Open Libernet [DRAFT]

Take back the Internet

www.openlibernet.org

Abstract

A decentralized global mesh communication network that spreads the burden of its expansion and maintenance among its users could potentially eliminate the need for large Internet service providers and government internet regulation. It makes the internet much more robust and fault tolerant, as the traffic no longer follows pre-defined routes, but can simply navigate around broken links or dead channels. This also comes with the benefit of eliminating the unjustifiably expensive bandwidth fees and the possibility for draconian laws and regulations empowering governments with the ability to breach people's privacy or monitor and censor their content, that are slowly becoming a reality today. The key however is to properly motivate the participants by providing a suitable incentive.

I. Introduction

The Internet is regarded as the largest communication network on the planet. Its power lies in its flexibility and ability to absorb localized damage without disrupting global communication. It has so fulfilled its initial promise of creating a robust and decentralized, interconnected communication channel that could survive potential catastrophes. Much as with any new technology, the prohibitive costs of developing, installing and maintaining networks meant that only large corporations or governments could afford such endeavors. This has led to the establishment of new global monopolies over the internet market. Also, the increasing dependency of the world on the internet and the tight grip of a few corporations over traffic routing and delivery mean that it was only a matter of time before governments would try to step in and enact restrictive laws to monitor and eavesdrop on communication and possibly even censor content.

The solution is to slowly transition off the traditional aging internet (that we will refer to as the current internet) to a new completely separate network, based on a new technology, we will be calling Open Libernet, which essentially relies on inexpensive existing equipment (off-the-shelf routers and access points, or old desktop computers) that users can install and maintain to form a mesh network.

While the idea of creating global Mesh networks has been considered by many technology enthusiasts and wifi amateurs around the world who have come up with revolutionary concepts and protocols, such hobbyist projects often hit a dead end when they have to face the reality that adoption cannot be stimulated by the sheer beauty of the technology alone, especially when coupled with the costs of developing, installing and maintaining equipment. The collapse is almost inevitable as the projects are faced with the vicious cycle of the Network Effect: it makes little sense to join a communication network, when there’s no one to communicate with, and there’s not much of an incentive for most people to participate.

Our approach borrows from existing Mesh network technologies, but adds the essential element of incentive. The system also borrows from the successful and tested concept of the peer to peer Bitcoin cryptocurrency [1]. The idea is to establish a new routing protocol that is built around a robust and fault-tolerant payment system that automatically and justly rewards users for installing and maintaining the infrastructure. Obviously, early adopters will be rewarded in abundance for spearheading the technology.

In this paper we describe Open Libernet, while briefly explaining its topology and routing protocol.

II. Objectives

- Create a global decentralized interconnected mesh network for efficient communication
- Provide an incentive for all members to actively and efficiently maintain, expand and upgrade the network
- Enhance privacy and security by introducing Layer 3 packet encryption
- Reward early adopters without creating a high barrier to entry for late-adopters

- Eliminate the possibility for worldwide bandwidth monopoly
- Eliminate the possibility for regulation or control over the internet or the communication channels by any entity
- Eliminate the pointless and restrictive traffic quotas and allow for the optimal use of the network equipments to their full capacity as opposed to artificially imposed bandwidth restrictions and connection speed limits
- Create a new democratized bandwidth economy with perpetually decreasing prices to mirror the constantly growing network capacity due to network expansion and advances in hardware and equipment quality
- Create a robust network that can truly survive any global catastrophe

III. Mesh Networks

The concept of mesh networks is simple. It is usually a network made up of Nodes that are connected together in no particular hierarchy. Every Node tries to make as many connections to other neighboring nodes as possible thus creating a Web-like structure. The nodes communicate with their neighbors in ad-hoc and rely on each other to get their messages delivered. Packets sent from a source node find their routes to the destination by means of intermediate nodes (or hops) following a certain routing protocol. Since all nodes are basically equal and the network is decentralized, it is similar in a sense to a Peer-to-Peer network. In this document we will use the terms ‘node’ and ‘peer’ interchangeably to refer to participants in the network.

In mesh networks, packets do not follow predefined static routes and constantly try to navigate around broken, congested or damaged nodes. This makes the network much more fault tolerant and robust, but adds to the complexity of the routing protocol, simply because smaller low-end nodes are less reliable than a dedicated high performance infrastructure of routers. Even more complexity is added when we consider the fact that mesh networks are inherently much more organic in the sense that nodes maintained by end-users may intermittently leave and join the network due to power-failures, bad equipment, wireless interference or malicious tampering. Such problems are less likely to occur with ISPs and large corporations with dedicated hardware and maintenance staff.

In an ideal and honest mesh network, you expect others to help you get your messages across, by returning the favor and routing their messages in return. When confined to the Layer 3 of the OSI model, Mesh routing protocols become agnostic to physical or Link layer media. This means it doesn’t matter whether the nodes are connected through wireless, fiber or copper cables. Likewise, it allows existing software that live on the application and transport layers to function with little to no modification.

IV. Problems with current community-driven mesh networks

- The vicious cycle of the Network Effect discourages early adopters and stifles network growth

- Lack of incentive means cheaper equipment and less reliable networks
- The decentralized network is made up of nodes of one type, no one is willing to install and maintain expensive and specific equipment for the sole purpose of serving the community
- Dishonest nodes could easily abuse the network by either generating much more traffic than they help route, or refusing to route other people's traffic efficiently, with no way to punish or restrict them

V. Incentive-driven approach

The concept is simple, users who join the mesh, will be contributing to the stability, robustness and security of the whole network. The network will repay them for their efforts according to their contribution. The reward is naturally expressed in traffic. In simpler terms, for every Megabyte of traffic you help route efficiently, you get 1 Megabyte of your own to spend. This means by simply adding a node to the network and keeping it online, you will be accumulating bandwidth for your own consumption. The bandwidth is of course transferable, meaning you can lend it, donate it, borrow it or sell it at will.

The system also defines different node types, with different functionalities. Some simply relay traffic, others provide a tunnel to other smaller subsets of the mesh network (by means of current internet connectivity), and others still process verification for payment transactions. The system defines an efficient reward mechanism for all those different components.

Under normal conditions, a well-positioned node could route more traffic than it is expected to consume. We aim that by the time Open Libernet becomes a reality, internet will be almost free for personal use. Service providers, such as web hosting companies and data centers will still need to purchase bandwidth to keep their services running. Much like they do today, but at a perpetually decreasing cost.

As for dishonest nodes, the system describes efficient mechanisms for punishing abusive behavior, from downgrading the fidelity of the node in question, to ostracizing it completely.

VI. The system components

The routing protocol offers a thorough description of the functioning of each of those components. In brief, here are the main components that make up the mesh.

1) Simple Node:

It's the main entry point of any user to the network. It can be as simple as a cheap off-the-shelf router or access point mounted on a rooftop or as complicated as a full high-end router with multiple long distance fiber-optic cable connections. It routes traffic belonging to its neighbors, as well as traffic originating from the node itself. It keeps a count of all packets it routes for each of its neighbors as well as traffic consumed by the node itself for accounting purposes. The location of the node as well as the quality of the hardware, both play a role in the total routed traffic, and hence generated profit.

2) Tunnel Node:

Same as a simple node, only it provides an optional tunnel to other Open Libernet networks, by means of an internet connection. The tunnel node maintains a number of application-layer P2P connections with other tunnel nodes from other Open Libernet networks that are otherwise physically inaccessible from where this gateway and its neighbors are, and exchange packets over the current internet, thus creating a bridge between 2 disconnected subsets of the Open Libernet network.
As our network grows, gateway meshes become increasingly redundant as faster, more stable connections become available. Until then, tunnel nodes will help route more traffic as they provide links to far away nodes and will therefore generate more revenue, at the expense of current internet bandwidth. People who have unused, cheap and excess internet bandwidth can easily share it to generate Open Libernet traffic using a tunnel node.

