Purple - A Fast, Programmable, Multi-Asset Blockchain Protocol

Octavian Oncescu
octavonce@gmail.com

Abstract

Purple is a blockchain protocol that has been designed with an emphasis on scalability and long-term use. This is achieved by using an asynchronous stateful consensus algorithm that is coupled with a synchronous consensus algorithm which serves as an adjustable bottleneck. It can be said that Purple is a semi-synchronous system with the synchronous part serving as it's rate-limiter. This enables fine control over the scalability and the safety properties of the system. In this paper we present how such a system, when using the correct parameters, can provide a sustainable blockchain protocol which can support the stress of mass usage.

1. Introduction

What enables a protocol such as Purple to exist is an algorithm which can effectively replicate data among a number of nodes on a network, even when a large portion of those nodes are behaving in a malicious manner. This property is called *Byzantine Fault Tolerance* and achieving it in a system requires solving the Byzantine Generals Problem[1]. Satoshi Nakamoto solved this problem when he created Bitcoin by using the *Proof of Work* algorithm[2]. It works by requiring participating nodes to provide mathematical proof that a problem of deterministic computational complexity has been solved. The node which solves the puzzle is the designated node which can write a block of transactions to the ledger. If the proof, along with the transactions inside the block are correct, the rest of the nodes will accept that block as part of their local ledger. If two nodes provide a valid proof at the same time, the block that will be indirectly referenced by most future blocks will be chosen as being the correct one.

While this works, it has a big problem: The network has to wait for a node to provide a valid proof before it can consider a set of transactions to be processed, which is a very slow process.

A few attempts have been made to address this, the best of which is called *Proof of Stake*. It works by requiring participants that wish to validate transactions to deposit a large sum in an account and providing proof of this. If the participant approves transactions maliciously, his deposit is taken away. In this way, a participant would lose a lot more by acting maliciously than by being correct.

Proof of Stake greatly increases the speed of the network but it introduces several problems of it's own: The deposited amount must be objectively valuable otherwise there is nothing to be lost for acting maliciously which means that a network using Proof of Stake can only bootstrap itself by requiring it's creators to act benevolently until the objective worth of the required deposit is great enough and until a sufficient number of other, unrelated entities amass it. Another problem is that the network's speed is directly dependant on the availability of the nodes which are owned by the entities that have stake. This means that in order for a network using Proof of Stake to function correctly, the entities which have stake are required to not act maliciously and their nodes to not ever become compromised.

2. Related work

Tracking causality and achieving consensus in a distributed system are not new problems. A solution to the former has been initially proposed by Leslie Lamport which involves the use of a monotonically increasing counter called a *Logical Clock*[3]. Since then, improvements have been made: *Vector Clocks* and later, *Bounded Version Vectors*[4]. Both of them provide a way of establishing a causal order of events in a system with a fixed size of replicas and have been successfully used in the construction of large-scale distributed databases. Another improvement on logical clocks has been later presented which allows a dynamic number of replicas called *Interval Tree Clocks*[5]. They provide a framework for the decentralized creation and retirement of replicas called the fork-event-join model.

Another component of establishing consensus in a distributed ledger is the method in which Byzantine Fault Tolerance is provided. For this, Purple borrows the last of the Byzantine-Resistant Total Ordering Algorithms presented in [6]. It assumes that a number of replicas are either byzantine or unresponsive and that the communication between them is asynchronous.

3. Semi-synchronous consensus

The consensus algorithm of Purple is called Semi-Synchronous Proof of Work, or $SSPoW$ for short. It is a variation of Satoshi's original Proof of Work model of consensus[2]. It's design is to remove the bottleneck that the Proof of Work algorithm imposes on the transaction throughput of the network.

This is done by decoupling the choosing of validator nodes, which is done via Proof of Work from the actual transaction validation process. When a node finds a valid Proof of Work, it is advanced into the validator pool where it has an allocated period in which it can validate transactions. This is done asynchronously while the choice of validator nodes (PoW) is synchronous. In this way, the consensus mechanism becomes semi-synchronous, greatly increasing the throughput of the network while providing a safety control mechanism which can be adjusted based on the current network conditions.

