BlockSim: Blockchain Simulator

Carlos Faria

carlosfigueira@tecnico.ulisboa.pt

Miguel Correia

miguel.p.correia@tecnico.ulisboa.pt

IEEE Blockchain 2019

ULISBOA | UNIVERSIDADE DE LISBOA

TÉCNICO LISBOA

FCT Fundação para a Ciência e a Tecnologia

inesc id lisboa

blockbird

Problems on the evaluation of blockchain systems

- ➔ **Blockchain systems** have received much interest both in research and industry
- ➔ However, there is a clear **lack of tools to evaluate these systems**
- ➔ **Emulation** is the most used method; it reproduces the behaviour of a system in a large number of machines
- ➔ Not **scalable** and **energy** efficient to evaluate large distributed systems

Simulation

- Network and distributed system simulators can evaluate the **performance** of protocols in a **large set of conditions**
- Simulators **simplify** the implementation and deployment of **existing** or **new** protocols/systems
- Large-scale system can be study with **thousands of nodes** in a single machine and gather results in reasonable time
- Existing **blockchain simulators** are restricted to one implementation, *not having the flexibility to easily simulate other blockchain systems*

Objectives

Provide a simulator capable of evaluating blockchains in **different environment conditions**, enabling, thus, a richer understanding of this technology.

- ❏ Capable to run **user defined** simulation **models**
- ❏ Capable to run **thousands** of nodes on a single host
- ❏ Should provide an **accurate** representation of a real blockchain system
- ❏ Users should be capable to **change** the simulated **environment conditions**
- ❏ Simulation should be performed in **reasonable time**
- ❏ Capable to provide a **report** with the simulated results when concluded

Solution BlockSim

A **flexible** blockchain simulator to evaluate **different** implementations on large scale networks

Chosen simulation models:

- ➔ **Stochastic**: works with probabilistic phenomena
  E.g: probability distribution of: block interval or block size
- ➔ **Dynamic**: represents the system as it changes over a certain time frame
- ➔ **Discrete-event**: keeps track of system state changes at specific points in time

Following a mechanism of **model abstraction** we can attain **flexibility** in simulating different types of blockchains

Solution Modelling of Random Phenomena

- ➔ **Certain event could happen**, but we do not know which particular outcome will happen
  - ◆ But, we can **observe** a regular distribution of outcomes in a **large number of repetitions**

- ➔ Our models always intend to mimic the behaviour of the entities in real world
  - ◆ *E.g.:* knowing the average block time interval on a public blockchain, its possible to predict the next outcome with a degree of confidence
  - ◆ By **extrapolating a probability distribution** for a given phenomena observed in a real system

- ➔ In practical terms, we assembled a **methodology** to measure, collect, and extrapolate a probability distribution that our models will use

