Gradient: A Fully Distributed, Anonymous Proxy Network Incentivized Through Bandwidth Mining

David L. Salamon with Brian J. Fox, Jay Freeman, Gustav Simonsson, Stephen F. Bell and Dr. Steven

September 19, 2017

Abstract

As methods for discovering private user information from browsing data have improved, interest in anonymization methods has increased. Unfortunately, existing anonymity systems such as I2P and Tor suffer from a tragedy of the commons – only a few thousand unpaid volunteers host proxies and exit nodes, allowing sophisticated attackers a tractable number of nodes to monitor or otherwise compromise. We present a market based, fully decentralized, anonymous peer-to-peer system based on "bandwidth mining" which we hope will address this lack of relays paying them.

Original contributions include an Etheruum-based implementation of stochastic micropayments suited for the sale of bandwidth, a Kademlia-inspired connection scheme for deterministic routing and auditing of seller behavior in a fully distributed bandwidth market, and preliminary application of payment mechanisms to hinder Sybil, Eclipse, DoS, and Inference Attacks.

Contents

1. High-Level Overview

1.1. System Goals

The Gradient Network was designed with two goals in mind:

1. To codify the right to unrestricted, private internet access on a networking level.
2. To build a system suited for daily use. To say that a freedom is not a part of daily life is to say that the freedom is dying.

These are ambitious goals, and are not realized by the system described in this document. This document does, however, contain an imperfect first attempt – an invitation to join us in design, construction, and use of what we believe is the future of networking.

1.2. Document Structure

This document begins with an overview of the system (Section 1), describes the kind of attacks and attackers that can be reasonably anticipated against such a system (Section 2), and then discusses lessons which can be learned from the previous deployment of similar systems (Section 3). We include these sections in the hope that they might help jog the readers memory, perhaps alert them to interesting references, and help the reader build intuition for the context in which the system will be deployed. These sections are to help you get up to speed.

Once the stage has been set, we will move on to a detailed discussion of the system components themselves. We will detail the external libraries we have chosen to rely on, and why (Section 4), describe a micropayment system suited to the sale of bandwidth (Section 6), a commodity specification for the sale of bandwidth (Section 7), a distributed marketplace for the sale of bandwidth (Section 8), and a method of employing our commodity specification for bandwidth to build an uncensorable and anonymously internet connection (Section 8). This is to say these sections provide a "nuts and bolts" view of the Gradient network. Each component uses the previous as building blocks, so it is recommended that you read these sections in order.

We close out with a discussion of what an attackers experience of the system might be (Section 9), both as a means of discussing the security of the system when viewed as a whole, and as a thought experiments for those wishing to assist us in improving system security. Next we discuss non-security features we hope to add in future releases (Section 12). Readers interested in assisting with the project are encouraged to read these sections carefully.

Because the system is to be fully decentralized, fully autonomous, and fully anonymous, much of this design document is centered on attacks. Although attack analysis is important, and will take up much of our time, it is ultimately no more than a necessary conceit to the context in which the market is to operate; please do not let thoughts of security distract you unduly from an understanding the system itself. We look forward to conversing about the system with experts in economics, autonomous markets, network engineers, applications programmers, and anyone with interest in these ideas.

1.3. Terms

- Node. A computer running a program which implements the Gradient Protocol.
- Medallion. Data used by a node to demonstrate it is in exclusive possession of computational resources.
- User. The owner of a node.
- Agora. A collection of nodes interacting through the Gradient Distributed Marketplace Protocol.
- *The* Agora. The Agora in possesion of the most global computational power.
- Econ. A node who is a member of an Agora's collection.
- Entry Econ. An econ who is willing to accept direct TCP connections from Users.

- Address. The location in Medallion-space inhabited by an Econ.
- Customer. A buyer on the Agora.
- Relay. A node willing to sell bandwidth to customers.
- Proxy. A relay which is additionally willing to interact with web servers on behalf of customers.
- Manifold. A chain of relays used for anonymous communication.
- Ticket. A stochastic micropayment.
- Attack. A method for causing the non-consensual transfer of money or information.
- Attacker. A person or group of people interested in performing attacks.

1.4. Core Security Assumptions

The security of the Gradient system as a whole ultimately rests on basic assumptions about the world. In the event that either assumption is ever falsified, the system will not function as intended.

1. No single Attacker will control the majority of global computation power.
2. The majority of global computation power will not cooperate on an attack.

Without these assumptions (or a similar ones for a different resource), it would be impossible for a fully anonymous and distributed system to function. No distributed quorum could be trusted, because the majority of users might vote in such a way as to facilitate attacks. Users would be left relying on web-of-trust approaches, which are by their very nature non-anonymous.

We use assumptions 1 and 2, combined with a method for rapidly demonstrating significant computational work has been done to construct the basis for trust on our system: medallions. Production of a medallion ties ownership of computation to a public key. They are presented as needed to prove that a given node is "real."

Readers who are familiar with other distributed market based networks will recognize these assumptions as forming the basis for proof-of-work systems (bitcoin, etc), and may be inclined to ask: why not use proof-of-stake, proof-of-idle, or other less energetically wasteful methods for proving "realness"?

