Blockchain for routing correctness?

Joseph Netti[1]

***Abstract*— Border Gateway Protocol does not guarantee correctness of routing updates. A given routing update for prefix X is *correct* if for every AS in the final AS-path the AS-path that was used to create the ASs respective routing update is still the most recent and valid for destination prefix X. Since BGPsec and RPKI do not provide correctness, this paper explores Blockchain-Merkle BGP (BM-BGP) which uses of blockchain and indexed merkle trees to provide correctness within the time frame of consensus in a chosen consensus protocol. The indexed merkle structure analyzed does not scale well because merkle roots on the blockchain cannot be updated and still have previous merkle verification work without at least log2(N) hashes (where N is leaf nodes in tree) sent to each node trying to validate the correctness of a transaction. This paper only explores theory and does not test the protocol. Sample code for IP indexed merkle trees (code)**

I. Introduction

External gateway protocols such as BGP suffer from data availability problems. This paper explores whether blockchain can be used alongside BGPsec to provide a proof of correctness for routing updates. Specifically, this paper explores whether IP indexed merkle trees can be used to create common knowledge of data correctness to the relevant parties whenever a routing update is received.

Proof of correctness has 3 parts: integrity, identity, and currentness. RPKI and BGPsec (when implemented) provide full integrity, full identity, and partial currentness. In BGPsec, there is a timestamp, however, there it does no guarantee that a given routing update is the most recent one because updates can be withheld maliciously or from new filters []. An important property of Nakamoto consensus is the incentivization of data sharing, which eliminates many data availability problems. However, Nakamoto consensus is not scalable without 2nd layer solutions and traditional 2nd layer solutions cannot be used to help scaling the blockchain proposed in this paper.

A. Shortcomings and Authors Note

There are several shortcomings in this draft. I (the author and researcher) am not confident in my ability to understand the severity of the data availability problem in BGP or any network relying on an external gateway protocol. I need guidance from an expert in BGP to (1) create a set of experiments to test the severity of data availability and (2) understand the current literature around data availability. Since it is one of the first iterations of this paper, I feel it is appropriate to add this disclaimer. I am much more confident in my understanding of blockchain than BGP. I am calling on those who are an expert in BGP to collaborate with me. Thank you.

B. Structure of Paper:

(1) BGP introduction: protocol, vulnerabilities, RPKI, and BGPsec

(2) Blockchain introduction: Byzantine generals problem, Nakamoto consensus, blockchain vulnerabilities, other consensus mechanisms (Ripple), merkle trees

(3) Correctness: causes of withholding and current solutions (IRR, looking glass)

(4) Blockchain for BGP Analysis: indexed merkle trees, variant of Ripple consensus, scalability

(5) Further Work: testing data availability problem in BGP and other networks (bitcoin lightning network)

II. Border Gateway Protocol (BGP)

A. Protocol Description

Border Gateway Protocol (BGP) is the external gateway routing protocol of the Internet [Butler, RFC1771, RFC4271]. Autonomous Systems (AS) are nodes that are controlled by ISPs, transit providers, backbones, governments and other Internet actors use BGP to propagate routes to IP address destination prefixes. For each IP prefix (issued by ICANN or RIR) controlled by an AS, the AS will send a new route to each of its peersthis originates the route. Each AS that receives updates for a specific destination prefix chooses one of the updates, often determined by the shortest number of hops (AS-path) to the destination. Then the AS adds its autonomous system number (ASN) to the route (sometimes multiple times) and forwards the route to its BGP peers. Each node in BGP has a different and incomplete view of the network. The diagram below shows a simple example of BGP.

B. Securing BGP: RPKI, BGPsec, Looking Glass

In December 2017, a Russian AS with number 39523 announced prefixes to Google, Apple, Facebook, and other high-traffic destinations (Toonk, 2017). Inaccurately announcing prefixes is an attack called prefix hijacking. Although the incident in December only lasted a few minutes, some prefix hijacks have lasted hours and significantly rerouted traffic. In 2008, the Pakistan government blocked Youtube for its citizens and accidentally blackholed the rest of the world along with it. The Pakistan incident lasted two hours and is one of the most disruptive prefix hijacks (Youtube Hijacking 2008).

