FREESKY

FREE INTERNET WHERE YOU NEED IT

Table of contents

Introduction

We offer a new concept of an Internet service provider in the market and return power to people. Our new product, a decentralized community-driven mesh network, allows users to monitor the network infrastructure.

Today, Internet access networks are owned by centralized, usually exclusive, providers. Internet service provider controls all routers and network equipment and charges a monthly fee for this service. Most areas have only one or two options. The market does not offer any reasonable alternative to this mechanism. The only solution is the creation of a new Internet provider, large enough to be an independent business and competitive with the other giants. It is not a cheap variant.

Freesky makes it possible. Our team supports the creation of new networks created by communities and for communities based on our routers, which pay each other for bandwidth. It allows people to create "decentralized Internet providers" in their communities, as well as easily manage and develop them.

Freesky advantages:

- The overall cost reduction as the nodes are able to select the route with the best characteristics of reliability, bandwidth and minimum value.
- Those who are using nodes, do not need to pay for additional marketing and advertising services. All the promotions in this system

are default announcements about the price and quality of the route between nodes. It facilitates the device use for the new participants

• All billing costs are generated automatically, and additional payment expenses are considerably reduced thanks to the blockchain-based payment system.

Network Overview

Let’s review the Freesky network architecture in more detail. In order to make its structure easier for understanding, we will explain several roles that this appliance can fulfill. Bear in mind that these are theoretical capabilities, not physical ones. So, one device can successfully cope with a few or even all of these tasks at the same time:

・The user nodes are set up by the users who want to purchase Internet access from Freesky. You may associate the user node with a router or a modem that is traditionally used by all Internet providers. But its unique characteristics is that it does not depend on any provider. User nodes provide traffic from other devices to the Freesky network. In other words, they form WiFi access points and wired LAN ports.

・The relay nodes are designed for people who want to get some profit by rerouting Internet traffic. These nodes are very powerful and can

be set up in favorable places with a good reach to other nodes.
Many of them will perform the function of both relay and user nodes.
You will need specialized software to buy and sell bandwidth.

・The gateway nodes are actually relay nodes. Their additional ability is they can connect to any source of cheap Internet bandwidth. It can be any Internet exchange point or any business class connection from a regular Internet provider. But they do not have any legal responsibility for the traffic they provide on the network with the output nodes.

• Output nodes are not always a part of the local physical network, but they can be located in a data center accessible via the Internet. They are connected to the gateway nodes through VPN tunnels. These nodes enable the users to check the quality parameters distributed by nodes in the network. It allows you to automatically select gateway nodes by routing protocol. Actually. the output nodes replace an Internet provider to some extent. They convert network addresses to route requests on the public Internet and process copyright

complaints, etc. Thus, the gateway nodes perform the function of pure bandwidth providers without any legal risk for the use of their services.

Routing Overview

We have used the Babel routing protocol for our development. It was chosen thanks to its beneficial features that perfectly match our purposes. But it is possible to modify any distance vector protocol according to the Freesky requirements. These protocols are being extensively used on the Internet. And BGP is one of them. The announced connection quality metric defines the routing in vector distance protocols. The nodes send a signal or a so-called announcement packet, advising of their identity and availability on the network.

Then these ad packages are being spread between the sites. Each node updates the metric based on the signal received from the closest point. This data enables each note to create a routing table of the best neighboring connection points to reach and destination on the net in just O (n) time for each node.

