Alternate *ACM* SIG Proceedings Paper in LaTeX Format*

[Slightly extended for DADS] †

No author info because of double-blind review.

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings. It is an *alternate* style which produces a *tighter-looking* paper and was designed in response to concerns expressed, by authors, over page-budgets. It complements the document *Author's (Alternate) Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX*. This source file has been written with the intention of being compiled under LaTeX2$_\epsilon$ and BibTeX.

The developers have tried to include every imaginable sort of "bells and whistles", such as a subtitle, footnotes on title, subtitle and authors, as well as in the text, and every optional component (e.g. Acknowledgments, Additional Authors, Appendices), not to mention examples of equations, theorems, tables and figures.

To make best use of this sample document, run it through LaTeX and BibTeX, and compare this source code with the printed output produced by the dvi file. A compiled PDF version is available on the web page to help you with the 'look and feel'.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Delphi theory

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

†A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX* at `www.acm.org/eaddress.htm`

SAC'19 April 8–12, 2019, Limassol, Cyprus

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

All cryptocurrencies, in particular, Bitcoin (the worlds most used cryptocurrency) maintain a decentralized record that keeps track of all transactions that have happened in a chronological order [1]. The ability to maintain a decentralized record that can be updated by almost anyone has attracted the attention of a lot of other fields beyond the cryptocurrency market where this concept was introduced. For instance, a distributed record could be used to keep a record of transactions, contracts and all other kinds of information that, if we wanted to save them, we would have to speak with a notary. [1]

This distributed record is usually maintained in the following manner. Firstly, for efficiency reasons, multiple transactions are grouped together, this groups of transactions are called a*blocks*. Afterwards, blocks are linked between them to form a linked list which is called *blockchain*. This linked list enforces a chronological order over the blocks and as a consequence over the transactions inside the blocks. An interesting aspect of cryptocurrencies, like Bitcoin, is that they use a decentralized open peer-to-peer membership system. This means that nodes do not have to know all the other nodes of the system, and any node can join or leave the network at any given time that the protocol assures consistency of the record even if a few nodes might have a rational or byzantine behaviour.

Usually, the system that maintains the blockchain works in the following manner. The nodes of the system, concurrently, receive transactions, validate them and relay them to other nodes. Furthermore, each node attempts to generate the next block in the chain, to do this they try to solve a challenging cryptographic puzzle called proof of work. When a node finally generates a block, it will broadcast it through the network which will make all other nodes cancel the generation of concurrent blocks to the one being broadcast and start trying to generate a new block. Before accepting and relaying a block, each node validates the blocks it receives.

By the brief description above, it easy to realise that the relay of transactions is a core process for any distributed ledger. Firstly, the transactions need to reach the nodes that are generating blocks so that they can be added to a block. Secondly, they also need to reach other nodes allowing them

[1]For a list of examples, refer to https://blockgeeks.com/guides/blockchain-applications/

to validate blocks. In Bitcoin, the broadcast of transactions works by nodes periodically advertising to their neighbours the transactions that they have. Their neighbours upon, receiving the advertisements for transactions that they still do not have, will send requests for those transactions to the nodes who sent the advertise. This process generates a redundancy of advertising messages. However, despite this redundancy begin desirable allowing the system to cope with failures, we found out that each node receives an excessive amount of duplicated advertisements for each transaction.

In this article, we propose a modification in the transaction broadcasting system of *Bitcoin* that improves its efficiency. Our adjustments take advantage of already existing asymmetries in the network. In fact, in a *Bitcoin* network only a fraction of the network, about 10%, spends resources generating new blocks without being associated with a mining pool (these nodes are called miners), the majority of nodes just relay information and maintain a copy of the blockchain. Our strategy consists in skewing the broadcasting of transactions so that they reach miners faster and at the same time lowering the amount of duplicated advertisements. This strategy takes advantage of, not only the weak latency requirements around the broadcasting transactions in Bitcoin between non-miners nodes, but also takes advantage of messages types introduced in the previous year, that allow for a much more efficient dissemination of transactions once they are added to a block. An experimental evaluation of the changes proposed show a reduction in 10.7% of the bandwidth consumed and a reduction of 40.5% in the total number of messages exchanged, without any negative impact in the resilience of the system.

