Distributed Bitcoin Pool miner

Ralph J Donato Jr

Graduate Project

StudentID:

Email: [rdonato@albany.edu](mailto:rdonato@albany.edu)

# Abstract

This is a project therefore abstract statement is more of a statement of what the student intends to learn and gain experience with by doing this project. Bitcoin mining is the primary goal. What has been created is a distributed bitcoin pool miner. Bitcoin is very relevant to the current world we are living in. Small nations are switching their currency to it. It is no secret that nations are using crypto currency to evade sanctions.

Along with the study of bitcoin is the study of distributed programming. This app is a distributed app. It receives a job from a mining pool then it will distribute it amongst workers(agents) then when a worker finishes a task it will return the job to the Pool.

Some of the technologies that are used are sockets. Existing distributed libraries have not been used. The student wanted the experience of working with and networking actual hardware.

Parallel programing has been explored also. Each a separate app is multi-threaded. Android Programming has been explored also.

To summarize the student wanted to gain more knowledge of the following areas blockchain, Bitcoin, Encryption, Distributed programming and network communications by sockets.

# Bitcoin

## Distributed Ledger

Distributed ledger is the foundation of the foundation of Bitcoin. Block chain is a distributed Ledger and Bitcoin is a block chain used for a crypto currency. Distributed ledger in its simplest terms is a database that is contained on all peers.

It runs on a peer-to-peer network where all peers will have a copy of the database. This prevents hacking because for a hacker to be successful they would have to attack all peers.

## Block Chain

Block chain is a distributed ledger. The difference is the database can only be added to. Existing records can’t be modified. Records are stored in blocks. It is a collection of transactions stored in blocks. This database can only be added to by blocks with transactions.

## Bitcoin

### Merkle Root

Merkle trees are a required data structure in blockchain. Merkle trees are hash trees. Hash trees were patented by Ralph Merkle in 1979.

Merkle trees are formed by putting the data in pairs. This data will typical be a transaction and will be referred as such for the remainder of this paper. First each transaction is hashed. Then the Each pair is hashed. Then those hashes get hashed. Hashing continues until single output remains. The final single hash is the Merkle root. This form of Merkle tree is a binary tree since it takes two inputs and produces one output.

The diagram Merkle Tree shows an example. In this example the bottom items G, H, I and J are the data. The row above C, D, E and F are all the hashes of the row below. A is the hash of c and D. B is the hash of E and F. For example, A is a hash. To get A the C and D would be concatenated. Then hashed.

The Merkle proof is used to verify that a transaction is included in the tree. Merkle proofs are important because they allow it to be known if a transaction is in the tree without having to replicate the whole tree. This saves bandwidth. In the case of the sample tree to determine if transaction H was in the tree the Merkle root would be needed. The hash of G which is C would be needed. The hash of E and F which is B would also be needed. These could be used to determine if H is in the tree.   
Chart, bubble chart

Description automatically generated

This project will have to generate a Merkle root.

### Mining

Bitcoin mining is the process of handling transactions. Mining is where new bitcoins are created. Mining is the process of processing transactions into blocks. This is also the step that creates new bitcoins. Miners are rewarded for processing transactions and adding a block to the Bitcoin Blockchain. In doing this mining also ensures that all members of the network will have a consistent view of the data. Since bitcoin doesn’t have a database that keeps track of who has how many bitcoins. This is determined by the blockchain having a log of all transactions in the network. Mining takes transactions and puts them into blocks.

When new transactions are in the network a mining node can start the process of mining. Mining is a very difficult process. It involves solving an extremely complicated mathematical function. This is intentional and build into the specification.

Mining is how new bitcoins are created also. It is the reward for mining. Unfortunately, as the network grows the reward gets smaller. The minimum requirements to run a mining node are small. Examples can be found that run on Raspberry PI. However, these are impractical and will never receive a reward.

Solving the mathematical puzzle is very difficult. It requires using cryptography with double SHA-256. To win the miner must be the first to come up with a hash that is less than or equal to the target hash. All the header values are concatenated. These are hashed until the hash is less than current target value. Every hash causes an increment of the nonce. The nonce has a maximum. It is a 32-bit number so at 4 billion if a suitable hash hasn’t been found the process has to restart. A small modification to the time stamp will happen then the process will start with 0 for a nonce.

A very simplified algorithm for bit mining would look like the following

While nonce< Nonce Max

hs =hash (header + nonce)

if( hash<current target

Successful has done

nonce = nonce +1

If a suitable hash isn’t found timestamp is modified slightly and process starts all over until acceptable hash is found.