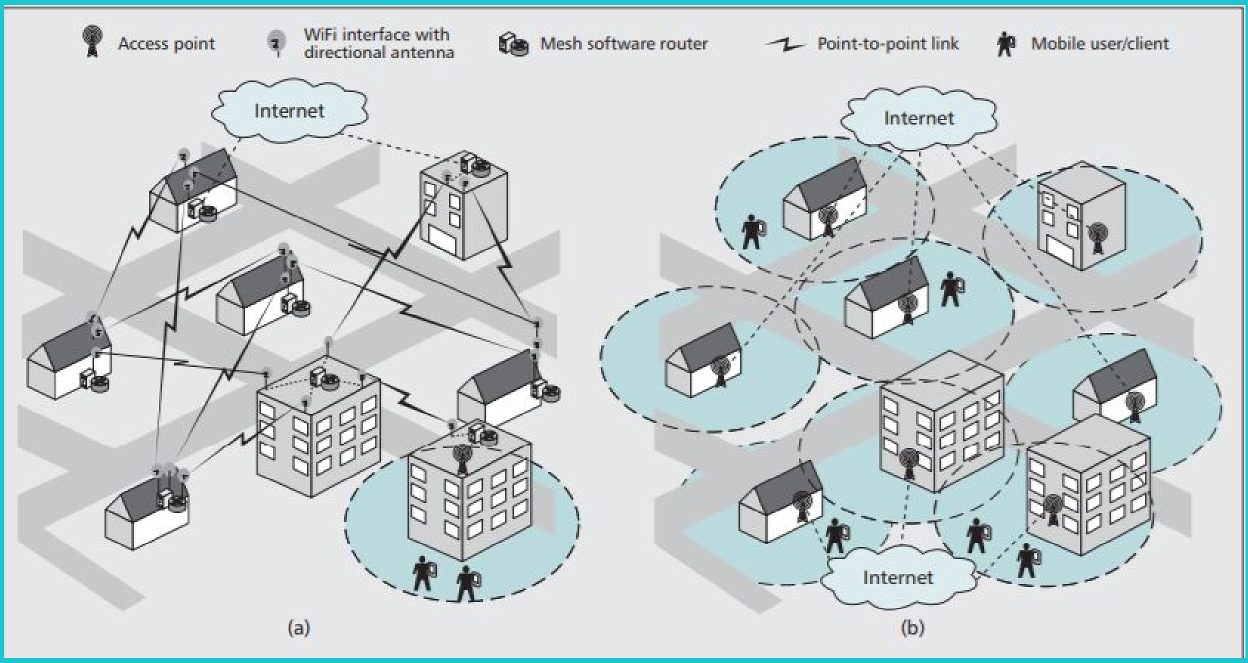
We offer additional features:

• Verifiable quality metric.

• Price assessment.

Quality metric is the connection parameter which is established by the host and the destination to which it sends signals. The Babel protocol enables nodes to identify the metrics announced by the neighboring points.

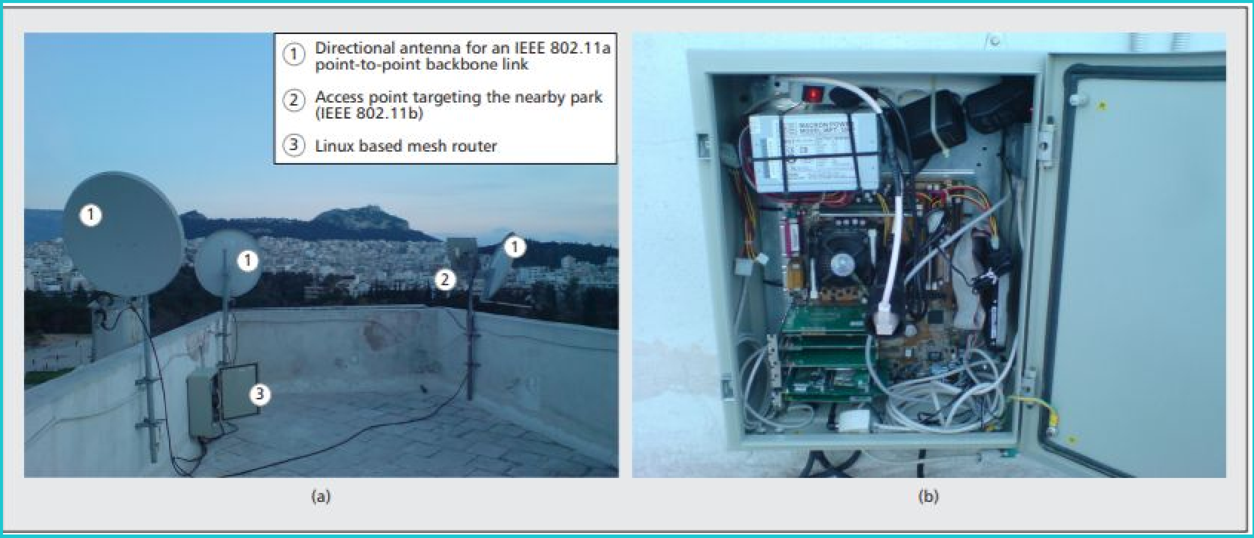
Price information is added to the routing announcements as the second metric. It is actually value for the agreed amount of transmitted information within a certain period of time. Every node updates a price bid every time it passes the announcement. Then it is possible to select the most appropriate routing from the point of view of the price metric.



