Assignment 3: Blockchain

Topics covered

- UDP communication
- Synchronization
- Consensus
- Messaging strategies
- State handling

Create a Blockchain peer that stores messages in blocks. Each block contains

- the name of who mined the block
- 1 to 10 messages
- a nonce (string [really bytes…])
- A known hash (this can be re-calculated every time) with a minimum difficulty. Difficulty is currently set to 8.
- A height (again, can be calculated)

Some constraints:

1. Messages have a maximum length of 20 characters (keeps us under MTU)
2. Blocks have a maximum of 10 messages in them.
3. A nonce is a string (really, bytes) and must be less than 40 characters

Consensus is proof-of-work, where mining finds a nonce that has causes the resulting hash to a certain number of `0`s after it - this is the "difficulty", which is usually expressed in a number. The miners must find a nonce that creates a full block hash with difficulty number of `0`s. True blockchains use binary `0`s (ie. 0b0010 has 2 leading 0s in binary), while we will use ascii `0`s.

Joining the network is done with gossiping. A well-known host is available at `silicon.cs.umanitoba.ca:8999`. This host will forward the `GOSSIP` message to all known peers. Gossiping will be used as a 'keepalive' ping - re-`GOSSIP` every 30s to ensure you are not assumed offline. If you have not been heard from in a minute, you will be dropped. The message will be repeated to 3 peers that your peer is tracking, who will then also gossip. All peers hosted by us also have a web server to show the current status on port 8998, example: `http://silicon.cs.umanitoba.ca:8998/`. After gossiping, you should see your name on the peer list.

Upon receiving a GOSSIP message, you should reply using `GOSSIP-REPLY` to the originating sender - to the host and port in the message.

You will need to keep a list of your peers to send updates to them.

Upon joining - do a consensus to find the longest chain. Request `STATS` from everyone. Choose the most-agreed-upon chain (same height/hash combination). Load it by sending `GET_BLOCK` requests distributed all the peers that agree on that chain. Remember: some requests will get lost! You will likely need to resend some requests

requests will get lost! You will likely need to resend some requests.

Once you have the chain, you must verify it. There are cryptographic properties that must be respected by blockchains.

This course is *not* a cryptography course! Use hashlib to do the cryptographic one-way hash. The hash **must** respect the following order

1. The hash from the previous block (unless this is the genesis block, in which case this is skipped)
2. The name of the miner in `minedBy`
3. The messages in the order provided
4. The time the block was mined
5. The nonce

This will create the hash for *this block*. Which is, in turned the the first input to the hash of the next block. This is called “cipher block chaining”.

Code example, if we were trying to add a new block to the top of our chain:

```
hashBase = hashlib.sha256()
# get the most recent hash
lastHash = self.lastBlockHash()

# add it to this hash
hashBase.update(lastHash.encode())

# add the miner
hashBase.update(newBlock['minedBy'].encode())

# add the messages in order
for m in newBlock['messages']:
   hashBase.update(m.encode())

# the time (different because this is a number, not a string)
hashBase.update(newBlock['time'].to_bytes(8, 'big'))

# add the nonce
hashBase.update(newBlock['nonce'].encode())

# get the pretty hexadecimal
hash = hashBase.hexdigest()

# is it difficult enough? Do I have enough zeros?
if hash[-1 * DIFFICULTY:] != '0' * DIFFICULTY:
   print("Block was not difficult enough: {}".format(hash))
```

You should validate your **full chain**, from genesis to the height of the chain on joining. Adding blocks to the chain you do not need to validate the full chain, but should validate the hash of the new block yourself.

Edge cases for joining:

- A new block was added while you were syncing. You may re-sync, or intelligently add the block to your chain

- The chain is not valid! Move to the next-highest chain.

Run a consensus every few minutes. Also run consensus when given a `CONSENSUS` message. Check all your peers: Are you at the same height and hash as the majority? Move to the longest chain - you may do this by rebuilding the entire chain. This is not optimal, but good enough for us! There are a number of optimizations possible, but not required.

Protocol

Your peer must fulfill the following protocol.

GOSSIP

Announce your peer to the network. Gossip to one well-known host. Reply any `GOSSIP` messages you receive, but do not repeat a gossip message (as known by the ID) that you've already repeated. Repeat your message to 3 peers you are tracking (make this a constant at the top of your file).

The `id` can be any string. The uuid library can be useful. Or, consider using random.

```
{
   "type": "GOSSIP",
   "host": "192.168.0.27",
   "port": 8999,
   "id": "5b29f4c7-40ac-4522-b217-e90e9587c1e5",
   "name": "Some name here!"
}
```

Reply with your info to the originator, not the sender (may be the same, but may be different if you are received a forward of the original message)

```
{
   "type": "GOSSIP_REPLY",
   "host": "192.168.0.28",
   "port": 8001,
   "name": "I have a name, too"
}
```

GET_BLOCK

Requests a single block from a peer. Peer returns the contents of that block.

```
{
   "type": "GET_BLOCK",
   "height": 0
}
```

Which replies with a `GET_BLOCK_REPLY`:

```
{   'type': 'GET_BLOCK_REPLY'
    'hash': '2483cc5c0d2fdbeeba3c942bde825270f345b2e9cd28f22d12ba347300000000',
    'height': 0,
    'messages': [   '3010 rocks',
                    'Warning:',
                    'Procrastinators',
                    'will be sent back',
                    'in time to start',
                    'early.'],
    'minedBy': 'Prof!',
    'nonce': '7965175207940',
    'timestamp': 1699293749,
   }
```

If given `GET_BLOCK` for an invalid height, return with `GET_BLOCK_REPLY` with `null`/`None` height, message, nonce, and minedBy.