3) Peer-Lookup Node (PLN):

Unlike a simple or tunnel node, which can be run on cheap readily-available equipment, Peer Lookup Nodes require more advanced hardware, such as a computer running Linux with sufficient external storage. These nodes need to store and maintain large data tables that are needed for efficient routing of traffic. In effect, our protocol proposes a Locator/Identifier separation functionality (similar to what the IETF LISP aims to achieve), but assigns to each node two different addresses. One is the permanent Peer Address, the other a temporary IP address that is obtained automatically and organically changes as the shape of the mesh changes due to joining and parting nodes. We need an efficient way to keep track of IP addresses and help translate them into Peer Addresses. This process is explained in greater detail in the routing protocol.

4) Payment Processing Node (Mining):

Those are the most complicated components of the network. They do not help route traffic, but are mainly responsible for validating payments among peers. The algorithm is tightly modeled after the Bitcoin protocol, where miners work together to solve blocks, by executing the very processing-intensive task of transaction block validation. This ensures that all payments are valid, and no double spending has occurred. It also provides a way for peers to transfer traffic to each other, or sell them at traffic exchanges. Miners are rewarded with free traffic for each block they solve, as well as transaction commissions. They also lay out the foundation for the traffic economy and provide a means for the network capacity to be expanded.

VII. Traffic economy

Since traffic as a transferable commodity with a high marginal utility and obvious intrinsic value can be seen as a currency, it is possible for one to assume that the system will create a new monopoly, now run by the earliest adopters. But this would be an inaccurate assumption. Our newly created Traffic economy is largely inflationary, meaning traffic is perpetually decreasing in price due to a systematic and planned increase in supply. This is only natural, since as the network grows, so does its capacity. It is only fit that bandwidth should become increasingly cheaper as technology advances and more links are created. It makes little sense from an economic-perspective for anyone to hoard something that is projected to decrease in value in the future. The network implements highly prohibitive measures to curb the use of traffic as a currency.

VIII. Payment Procedure

Accounting is calculated at Layer 3 packet level. Since transaction validation is computationally intensive and adds to the network overhead, it is obviously impossible to issue payments on a per-packet basis. Nodes will aggregate routed packets and only initiate a payment request when a set threshold is reached.

Since neighboring nodes are more likely to owe each other packets than farther away nodes and are therefore more likely to reach that threshold, payment is only initiated between neighbors.

The payment procedure is straightforward. Each node keeps accounting information for each of its neighbors and updates their balances as packets flow to and from each. Once a threshold limit is hit, the peer that is owed traffic initiates a payment request by sending a Payment Request packet to the neighbor that owes it traffic. The neighboring peer validates the requested amount against its own books. If the transaction is deemed valid, the Payment Request is signed and returned to the initiating peer. The initiating peer now has proof that the payment has been approved by the payer and is therefore valid. It then forwards it to any Payment Processing Peer (miner) it chooses and waits for the transaction to be approved. Ideally, the reciprocal balances of two neighbors should match, unless the protocol on one of the nodes is maliciously tampered with, or due to packet loss. We will cover those cases in more detail later.

IX. Payment Strategies

Payments are propagated and distributed among all nodes in any given route based on four distinct payment strategies.

We define:

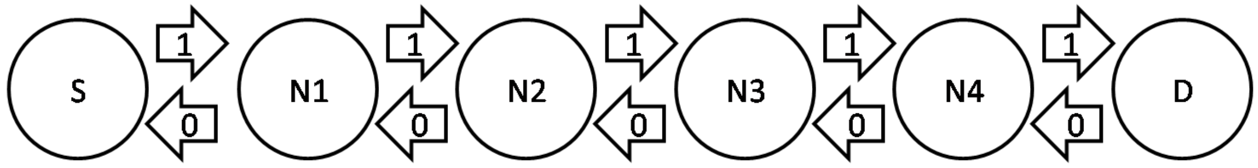
- P the price of a single Packet
- N the total number of hop between the Source and the Destination
- i the hop position of each node along the route
- C_{i-1} and C_{i+1} the credit of the accounts of a node's direct neighbors in the route
- D_{i-1} and D_{i+1} the debit of the accounts of a node's direct neighbors in the route
- B_i the total balance of each node

1) Free Packets:

Those are the ones reserved for network maintenance and topology discovery. They should be routed free of charge. Failing or refusing to route those packets will only harm the node itself as it only stops it from being reachable.

2) Forward paid Packets:

Those are packets that serve a function, but also carry a payment to their receiver. Routing nodes are expected to pass the packet for free.



In this case a sender node S is sending a packet priced P to a destination node R, by means of a route R = {N1, N2, N3, N4}. By the time the packet reaches its destination D, below will be the individual node balances:

B_S = -1P

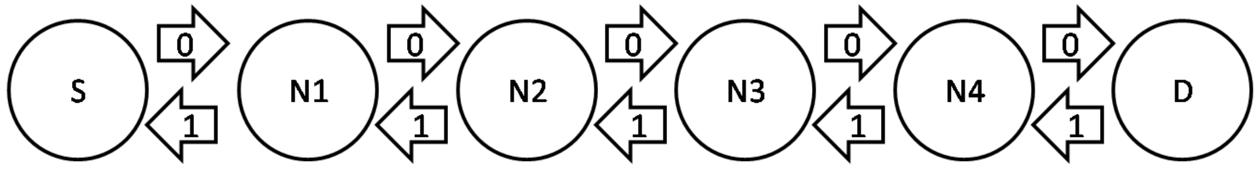
$B_1 = B_2 = B_3 = B_4$ = 0P

B_D = 1P

In effect, this payment strategy is never used as it lends itself to abuse. Obviously, any node along the route can simply absorb the packet and consume the payment it holds. It is only explained for demonstration purposes and to make the following strategies easier to understand.

3) Backward paid Packets:

Those are packets that serve a function, but also carry a payment back to the sender. Routing nodes are expected to pass the packet for free.



In this case a sender node S is sending a packet priced P to a destination node D, by means of a route R = {N1, N2, N3, N4}. By the time the packet reaches its destination D, below will be the individual node balances:

B_S = 1P

$B_1 = B_2 = B_3 = B_4$ = 0P

B_D = -1P

No node has any interest in refusing to forward a backward-paid packet, since absorbing the packet means they will also absorb the cost associated with it.

4) Two-way paid Packets:

Those represent the majority of all packets in circulation. They contain the payload of the data being transferred, and they carry a payment to all the nodes that help route the packet.

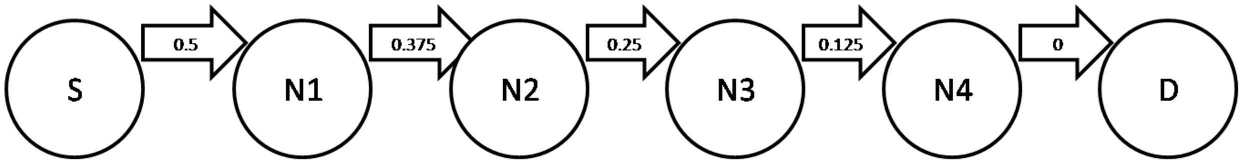
Since both upload and download should be accountable, both the Source and Destination should equally pay for every packet an equal sum $\frac{P}{2}$

The nodes that make up a route should all be rewarded equally with an amount $\frac{P}{2N}$ from each side of the transmission, totaling $2 * \frac{P}{2N} = \frac{P}{N}$

This requires that every node in a route know the total number of hops from source to destination, as well as their position in the queue. We will later explain in detail how the routing protocol ensures such information is available to all nodes in the route.

For demonstration purposes, we will break up the payment stream into two diagrams, showing how each node adjusts its books to account for payments originating from both the Source and the Destination.