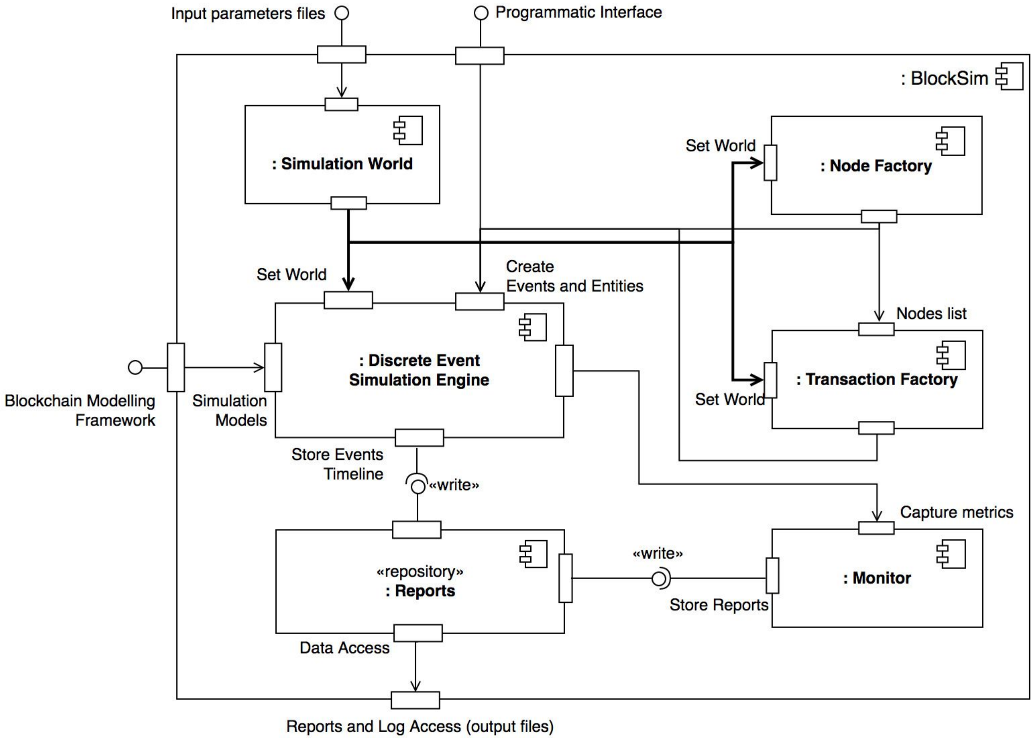
Solution Modelling of Random Phenomena

To calculate the throughput when sending and receiving TCP packets between different geographic locations, our procedure was:

1. Instantiate 2 instances on AWS on the desired geographic locations with ***iPerf3***
2. Measure the throughput **received** and **sent** between each instance using *iPerf3*, at each hour, for 24 hours
3. At the end of 24 hours, we **collect** the iPerf3 logs
4. We use the **Kolmogorov–Smirnov test** to know which distribution and its input parameters that best fit the samples collected
5. The distribution name and its input parameters are then used by the simulator to extrapolate the values of throughput between different geographic locations during the simulation

Solution Architecture

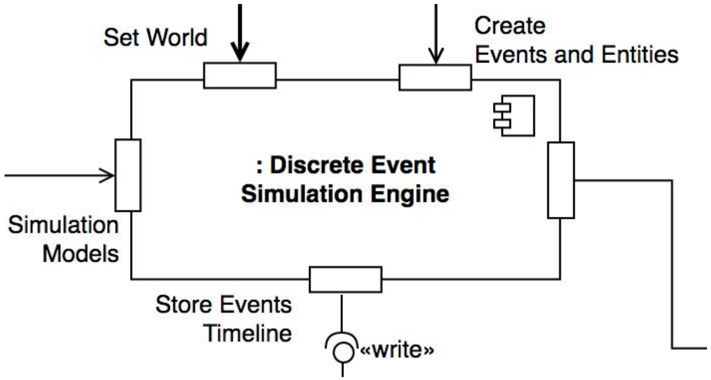
Component & Connector View of BlockSim:



Solution Discrete Event Simulation Engine (DESE)

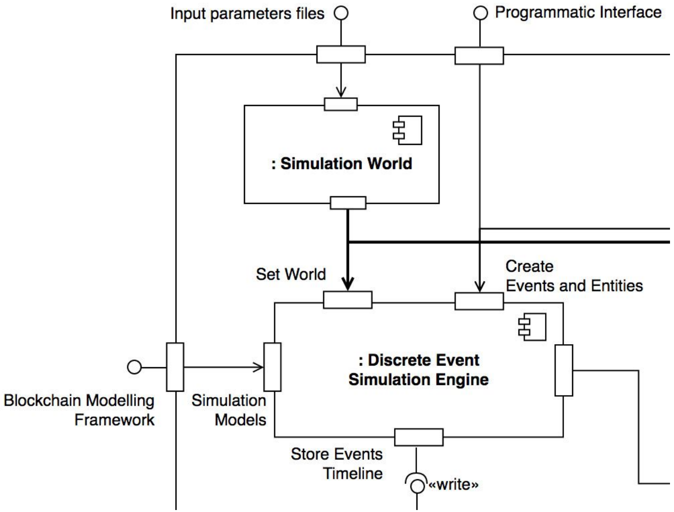
Overall functionality

- ➔ Process-based discrete-event engine
- ➔ Processes are based on Python **generator functions**
- ➔ All processes live in an environment and the interaction is through events
- ➔ **Shared resources** between processes can model limited capacity congestion points



