A BAR Resilient P2P network for distributed ledger systems

Archie Chaudhury
Adamnite Labs
archchaudhury@adamnite.org

In distributed computing systems, a peer to peer (P2P) network is often used to relay information between different nodes. Peer to peer systems are typically used to reach consensus, keep a record of the system, and to send messages in the context of the wider protocol. In distributed ledger systems, most specifically in blockchain and cryptocurrency protocols, peer to peer networks are used in conjunction with a timestamped server to broadcast information about new transactions or blocks, relay updates to the system, and perform consensus-related tasks such as selecting witnesses. In most contemporary implementations of such technology, a gossip protocol is used to propagate information across the P2P network, allowing nodes to connect randomly with each other and receive information about one another. The Hedera Blockchain, for example, uses a Directed Acyclic Graph (DAG) that stores specific events within its Gossip Protocol to act as a ledger for transactions, in contrast to the chain of timestamped blocks used by Bitcoin and others. However, with such systems becoming more and more popular, the likelihood of Byzantine and rational actors attempting to attack or take advantage of the network respectively becomes more and more likely.

In this work, we present an alternative P2P network/Gossip mechanism optimized for BAR (Byzantine, Altruistic, and Rational) Resiliency that allows for all three types of participants to coexist without compromising the integrity of the network while maintaining an acceptable degree of efficiency. Our algorithm can be considered an implementation of the general BAR Gossip protocol proposed by Li, Clement, Wong and others; however, we optimize our implementation for a blockchain rather than a streaming service that can rely on some form of centralized storage. Our method incorporates whitelisting and pseudorandomness to increase the likelihood that the set of nodes that node *x* connects with are either altruistic, or are rational nodes with a self-interest in relaying legitimate information through an optimistic push or long-term fair exchange. Unlike contemporary P2P network implementations (such as Ethereum), we do not base our network implementation on a preexisting hashtable model such as Chord or Kademila. We assume implementation on a Byzantine Fault Tolerant Ledger that has some amount of defense against Sybil Attacks (ie by selecting block proposer through the validation of some scarce resource, such as computing power, units of currency, or stake holder votes), and that both altruistic and rational nodes ignore messages that do not follow the underlying network's structure. Our method is meant to optimize BAR Gossip for distributed ledgers, and hopefully can serve as a standard for secure P2P network development for distributed ledger projects.

The importance of BAR Resiliency

BAR Resiliency ensures that gossip channels in a network do not become corrupt due to the actions of Byzantine or Rational individuals. This is especially a problem in nascent Proof of

Stake blockchains, where both the hardware costs for running a node to interact with others in the P2P network and the size of the P2P network itself are significantly lower. Without the proper protections, a malicious individual can easily execute some version of an eclipse attack and target individual nodes (some of which could be the validators for the network), preventing them from getting updates and thus altering their view on the current state of the blockchain. In Delegated Proof of Stake Blockchains, elected witnesses (or in the case of random witness selection, those likely to be elected) can be targeted, potentially causing the network to stagnate, or worse, putting malicious witnesses in charge indefinitely. The aforementioned BAR Gossip model is a potential solution, as it provides a basis for how “free-riding" rational nodes can coexist with altruistic nodes. In a BAR Resilient network, byzantine nodes who refuse to abide by the rules of the protocol are naturally excluded, providing the network with a certain level of security. Rational nodes, on the other hand, are naturally incentivized to participate in a fair exchange of information and provide optimistic pushes to other nodes as necessary.

