2CSDE93- Blockchain Technology

Lab 3- To perform thorough study and installation of Anaconda 5.0.1 and Python 3.6 and perform proof of work (POW) consensus mechanism. Also, notice the changes in mining rewards and nonce requirement.

Outline

- Prerequisites
- Bitcoin Fundamentals
- What is Proof-of-Work (PoW)
- Implementation of PoW (with changes in nonce and mining rewards)
- Conclusion

PREREQUISITES

PREREQUISITES

To implement this lab, you’ll need to have the following:

1. Anaconda 5.0.1 installed on your machine. You can download it from

(https://docs.anaconda.com/anaconda/install/)

2. Python 3.6 installed in your machine. You can download Python 3.6 from (https://www.python.org/downloads/) , or can use Google Collab, or Jupyter Notebook

(https://docs.anaconda.com/anaconda/install/)

ANACONDA. DOCUMENTATION

- Home
- Anaconda Individual Edition
  - Installation
    - Installing on Windows
    - Installing on macOS
    - Installing on Linux
    - Installing on Linux-aarch64 (arm64)
    - Installing on AWS Graviton2 (arm64)
    - Installing on Linux-s390x (IBM Z)
    - Installing on Linux POWER

Installation

Review the system requirements listed below before installing Anaconda Individual Edition. If you don't want the hundreds of packages included with Anaconda, you can install Miniconda, a mini version of Anaconda that includes just conda, its dependencies, and Python.