- ➔ Management of the **simulation clock**
- ➔ **Scheduling**, **queuing** and **processing** events
- ➔ Control the access of resources by the entities
- ➔ Creation of blockchain system entities (nodes, blocks, transactions)

Solution Simulation World and Programmatic Interface



Simulation World **functionality**:

Management of the simulation input parameters:
- ➔ Configuration file
- ➔ Delays
- ➔ Latency
- ➔ Throughput received and sent

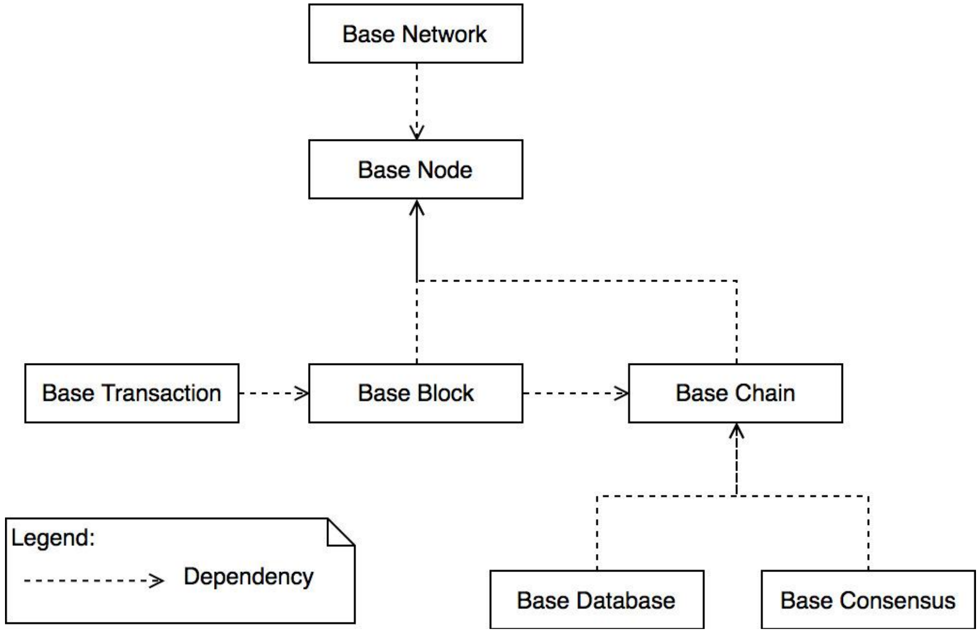
```
world = SimulationWorld(
  duration,
  now,
  “input/config.json”,
  “input/latency.json”,
  “input/throughput-received.json”,
  “input/throughput-sent.json”,
  “input/delays.json”
)

network = Network(world.env, “ETH”)

miners = {
 'Ohio': {
   'how_many': 300,
   'mega_hashrate_range': "(20, 40)"
 }
}

non_miners = {
  'Tokyo': {
    'how_many': 100
  }
}

node_factory = NodeFactory(world, network)

nodes_list = node_factory.create_nodes(
  miners,
  Non_miners
)

world.env.process(network.start_heartbeat())

for node in nodes_list:
  node.connect(nodes_list)

transaction_factory =
TransactionFactory(world)

transaction_factory.broadcast(
  40,
  200,
  300,
  nodes_list
)

world.start_simulation()
```

Programmatic Interface

Solution Blockchain Modelling Framework

To model **any blockchain implementation**, we need to **split** it into **submodels**, creating an **abstraction** that does not follow a specific implementation