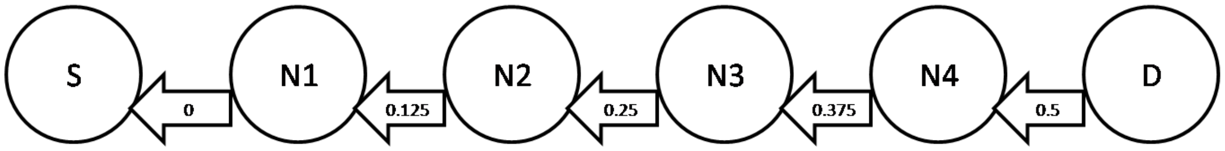
By splitting the amount paid by the Source equally among routing nodes, each node will adjust the balances for its neighbors as follows:



S: Credit = 0.5P, Debit = 0P
N1: Credit = 0.375P, Debit = 0.5P
N2: Credit = 0.25P, Debit = 0.375P
N3: Credit = 0.125P, Debit = 0.25P
N4: Credit = 0P, Debit = 0.125P
D: Credit = 0P, Debit = 0P

The respective balances:
B_S = -0.5P
$B_1 = B_2 = B_3 = B_4$ = 0.5 / 4 = 0.125P
B_D = 0P

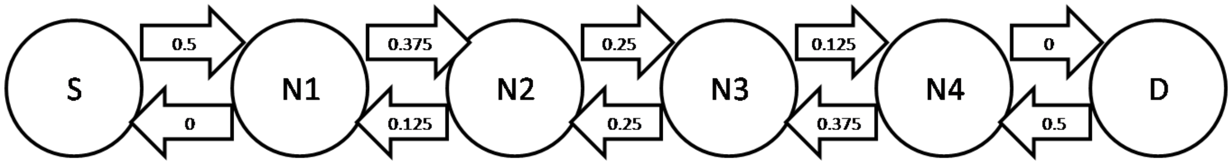
Conversely, by splitting the amount paid by the Destination equally among routing nodes, each node will adjust the balances for its neighbors as follows:



S: Credit = 0P, Debit = 0P
N1: Credit = 0P, Debit = 0.125P
N2: Credit = 0.125P, Debit = 0.25P
N3: Credit = 0.25P, Debit = 0.375P
N4: Credit = 0.375P, Debit = 0.5P
D: Credit = 0.5P, Debit = 0P

The respective balances:
$B_S = 0P$
$B_1 = B_2 = B_3 = B_4 = 0.5 / 4 = 0.125P$
$B_D = -0.5P$

The complete payment scheme is as follows:



Debit for both neighbors (amounts to earn from neighbors)

$$D_{i-1} = \frac{P}{2} - (i-1)\frac{P}{2N}$$

$$D_{i+1} = \frac{P}{2} - (N-\ i)\frac{P}{2N}$$

Credit for both neighbors (amounts to pay neighbors)

$$C_{i-1} = \frac{P}{2} - \frac{iP}{2N}$$

$$C_{i+1} = (\mathrm{N}+1-\mathrm{i})\frac{P}{2\mathrm{N}} - \frac{P}{2}$$

X. Routing Protocol

1) Overview

The problem with routing protocols is that they are resource-intensive. They need to store huge databases for their routes and will occasionally need to exchange routing table information with other connected routers. Furthermore, without the intervention of IT Professionals to set up the network topology and configure the routers, maintaining a network is almost impossible. For a decentralized mesh network to be deployed efficiently, it needs to rely on low-budget off-the-shelf equipment that can be installed and maintained by non-IT professionals.

The idea is to subdivide the network into smaller clusters and define the protocols needed to maintain the topology. This allows us to shrink our routing tables and distribute it over the network nodes. Since every node is only physically connected to a few other neighbors, routing decisions will be made independently on each node in a greedy process. Once this is done, any routing protocol can be applied. We propose a model based on the Bellman-Ford Distance Vector algorithm, with a few modifications to avoid routing loops.

The subdivision process that defines the topology of the network is borrowed from the Netsukuku Mesh protocol (Ntk) [2]. You may check out the work done on Ntk for more information on how it works. This document will go through all the concepts again and explain things in detail, highlighting all the differences. The implementation details will be released in separate RFCs later, along with detailed procedures and pseudo-code. This document will only explain the concept and provide an entry point to the problems of the efficient routing of traffic and the payment procedure.

Our protocol assigns to each node (or peer) two different addresses: one is the Peer address, which is used to uniquely identify the node, the other is the IP address which is a temporary dynamically obtained address used only for routing purposes.

2) Peer-addresses:

A peer address is the hash of a randomly generated cryptographic public key whose private key is only possessed by the peer who generated it. The public/private key combination are used to encrypt/decrypt some special packets sent from/to the peer, as well as serve as a cryptographic signature to verify that the peer is indeed the authentic source/destination of any data that requires authentication, such as payment transaction approvals. This helps eliminate impersonation attacks and ensure privacy and security.

The peer address also serves as an account for payments and transfer of traffic in a fashion similar to Bitcoin addresses. Most importantly, the peer address represents the actual address of the machine to which traffic is destined. It is important not to confuse Peer Addresses with IP Addresses, which provide an unreliable reference to the address of the machine, but are still needed for routing traffic.

Just as in CJDNS [3], the peer address is the node's permanent address. But unlike CJDNS – which utilizes a distributed hash table to allow routing of traffic – Peer addresses in Open Libernet are completely independent of the routing process as is explained later.

3) Peer-address collision:

There are $2^{128} \approx 3.4 \times 10^{38}$ possible peer addresses. That's many times more than there are stars in the observable universe. Even when considering the Birthday paradox, the statistical probability of a collision is negligibly low. Also, it is impossible to register a peer address (public key) on the network without having the private key associated with it. It is also impossible to receive traffic destined to another Peer, therefore eliminating the risk of impersonation attacks and boosting security.

4) IP Addresses:

Whenever a peer joins the network, it is assigned an IP Address automatically within the range of its neighboring peers. The node must then register its new IP Address on the network, binding it to its Peer Address by providing proof of ownership of the Peer Address by means of the Peer Lookup node. As new nodes join, it is possible for the cluster to fill up, and the node may be forced to move to a different cluster. This may result in the node's IP Address changing, hence the temporary and unreliable nature of IP Addresses in our system.

5) Network Topology

In a mesh network, routes do not need to contain the full path from source to destination, since every node is only physically connected to a few neighbors through Layer 2 Gateways. Each node needs only know which of those gateways can successfully lead to the destination.

But even then, storing a route for every other node in the network will quickly fill up the limited storage space of a small router and make efficient querying of the map impossible. It is therefore necessary to structure the network into smaller clusters.

In multi-hop networks, it is not possible to use hierarchical addressing schemes. This is because following the failure of a link; a node's IP Address may have to change to maintain the hierarchical scheme, which renders communication unreliable. This will break any TCP connection that may have been established and disrupt all protocols from Transport up to Application layer.

a) Hierarchical Subdivision

The hierarchical subdivision of the network is straight-forward:
The whole network is subdivided into 256 groups, which we will call gnodes (Group Nodes), that make up the first Level of the network, or Level 1.
Each Level 1 gnode, is again subdivided into 256 subgroups, making up Level 2.
Each Level 2 gnode is yet again subdivided into 256 subgroups, and so on…

The recursive subdivision of the network will lead us to the base-Level n, which contains the elementary unit nodes, which represent the actual peers (routers). For this, we will consider the IP address model. Since we're using IPv6 addresses, the network can be broken down into a total of 16 levels.

A node needs only worry about reaching nodes in its direct local group n, and the nodes that lead to each of the gnodes of higher levels. In other words, each gnode needs to store a route in its internal map for every other gnode in its direct local group of level n, as well as a route for each of the gnodes of Levels 0 through n-2. That is a maximum of 256 routes per level.

The implications of this subdivision, is that now each node only needs to store a total of 256n routes, instead of 256^n.

Consider a hypothetical network of 3 Levels (for simplicity), with a node A=1.2.3
The internal map of the node A contains the following routes:
Level 3: 1.2.[0..255] (256 routes)
Level 2: 1.[0..255] (256 routes)
Level 1: [0..255] (256 routes)

A route to the node B = 1.3.4 does not exist in the internal map of A, neither does a route to the node C = 2.3.4. To reach the node B, A needs to forward the packet down any gateway that either knows how to reach B, or knows a route to a node that does (in other words, any node from the gnode 1.3.[0..256]).

