Individe

A general-purpose mesh networking protocol stack based on IPFS

DRAFT v0.1.0, October 2022

Aleksa Milošević
aleksa@endlesshorizons.xyz

ABSTRACT

Individe is a wireless and wired ad-hoc mesh networking protocol taking a new approach to mesh networking, by leveraging a peer-to-peer distributed file system. Its main purpose is to improve the speed and efficiency of mesh networks, providing the world with simplified technology stack needed for networking, without tradeoffs. It provides a cheap, easy, fast and reliable way to set up large and small networks with a minimum of very little to no infrastructure (depending on the network size), that can replace the Internet, but also work in parallel with it, that is resilient to disasters and infrastructure failure and completely free for everyone, being the ideal solution for regions that do not possess the capabilities to create or fund infrastructure. Individe achieves better speeds and bandwidth throughput as a mesh network, by drastically optimizing content retrieval through distributed content-addressed block storage mechanism, known as Interplanetary File System on top of WANET/MANET routing protocol(s). This paper describes Individe protocol that fuses IPFS on top of WMNs (wireless mesh networks) and discusses, the principles that make this fusion of technologies so robust, the optimizations it introduces and possible future improvements and directions of the protocol worth implementing and exploring.

1. INTRODUCTION

To date, we've seen several successful general-purpose mesh networks that route Internet Protocol (IP). Some of the more notable ones are Guifi and FreiFunk. These projects have successfully created a network of routers, antennas and user devices, that work with no configuration and without the central point of failure. Wireless ad-hoc mesh networks, proven to work, unfortunately, end up routing everything back to the Internet. So, even though they're not centralized, they serve just as a more convenient way to provide Internet access. Without Internet access, mesh networks remain useless. Their purpose, although maybe not living to its potential, is still just routing the traffic, at which they've gotten quite good and robust over the previous years.

Here, we propose an innovative way to employ mesh networks to create a completely decentralized networking solution, that would redistribute the Web and create a novel way of using it. Interplanetary File System (IPFS) has created a way to distribute data in a peer-to-peer manner using content-addressed block storage, which is a quite convenient way of distributing data in decentralized networks, distributing it more and more, evenly through time.

This paper introduces Individe, a wireless-first ad hoc mesh network based fused with a peer-to-peer distributed content-addressed file system aiming to allow for a fully decentralized general-purpose mesh networking protocol that can work as a standalone solution, with no configuration and no infrastructure, but also alongside Internet Protocol, making it resilient to natural disasters, easy to setup everywhere on a large or small scale, but not depriving it of Internets absurd amount of data.

2. BACKGROUND

This section reviews important properties of protocols, which Individe combines.

2.1 WANET, MANET or WMN

WANET, short for Wireless Ad-Hoc network, MANET, short for Mobile Ad-Hoc network or WMN, short for Wireless Mesh network is a type of decentralized wireless network. The network is considered ad-hoc (as is), since it does not rely on pre-existing infrastructure and configuration, therefore, not requiring routers, access points, routing hubs, Internet providers, etc. Instead, each node (user or device) participates in routing data within the network, by forwarding data for other nodes. Determination of which node forwards the data and how it is forwarded is made dynamically via the so-called routing algorithm or protocol.

Some of the most popular and most mature mesh routing protocols are Babel, B.A.T.M.A.N, B.A.T.M.A.N Advanced (B.A.T.M.A.N-adv), BMX6 (and BMX7), OLSR, AODV, 802.11s, HWMP and Static Routing. Each of them employs different tactics on how to route the data, how to distribute routing tables, how to recover the routes and the network, when a certain node is removed, moved or becomes out-of-range and how to discover malicious and faulty nodes.

In general, routing algorithms and protocols currently available are designed with a single goal in mind. Routing traffic to and from the Internet gateway (a mesh node connected to the Internet), to expand the Internet coverage or to create small ad-hoc local wireless networks. Currently, mesh networks tend to be slower than traditional ones and have decreased bandwidth, but for their intended purpose, they are working pretty decently.

