SIEMENS

Ingenuity for life

Setup and Evaluation of Tendermint

Harsh Kumar harsh.kumar@siemens.com

Why Tendermint ?

Monolithic Architecture vs Modular Architecture

- Monolithic architecture means that everything is composed all in one piece.
- When software is deemed “monolithic” the components are interconnected and interdependent with each other and the design is more self-contained.
- Plus, there is another problem with this system. If any component of the program needs to be updated, the whole application will have to be reworked.

On the other hand, we have modular architecture.

- Unlike Monolithic, the layers are not that linked to each other. So, while it may not be as robust, it is quite easy to update the whole application by working with the different separate modules.
- Since the modules are so independent, modular architecture allows you to actually update a particular section without causing unforeseen changes to the rest of the system. Iterative processes are also much more simple in modular programs, as opposed to monolithic.

Basics of Tendermint

Tendermint Core would be responsible for

- Sharing blocks and transactions between nodes
- Establishing a canonical/immutable order of transactions (the blockchain)

The application will be responsible for

- Maintaining the UTXO database
- Validating cryptographic signatures of transactions
- Preventing transactions from spending non-existent transactions
- Allowing clients to query the UTXO database.

kvstore app

The kvstore app is a Merkle tree that just stores all transactions. If the transaction contains an =, e.g. key=value, then the value is stored under the key in the Merkle tree. Otherwise, the full transaction bytes are stored as the key and the value