Resource Public Key Infrastructure (RPKI) was proposed in 2012 to secure BGP routing and primarily stop prefix

[1]Blockchain Researcher at Draper and RIT undergraduate. Email: jxn1558 at rit.edu

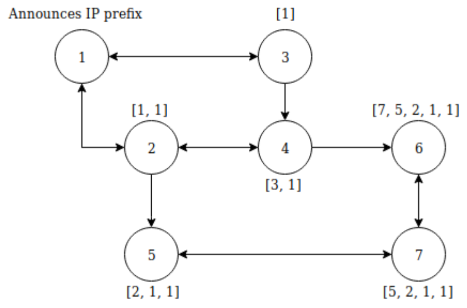


Fig. 1. Updates are sent around the network and the final route chosen by a particular node is printed next to the node in format: [1st hop, 2nd hop, destination]. Arrows show the exchanging of routing updates between nodes. Each hop adds its own AS number (1, 2, 3, etc.). One way arrows mean that the node that is receiving the update has a routing output policy that filters out routing update propagations to the other node for that specific destination prefix. Notice that node 1 prepends twice in the update sent to 2 and once in the update sent to 3. This is a common load balancing technique often used for multihoming and it is called AS-prepending (Butler). Node 1 prepends the AS number twice in the update to 2 so that node 4 chooses [3, 1] as the best path to 1. However, nodes do not have to choose shortest path. Node 6 prefers [7, 5, 2, 1, 1] more than [4, 3, 1] even though it has more hops. One reason a node may prefer a path that has more hops is that it has a low expected latency.

hijacking (RFC8210). With RPKI, an AS that announces a prefix has to provide a resource certificate to prove that the AS has ownership of the prefix. The RPKI trust hierarchy mimics the IPv4 and ASN allocation hierarchy, where ICANN/IANA distributes rights to RIRs which distribute it to ISPs and governments. Each RIR has deployed RPKI, but very few networks have deployed it (NIST RPKI Monitor, 2018).

RPKI is the foundation of a new standard called BGPsec that further secures inter-domain routing (RFC8205). BGPsec verifies each hop in a path to a destination though a digital signature stored in the routing update. Each time an AS prepends their ASN to a path, a signature of the path and prefix must provided. These new update messages replace the AS-Path attribute with a BGPsec-path attribute. BGPsec stops most forms of route forgery and man-in-the-middle attacks. BGPsec is less deployed than RPKI because it requires RPKI to deploy it.

III. Blockchain

The following section is a brief summary of blockchain.

A. *Byzantine Generals Problem*

Blockchain is a decentralized solution to the Byzantine generals problem. The Byzantine generals problem is agreement on some state ordering of valid data while some nodes and communication channels are untrusted or faulty (Lamport). If the data across all honest nodes is consistent with a high probability, the network has achieved agreement or consensus and the Byzantine generals problem is solved. Blockchain in the traditional application is built on top of a P2P network that nodes can freely join and leave (Nakamoto 2008). In blockchain, consensus between honest nodes is simplified to consistent ordering of transaction/state changes across all honest nodes over a reasonable amount of time. This simplification is possible because each honest node in the network has consensus rules that limit conflicting state changes and each transaction is signed by at least one private key that references at least one previous state change. Therefore, disagreement of data is reduced to ordering of transactions. Without agreement on the ordering of transactions, malicious nodes can double spend, which is when a node sends conflicting state changes to different nodes leading to differences in the ordering of transactions (Nakamoto).

B. *Double Spend, Forks, Mining*

Blockchain solves the double spend problem by creating a second layer data structure to hold transactions, and this data structure is called the blockchain. Every state change is propagated across the network in a P2P fashion, but state changes are only fully confirmed if they are in the blockchain and nodes are confident enough that those transactions will remain in the blockchain. The blockchain is a list of blocks that are linked to a single block called the genesis block, which all nodes hold. Blocks are able to form a chain because each block references one previous block by putting the hash of the previous block into the more recent block. Since collisions are highly improbable in hash functions, it is computationally impossible to form a new block that has the same value when hashed as the hash value indicating the previous block hash stored inside a block. The result is a chain that always moves forward. Although new blocks cannot be created that fit behind a block in a chain, new blocks can be very easily created that fit after a block in the chainone sets the previous block hash value as a block hash that one wants the block to come after. When different nodes in the network receive different valid blocks, this is called a fork.

In order to limit the amount of forks, traditional blockchain solutions, such as in Bitcoin, make it costly to produce blocksthis is called mining. The two general forms of mining that make valid block creation more costly in a decentralized fashion are Proof of work (POW) and proof of stake (POS). POW adds an extra limitation to what constitutes a valid block, that is, valid blocks have to hash to a value that is smaller than some predefined number. The predefined number changes periodically in order to keep block creation within some average window of time, limiting forks. In POW, the longest valid chain is the correct chain because it has the most work attached to it. POS is another mechanism where block producers are chosen based on some stake in the system, more specifically some amount of coin and often some attributes of the coin, such as how long the coin has been stored for. POW and POS are mechanisms for ordering transactions and they successfully stop double spending (note that POS research is still in the works with Casper, Ouroborus, etc).