Implementation

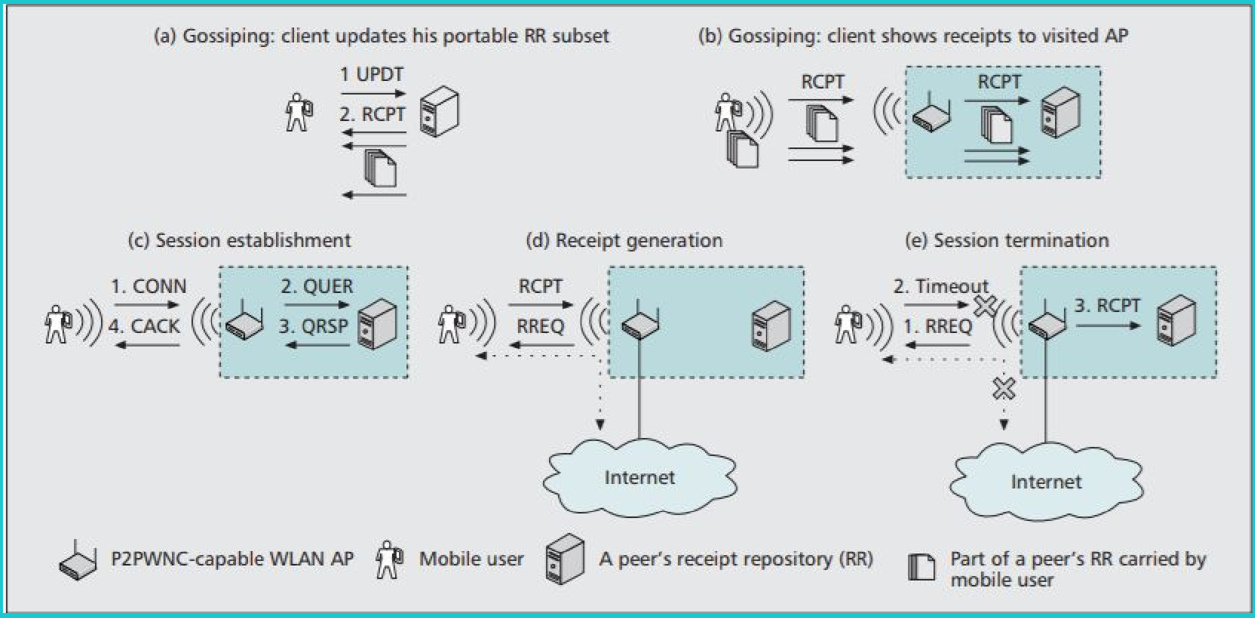
The whole mechanism is built on the Babel routing protocol. It is a vector distance protocol known by its high performance characteristics. Every distance protocol works using a distributed Bellman-Ford path algorithm. Initially, nodes are establishing test connection with the neighboring points. It is a so-called link cost. The next stage is to check what other destinations they can reach.

Every node gets the information about the potential destination, then it links it to the data of the neighboring communication channel from where the information was obtained.