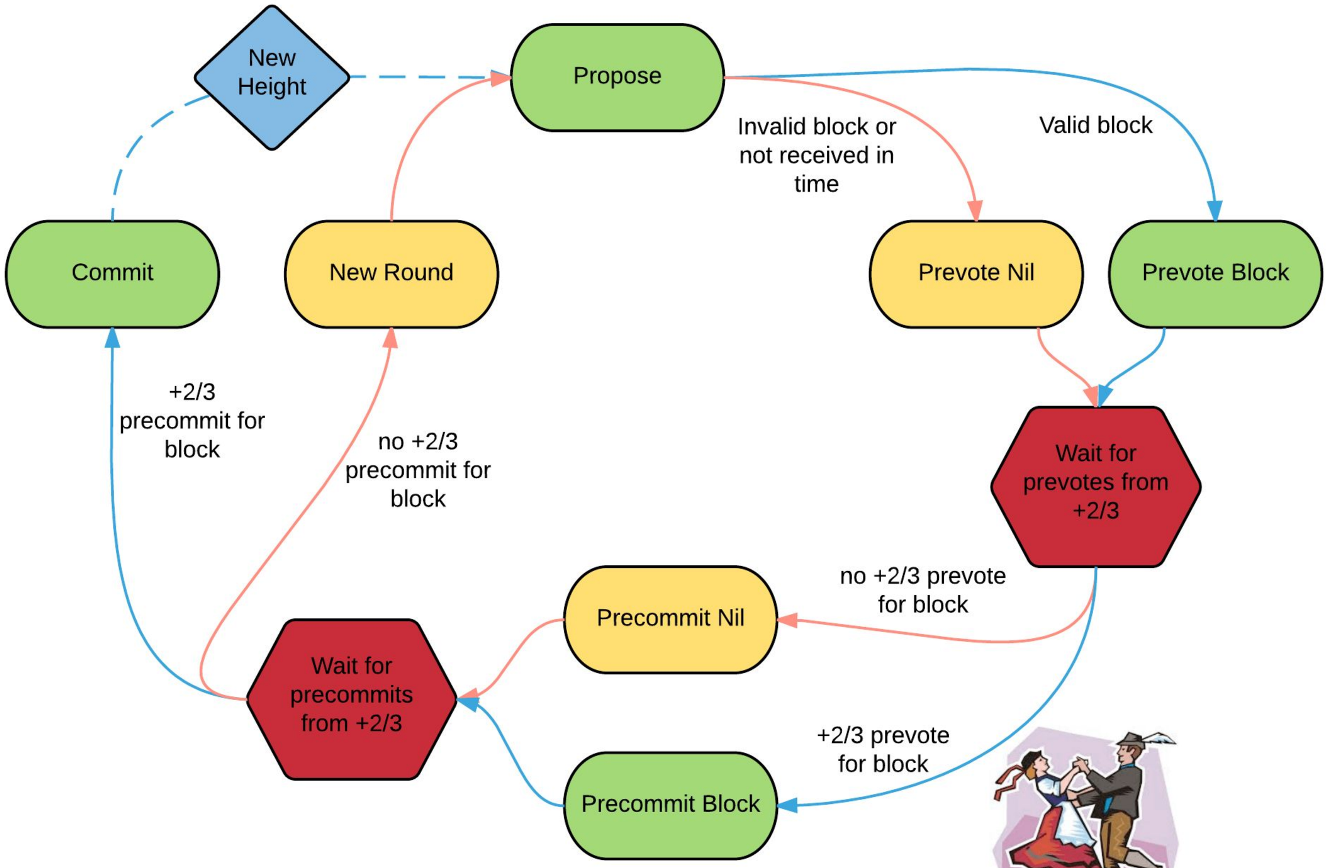
Consensus Overview

SIEMENS
Ingenuity for life

Tendermint = pBFT + PoS(Proof-of-Stake)

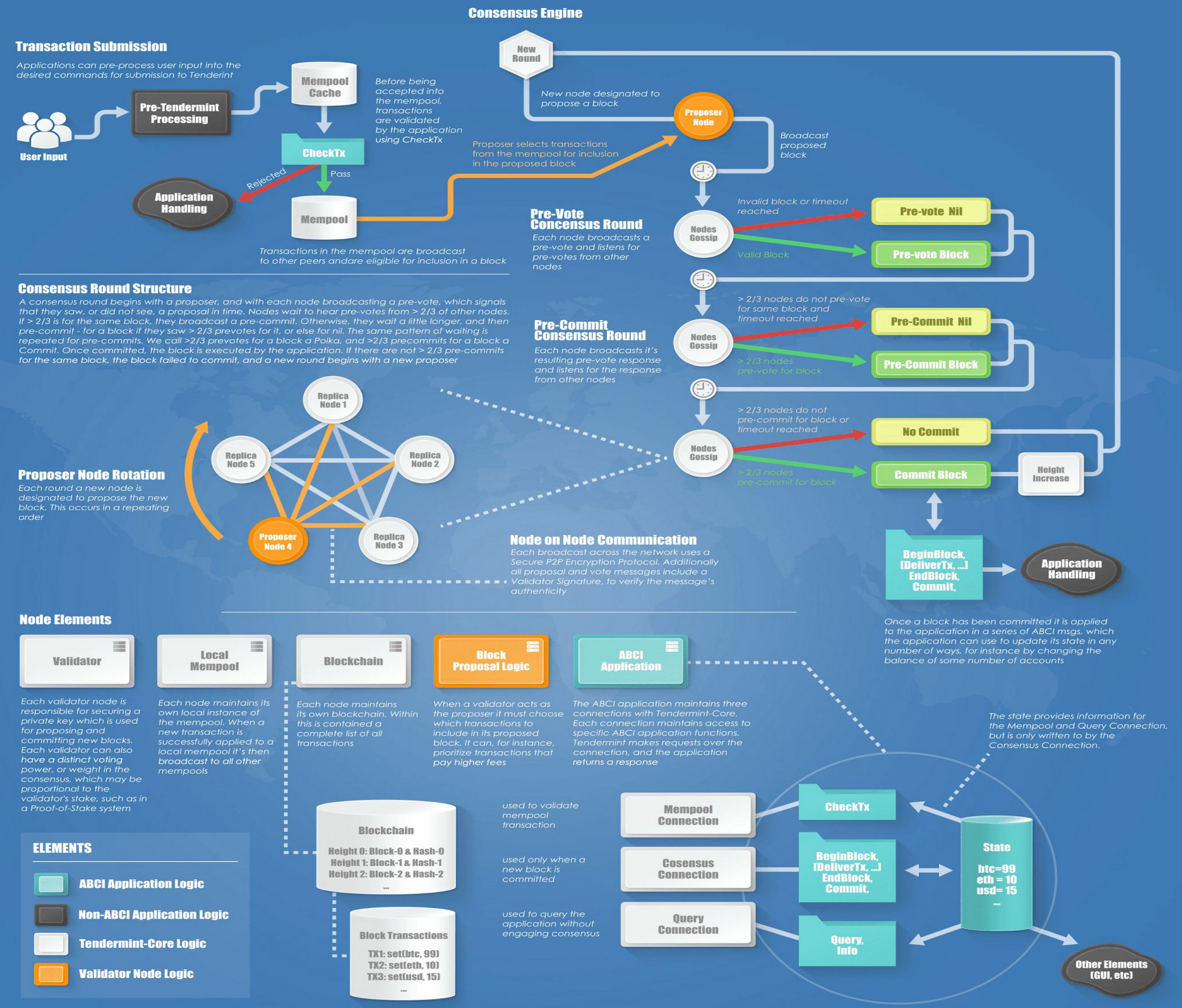
Proof of stake (POS) system:

we have certain people called “validators”. These validators lock up a stake inside the system. After that, they have the responsibility of betting on the block that they feel is going to be added next to the blockchain. When the block gets added, they get a reward proportional to their stake.