With this subdivision in mind, it is simple to see how the levels are flattened. The routing decisions can be executed at each level independently.

b) Communicating Vessels

The above subdivision requires that the network be configured conveniently, and the IP addresses assigned accordingly. But without a centralized authority to assign IP addresses and maintain the hierarchy, it may turn into a mess.

For this we will define the following concepts:

- Internal Node of G: It's a node that belongs to a group G, and whose neighbors all belong to G
- Bnode (or Border Node): It's a node that belongs to a group G, but is also physically connected to one or more nodes of different Groups.
- Internal Connection: summarized by the following rule:
$$\forall\, x, y \in G \;\exists\; a\ path\ x \to y\ all\ contained\ in\ G$$
- Allocated Group: We say that G is allocated iff it isn't empty
- Saturated Group: We say that G is saturated iff all its nodes are allocated

We need to consider the following restrictions:

- The groups must always remain internally connected
- The address space is finite
- It is possible for a gnode to be saturated even if its lower level gnodes are not, hence the need to make sure the address space is allocated properly
- The network is dynamic as nodes frequently join and part

The solution to these problems is inspired by the Communicating Vessels of fluid dynamics, and is loosely based on the Ntk implementation of the communicating vessels.

From Wikipedia: "*Communicating vessels, sometimes referred to as communicating vases or pounding utensils, is a name given to a set of containers containing a homogeneous fluid: when the liquid settles, it balances out to the same level in all of the containers regardless of the shape and volume of the containers. If additional liquid is added to one vessel, the liquid will again find a new equal level in all the connected vessels. This process is part of Stevin's Law and occurs because gravity and pressure are constant in each vessel.*"

By emulating the Communicating Vessels, we aim to keep all groups uniformly allocated, keeping the gnodes as compact as possible, while allowing new nodes to join and part freely.

c) Hooking Phase

The hooking phase begins when a node is connected to the network and requests to join a gnode and deals with the dynamic IP Address assignment.

The system must ensure that:

- A node must be able to join the network at any time
- The IP Address of the joining node must fall within the range of its neighbors, such that the group this node is trying to join remains internally connected
- IP Address conflicts must be avoided

First, and to eliminate the churn caused by unstable nodes, a Layer 2 link becomes active only after it has been alive for a set amount of time. This artificial limitation would help avoid flooding the network with routing table updates every time an unstable route is established or broken. This also closes the door to mobile nodes. During that time the new neighbors could probe the link to determine its initial Estimated Transmission Cost. More on this later…

d) The Hooking Procedure

The first thing a hooking node does is initiate a handshake process with its neighbors by looking at their maps. At this point the node will acquire the following information:

- The Peer Address of each neighbor
- The full gnode address of each neighbor
- A unique identifier (fingerprint) for the gnodes at each level of each of the neighbors (generated when the gnode was first established as we'll see later) which includes the up-time of the gnode
- The number of nodes in the base level gnode of each neighbor
- The PLN's peer address and IP Address that each neighbor is using

Then, it must choose a gnode to join, by requesting an IP Address. For this, it will send a gnode Join Request packet to a neighbor it chooses as its mediator.

The mediator is chosen based on the following rules:

- If all the node's neighbors are part of the same base level gnode, the mediator is chosen at random
- If a node has neighbors that are part of different gnodes, it will choose to join the gnode that is not saturated, that has the highest number of unallocated nodes
- If all the neighboring gnodes happen to be saturated, it will choose – at random – a neighbor from the gnode with which it has the most connections

Once a mediator is chosen, the node sends a Gnode Join Request Packet. The mediator, looking at its internal map, will proceed with the following:

- If the base-level gnode is not saturated:
    - Respond with a Join Accept Packet with a randomly selected unallocated address from the base-level gnode
- If the base-level gnode is saturated:
    - Try to establish a new gnode, if possible, and respond with a Join Accept Packet, with a randomly selected address from the newly created gnode
    - If it's not possible to establish a new gnode, respond with a Join Deny Packet, demanding the hooking node to back off for a set period of time, before trying to hook again
    - Initiate a Request Departure Flood, asking some of its gnode peers to leave, if possible, to make room for the new hooking node

e) Request Departure Flood (RDP)

The RDP is a special packet, that propagates free of charge within the base-level gnode.

A node that receives a Request Departure Packet will act upon the following rules:

- If the node is a border node, it will request to join the neighboring gnode it is bordering (based on the Hooking Procedure explained above), then leave the current gnode. We will call this process Node Migration.
- If the node is not a border node, it will forward the packet to all its neighbors, except the one it received it from
- A node will only forward an RDP from the same flood once

f) Node Migration

Any node that receives an RDP and satisfies the conditions for migration must honor it. The migrating nodes, will initiate a flood of their own in the gnode they just left, informing their peers of their departure. The floods will eventually reach the mediator node, which will now have a list of unallocated addresses to assign to the hooking node, when it tries to re-hook after its back-off period elapses.

Similarly, the nodes that join a new gnode will send a flood informing their peers of their arrival. The arrival/departure floods will be covered later in this document.

Finally, every time a node obtains a new IP address, it must register it with a PLN, binding the new IP address with its Peer Address. This process will also be covered later.

It is important to note that departing nodes will not declare their departure until they have successfully secured their position within the new gnode. This may include intentional delays to give the new gnode peers time to update their routing tables to include the new node as well as making sure the PLN has updated its lookup-table and spread the word across the PLN network. In the meantime, the migrating node can be seen as having 2 IP addresses. This ensures an almost seamless transition.

g) Establishing a new Gnode

To establish a new Gnode, a mediator node will claim the base-level address of the lowest possible level n-1 and above unsaturated gnode. It can find it by looking in its maps. It is possible for the address space to be completely allocated. In that case the whole process is dropped.

If the node is able to find an unsaturated higher level gnode, it will simply claim it then start a flood informing the network of the creation of this new gnode. Ideally, the lower level gnode IDs can be zero.

To give an example, suppose a node with IP 10.11.12.13 is trying to establish a new gnode to assign to a hooking neighbor. It will look for an unallocated gnode in 10.11.*.*.

Suppose 10.11.15.* is free, it will create the gnode 10.11.15.* and claim the IP Address 10.11.15.0.

If 10.11.*.* is saturated (all the gnodes of 10.11.*.* are allocated), it will look for an unallocated gnode in 10.*.*.*.

Again, if it finds an unallocated gnode, and suppose that gnode is 10.15.*.*, it will create the gnodes 10.15.*.* and 10.15.0.* and claim the IP Address 10.15.0.0.

If all the gnodes are saturated (including the top-level gnode of level 1), the mediator will be unable to create a new gnode, and the whole process is dropped.

h) IP Address Conflict

It is possible for two nodes that join a single gnode within a very short time difference to be assigned the same IP Address by two unsuspecting mediators. This will cause an IP Address conflict and confuse the gnode peers.

But remember that the peers at the base-level n gnode are gnodes themselves. An IP Address conflict is therefore from the network's perspective the same problem as a gnode conflict and is resolved the same way. One of the gnodes in conflict will have to rehook, by going through the normal hooking procedure again.

i) Gnode conflict

Gnode conflict may occur if a mediator decides to declare a gnode that has already been declared on the network. This may be problematic as it will cause IP Conflicts and confuse the network as to where traffic should be directed.

To resolve the problem of Gnode conflict, we will introduce a fingerprint function, which creates for each gnode a unique ID, generated at the time the gnode is established. A node that receives a route to a new gnode will store it and will no longer accept alternative routes to the same gnode unless the fingerprints match, as long as it still has a route to the previous gnode.

The process is simple:

- When each of the gnodes in conflict is first advertised, a flood is propagated updating the routing tables of all the peers on its way, effectively creating 2 Zones

- The flood eventually reaches nodes that have recognized the other gnode as per its fingerprint and will reject the new gnode. However, this doesn't stop the flood.
- A node N that receives a route to a gnode with a mismatching fingerprint will respond with an advertisement of its own route instead, but still propagate the flood advertising the "bad" route
- The two floods advertising routes to both gnodes in conflict will cross the borders of both Zones and eventually reach the border nodes of both gnodes
- Any node from a gnode that finds there's another gnode with a different fingerprint will leave its gnode and hook with the border gnode, then start a flood advertising the move
- This will cause floods back and forth as each layer of border nodes escape both gnodes into the neighboring gnodes, causing the gnodes in conflict to shrink in size until one of them disappears completely

It is important to note that this condition should be extremely rare as it only happens when two subsets of the network meet for the first time. Once the network has been established, this case should no longer happen.