Hard and soft forks are different types of forks because they result from software changes and often cause permanent chain splits (Buterin 2017). Soft forks cannot expand the amount of valid state changes. They add more rules consensus or restructure blocks in a way that is backwards compatible: blocks produced by upgraded nodes can be put in a format that can be accepted by legacy nodes. Segwit user-activated soft fork (UASF) did just this. Segwit created a new transaction type that was represented to legacy nodes as an any-one-can-spend and to upgraded nodes as a segwit transaction which referenced the witness block (which was an addition to the normal block structure) (UASF 2017, BIP148). Hard forks change the rules of consensus such that a subset of blocks produced by the upgraded nodes will be invalid according to legacy nodes (Buterin 2017).

C. Attacking the Blockchain

Various attacks come out of POW and POS when too much of the nodes mining (miners) control block production, meaning that they have the resources to fork at any point in the chain and eventually create a longer valid chain. These attacks are all possible if miners get more than 50% of the block production resources, which is why it is called the 51% attack (although more accurately named the >50% attack) (Nakamoto p.6). The 51% attack allows for censorship and double spending. Miners have an incentive to pool resources and share rewards because miners want more frequent payments. If miners pool into a large enough group (>50% of resources) then the 51% attack can be performed. Dr. Emin Sirer found that the 51% attack can occur lower if miners perform selfish mining. Selfish mining is strategically withhold- ing blocks and mining on the hidden blocks, in order to maximize profit. With selfish mining, the 51% attack can be performed by a mining pool/entity that has >25% of the resources (Eyal 2013).

There are other attacks on blockchain: nothing at stake attack (for POS), sybil attacks, and quantum computers. Briefly, nothing at stake is the problem that block producers do not have any incentive to choose one block, and therefore, will mine after any block as long as they are confident that the blocks before the one they are mining will be accepted by the network. The worry is that miners will mine on top of forks in order to increase the likelihood of a payout, and this does not solve the double spend problem. Sybil attacks occur when malicious nodes cut of honest nodes from the network [Heilman 2015), but it is hard to successfully do if the P2P network is interconnected well (from good seeding). Quantum computers are only a problem if the signature algorithms that are used in transactions are based on the discrete log problem (RSA or ECC) because Shors algorithm makes it computationally feasible to factor large numbers (Shor 1994). Hash functions are not as vulnerable from quantum computers as signatures are because key sizes only have to be doubled.

D. Scaling Blockchain

There are various scaling concerns with blockchain because all blockchain-based cryptocurrencies have low transaction throughput (3-7 per sec for Bitcoin, 20 per sec for Ethereum). The immediate reason for scaling concerns is blocksize limitations. In order to stop forking, limit blockchain size, and not discourage full nodes (too much) blocksize should have a hard cap, but it is unclear what the cap should be. Bitcoin had a cap of 1 mb pre-segwit, which is likely too low. Even if a blockchain sets its blocksize to the unknown maximum, the blockchain would still not be able to handle enough to be usable at world-wide scale (even with Graphene and compact blocks/xthin). Some proposed solutions are the Lightning Network, a 2nd layer off-chain solution (using hash-based time-locked contracts), plasma (for ethereum), sharding, and sidechains (Poon 2015, Back 2014, Buterin Plasma 2017, Buterin Minimum 2017).

The blockchain analyzed later in the paper is similar to Plasma in its use of merkle trees. For this reason, Plasma will be described here. Second layer solutions setup a game through contracts on chain that have an outcome that can be determined with high probability from messages passed off-chain. [1] Plasma is a smart contract on chain that allows a operator to submit blocks in a new chain inside of the ethereum chain. Blocks are only merkle roots of transactions in the simplest case (Buterin Minimum 2017). If someone deposits money on a plasma contract, they can send it to someone else without having the transaction move on-chain. The transaction only moves on chain when a person claims to own X coins by calling exit() on the smart contract. There is a window of time for exits to be contested and if a new UTXO in a more recent block is shown to exist along with the appropriate signatures then the exit fails. Plasma is similar to the blockchain proposed in this paper only in that merkle trees are used to confine validity checking of some information only to the nodes who are interested in consensus of that information.

E. Ripple Consensus Protocol

A variant of the Ripple consensus protocol is analyzed in this paper, which is why this section is necessary. Ripple is a consensus protocol for a closed group of voting nodes that has fast consensus times and claims to have byzantine fault tolerant up 20% faulty nodes (Schwartz 2014). However, Ripple does not in fact have fault tolerance up to 20% because the security of the network also depends on network connectivity. Ripple fails to resolve double spend in cases of extreme network fragmentation (Fig. 2).