2. THE *BITCOIN* LEDGER

Bitcoin was created back in 2009 with the main objective of providing a system where two entities can exchange goods in a secure and anonymous way without having to trust each other or any third entity. To achieve this goal the system uses a crypto coin, that can be exchanged between the parties involved in a transaction. All the transactions are grouped in blocks and registered in the distributed ledger. Furthermore, the ledger keeps track of all the coins that have been spent, which forbids users from trying to spend the same coin twice (attacks that try to defeat this system are entitled as *double spending* attacks). The Bitcoin ledger is built by linking each block to its predecessor hence forming an infinite chain of block named *blockchain*.

The protocol to maintain the Bitcoin ledger is quite complex with many moving parts and functionalities that complement each other. Firstly, Bitcoin has membership algorithms that try to ensure that each node maintains connections to other nodes of the systems chosen randomly. This connections between the nodes form an overlay that is then used to disseminate information, namely: the transactions created by clients and the blocks generated.

As referred previously, new blocks are generated concurrently by nodes named miners. Each miner picks a group of transactions to form a block. For a block to be considered valid, not only has to contain only valid transactions inside but also, has to contain a proof that the node solved the cryptographic puzzle. This cryptographic puzzle is influenced mainly by the content inside the block like the hash of the previous block and the transactions inside the new block. This has two advantages. First, it disincentivizes the creation of blocks with invalid transactions inside it, because the only way a node get recompensated by creating a block is if his block gets accepted into the blockchain. Furthermore, the difficulty of the puzzle lowers the probability of two nodes generating a block at the same time, which if it happens would generate a fork in the chain. Note that as transactions take different times to reach different nodes, it is certain that the group of transactions chosen by two nodes to add to their blocks will be different. Hence, this will result in two different crypto puzzles for each node to solve. When a new block is generated it is broadcast through the network. To avoid corrupted or invalid blocks from being broadcast, each node has to validate a block before broadcasting it. For a block to be considered valid it has to has the following information: hash of the previous block and all the transactions inside it (to be able to validate the transactions). A transaction is valid when it does not try to spend an already spent coin. If a node does not have all these pieces of information it has to wait before relaying the block.

If a node receives a block at the same height as the one he is trying to mine, the process of mining is interrupted. This behaviour also lowers the probability of two different blocks at the same height begin generated at the same time. However, in the case, they are and are broadcast through the network a fork will happen in the chain and to solve this fork the network will then adopt the branch which is higher. This is done by nodes aborting the mining process when they realise they are working on a smaller branch and start trying to mine on the higher branch.

The algorithms used by Bitcoin to disseminate transactions and blocks have evolved over the years and recently introduced mechanisms to prevent wasting bandwidth. We are now going to go over a simple overview of the protocol. As referred previously, transactions are broadcast through advertisements sent in *Inv* messages that contain the set of transactions known by the sender. When a node receives a *Inv* message it determines which transactions it does not possess and sends a *GetData* message requesting those transactions. Finally, when a node receives a *GetData* it will reply with a *TX* message for each transaction requested. Regarding blocks, blocks are disseminated mainly in two ways. The first one and older is through advertisements similar to transactions. Once a block is found is sent a *Headers* message advertising the block. When a node receives a *Headers* message containing the header of a new block it will request it using a *GetData* message and will then receive a block through a *Block* message. The second strategy for broadcasting blocks consists in sending a summary o the block through the *CmpctBlock* (compact block) message. When validating a block after having received it by a *CmpctBlock* message if the node does not possess all the transactions to validate the block it can send a *GetBlockTX* requesting them. The first strategy has as an advantage the fact that the nodes do not need to exchange more message between them but as a disadvantage, it generates more redundancy since all the transactions are sent with the block and there is a high probability that the node already has all transactions needed to validate the block.