1 Directional antenna for an IEEE 802.11a point-to-point backbone link
2 Access point targeting the nearby park (IEEE 802.11b)
3 Linux based mesh router
1
1
1
2
3
(a)
(b)

This analysis is defined as the "route metrics". Overall, it describes the performance parameters the nodes are capable of. The neighboring point which shows the best results for this destination is chosen as the next transition point. All data sent to this destination will also come through this neighboring point. The Babel protocol allows to add and remove routes from the Linux kernel routing table.


(a) Gossiping: client updates his portable RR subset
1 UPDT
2. RCPT
(b) Gossiping: client shows receipts to visited AP
RCPT
RCPT
(c) Session establishment
1. CONN
4. CACK
2. QUER
3. QRSP
(d) Receipt generation
RCPT
RREQ
Internet
(e) Session termination
2. Timeout
1. RREQ
3. RCPT
Internet
P2PWNC-capable WLAN AP
Mobile user
A peer's receipt repository (RR)
Part of a peer's RR carried by mobile user

Its specification reads the following:

- Similar to other routing algorithms, Babel estimates the value of connections between two neighboring points. These abstract values are linked to the edges between two nodes.
- The route metric between the nodes is actually the sum of the values if all these edges that are met on the route. The main task of the routing algorithm is to find all the possible routes to a certain point with the lowest metric parameter.

Currently, all distance vector protocols have a lot of weak points. The information of the channel cost and other route metrics cannot be rechecked in any way. Thus, practically every node can be treated as the best option for any destination. Actually, it is not an issue, as nearly all networks belong to one owner. But at Freesky, all nodes belong to different owners, and each of them is trying to prove its unique characteristics. This weak side should not be neglected, as it can be used a tool to get business predominance of one node over the other.

We have worked out a modified model of the Babel protocol. Its main idea is to add a “verifiable metric” to its parameters. What is that? It is the metric of a distance or “testable metric” that remains unchanged for individual nodes.

To demonstrate it, let’s take round-trip time (rtt). This indicator is calculated by summarizing along the route or it can be estimated by the source and destination. With a variable metric, you can avoid the fraudsters and prioritize route check packets. The obtained metric will be close to the general metric advertised for the destination by the neighbor node that is currently sending packets to the final destination. It allows us to state precisely the correctness of the chosen routes. In case of some discrepancies between these metrics, the accuracy indicator of the neighbor will be changed. In this case, the metric is going to be corrected by this neighbor.

We have made a lot of tests to verify the reliability of the routes claimed by neighbors. But we had to choose the method identifying what routes should be checked. We use a timer to for verification purposes. The procedure for each node consisted of the following steps to select the route r to verify:

• Choose all destinations that were engages in the last x seconds. X parameter can be fixes or can changed to monitor the control the focus of the testing.

• Then specify destination d from this set randomly.

• Get a set of all routes that are possible and have a destination d.

• Specify a route r from this set in random order.

Accuracy check. After obtaining a new verified route metric, we Upgrade the neighbor s accuracy score making the following steps:

```
d = destination
n = neighbor
c = route cost over tunnel to d
m = route metric advertised for d by n
A = accuracy scores of n, indexed by destination
S = lowest allowed average accuracy score
a = minimum (m / c, 1)
A [d] = (A [d] + a) / 2
s = average (A)
if s <S
n's routes removed from routing table
```

How to calculate the accuracy score for each destination? It is considered as the proportion of the value of the verified checked route c to the route metric m advised by neighbor n.

Then it is averaged with the current accuracy indicator for this destination

A [d] and A [d] is set to a new value. Then we take s the average value of all estimates of the accuracy of the active assignment for the neighbor. If s drops below some custom value of S, neighbor routes are eliminated from the kernel routing table.

Configuration of route metrics. The above approach is efficient for identification and deletion of chronically inaccurate neighbors. Besides, it is also a good source of a stream of valid route metrics. Then these metrics can be used to upgrade the routing table even before this inaccurate neighbor is deleted. After obtaining the cost of route c above, it is possible to start using it instead of the declared neighbor metric when choosing routes. We continue to use c for a long time D.

What value to set for D? If D parameter is very short, then the nodes will keep trusting inaccurate metrics that they had to amend. If D is too long, the nodes will skip legitimate updates about new updated routes.