Proof-of-stake rests on the assumption that no attacker will ever control the majority of tokens. As our attack model includes oppressive governments, this can not be counted on. Even Bitcoin's astonishing market capitalization is far less than the GDP of a modestly sized country. Making matters more complicated, in the near future we intend to extend the system to support anonymous payments, which will make detection of such a "hostile takeover" much more difficult. In short: we did not use proof-of-stake because we did not want to engineer a system in which our users' right to privacy might be sold to the highest bidder.

Proof-of-space is another interesting approach. Users in proof-of-space systems demonstrate their "realness" by saving data to their disk, and periodically required to answer queries about this data to demonstrate continued possession. The issue with its application to Gradient are two-fold: (1) the proof of work variant we use can be thought of as "proof of key" – ie it verifies that a given key is in possession of computational power. It is not obvious how to apply proof-of-space to this use case, and (2) most home users do not have plentiful disk space, while most data centers do. This means that by selecting proof-of-space, we would be ensuring that datacenter based nodes would be over-represented, due to decreased Econ hosting by home users.

Proof-of-idle rests on the additional assumption that periodic, synchronized proof-of-work is sufficient to demonstrate a User's share of the global computational power. Unfortunately, while the network is in its infancy ($\leq$ 10 million Econs), this leads to a situation where any company in control of a supercomputing center may, with only the sacrifice of 1% of their computational power, take control of the network. As we expect it to be quite a while before we have sufficient numbers of econs for this attack to cease being devastating, we are not using proof-of-idle for this release.

1.5. High-Level System Overview

1.5.1. The Agora

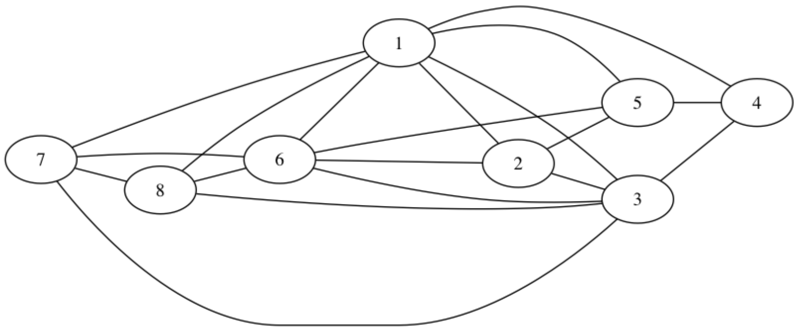

Figure 1: An Agora with eight Econs

The Agora's purpose is to serve as a market place for the buyers and sellers of anonymous bandwidth. Although it is tempting to believe such a market should take the form of a classic order book (a list of currently unfilled bids and asks at different prices), a very important property of bandwidth to be used for anonymity purposes is that purchase decisions are made in a way that places severe limits on what influence can be exerted by an attacker. For this reason, rather than provide an order book, the Agora provides a way to traverse the space of currently valid medallions. Once a location in medallion-space has been selected, a miniature pseudo-auction is held. For example, if the econ at position 2 in the figure wished to buy bandwidth, it would select the median of 1's, 3's, 5's and 6's price, rounded down. For users not seeking anonymity, the lowest price may be selected instead.

1.5.2. Manifolds

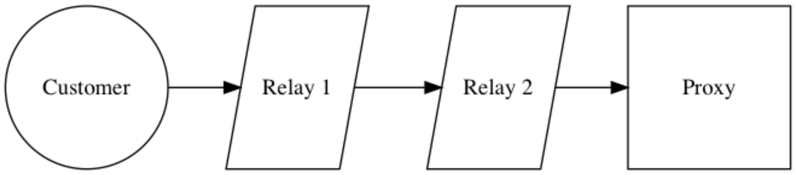

Figure 2: A three-Econ Manifold routing traffic for a Customer

Once relay and proxy bandwidth has been purchased, the customer configures them into a connected chain termed a manifold. In the above picture, Relay 1 receives bandwidth from the Customer, performs cryptographic operations on it, and forwards it to Relay 2. Relay 2 performs a similar role with respect to Relay 1 and the Proxy. The Proxy then sends this data to a web server. Any data returned by the web server is sent back to the Customer along another manifold, perhaps implemented using the same nodes in reverse.

1.5.3. Web Browsing

The majority of initial usage of the Gradient Network will likely happen in support of uncensorable, anonymous web browsing. In this use case, the client software will automatically use the Agora to create a manifold, and provide additional checks to verify that SSL/TLS is being used in a way suited to this networking model. Unless two or more nodes in a manifold are working together (perhaps because they are run by the same user), no single node knows the Customer's IP Address and the server they communicated with.

1.6. User Stories

1.6.1. Bandwidth Miners

Users with excess computational power and bandwidth may choose to support the Gradient project, as well as earn money, by running an econ on their computer. To do so, they install and run the Gradient software package and supply a payment address. To verify realness while preserving anonymity, the software will convert computational resources into a medallion. Econs are compensated for their lost computation by bandwidth customers.

1.6.2. Resisting Oppressive Governments

Users who live under oppressive regimes (where, for example, it is illegal to access Wikipedia), can use the system to hide information about the websites they are browsing. To do so they install and run the Gradient software, provide an initial payment to their Gradient wallet, and browse the web as normal. Unlike similar services[Section 3], the network pays hosts for their participation. We hope this will lead to greater participation, which in turn will lead to grater anonymity[Section 8].