New Height
Propose
Invalid block or not received in time
Valid block
Commit
New Round
Prevote Nil
Prevote Block
+2/3 precommit for block
no +2/3 precommit for block
Wait for prevotes from +2/3
no +2/3 prevote for block
Precommit Nil
Wait for precommits from +2/3
+2/3 prevote for block
Precommit Block

Tendermint in a Nutshell



Transaction Submission

Applications can pre-process user input into the desired commands for submission to Tenderint

Before being accepted into the mempool, transactions are validated by the application using CheckTx

Proposer selects transactions from the mempool for inclusion in the proposed block

Transactions in the mempool are broadcast to other peers andare eligible for inclusion in a block

Consensus Engine

New node designated to propose a block

Broadcast proposed block

Pre-Vote Consensus Round

Each node broadcasts a pre-vote and listens for pre-votes from other nodes

Pre-Commit Consensus Round

Each node broadcasts it's resulting pre-vote response and listens for the response from other nodes

Consensus Round Structure

A consensus round begins with a proposer, and with each node broadcasting a pre-vote, which signals that they saw, or did not see, a proposal in time. Nodes wait to hear pre-votes from > 2/3 of other nodes. If > 2/3 is for the same block, they broadcast a pre-commit. Otherwise, they wait a little longer, and then pre-commit - for a block if they saw > 2/3 prevotes for it, or else for nil. The same pattern of waiting is repeated for pre-commits. We call >2/3 prevotes for a block a Polka, and >2/3 precommits for a block a Commit. Once committed, the block is executed by the application. If there are not > 2/3 pre-commits for the same block, the block failed to commit, and a new round begins with a new proposer

Proposer Node Rotation

Each round a new node is designated to propose the new block. This occurs in a repeating order

Node on Node Communication

Each broadcast across the network uses a Secure P2P Encryption Protocol. Additionally all proposal and vote messages include a Validator Signature, to verify the message's authenticity

Once a block has been committed it is applied to the application in a series of ABCI msgs, which the application can use to update its state in any number of ways, for instance by changing the balance of some number of accounts

Node Elements

Validator — Each validator node is responsible for securing a private key which is used for proposing and committing new blocks. Each validator can also have a distinct voting power, or weight in the consensus, which may be proportional to the validator's stake, such as in a Proof-of-Stake system

Local Mempool — Each node maintains its own local instance of the mempool. When a new transaction is successfully applied to a local mempool it's then broadcast to all other mempools

Blockchain — Each node maintains its own blockchain. Within this is contained a complete list of all transactions

Block Proposal Logic — When a validator acts as the proposer it must choose which transactions to include in its proposed block. It can, for instance, prioritize transactions that pay higher fees

ABCI Application — The ABCI application maintains three connections with Tendermint-Core. Each connection maintains access to specific ABCI application functions. Tendermint makes requests over the connection, and the application returns a response

The state provides information for the Mempool and Query Connection, but is only written to by the Consensus Connection.

used to validate mempool transaction

used only when a new block is committed

used to query the application without engaging consensus

ELEMENTS

🟦	ABCI Application Logic
⬛	Non-ABCI Application Logic
⬜	Tendermint-Core Logic
🟧	Validator Node Logic