Also, the longer a gnode has been around, the less likely such event would cause any harm. And when it does, the newer and smaller gnode will quickly be plucked out.

Finally, the problem does not cause any disruption in the communication since either of the gnodes didn't know the other one existed in the first place, and all the nodes in between making up the 2 zones are only recently finding out about the new subset of the network. The larger the 2 zones the longer it takes for the network to absorb all the new information, which is only natural.

j) Broken Gnodes

Gnodes can become broken if some of their gnodes are no longer internally connected. This can only happen if a node that happens to be the single connection between two subsets of the gnode dies, or a tunnel node connection breaks leaving orphaned nodes on two separate sides of the network. This is a real problem, as it does not fall under the category of Gnode conflict since both subsets actually have the same fingerprint.

A node that receives news of a broken link can easily find out if its gnode is broken, and if it does, whether or not the node itself falls in the smaller subset of the broken gnode. This is as simple as comparing the number of nodes that are no longer reachable with the number of nodes that are still reachable.

If the node is on the losing side (the smaller subset), it will attempt to leave the broken gnode. First the border nodes will escape, then their neighbors, and so on… until the smaller subset disappears.

6) Building the Routing Tables

The routing table of each node is a hierarchical map made up of 16 levels, representing the gnode levels described by the Network's Topology.

For that, an entry in a node's map or routing table can be described by the following tern:
rN := (dst, f(dst), gw, etx)
where

- dst is a node: the destination node of the route
- f(dst): the fingerprint of the dst gnode
- gw is a node: the gateway of the route
- etx is a metric that defines the Estimated Transmission Cost

a) The Bellman-Ford Algorithm (Distance Vector)

The etx of each route in the node's routing table is the total cost to reach a destination dst. This is seen as the sum of all the costs of all the links along the path. The calculation of the etx can be done using different methods, and could include bandwidth, round-trip time, hop count and packet loss.

With that in mind, the network is seen as a weighted directed graph and the problem of routing packets efficiently is the process of keeping the routing table up-to-date with the lowest cost route to any destination node dst.

The algorithm can be summarized as such:

- Initially all nodes have a route to a node dst with a maxed-out etx, meaning the link is either dead or non-existent
- Direct neighbors of dst will advertise their route to dst, using as etx the efficiency of their direct links to dst
- A node that receives the advertised route will first adjust its etx, adding to it the cost of the link to the advertising node, then compare the new etx with the one in their table:
    - If the new route is better than their own, they will replace the old route, then create a flood advertising that they can reach dst at the adjusted etx
    - If the route in their table is better than the advertised route, they will respond with a counter-advertisement, with themselves as the better route with the better etx

b) Routing Loops (Count-to-Infinity problem)

The Distance Vector Algorithm as described till now suffers from the Count-to-infinity problem, which creates routing loops and makes route convergence slow or impossible.

To better understand the problem, consider the following case:

The nodes A, B, C and D can reach a destination node dst through D.

A uses C to reach dst with a etx of 3

C uses D to reach dst with a etx of 2

B uses D to reach dst with a etx of 3

When D finds it is unable to reach dst, it will tell all its neighbors (B and C).

If, for some reason, the message doesn't reach C in time (or doesn't reach C at all), and B was able to alert A of the change. The following chain of events will occur:

1) B will tell A "I can no longer reach dst" (etx is maxed out)
2) A will look in its table and find that it has a better route C, which can still reach dst
3) A will respond with "I have a better route to dst, with a etx of 3"
4) B has no way of telling A is using the same node D to reach dst, and both A and C are still unaware of the change. So B will now be convinced that A has a route to dst, and will update its routing table.
5) B will send a message back to D "I have a better route to dst, with a etx of 4"

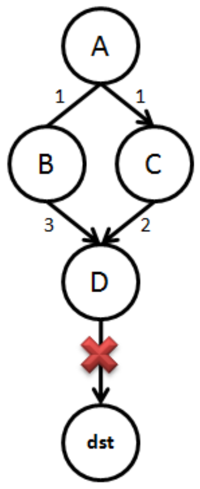


Figure 1 Count-to-infinity problem in the Bellman-Ford algorithm

This will create the loop ABDC, with each node relying on the next to reach a destination that is no longer reachable through either.

In reality, D will have to inform C of the increased cost of its link to dst, and the route downgrade feedback will keep traversing the loop until the etx is maxed out and the loop is eliminated. But the process is costly and time-consuming, and in the meantime, packets that are using this path will be sucked into the loop.

c) Eliminating Loops - Route Probe Packet (RPP)

Our proposed solution to the Count-to-infinity problem is straight-forward:

A node that receives a packet signaling a broken or deteriorating route to a gnode dst from a neighbor whom it wasn't using as a gateway to dst, will first probe its own route to make sure it is still up. For that, it will send a backward-paid Route Probe Packet (RPP) that contains a Randomly Generated Unique RouteID, and the gnode of dst.

The RPP is propagated based on the following rules:

- A node that fails to route the RPP will respond with a backward-paid Route Error Packet notifying the node of the failure
- Unlike other special packets, an RPP can be sent to a gnode of higher levels. Example, an RPP with a destination IP = 10.11.*.* will be consumed by any node of the gnode 10.11.*.*
- Each Node saves the RouteID and the Input and Output Gateways that were chosen for this route, effectively marking the return route (Virtual Path)
- To avoid flooding the network with RPP packets every time a link breaks, it is possible for nodes to withhold further RPPs sent to the same dst until it gets a response to the first one it sent. This could be done by caching the subsequent RouteIDs and creating its own response packets based on the response it got

- Likewise, RPPs sent to a dst node of a certain gnode level, can be grouped down with other RPPs sent to higher-level gnodes and withheld. If a route to a lower level node is reachable, surely a higher level gnode is reachable aswell
- When the RPP reaches the dst node (e.g. a member of gnode in question), the node must respond with a backward-paid Route Probe Confirmation Packet
- If the initiating node receives a Route Error Packet, or it fails to receive a timely response, it will assume that its route is broken. It will then update its routing table maxing the etx to dst, and start a flood of its own advertising the broken route
- Only if the initiating node receives a Route Probe Confirmation Packet will it advertise its better route and only to the node that just advertised the broken link

Admittedly, without a hierarchical subdivision, probe packets would quickly cause floods and saturate the links, which is why they would be impossible to use to resolve the count-to-infinity problem in other Distance Vector routing protocols, such as RIP or BGP. But our topology ensures that those packets will only be confined within their gnodes and therefore not cause a major problem. This needs to be tested and simulated though before a definitive conclusion could be reached. Any contribution to this problem is very welcome.

d) Useful Information Rule

A useful Topology Control packet is one that carries new or useful information.

The 'useful information' rule stipulates that any topology control information will propagate through the network as long as it is useful. A node that receives information will evaluate it then decide on whether to forward it or discard it, based on whether or not it found it useful. Note that if the information is not useful for the node, it is also not useful for any of its neighbors.

e) Tracer Packet (TP)

The TP packet is the basis upon which nodes build their routing tables. It is a packet that carries information about a single route (or tern) and propagates according to the useful Information rule.

Examples of a useful TP:

- A TP that contains a route to a new gnode, or one that was previously unreachable
- A TP that contains information about a change in the etx of a route that the node relies upon
- A TP that contains a better route to a gnode that was previously reachable through a less reliable link
- A TP that signals a broken link in a route that the node relies upon in reaching other nodes

Example of a useless TP:

- A TP that contains a route to a dst gnode that does not belong in the node's routing table (e.g. a node from the base-level n of a different gnode)

The TP is tasked with ensuring routing tables are built in such a way as to always favor the more reliable routes.