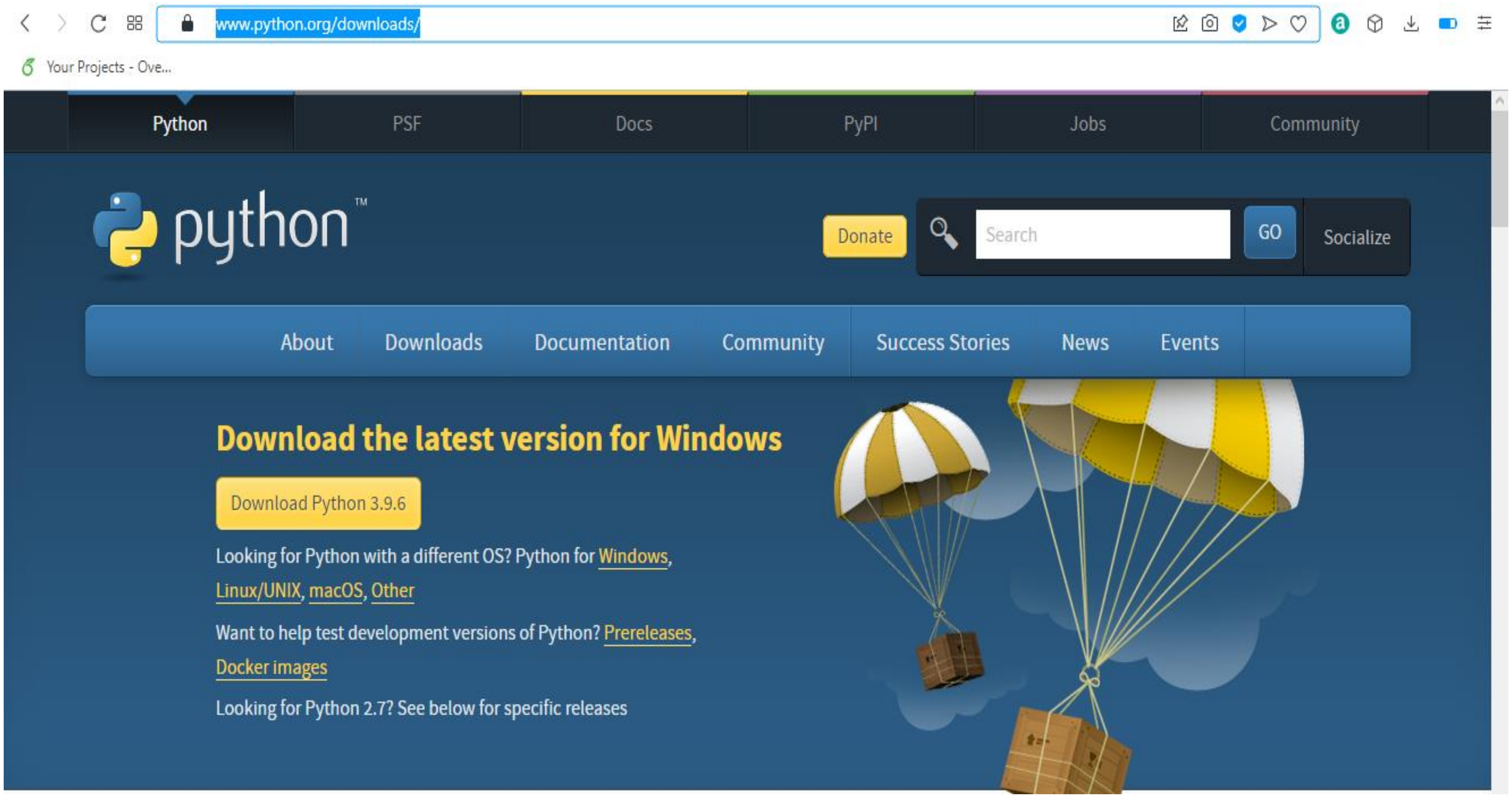
Tip

Looking for Python 3.5 or 3.6? See our FAQ.

System requirements

- License: Free use and redistribution under the terms of the EULA for Anaconda Individual Edition.
- Operating system: Windows 8 or newer, 64-bit macOS 10.13+, or Linux, including Ubuntu, RedHat, CentOS 7+, and others.
- If your operating system is older than what is currently supported, you can find older versions of the Anaconda installers in our archive that might work for you. See Using Anaconda on older operating systems for version recommendations.

(https://www.python.org/downloads/)



BITCOIN FUNDAMENTALS

BITCOIN FUNDAMENTALS

- Authentication → Public Key Crypto: Digital Signatures
  - Am I paying the right person? Not some other impersonator?
- Integrity → Digital Signatures and Cryptographic Hash
  - Is the coin double-spent?
  - Can an attacker reverse or change transactions?
- Availability→ Broadcast messages to the P2P network
  - Can I make a transaction anytime I want?
- Confidentiality→ Pseudonymity
  - Are my transactions private? Anonymous?

BITCOIN FUNDAMENTALS

- Validation
  - Is the coin legit? (proof-of-work) → Use of Cryptographic Hashes
  - How do you prevent a coin from double-spending? → Broadcast to all nodes
- Creation of a virtual coin/note
  - How is it created in the first place? → Provide incentives for miners
  - How do you prevent inflation? (What prevents anyone from creating lots of coins?) → Limit the creation rate of the BitCoins

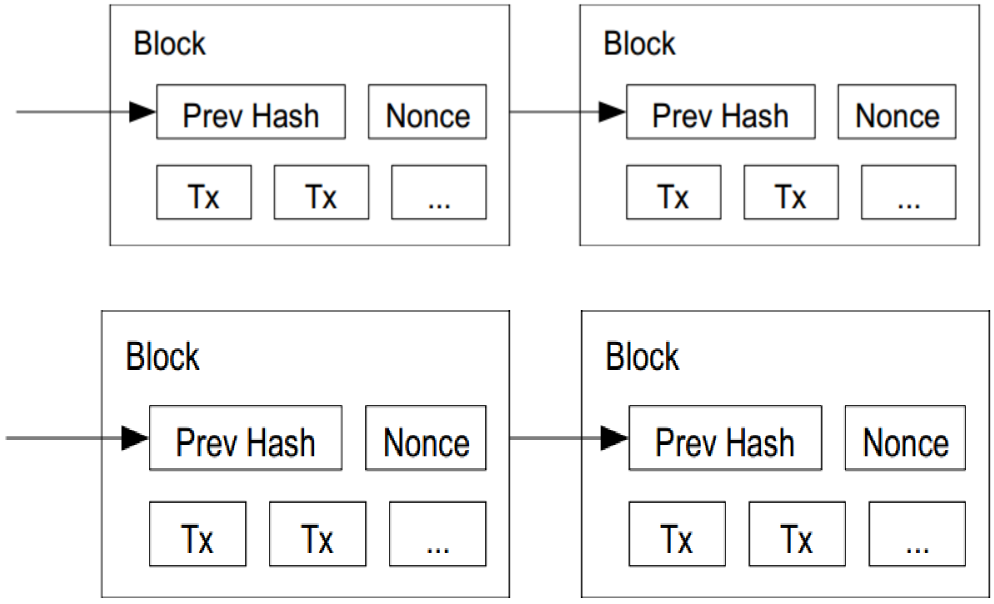
How Bitcoins Work??