These basic models can be **extended** to simulate specific blockchain implementations



Network Model

Contains the state of each node; build **connection** channels; apply network latency

Nodes are selected to **broadcast** their candidate block; Interval between each selection is the time between blocks

Nodes have a **hash rate**; greater hash rate, greater the probability of the node being chosen

It also simulates the occurrence of **orphan blocks**

Node Model

P2P network functionality

Origin node starts listening for inbound communications from a *destination node*; a node can send a direct message or **broadcast** a message to all neighbours

It also apply a **delay** when receiving and sending messages, corresponding to **node throughput**

This model is normally extended to implement a specific blockchain client implementation

Chain Model

Mimic the behaviour of a chain:

- when adding a block, checks if the block is being added to the **head**; if the case, adds a block to the chain. Otherwise, the block is **added to a queue**

- when is not being added to the head, and the previous hash points to an old block, it creates a **fork** on the chain by creating a **secondary chain**. Then, it checks if the block should be the new head by **calculating the difficulty of the chain**. If this is the case, it accepts the secondary chain as the main chain

Consensus Model

We do not perform block or transaction validation, it adds a **delay** that **simulates** the validation process

It also defines a **simple equation** to calculate the difficulty of a new block:

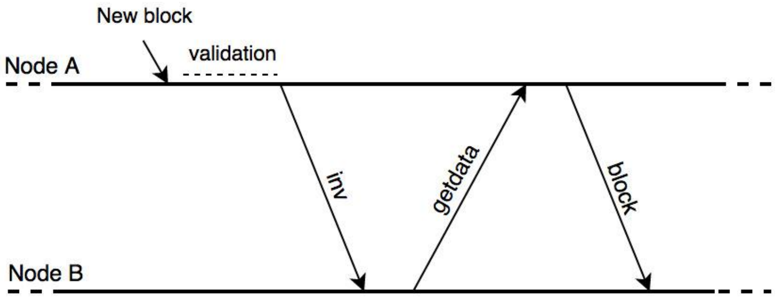
$$difficulty = P_d + (B_{TS} - P_{TS})$$

It **simplifies** and **resembles** ideas from Ethereum and Bitcoin by **incrementing** the difficulty of a block **when it is created in less time**

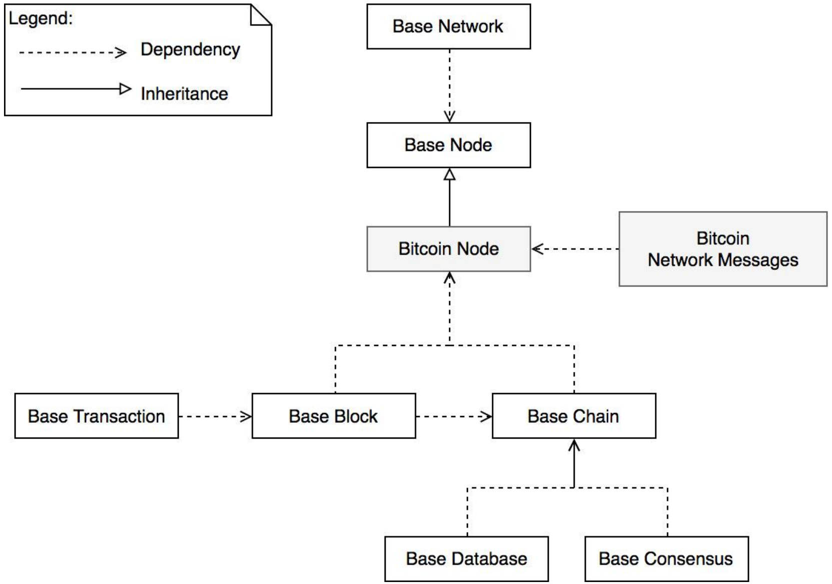
Solution Modelling Bitcoin

We can easily model the Bitcoin blockchain, by reusing the **base models** already created