1.6.3. Resisting Oppressive Web Applications

Users who have subscribed to oppressive web applications (sites who change the services offered to their subscribers based on geographic location), can use the system to appear to be browsing from the country of their choice. Unlike similar services[Section 3], traffic will not come from IP ranges allocated to server farms, and so will be much harder for oppressive websites to block. Additionally, because bandwidth is incentivized, it will likely prove much more plentiful on the Gradient Network, i.e. making the real-time playback of high resolution video viable.

2. Attacks

2.1. Bug Bounty / White Hat Policy

TODO: we should offer a "hall of fame", a cash/token reward, and maybe a t-shirt or coffee mug or something?

Note that token rewards are very interesting, since we could set aside a fixed amount and pay out x% of that amount for each vulnerability which is found.. it would likely become the largest bug bounty program in history. If the issue is that putting those tokens aside might be difficult at this point, I also think that incentivizing vulnerability disclosure this a very compelling reason to inflate the currency.

2.2. Attacker Goals

2.2.1. Information Gathering

The largest class of attacks against which the Gradient Protocol must defend against are those which reveal information about its users. Because Gradient is implemented as an overlay on the existing internet, some information will be unavoidably shared with some peers. In the list below, such information is marked with a "*". Any information which is not specifically listed as unavoidably shared in this document, but for which a method is discovered to uncover that information is termed an *informational attack* and is covered by Gradient's White Hat Bug Bounty. For more information on what is shared, see the protocol specification in sections ? and ? and our reference implementation of the network.

Types of data which are assumed to be of interest to an attacker (timeless):

- Real-World Identity Information. A user's given name, SSN, address, etc.
- Website Account Information. The user accounts at a specific website. Note this can be different from Real-World Identity Information.

- *IP Information. The IP address from which a user is accessing the Gradient Network. Note that in some usage scenarios this may be equivalent to learning Real-World Identity Information.
- *Ethereum Information. The keys associated with a user's wallet (*public or private). Note that in some usage scenarios this may be equivalent to learning Real-World Identity Information.
- *Gradient Network Information. The keys associated with a node's cur- rent business on the Gradient network (*public or private).

Types of Behavioral information which are assumed to be of interest to an attacker (time and manifold associated data):

- *Customer Identification. The attacker learns the IP address of a customer.
- *Relay Identification. The attacker learns the IP address of a relay.
- *Proxy Identification. The attacker learns the IP address of a proxy.
- *Link Identification. The attacker learns that two IP addresses where employed in a manifold.
- *Website Access. The attacker learns that an outbound connection was made from the Gradient network to a specific website.
- *Webserver Access. The attacker learns that an outbound connection was made from the Gradient network to a specific webserver (which may host multiple websites).
- *Ethereum Linking. The attacker learns that an Ethereum public key is held by a Gradient user.
- *Purchase Linking. The attacker learns that two transactions share a common payer.
- *Purchase Quantity. The attacker learns the amount of bandwidth sent over a Manifold.
- *Purchase Timing. The attacker learns the temporal spacing of bandwidth over a Manifold.

Although all of the above behavioral information is shared with other nodes on the Gradient Network during normal operation, as described below, in most contexts it is assumed that users will only be directly harmed by Behavioral Information Gathering if the attacker can learn several pieces of information at once (for example, to say that user X accessed website Y the attacker would need: buyer identification, website access information and several pieces of link identification). It is for this reason that peers following the reference spec do not store or share any of the above information, except as required to provide the services a customer has purchased.

2.3. QoS Attacks

Some adversaries may be satisfied simply slowing down the system performance of Gradient network users generally, thereby potentially deminishing usage.

2.4. Economic Attacks

Unlike similar systems, Gradient must also concern itself with attacks on payment mechanisms. The taxonomy used in this paper is:

1. Economic Exploits. Profitable undesirable behavior. (Example: if a user gives out "free sample" bandwidth, some users might exclusively use free sample bandwidth).
2. Economic Denial of Service (EDoS). The ability to pay to overwhelm another node on the Gradient Network with purchases, thereby taking them offline.

2.5. Man-in-the-Middle Attacks

Actions that can be performed only after inserting oneself between two interacting parties are collectively referred to as man-in-the-middle attacks. Encrypted information may be logged for analysis of metadata [Section ?], non-encrypted data may additionally be changed to control behavior.

2.6. Sybil Attacks

Malicious actions, performed by pretending to be multiple users, are termed Sybil attacks (after a patient suffering from multiple personality disorder.) Applications of this type of attack include:

- Submitting multiple reviews to Yelp, Amazon, etc.
- Achieving faster downloads on BitTorrent by pretending to be multiple leachers[?].

2.7. Eclipse Attacks

In an Eclipse Attack the Attackers goal is to hide part of a P2P network from itself. The methods employed are generally the network equivalent of privilege escalation attacks: gain control of network positions which have more control of the network, then use that control to acquire more control.

- Segmenting the Bitcoin mining P2P network, allowing for so-called “51% attacks” when the attacker controls substantially less than 51% of the compute power[?].
- Removing a file from the BitTorrent DHT by taking over the address space associated with its magnet link[?].

