sharded**

...each transaction is verified by $t+1 \leq k \leq 2t+1$ nodes
The Red Belly Blockchain

Each client sends a transaction to $t+1$ proposers

...each proposer maps to $t+1$ primary proposers and $t$ secondary proposers
The Red Belly Blockchain

Execution example with $n=4$ and $t=1$
The Red Belly Blockchain

1. client sends to $t+1$ proposers
The Red Belly Blockchain

1. one proposer is primary, t others are secondary
The Red Belly Blockchain

2. another client sends to $t+1$ proposers
The Red Belly Blockchain

3. after a timeout or enough tx, proposers extract

...a proposal of txs and send it to all deciders
The Red Belly Blockchain

4. deciders exchange hashes of the proposals

\[ h(tx2, \ldots) \]
\[ h(tx1, \ldots) \]
The Red Belly Blockchain

5. upon reception of $2t+1$ same hashes
5. upon reception of $2t+1$ same hashes, all verifiers

...verify the transaction signatures and send results

The Red Belly Blockchain
The Red Belly Blockchain

6. If non identical t+1 responses are received

- $h(tx2, \ldots)$, incorrect!
- $h(tx2, \ldots)$, correct!
- $tx2$ ko
- $tx2$ ok
The Red Belly Blockchain

6. if non identical t+1 responses are received

\[ h(tx2, \ldots), \text{ incorrect!} \]

...the t secondary verifiers verify and break ties
The Red Belly Blockchain

7. the verified transactions are stored in an array
The Red Belly Blockchain

8. a binary consensus is invoked for each proposal

...with input value 1 if no binary consensus output 1
The Red Belly Blockchain

8. the decided binary values serve as a bitmask
8. the decided binary values serve as a bitmask

...to create a block of transactions to be appended
How does it perform?
Benchmark

• Initiator sends a message to n nodes to start (with same genesis block)

• Each node connects to each other through SSL/TLS

• Average over multiple instances of consensus in which:
  • Each of the n nodes proposes a block of 10K txs
  • Each node spawns n instances of RBbcast and BBC
  • Each tx is a 350-byte UTXO transaction
  • Each transaction gets validated by $t+1 \leq k \leq 2t+1$ nodes
  • Each node stores the blockchain locally
Scalability

Amazon EC2 c4 instances, 18 HT cores, 60 GiB mem, 2 Gbps, t=6
Amazon EC2 c4 instances, 18 HT cores, 60 GiB mem, 2 Gbps, t=6
World-Wide Deployment
World-Wide Deployment
World-Wide Deployment

140 machines
World-Wide Network

c4 instances, 4 vCPU, 7.5 GiB, 750 Mbps, n=140, t=46

![Iperf bandwidth (MB/s)]
World-Wide Network

c4 instances, 4 vCPU, 7.5 GiB, 750 Mbps, n=140, t=46

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokyo</td>
<td>0</td>
<td>6</td>
<td>33</td>
<td>69</td>
<td>106</td>
<td>110</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>90.1</td>
<td>96.2</td>
<td>129</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>Seoul</td>
<td>6</td>
<td>0</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Mumb.</td>
<td>33</td>
<td>0</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Singa.</td>
<td>106</td>
<td>106</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Sydney</td>
<td>161</td>
<td>196</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Canada</td>
<td>106</td>
<td>106</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Frankf.</td>
<td>161</td>
<td>196</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Ireland</td>
<td>196</td>
<td>196</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Lond.</td>
<td>196</td>
<td>196</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>São P.</td>
<td>196</td>
<td>196</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>N.Virg.</td>
<td>196</td>
<td>196</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Ohio</td>
<td>196</td>
<td>196</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>N.Cal.</td>
<td>196</td>
<td>196</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Oregon</td>
<td>196</td>
<td>196</td>
<td>137</td>
<td>67</td>
<td>90.9</td>
<td>161</td>
<td>196</td>
<td>55.4</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td></td>
</tr>
</tbody>
</table>

