CS765-HW1

Namoju Karthikeya, 200050084
Dhvanit Beniwal, 200050035

What are the theoretical reasons of choosing the exponential distribution?

If the time until an event happens is chosen from an exponential distribution then the event is modelled as being memory-less in that the probability of an event happening at a particular time does not depend on the time that has elapsed.

So if the probability of the event happening at time $t = t_1$ is $P(t_1)$ and it hasn't happened for $t_2 < t_1$ seconds then the probability that it still happens at $t = t_1$ is $P(t_1 - t_2)$. This way we can model each transaction as being generated independently of any generated so far, and there being no bound on how soon it must be generated.

Why is the mean of d_{ij} inversely related to c_{ij}?

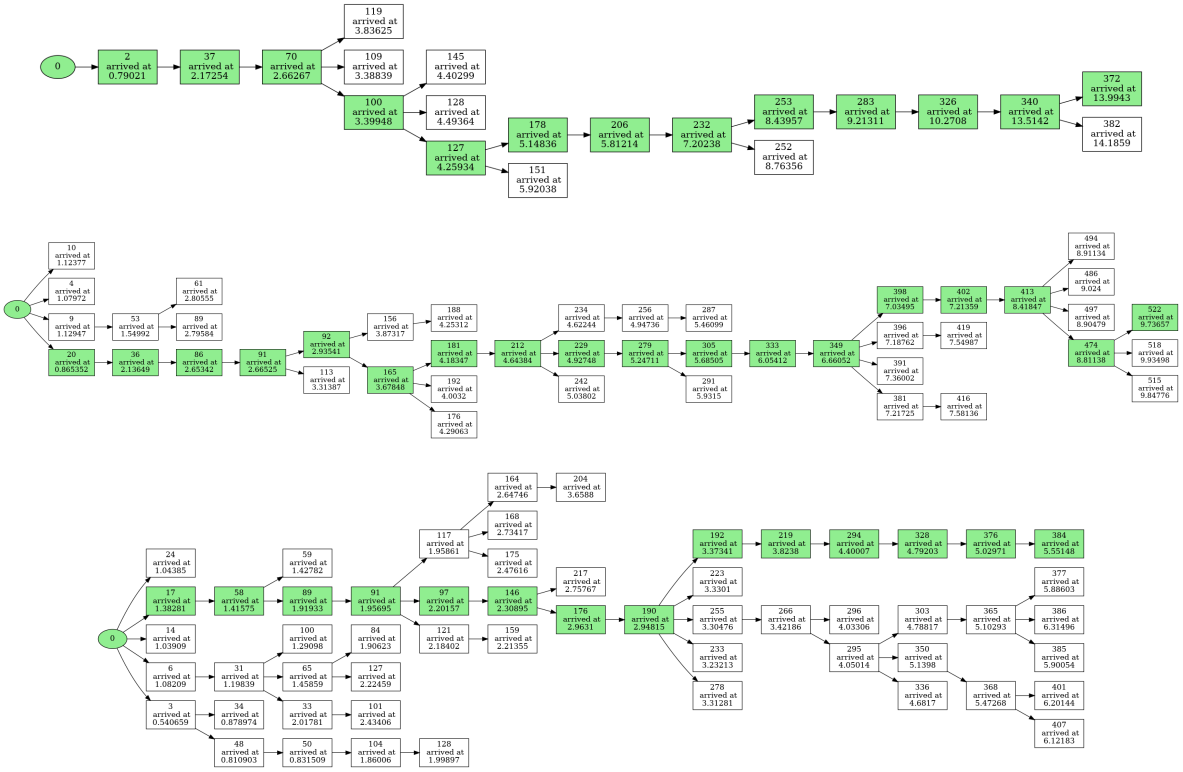
If at any given moment x bits of data are queued up waiting for their turn to exit the network interface, this queue will decrease exactly as fast as the link speed of outgoing data (c_{ij}). Which means it takes $\frac{x}{c_{ij}}$ seconds to empty the queue which is the amount of queuing delay we are modelling. If x is replaced by an average queue holdup size (in bits) then it makes sense for mean of d_{ij} to be inversely proportional to c_{ij}

1 Functions & Structs of interest

Please refer to the following functions or structs in the code for the implementation of critical aspects of the simulation

- `Network::GenerateRandomGraph()`
- `Network::Link::latency()`
- `Peer::ScheduleTxnGen()`
- `Peer::ReceiveAndForwardTxnOp()`
- `struct Blockchain::Block`
- `struct Blockchain::BlockMetaData`
- `Blockchain::CreateValidBlock()`
- `Blockchain::IsValid()`
- `Peer::ReceiveAndForwardBlkOp()`

2 Typical blockchain trees



3 Contribution of different types of peers

We observe that in the long run most blocks (whether they become a part of the longest chain or not) are generated by the High CPU subset of peers, which is to be expected. But if we are only interested in the percentage of blocks generated that happen to end up in the longest chain, we see that CPU power does not affect that distribution at all (except that there are less high CPU outliers where all or none of their blocks are accepted but that's only because they generate more blocks in total).