This project will not be obtaining a copy of the complete blockchain. It will be connecting to a mining pool.

#### Mining a pool

Introduction

The above gives a very brief description of how to mine. This project will be joining a mining pool. It is important to define the header and the target. This project will use the Stratum Protocol to connect to a POOL.

The controller application will connect to the Pool using sockets. The data is transferred in plain text. Communication serialized as json and communicated between pool and Controller.

Why use a pool

Mining pools allow multiple miners to join and share their resources and increase the hash rates.

Stratum Protocol

This project will use the Stratum V1 Protocol to connect to a mining pool. This isn’t the only protocol available however it is very suitable for this project.

Stratum has Client Methods and server Methods. All methods are communicated as Json sent over sockets.

The client methods are

mining.subscribe

This subscribes to the mining pool.

mining.authorize  
This authenticates with the Pool   
  
mining.submit

This submits the job. It is called if the miner solves the puzzle.

There are also methods sent to the client from the pool

mining.notify

Notify sends the job data to the client

mining.set\_difficulty

This sends the difficulty. It can be changed every 10 minutes. Difficulty when mining a pool is always less than when mining as a full node on the Bitcoin network.

Header

The block header is obtained by the following elements in from the mining.notify command of the pool. Mining notify returns 10 Items.

An example is as follows.

{“id”: null, “method”: “mining.notify”, “params”: [“bf”, “4d16b6f85af6e2198f44ae2a6de67f78487ae5611b77c6c0440b921e00000000”,

“01000000010000000000000000000000000000000000000000000000000000000000000000ffffffff20020862062f503253482f04b8864e5008”,

“072f736c7573682f000000000100f2052a010000001976a914d23fcdf86f7e756a64a7a9688ef9903327048ed988ac00000000”, [],

“00000002”, “1c2ac4af”, “504e86b9”, false]}

To build the header the following are needed, and a string is concatenated as follows

Params 5 which is the version

Params2 which is the previous hash

The Merkle root which. Calculation will be shown in the next section

Params 6 which is the network time

Job difficulty is returned in the mining.set\_difficulty response from the Pool

The above is concatenated in order to get the header. The header is a byte array as a string.

Merkle Root

When mining as part of the pool the Merkle Root is calculated as follows. It gets the extra nonce 1 from the return of mining.subscribe ack.

It uses params two and three from mining.notify these are the prefix Coinbase and the suffix Coinbase.

Coinbase is built as the concatenation of prefix Coinbase, ExtraNonceOne, Extranoce2, and the suffix Coinbase. These are all converted to hex strings and concatenated. This gets converted to a byte array. Then SHA 256 Hashed twice.

The Merkle root is calculated by hashing each of the Merkle Roots from mining notify with the Coinbase.

Target Hash

Mining in the pool still requires hashing with a nonce to get a hash that is less than a target. The target is build based on the response from the mining.set\_difficulty method. The value from set difficulty is used to get a target. The default difficulty is 1 which produces a target of

0x00000000ffff0000000000000000000000000000000000000000000000000000.

# Technology

## Multi threading

The application is written primarily using the C# programming language. It makes extensive use of multithreading. The main technology it uses is the Thread class. This class closely resembles the C Pthread. One object that gets made use of is the thread safe singleton. A singleton that allows access by only one thread at a time.

Threads are created in C# by passing a ThreadStart object to a Thread class. ThreadStart is a class that takes a delegate to a function that returns void as its parameter. Delegates and C# can best be described as void \* in C. Delegates are used in the callback pattern. Basically the way to pass a function to another object.

The easiest way to protect critical sections of code in C# is to use a lock. A lock locks on an object.

For example

Class TestThread

{

Object obj = new Object();

Lock(obj)

{

// put critical section here

}

}

## Asynchronous programming

In C# asynchronous programing is like multi-threading. Asynchronous focuses more on having the program do something else while long running io processes are waiting to return.

This application takes advantages of Asynchronous Threads using BeginAcceptTcpClient and BeginRead

## Sockets

The discussion of sockets and asynchronous programming can really be joined here. C# provides wrappers for sockets that also incorporate asynchronous programing.

I used Begin read to asynchronously read from a stream. BeginAcceptTcpClient was used for when a server socket is listening for a connection and a new client tries to connect. This allows a new client to connect without blocking.

