**Experiment Setup:**

1. After deploying the multi physical node sawtooth setup, perform the following coherence checks to ensure that the setup is working correctly and as expected.
   1. The CPU usage %age in idle mode when the sawtooth components are running:
      1. Ideally you should have the following which are running processes in a barebone installation:
         1. Sawtooth-validator
         2. Sawtooth-rest-api
         3. Intkey-transaction-processor
         4. Settings-transaction-processor
         5. Identity-transaction-processor
      2. Check if all the nodes respond to a REST api query:
         1. The simplest query to make is the /blocks API query, which also is an indication that the other services might be running
2. Deploy the dAuth processor on all the nodes and connect the transaction processor to the local validator node on that machine. Perform a basic coherence check to see if the service or the foreground process is running correctly. You could probably do this by issuing a single transaction and seeing if all the nodes are successfully able to commit a block. Furthermore, validate if there is only one transaction in the new block that has been created by querying the /blocks API and looking at the transactions list.
3. Use iperf to find out the available bandwidth between the machines:
   1. Client ← → Node 1
   2. Node 1 ← → Node 2
   3. Node 2 ← → Node 3
   4. Node 1 ← → Node 3

**Experiment:**

1. Dedicate one single node as the node which receives all the transactions.
   1. You could issue the transactions from a personal laptop on the network to this sawtooth node or from the node itself. Either way, it’d be nice to have the following measurement at this step on the client issuing the transaction:
      1. Transaction ID: Hash(protobuf struct byte array)
      2. Issuance Timestamp : epoch when the transaction was issued and sent to the validator node
      3. Outbound propagation Timestamp: epoch when the transaction was treated as a local transaction and sent to be queued into the batcher.
      4. Propagation Request Timestamp: epoch when the transaction(s) batch was actually issued to the network.
      5. Propagation Response Timestamp: epoch when the transaction(s) batch request through \_send\_request() is obtained.
      6. Size of the transaction: in bytes
   2. On the validator node receiving the transactions collect the following information:
      1. Transaction ID: Hash(protobuf struct byte array received)
      2. Arrival timestamp: epoch when the transaction was received
   3. NOTE: Before using the timestamps here, also try and find out the clock drift in the timestamps on the pairs of the following machines:
      1. Client ← → Node 1
      2. Node 1 ← → Node 2
      3. Node 2 ← → Node 3
      4. Node 1 ← → Node 3
2. Log the information when the transaction was received at the other nodes on the network:
   1. Transaction ID: Hash(protobuf struct byte array)
   2. Arrival Timestamp on the node: epoch when the TX was received to be applied locally. (We might have to look through sawtooth logs for this information, it’s okay if this doesn’t exist, we could push all of this information into the consensus time)
   3. TX confirmation timestamp on the node: epoch, this is when the transaction is received to the do\_operation() method to apply the changes of the transaction.
3. At each of the nodes, log the following:
   1. TX Identifier: This could be the same as the Hash() described above.
   2. TX apply: epoch of when the transaction was applied from the Transaction Processor
4. At each of the nodes, log the following:
   1. Block commit timestamp: epoch
      1. The easiest way to get this information is from the ZMQ events subscription that sawtooth allows you to create which triggers an event when you’re subscribed to the “sawtooth/block-commit” event.
   2. State change timestamp: epoch
      1. This is optional but it's very easily provided by the ZMQ event subscription when subscribing to the “sawtooth/state-delta” event.
   3. More information: <https://sawtooth.hyperledger.org/docs/core/releases/latest/app_developers_guide/zmq_event_subscription.html#correlating-events-to-blocks>
5. EPC ← → Sawtooth Overheads:
   1. Transaction Issuance:
      1. This can be evaluated at Node 1 where the transactions are issued as NextEPC handler requests
      2. Every MongoDB insert operation contains an ObjectID structure along with the actual data which contains the timestamp when the data was actually inserted into the database.
      3. Log the timestamp as an epoch when the database handler was triggered for the specific transaction
         1. Use the ObjectID here as the unique identifier during logging or create your own.
         2. The difference between the timestamp recorded compared to the insert timestamp in the database will indicate possible delays in triggers and their handling and indicate any potential bottlenecks that could exist here.
   2. Post Transaction Confirmation and Block commit:
      1. Once the block is committed, the sawtooth events trigger which need to be applied to the local database through the do\_operation(). Log the timestamp before the do\_operation() is executed and if possible dynamically insert the hash value of the transaction used as the identifier into the value being inserted.
      2. The idea is to capture the timestamp of the actual insert compared to the insert that was issued by do\_operation() and see the final persistence time.
6. Network Consumption and Usage:
   1. Use psutil / poll the “/sys/class/net/<iface>/statistics/(rx|tx)\_bytes” and process the information of the RX/TX bytes at each node of the network at each second. Run as a separate process / in a separate thread.
   2. On each of the nodes, it’d be nice to have this information along with the epoch so that the events can be overlaid and correlated:
      1. TX Issuance DB Insert/Update in Mongo
      2. Transaction triggered to Sawtooth
      3. Block Commit events
      4. Blockchain State Persistence event
      5. Peer TX Update in Mongo

**Questions to ask ourselves based on the data before taking any other steps:**

1. What is the average delay between the information being updated by the EPC to it actually being triggered as a transaction on the blockchain?
2. What is the confirmation time of each transaction in the blockchain network?
   1. i.e. How long does it take in terms of time for the transaction to be committed to the blockchain? To the EPC database being used?
   2. i.e. How long does it take in terms of average number of blocks (block delay) for an issued transaction to be committed?
3. What is the average number of transactions which can be batched into a specific block when the configuration block transactions batch size are set at 100? Are we able to ever hit 100 tx/block?
4. What is the colte transaction processor overhead i.e. time delay between:
   1. MongoDB Update → TX Sent via propogate\_request()?
   2. TX Receive Post Confirm → MongoDB update
5. What is the bandwidth used at each node in the network? How much of the available iperf bandwidth do we actually consume?
   1. What is the exact expected consumption byte size? (SUM(tx\_Sizes))
   2. What is the overhead in the network transmission inserted by sawtooth?

**Design questions:**

1. What is the expected roaming movement in the real world?
2. What is the expected start number of users and the growth of the network users?
   1. 10s, 100s, 1000s?