- **Simulation World** receives the block size limit and the probability distribution for the number of transactions per block
- There are **miner** nodes and **non-miner** nodes
- Miner nodes: broadcast its candidate block to the network (when selected by the **Network**)



Messages exchange in Bitcoin protocol between nodes in order to obtain a new block

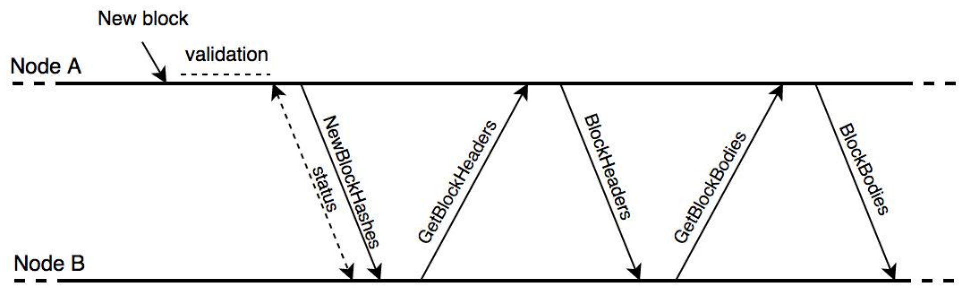


Class diagram for the Bitcoin modelling

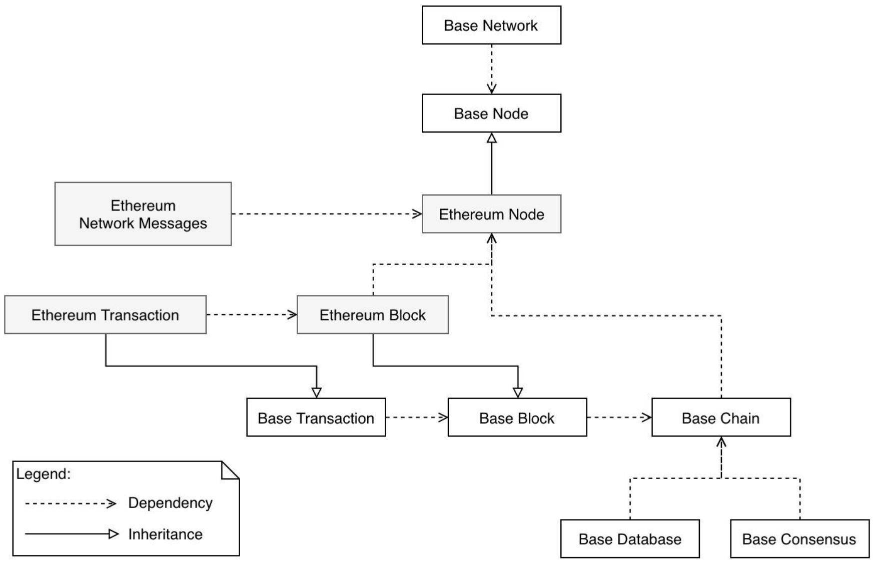
Solution Modelling Ethereum

➔ **Simulation World** receives the **block gas limit** and **start gas** for every transaction

E.g.: if we set the simulation to have a block gas limit of 10,000, and for a transaction start gas of 1,000, then we can fit **10 transactions**



Messages exchange in Ethereum protocol between nodes in order to obtain a new block



Class diagram for the Ethereum modelling

Evaluation

1. Perform a **verification** and **validation** of BlockSim running our Ethereum models

2. **Explore** and **evaluate** real **use cases** for BlockSim

Evaluation Verification and Validation

1. Identify a question to be answered in or study:

   How long it takes to propagate a block and a transaction from one node to another?

2. **Conceptualise** the simple building models needed to answer the question
3. Determine the **input parameters** for the models:

   block and transaction gas limit, message size, distribution of latency and throughput, etc.