2.2 Interplanetary File System (IPFS)

IPFS[11] is a peer-to-peer distributed file system with increased mainstream adoption aspiring to re-decentralize the way the Internet operates. It is a content-addressable network that combines successful protocols from other peer-to-peer systems, but also evolves them providing a single

cohesive system. It uses Distributed Hash Tables (DHTs) as a lookup service and as a routing table to find the stored data, a BitTorrent-inspired protocol to exchange the data and a content-addressable way for storing data inspired by Git's Merkle DAG.

For a better understanding of IPFS, I suggest reading the IPFS whitepaper written by Juan Benet[11]. But, here is a quick rundown of basic principles.

2.2.1 Identities

Nodes on IPFS are identified by a NodeId, the cryptographic hash of a public key, created with S/Kademlia's static crypto puzzle. Nodes store their public and private keys.

IPFS uses self-describing values, hash digest values are stored in multihash format, which includes a short header specifying the hash function used and the digest length in bytes. Example:

```
<function code><digest length><digest bytes>
```

2.2.2 Routing

IPFS nodes require a routing system that can find other peers' network addresses and peers who can serve particular objects. IPFS achieves this using a DSHT based on S/Kademlia and Coral. The size of objects and use patterns of IPFS are similar to Coral[14] and Mainline[13], so the IPFS DHT makes a distinction for values stored based on their size. Small values (equal to or less than 1KB) are stored directly on the DHT. For values larger, the DHT stores references, which are the NodeIds of peers who can serve the block. The interface of this DSHT is the following:

2.2.3 Transport

Important to mention is the fact that IPFS operates on top of libp2p, which is transport agnostic. It can operate on top of any transport as long as it is implemented.

2.2.4 Block exchange - BitSwap protocol

In IPFS, data distribution happens by exchanging blocks with peers using a BitTorrent-inspired protocol: BitSwap. Like BitTorrent, BitSwap peers are looking to acquire a set of blocks (**want_list**), and have another set of blocks to offer in exchange (**have_list**). Unlike BitTorrent, BitSwap is not limited to the blocks in one torrent. BitSwap operates as a persistent marketplace where nodes can acquire the blocks they need, regardless of what files those blocks are part of. The blocks could come from completely unrelated files in the filesystem. Nodes come together to barter in the marketplace. This works fine when the distribution of blocks across nodes is complementary, meaning they have what the other wants. Often, this will not be the case. In some cases, nodes must work for their blocks. In the case that a node has nothing that its peers want (or nothing at all), it seeks the pieces its peers want, with lower priority than what the node wants itself. This incentivizes nodes to cache and disseminate rare pieces, even if they are not interested in them directly. The protocol must also incentivize peers to seed even if they do not have anything they want at the time. Thus, BitSwap credit is used, where nodes track their balance (in bytes of data sent and received) with other nodes. Then, peers can decrease their debt by sending to their peers' debtors. Let the debt ratio r between a node and its peer be:

$$r = \frac{bytes\_sent}{bytes\_received + 1}$$

Then the sigmoid can be used that is scaled by debt ratio (Figure 1). Given r, let the probability of sending to a debtor be:

$$P(send|r) = 1 - \frac{1}{1 + exp(6 - 3r)}$$

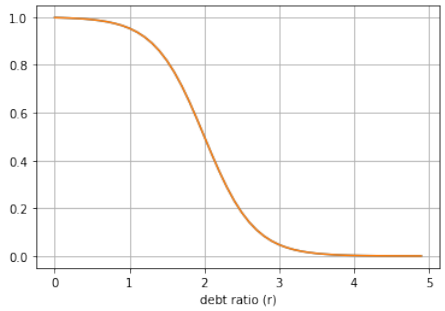

Figure 1: Sigmoid probability function, scaled by a debt ratio (r).

The debt ratio is a measure of trust: lenient to debts between nodes that have previously exchanged lots of data successfully, and merciless to unknown, untrusted nodes.