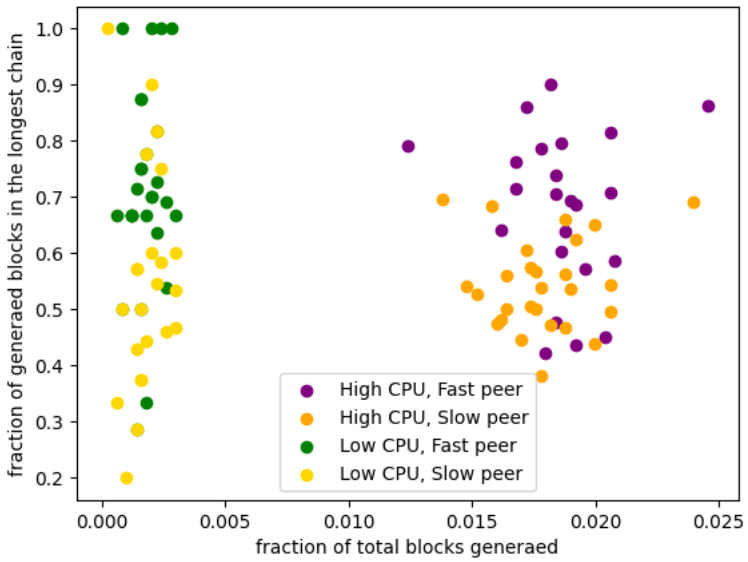


Figure 1: 100 peers, 50 slow and 50 low CPU

On the other hand whether the peer is fast or slow (in terms of link speed) directly affects the ratio of 'accepted' blocks. Given that a block has been generated, it is more likely to be on everyone's longest

chains if it reaches them earlier. Not only does it make other peers abort their current mining process sometimes, time of arrival is directly used to break ties if there is no unique longest chain.

4 Effects on branching of blockchain tree

In general, it would become a common theme where we notice that if any factor directly/indirectly increases link latency between peers, it would lead to an increased amount of branching. This is broadly because in a network of honest peers branches happen only because a peer broadcasts a block it was mining before it can be notified that another block for that parent has already been mined.

For large enough values of I (like $600s$ or any other reasonable value around this) the simulation almost always finishes with branch-less blockchains. Branches require high link latency *relative to* I. While good feature of a blockchain network, in our honest peer simulation it leaves us with no insight. So for all observations $I = 0.1s$ has been used as it was found to produce more dynamic blockchain trees this way.

We will study blockchain trees by the number of branches they have (number of leaves that are not the longest chain leaf) and the length of the longest/average branch.

4.1 Effect of z

We notice that as we increase the percentage of slow peers (z_0) we experience branching more often, and on average both the mean branch length and maximum branch length also increase.

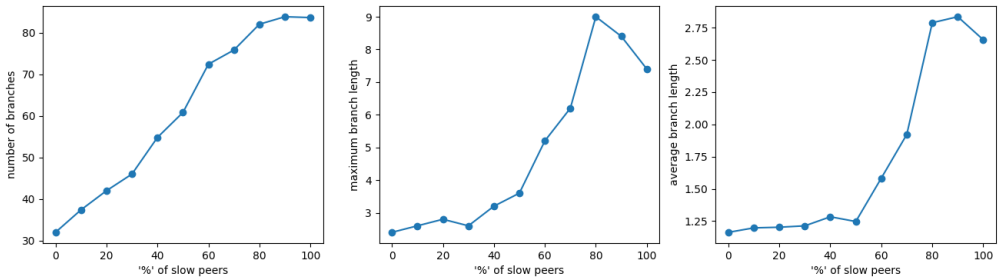


Figure 2: changing number of slow peers, total 100 peers, $I = 0.1s$

This makes sense because as we saw before, fast peers are more likely to only send blocks that would end up in the longest chain. If too many peers are slow then most of their generated blocks don't end up in the main branch. In fact they seem to reinforce this behaviour among each other as there is a sudden increase in branch lengths if slow peers become the majority ($> 50\%$). Number of branches, however, changes rather gradually. This also suggests that pockets of slow peers are likely to get stuck building an alternate branch if it takes too long for the network to correct them.