Additionally, each node maintains or each neighbour a queue that containing messages to be sent in the future. When a new *TX* is received, after being validated is added to the queue of each neighbour. These queues are updated every time a *TX* or a *Block* is received to prevent the nodes

from sending information that a node already has. Periodically these queues are flushed and the node sends the *Inv* messages to the respective neighbours containing the content of the queue of each one.

Although there are already some attempts to lower the amount of bandwidth consumed and the number of messages sent this system still possesses some inefficiencies discussed in the next section.

3. IMPROVEMENT IN THE BROADCAST OF TRANSACTIONS

In this section, we propose a set of changes to the dissemination algorithm of transactions with the objective of making it more efficient, namely by lowering the number of redundant advertisements that each node receives. Our proposal is based on the following observations:

- Currently, each node receives on average 6.6 duplicated advertisements for each transaction (when would be enough receiving a single one to ensure the reception of a transaction).
- The network currently possesses two procedures to disseminate transactions: exhange of advertisements (used when a transaction is not in a block) exchange of block (used when a transaction is already added to a block).
- For historical reasons, the second mechanism is more efficient than the first one, because all missing transactions a node might request are sent in a single message (meanwhile through the advertisement method a node has to send a message for each individual transaction).
- In Bitcoin, the constraints for the broadcasting of transactions are weak because the rate of generation of the blocks is much slower than the processes of dissemination of transactions (on average a block is generated once every 10 minutes)
- Miners are only a small fraction of the total number of nodes in the network. And although it essential that transactions reach miners the protocol does not distinguish miners from the rest of the nodes.
- In the current protocol, nodes send their advertisements to all their neighbours (125 in the worst case). This value is substantially higher than the theoretical value for epidemic broadcasting algorithms, which suggests that even in the presence of failures its enough to send information for a number of neighbours proportional to the size of the network which is approximately around 10 000 nodes hence it would be enough to send to $ln(10\ 000) \approx 10$ neighbours.

The main objective is to lower the amount of duplicated advertisements in the network and at the same time ensure that the transactions reach the miners. The intuition for the proposed approach is to skew the process of dissemination in the direction of the most productive miners. Without however putting the resilience of the system at stake by broadcasting the information to the rest of the system through alternative paths. This is achieved through tree changes to the protocol. The first modification consists in saving for each block that we receive from a neighbour the time that neighbour took to disseminate the block to us. The second one is also for each neighbour maintain a list with the transactions we sent to him a how much time it took for those transactions to be added to a block. Finally, the third change is locally determining which path is better for sending our transactions so that they reach the miners faster and give priority to these paths in the dissemination process. Note that once a transaction is added to a block the epidemic dissemination process becomes irrelevant.

3.1 Ranking neighbours

As referred previously, the strategy proposed to lower the bandwidth used, has a requirement that each node discovers which path is the fastest to reach the miners. The miners to where we have the fastest paths will rank higher in the list, and the ones with the slowest paths will rank at the bottom of the list.

To obtain this ranking of the neighbours each node will have to maintain for each neighbour five variables:

- n total number of blocks received by that neighbour;
- k accumulated time it took to disseminate a block to us;
- a total number of blocks received;
- z total number of transactions we sent to that neighbour;
- y accumulated time it took for those transactions to be accepted in a block.

The time it takes for a neighbour to relay a block until a node is given by the difference between the current time and the last time the node received a new block from that neighbour. If a neighbour takes more than four hours to relay a block to a node the difference will be the current time minus four hours. If Given the large number of transactions that flow through the network, instead of maintaining timers for all of them we only maintain a timer each one hundred transactions. This way we do not overburden nodes with metadata.

$$\text{class}^T = (\frac{k^T}{n^T} + a^T - n^T + \frac{y^T}{z^T})$$

With this classification, we prevent nodes that only generate a block once in a while from having a good classification eternally.

This is important because as we have said, there is still a percentage of blocks being mined by random nodes, and if we would, for instance, send all our transactions to those nodes they would take a lot of time to appear in a block. Which is not desired, given that the average time it takes for a transaction to be accepted in Bitcoin nowadays is ten minutes.

