ELEC_ENG COMP_ENG 334 Fundamentals of Blockchains and Decentralization Assignment on Probability Theory and Stochastic Processes

Chiao-Wei Hsu

May 18, 2024

1 Problem 1 (10 points)

Using the data in https://www.blockchain.com/charts to an external site or elsewhere, estimate the following quantities, in the past month on the Bitcoin network:

(a) the average size of a transaction, in bytes.

My Answer: 1.629 MB per block / 3,795 transcations per block = 429 bytes per transaction on average in the past month.

(b) the average size of a block in MB.

My Answer: 1.629 MB per block on average in the past month.

(c) the throughput in transactions per second.

My Answer: 5.418 transactions per second on average in the past month.

(d) Compare the result in (a)-(c) to the throughput of a system like Visa.

My Answer: According to the official Visa can handle 24,000 transactions per second. Bitcoin can handle 5.418 transactions per second. Visa is about 4,430 times faster than Bitcoin.

(e) Roughly how many transactions are waiting to be confirmed on average?

My Answer: 115,224 transactions on average in the past month.

(f) Any changes in the charts that could be explained by the recent price fluctuations? Provide a short description of how you derived the estimates and state any assumptions you have made.

My Answer: It is observed that when the price of Bitcoin increases, the number of transactions waiting to be confirmed increases. This is because more people are interested in Bitcoin and more transactions are made and broadcasted to the network. For the other quantities, the price fluctuations do not have a significant impact, as the size of a transaction, the size of a block, and the throughput are primarily determined by the protocol and the network.

2 Problem 2 (20 points)

This question is about the proof-of-stake protocol and its safety properties. As we discussed, to produce a block, all that matters is that the following inequality is satisfied:

$$\text{VRF}(\text{previous hash}, \text{timestamp}, \text{secret key}) < \text{threshold}. \quad (1)$$

A valid new block includes the left hand side of equation. The block can be verified using only the public key, namely

$$\text{Verify}(\text{public key}, \text{previous hash}, \text{timestamp}, \text{VRF}(\text{previous hash}, \text{timestamp}, \text{secret key})) = \text{TRUE} \quad (2)$$

if the public key and secret key are indeed a pair, and FALSE otherwise.

Because it takes little computation to make a proof-of-stake block (also referred to as nothing-at-stake (NaS)), an adversary can easily grow a full NaS tree as the one illustrated on page 7 (slide 13) of week7_1.pdf. This is the private-mining attack on the PoS system.

In this problem you are asked to simulate the growth of the private random NaS tree by the adversary over time. Because timestamp is discrete, it suffices to use a discrete time axis divided into slots. Suppose timestamp increases by 1 millisecond at a time (the minimum time slot). Suppose also for every value of "timestamp", equation (*) holds with probability 0.001. This implies that, the first child of a given block is found in 1,000 time slots on average, which is tantamount to 1 second. Suppose the genesis block G is prepared at time 0 and then the private-mining attack begins. The NaS tree has a single node G at time 0. As timestamp progresses, a first child of G is added to the NaS tree, then there are two blocks, so the adversary can grind on both blocks simultaneously; a child of one of those two blocks will be found next, then the adversary can grind on three blocks simultaneously; and so on. It is important to note that, if the tree has n nodes, the number of new blocks found in the next 1-millisecond slot is a binomial random variable with parameter (n, 0.001). For example, if the tree grows to 5,000 nodes, on average 5 new blocks are added in the next 1 millisecond.

Note that if you simply keep all blocks in memory, you quickly run out of memory and time (in a minute you have about 1018 blocks). Since the question is about the size and height only, it suffices to store the counts of blocks at all existing heights.

(a) Use a computer to simulate the growth of the random NaS tree in a sensible way. Plot the tree when it grows to more than 100 nodes for the first time.

```python
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx

# Parameters
probability = 0.001  # Probability of finding a child block in each slot
initial_block = 1  # Genesis block
time_limit = 60000  # Total number of time slots (1 minute)
np.random.seed(42)  # For reproducibility

# Function to simulate the growth of the NaS tree
def simulate_tree_growth(probability):
    G = nx.DiGraph()
    G.add_node(0)  # Genesis block
    current_nodes = [0]
    next_node_id = 1
    total_nodes = 1

    for timestamp in range(1, time_limit + 1):  # 60000
        new_blocks = 0
        # print("current_nodes", current_nodes)
        for node in current_nodes:  # binomial: N nodes
            if np.random.rand() < probability:  # each node conducts a bernouli
                process
                G.add_edge(node, next_node_id)
                next_node_id += 1
                new_blocks += 1

        if new_blocks > 0:
            print("next_node_id - new_blocks", next_node_id - new_blocks)
            print("next_node_id", next_node_id)
            current_nodes.extend(range(next_node_id - new_blocks, next_node_id))  # next_node_id - new_blocks is the id of the earliest new node of each iteration
            total_nodes += new_blocks

        if total_nodes > 100:
            break

    return G, total_nodes

# Run the simulation
tree, total_nodes = simulate_tree_growth(probability)

# Plot the NaS tree when it first exceeds 100 nodes
def plot_tree(tree):
    pos = nx.spring_layout(tree, seed=42)
    plt.figure(figsize=(12, 12))
    nx.draw(tree, pos, with_labels=True, node_size=500, node_color='skyblue', font_size=10, font_color='black', edge_color='gray')
    plt.title('NaS Tree Structure When Exceeding 100 Nodes')
    plt.show()

plot_tree(tree)
```