The algorithm works by establishing a byzantine partial causal ordering on the network events that are sent between the validator nodes and by transforming it into a total order which is assured to be consistent as long as less than a third of the validators are either byzantine or unresponsive. However, the Total Ordering Algorithms assume that the communication medium between the nodes is reliable so it falls on the CA side of the CAP spectrum. Another step must be included in the algorithm in order to provide partition tolerance. If a network partition happens and this step is not provided, each partition will consider that the nodes from the other partition have crashed and will remove them from their configuration producing a fork of the ledger state.

3.1. Validator pool

Transactions are validated by nodes that are in the validator pool. In order to participate in the validation process, nodes have to issue network events, in a deterministic order. A network event issued by a node contains pending transactions which the node wishes to include in the ledger.

When nodes join the pool, they are placed on a circle which is represented by the interval $[0, 1)$. Each validator in the pool owns a share of this circle which is represented by a sub-interval of $[0, 1)$.

The order in which the nodes are placed on the circle determines the order in which they are required to issue events. The node owning the lowest share of the interval is always required to be the first to issue an event.

When a node joins the pool, the node with the largest share must give half of their share to the joining node. If a node leaves the pool, a node is deterministically chosen to receive the leaver's share.

Pending transactions are deterministically partitioned among all curent validators in order to prevent two nodes from validating the same transaction. A validator node receives the transaction fees of all the transactions that it has processed as reward if it isn't found to be byzantine or crashed.

3.2. Byzantine fault tolerance

Byzantine fault tolerance is provided by converting a set of network events, some of which might be originating from byzantine nodes, into a consistent total order. This is done by using a modified version of the last of the Total Ordering Algorithms which consists of using the causality relationships that are formed between the events that are residing in the partial causal order.

Using this, a vote on which events are to be added into the total order can be performed. The voting is done by other pending events that are continously sent by the validators. When an event is sent by a validator, it votes for all of the events that it follows that are also contained in the partial causal order.

3.3. Partial causal order

A partial causal order between events is established by requiring validators to include their current logical timestamp in each event that they create. This timestamp is the two element tuple $(i,\ e)$ where i is the node's assigned id and e is the logical timestamp of that node. This will be stored locally on each validator node and can be thought of as a buffer where new events are being stored.

Using these, a happened-before relationship can be established between two network events: Given the logical timestamp $t_1 = (i,\ e)$ we can say that $t_2 = (i,\ e') = event(t_1)$ and that $t_1 < t_2$. If given 3 network events t_1, t_2, t_3 with the relationship between them being $t_1 < t_2 < t_3$ then we can say that $t_1 < t_3$.

3.4. Causal graph

Given a set of arbitrary network events and using the properties presented in the last sub-section, we can determine a set of follow relationships between these events. A causal graph can then be constructed using these relationships. Each event is a vertex of the graph and the edges of the graph represent the follow relationships between the events.

Given two timestamps t_1 and t_2 where $t_1 < t_2$ and each of them belonging to different network events, we say that t_2 follows t_1 in the causal graph. If instead, $t_1 = t_2$ then no causal relationship is formed between the two timestamps and we can say for certain that the node they came from is byzantine since no non-byzantine node will ever send two different events with the same timestamp.

If $t_1 \neq t_2$, $t_1 \not< t_2$ and $t_1 \not> t_2$ then we say that t_1 and t_2 are concurrent, meaning that no follows relationship between them can be established.

3.5. Total ordering

Given a causal graph between a set of arbitrary network events (some of which might be byzantine) we can say that a sub-set of the events are eligible for being advanced to the total order if they follow each event from the current total order and if they do not follow other events that are currently in the partial causal order. A sub-set that follows these conditions is called a *candidate set*. There can be multiple candidate sets that are eligible at the same time, however, only one of these sets will be chosen for advancement into the total order while the other ones will be discarded.

The criteria for advancing a candidate set into the total order by a node is determined by votes and proposals from other events that are currently in the partial causal order of that node. We can say that an event e is eligible to vote on a candidate set S if either $e \in S$ or S meets the eligibility criteria. An event is eligible to vote on a candidate set if it is followed by E events from distinct nodes where $E = (N + 1)/2$ and N is the number of validator nodes that are currently in the validator pool.

A candidate set is advanced into the total order if it has P_n proposals, $P_n = N + 1$. An event e proposes for a candidate set S if it follows at least P events that vote for S where $P = (N + 1)/2$.