F. Merkle Trees

Merkle trees are a data structure that the blockchain uses to keep integrity of state changes in a compact way (Merkle 1980, Massias 1999). Merkle trees are formed bottom up, from leaf to root, through hashing the leaf nodes. Leafs are put into pairs, concatenated, and then hashed. The outputs of the first phase of hashing are put into pairs and then hashed, and this is repeated until there is only one hash called the root. Merkle trees can verify a single leaf node in the tree with O(log2(n)) hashes. If a hash is provided for each leaf node, the merkle root can be rebuild, proving that the leaf is

[1]Author's note: this is paraphrased from a talk by Vitalik Buterin but the source is not known.

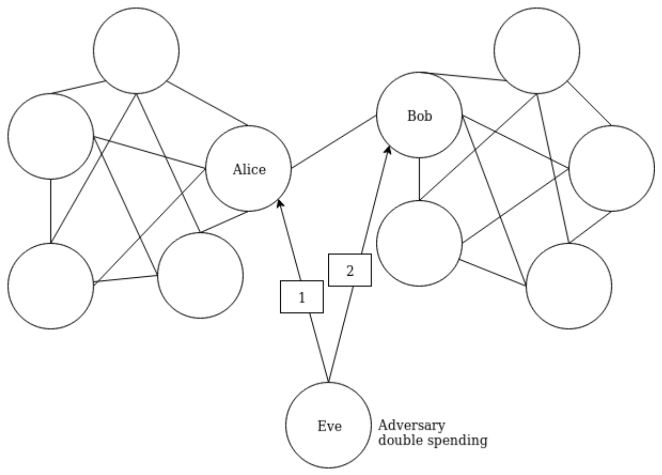


Fig. 2. Eve has 1/2 chance of double spending successfully if there is extreme network fragmentation where <1/5 of Alices UNL is connected to Bobs clique that received transaction 2. Alice can achieve consensus in her clique without Bobs agreement because he makes up <20% of her UNL and the same goes for Bob. This scenario is NOT covered in the Ripple whitepaper unfortunately.

in the tree, because it is computationally impossible to form hash functions collisions.

IV. Correctness: Update Withholding and Looking Glass

A. Definitions

Correctness: A given routing update for prefix X is *correct* if for every AS in the final AS-path the AS-path that was used to create the ASs respective routing update is still the most recent and valid for destination prefix X.

Strong correctness: A chosen routing update is *strong correct* if it is correct and the outcome of sending traffic along the route does not conflict with any policy of every AS in the AS-path.

B. Causes of Incorrectness: Latency and Withholding

If BGPsec and RPKI were fully implemented by every network and on every allocated IP prefix there would not be a guarantee of correctness of routing updates in BGP. The two causes of incorrect data that are undetectable in RPKI and BGPsec are from (1) network latency and (2) withholding routing updates. The severity of incorrectness caused from network latency of BGP network is mitigated from anti-flapping measures and faster networks. Preventing network latency incorrectness entirely can never be accomplished because of physics.

The second type of incorrectness, withholding routing updates, is the type of incorrectness focused on in this paper. There are two reasons for withholding routing updates, although the list is probably not exhaustive. The first reason is from misconfigurations. A router may not have the appropriate input or output policy configured correctly for a specific IP prefix [Wang, p.1-22009; Feamster , cite Mahajan]. The second reason is malicious transit providers. The diagram below shows both scenarios.[2]

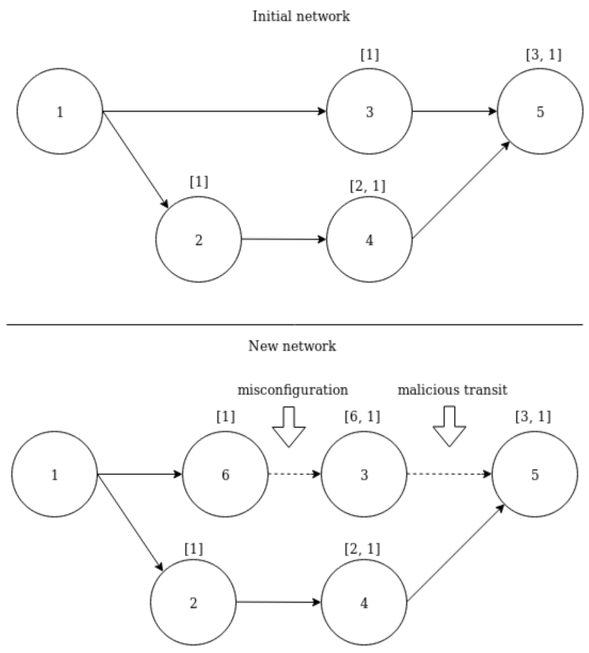


Fig. 3. The dotted lines show a possibility of not sending an update. The misconfiguration and malicious transit are on the same diagram because the effects are similar enough to each other in this simple scenario.