This way for a neighbour to have good classification it has to have a good ratio of time it takes to disseminate blocks/number of blocks we received from him, a good ratio of blocks received from him/blocks received and finally a good ratio of time it took for a transaction to be added to blocks if we sent it to him.

Give that the classification of neighbours is prone to change over time, the actual value used to order neighbours is given

Algorithm 1 Top neighbours computation

```
1: function UPDATE_NODES_CLASSIFICATION(node_to_update)
2:     scores ← [ ]
3:     for node in neighbourhood do
4:         score ← get_classification(node)
5:         scores.append([score, id])
6:     end for
7:     sort(scores)
8:     top_nodes ← [ ]
9:     for i in range(0, max_top_nodes) do
10:        top_nodes.append(score[i][1])
11:    end for
12: end function
```

Algorithm 2 Nodes to send transactions advertisements computation

```
1: function NODES_TO_SEND(tx)
2:     if (ip == True and tx.source() == self) then
3:         return neighbours
4:     end if
5:     total ← max_t_nodes + max_r_nodes
6:     if size(neighbours) < total then
7:         total ← size(neighbours) − max_t_nodes
8:     else
9:         total ← total − max_t_nodes
10:    end if
11:    if total > 0 then
12:        r_nodes ← rand_choice(neighbours, max_r_nodes)
13:    end if
14:    return t_nodes + r_nodes
15: end function
```

by the following sliding average of the classification presented previously:

$$\text{class}^t = (1 - \alpha) \cdot \text{class}^{t-1} + \alpha \cdot \text{class}^T$$

The α factor exists to avoid nodes that once generated a lot of blocks but currently do not from having a good classification forever. In our experiments, we used an $\alpha = 0.3$ and a T configured to be an interval of four hours.

Each time a node receives a block from a neighbour the classification of the neighbours will be updated using Algorithm 1

3.2 Skewed relay

Given that our objective is that our transactions reach a miner as fast as possible we can use the mechanism described previously to do it. Hence, if all the nodes were to follow the protocol correctly, and the paths created by out solution were resilient enough we could send our transactions to only one node, and they would eventually appear in a block.

However, even if we do not consider the problem of nodes disconnecting from the network, there is the problem of time it takes for a transaction to be committed. So to solve this problem, we will send our transactions not only to the t top nodes but also to r random nodes, as it is described in Algorithm 2. This way we assure that our transactions are still broadcast through the rest of the network and will be committed in a timely manner.

The value of Initial Push (ip) indicates that if a transaction is generated by a node, the node has the option of either sending it for only t plus r or to all his neighbours.

Hence we can then configure the following variables *ax_top_nodes*, *max_top_nodes* and *ip* to obtain different results in the information dissemination. In Chapter **??** we will dicuss diferent configurations tested.

Algorithm 3 Increase or decrease top and random lists computation

```
1: function INCREASE_RELAY
2:     avg_time ← get_avg_time_unconfirmed()
3:     timeout ← avg_time() > TIME_TX_CONFIRM
4:     space ← max_t_nodes + 1 <= neighbourhood / 2
5:     cooldown ← time_since_last_inc + TIME_TO_WAIT <= now
6:     if timeout and space and cooldown then
7:         increase(t, r, 1)
8:         had_to_inc ← True
9:         update_nodes_classification()
10:        time_since_last_inc = now
11:        relay_delayed_TX()
12:    end if
13:    cooldown ← time_since_last_dec + TIME_TO_WAIT <= now
14:    if not had_to_inc and cooldown then
15:        avg_time ← get_avg_time_confirmed()
16:        timeout ← avg_time() <= TIME_TX_CONFIRM
17:        space ← max_t_nodes - 1 <= 0
18:        if timeout and space then
19:            decrease(t, r, 1)
20:            update_nodes_classification()
21:            time_since_last_dec = now
22:        end if
23:    end if
24: end function
```

3.3 Complying with network changes

A key aspect of peer-to-peer networks as stated previously is that nodes can leave or join the network at any given time. With this in mind and also the fact that is not possible to exactly determine which node is going to mine the next block, we designed an algorithm that adapts to the network in order to maintain the commit time of the transactions while still trying to send as few messages as possible. Our algorithm increases or decreases the values of *max_t_nodes* and *max_r_nodes* by one depending if either, our transactions are taking too long to be committed to a bock or if they are being committed in a reasonable time as depicted in Algorithm 3.