2.8. Inference Attacks

Deanonymization which stems from a statistical modeling of system behavior are termed Inference Attacks or Monitoring Attacks. These can be especially effective when combined with “probing moves” such as carefully crafted or timed requests, or other attacks such as DoSing a specific peer off of the network and observing how traffic patterns respond.

- Inferring medical illnesses, family income, and investment choices of end- users from SSL encrypted web traffic[?].
- Deanonymizing Tor traffic from global traffic logs[?].
- Learning the private key of an OpenSSL server through timing analysis[?].

2.9. Denial of Service Attacks

Attacks centered around taking a specific resource offline are termed Denial of Service Attacks. Because system behavior during “unexpected” circumstances is often poorly specified and tested, perhaps surprisingly, DoS attacks are quite useful for deanonymizing nodes in P2P networks. Notable examples:

- Targeted DoS attacks used in concert with sybil-attack based monitoring to deanonymize Tor traffic[?].
- DoS off lining for complete control of I2P’s floodfill database, requiring only 20 sybil nodes, thereby deanonyimizing all traffic on the network[?].

2.10. Hacking

Particularly motivated attackers might directly compromise nodes on the network, converting historically trustworthy peers into attack vectors. When bandwidth is deployed using Manifolds, such hacking may potentially be performed iterative, allowing an attacker to eventually backtrace a connection. Such attacks are out of scope for the Gradient system, but do have important security implications. If the Gradient Network’s security achieves its goals, this will be the main attack against the system.

3. Existing Approaches

3.1. Unprotected Internet Access

One approach to anonymous internet access is to ignore connection level security concerns and perform the actions of interest from a suitably anonymous location. Examples of this include the digital equivalent of so-called "dead drops" – for example users might edit the Wikipedia entry for African Elephants with information on a specific group. This approach works, but requires a significant amount of coordination to accomplish. One might be tempted to ask: what system might be used to perform that coordination?

For this reason we will avoid analyzing such systems in this paper. However, any users of Gradient with serious security needs – especially those users who suspect they are already under active surveillance – are encouraged to carefully select which network will carry their traffic. Traffic shaping, etc can only take you so far.

3.2. Virtual Private Networking (VPN) Services

VPNs create encrypted connections between an end user and the VPN server. If you are worried about SSL Stripping[1], or are wanting to hide the country of origin for your web requests, this can be an effective method. Unfortunately, VPN services are a very poor approach for those users seeking anonymity – the VPN provider has complete payment information for the subscriber in question. If you expect that an adversary might be capable of wiretaping a VPN, or otherwise force VPN cooperation, VPN providers are therefore a poor choice.

3.3. Tor

Tor is a free software project famous for introducing the idea of Onion Routing. Onion Routing is a scheme wherein messages from A are passed to B, messages from B are passed to C, messages from C are passed to D, and so on. To prevent eavesdropping encryption is used as follows:

$$M_d = encrypt(M_{plain}, K_d)$$

$$M_c = encrypt(M_d, K_c)$$

$$M_b = encrypt(M_c, K_b)$$

Where Mx is the message passed to peer X, and Ky is peer y's public key.

3.4. Tor with Incentives

3.5. I2P

I2P is billed as a "next generation" onion router.

3.6. Mixmaster, Mixminion, and other Chaumian relay schemes

These are cool!

4. Gradient's Building Blocks

Gradient's functionality is built on several important primitives. As some readers may not be familiar with these primitives, or be familiar with the properties specifically used in Gradient Network, we briefly summarize them here.

4.1. WebRTC

For connections between nodes on the system, WebRTC was selected.

- Can't make a direct connection, requires 3rd party signaling.
- NAT traversal.
- full node-to-node encryption.
- Can be MitM attacked, which we prevent elsewhere.

4.2. NACL

NACL is a cryptography library release by DJB. It's good!

4.3. Ethereum

Ethereum is a blockchain based cryptocurrency. Unlike previous efforts (Bitcoin, etc), Ethereum has smart contracts. Also we use the current Ethereum Block as a clock all over the place.

5. Medallions

Medallions form the bridge between our core security assumptions and the network as a whole. To produce a medallion, a peer takes a public key K, and the previous Ethereum block hash E, then performs a large number of computations to locate a salt S such that $H(K, E, S) \geq N$, where N is some difficulty scaling factor. When a new Ethereum block is added to the chain, a new S must be calculated to keep the Medallion current.

6. Tickets

Paying for bandwidth presents a rather unique set of challenges. In most other payment systems, the cost of an item is substantially greater than the cost of sending a packet, and so the networking cost may be safely ignored as just another transaction cost. In the Gradient Network however, the cost of a packet is the *price being paid*, and so even if the transaction costs for sending payment are as low as a single packet, they would be equal in cost to the purchased item. Making matters more troublesome, we will soon see that the transaction costs also include paying miners on the Ethereum block chain.

6.1. How Much Will A Packet Cost?

For the purposes of this discussion, let us assume that a packet is 1e3 bytes in length. To calculate an upper bound, we observe that the most expensive bandwidth known to mankind is AWS's Singapore CloudFront bandwidth, at \$0.14 per 1e9 bytes. This yields a per-packet cost of 1.4e-5 cents (\$0.00000014). Because bandwidth is a wasting good (any unsold bandwidth is lost forever), the actual price is likely to be significantly lower than this upper bound.