- Each P2P node runs the following algorithm:
  - New transactions are broadcast to all nodes.
  - Each node (miners) collects new transactions into a block.
  - Each node works on finding a proof-of-work for its block. (Hard to do. Probabilistic. The one to finish early will probably win.)
  - When a node finds a proof-of-work, it broadcasts the block to all nodes.
  - Nodes accept the block only if all transactions in it are valid (digital signature checking) and not already spent (check all the transactions).
  - Nodes express their acceptance by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

What in case of Mining Tie??

- Two nodes may find a correct block simultaneously.
  - Keep both and work on the first one
  - If one grows longer than the other, take the longer one

Two different block chains (or blocks) may satisfy the required proof-of-work.



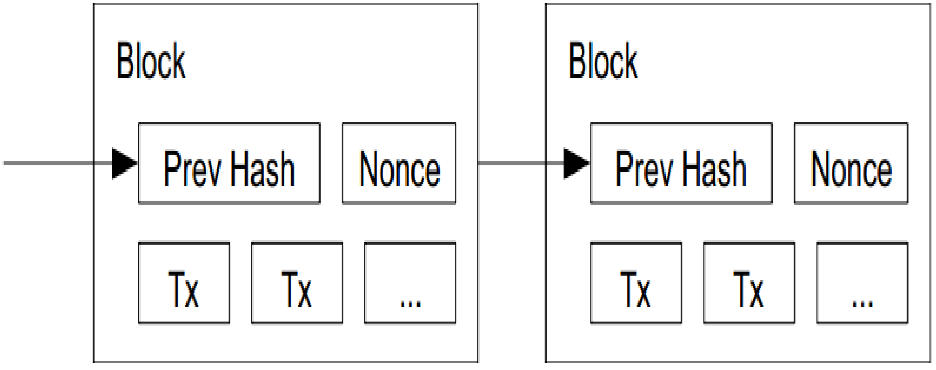
Bitcoin Economics

- Rate limiting on the creation of a new block
  - Adapt to the “network’s capacity”
  - A block created every 10 mins (six blocks every hour)
    - How? Difficulty is adjusted every two weeks to keep the rate fixed as capacity/computing power increases
- N new Bitcoins per each new block: credited to the miner → incentives for miners
  - N was 50 initially. In 2013, N=25.
  - Halved every 210,000 blocks (every four years)
  - Thus, the total number of BitCoins will not exceed 21 million. (After this miner takes a fee)

WHAT IS A PROOF-OF-WORK?

What is Proof-of-Work?

- Block contains transactions to be validated and previous hash value.
- Pick a nonce such that **H(*prev_hash*, *nonce*, *Tx*) < *E***.
- ***E*** is a variable that the system specifies. Basically, this amounts to finding a **hash value who's leading bits are zero.**
- The work required is exponential in the number of zero bits required. Verification is easy.
- But proof-of-work is hard.