Benchmarking

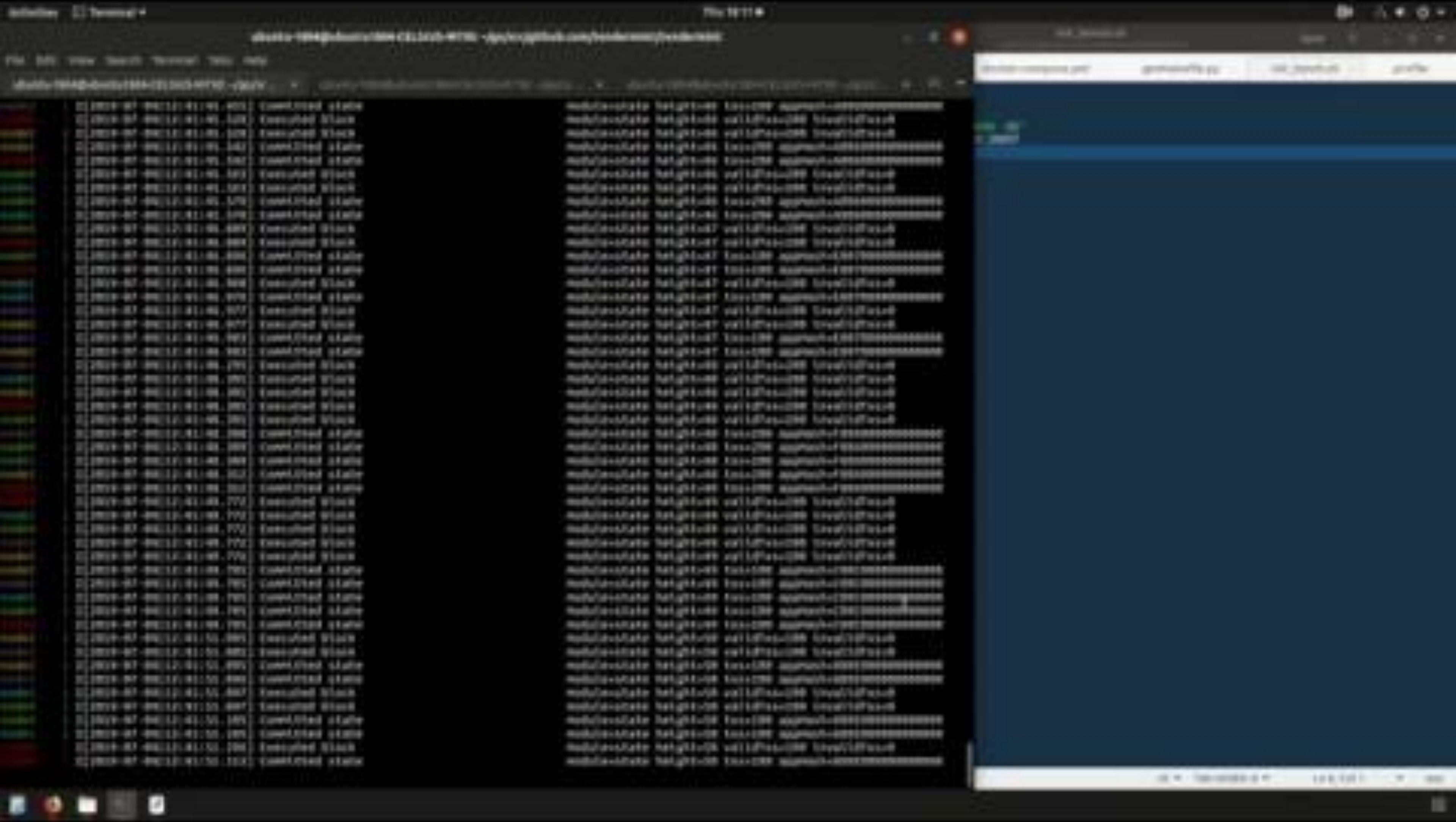
Tm-bench

Insert tm-bench [-c 1] [-T 10] [-r 1000] [-s 250] [endpoints]
Examples:
tm-bench localhost:26657
Flags:

-T int :	Exit after the specified amount of time in seconds (default 10)
-c int :	Connections to keep open per endpoint (default 1)
-r int:	Txs per second to send in a connection (default 1000)
-s int:	Size per tx in bytes
-v :	Verbose output