Network Structure and Protocol

For our BAR Resilient Network model, our primary goals are:

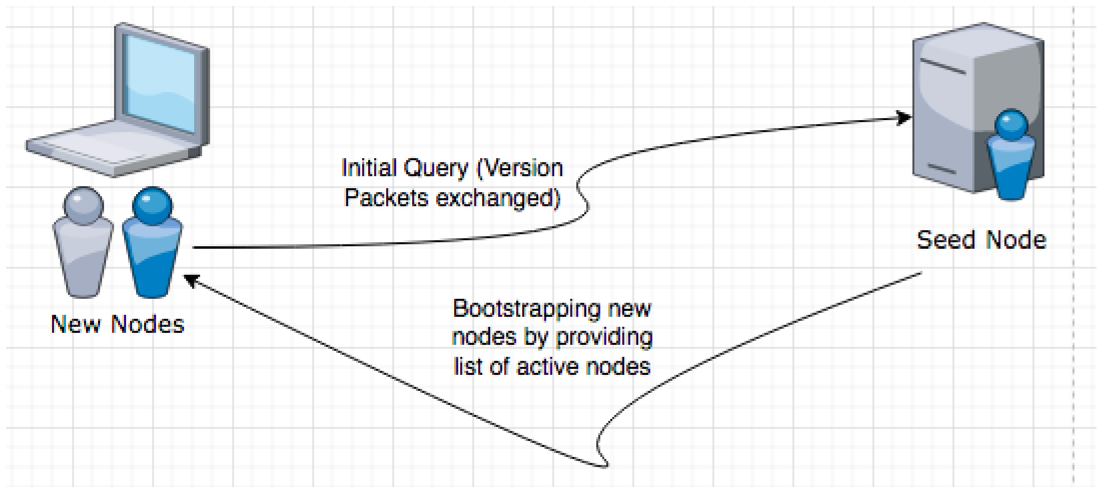
- **Security from basic eclipse attacks**- The network should be isolated from man in the middle or isolation attacks where byzantine nodes try to isolate a specific node and alter their view on the current state of the blockchain, potentially before executing a more malicious or complex attack.
- **Efficiency**- Efficiency based on a traditional epidemic gossip model should be maintained, without compromising the intrinsic security measures of the protocol.
- **Ease of entry for new nodes**- New nodes entering the network should be able to easily connect and get new information via an optimistic push with altruistic nodes or incentivized rational nodes. We assume the existence of a secure point of entry, either controlled by the core developers of the underlying distributed ledger or a group of trusted altruistic nodes.
- **Long-term scalability**- The protocol should be able to handle a theoretically unbound amount of new nodes, regardless of their client implementation (light, full, etc).

We set the following constants and generalized black-box functions:

- **T-** The amount of time that a node will wait for synchronization.
- **K-** The amount of outgoing connections a node will accept.
- **J-** The amount of incoming connections a node will accept.
- **H-** A blackbox PRNG (Pseudorandom Number Generator) that governs how nodes are paired with each other.
- **R-** A round within the consensus protocol (this can be a single block proposal, as with Proof of Work chains, or multiple blocks set within a predetermined frame).

We specify the protocol as follows. Every node has the following parameters: node id (an unique identifier for the node), a whitelist (a list containing a list of reputable peers that the node can connect with), and a greylist (a list containing alternative peers). By default, each node maintains **K** outgoing connections and **J** incoming connections (for simplicity, we assume that all nodes. Each node is required to maintain at least one incoming connection and one outgoing connection. We classify nodes into three different categories, as previously established: byzantine, altruistic, and rational. We assume that the underlying protocol is TCP or some other similar networking protocol with reasonable defenses against IP spoofing.