6.2. Ethereum Transaction Costs

Because the Gradient System uses Ethereum as its payment processor, we might also be curious about what the expected cost of sending money in the system is. Although the transaction costs may vary according to [Gustav please save me]. A reasonable estimate of likely transaction costs are \$0.20 for urgent type transactions (happening on the order of 20 seconds), down to \$0.01 for non-urgent transactions (happening on the order of 20 minutes).

6.3. Building Micropayments from Macropayments

With transaction costs now discussed, and looming large, let us now look at what methods exist for controlling them.

One potentially interesting approach, which was employed in MojoNation[], is to have a ”balance of trade” between each pair of nodes. As bandwidth flows between them, they periodically settle up when the balance gets too far from zero. However, as we have seen, the transaction costs of settling up using Ethereum as a payment processor in this scheme would result in at minimum a $0.01 transaction fee. Using our upper bound, we can see this price is around 100 megabytes of bandwidth. A secondary issue with this approach, is that peers nearing the reconciliation threshold would know that fact, and be tempted to simply disconnect and create a new identity rather than pay the fee. For these reasons, is tempting to believe that there exists no deterministic payment scheme suited to fully anonymous bandwidth sales using Ethereum as the payment processor. We therefore turn our attention to stochastic payments.

To explore stochastic payments, let us consider a lottery ticket (hereafter simply *ticket*) which has a 1e-5 chance of being worth $1.40. How much is it worth in expectation? The answer is 1.4e-5 cents, exactly the upper bound we established for the cost of a packet. In the event that the lottery ticket is a winner, cashing in on it with urgency will cost around 14% ($\frac{\$0.20}{\$1.4}$) in transaction costs, and with non-urgency around 0.7% ($\frac{\$0.01}{\$1.4}$). A surprising outcome of this approach is that the effective Ethereum transaction costs may be set arbitrary low by simply multiplying the face value by some factor, and reducing the odds of winning by the same. As this approach possesses the desired properties, we have chosen it as the method of payment in the Gradient Network.

6.4. Implementation Constraints

Now that we have located a suitable abstraction for our payments, the question becomes: how should they be implemented? The main requirements are:

- The method for constructing new tickets must be reusable, as otherwise transaction fees will once again be an issue.
- Double spending must be prevented, or failing that not profitable.
- The system must be sufficiently performant in terms of computational cost so as not to overwhelm the cost of a packet.

Of those requirements, the last element is perhaps the most troublesome. To the best of our knowledge, no method for constructing lottery tickets exist which does not depend on computation similar to that of asymmetric encryption. A modest computer can do around 1e4 such computations per second, but can send 1e6 packets per second when connected to high speed internet. For this reason, although it was not sufficient for use alone, we are forced to employ a balance-of-trade approach similar to the one mentioned above. This in turn leads to a new requirement, namely ”the balance of trade must be kept sufficiently small so as to not cause an incentive to disconnect mid-sale”. As this is a mechanism design issue caused by an implementation reality, let us for now focus on implementation by assuming a solution exists, and defer further discussion until Section 6.5.

[Better version solicited]. We now describe a design which meets the remaining two requirements. Setup phase:

1. Alice deposits money into an Ethereum smart contract which will control outbound payments until some time in the future ($\geq$ 1 minute from now).
2. Bob generates a random number N, and sends the hash $h = H(N)$ to Alice.

Ticket production:

1. Alice creates the tuple $t = (timestamp, facevalue, odds, nonce, h)$, signs it $s = SIG(t)$, and sends the pair (t, s) to Bob.

2. Bob then computes XOR(N, s), and if the result is $\geq$ *odds*, wins.

3. To redeem winners, Bob sends (N, t, s) to the Ethereum smart contract managing Alice's lottery tickets along with a payment address.

To prevent double-spending two methods are employed. First, some portion of Alice's initial smart-contract deposit exists only to prevent double-spending. In the event of a double spend, Alice will suffer losses of 2x the double-spend amount. However, this alone is not sufficient to prevent double-spending, because Alice might decide to overspend on a grand scale. To address this second issue, the value of winning lottery tickets begins decreasing exponentially 20 seconds after *timestamp*, thereby providing a strong incentive for winners to cash in immediately. This immediacy is then used by Bob to compute the "wasting rate" of Alice's smart contract balance.

Lottery ticket type payments are not novel to Gradient. They were first explored in [], with an extremely detailed and helpful expansion described in[]. As described in [], they can also be extended to support anonymous payments, which is a feature we are excited to implement in future versions.

6.5. Balance of Trade

As mentioned above, the realities of symmetric encryption performance prevent us from from sending payment with every packet, and so we need a good understanding of the risks inherent in employing a "balance of trade" approach. We do so here in a general setting: imagine Alice and Bob wish to transact in a fully anonymous manner. Bob is to perform some task for which he charges x, and Alice is to pay him once every y tasks. Unfortunately, the nature of anonymity is such that without prior transactions, Alice and Bob have no mechanism to trust one another. Can they cooperate?