Ping latency (ms)
### World-Wide Network

c4 instances, 4 vCPU, 7.5 GiB, 750 Mbps, n=140, t=46

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokyo</td>
<td>6</td>
<td>551</td>
<td>129</td>
<td>240</td>
<td>161</td>
<td>106</td>
<td>74</td>
<td>66.4</td>
<td>59</td>
<td>55.4</td>
<td>90.1</td>
<td>96.2</td>
<td>129</td>
<td>132</td>
</tr>
<tr>
<td>Seoul</td>
<td>33</td>
<td>137</td>
<td>157</td>
<td>141</td>
<td>91.5</td>
<td>54</td>
<td>60.8</td>
<td>54.7</td>
<td>56.6</td>
<td>84.2</td>
<td>114</td>
<td>84.2</td>
<td>116</td>
<td>119</td>
</tr>
<tr>
<td>Mumb.</td>
<td>133</td>
<td>6</td>
<td>121</td>
<td>67</td>
<td>90.9</td>
<td>176</td>
<td>178</td>
<td>145</td>
<td>46.7</td>
<td>81.9</td>
<td>80.5</td>
<td>69.1</td>
<td>64.2</td>
<td>77.3</td>
</tr>
<tr>
<td>Sing.</td>
<td>69</td>
<td>100</td>
<td>67</td>
<td>90.5</td>
<td>83.2</td>
<td>90.7</td>
<td>86.1</td>
<td>90.4</td>
<td>40.8</td>
<td>59.5</td>
<td>64.9</td>
<td>80.5</td>
<td>80.5</td>
<td>77.3</td>
</tr>
<tr>
<td>Sydney</td>
<td>106</td>
<td>115</td>
<td>235</td>
<td>130</td>
<td>77.1</td>
<td>61.3</td>
<td>53.8</td>
<td>51.2</td>
<td>40.2</td>
<td>74.9</td>
<td>99.7</td>
<td>135</td>
<td>119</td>
<td>119</td>
</tr>
<tr>
<td>Canada</td>
<td>166</td>
<td>185</td>
<td>156</td>
<td>225</td>
<td>166</td>
<td>250</td>
<td>164</td>
<td>139</td>
<td>83.8</td>
<td>79.0</td>
<td>23.5</td>
<td>168</td>
<td>168</td>
<td>168</td>
</tr>
<tr>
<td>Frankfurt</td>
<td>244</td>
<td>273</td>
<td>112</td>
<td>292</td>
<td>102</td>
<td>9</td>
<td>477</td>
<td>823</td>
<td>92.9</td>
<td>222</td>
<td>220</td>
<td>144</td>
<td>83.7</td>
<td>83.7</td>
</tr>
<tr>
<td>Ireland</td>
<td>226</td>
<td>246</td>
<td>122</td>
<td>286</td>
<td>78</td>
<td>25</td>
<td>8</td>
<td>825</td>
<td>114</td>
<td>135</td>
<td>133</td>
<td>134</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>London</td>
<td>255</td>
<td>284</td>
<td>111</td>
<td>281</td>
<td>90</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>107</td>
<td>190</td>
<td>195</td>
<td>137</td>
<td>85.5</td>
<td>85.5</td>
</tr>
<tr>
<td>São Paulo</td>
<td>771</td>
<td>783</td>
<td>402</td>
<td>312</td>
<td>113</td>
<td>7.5</td>
<td>184</td>
<td>197</td>
<td>6</td>
<td>133</td>
<td>126</td>
<td>77.7</td>
<td>81.7</td>
<td>81.7</td>
</tr>
<tr>
<td>N. Virginia</td>
<td>62</td>
<td>209</td>
<td>182</td>
<td>205</td>
<td>15</td>
<td>89</td>
<td>5</td>
<td>76</td>
<td>122</td>
<td>9</td>
<td>827</td>
<td>232</td>
<td>186</td>
<td>186</td>
</tr>
<tr>
<td>Ohio</td>
<td>109</td>
<td>159</td>
<td>133</td>
<td>190</td>
<td>25</td>
<td>99</td>
<td>91</td>
<td>87</td>
<td>133</td>
<td>13</td>
<td>424</td>
<td>219</td>
<td>219</td>
<td>219</td>
</tr>
<tr>
<td>N. California</td>
<td>120</td>
<td>150</td>
<td>262</td>
<td>176</td>
<td>148</td>
<td>76</td>
<td>142</td>
<td>138</td>
<td>182</td>
<td>64</td>
<td>52</td>
<td>9</td>
<td>601</td>
<td>601</td>
</tr>
</tbody>
</table>
World-Wide Deployment

c4 instances, 4 vCPU, 7.5 GiB, 750 Mbps, n=140

Throughput (TPS)

Fault tolerance (t)

- VISA
- RBBC
Consensus Comparison

- **PBFT**: State-of-the-art Byzantine consensus implementation [OSDI’02]. It relies on a leader and decides on one of the proposed value.

- **HBBFT**: The Honey Badger BFT [CCS’16] is based on the binary randomized consensus algorithm [PODC’14], a consensus reduction [PODC’94] and uses erasure codes.

- **DBFT**: The Democratic BFT [CGLR17] we introduced for RBBC. It is leader-less, does not exchange erasure codes but block hashes.
Consensus Comparison

c4 instances, 4 vCPU, 7.5 GiB, 750 Mbps, n=140, t=46

Throughput (TPS)

- HBBFT
- PBFT
- DBFT
- VISA

Block size (#transactions)
Consensus Comparison

c4 instances, 4 vCPU, 7.5 GiB, 750 Mbps, n=140, t=46

Block size (#transactions)

Throughput (TPS)

HBBFT  PBFT  DBFT  VISA

1 100 1000 1000000
Consensus Comparison

c4 instances, 4 vCPU, 7.5 GiB, 750 Mbps, n=140, t=46

Block size (#transactions)

Latency (milliseconds)

HBBFT  PBFT  DBFT
Latency vs. Throughput

c4 instances, 4 vCPU, 7.5 GiB, 750 Mbps, n=140, t=46
Conclusion

• Blockchain systems provide guarantees not well-understood
  • This may lead to dramatic anomalies (and then double-spending attack)

• We propose the Red Belly Blockchain
  • Efficient: 660+ KTPS all committed within 4 seconds
  • Scale to 260 consensus participants (not limited by #clients)
  • Deterministic: with a new leaderless Byzantine consensus algorithm
  • Dynamic: A membership reconfiguration

• The next steps is to deploy it and build applications on top
References


• [PODC’14] A. Mostefaoui, H. Moumen, M. Raynal. Signature-free asynchronous byzantine consensus with t<n/3 and o(n ) messages. PODC, 2014.


More information

http://redbellyblockchain.io