There is a stronger view of correctness would also be beneficial if it could be reached as it would by definition prevent unexpected traffic flow scenarios. If there was a way to knowledge about selective of route advertisements, unexpected traffic flows could be more easily avoided [cite rfc 7789] and strong correctness achieved.

C. Current Solutions: Looking Glass (LG) and Internet Routing Registries (IRR)

Network operators have a limited view of the BGP network which is by design to reduce unnecessary routing updates to prefixes that are not obtainable and reduce the complexity of path selection and local policy management. Looking Glass servers and Internet Routing Registries have been used to give network operators difference perspectives of the network that they could see from an AS they controlled. LG and IRR help with detecting misconfigurations [cite Bruno].

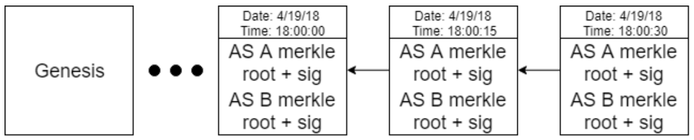
V. Blockchain Merkle Border Gateway Protocol (BM-BGP)

Definitions Routing root: the root of the indexed merkle tree of routing updates

Correctness of routing updates can be achieved with consensus of routing tables. BM-BGP provides consensus of routing tables in a way that preserves privacy and is more scalable than creating a global consensus of routing tables.

[2]Author's note: I have not been able to find malicious transit providers in the literature but it is theoretically possible.

Each AS sends a merkle verification alongside a routing update that builds the most recent root in the blockchain for that AS. Like BGPsec signatures, the merkle hash lists for each AS are passed on in each new downstream update so that every AS has all the previous hash lists to rebuild all of the merkle trees for all the ASes in the AS-path. Although merkle trees reduce on-chain data substantially, they are not scalable enough to make BM-BGP feasible because old merkle proofs have to be retained in every routing root update (see Consensus Protocol section). Fig. 4. shows the blockchain with merkle roots.



In this setup, routes can be checked as correct by an AS by running an isCorrect() function that iterates through each route, grabs the routing root for the AS in the blockchain in the most recent block (unless local time is not yet greater than the commitRoutingRootAtTime parameter that is in the update), and use the hashes to build the merkle root. If the root does not match the hash path, there is a grace period where the route is used even though correctness is not proven until there has been a substantial wait since commitRoutingRootAtTime to make the AS very confident that the routing root commit would have been part of the blockchain by now if it had been in fact submitted on time.

A. *Integration with BGPsec*

MBGP integrates well with BGPsec because the path that is added to the blockchain does not include AS prepending. The path that is put on the blockchain is only the path created from each secure-path segment of a BGPsec update ,not including prepending count (pCount).

VI. Indexed Merkle Tree

Each AS has the ability track two routing roots on the blockchain, one sequence of routing roots for IPv4 and the other sequence for IPv6. In order for merkle trees to be effective for providing correctness to routing updates they most have the following property.

Correctness requirement: the hashes that verify that a route to prefix X is in the tree must also show that the route is in a subtree of all other routes to the same prefix and the subtree is in the correct position in the main tree.

Routes (or groups of routes to the same prefix) must have a single allowed and anticipated position in the tree. Otherwise, an AS can put more than one route to a specific destination in the tree, hiding one of the routes to a subset of peers. If there is allowed to be more than one route to a specific destination in the tree, a node receiving a routing update is not as confident that the route it receives is the most recent route to the destination. The routing hashes must convey information that a route is in the tree and in a specific position in the tree without revealing any meaningful routing information because other meaningful routing information might want to be kept private by the AS.