Proof-of-Work (PoW) illustration

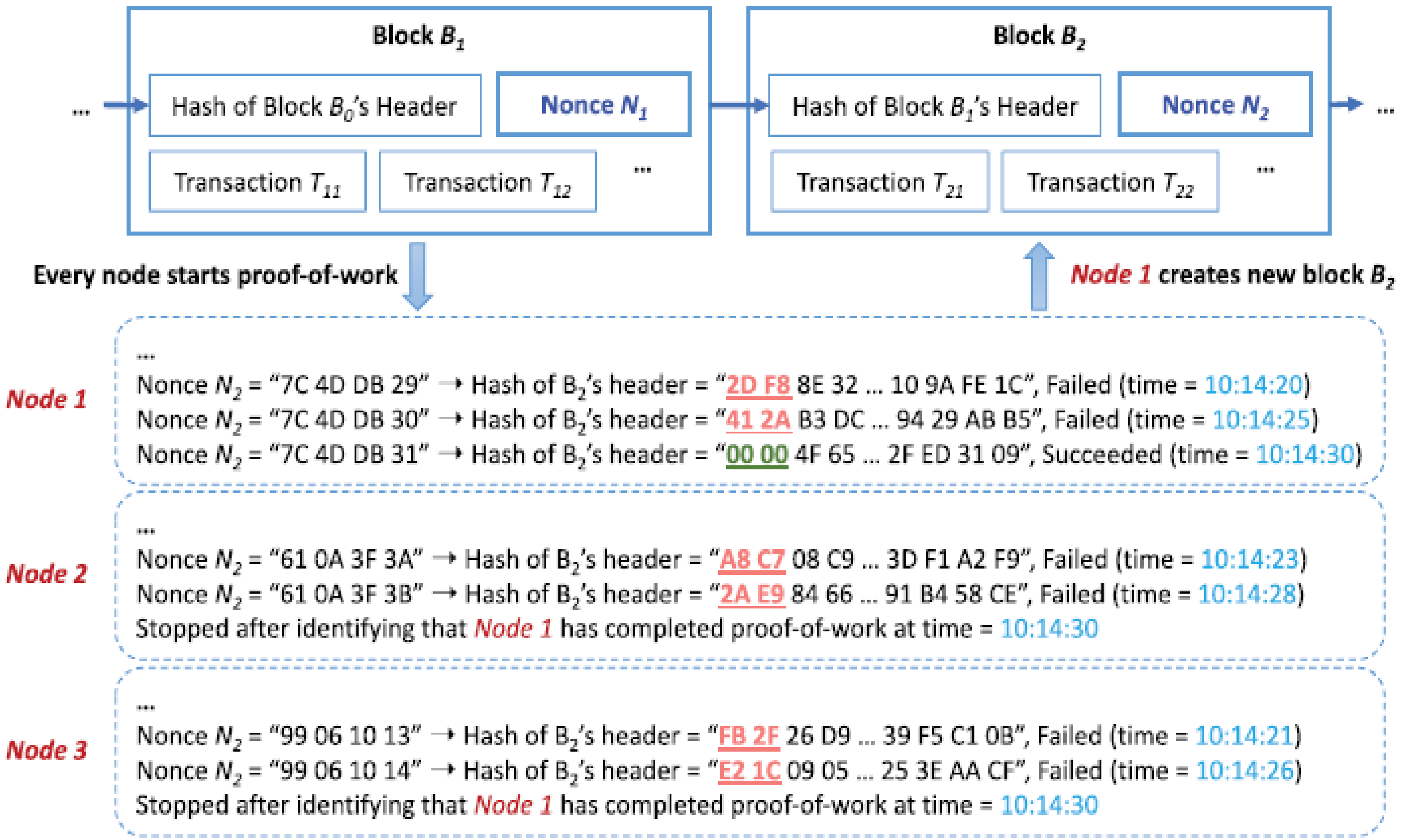


Figure adapted from : Kuo, Tsung-Ting, Hyeon-Eui Kim, and Lucila Ohno-Machado. "Blockchain distributed ledger technologies for biomedical and health care applications." *Journal of the American Medical Informatics Association* 24.6 (2017): 1211-1220.

PoW Explanation (1)

- An example of the **nonce mechanism** for the proof-of-work protocol.
- Each block contains an additional "***nonce***" (**32-bit or 8-hex-digits** in this example), which is a **counter** that serves as one of the inputs of the hashing function.
- To "**proof**" the hashing work, the **nonce is incremented by one bit each time** for the **hash computation (ie, the "work")**, until the hashed value (**256-bit or 64-hex-digits in this example**) contains a **predefined number of leading zero bits** (ie, "**proof**" of the work, **16-bit or 4-hex-digits** in this example).