4. **Collect** data from existing deployments for each input parameter
5. **Code** the conceptual models using the BlockSim Modelling Framework
6. Perform **verification** of the models
7. **Validate** if the models are an accurate representation of the real system

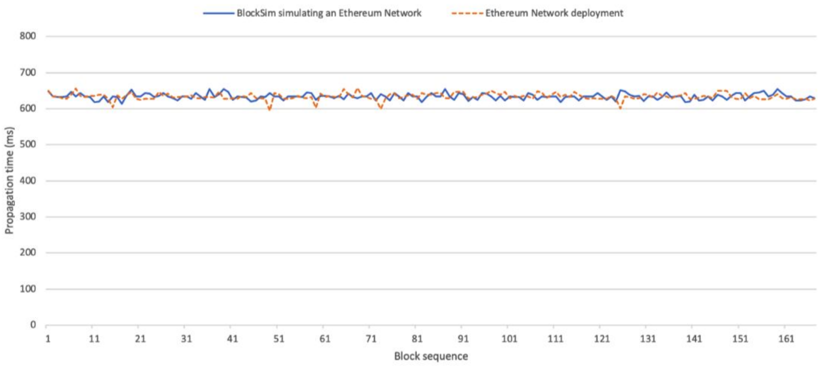
Evaluation Verification and Validation

To **validate** if the Ethereum models are an accurate representation of a **real Ethereum system:**

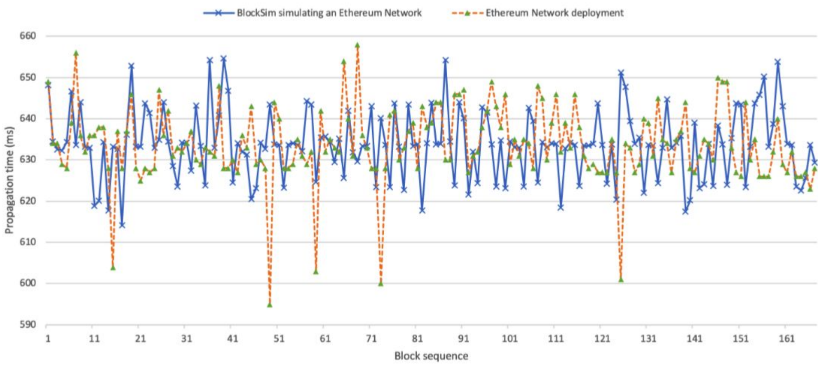
1. In the **simulation** we calculated the block and transaction **propagation time** between two nodes
2. Changed an Ethereum **client reference implementation** (Geth), to record the time when a block and transaction is **sent** and **received**
3. Deployed a **private Ethereum network** using the changed Ethereum client in AWS EC2 instances
4. **Collected the times** from the two nodes and **calculated** the propagation time for a block and transaction

Following this process we **validate** the Ethereum models and also **verify** if BlockSim is working properly, **by comparing the results from the simulation with a real network**

Evaluation Results for the Verification and Validation



(a) Overview.



(b) Detailed view.

Block propagation between Ohio and Ireland

Results for **block propagation** between Ohio and Ireland:

	Average		Standard deviation	
	Real network	BlockSim network	Real network	BlockSim network
Between Ohio and Ireland	634 ms	634 ms	9.2 ms	8.28 ms

➔ Achieved **exact same average** in the real Ethereum network compared to the simulation

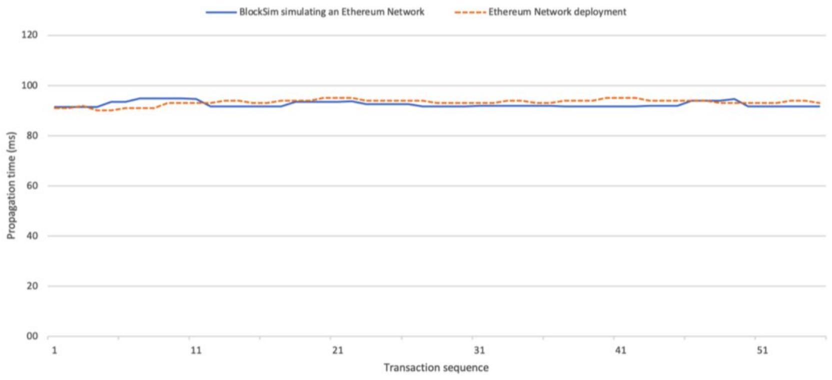
➔ Simulation has slightly low values for standard deviation compared to a real network