Naively, no. If Alice pre-pays then Bob has no incentive to deliver work, if Bob pre-works then Alice has no incentive to deliver payment. However, in the event that some setup cost to Alice and Bob's relationship (S_{Alice}, S_{Bob} s.t. $S_{Alice} > xy, S_{Bob} > xy$) exists, a delightful property emerges, namely: paying the setup cost becomes economically irrational unless the total amount of work Alice desired Bob to perform is $\geq xy$. As we will see in our discussion of the Agora [Section 8], such a setup cost does exist suited to support trade imbalances on the order of 1e3 packets.

Though it is not used in the system, it is interesting to note that a similar solution exists in situations where there are no setup costs. Alice and Bob simply manufacture a setup cost by burning money.

7. Bandwidth Mining

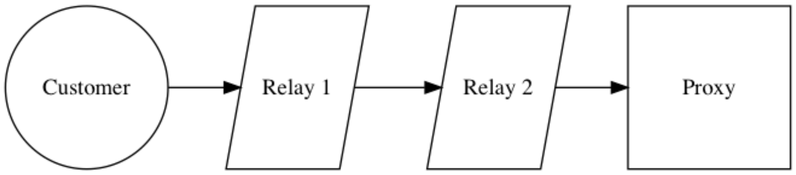

Figure 3: A three-Econ Manifold routing traffic for a Customer

7.1. Relay Nodes

Relay nodes implement a relatively simple behavior pattern:

1. Maintain two connections: up, down.

2. Maintain a keys for symmetric encryption.

3. Messages coming from up are decrypted and passed to down.

4. If the contents of an decrypted message contains tokens for up, send them to up.
5. Messages coming from down are encrypted and passed to up.
6. Monitor the balance of trade, and check/cash-in any winning lottery tickets.

Note that communication in this system is defined only in a single direction. Bi-directional bandwidth is implemented as two Relay Nodes hosted on a single machine which happen to share WebRTC connections. Extensions to this protocol (Section 12.1) are implemented between phases two and three as non-forwarded packets with special instructions. For example, one extension might be "send a packet of all zeros every 3 seconds". Users interested in relays which support specific extensions can include that as a requirement when querying the Agora. Although our initial release will not support user-provided extensions, we hope to do so in the future.

7.1.1. Informational Attacks

Due to the inherent structure of IP-based networking, and the Gradient protocol, relay nodes gain access to the following information:

- The IP Addresses of both up and down. This is required knowledge to perform the IP communication.
- The timing and number of any packets they forward.
- The size of packets they are forwarding, both before and after decryption.
- A public Ethereum key owned by the user who is paying them.

7.2. Guard Nodes

Because relay nodes implement such a simple function, they are relatively disposable. However, the relay that a customer is connected to has a very important piece of information: the customer's IP address. For this reason, users are assumed to assign a preference to long-lived peers as their first node on the system.

(TODO this desire to have long lived peers may itself cause anonymity issues – attackers who are specifically interested in end user IP addresses may disproportionately provide long lived relays.)

7.3. Proxies

Proxies are relay nodes who are willing to proxy requests to the open internet. For this, they may at their option charge an increased fee.

7.4. SSL and TLS Vulnerabilities

SSL Downgrade

7.4.1. The old browser problem

7.4.2. The old phone app problem

7.4.3. Mitigation of 8.1.1 and 8.1.2

1. Check SSL Certificates
2. Check SSL Versions, Cipher Suites
3. Check Basic Constraints
4. "Boring SSL"

8. The Agora

As mentioned in the introduction, the Agora provides a way to traverse the space of currently valid medallions. This is achieved through the emergent behavior of many econs working in concert. The core system requirements provided by the Agora are:

- A method for Econs to join the network.
- A method for interacting with Econs to purchase bandwidth.
- A method for selecting a subset of all peers interested in selling bandwidth, randomly weighted by computational resources, such that the *lookup property* holds:

$$lookup(rand(nodeID)) \approx rand(node)$$

The core functionality exposed by Econs on the Agora are:

- *Get Routing Table.* This returns a signed routing table for inspection.
- *Get Medallion.* To check the current validity of the Econ's proof-of-work.
- *Proxy Traffic.* Pays the Econ to forward traffic to one of the peers in its routing table.
- *Get Offers.* Asks the Econ for a list of services on offer.
- *Purchase.* Employ the Econ in a role contained in its offers.

Typical usage of the Agora for bandwidth purchase is as follows:

1. The Customer generates a random Agora address, A
2. The Customer locates some number of Entry Econs.
3. The Customer checks the Entry Econ's Medallions, and estimates the computational power of the Agoras they are connected to. Whichever Agora has the most computational power is chosen.
4. The Customer gets the routing table, locates the Econ on that list with address closest to A, buys traffic to it and repeats this step until no closer node can be found.
5. The Customer requests offers from all Econs on the routing table it has connected to, filters them according to the capabilities it is seeking, and chooses the median price.
6. The Customer sends its chosen Econ a Purchase message.

The total number of packet sends required to implement the above algorithm is $O(log_2(N)^2)$ where N is the network size.