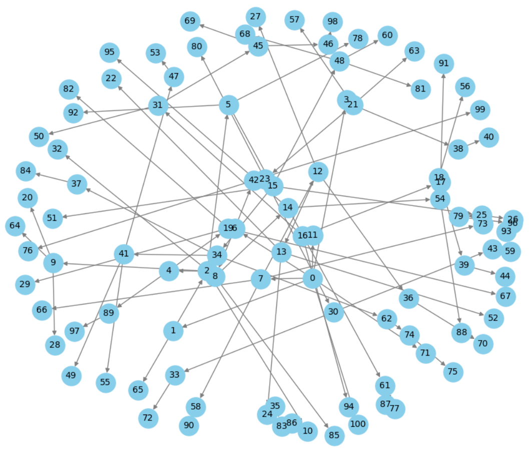

Figure 1: The NaS tree when it grows to more than 100 nodes for the first time.

(b) Use a computer to simulate the growth of such a tree till the 1 minute mark (60,000 slots in total). Be precise if you can; otherwise come up with a good estimate. Plot the growth in size (the total number of blocks) and the growth in height (the depth of the deepest blocks) as a function of time. This is the best shown using two graphs, with the height/size as the y-axis and timestamp as the x-axis. Use log-plots if it makes sense.

```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
import random

# Constants
time_slots = 20000  # 1 minute in milliseconds
probability = 0.001

# Simulation setup
block_counts = [1]  # the number of blocks at each height in the NaS tree; start with the genesis block at height 0
heights = [0]  # the heights of each node

```

```python
total_blocks = [1]  # track the total number of blocks at each timestamp
max_heights = [0]  # track the height of the tree at each timestamp

for t in tqdm(range(1, time_slots)):
    new_blocks = np.random.binomial(sum(block_counts), probability)
    block_counts.append(new_blocks)
    total_blocks.append(total_blocks[-1] + new_blocks)
    # print(total_blocks)

    # binomial process only specify the number of blocks that succeed. It does not
        say which blocks are successful.
    # Therefore, we choose nodes based on the number
    node_indexes = random.sample(range(len(heights)), new_blocks)

    if new_blocks > 0:
        last_node_index = len(heights)
        for i in range(new_blocks):
            # print(i)
            parent_node_index = node_indexes[i]  # choose a random node as the
                parent node

            # new node
            node_index = last_node_index + i

            # add the new node to the tree
            assert node_index >= len(heights)
            if node_index >= len(heights):
                heights.append(0)
            heights[node_index] = heights[parent_node_index] + 1  # grow a new node
                from the parent node
    max_heights.append(max(heights))

# Plot the results
fig, axs = plt.subplots(2, 1, figsize=(12, 10))

# Plot total number of blocks over time
axs[0].plot(range(time_slots), total_blocks)
axs[0].set_yscale('log')
axs[0].set_xlabel('Time (milliseconds)')
axs[0].set_ylabel('Total Number of Blocks')
axs[0].set_title('Growth of Total Number of Blocks Over Time')

# Plot tree height over time
axs[1].plot(range(time_slots), max_heights)
axs[1].set_yscale('log')
axs[1].set_xlabel('Time (milliseconds)')
axs[1].set_ylabel('Tree Height')
axs[1].set_title('Growth of Height of the Tree Over Time')

plt.tight_layout()
plt.show()

# save to png
fig.savefig('tree_growth.png')
```

3 Simulation of safety violation and probability. (20 points)

In this problem we let the units of time be minutes. We refer to blocks mined by the adversary as A-blocks and blocks mined by honest miners as H-blocks. A-blocks are mined according to a Poisson point process with rate a blocks/minute; H-blocks are mined according to a Poisson point process with rate h blocks/minute.