The idea is simple:

- A node G initially broadcasts a TP with a route to a gnode, along with its etx as per its own table to all of its neighbors
- A node N that receives a TP will subject it to the following set of rules:
    - If the gnode advertised by the TP does not belong in its routing table, it is useless. The packet is dropped
    - If the route is useful. Proceed with the following:
        - Is it the same route N was using? (N was using G as its next hop to dst)
            - Adjust the etx of the received route in its own table as per the received TP. Include the cost of the link between N and G
            - Forward the TP with the adjusted etx to all its neighbors except G
        - Is it a new route? (N was NOT using G as its next hop to dst)
            - Adjust the etx of the received route to include the cost of the route between this node and the gateway in question
            - Compare the adjusted etx with the etx of the one in the routing table
                - Is it better?
                    - Replace the old route with the new one
                    - Forward the TP with the adjusted etx to all other neighbors
                - Is it worse?
                    - Send a TP back to G advertising N as a better route

f) Extended Tracer Packet Flood (ETP)

The ETP is a packet that contains multiple TPs grouped together. As such, an ETP can be seen as a subset of a node's routing table. The useful information rule is applied to each TP independently; as such the ETP will start out as a large packet and will be trimmed as it propagates through the network, as each node filters out the useless TPs before forwarding what's left of it, until it is consumed completely.

7) The routing process

At this point, every node has built its routing table properly and is ready to communicate with the network. The routing process defines how a node can successfully establish a route to a destination in the network, and begin sending/receiving data.

The first thing a node will do is look up the IP Address of a given Peer Address. This will be covered later, but for now we assume we have an IP Address. The protocol requires that all nodes have bi-directional connections to their direct neighbors. The process is similar to Route Request and Route Responses in AODV in the way the route is found on-demand and marked along the path, but our route establishment process also lays the foundation for the 2-way payment strategy as well as the symmetric encryption of traffic.

a) Scout Packet (SP)

A Scout Packet is a backward-paid Packet sent by the Source peer to the Destination peer that serves two purposes:

1. Mark the route it took to reach the destination
2. Find the end-to-end hop count as well as the hop position of each node in the route, which will be needed for the two-way payment strategy calculation among neighbors

The SP contains the following information:

- Peer Address of the Source
- IP Address of the Source
- Peer Address of the Destination
- IP Address of the Destination
- A Randomly Generated Unique RouteID
- A number representing the TTL/Hop count
- A randomly-generated cryptographic key to be used for the stream, that is itself encrypted with the Destination node's Peer Address

The SP propagates as follows:

- Each Node saves the RouteID and Source Peer Address as well as the Input and Output Gateways that were chosen for this route, effectively marking the route.
- Each Node saves for the same RouteID and Source Peer Address combination its current hop position in the route. This information can be obtained by looking at the value of the TTL
- The TTL is updated
- The node Looks at the Destination IP and decides down which gateway it should be sent as per its routing table

Finally, routing of the SP obeys the following set of rules:

1. Every node along the route will cache the RouteID and mark the 2-way route. Future packets with the same RouteID will be sent down the same gateway
2. A node that receives a SP it cannot route will respond with another Backward paid Route Error packet that follows the same marked route
3. A node that receives a SP that is destined to its IP Address, but a different Peer Address responds with another backward paid Destination Error packet
4. A node that receives a Destination Error packet will flush the IP Address from its internal cache and initiate a new PLN IP Address Lookup query
5. Any node that has to forward a Destination Error packet will also remove the RouteID from its cache and unmark the route

b) Return Scout Packet (RSP)

A node that receives a valid SP will have to return an RSP, which will take the same marked route the SP took on its way to the node.

The RSP is a backward-paid packet sent by the Destination peer back to the Source peer that contains the following information:

- The same RouteID from the SP
- The total hop count as per the SP

The hop count can be signed by the Destination peer to stop the nodes along the return path from tampering with it.

Note that from this point on, IP Addresses are no longer needed since the routes have already been marked. This means that even if the IP Address of any of the nodes making up the path changes, including that of either the source or the destination, the route is not affected and the data stream is maintained.

c) Packet Stream (Data Payload Packets)

Once the route has been established, data transmission can begin. All data is encrypted with the symmetric key contained in the SP, making the communication secure.

The Data Payload Packets are two-way paid packets sent by the Source peer to the Destination peer and contain the following information:

- The same RouteID used in the SP
- The payload data, encrypted with the symmetric key from the RSP

A few things to note:

1. Any node that receives a packet with a RouteID it does not know will simply discard it
2. Every node along the path, including the source and the destination will keep count of all the packets they forwarded, sent or received throughout the stream
3. The route establishment process has nothing to do with TCP or UDP connections; it's strictly a Layer 3 protocol. As such, the return route is never used to transfer data or send Transport-Layer acknowledgment packets. If the destination wants to send data back to the source, it must establish its own route
4. If a node dies along the established virtual route or a link is broken, the neighboring nodes on each side of the broken link will send a Route Error Packet informing the communicating peers of the need to reestablish the route and prompting the nodes along the path to unmark the route

d) Close Stream Packet (CSP)

The source peer can decide to close down a stream anytime it wishes to. Ideally, the node may choose to end a packet stream if it has been idle for a set amount of time. Again, this has nothing to do with Transport Layer connections as a stream may be closed even if the underlying TCP connection is still alive, on the basis that no data is being sent.

The CSP is a two-way paid packet sent by the Source peer to the Destination peer and contains the following information:

- The same RouteID used in the SP
- Total count of packets sent by the peer

The CSRP propagates as follows:

- Each node will cache the count from the packet, and replace it in the CSP with its own count before forwarding it
- A node that receives a packet where the total count of packets is lower than its own counter can automatically flag the neighbor as fraudulent. This will result in future payment disputes

e) Return Close Stream Packet (RCSP)

In response to a CSP, the RCSP is a two-way packet sent by the Destination peer back to the Source Peer that contains the following information:

- The same RouteID used in the SP
- Total count of packets received by the Destination

The RCSP propagates as follows:

- Each node will cache the count from the packet, and replace it in the RCSP with its own count before forwarding it
- A node that receives a packet where the total count of packets is greater than its own counter can automatically flag the neighbor as fraudulent. This will result in future payment disputes.
- Each node will discard the cached RouteID signaling the end of this packet stream before forwarding the RCSP

f) Efficiency Updates

At the end of a stream, all nodes along the forward route should have the following information:

- Number of packets that each node has forwarded as per its internal counters
- Number of packets that each node should have forwarded as per its neighbors counters

It is therefore possible to know if packet loss has occurred based on the information above, and if it has, which link to either of the direct neighbors is responsible for the loss.

As for payment calculations in regards to lost packets, here are the main considerations:

1. The Source will pay for all the packets it sent
2. The Destination will only pay for the packets it received
3. Any node along the route is only paid for the packets it forwarded

4. The nodes on both sides of the link that has sustained packet loss will have to share the losses. They will also downgrade the quality of the link and let everyone know if only to avoid future losses

g) Changing IP Addresses

It is possible for either of the 2 nodes at both ends of the communication to exchange packets to inform the other peer of an IP Address change. Those will also be backward-paid and instruct the receiving end to replace the IP Address of the cached Peer Address with the one they just received. This eliminates further overhead and cost of PLN Lookup queries.

h) Packet Streams vs Multi-path routing

The route establishment makes it possible to implement the 2-way payment strategy, but also serves to eliminate the confusion caused by IP Address changes due to gnode migration, which may happen frequently. On the down-side, fixing the route may cause the links that make up the route to saturate during large file downloads or real-time data transfers. Multi-path routing where packets can follow different routes can arguably distribute the load more efficiently across the network.