The tree follows the integrity property because the tree is indexed by IP addresses. Left branches are an additional zero from the parent node and right branches are an additional one from the parent node. For example, the position of a route to IP prefix 160.0.0.0/3 would be right branch (for first bit), left branch (for second bit), right branch (for third bit). If there are routes to destinations that are of a greater prefix, say 160.0.0.0/17, then 160.0.0.0/3 becomes a middle node. There are three possible children from a parent: left, middle, right. Middle is used when there are routes underneath the position of the route in the tree. Middle nodes are needed in order to keep the integrity of the tree so that the integrity property is followed. There is no other way to guarantee that there is not more than one route to 160.0.0.0/3 in a tree that also has 160.0.0.0/17 without adding a middle node for 160.0.0.0/3. Therefore, the routing tree differs from a normal merkle tree not only in that the routing tree is indexed but also because the routing tree needs to allow for leaf nodes below any parent of the tree. A. Tree creation algorithm The algorithm for forming the most efficient tree is three steps: create an inefficient tree, condense the tree, recursively hash nodes to form root. To create an inefficient tree, first you take the list of routes in the routing table and add them to the tree. Routes are put into an array and recursively added to the tree based on the binary prefix of each route. The recursive function searches through each route in the array and calls the function again with a new list of routes that have a specific bit sequence in the prefix. This process continues, the bit sequence gets longer and more and more routes get filtered out. A route is added to the tree when the bit sequence matches the route prefix exactly. If a node is added tree but there are still nodes left that get added under it, that node turns into a middle node.

The first step creates an inefficient tree that can be condensed. The condense phase has a couple steps. First, left and right nodes in the tree are brought up the tree if they do not have any neighboring middle, left, or right nodes. Left and right nodes that are brought up the tree replace their parent, which means that they become the children of their parents parent. Middle nodes cannot be brought up the tree without instructions because then integrity is lost. If a middle is to be brought up the tree, there has to be an instruction that notifies the verifier that a specific branch skips to a certain bit sequence. These instructions must be part of the tree in order to keep integrity. If there are any instructions, they are stored in the middle child of the root node.

Now that there is an efficient tree the merkle root can be created. The tree is recursively hashed from bottom up. First, we must must make sure that every middle children has left and right siblings that are not None. Siblings are the other children of ones parent. For each sibling slot that is none (for a middle node), we add a node with a data string LEFT or RIGHT because otherwise, middle node hashes could not be distinguished from left hashes if there are only a right and middle child. Now the merkle root can be created with

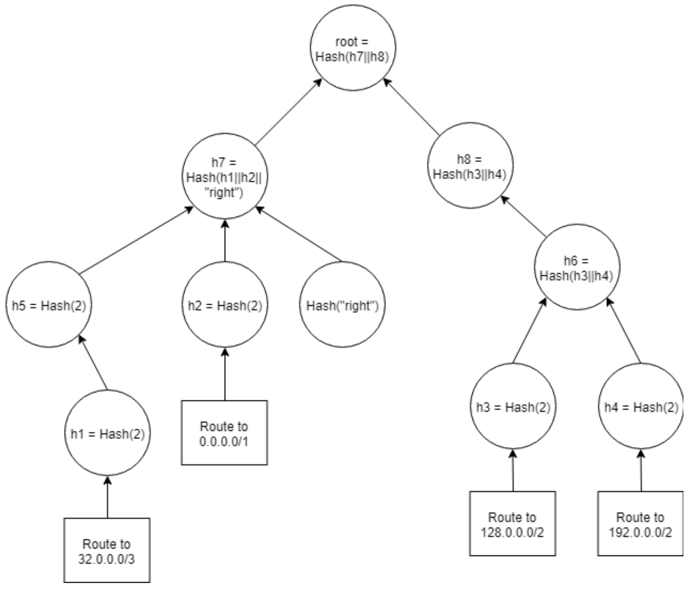


Fig. 4. This tree can be condensed by moving Route 32.0.0.0/3 up to be a left sibling of h2. Routes 128.0.0.0/2 and 192.0.0.0/2 can be moved up with instructions to skip the 2nd after if the first bit is a 1.

integrity of position of the leaf nodes. We start at the root node, find the roots hash by hashing each of its children, which we get by hashing their children, and so on until a leaf node is reached (base case of recursive function) and then we backtrack.

A. *Verification algorithm*

Receivers of BGP updates need to verify that a route is in the correct position in the tree. The data structure that proves that the route is in the tree is indexed with the prefix destination, just like the tree. Here is an example of a the data structure using python lists: [0.../1hash[101../3hash, 11111.../5hash]] In this example, to verify route with destination 101.../3 we move through this data structure bit by bit. We index the outer list with a 1 because 101.../3 starts with a 1. If list[1] is also a list, we continue, if not we check if the hash is the same as the hash we are searching for, and if not, we fail the verification. If it is a list, as it is in this case, we check the prefix length N of the destination we are verifying and if we are already went into n lists, then we fail, otherwise, we continue into the next list. The next list is [101../3 hash, 11111.../5 hash], both there are more lists so we index them element at the 0 position. If the hash that is found matches the route we are trying to verify, we stop, concatenate the hashes in that innermost list, hash them, and then backtrack. We backtrack by replacing each list by the hash of its inner elements and eventually we get the merkle root. The generated root = hash(0.../1 hash —— hash(101.../3 —— 11111.../5 hash)). If the generated root is the same as the one in the blockchain then we know that the route is the valid route. Notice how throughout the process we made sure that the route was in the correct position in the tree.