To achieve the right balance, an exponentially longer latency will be used to correct bandwidth beyond a small tolerance. The node that was engaged in the correction will calculate the duration d, how long it has performed the correction for this route and the correction size s. When participating in another correction on the same route, the duration D of applying the bandwidth correction will be determined as:

$$D = (C1 / d) \wedge C2$$

Where C1 and where C2 constants must be adjusted and hardcoded.

Because of this formula, it will take much longer to return to using the announced neighbor metrics if the advertised metrics needed to be recently corrected and / or require a big correction.

We also need a way to distribute the price for each route and take this price into account when deciding on forwarding. Babel already has a mechanism for adding arbitrary "external sources of readiness." This works when the nodes add a number to the metric they calculated for the route. This does not work for us for two reasons:

First, the route metric is checked in our system. An arbitrary change may violate this check.

Secondly, the price should be different from the route metric, because it will be used to calculate the amount of payments. For these reasons, we use a separate price metric.

The requirements of this price metric are simple. It consists of an additional 16-bit price field in each Babel update and in each route in the node's routing table.

As service packs are distributed across the network, each node increases the cost of the route by a certain amount. The easiest way to determine how much you need to add to the route price, using the constant set by the node operator. However, there can be many different types of automatic pricing algorithms to adjust prices based on demand or competition.

The Babel route selection procedure has been expanded to take into account this price field. Instead of selecting routes based solely on the route metric, the extended metric m 'is calculated with

$$m' = n * \log2(m) + \log2(p)$$

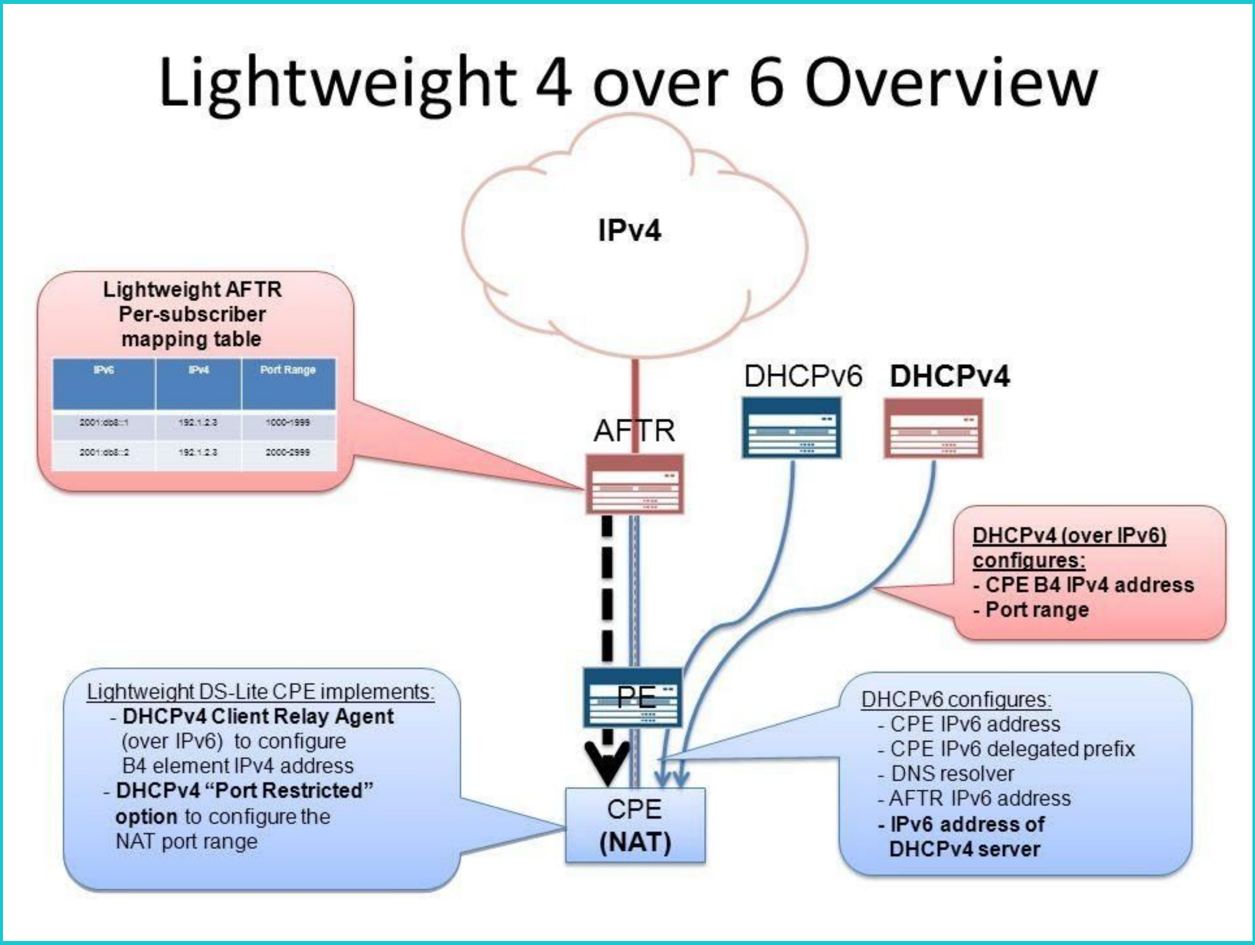
Where m is the route metric, p is the price, and n is a constant factor. Routes are then selected based on m '. The logarithmic function is chosen to normalize, so increasing or decreasing the metric or price will change the advanced metric by a constant value, regardless of their absolute values. By adjusting n, the nodes can determine how much weight to give the price in the calculation. In other words, the node will switch to the connection, which is 2 * n times more expensive per byte if its rate is 50% lower.