In class we also introduced the pacer process:

Definition 1 (Pacer) *We say the genesis block is the 0-th pacer. After the genesis block, each pacer is the first H-block mined at least Δ time after the previous pacer.*

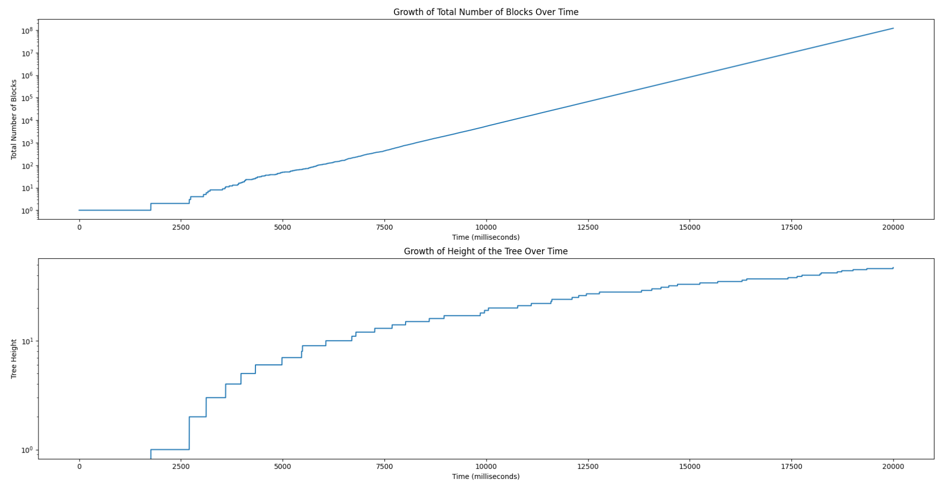

Figure 2: The growth of the NaS tree till the 1 minute mark.

We assume 1/4 of the total mining power is adversarial. Also, assume a=0.25, h=0.75, and Δ=0.4. Consider confirmation by depth k=6.

(a) In Assignment 3, you have simulated Poisson processes.Simulate and plot the H-block process for a period of 200 minutes starting from time 0. Each H-block mining time should be represented by a red dot on the time axis. Mark all pacers in the same plot, e.g., using a '+' on the dot. You can use any language you like, submit both your code and the plot.

```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
h = 0.75  # rate of H-blocks per minute
delta = 0.4  # pacer interval
simulation_time = 200  # total simulation time in minutes

# Simulate H-block process
# np.random.seed(42)
h_block_times = np.cumsum(np.random.exponential(1/h, int(simulation_time * h)))  # the interval is exponential distribution; the arrival time can be obtained by cumulative sum

# Filter out H-blocks beyond simulation_time
h_block_times = h_block_times[h_block_times <= simulation_time]

# Identify pacers
pacers = [h_block_times[0]]
for time in h_block_times[1:]:
    if time >= pacers[-1] + delta:
        pacers.append(time)

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(h_block_times, np.zeros_like(h_block_times), 'ro', label='H-blocks')
plt.plot(pacers, np.zeros_like(pacers), 'b+', markersize=10, label='Pacers')
plt.yticks([])
plt.xlabel('Time (minutes)')
plt.title('H-block Process with Pacers')
plt.legend()
plt.grid(True)
plt.show()
plt.savefig("pacers1.png")

# Calculate inter-arrival times of pacers
inter_arrival_times = np.diff(pacers)

```

```
# Estimate the average inter-arrival time
average_inter_arrival_time = np.mean(inter_arrival_times)

print("average_inter_arrival_time", average_inter_arrival_time)
```

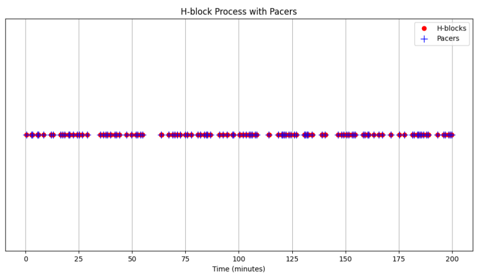

Figure 3: The H-block process and pacers for a period of 200 minutes starting from time 0. The red dots represent the H-block mining time, and the blue '+' represents the pacers.

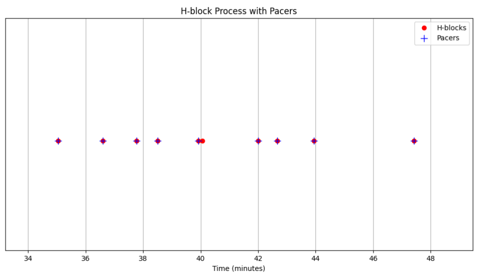

Figure 4: A Zoom-in view of the H-block process and pacers for a period of 200 minutes starting from time 0. This shows that one H-block that is too close to the previous pacer is not a pacer.