However if we now (keeping z_0 fixed) change only the percentage of low CPU peers (z_1), there is little variation in the number of branches and otherwise no such correlation to branch length either.

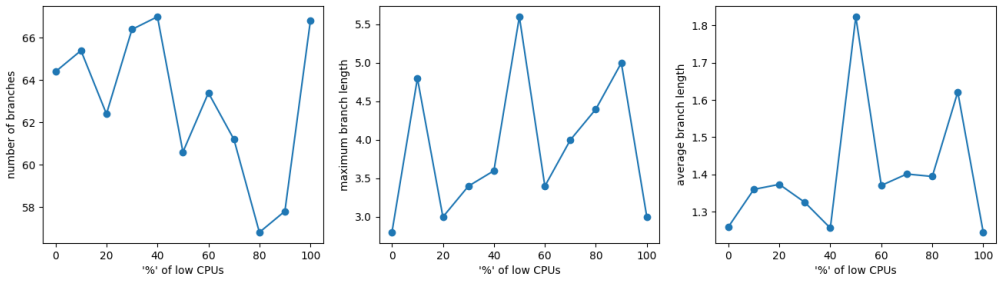


Figure 3: number of low CPUs seems to have no correlation

This is consistent with the previous observation that CPU power doesn't give an advantage to a given block ending up in the longest chain. Even though the above graphs look erratic, they represent a much smaller variation (on the y-axis) compared to when we were changing z_0, and they are only produced to argue the lack of correlation.

4.2 Effect of I (or mean T_k)

We might expect mean mining time to have similar effects as z_1 (CPU power). Instead we see that as we decrease I we get more branches because often multiple peers will mine their own blocks as the next one in the longest chain and broadcast them before receiving each others versions. It's as if the link latency was relatively increased.

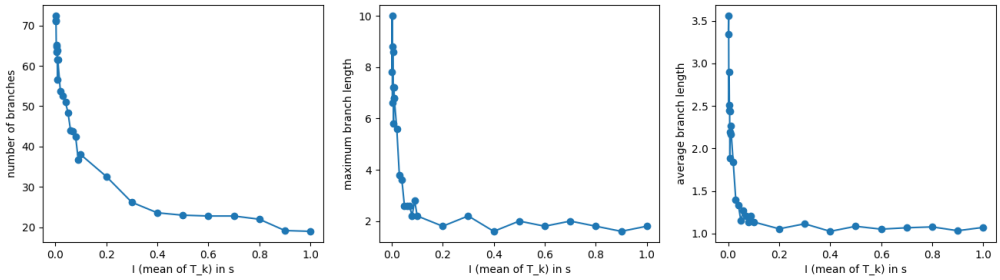


Figure 4: I needs to be very small to see reasonable increase in branch lengths

The reason for contrast in results from z_1 suggests that if the discrepancy between low and High CPU peers was much higher (than 10x), so that it has as discernable an effect on the average mining time as changing I directly, then we might have seen a similar trend in varying z_1 as well.

4.3 Effect of the average queue holdup ($T_{d_{ij}}$)

Its not clear what we are modeling exactly when we vary the average queue holdup at a peer (for example 96kbits) since it might represent the amount of data queued at the network interface that the peer is using to participate in this simulation. But as we increase this value, it's as if we are modeling a slower network , just with less significant effects since queueing delay is only partly responsible for link latency.

For the same reason, its effects are more pronounced with more percentage of slow peers, even if the same trend is followed otherwise. So for ease of visualisation, we assume the network has 100% slow peers. Overall we see that it leads to more branching, which is consistent with our other observations.