## Systems Programming

Processor Usage

Bitcoin mining is very processor intensive. It is important to gauge programmatically how hard the processers are working. The technology stack for this is .net core. The algorithm to determine the processor usage is to calculate the the processor usage over a time interval then divide this by the actual time that elapsed. This is done by converting

## Android

# Project Documentation

## Technology Stack

### How to run

This software is built mostly using DOTNET core.

DOTNET core is an open-source implementation of Microsoft .net. It can be run on Windows, Mac or Linux. This has not been tested on any apple products.

There are two separate Android Applications a miner agent and a monitor

DOTNET CORE

Windows – Use the following link to install .net core. 5.0 is the recommended version

[Install .NET on Windows - .NET | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/core/install/windows?tabs=net60)

Linux - [Install .NET on Linux distributions - .NET | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/core/install/linux)

IDE - Visual Studio is free for community edition. This only works on the windows operating system.

Visual Studio code works on both Windows and Linux. I would recommend this. It is a very good lightweight IDE.

This link has directions for each of the operating systems. [Download Visual Studio Code - Mac, Linux, Windows](https://code.visualstudio.com/Download?msclkid=d1579481cf2511ec96d66b2cccbf3d10)

Android

I used Android Studio for the android components.

## Project Components

Graphical user interface

Description automatically generated with medium confidence

Bitcoin mining Pool(POOL) – This is a Bitcoin pool. For this project I am using Slushpool. Any pool that uses the Stratum v1 protocol will work.

Controller is the main part of the system. In the simplest definition this splits up a job it receives from POOL. It distributes these jobs to the agents. Then if the agent completes a job before another miner completes it will submit it to the pool.

Agents do one thing. They receive a piece of the job and try to solve the proof of work. If they complete it before either another agent in the network or another miner in the pool complete it. There is an android agent and a .net Core agent. .net core agent is multithreaded. It actually breaks the piece of the job it received into threads.

The monitors are the last component. There is an android app that provides status about the controller.

### There is a also a windows tray application for monitoring the application. This only runs on

## Classes and Files

Source code is located at <https://github.com/Ralphd42/GRADPROJECT/>

### Controller

Controller is the largest piece of code in the app.

It has 16 classes that interact with each other. It is multi-threaded. It is a console app. The user interface is the console. Although there are plans to have a web UI that won’t take place for the MAY 9th Release.

The following is a description of the classes in the Controller

ExcelLogger

This is a class with one purpose. It logs data to an excel file. It is used to log json to and from the POOL. The output file is stored in the params file as JsonLog. I use this class as a thread safe singleton and by dependency injection. The reason for keeping the logging in one place is to easily change the way things are logged without having to change them in multiple locations.

The main function is LogJson. This function takes the json as string and logs it to the file

Logger

Logger is a simple logger to a text file. Data is logged to a file. The file name is stored in the params.json file as LogFile.

LogError(Exception exp, string message) Logs an error with a custom message

LogMessage(string Message) Logs a message to log file

LogWIthDate Logs a message and automatically adds a Date.

WorkerManager

Worker manager manages the worker(agents) in the system. The function AddWorkerThread is run as a thread in the program class it listens on a tcp socket for workers to either join or leave the network.  
this class has a private object List<Worker> which contains a list of all workers in the system. This class is exposed to other areas of the system when needed it.

Worker

Worker is a representation of a worker(agent). It contains the relevant information about the agent. This consists of number of threads the agent can run. These can be set to a different amount. It has a name, IP Address and a status.

public bool addWorker(Worker w) this adds a worker to the collection of workers

public void ShutdownWorkers() This send the shoutdown notification <T># to the agents

AddWorkerThread() this is run as a thread to listen for workers trying to join or leave the network.

ResponseLoop

This class listens for the agents to return with a completed job. It then calls the Poolconnector function to send the job to the POOL. It also notifies all other agents that the job is over, and they can stop processing this job and prepare for the next.

RunLoop() run as a thread. It listens for agents to report that they found the nonce and solved the hashing puzzle.

ProcResponse() this takes the response and sends it to PoolConnector to be sent back to the pool.

Program

Program is the entry point it contains the main function. It handles the User interface. It has functions to get the configuration from the params.json file.

PoolManager

Pool manager is probably unnecessary. It essentially wraps the PoolConnector class. In the future I will investigate ways to remove this class.

PoolConnector

This class handles all communication between POOL. This has functions handling all operations involving the POOL. It has functions to handle sending all the client methods  
mining.subscribe, mining.authorize and mining.submit.   
It handles the pool methods mining.notify, mining.set\_difficulty.

It handles the acks for the for the client methods. It processes acknowledgements for mining.subscribe, mining.authorize and mining.submit.