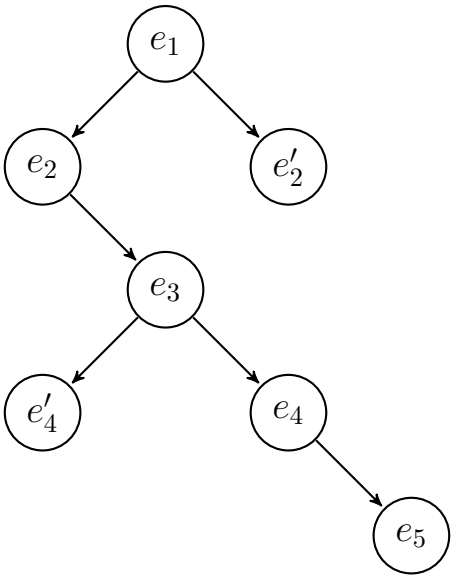

Fig. 1 Anatomy of a causal graph. e_5 follows e_1 through e_2, e_3 and e_4 while e'_2 and e'_4 are messages from byzantine nodes since they have equal stamps with e_2 and e_4.

3.6. Partition tolerance

Once an event has been advanced from the partial causal order into the total order, it is added to a local buffer by each validator node. When the buffer is filled, each node will take it's local events, it will package them into a block and then it will broadcast the block to all of it's peers. If the events in the block and their causal relationships are correct, other nodes will consider the block as a candidate for being written to the ledger state. The way in which a block is chosen as being the correct one to be written to the ledger is the same way as in Bitcoin, meaning that the block which is followed by the most subsequent valid blocks will be chosen as being the correct one.

3.7. Preventing sybil attacks

The only problem so far is the way in which validator nodes are chosen. If any node can become a validator when it wishes to do so, the consensus mechanism would be obstructed by any moderately capable sybil attacker. For this, the probabilistic model of Bitcoin will be utilized, not for achieving consensus on the ledger state but to achieve consensus on which nodes are currently validators. Thus, nodes participating in the Purple Protocol will store, besides the current state of all accounts, balances and contract storage, information about the current validator nodes.

A node becomes a validator by providing a valid Proof of Work on the latest join event that has been witnessed. When a valid proof is found, the node broadcasts the proof to all nodes and the node is synchronously added to the validator pool. Thus, it can be said that Purple is a semi-synchronous system with the synchronous part serving as an adjustable bottleneck to the asynchronous part of the consensus algorithm.

3.8. Transactions

Transactions are processed by being included in the network events that each validator broadcasts. A validator node does this by taking a number of valid transactions from the pool of pending transactions and then including them in a network event which is then broadcast to all of the other validator nodes[Fig.2]. The order between events is then established without further processing of transactions but instead only using their parent event's causal timestamp.

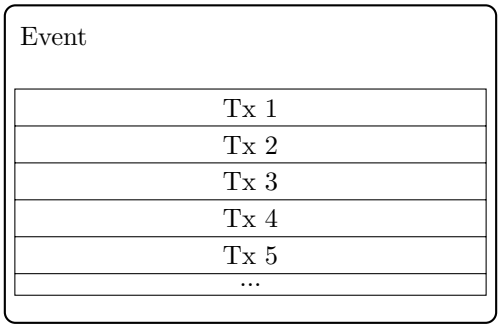

Fig. 2 Multiple transactions are included in a network event, making it possible to approve transactions in batches.

Once the event has been advanced to a block, the transactions are considered to be included in that block and will be applied to the state of each node that has received the block[Fig.3].

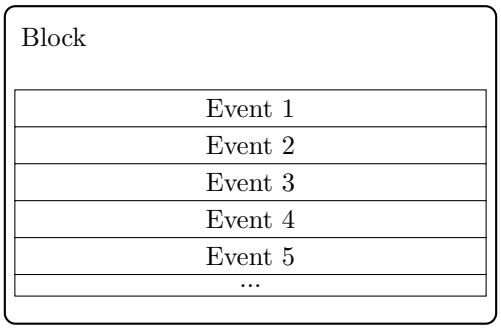

Fig. 3 A block is composed out of multiple network events that satisfy the order $e_1 < e_2 < e_3 < ...$

4. Multiple Assets

One of the most important aspects of the Purple Protocol is it's capability of handling different assets in the same way as handling the main currency of the protocol. There are several reasons for this, the most important one being that any created asset will follow the same rules as the main currency of the protocol, creating a common format which can be used by both developers and users.