VII. Consensus Protocol

The core problem behind route omission is that ASes end up with inaccurate updates for a destination, which may lead to suboptimal best path decisions. If all ASes commit publically to a specific table, then route omission cannot happen because there is only one possible path that is timestamped for every subset of the path. However, committing publically to a routing table is not scalable or desired by ASes. The amount of data of every update of every AS is substantial, and it is obvious that a consensus protocol could not handle this much data. Committing every update is also not desirable because many ASes do not want anyone to be able to see their routing table. As addressed in the previous section, indexed merkle trees are used for compact integrity which hide routing information from non-peers of an AS.

Before describing the exact consensus protocol, there is another reason to use a consensus protocol outside of ensuring current routes: verified routing update audit trail. If routing tables are timestamped, an AS can choose to save verification data structures and therefore have a clear provable timeline of a ASes and current routes. This may be helpful bargaining tool. Although not clear at first glance, a merkle trees may lead to more trusted data sharing of past events. Different ASes might choose to save different old routing updates for recording purposes. If an AS is missing a specific update that they might want and they can find another AS to send them the old update, the AS requesting the update can prove that the update was current at the specific time because the merkle verification builds the same routing root that is in the expected part of the blockchain.

Blockchain without 2nd layer is not scalable because consensus–the process of minimizing the probability of forks and maximizing the probability of state change immutability–is achieved by setting a sufficiently high block creation interval through mining and a cap on valid block size. Traditional blockchains need to achieve agreement/consensus with the free entry and exit of nodes, and there is no hierarchy or identity of nodes where nodes may have special permissions–this is part of what we mean by decentralization. Since the cost of creating a node is so low in traditional blockchains, it is very hard to achieve consensus, and a mining mechanism must exploit a scarce resource in order to bring scarcity to valid block producers.

The node environment for BGP is substantially different than completely decentralized applications. If consensus is restricted to nodes that have IP space allocated them, then a more efficient consensus algorithm can be chosen. A variant of the Ripple consensus protocol can probably be used instead of POW or POS. The Ripple consensus protocol is much more scalable than POW and POS because voting rounds are much more frequent. Voting can only be done by nodes that are on accepted node lists. The node lists for a BGP blockchain would be managed by the Regional Internet Registries. Only public keys that have an up to date certificate attaching them to IP space are allowed to participate in consensus. ASN could be used instead of IPv4 allocations. For Ripple Consensus to work there would need to be sufficient network connectivity, otherwise Ripple is vulnerable.

A. *Multi-block Merkle Verification*

A critical problem with the indexed Merkle tree construction is that hashes have to build up to the merkle tree for every route when new routing roots are committed. One way to do this is to attach hashes of all of the leaf nodes in the tree to every update–however this defeats one of the main purposes of using a merkle tree in the first place. This critical problem is one that I have not solved yet and may be fatal to the project.

B. *ICANN and RIRs as trusted orderers*

Instead of a consensus algorithm ordering routing routes, ICANN and RIRs can order routing routes because they are trusted entities. It is unlikely that these organizations would take on this role, but in a hypothetical world where they ordered routing roots, route omission and selective lying could be solved in the same way a consensus protocol solves route omission and selective lying. Centralizing ordering of routing roots has a scaling advantage because a centralized algorithm can order transactions much more quickly than any consensus protocol. The algorithm that the centralized entity runs could be as simple as choose the first valid and received routing root, which would solve any data conflicts where there are more than one new routing root for an AS.

Another advantage of a centralized trusted authority is that many ASes already have connections with ICANN and a RIR to some degree because of RPKI. It is probably easier to convince ASes to add a connection to their RIR than join a p2p network. Further, a central authority can fix a bug faster than a consensus protocol, although it may take longer to find the bug there are less parties running the software than in a consensus protocol. The main disadvantage to a central authority is that there is single weak point. If the trusted authoritys servers fail, there could be significant outages. This weakness can be mitigated through distributed and fault-tolerant computing. It is not clear whether an attack on an authority's servers is easier than the ripple consensus protocol (described in previous section) because ripple has a ¿20% attack.