Monitor

Monitor allows an application to connect to Controller and get status data. It also can kill the controller. There is not a way to restore the Controller once it has been shut down.

JobQueueWatcher

This class maintains the job queue. It has a function WatchQueue that is run in a thread. This thread is responsible for taking jobs from the job queue and sending the job to the agents.

JobQueue  
 This is the Job Queue PoolConnector puts jobs here and JobQueueWatcher removes them. This class wraps a ConcurrentQueue object. JobQueue contains a reference to the Jobdata that is currently being processed. Jobs are stored as MineJob objects.

JobManager

Job Manager handles taking a job which is stored in a MineJob object and splitting the job and sending to the agents. This class handles breaking apart the job into MineThreadData and creating JobThreads to Process this data.

JobThread

This class takes the MinethreadData from job manager and sends it to a client. This class does not handle the return from the client, it only sends the request to the client.

Ackman

Ackman handles acknowledgements from client methods to the POOL. When the client uses a method to call POOL it sends an ID. This class stores that ID and the method name. When acks come back from the pool this how it is determined which method call is being acknowledged.

MineJob

This is the data needed for a job. It is the complete data. It is split into MinethreadData data by the JobManager class. This also calculates the blockheader. Which is the data sent to the agents to be hashed to find a nonce. This is calculated as follows the following items are concatenated.

It takes the bitcoin version from the MiningNotify method

Prevhash from MiningNotify the Merkle root, the nettime also from mining.notify and the job difficulty from the mining.notify Pool method. Please note this is not the difficulty used to generate the target.

MinethreadData

This is the data broken up to be delivered to the agents.

MineTools.CryptoHelpers

This class contains cryptography functions. This is the basis of mining bitcoin each will be discussed.

**CoinBaseHash -** This generates the coinbase. Coinbase is the first leaf in the Merkle tree and is a critical component to getting the Merkle root. Coin base is the concatenation in order of the prefix coinbase operation. This comes from the mining.notify Pool method. It then concatenates the ExtraNonce which is returned in the mining.submit acknowledgement. It next takes the extranonce 2 which is generated in poolconnector class. Finally the suffix coinbase operation which comes from the mining.notify message is appended. These are appended as HEX strings. It is converted to a byte[] then SHA256 hashed twice. This is the coinbase. Which will be used as the Merkle Root.

**MerkleRoot** – Merkle root is obtained by concatenating the coinbase and each element of the Merkle array. They are concatenated as hex strings. These are then converted to byte[] hashed twice and the process continues until the whole Merkle array is processed.

Merkle root returns a hex string representation of the MerkleRoot

### Agent(worker)

The second component is the agent the agents have one role. That is to hash the header until it is less then target. Then it reports the nonce used to achieve this.

The following classes are used in the agent.

Program

Main entry point to application contains static variables for accessing configuration handles any user interface information.

MineOperations

The MineOperations class has one purpose. It connects the agent to the controller. It sends a message to the controller with a request to join. All communication is done by sockets. It sends a message with the minimum possible information needed. <A>threadcount:agentName# is the format of the message. The controller determines the IP address when it receives the message. On the controller side WorkerManager handles receiving the request to join the network and creates a Worker object in controller.

JobListner

This class oversees listening to the server for job information. It has one function ListenToController that listens on a socket for commands from the Controller. There are three commands. A job. A job will be sent as an object serialized as json.

The other commands are <K># and <T># K is for kill this means to kill all threads currently running. This means that one of two things happened. The first is a that another Miner in the Pool already solved the HASH, or another agent in the network found the correct nonce. In simplest terms it means this agent lost. Stop all threads and wait for a new command.

<T># means the controller has shut down. It will terminate the agent.

MineBatch

Mine batch is the class that handles when a job is received in JobListner. Mine batch has the role of splitting the job into pieces based on the number of threads it is running. It spawns these threads. It handles the event of a nonce being found by one of the threads and sending it back to the Controller. If successful it sends a message <F>AgentName:NewNonce#". The controller will then try to submit the job to the POOL.

Miner

Miner does the actual hashing and comparing of data. Its goal is to generate the nonce for needed to solve the Proof of Work. The Nonce is generated here.

### Agent(worker)Android

The android agent is a single threaded agent. It only runs one thread of the hashing algorithm. It is not very practical. It was added as a way fore the developer(student) to gain experience in an area new to the developer. The developer gained an understanding of threading and sockets communication with android applications. Even though the job itself only uses one thread. Threads still had to be created in the application this is because Android doesn’t allow sockets to run in the main thread

Android Studio was used to develop this application. Unlike the pc miner it does have a User Interface.