Nevertheless, here are some things to consider:

1. Packets that follow multi-path routes will often arrive in random order, causing buffer delays for packet reordering at the transport layer, which often beats the purpose
2. It is possible to establish multiple routes between a Source and a Destination, and spread the data across multiple streams, therefore emulating multi-path routing
3. A stream does not have to last for as long as a TCP connection lives. Ideally, a source node could establish two routes and alternate between them, closing one and reopening it to make sure the streams are always using the best possible route
4. The best way to estimate the efficiency of the proposed protocol is to actually build, simulate and test it

i) Dealing with Loops

Although the routing tables are built and updated in a way to avoid loops, it is still possible that loops may still appear due to one of the possible conditions:

- One of the nodes along the route have a faulty routing table due to malicious tampering or erroneous cost function calculations
- A broken or deteriorating link has just appeared and the changes haven't had enough time to propagate among the affected nodes

The second case is not an issue as it should be fairly rare and won't cause any damage to the network as it may quickly resolve itself. The first case, however, could be used to sabotage the network and cause temporary localized damage.

As such, the first node that receives a Scout Packet with a RouteID that is already marked will automatically assume it is the neck of a loop and will proceed with the following:

1. Initiate a Route Error packet to inform the Source node of the problem
2. Max out the etx of the route in question and start a flood advertising the change

8) Peer Address Lookup Node (PLN)

We know that IP Addresses in our protocol are unreliable, since they can change at any time and cannot be used to accurately identify a peer. Peer addresses, on the other hand are unique to each peer. Higher Network Layers (Application, Transport) can use the Peer address as their reference and need not know the actual IP Address, which is restricted to the Network Layer, where our protocol functions.

The first step towards establishing a route to a destination is to actually find the IP Address of the Peer we're trying to reach. For that, a node needs to query a PLN, which will use its lookup table to return the current IP Address of the peer in question. The node can then cache the IP Address for future use, to minimize the costs and delays caused by IP Lookup requests.

a) Peer Lookup Node Zoning

Nodes will frequently have to communicate with PLNs to Lookup IP Addresses, or to update their IP Addresses every time they change. It is therefore very important for a node to have a fast and reliable connection to a PLN to avoid delays. As such, a node must choose the PLN with which it has the highest etx, typically the closest PLN.

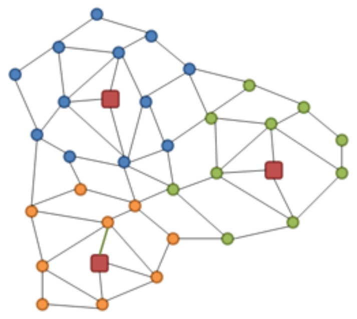


Figure 2: PLN Zones automatically adjust themselves optimally

The PLN Zoning process has the following objectives:

- Ensure the fastest routes possible from each node to the best available PLN
- Make sure new PLNs can join the network and get their fair share of the load
- Distribute the load optimally among PLNs

For this consider the following:

- When a PLN joins the network, it starts a PLN Advertise Packet flood advertising its existence, which contains the PLN's Peer Address and current IP Address
- Any node that receives a PLN Advertise flood will compare the advertised PLN with its current PLN:
    - If the advertised PLN is closer or has a better etx than the one it is currently using, it will switch to the new PLN and forward the flood to all its neighbors
    - If the PLN it is currently using has a better etx, it will discard the packet to end the flood, and send a packet back to the advertising PLN with the Peer Address, and the IP Address of the (better) PLN it is currently using
- Every time a PLN obtains a new IP Address, it initiates a new PLN Advertise flood to inform the nodes in its Zone of the change

- Every time 2 nodes meet for the first time (hooking or otherwise), they will exchange their PLNs. Again, peers may decide to switch their PLN if it has a better etx
- Any time a node switches PLN, it sends a packet telling its neighbors of its decision so that they may do the same

b) Peer Address Update

Any time a node obtains a new IP Address, it must register it on the network. For that it will reach its PLN and send a Peer Update Packet (PUP).

A PUP is a Backward-paid packet that contains the following information:

- The node's new IP Address
- The node's Peer Address
- A signature using the private key associated with the Peer Address to authenticate ownership

The PLN will update its lookup tables binding the new IP Address to the Peer Address in question and respond with another backward-paid packet to confirm the update. This effectively renders the Peer Address Update query free of charge, while forcing peers along the route to forward the packet, to avoid absorbing the cost.

c) IP Address Lookup Request

A Source peer that needs to initiate communication with a Destination peer will first need to look up its IP Address. The main job of a PLN is to translate Peer Addresses into IP Addresses. This is also the main source of profit for PLNs.

The IP Address Lookup Request Packet is a backward paid packet with a cost of 1 that contains the following information:

- IP Address of the Peer Initiating the request
- Peer Address of the node to look-up

The PLN will query its lookup table and return the IP Address of the peer in question.

The IP Address Response packet is another backward-paid packet, but its price is much higher than other backward-paid packets. The appropriate value will have to be estimated later, in such a way as to provide the PLN with enough profit to keep them motivated in offering the service, without it being prohibitively costly for the other nodes.

d) Inter-PLN communication

PLNs need to maintain their lookup tables by keeping them up to date with all the Peers on the network. But since nodes only communicate with a single PLN as per the PLN-Zoning protocol explained above, it is necessary for PLNs to exchange IP Address updates among each other.

For that, inter-PLN communication is not based on special-packets, and is done using an application-layer TCP connection that PLNs establish between each other. Inter-PLN traffic is therefore paid traffic, just as any other traffic on the network. It would be unfair to the nodes that happen to be close to a PLN to have to route PLN traffic for free.

Some notes:

- When a PLN joins the network, and broadcasts a flood which will travel to establish the PLN-Zone as described above, and will return with the Addresses of the PLNs of the bordering PLN-Zones. Those packets can be consumed as UDP packets and used to establish a TCP connection with those neighboring PLNs
- A PLN can initiate a new flood to discover new neighboring PLNs and maintain stable connections with as many as possible
- The more the PLNs in any given neighborhood, the smaller its zone and the lower the profit. This may force PLN owners to shut them down or relocate them in such a way as to keep the network balanced
- The PLN system is arguably the weakest component of Open Libernet as it offers potential for abuse, even if the damage is localized
- A PLN may choose to maliciously and intentionally not update its tables by not communicating with other PLNs. This is why there needs to be a mechanism to alert the network to limit abuse, and that's why we define the PLN Reputation system

e) PLN Reputation

The PLN Reputation system is designed to punish lazy PLNs that fail to keep their tables up to date by not exchanging data with other PLNs as often as they should, by gradually reducing the size of their zones.

The idea is to introduce a new fidelity measure for the PLN known as the PLN's Reputation.

The proposed solution obeys the following rules:

- Every time a PLN returns an expired or invalid IP Address in response to an IP Address Lookup Request, the requesting peer will downgrade the PLN's reputation
- When a peer is offered an alternative PLN by a PLN Advertise Flood, it will consider the PLN's reputation along with the etx before making the decision of whether to switch or not, based on a weight function that will be defined
- The lower the PLN's reputation, the more likely it is that nodes at the border of the PLN Zone will escape, the smaller its zone and hence, the lower its profits

9) Payment System

The payment system is based on the bitcoin protocol. We won't go through the actual mining process, but will only explain the special packets that allow nodes to execute queries on the miners.

First a node will choose a miner whose peer address could be manually configured. Then the node proceeds with a normal IP Address Lookup Request to find the Peer node's current IP Address. From then on, packets can be sent to the miner normally.

a) Check Balance

A node may choose to check the balance of a neighbor so it can decide whether or not to accept traffic originating from that neighbor. But a mining node may not be enticed to honor such requests without proper motivation.

A check-balance packet is therefore a special backward-paid packet with a cost of 1 that contains the following information:

- The querying node's IP Address
- The Peer Address whose balance is being checked

The response is another backward-paid packet but priced at higher than 1 that contains the following information:

- The Peer Address whose balance is being checked
- The actual balance in question
- A cryptographic signature on the peer address , the balance and a nonce to keep the packet from being tampered along the route

A node may request to check its own balance using the same procedure.

b) Request Payment

When the balance of a node's balance hits a preset threshold, the node will request a payment. For that it needs the neighbor's approval. A special backward-paid packet is sent to the neighbor in question that contains the following information:

- The IP Address of the node requesting payment
- The Peer Address of the node requesting payment
- The IP Address of the node that is being demanded
- The Peer Address of the node that is being demanded a payment
- The requested amount
- A Transaction ID that uniquely identifies the payment
- A cryptographic signature on the peer addresses, the requested amount and a nonce to keep the packet from being tampered along the route

Ideally, the two nodes should be directly connected. But in the possible cases where a physical connection with a demanding node breaks, it is still possible for payment requests to be routed as any other packet.

c) Signed Payment

A node that receives a payment request will first validate it against its own accounting books. If the payment is deemed valid, the node will sign the payment request and send the packet back.