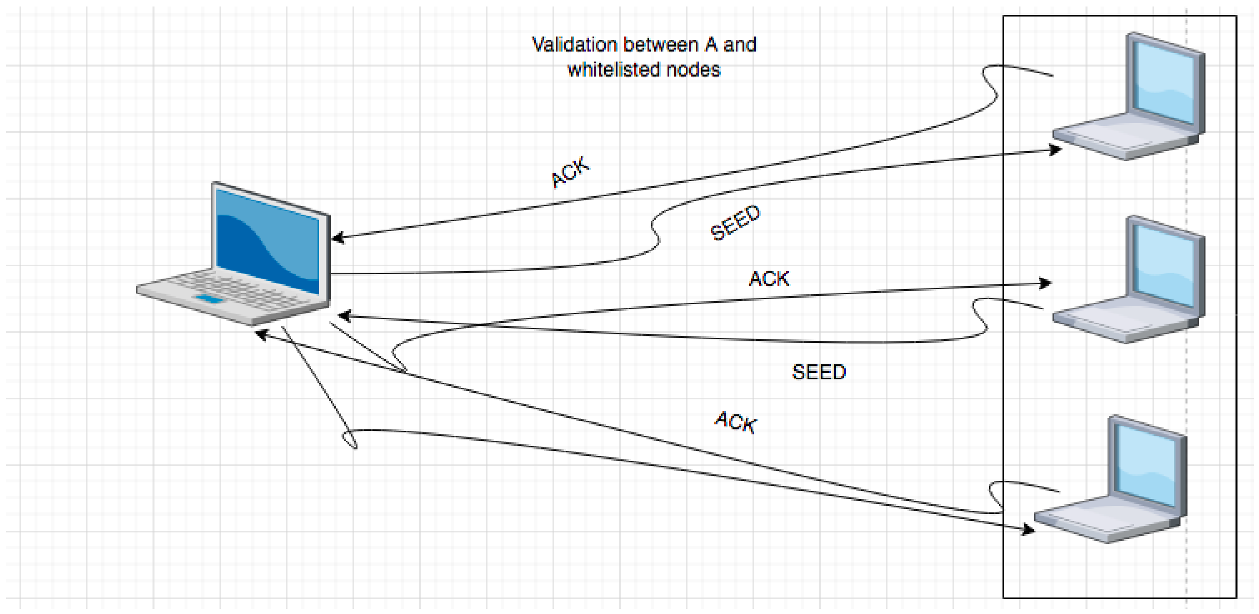
The node entry process is defined in the following diagram:



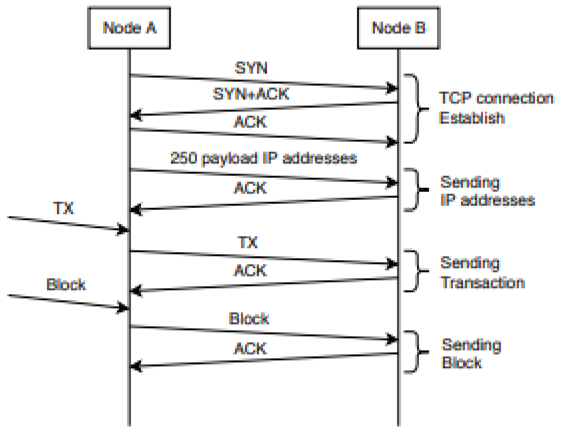
A seed node, presumably maintained by an altruistic party (either the core team of developers for the project or a trusted external party), serves to onboard new nodes that may join the protocol. Although connecting to a seed node is not necessary, we assume that a majority of nodes (regardless of whether they are byzantine, altruistic, or rational) will want to start by connecting with a seed node. The initial connection, with a majority of other distributed ledger protocols, involves the exchange of node versions and information. Each version packet has the following information: client_version, timestamp, addr_recieved, addr_from, last_round, and nonce. Following the terminology described in the BAR Gossip Paper, we define the process for how a new node receives information and propagates information throughout the network. Once a new node has finalized an initial whitelist (from a seed node), it will query initial block information from those peers through a protocol handshake. These peers will perform an optimistic push to the new node, essentially giving the entirety of the block history to the requesting node (variations, such as just block headers, are acceptable for light nodes). While altruistic nodes will naturally perform this action for the benefit of the network, rational nodes are encouraged to play by the rules as well: nodes that do not perform an optimistic push for a certain period of time (say 10 **T** seconds, or the completion of 10 rounds within the protocol) will be removed from the whitelist of their peer, and nodes that are removed from all whitelists due to prolonged inactivity are marked as inactive by seed nodes.