MainActivity  
The main activity of the application. It shows the ip address of the android device. It is a useful feature even if not used for Bitcoin mining

IpCommI

This class handles getting the IP address that main activity displays. It must be run in s separate thread since all socket related functionality can’t be run on main thread  
DisplayMessageActivity

This class displays the screen that will show when the miner is working. It calls the JoinNetwork class. It also uses android notifications.

JoinNetwork

This connects the agent to the controller. Similar functionality as the desktop version.  
MineBatch  
 Mine batch does the work. Only a single thread. This could probably be enhanced to approximately four threads. It also sends a response back to the controller if successful.

### Monitor

Monitor is also an android app. Monitor pulls data via sockets from the controller monitor class. It has status information about the system. It also allows the system to be shut down. Unfortunately, it doesn’t have a way to turn the system back on. That will need to be done from the machine running controller. It will shut the whole system including agents though.

### Tray Monitor

Tray monitor a user interface windows executable. It creates a tray application for Windows 10 and Windows 11. This is not a multiplatform app.

It allows users to check the status of the Controller. They can shut the controller down if necessary.

The screen appears as follows below.

Graphical user interface, text, application

Description automatically generated

## Testing

CONFIGURATION

Source code is located at <https://github.com/Ralphd42/GRADPROJECT/>

Please email if there are any issues getting source from this destination. I can send a zip but there is a large amount of source code.

the controller has a configuration file. Params.json. This contains settings. The settings it currently has are

"debug"            :"false"   ,

      "LOGFILE"          :"bc.log" ,

      "WorkerManagerPort":"13001"  ,

      "JobPort"          :"13002"  ,

      "JobRPort"         :"13003"  ,

      "MonitorPort"      :"13004"  ,

      "MinWorkerCount"   :"1"      ,

      "JsonLog"          :"bc.xlsx"

Debug is a way to force data into the controller instead of getting the data from an actual pool.

Next are a few port numbers. These can be set to any available ports. I don’t really have an explanation for why I used the ones I did. Workermanager port handles management of workers. Job port handles sending jobs to the agents. JobRPort handles responses from the job. This is thew port an agent report that it won. Monitor port is the port used by the monitor. LOGFILE is the location of the logfile. JsonLog is the location of the excel file that will contain a history of all JSON communicated between the Pool and the Controller. MinWorkerCount is the minimum number of workers that must be available to start a job.

The Agent also has a configuration file it will be the same name as the Controllers config.

"debug": "true",

    "LOGFILE": "agent.log",

    "WorkerManagerPort": "13001",

    "JobPort": "13002",

    "ControllerIP": "192.168.1.12",

    "ThreadCount": "10",

    "AgentName": "AGENT.ONE",

    "JobRPort" : "13003"

ControllerIP is the Ip address of Controller. ThreadCount is the number of threads that this agent will run. AgentName is the name of the agent.

**The important thing about ports is that they may need to be opened. I tested this on Android, Linux , windows10 and Windows 11. I needed to create rules in my windows firewall for Windows10 and Window11. Ubuntu 20.14 and Android I did not.**

**Once .net core is installed it is very easy to run the .net core apps. Simply go to the directory with the .csproj file and type dotnet run.**

**Unfortunately, the android components are a little tougher to run. I did not put these on the play store. You must install them by using android studio and a usb cable.**

**I tested this application in two ways. One is to just run it and then compare the log files from the agents and the Controller. I check the dates of the actions to make sure they match properly. I make sure Sockets aren’t dropping connections.**

**Also check for exceptions. All exceptions are handled. However, there shouldn’t be exceptions that aren’t handled.**

**In my Program.cs in the controller I write functionality to test in disconnected from the Pool way.**

**This allowed me to check that if the Bitcoin network was slow, I would be able successfully mine.**

# Closing statement

As a developer I gained a knowledge of Android software development. I learned to work around how to handle threading and the user interface. Sockets and the main thread problems.

I got to develop a distributed application that communicated with multiple systems.

# References

1. <https://corporatefinanceinstitute.com/resources/knowledge/other/distributed-ledgers/>
2. <https://101blockchains.com/distributed-ledger-technology/>
3. <https://101blockchains.com/blockchain-vs-distributed-ledger-technology>
4. <https://btc.com/stats/diff>
5. <https://en.bitcoin.it/wiki/Stratum_mining_protocol>
6. https://reference.cash/mining/stratum-protocol