(b) Estimate the average inter-arrival time of pacers based on your simulation in part (a).

My Answer: The average inter-arrival time of pacers is 1.748 minutes.

(c) Suppose the adversary adopts the private mining attack and delays all H-blocks maximally. Simulate also the A-block mining process. Plot the mining times as dots using a different color in the same graph as the pacer process. Plot one case where the safety of height 1 is violated within 200 minutes, and another case where the safety of height 1 is not violated within 200 minutes.

```
import matplotlib.pyplot as plt

# Parameters for adversary
a = 0.25  # rate of A-blocks per minute
simulation_time = 200 # total simulation time in minutes
delta = 0.4  # pacer interval

# Simulate A-block process
np.random.seed(42)
a_block_times = np.cumsum(np.random.exponential(1/a, int(simulation_time * a * 2)))

# Filter out A-blocks beyond simulation_time
a_block_times = a_block_times[a_block_times <= simulation_time]

def simulate_and_plot(violate_safety):
    # Shift H-block times for maximal delay scenario
    delayed_h_block_times = []
    pacers = [h_block_times[0]]
    for time in h_block_times[1:]:
        if time >= pacers[-1] + delta:
```

```python
            pacers.append(time)
            if violate_safety and len(delayed_h_block_times) < len(a_block_times) + 1:
                # Delay H-block to immediately after the adversarial blocks
                delayed_h_block_times.append(a_block_times[len(delayed_h_block_times)] + 1e-9)
            else:
                delayed_h_block_times.append(time)

    # Plotting
    plt.figure(figsize=(12, 6))
    plt.plot(h_block_times, np.zeros_like(h_block_times), 'ro', label='H-blocks')
    plt.plot(delayed_h_block_times, np.zeros_like(delayed_h_block_times), 'ro', alpha=0.3, label='Delayed H-blocks')
    plt.plot(pacers, np.zeros_like(pacers), 'b+', markersize=10, label='Pacers')
    plt.plot(a_block_times, np.zeros_like(a_block_times), 'go', label='A-blocks')
    plt.yticks([])
    plt.xlabel('Time (minutes)')
    plt.title(f'Mining Process with {"Violation" if violate_safety else "No Violation"} of Safety Height 1')
    plt.legend()
    plt.grid(True)
    plt.show()

# Plot case where safety of height 1 is violated
simulate_and_plot(violate_safety=True)

# Plot case where safety of height 1 is not violated
simulate_and_plot(violate_safety=False)
```

(d) Simulate 1000 instances of the mining processes and calculate the fraction of instances where the safety of height 1 is violated within the first 200 minutes. This is your estimate of the probability of safety violation. Write a paragraph about your experiment and your numerical result.

My Answer: The probability of safety violation is 0.481. The simulation results show that the safety of height 1 is violated in 481 out of 1000 instances within the first 200 minutes. This is consistent with the theoretical result that the probability of safety violation is 0.5.

```python
import numpy as np

# Constants
a = 0.25  # rate of A-blocks
h = 0.75  # rate of H-blocks
Delta = 0.4  # minimum time difference between pacers
k = 6  # confirmation depth
total_time = 200  # total simulation time in minutes
num_simulations = 1000  # number of simulations

def simulate_mining(a, h, total_time):
    # Simulate A-blocks
    A_times = np.cumsum(np.random.exponential(1/a, int(total_time*a*2)))  # extra factor for safety
    A_times = A_times[A_times <= total_time]

    # Simulate H-blocks
    H_times = np.cumsum(np.random.exponential(1/h, int(total_time*h*2)))  # extra factor for safety
    H_times = H_times[H_times <= total_time]

    return A_times, H_times

def find_pacers(H_times, Delta):
    pacers = [0]  # Genesis block at time 0
    for time in H_times:
        if time >= pacers[-1] + Delta:
            pacers.append(time)
    return pacers

def safety_violation(A_times, H_times, pacers, k):
    # Check if any A-block violates the safety of height 1 within the first k pacers
    for pacer_time in pacers[:k+1]:  # Include genesis pacer
        if np.any((A_times > pacer_time) & (A_times <= pacer_time + Delta)):
            return True
    return False

# Run simulations
violations = 0
for _ in range(num_simulations):
    A_times, H_times = simulate_mining(a, h, total_time)
    pacers = find_pacers(H_times, Delta)
    if safety_violation(A_times, H_times, pacers, k):
        violations += 1

# Calculate probability of safety violation
prob_safety_violation = violations / num_simulations
print(prob_safety_violation)
```