One could even populate several routing tables with routes selected with different n values and distribute routes from these tables under different router identifiers. This or a similar mechanism can be used to allow neighbors to choose from a range of price-quality trade-offs.

We selected nodes for payment after receiving the service. This is due to the fact that if the nodes pay earlier, the malicious node could re-accept the payment and not provide services (perhaps changing the identification data each time). They can accumulate their dishonest incomes and earn money with this strategy. On the other hand, if the nodes pay after receiving the service, malicious nodes that constantly receive the service and do not pay will receive only a bad connection (you cannot save the stolen bandwidth).

Authentication

The main idea of the created system is that nodes pay for the traffic. But what if they fail this operation? It makes it necessary to find a method to control which neighbors use the Internet access. We need cryptographic authentication in this case to avoid falsifying of the MAC address. For the radio channels, WPA authentication method is effective. But what solution to find for wired channels?


Lightweight 4 over 6 Overview
IPv4
Lightweight AFTR Per-subscriber mapping table
AFTR
DHCPv6
DHCPv4
DHCPv4 (over IPv6) configures: - CPE B4 IPv4 address - Port range
PE
CPE (NAT)
Lightweight DS-Lite CPE implements: - DHCPv4 Client Relay Agent (over IPv6) to configure B4 element IPv4 address - DHCPv4 "Port Restricted" option to configure the NAT port range
DHCPv6 configures: - CPE IPv6 address - CPE IPv6 delegated prefix - DNS resolver - AFTR IPv6 address - IPv6 address of DHCPv4 server

IPv6	IPv4	Port Range
2001:db8::1	192.1.2.3	1000-1999
2001:db8::2	192.1.2.3	2000-2999

Wireguard can become an effective tool for this purpose. It is encrypted and authenticated tunneling software. We suggest to add some optimization to it for future. It will be needed to include authentication data in the IPv6 header extension and not to encapsulate tunnel packets.