Some points to note:

- It is possible for the accounts not to match exactly. The discrepancy is likely due to packet loss, and not necessarily malicious tampering. A discrepancy tolerance ratio could be established here to avoid disputes as much as possible
- After a payment is signed, a node may update its accounting book for the peer in question
- A node that receives a signed payment will also update its accounting book for the peer in question
- It is possible for the paying node to cache the TransactionID and the amount in case it was asked to sign it again following a lost packet

d) Declare Payment

The last step towards successfully receiving a payment is declaring the signed transaction with a miner. This is done through a special backward-paid packet to a miner of choice. The miner will respond with a backward-paid confirmation packet. A Payment Declare packet may be resent until a confirmation is received.

e) Dispute resolution

In the possible case where a payment request is received that does not match with the node's accounting books, or falls outside the bounds of the accepted discrepancy tolerance, the 2 nodes enter into dispute.

The reasons for such an event to occur include:

- Malicious manipulation of the protocol on the node requesting payment
- Malicious manipulation of the protocol on the node itself
- Extreme packet loss sustained by the link connecting the two neighbors

All 3 cases above call for the link to be terminated. In any case, a node that does not receive a due payment will terminate the link with the offending neighbor. A malicious node will quickly find itself ostracized from the network.

It is possible for the node to provide an optional manual dispute resolution system (could be part of the router's configuration panel) where the node's owner may choose to override the default procedure and pay the due amounts to bring the broken links back.

f) Inter-miner communication

Inter-miner communication is based on a peer to peer protocol that functions on application layer. Miners will exchange blocks just as with bitcoin. Miners do not need incentive to exchange this paid-traffic as with PLNs, since failing to do so will cause the blockchain to split and will result in their hard-work to be discarded, unless they command the majority of the network's processing power.

10) Tunnel Nodes

The proposed design of the tunnel nodes is straightforward. A tunnel node will first establish Peer to Peer connections to other tunnel nodes using an internet connection. Ideally, the tunnel will choose a few other tunnel nodes based on their geographical locations. Each tunnel could build a route to 2 other tunnels per continent.

A tunnel node will absorb every packet and send it up the network layer stack as UDP traffic, which will be consumed at application layer and sent down the appropriate p2p connection.

XI. Transaction Processing (Mining)

The transaction processing system is very similar in function to that of the Bitcoin peer to peer currency. It is handled by special "mining" nodes that function at application layer. We will not go into much detail on how exactly transaction validation is processed, or how the blockchain is maintained by the mining nodes, as it is clearly demonstrated in theory and in practice by the bitcoin payment system. We will only list the proposed differences between our payment system, and that of the bitcoin protocol as conceived by its creator.

1) No artificial scarcity

Unlike bitcoin mining, where the maximum bitcoin supply is capped at 21 million, and where miners are rewarded with a pre-set total amount of bitcoins per block, Open Libernet traffic mining does not impose such restrictions. Our traffic supply keeps expanding and miners will perpetually be rewarded for their efforts. Also, the total amount of traffic introduced into the economy as reward to the miners is linked to the total amount of traffic verified in the block. This means the traffic supply is increased proportionally to the total traffic handled by the network, hence the network capacity. This ensures a steady supply of traffic to cope with the increase in network capacity, and a perpetually increasing abundance, which translates into a steady decline in price.

2) Mining difficulty and Block Size

Mining block difficulty in our system is not set at 10 minutes such as in Bitcoin. The fact that we don't mind transactions taking hours to be validated and verified means we can get away with much larger blocks and much higher mining difficulties. Again, this can be justified by the fact that traffic is not a currency and should not be regarded as barter in a payment system.

3) Transaction fees

To curb the use of traffic as a currency, which would add unnecessary stress on the mining nodes, overload the network with payment packets and encourage people to hoard, transaction fees are deliberately kept high. Unlike the bitcoin protocol where fees are optional, payment nodes claim a set percentage of the contents of any transaction they process.

XII. Domain Name Registry

The domain name registry system of the current internet is fundamentally flawed. It suffers from most of the inherent problems of centralization. Unfortunately, without a centralized authority to authenticate and guarantee ownership of domain names, very little can be done to protect internet users from fraud and impersonation attacks. This problem falls beyond the scope of our work, and we will ignore it for the time being. DNS servers can be deployed on top of our network's application later with no modification. It may be best to pick our domain name governing authority wisely, and make sure serious measures are taken to limit abuse by domain name parking services, registrar front running and copyright infringement.

XIII. Risks, Problems and Limitations

The system at its current state suffers from some problems that we were able to identify. It is possible that other problems may arise in the future, as the development of the project begins. Contributions to finding potential problems or security holes in the system are most welcome.

Nevertheless, here are the weaknesses and limitations that we feel the system is most vulnerable to:

- The protocol does not support mobile nodes. This is fine though, as mobile ad-hoc mesh networks are too complex and do not scale well. The intentional delay for Layer 2 Link establishment should solve the problem of churn caused by joining and parting nodes and should keep mobile nodes out. Mobile nodes could still use the NAT of static nodes to connect to the network
- The modified Distance-Vector protocol used may not be the best in terms of speed of route convergence and may have to undergo changes and optimizations
- The PLN system is probably the weakest part of the system and is vulnerable to fraud and denial of service. Eavesdropping is not an issue, as Peer Addresses cannot be used to identify the actual person: they're random and they can be changed. But the fact that nodes cannot simply choose which PLN they wish to communicate with (since they can't keep track of the PLN's changing IP Address) and are forced to use the one that is being used by their neighbors makes abuse more likely. Ideas or suggestions on how we can overcome this limitation are most welcome
- Layer 3 Encryption/decryption may be too expensive on low-end routers and access points and could cause delays. For that we may have to drop such devices and opt-in for custom devices. A proposed solution is building our own linux-based router using Raspberry-pi boards with wifi and Ethernet cards. Old computers running linux are always a good alternative
- A subset of the mesh network may occasionally find itself disconnected from the rest of the network should a tunnel node die. This may cause a split on the miners' blockchain. As such, it is best if miners were deployed the closest possible to a tunnel
- Peer Addresses are 128bit long, which makes software that do not support IPv6 unsupported by the network

XIV. Proposed Plan

Much remains to be done in terms of refining the payment system and the routing protocol, and closing down all gaps on abusers and malicious nodes. For that, we rely on community support. We will take into account any feedback, opinion, suggestion or critique, that could help point out potential problems or improve the protocol.

Once the final protocol has been defined, we will begin with the development. Support from the developers' community is also welcome at this stage. For the payment system, we will start a fork on Bitcoin and make the minor modifications we proposed. The rest of the protocol may have to be developed from scratch.

The kick-off phase of the network is crucial. First we need to install at least one PLN, a DNS server and a Mining node. We also propose to mint a few Petabytes of traffic in the genesis block of our mining system, and put it out for grab in special Traffic faucets to provide new-comers with free access. It may be a good idea to keep special faucets set up where people can donate traffic that will be distributed among needy peers.

Also, we will need to set up some free services to break the vicious cycle of usability of our network. Such services could include a messaging service, a telephony/video-conferencing service, mirrors to some popular sites such as Wikipedia, Youtube, etc… All in good time.

Finally, we would like to hear what everyone thinks of the work done here. Any feedback is welcome.

CHANGE LOG

- **2/18/2014: Fixed Layout and typos; added draft status**

References

[1] Bitcoin: A peer to peer Electronic Cash System – Satoshi Nakamoto https://bitcoin.org/bitcoin.pdf

[2] Netsukuku – http://netsukuku.freaknet.org/

[3] CJDNS – https://github.com/cjdelisle/cjdns/blob/master/doc/Whitepaper.md