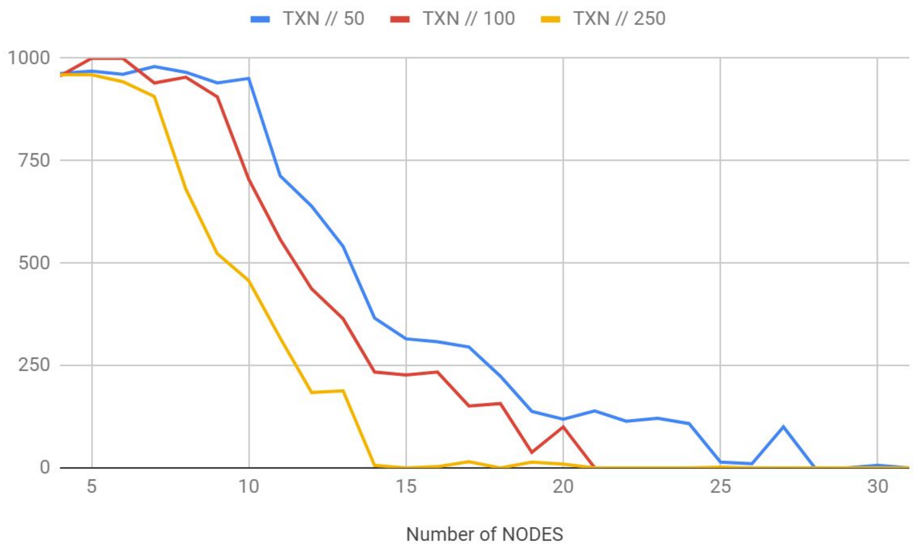
Reading

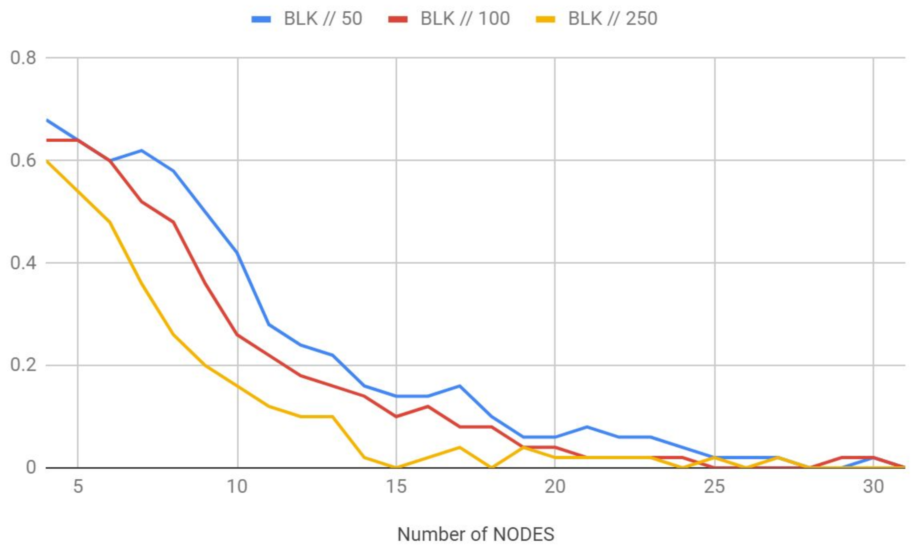
Stats	Avg	StdDev	Max	Total
Txs/sec	818	532	1549	9000
Blocks/sec	0.818	0.386	1	9



```
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=45 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=46 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=46 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=46 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=46 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=46 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=46 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=46 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=46 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=47 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=47 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=47 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=47 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=47 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=47 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=47 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=47 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=47 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=47 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=48 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=48 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=48 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=48 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=48 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=48 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=48 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=48 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=48 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=48 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=49 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=49 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=49 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=49 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=49 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=49 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=49 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=49 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=49 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=49 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=50 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=50 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=50 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=50 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=50 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=50 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=50 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=50 txs=200 appHash=[illegible]
I[2019-07-09|12:41:[illegible]] Executed block         module=state height=50 validTxs=200 invalidTxs=0
I[2019-07-09|12:41:[illegible]] Committed state        module=state height=50 txs=200 appHash=[illegible]
```

Variation with respect to transaction size





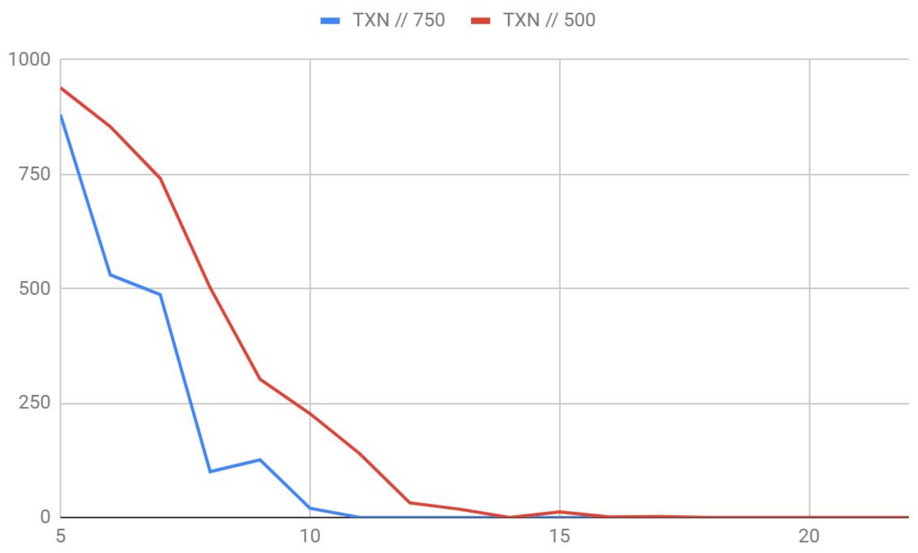
Transaction/Sec Vs Node : Small Transaction Size (<=250 bytes) || Rest parameter - [*Rate of transection : 1000 txn/sec Time duration : 50 sec*]