8.1. Medallion Requirements

The foundation of security on the Agora relies upon requiring that every econ who participates in it holds a current medallion. To ensure that presence on the network will be proportional to computational power, if a single node is more than twice as powerful as our minimum requirements, that node may run more than one econ on the network.

The medallion requirement is ensured two ways: (1) if a non-medallion bearing node attempts to connect to an Econ, that Econ terminates the connection at the earliest possible point, and (2) if a member of the Agora fails to provide a new Medallion after an Ethereum block update, all cons connected to that Node disconnect from it.

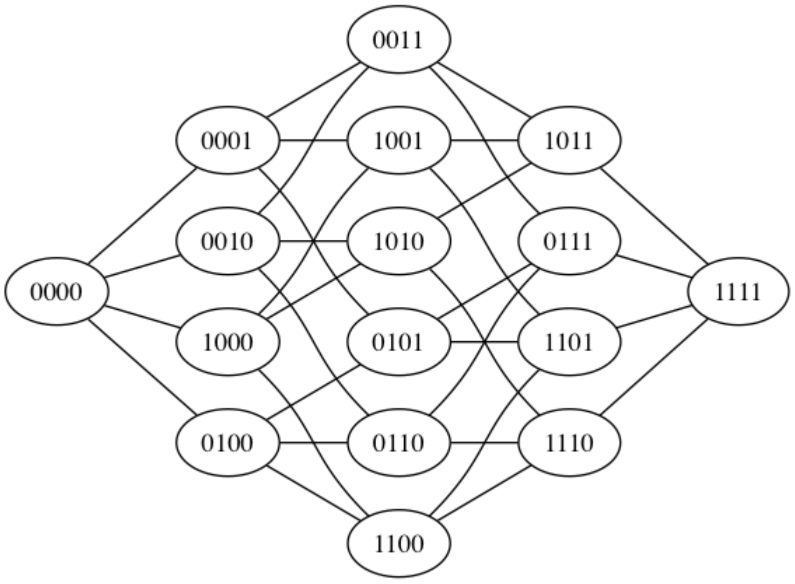

Figure 4: A fully-connected Kademlian graph on a 4-bit address space

8.2. Kademlian Routing

The foundation of efficient queries on the Agora is a method of connecting Econs into a distributed graph. For our connection algorithm, a variation on Kademlia's[] approach was chosen both because of its impressive $O(log_2(n))$ look-up performance, but also for the natural eclipse resistance that can come from its use of randomized connections.

In the approach we call Kademlian routing, each node generates a random address, A, and then forms connections to the nodes with addresses closest to $xor(A, 1)$, $xor(A, 2)$, $xor(A, 4)$ ... $xor(A, 2^{256})$. The number of WebRTC connections a peer must maintain is therefore ≤ 256.

Readers familiar with Kademlia will note this differs from the so-called k-buckets approach, wherein k connections are maintained to peers near a target point, rather than just a connection to the single closest. The reason for this difference is to allow for signed routing tables. [Jay, care to comment? Ignoring the increased traffic I think we could support them.]

8.3. Signed Routing

One of the issues that arises in distributed networks is that no one (except, perhaps, an attacker) has a global view of the network. For example, imagine if in the above routing scheme an attacker chose to lie about what connections it has – if the buyer has no way of detecting this, they might be led off to a fake Agora in which all "participants" were owned by the attacker. To put a stop to this situation, Agora's routing tables are verified non-malicious by the peers contained on them.

When a node would like to establish a forced connection, that node must prove to each node on its forced connections list that each other node on that list is a member of the same Agora. To do this, we first select a random econ G by finding the Econ with an address closest to the hash of all the connections in the routing table $H(C_i)$. Then we supply:

1. Proof that all the Econs on the list can all route to G.
2. Proof that G can route to each Econ
3. Proof that each Econ on the list is indeed a forced connection.

These proofs all take the form of signed routing table chains which lead from C_i to G, or in the case of (3) the chain of signed routing tables which led from the Entry Econ to each C_i. Once such proof has been provided, all of the peers on the new routing table sign the table, and the connecting Econ signs theirs. For those elements of C_i for whom the new Econ is a forced connection, the same proof is sent to each of their connections for signature.

Because this is the only method for adding Econs to the Agora, these requirements form an inductive proof of the Agora's soundness. If one of the nodes in C_i attempts to supply a fake routing table, it will not route to the same G as the other Econs in C_i. If one of the nodes C_i is not a member of the Agora, G will not be able to route to them. If the Econ seeking to connect has lied about C_i being nearest nodes to his forced connection points, (3) will demonstrate that to be false.

From these properties, we can see that the avenues left for an attacker are:

- If an attacker can generate a Medallion address such that all C_i are controlled by them, the above system will cease to function. This will happen with probability $(\frac{attaker}{agora})^{log(agora)}$. If such a collision occurs, $(1 - \frac{agora-1}{agora})^{log(agora)}$ percent of all queries will be compromised. To put these numbers in perspective, if an attacker controls 10% of the network, at 1 million nodes there is a 1e-8% chance of such a collision happening, and if it does occur around 1e3% of all system queries will be impacted. At 100 million nodes the chance drops to 1e-12%, causing disruption of 1e-5% of queries. Note this damage is repaired during Agora Regeneration 8.4.
- If an attacker has now joined the network, but was forced to use a valid routing table, the only attacks it can perform are related to selling services, not routing traffic on the Agora. As this is the situation expected in the rest of our attack models (that an attacker will control a number of econs proportional to the computational resources), we do not consider this an attack.