VIII. Further Research

1. Search Looking Glass servers for potential malicious transit providers
2. Test BH-BGP alongside a simulation BGP network.
3. Compare effectiveness of Looking Glass with BH-BGP
4. Explore zk-snarks to help solve the multi-block merkle verification problem
5. Apply BH-BGP to Bitcoin Lightning Network protocols

References

[1] Back, A., et. al. (2014). Enabling blockchain innovations with pegged sidechains. https://blockstream.com/sidechains.pdf

[2] BIP 148. github.com/bitcoin/bips/blob/master/bip-0148.mediawiki

[3] Bruno., L., et. al. (2014) Through the Looking-Glass, and What Eve Found There. http://s3.eurecom.fr/docs/woot14_bruno.pdf

[4] Buterin. V (2017). Hard forks, soft forks, defaults and coercion https://vitalik.ca/general/2017/03/14/forks_and_markets.html

[5] Buterin., V. (2017) Minimum viable plasma. ethresear.ch/t/minimal-viable-plasma/426

[6] Buterin, V., Poon, Joseph. (2017). Plasma: Scalable Autonomous Smart Contracts https://plasma.io/plasma.pdf

[7] Butler, K., Farley, T., McDaniel, P., & Rexford, J. (2008). A Survey of BGP Security Issues and Solutions. 1-37. https://www.cs.princeton.edu/ jrex/papers/bgp-security08.pdf

[8] Eyal, I., & Sirer, E. G. (2013). Majority is not enough. Retrieved from http://arxiv.org/abs/1311.0243

[9] Feamster, N., Balakrishnan, H. (2005) Detecting BGP Configuration Faults with Static Analysis. https://www.usenix.org/legacy/event/nsdi05/tech/feamster/feamster.pdf

[10] Hari, A., & Lakshman, T. V. (2016). The Internet Blockchain. Proceedings of the 15th ACM Workshop on Hot Topics in Networks - HotNets 16. Retrieved from http://web.kaust.edu.sa/Faculty/MarcoCanini/classes/CS390G/S17/papers/InternetBlockchain.pdf

[11] Heilman E., Kendler A., Zohar A., and Goldberg S. (2015). Eclipse attacks on bitcoins peer-to-peer network. In USENIX Security Symposium. 129144

[12] Poon, J., Dryja, T. (2015) The bitcoin lightning network: Scalable off-chain instant payments draft 0.5.9.1. https://lightning.network/lightning-network-paper.pdf [12] Lamport, L.; Shostak, R.; Pease, M. (1982). "The Byzantine Generals Problem" (PDF). ACM Transactions on Programming Languages and Systems.

[13] Mahajan, R., et. al. (2002) Understanding BGP Misconfiguration. https://www.microsoft.com/en-us/research/wp-content/uploads/2017/01/Understanding-BGP-Misconfiguration.pdf

[14] Merkle, R. (1980). Protocols for public key cryptosystems. Symposium on Security and Privacy. IEEE Computer Society. 122-133.

[15] Massias, H., Avila, X., Quisquater, J., (1999). "Design of a secure timestamping service with minimal trust requirements," In 20th Symposium on Information Theory in the Benelux.

[16] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. 1-9. Retrieved from https://bitcoin.org/bitcoin.pdf

[17] NIST. RPKI Deployment Monitor. (2018). Retrieved from https://rpki-monitor.antd.nist.gov/

[18] Rekhter, Y., Li, T. (1995) RFC 1771 A border gateway protocol 4 (BGP-4).

[19] Rekhter, Y., Li, T. Hares, S. (2006) RFC4271 A Border Gateway Protocol 4 (BGP-4).

[20] RFC7789 Impact of BGP Filtering on Inter-Domain Routing Policies

[21] RFC8205 BGPsec Protocol Specification. M. Lepinski, Ed., K. Sriram, Ed.. September 2017. (Format: TXT=115900 bytes) (Updated by RFC8206) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC8

[22] RFC8210 The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1. R. Bush, R. Austein. September 2017. (Format: TXT=78467 bytes) (Updates RFC6810) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC8210)

[23] Schwartz, D., Youngs, N., & Britto, A. (2014). The Ripple Protocol Consensus Algorithm. 1-8. Retrieved from https://ripple.com/files/ripple consensus whitepaper.pdf

[24] Shor. P. (1994) Algorithms for quantum computation: Discrete logarithms and factoring

[25] Toonk, A. (2017, December 12). Popular Destinations rerouted to Russia. Retrieved from https://bgpmon.net/popular-destinations-rerouted-to-russia/

[26] UASF Working Group. (2017). https://www.uasf.co/

[27] Wang, Y., et. al. (2009). Neighbor-Specific BGP: More Flexible Routing Policies While Improving Global Stability. https://arxiv.org/pdf/0906.3846.pdf

[28] Youtube Hijacking: A RIPE NCC RIS case study. (March 2008). ripe.net/internet-coordination/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study.