A new node **A** receives an initial list of peers to connect to from the seed node it initially connected with. These nodes are initially sorted into the node's greylist; from there, the node executes a *Handshake* protocol with each of the nodes in the greylist to determine the nodes that will be promoted to the whitelist. This process is repeated every round **R**, and whenever the total number of nodes in the whitelist drops below the number of incoming connections **A** wants to maintain. The *Handshake* protocol, as with Monero, is simply a ping: a response results in the node being promoted to the whitelist, and therefore makes it eligible to be connected with. The *find_nodes* algorithm, on the other hand, dictates how nodes in the whitelist are selected for any particular round. The *find_nodes* algorithm is especially important for maintaining BAR resiliency, as it dictates how nodes pair with others and is the first barrier to filtering out byzantine nodes.

Every round **R** (beginning with the first round for which a node is eligible to receive information), **A** runs **H** with a random seed, generated when the node is first created. This random seed is incremented with a salt (known to A) for each **R. A** then maps the first eligible **J** numbers generated by **H** into its whitelist to determine the incoming connections it will use for that round. When trying to connect with a node **B** in its whitelist, **A** performs the customary version exchange, but also provides the seed it used for that particular round. **B** then checks to see if the value provided is valid based on **H**, and if **H** actually maps to **B** within the context of the protocol. **B** only accepts **A** (by sending the traditional ACK message) if these conditions are met, as there is not an incentive for it to provide or get information from a byzantine node. If all of the nodes in the whitelist suddenly drop out, the node will wait for **T** seconds (one synchronization period) before going back to the greylist.
This process is visualized below:



Upon finalization, nodes exchange information on blocks, transactions, and general messages through a TCP protocol. The diagram below, from a paper describing Monero's P2P network, is a good approximation for the model we want to follow:



The exchange of information (defined after the payload of node addresses) is identical to our protocol

Reputation for further filtering

Our system is not yet fully BAR Resilient, as there is still the probability that a Byzantine node (or a collection of Byzantine nodes) manages to act as a rational or altruistic node before suddenly attempting to feed its peers wrong information or spamming the network with malicious information. To solve this, we employ a reputation system within the P2P protocol itself; please note that this is separate from on-chain reputation that may be employed by some protocols. The idea of node reputation is not entirely novel: it was proposed by Gavin Andreessen for the Bitcoin Core protocol. We integrate the idea of node reputation with Proof of Misbehavior (POM) for our protocol: if a node **A** receives incorrect information (an incorrect version packet, for example) from a node **B**, then **A** takes action by increasing node **B**'s POM score within its local record. The POM scores are stored within **A**'s local memory; to avoid congestion, a node only stores the records for nodes that it was interacting with within the last 10 rounds. Within the last 10 rounds, if **B** exceeds a certain POM score *K*, it is immediately disconnected from and demoted to the greylist. If **B** is selected from the graylist (due to shortage, for example), and subsequently exceeds a POM score of *L*, where *L* is thought to be substantially larger than *K* in the context of the protocol, then it is evicted from the greylist and placed on a ban-list for the remainder of the 10 round period, when the memory for node **A** resets. While this does not fully evict malicious nodes, it does prevent them from spamming the network: a Byzantine node will be ousted by the rational and altruistic nodes, thus leaving them unable to propagate wrong information within a short period of time, which is necessary for executing most DDOS attacks.

Conclusion

We have presented a BAR Resilient P2P Protocol for use within distributed ledgers. Our protocol leverages a greylist/whitelist ordering mechanism, a deterministic pseudo random peer selection process, and a memory-based reputation system to help create an ecosystem where altruistic and rational nodes coexist, and Byzantine nodes are kept at bay. This protocol will drive the development of Adamnite's P2P network. While our implementation focused mainly on blockchain-based implementations, optimizations can make it a good choice for other ledger based protocols. Particular attention could be given to a more formal description of the POM process, and the addition of digital signatures to seed generation for more secure verification. While this is not required for a blockchain-based network, a DAG-based protocol such as Hedera might need additional guarantees for BAR Resiliency.