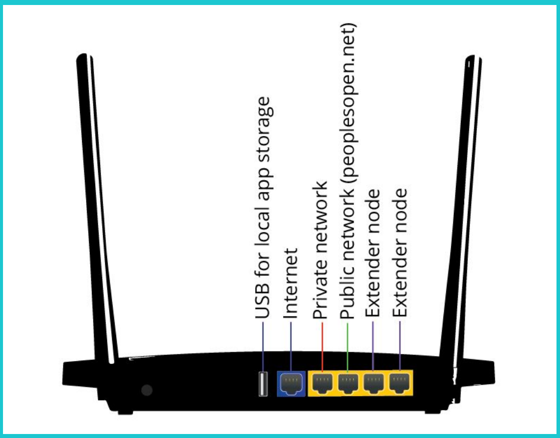
Nodes form tunnels with every neighboring point. Thus they can monitor the traffic in each channel, depending on the payment. With Wireguard, you can create a virtual interface for every tunnel. These virtual interfaces are then combined together to another virtual interface to launch Babel. So if payment from a neighbor stops, its packets are blocked by the firewall and are no longer supported.

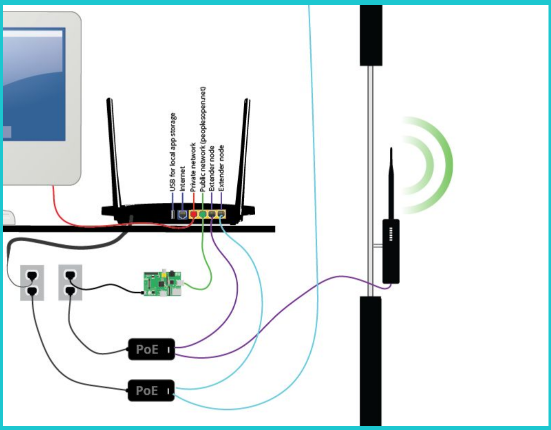
The same algorithm is used to make measurement from one output node to the other user node. There is also a Wireguard tunnel from the output node to the user node. It is used for privacy reasons and as a way to use IPv6 in a mesh network, like Lightweight 4over6. This tunnel provides a good reference point for the exit node to block the service for the user node in case of non-payment.

Network

Freesky performance is based on pay-for-forwarding principle. If all the elements of the algorithm work efficiently, you get a network where nodes pay to each other for the data transmission and verify the correctness of their performance. Let’s try to analyze how this network can serve as a provider of Internet access.

To make the whole system work, we need at least one host. And it is expected there will a number of such hosts. They are all called gateway nodes. These nodes establish connection with exit nodes via the tunnels.