As shown in Algorithm 3 if the current average, of the time it takes for the unconfirmed transactions of a node to be committed to a block, gets bigger than the constant *TIME_TX_CONFIRM* (30 minutes) then the node is going to increase both the size of *max_t_nodes* and *max_r_nodes*, relay the transactions that took more than 30 minutes to commit. If for instance, all the current average of the unconfirmed transactions does not surpass the threshold of the 30 minutes then, the node will check if the average time it took to confirm its transactions confirmed in the last hour took less than *TIME_TX_CONFIRM* if yes then the node will reduce *max_t_nodes* and *max_r_nodes* by one else it will not do anything. Each time *max_t_nodes* and *max_r_nodes* are changed the node will not be able to change these values in the next 2 hours to prevent fluctuations in these values. Furthermore, every time a node increases *max_t_nodes* and

max_r_nodes it will not be able to decrease these values in the next 4 hours in order to preserve resilience over performance.

4. EVALUATION

To evaluate the proposed approach, we built an event simulator that modules the broadcast of transactions and blocks in the Bitcoin network (this simulator was developed in python). We decided to implement our own simulator because all the simulators that we found were either outdated or were not working.

4.1 Simulator tunnig

In order to tune our simulator so that it simulates the original protocol faithfully, we extended the *Bitcoin Core* client, the most used Bitcoin client to log metrics about the messages exchanged between clients. The metrics logged were the folowing: i) transactions advertisements; ii) received transactions; iii) transactions present in compact blocks that the node had to request to be able to rebuild the block. We deployed two instances of this client in two distinct physical locations for a whole month and used the metrics logged by these two clients to tune our simulator. Furthermore, we also used information publicly available on the website https://blockchain.info/ to determine the number of transaction generated, the distribution of blocks generated by miners and the average transaction size. With all these metrics we implemented the original protocol, and then we added our changes to the protocol. We experimentally tunned our simulator so that the results observed were the same as the ones observed in the real client. The network model that we used in our simulations was composed solely of nodes that followed the protocol accordingly.

Running simulations of big Bitcoin networks in a language not optimized for efficiency resulted in the simulations being very demanding in terms of computations and space required which resulted in the simulations taking a long time (scale of days) to complete. Because of this, we tried to see if lowering the size of the network was going to impact the results. To do this we ran the original protocol with 6000 nodes and with 625 nodes and compared the metrics discussed further. The results we obtained were equivalent for both network sizes hence, for the rest of this section we consider a network size of 625 nodes. This proportional scaling between 6000 nodes (size registered when we started experimenting) and 625 nodes, allowed us to explore quickly the possible solutions and run multiple instances of each test. The results presented are an average of 3 independent runs that correspond to 24h in the real world.

4.2 Skewed relay impact

We started exploring the different possible solutions that would help us lower the resources used without having a negative impact on the system. To make it easier differentiate between the different experiences we will be using the following notation: *Tn* where n specifies the value of the variable *max_t_nodes*; and *Rn* specifies the value of the variable *max_r_nodes* present in the previous algorithms.

Initially we tested with multiple combinations of $n = 1, 2, 3, 4$ for both T and R. After these preliminary experiments, we observed that for values of $n = 3, 4$ the results were practically the same as the results without ou approach. However, with $n = 1, 2$ we observed a considerable reduction in the number of duplicated advertisements. These results also support our logs in the real client, where the average number of duplicates was 6.6. With this in mind, for the rest of the experiments, we considered only the combinations of: T2_R2; T2_R1; T2_R0; T1_R1; T1_R0. Additionally, for each configuration, we also experimented with both values of the variable *ip*. In the results presented we also discarded the first and last 5 hours of each run in order to study the system in stable statel.

5. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

6. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

7. REFERENCES

[1] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.