- expected because our network model does not consider packet loss, routing and other variations that influence packets deliver in a wide area network (WAN)

Evaluation Results for the Verification and Validation

Results for **transaction propagation** between Ohio and Ireland:

	Average		Standard deviation	
	Real network	BlockSim network	Real network	BlockSim network
Between Ohio and Ireland	93 ms	93 ms	1.22 ms	1.12 ms



Transaction propagation between Ohio and Ireland

➔ Achieved **exact same average** and **standard deviation**

Conclusion

BlockSim runs our **Ethereum models** presenting an accurate representation of an Ethereum system with regards to **block and transaction propagation**

Evaluation Use Cases

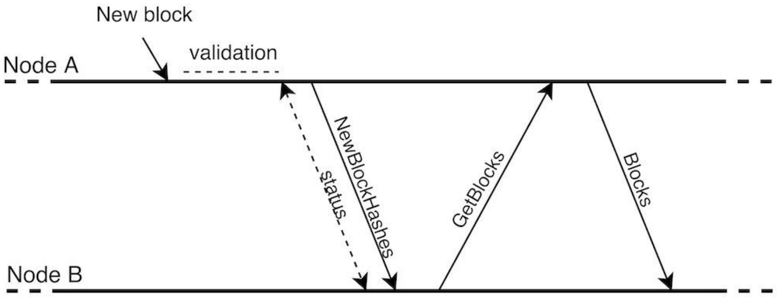
- ➔ We used BlockSim to study 4 use cases
- ➔ For each use case, we created **8,000 transactions** with a total of **400 nodes**
  - ◆ 300 non-miner nodes across Tokyo, Ireland and Ohio
  - ◆ 100 miners across Ireland, Ohio and Tokyo
- ➔ 1 PC with 2 GHz Intel Core i7; 8 GB RAM

	Distribution	Location	Scale	Other parameters
Block validation delay	Log-normal	0.229 s	0.002 s	-
Transaction validation delay	Log-normal	0.004 s	0.00005 s	-
Time between blocks	Normal	15.79 s	3.00 s	-
Latency Ohio-Ireland	Normal	73.70 ms	0.09 ms	-
Throughput Ohio-Ireland	Beta	39.13 Mbps	59.02 Mbps	α = 0.463 β = 0.461
Latency Ireland-Tokyo	Normal	105.42 ms	0.23 ms	-
Throughput Ireland-Tokyo	Beta	51.33 Mbps	89.06 Mbps	α = 0.512 β = 0.914

Input parameters for the BlockSim

Evaluation Use Case #1: Simplified New Block Delivery

- ➔ We model a **new message exchange protocol**, used to obtain a new mined block
- ➔ **Request the full blocks** (headers and bodies) when the message ***NewBlockHashes*** is received
- ➔ To adapt our model, we created **2 new network messages** (*GetBlocks*; *Blocks*) and **adapt our Ethereum node model**



Protocol	Transactions per block	Block size	Average block propagation time
Standard (PV62)	100	20.135 kB	847 ms
Simplified (Figure 4.6)			610 ms