Assets are handled at the protocol layer, which means that they can now effectively behave in the same ways that the main currency does i.e. users can pay transaction and gas fees in any currency. This makes the Purple Protocol very suitable for long-term usage: if for any reason the main currency of the protocol is devalued or is otherwise lost, any other listed currency can compete for taking it's place as the main one.

4.1. Vanilla tokens

Vanilla tokens represent fungible assets of the same type that are either mintable or non-mintable. They can be used to pay transaction fees and gas fees for smart contracts. A non-mintable token has a fixed supply and once created, no more of that currency can ever be created. A mintable currency on the other hand, has a current supply and a maximum supply. Then at a later time, more tokens can be issued.

4.2. Security tokens

Security tokens in Purple are owned shares in a shareholder controlled account. Addresses that own part or all of the shares of a shareholder account are eligible to sign transactions made from that account. For example, let us suppose that a shareholder account has a total of 1000 issued shares and the required share percentage for approving a transaction is 60%. Alice owns 300 shares, Bob owns 400 shares and Carol owns the remaining 300 shares. In order then for a transaction to be approved, a total of 600 shares of accumulated voting power is required, meaning that a transaction must be signed by either Alice and Carol, Alice and Bob, Bob and Carol or all three of them. A transaction cannot be approved by a single one of them since none of them own more than the required percentage.

The shares of a shareholder controlled account are transferable. Meaning that any shareholder can transfer part or all of his shares to another address. The majority of a shareholder controlled account can later choose to issue more shares, effectively diluting their own current shares in the favor of a receiving party.

4.3. Decentralized exchange

As a result of having native multi-asset support, a decentralized exchange can be deployed as part of the core protocol. When a vanilla or security token is created, it will be immediately listed on the exchange if there are any available sellers. The exchange can also function as an oracle for smart contracts who wish to inquire about the current price of an asset.

5. Virtual machine

The virtual machine is used by the protocol to run smart contract code. In Purple, a structured stack machine is used, very simmilar to webassembly but optimized for blockchain systems. The structured stack machine approach enables formal verification at the protocol layer but the instruction set of webassembly is extended with blockchain specific instructions and types. At the same time, a lighter module structure is used when compared to webassembly, because of storage optimizations.

The Purple assembly format, or PASM for short, is designed to be a compiler target for modern compiler infrastrcture such as LLVM. This potentially enables developers to write smart contracts in any compiler front-end targeting LLVM, which includes many well known programming languages.

6. Treasury

In order to create an incentive for using the main currency and to provide a way to foster future development of the protocol, a treasury is defined at the protocol layer. If a miner approves transactions with fees paid in other currencies than the main one, a small portion of the fee that they receive will be transferred to the treasury account.

The treasury account will be owned by multiple shareholders who can approve transactions made from it and will have a fixed limit on what percent of the token supply it will be able to hold. If deposited funds overflow this limit, the excess will be paid to the currently active miners.

References

[1] The Byzantine Generals Problem. Leslie Lamport, Robert Shostak and Marshall Pease. SRI International. *https://people.eecs.berkeley.edu/ luca/cs174/byzantine.pdf*

[2] Bitcoin: A Peer-to-Peer Electronic Cash System. Satoshi Nakamoto. *https://bitcoin.org/bitcoin.pdf*

[3] Time, Clocks, and the Ordering of Events in a Distributed System. Leslie Lamport. Massachusetts Computer Associates, Inc. 1978. *https://amturing.acm.org/p558-lamport.pdf*

[4] Bounded Version Vectors, Jose Bacelar Almeida, Paulo Sergio Almeida, and Carlos Baquero, Departamento de Informatica, Universidade do Minho, 2004. *http://haslab.uminho.pt/jba/files/04bvv.pdf*

[5] A Logical Clock for Dynamic Systems. Paulo Sergio Almeida. Carlos Baquero. Victor Fonte. DI/CCTC, Universidade do Minho, Braga, Portugal. 2008. *http://gsd.di.uminho.pt/members/cbm/ps/itc2008.pdf*

[6] Byzantine-Resistant Total Ordering Algorithms, Louise E. Moser and P. M. Melliar-Smith. 1999. *https://ac.els-cdn.com/S0890540198927705/1-s2.0-S0890540198927705-main.pdf?*$_t id = 90201aa1 - dacf - 4bfe - 984c - 5a71fffe7b30acdnat = 1529842769_3dbf938d6bdb5257874eba46ec39f0bb$