BLOCKS/SEC VS NODE : Small Transaction Size (<=250 bytes) || Rest parameter - [*Rate of transection : 1000 txn/sec Time duration : 50 sec* **]**

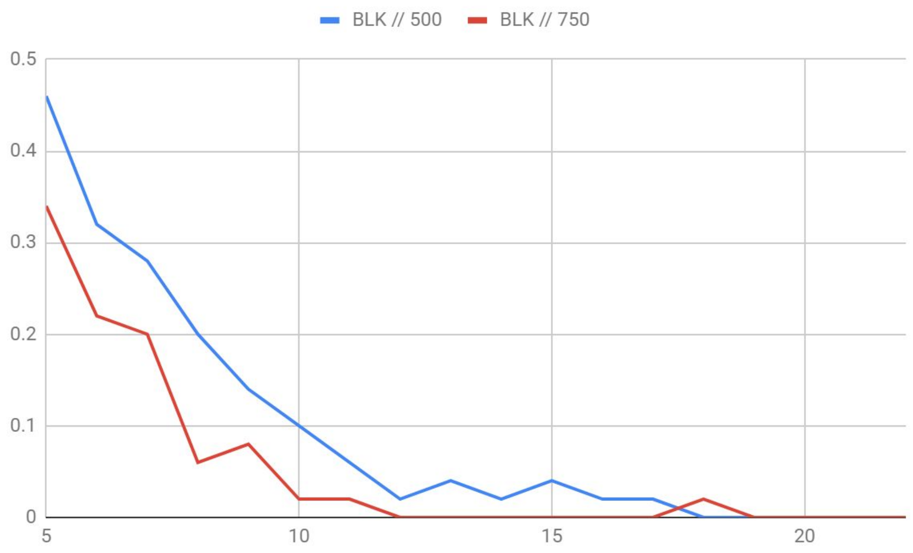
ANALYSIS- These Peaks in the graph are due to failed synchronous Tendermint Network, If the number of Validator or Transaction size is bigger. This leads to longer dialing up for peer by the tendermint node. If was tm -bench runs for that interval, the network results in no transaction processing.

CPU: Intel Xeon E5-1620 v2 @ 8x 3.9GHz [34.0°C]	**RAM**: 128814MiB	OS: Ubuntu 18.04 bionic

Variation with respect to transaction size



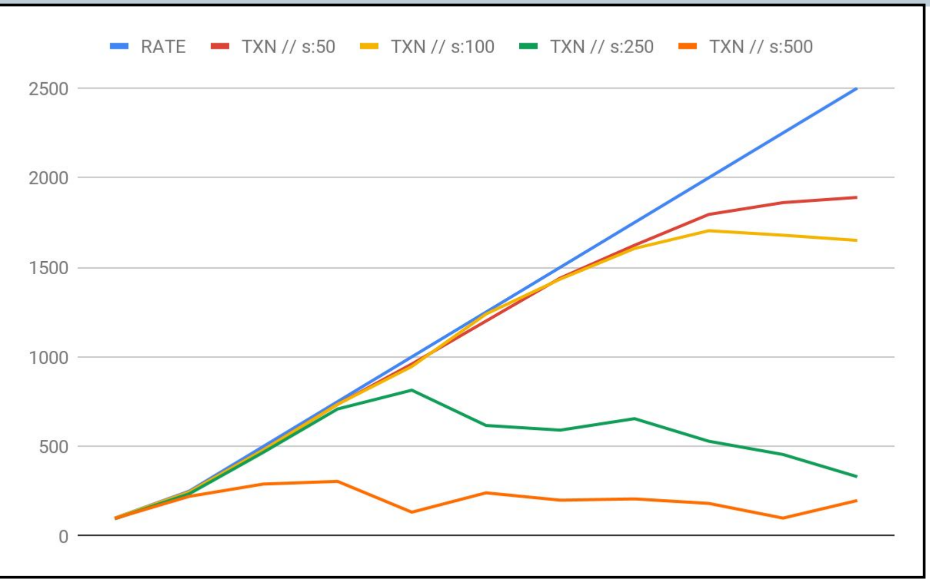
TRANSACTION VS NODE : Large Transaction Size (>=500 bytes) || Rest parameter - [*Rate of transection : 1000 txn/sec Time duration : 50 sec*]