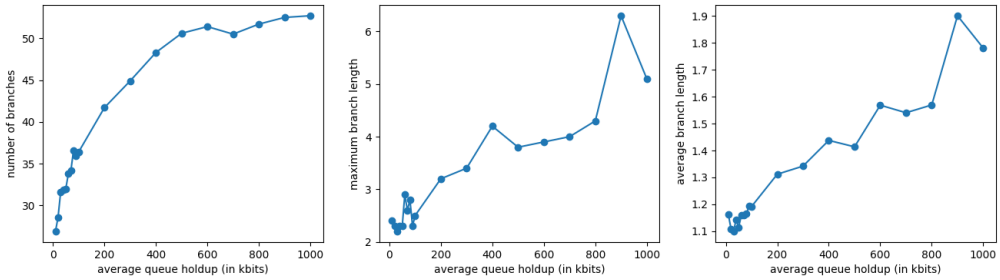


Figure 5: 100 peers, all slow, 50% low CPU, $T_{tx} = 10, I = 0.1$

4.4 Effect of T_{tx}

As T_{tx} keeps decreasing, the circulation of invalid blocks and transactions increases which might in practice affect processing time but we don't account for that in our simlulated time. The only effect

we see is that on average, block sizes are more likely to be larger which would directly increase link latency, even if not as much. And for the same reason as before (partly affecting link latency) the effect is more pronounced if network has more slow peers already. Overall, again, we get more branching.

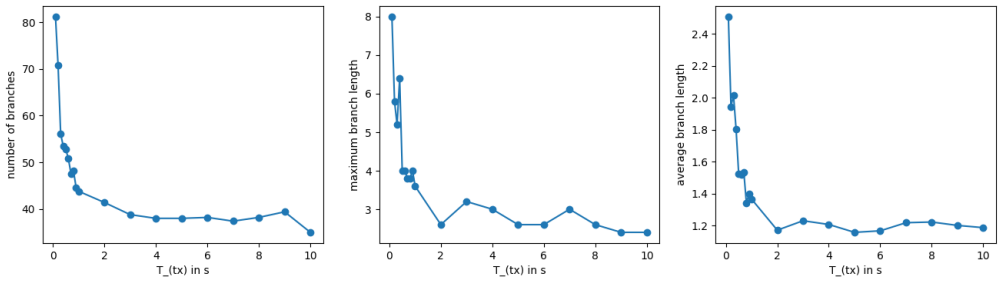


Figure 6: 100 peers, all slow, 50% low CPU, $I = 0.1$

We also notice that it takes a certain threshold above which there wasn't any significant variation in branching and below which there is a sudden increase. This might have to do with the fact that given the average mining time, CPU power of the network etc, there exists a particular rate of generation of transactions only after which can transactions reasonably accumulate for peers to bundle thousands of them in a block.

4.5 Effect of n (total number of peers)

As we increase the number of peers while keeping everything else the same, we witness more branching but don't witness an increase in branch length. This might suggest that the increase in branches is a false observation because it is only a result of more blocks being produced in total and doesn't say anything about the involvement of n.

However we can set our simulation stopping criteria as : everyone will stop mining new blocks when a peer has reached a blockchain size of `max_blocks` (say 100). Peers might end up with a few blocks over this limit as blocks mined in the past might still reach them after simulation has decided to finish. So as n increases, if we now see blockchains of similar size each time but more branches at each step then that is not a false observation. We therefore have plotted this 'branches per unit block' metric.

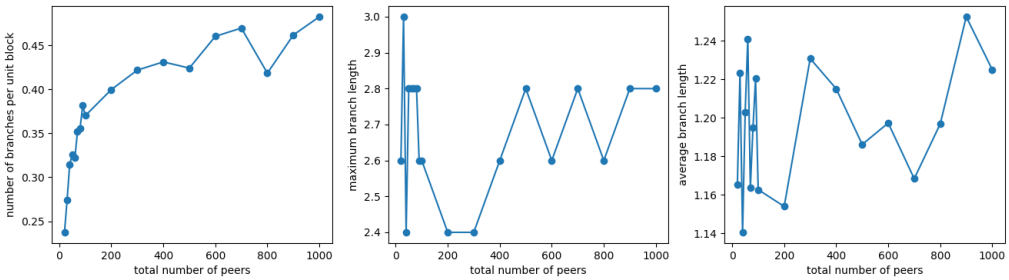


Figure 7: all slow, 50% low CPU, $I = 0.1$

The reason for increased branching is still link latency. Even though latency between connected peers is the same, the apparent latency between far away nodes in the network keeps increasing because of an upper bound on the degree of nodes. Interestingly, this is different from other instances of increased link latency as only nodes far away are affected and not all nodes.

The lack of correlation with branch length is perhaps due to the fact that for branches to keep growing, you can't rely on the same distant nodes that started the branch, but all the nodes in between must also be equally likely to continue that branch. It also gives intuition for why the graph depicting increased branching starts to stagnate. It's because the number of nodes that are k links away from a peer needs to increase exponentially as k increases linearly.