Test with

```
echo '{"type":"GET_BLOCK", "height":0}' | nc -u 130.179.28.37 8999
```

Adding words to the blockchain

Test sending new words with

```
echo '{"type":"NEW_WORD", "word":"test"}' | nc -u [somehost] [someport]
```

There is no reply. You only need to implement a handler for this for the bonus, if you are doing the bonus.

Announcing new block on the chain

Add a new block to the chain. You must handle receiving these messages appropriately. Verify the hash before adding to your chain.

Sending these messages only required for bonus.

```
{
   "type": "ANNOUNCE",
   "height": 3,
   "minedBy": "Rob!",
   "nonce": "27104978",
   "messages": ["test123"],
   "hash": "75fb3c14f11295fd22a42453834bc393872a78e4df1efa3da57a140d96000000",
   "timestamp": 123456
}
```

There is no reply.

Get statistics

Get some statistics about the chain in this host.

```
{
   "type":"STATS"
}
/*
reply with the height of your chain
and the hash of the block at the maximum height
*/
{
   "type": "STATS_REPLY",
   "height": 2,
   "hash": "519507660a0dd9d947e18b863a4a54b90eb53c82dde387e1f5e9b48f3d000000"
}
```

Can test with

```
echo '{"type":"STATS"}' | nc -u [your host] [your port]
```

Do a consensus

Force the peer to do a consensus immediately. If doing a consensus, can be ignored.

```
{
   "type": "CONSENSUS"
}
```

Limitations

Ideally you would not add a block with a word that you have not seen announced. For our needs, we will ignore this, accepting any mined block. In a real blockchain this would be a SERIOUS issue.

Hand-in

Hand in all the files to make your assignment go, and a readme that is markdown formatted.

In your readme, detail:

- How to start your peer, any command line arguments, and what to expect. How long does it take to synchronize, what will we see when it is synchronized
- Tell us where your consensus code is - the part that chooses which chain to synchronize. Give us the name of the file, and the line number with a 2 sentence description of how it does it.
- 2 sentences on how you clean up peers, give us the file name and line number for this, too

Bonus: Mining

Add a miner to your code! There are a number of ways of doing this, with increasing difficulty.

1. Add a miner in your main loop +5%. Every so often (maybe a timeout amount…) do

some mining for a short period of time, and return back to listening for requests

2. Add a mining thread +10%. Keep a thread continuously running that is mining blocks. You must handle synchronization with your main thread
3. Add mining clients that communicate with TCP +20%. Create a different program or add command-line arguments to create a mining process. This process calls home to your Blockchain-synchronized process. Your “Blockchain” process forwards new words to mine on to the n workers. The workers report back if they have successfully mined a block to your Blockchain process, which in-turn notifies the network.

Be sure to add your name, or other unique identifier, to your mined blocks. Rob will take the person who has mined the most blocks (other than himself) to lunch. Or, he will just buy them lunch if they don’t want to hang out with him.

You may only get one of the mining bonuses.

In case you’re tempted to go crazy with this: If you’re mining, don’t use more than 5 machines and within those machines don’t use more than 2 cores. Max 2 GPUs in the lab. If the server room gets too hot, the competition would likely be shut down. So, play nicely!

Your peers should be *well behaved* and accept blocks from other peers.

Note that the permitted imports are still in play - if you want to write GPU code, you have to do it *from scratch*.

Networks

- Test network: amber.cs.umanitoba.ca(192.168.101.249)/umber.cs.umanitoba.ca(192.168.101.248) - difficulty 8 - quieter with a shorter chain. Will be reset every few days to keep the chain small.
- Main network: aviary lab - difficulty 8 - you must mine a block here for the bonus.
- Competition network: omber.cs.umanitoba.ca(192.168.102.145)/ember.cs.umanitoba.ca(192.168.102.146) - difficulty 9 - top miner in this chain gets lunch.

Some notes

Test and challenge networks are not on the “NFS”. That is to say: your files will not automagically be there. You will have to copy them to each machine you want to run on.

You can’t look up your IP address in a nice way on the test or challenge network. You must use a 192.168.x.y address otherwise you will get an error.

Usually looking up your IP looks something like this:

```
>>> hostname = socket.gethostname()
>>> IPAddr = socket.gethostbyname(hostname)
>>> print (IPAddr)
130.179.28.138
```

But if you use a 130.x.y.z address, you’d do something an error saying “not permitted”

```
>>> import socket
>>> peerSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
>>> peerSocket.bind(('', 9000))
>>> peerSocket.sendto(b'test', ('130.179.28.37', 8999))
```

Reflect in ePortfolio Download Print

Open with docReader

Activity Details

✓ You have viewed this topic

Last Visited Nov 26, 2024 2:51 PM