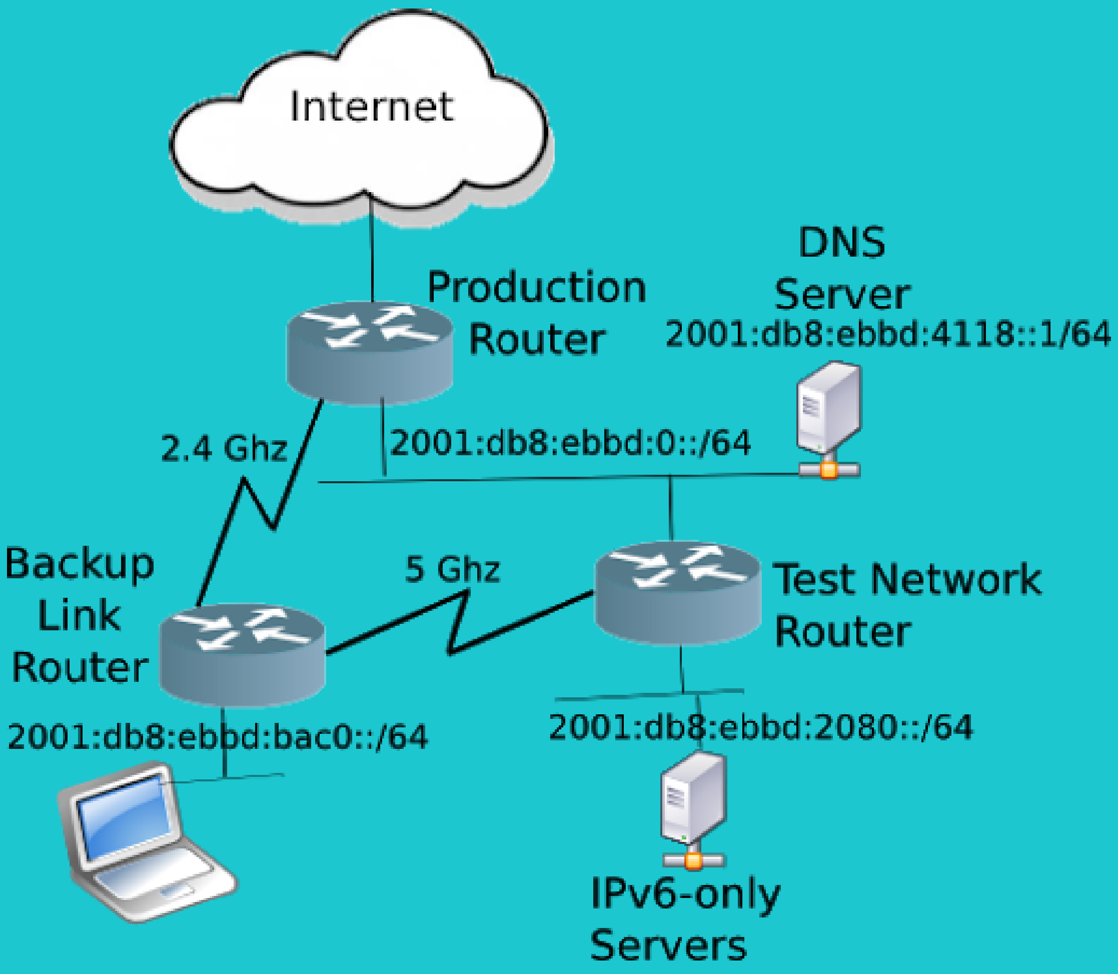




The tunnel builds the virtual interface for Babel. In its turn, Babel reaches the destinations through tunnel connections in the same manner as it would do through real connections. So, the nodes do not have to choose any explicit gateways. The gateway nodes set a price and get paid for routing to the output nodes. So the user nodes travel through encrypted tunnels to selected nodes and reach internet access. And in this situation, the gateway and relay nodes can see only the encrypted traffic between the user and final destination.

Output nodes

These nodes fulfill the same functions as every Internet provider. The only difference is portable packages. It makes it possible for the other nodes to concentrate on the connection. And output nodes het payment for connecting Freesky with the global Internet. Output nodes serve to merge Freesky into one unity with modern Internet and society. You just need to establish connection and the nodes will do the remaining job.



Main functions of output nodes:

Interaction with public IP addresses. Output nodes have public IP addresses and use them to trace traffic for connected user nodes to and from the Internet. They serve as NAT for user nodes or supply them with

public IP addresses (it happens in the cases when the user nodes themselves perform NAT like in Lightweight 4over6) Protection of Encrypted Tunnels. All traffic between user nodes and output nodes is secured in the tunnel and encrypted. It means that Freesky network users should only trust the exit site's history of visited pages and unencrypted traffic. Neighbors will not have access to any additional information.

Route verification. Our Babel extensions enable nodes to check routes between themselves and their destination. The output nodes are the destination for the outgoing traffic of the user node, and the user nodes are the destination for the return traffic. User nodes and output nodes work together to ensure that smooth nodes performance in the Freesky network. User nodes implicitly trust the output nodes to accurately verify the route.

Take legal considerations into account: we want someone to install the gateway node as easily as possible. In part, this frees them from any legal problems associated with the use of their connection. Any legal complaints regarding the use of the Freesky network can only be directed to the exit nodes, as they are the only ones that direct traffic to the Internet and make it visible to the whole world. Gateway and relay nodes always see only encrypted packets.

Payment for traffic return. User nodes pay their neighbors to forward traffic to the output node they use and to the Internet. And who pays for the return traffic? User nodes send the output nodes a certain amount of money, which the output nodes use to pay their neighbors for the traffic return.

Whitepaper

FREESKY