PoW Explanation (2)

- Meanwhile, the newly generated **unconfirmed transactions** are collected in a memory pool on each node.
- The first node that successfully completes the **proof-of-work (Node 1 at 10:14:30 in this example)** has the privilege to create a new block (B2 in this example), verify the transactions, move the **confirmed transactions from the memory pool to a newly created block**, and **add the block to the end of the longest chain** (if there are competing chains).
- It also gets paid (eg, **6.125 bitcoins**) for this work in Bitcoin Blockchain. Also, the remaining nodes (Nodes 2 and 3 in this example) stop the proof-of-work mining for B2 when Node 1 completes the proof-of-work.

PoW Explanation (3)

- This way, the **mining process** becomes difficult (ie, one needs to compute the difficult hashing problem by trying different “nonce” values), while the **checking process remains easy** (ie, just one hash to see if the predefined leading bits are all zeroes).
- In our example, Alice cannot easily create an invalid block for **her double-spend transaction**, while Bob and Charlie can easily check that the block Alice created is invalid.
- It should be noted that the **system clocks** on the nodes may not be synchronized, therefore we use a **global time** for demonstration purposes in this example.
- Also note that, if **an attacker** modifies any of the transactions in block B1, the value of “**hash of block B1’s header**” and thus **block B2** need to be recalculated, and consequently **all blocks after B1** (ie, B2 B3, B4, B5,. . .) also need to be recomputed. Therefore, the computational cost of attacking becomes prohibitively high.

IMPLEMENTATION OF PROOF-OF-WORK (PoW) (with changes in difficulty and mining rewards)

Implementation Steps

- Import the hashlib library
- Make a block with list of transactions
- Import the Pickle library to create a block dump
- Compute the digest of the block
- Code the difficulty problem in terms of nonce.
- Mine the blocks

Transactions added to blocks

```
import hashlib


block = {
    'transactions': [
                     {
                        'from': 'Prof. Pronaya Bhattacharya',
                         'to' : 'Prof. Umesh Bodkhe',
                         'amount': 15,
                         'message': 'Transferred 15 coins from Prof. Pronaya Bhattacharya
                     },
                     {
                        'from': 'Prof. Umesh Bodkhe',
                         'to' : 'Prof. Rajesh Gupta',
                         'amount': 20,
                         'message': 'Transferred 20 coins from Prof. Umesh Bodkhe to Prof.
                     },
                     {
                        'from': 'Prof. Rajesh Gupta',
                         'to' : 'Prof. Sudeep Tanwar',
                         'amount': 22,
                         'message': 'Transferred 22 coins from Prof. Rajesh Gupta to Prof.
                     }
                    ]
      }
```

Creating hash of block dumps using pickle

```
import pickle

pickle.dumps(block)
```

```
b'\x80\x03}q\x00X\x0c\x00\x00\x00transactionsq\x01]q\x02(}q\x03(X\x04\x00\x00\x00fror
```

```
m = hashlib.sha3_256()
m.update(pickle.dumps(block))
m.digest()
m.hexdigest()
```

Setting of difficulty value

```
import time

difficulty=1

difficulty_string = ''.join(('0' for x in range(difficulty)))

print(difficulty_string)
    0
```

PoW Mining Code Snippet

```
nonce = 1
last_block['nonce'] = 1
start = time.time()
q = hashlib.sha3_256()
while q.hexdigest() [:difficulty] != difficulty_string:
            nonce += 1
            last_block['nonce'] = nonce
            q = hashlib.sha3_256()
            q.update(pickle.dumps(last_block))
            print(nonce, q.hexdigest())
total_time = (time.time()-start)
print('PoW Mining took' + ' ' + str(total_time) + ' ' + 'seconds')
```

Conclusion

- That’s how you can build a PoW mining algorithm, and change the difficulty and compute the mining incentive.

Thank You