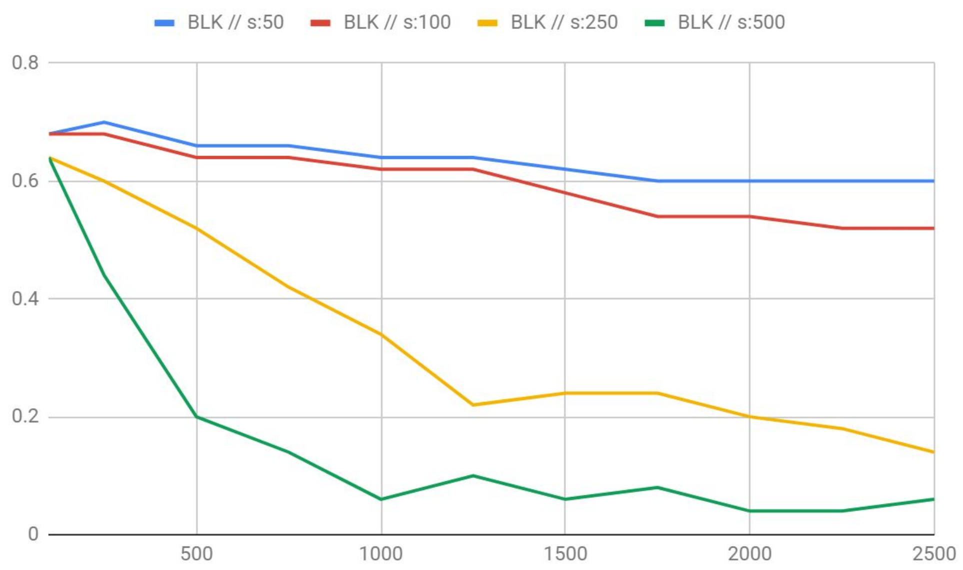


BLOCKS/SEC VS NODE : Large Transaction Size (>=500 bytes) || Rest parameter - [*Rate of transection : 1000 txn/sec Time duration : 50 sec*]

CPU: Intel Xeon E5-1620 v2 @ 8x 3.9GHz [34.0°C]	**RAM**: 6306MiB / 128814MiB	OS: Ubuntu 18.04 bionic

