Joshua Dahl

The GitHub repository for this project can be found at:
https://github.com/joshuadahlunr/CS491-Blockchain-Final-Project

Abstract

This project presents a prototype for a new Blockchain-like application based on IOTA's Tangle technology. It aims to provide many of the advantages of Blockchain's distributed ledger but with real-time performance. This aim is accomplished via a novel pruning algorithm, Latest Common Genesis Pruning, which reduces the size of the Tangle so that it can be stored entirely in memory, without the need to write certain parts to disk.

Introduction

Blockchains provide a structured method for distributing large amounts of information through a network of connected peers. Similarly, many Multiplayer Video Games demand the synchronization of huge amounts of data among thousands of peers in real-time. Thus Blockchain technology seems like it would provide a good platform for video games with its decentralized information distribution abilities. However, most current Blockchain applications have a strong focus on security and auditability; both concepts which many Video Game applications care little or nothing about.

Alternatively, the IOTA Tangle provides an alternative to Blockchain technology, following many of the same paradigms but replacing a list of transacted blocks with a graph of transactions. It doesn't rely on dedicated miners, instead, every peer seeking to add transactions to the Tangle must validate a selection of previously unvalidated transactions,called tips, in the Tangle. This allows for immense speed increases over a traditional blockchain, but there are still limitations on its performance. Thus this project presents a foundation for a real-time Blockchain-like application.

Novelty

This project's main contributions over IOTA and Blockchain at large are twofold; first, we implement a policy for pruning the Tangle and second, we keep the Tangle entirely in memory instead of saving it to disk.

Before explaining our pruning policy, a few details of IOTA's implementation must be understood. First, whenever a new transaction is added to the Tangle, it becomes a tip. Additionally, several of the transactions which were previously tips, that this new transaction validates, are no longer considered tips. However, different nodes on the network will see slightly different versions of the Tangle due to network latency, and several transactions may be

added at once which validate the same tips. Thus over time, the set of transactions that are considered tips fluctuates, sometimes expanding under high network low and sometimes shrinking under lower loads.

Additionally, every node is connected to the current set of tips through the network. As more tips get added and more connections are formed, nodes deeper in the Tangle become indirectly connected to more and more tips. At some point, a node will eventually become connected to every tip. The amount of tips a node can see defines the network's confidence in that node;how much the network believes in the validity of a node.

With those two concepts understood, we can discuss the pruning policy: Latest Common Genesis Pruning (LCGP). LCGP relies on the discovery of a set of transactions called the Latest Common Genesis. Whenever a new transaction is added to the Tangle, we check if the number of tips in the current set of tips is less than some threshold (three in this implementation) and if so, add them to a list of conversion candidates. When we seek to prune the Tangle, we look through that conversion candidate list and find the set latest in time where every transaction has 100% confidence. We then determine the state of the network as seen by those tips, adding up the balance of every account seen before them and combining that information into a single node, which we call the Latest Common Genesis. That node is then given some special "aliases" so that when later nodes in the network look for any of the tips which were merged, they find the Latest Common Genesis. Then any nodes before the Latest Common Genesis can be removed from the Tangle without loss of information.

Due to LCGP the size of the Tangle can be kept rather small, if the Tangle starts approaching a certain size, it can be pruned to reduce its size again. This property allows for an LCGP pruned Tangle to be stored entirely in memory instead of being written to disk (although nothing is stopping certain "auditing" peers from continuing to write the Tangle to disk for auditing at a later time); this provides a significant performance improvement. This same property also allows for some new kinds of security checks that would be infeasible in a regular IOTA Tangle. For instance, the current implementation validates every node added to the Tangle regardless of source, which can be done quickly on the relatively small Tangles produced by LCGP. This project's implementation can routinely perform this validation in less than two milliseconds [often around one millisecond]. Being able to feasibly check every transaction means that honest nodes won't even add nodes deemed malicious to their Tangles, making many types of attack substantially more difficult.

