AutoRoute

People

Eric Oswald | Dimitar Dimitrov | Colin Rice

What is this thing?

Autoroute is a new way to do networking. It is based on intelligence at the individual node level, and strong cryptographic guarantees that messages have been delivered. There are several different components which need to be understood.

- People
- What is this thing?
- Link Level (Layer2)
- Reachability Level
- Routing Level
  - Actual routing system
  - Consequences of the design.
    - Host Liability + Non Payment
    - Relaying Profit
    - Intelligent Routing
    - Security / Abuse Benefits
    - Distributed Systems are “Free”
    - All computers have money

Link Level (Layer2)

The basic component of AutoRoute is an abstraction over layer two networking. Using ethernet packets, auto route announces its existence and learns of neighbours which are participating in the networking protocol.

In particular, when an AutoRoute program comes online, it sends an ethernet packet out to the broadcast mac address, where the type is (31337) and the contents are a hash of it's private key.

When AutoRoute sees another instance coming online, it responds with it's hash in an ethernet packet directed to it.

There are two failure modes, specifically each of these ethernet packets getting lost. In the case of the lost broadcast packet, a failure is indicated by a host on the physical link not connecting. For this reason when coming online the host should broadcast ethernet packet multiple times, and re broadcast every O(5 minutes) in case a new host has come online.

If the response packet is lost, the host receiving the response packet will not connect. In that case resend the packet, several times i.e. once every second for three seconds.

When a host has discovered another host on the network and decides it wants to connect to it, it opens up a TLS connection to the link local IP (fe80::/64 + the last 64 bits of the keys hash) and verifies that the public key associated with the connection matches the hash. This connection allows the two nodes to exchange control messages and establish routing.

Note this cryptographic verification is functionally worthless, but there really isn't any cryptographic guarantee here either, this connection is trivial to spoof and really doesn't do much, it's mainly encrypted to stop passive analysis of the connection.

Reachability Level

The basic level of reachability, is knowing what neighbours are present beyond your immediate neighbours.

Reachability is presented as a set of depth bloom filters, of various sizes. This is primarily an optimization, although it is a powerful one.

Ideally hosts will send the smallest possible bloom filter and then larger ones until the receiving host indicates that is has filled it's available memory. The smallest possible bloom filter is a single bit indicating whether the host is a relay node (i.e. 1 if it has other hosts connecting to it). Larger bloom filters will obviously have more accuracy.

In addition there are two dimensions bloom filters may expand upon. The first is in the hop counter, which indicates how many hops away various hosts are. More bits will allow more accurate weighting of various travel options.

The second is the number of individual hosts it can accurately represent, i.e. the actual bit size of the bloom filter. There should be a standard set of bloom filters, so that hosts can easily compose them.

Reachability information will be exchanged over the TLS connection established in the preceding section.

Routing Level

Routing is mechanically quite simple, although there are very large layer of caveats and things that fall out of the basic routing model.

Actual routing system

When a host wants a packet sent to a destination, it gives a neighbouring host the packet, as well as a deliver value which is how much money the host is willing to pay for proof of delivery. Initially this value will be exchanged via bitcoins, however there is nothing stopping it from being done in some other currency. Packets are targeted based on hashes of cryptographic keys.

When the host it sent the package to responds with a proof of delivery (a signed signature containing the packet hash) the sending host is supposed to pay the host which delivered the host (although it may not).

Note that it may make sense to add additional caveats onto the packet payment system, such as a maximum time for payment, or perhaps a maximum number of hops traveled.

Consequences of the design.

That is all the system does. However there are a number of obvious caveats.

Host Liability + Non Payment

The first is that the sending host may not pay the receiving host. This means that the receiving host must have a mechanism in place for handling this, probably by disconnecting the host. However assuming the receiving host was not the destination then it means that the receiving host relayed the packet to another host, and if the relaying host does not pay the next host in the chain, it will have it's connection dropped. For this reason relaying a packet is taking a risk, and thus routers must not relay a packet if they cannot handle the consequence of non payment, and thus should start non trusted hosts with relatively small amounts of total liability in relayed traffic

Relaying Profit

Additionally, there is nothing specified in how much money the relaying host deducted from the amount of money attached to the packet. The host will not get paid if the packet is not delivered, but otherwise will likely deduct as much money as it can, to maximize it's individual packet. This also means that if it has multiple packets it can relay, it should relay the one it believes it can make the most money off of.

It is worth noting that all hosts should deduct some money, to prevent routing loops occurring which involve them, and should refuse to relay packets with no money attached.

Intelligent Routing

This document does not mention how routing should occur. That is intentional. The point of this system is that nodes should make intelligent routing decisions which maximise their profits. Ideally they would only relay to nodes which can actually relay to the connecting node, as indicated by their reachability information.

Additionally they should monitor packet loss and try and not oversaturate connections, only relaying as much traffic down each connection as possible. This is non specific since we expect that the first draft will relay sub optimally, but there are a large number of improvements that are possible and putting the onus of intelligent routing onto the nodes, rather than at a higher level will incentivize network operators to use more interesting routing methods.

Security / Abuse Benefits

The main reason this system is proposed is to solve the large scale abuse visible on the internet. Things such as DDOS and DNS amplification become much more manageable problems if paying more means that you can prioritize your packets over abusive ones, and will provide an incentive for nodes closer to the abusive traffic to drop the traffic, rather than having it be dropped due to later congestion.

The secondary additional benefit is that it becomes trivial to do end to end encryption for all nodes, and assuming that relay services exist, establishing a tor routing network becomes a much easier proposition as all nodes which are willing to de encapsulate traffic destined for them and send it somewhere else are valid relays.

Distributed Systems are “Free”

Since distributed systems that involve most of the nodes on the system, will have direct connection between nodes, there is no actual “relaying” occurring, which means that they don’t have to pay for network costs. This will most likely become obvious since all nodes need access to the bitcoin network (or some equivalent payment network) and will likely want to relay transactions and blocks.

All computers have money

This may not be initially obvious but if all computers are doing economic routing along these lines, there is nothing stopping other digital services from using these sorts of microeconomics based decisions for other commodities such as API calls.

Stack

Go. Linux Kernel. Networking Hardware. That’s it.

Timeline

Weeks 1-2: Migrate l2 library, get basic link level working

Weeks 3-4: Get a basic reachability item working

Weeks 5-6: Make some packets flow, ping should be working

Weeks 7-8: Make reachability awesome (adaptive loading, duration, depth, routing loops etc…)

Weeks 9-10: Make packets flow well and sensibly (liability limits, automatic rerouting of packets etc…)

Weeks 11+: Get real networks using it, fire all the network engineers, Yeehaw!