Variation with respect to transaction rate



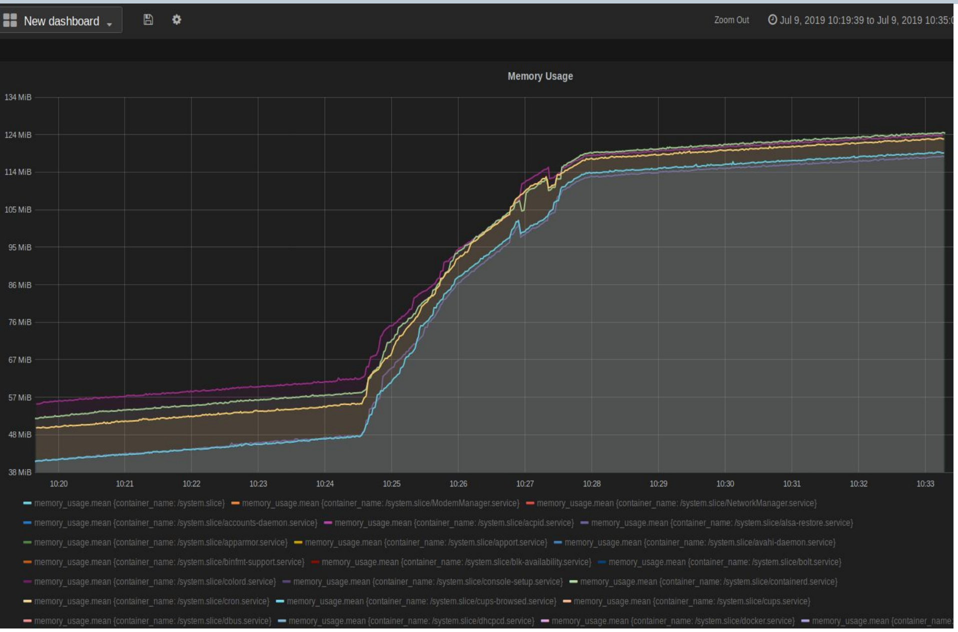


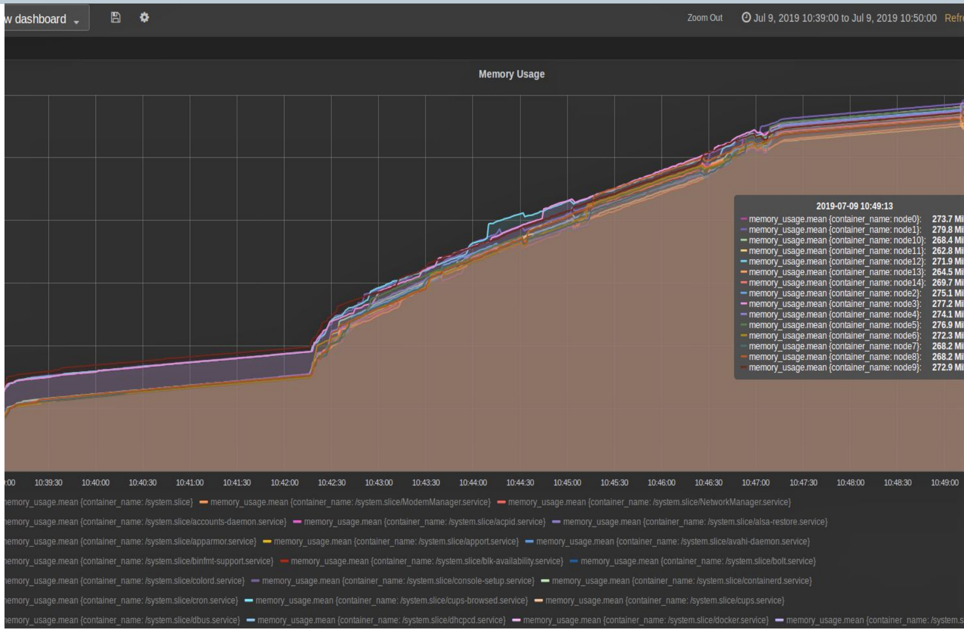
TRANSACTION/SEC VS NODE : Node - 5 || Rest parameter - [*Rate of transection : variable, Time duration : 50 sec* **]**

BLOCK/SEC VS NODE : Node - 5 || Rest parameter - [*Rate of transection : variable, Time duration : 50 sec* **]**

CPU: Intel Xeon E5-1620 v2 @ 8x 3.9GHz [34.0°C] | **RAM**: 6306MiB / 128814MiB | OS: Ubuntu 18.04 bionic

Memory Usage





Memory Usage Percentage Vs Node : Small Number of Node(=5 Node) || Rest parameter - [*Rate of transaction : 1000 txn/sec Time duration : 200 sec*]

- **Memory Usage Percentage Vs Time** : Number of Node(=5 Node) || Rest parameter - [*Rate of transection : 1000 txn/sec Time duration : 300 sec*]

CPU: Intel Xeon E5-1620 v2 @ 8x 3.9GHz [34.0°C]	**RAM**: 6306MiB / 128814MiB	OS: Ubuntu 18.04 bionic

Conclusion & Recommendation

- Shows that tendermint has a limit of 30 Nodes, after adding more node on top of it results in P2P Network breakdown.
- On increasing number of nodes, throughput decrease exponentially, and Increasing transaction size it gets more and more steeper.
- Also if I increase benchmarking and “peer dialup timeout” time then it shows a little transaction getting passed the consensus.
- Tendermint core perform well in number of nodes upto 17 for any size of transaction size within 500 Bytes, and upto 10 nodes with transaction size less than 1000 bytes.
- When rate of transaction is increased the then lesser the transaction size better it gives the throughput. For the transaction size greater than 500 give poor throughput.

CPU: Intel Xeon E5-1620 v2 @ 8x 3.9GHz [34.0°C]	**RAM**: 6306MiB / 128814MiB	OS: Ubuntu 18.04 bionic

Thanks For your Patience!

Q&A?