8.4. Regeneration

Long lived Kademlian networks suffer from Eclipse attacks. In these attacks, an attacker gradually surrounds a node by repeatedly creating keys until ones with the desired addresses can be found. By joining the network with these keys, an attacker can isolate an unsuspecting peer and remove it from the real network permanently. To prevent this behavior from posing any significant risk to the network, Econs on the Agora must change keys every 10 Ethereum blocks.

[Todo: finalize randomization schedule here.]

8.5. Identifying *the* Agora

The above discussion discussion of security is ultimately meaningless if there is no way to locate "the right Agora" on a fresh machine. Any distribution method which exists for *Entry Econs* will be subject to such attacks. To do so, we simply estimate the computing power of a given Agora, and select the right one. This is justified by our core security assumptions [TODO: cross-ref].

- Density Estimation. Because an Econ's forced connections are defined to be the econs nearest to some set of points in a 2^{256} address space, in any real-world situation there will be measurable gaps between the ideal connections and the actual ones. To estimate density in this space, we can observe that these connections as the result of a random binomial process: every point between the ideal point and the actual point is a failure, and the actual point is a success. Therefore, for a given number of missing nodes M and a given number of realized connections C, The uniform prior MAP estimate of network density is
$$\frac{C}{C+M}2^{256}$$
.
- Traversal Distance. The Agora provides address look ups in $O(log_2(N))$ hops. We can use this in reverse to estimate network density.

One might be inclined to believe that density estimation is enough, however a clever attacker in possession of a sybil network of modest size (say s% the size of the real Agora), will have free choice for which node is to be put forward as the the Entry Econs for the false network, while the entry econs from the "real Agora" will have a density which is a random sample from the network. To model the effectiveness of this attack, observe that distance from a point can be modeled as a Poisson process with $\lambda = \frac{C}{C+M}$.

[Pausing this section for return with a fresher brain. We can just simulate the answer. Goal is to ask, given the above possion, what the expected luckiest outcome is relative to s%]

To make matters worse, if the traversal distance is chosen as the metric, one might imagine an attacker who anticipates this, and so creates sub-optimal routing tables which require longer than the $O(log_2(N))$ to traverse.

To recap: if density estimation is chosen as the metric, the attacker will cherry pick dense nodes. If traversal distance is chosen, the attacker will create poorly connected Agoras. The solution here is to observe that poorly wired up Agoras will perform *worse* on the density metric. The attacker can't have it both ways. Therefore the solution used in the Gradient System is pick a random address, traverse to it saving the routing tables along the way, and then perform a density estimate using the routing table from all but the first two hops.

8.6. Whitelists and Capability Lists

Some users wishing to offer Proxy services may not be comfortable offering "open access". For example, allowing users to access facebook.com has a risk profile similar to acting as a relay, while allowing arbitrary connections to the internet may result in a visit from local law enforcement. Econs on the Agora may therefore set a whilelist of websites they will allow users to contact when using them as a Proxy.

9. Attack Analysis and Attacker User Stories

9.1. Economic Attacks / Unjust Enrichment

9.2. Restrictive Internet Providers

9.3. Oppressive Governments

9.4. Oppressive Companies

9.5. Exploitative End Users (Vigna)

9.6. Bad Actors / Anarchists / Small Time Crooks / Petty Hackers

9.7. Scale of attack / resources used. (Global, Boundary, Node)

Imagine that a user wishes to browse Wikipedia while inside a country

9.8. "Great Firewalls"

1. Bootstrapping scheme (tues or wednesday) [ref bonet]
2. WebRTC protocol-level hiding
3. Traffic injection techniques [ref bonet]
4. Cycle amazon IPs / ban all of amazon
5. IPs not listed, requires lots of PoW / money to locate / identities cycle.

10. Performance Scaling

11. Current State of the Software

12. Future Work

12.1. Protocol Extentions

12.1.1. Rendezvous Nodes

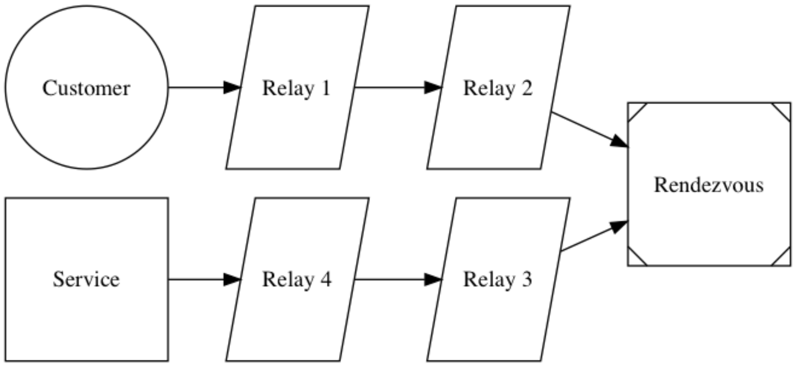

Figure 5

Rendezvous Nodes are relay nodes who connect two Gradient users.
[references]