This implementation is still a prototype and thus many things could be improved. Notably, LCGP pruning must be manually performed, a system for automatic pruning once certain conditions are met would be a very good addition. Additionally, for this type of distributed ledger to be useful in Video Games, the current financially focused transactions must be replaced with arbitrary transactions capable of manipulating an arbitrary state. There are interesting ramifications that change would have on the pruning algorithm.

User Manual

Arguments and Operation

The command to run the program is:

```
./Tangle [IP to connect to]
```

For the most basic example:

1) run:

```
./Tangle
```

In one terminal to create a new network (you will need to press enter once the application starts to generate an account).

2) Followed by:

```
./Tangle 127.0.0.1 # You may replace 127.0.0.1 with a remote IP address if needed
```

In a second terminal to connect to the network (you will need to press enter once the application starts to generate an account).

2) Type 'd' to print out a visual representation and note the topology of the Tangle (you will need to press enter to skip transaction display).
3) Once both peers have booted up and connected, press 'b' to check the account key for each peer.
4) Then on one peer type 't', paste in one of the account keys which were just noted, an amount, and a mining difficulty (difficulties higher than 3 take a long time).
5) Once the transaction has been received by the other peer, type 'g' to prune the Tangle.
6) Type 'd' again and note the differences in the Tangle's topology.

Arguments

- If an IP address is NOT provided, it will create a new network.
- If an IP address IS provided, it will attempt to connect to an existing network.

Operation

You will need to select a key file at the beginning of the program, alternatively if you leave the prompt blank it will generate you a new set of keys. It will take a moment to connect, once done you will be given the option to enter several commands:

- (B)alance - Query our current balance (also displays our address)
- (C)lear - Clear the screen
- (D)ebug - Display a debug output of the Tangle and (optionally) a transaction in the Tangle
- (H)elp - Show a help message (similar to this one)
- (G)enerate - Generates the Latest Common Genesis and prunes the Tangle
- (K)ey management - Options to manage your keys
- (P)inging toggle - Toggle whether received transactions should be immediately forwarded elsewhere (simulates a more vibrant network)
- (S)ave <file> - Save the Tangle to a file
- (L)oad <file> - Loads a Tangle from a file
- (T)ransaction - Create a new transaction
- (W)eights - Manually start propagating weights through the Tangle
- (Q)uit - Quits the program

Developer Manual

Project Layout

Transaction.h/cpp contains an implementation of a transaction. Tangle.h/cpp contains both a node in the Tangle, and a manager for a Tangle. Networking.hpp contains code for an automatic connection handshake and a messaging extension. These three files build on each other, adding additional functionality to the previous file's classes. Main.cpp contains a driver for the Tangle, it performs some initialization and starts the menu loop. Utility.hpp contains some helper functions used by the rest of the program. Monitor.hpp provides a simple thread safe wrapper used by the Tangle implementations to increase the thread safety of vectors. Keys.hpp provides a cryptography wrapper, containing everything for ECC signatures.

Dependency Instructions

The project depends on a local installation of Boost. The remaining dependencies are included as git submodules and can be acquired by running:

```
git submodule init
git submodule update
```

However, Breep, our networking library, requires an additional patch. See the section below for specific instructions.

Boost

Boost must be locally installed as a system library. Instructions on how to do this can be found at Windows or *UNIX.

Breep

Breep will be downloaded as a submodule automatically. If git is not being used, the source code can be downloaded from https://github.com/Organic-Code/Breep. In this case you will need to move the cryptopp file downloaded into the thirdparty directory.

We have applied an additional patch over breep. Once you have a Breep folder in thirdparty, the following commands will apply the patch.

```
cd thirdparty # If not already there
cp Breep.patch Breep/Breep.patch
cd Breep
git am Breep.patch
```

Crypto++

Crypto++ will be downloaded as a submodule automatically. If git is not being used, the source code can be downloaded from https://github.com/weidai11/cryptopp.git. In this case you will need to move the cryptopp file downloaded into the thirdparty directory.

Building Instructions

A compiler capable of compiling c++20 code is required to compile this code. Any compiler shipped with a modern distribution of Linux should be sufficient. Simply running make should compile the project:

```
make # Must be run in the root directory of the project
```