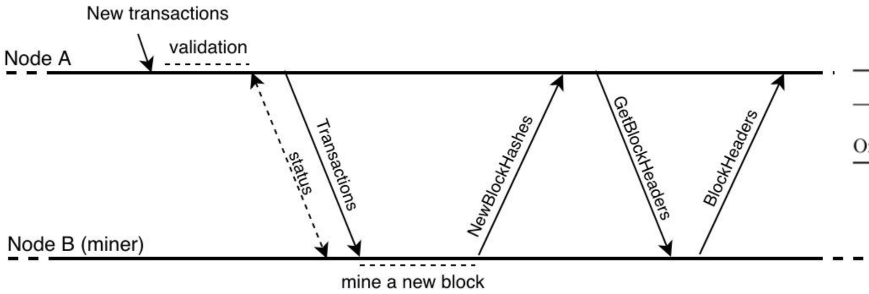
Average block propagation with simplified new block delivery between Tokyo and Ireland

- ➔ **27.9% decrease** in block propagation time

Evaluation Use Case #2: One Transaction Propagation

- ➔ There are inefficiencies in the blockchain network layer:
  - ◆ Transactions are first propagated between nodes, and then a full block when it is mined, that contains the previously propagated transactions
  - ◆ It requires **each transaction to be transmitted twice**

- ➔ We can rely on a **reconciliation protocols**: nodes only fetch transactions that they do not own in a newly mined block



Protocol	Transactions per block	Block header size	Average block propagation time
Standard (PV62)	100	0.09 kB	847 ms
One Transaction Propagation (Figure 4.7)			600 ms

Average block propagation with one transaction propagation between Tokyo and Ireland

- ➔ **29.2% decrease** in block propagation time (simulation time: 22 minutos and 10 seconds)

Evaluation Use Case #3: Different Block Gas Limits

➔ **Block propagation** time impact when **increasing the block gas limit**

➔ For each execution, we change the value of block gas limit, adding more transactions per block

➔ Standard Ethereum transaction has a **21,000** gas limit, with a size of **~200 Bytes**

Transaction gas limit	Block gas limit	Transactions per block	Average block propagation time	Block size
21000	2100000	100	847 ms	20.045 kB
	4200000	200	858 ms	40.045 kB
	6300000	300	869 ms	60.045 kB
	8400000	400	879 ms	80.045 kB

Results for average block propagation with different block gas limit between Tokyo and Ireland

Successfully simulated in 36 minutes and 21 seconds:

➔ **20 kB block size** grow between each execution (corresponds to an additional **100 transactions**)

➔ For each execution, **an increasing propagation time of ~10 ms**

Evaluation

Use Case #4: Encrypted Network Messages

- **Block propagation** time impact when a node **encrypts** and **decrypts all the network messages**
- Node **receives** 4 messages: *Status*, *NewBlockHashes*, *BlockHeaders* and *BlockBodies*, and **sends** 2 messages: *GetBlockHeaders* and *GetBlockBodies*
- We have added to our **basic node model** a **fixed delay** when **receiving** and **sending** a network message

Encrypted	Transactions per block	Encrypt and decrypt delay	Average block propagation time
No		-	847 ms
Yes	100	50 ms	1297 ms
Yes		100 ms	1747 ms

Results for average block propagation with different block encryption and decryption delay between Tokyo and Ireland

- **25.8% increase** in block propagation time, with a encryption delay of **50 ms**
- **51.6% increase** in block propagation time, with a delay of **100 ms**

Final conclusions

- ➔ Blockchain systems are **complex** distributed systems
- ➔ There is a broad interest in developing methods to evaluate these systems
- ➔ First effort to provide a blockchain simulator that is **not restricted to a concrete blockchain implementation** and can be used to model **different blockchain systems**
- ➔ Run **thousands of nodes and transactions** in a **single host** in **reasonable time**
- ➔ We have shown an **accurate representation of the Ethereum system** and how easy it was to change the simulated environment conditions and models to study peculiar use cases

Thank you

BlockSim is available at https://github.com/BlockbirdLabs/blocksim

Carlos Faria
carlosfigueira@tecnico.ulisboa.pt

TÉCNICO LISBOA

FCT
Fundação para a Ciência e a Tecnologia

inesc id lisboa

U LISBOA | UNIVERSIDADE DE LISBOA

blockbird

IEEE