Also, BitSwap keeps a ledger, storing the history of exchange with other nodes. If ledgers between two nodes do not match, they are reinitialized from scratch, thus preventing malicious nodes from erasing their dept.

2.2.5 Object Mekle DAG

The DHT and BitSwap allow IPFS to form a massive peer-to-peer system for storing and distributing blocks quickly and robustly. On top of these, IPFS builds a Merkle DAG, a directed acyclic graph where links between objects are cryptographic hashes of the targets embedded in the sources. Merkle DAGs provide IPFS with many useful properties, including:

- Content Addressing: all content is uniquely identified by its multihash checksum, including links.
- Tamper resistance: all content is verified with its checksum. If data is tampered with or corrupted, IPFS detects it.
- Deduplication: all objects that hold the exact same content are equal, and only stored once. This is particularly useful with index objects, such as git trees and commits, or common portions of data.

The IPFS Object format is:

```
type IPFSLink struct {
    // name or alias of this link
    Name string

    // cryptographic hash of target
    Hash Multihash
```

```
    // total size of target
    Size int
}

type IPFSObject struct {
    // array of links
    links []IPFSLink

    // opaque content data
    data []byte
}
```

Links can be used to create data structures or to store larger files, by chunking them into smaller objects.

The objects can be **pinned**, meaning that their persistence will be guaranteed by saving them in the node's local storage.

2.2.6 *IPNS - Interplanetary Name System*

Since IPFS is content-addressed, addresses change unpredictably as the content is changed. That can be quite challenging to manage in applications and systems and could quite negatively impact user experience. Thankfully, that is easily solved with one of the integral parts of the IPFS, the Interplanetary Name System (IPNS). It provides static names for changing content addresses, by using asymmetric cryptography. Content hash can be signed with a private key and a record (consisting of signature, public key and hash) propagated via DHT so that anyone with the hash of a public key can easily retrieve a record (static hash, i.e. name) from DHT and verify that it is valid and then easily resolve it to the actual content address, i.e. hash.

3. DESIGN

All participants of the network are equal and they connect, with no centralized servers and infrastructure, forming a completely or partially wireless mesh in which they can efficiently exchange content/data and communicate via IPFS, without relying on centralized entities to coordinate their requests. This protocol works standalone, by itself, with no need for Internet protocol and its infrastructure, but also can work synergetically with the Internet.

As aforementioned, the main problem of mesh networks is bandwidth, so all optimization will aim to reduce bandwidth and increase the speed of IPFS on top of WMN at no cost to the end user.

IPFS is based on libp2p library that handles networking, routing and block exchange. Libp2p is extremely extensible, making it easy to include optimizations and new ways of transport more suitable for Individe. Most of the changes and optimizations Individe specifies will be implemented at the libp2p level or mesh routing algorithm.

3.1 Network and routing

For the initial version of Individe, a proposed routing protocol is B.A.T.M.A.N Advanced. Even though it underperforms in some metrics compared to the other mesh routing protocols[1][2][3] and underperforms Babel, which from available data is most performant, batman-adv offers certain mechanics that are not present out-of-the-box in the other protocols, such as Network Coding and Multi-Link optimizations. The initial goal is to rapidly prototype and establish improvement in speeds and bandwidth in Individe compared to a mesh using the same routing protocol without proposed optimizations. Thus, establishing Individe's performance. Also, batman-adv works at the kernel level (as a module), as opposed to the userland, thus making packet processing much cheaper in terms of CPU cycles as each packet does not need to be read and written to the kernel and back. This allows for decent and sustainable bandwidth on low-end and embedded devices and this logic should be persisted even after the prototype phase.

For future versions, the way to go would be designing the most performant routing protocol (probably by combining multiple algorithms, as observed in Babel protocol and LibreMesh[5] networks) for the purpose of routing P2P traffic. Also, redefining physical layer 1 and data link layer 2 should be considered, to use addressing that would conform to the one used by IPFS and libp2p, by using **PeerID**s, instead of IP and MAC addresses, thus identifying both nodes and peers via the same address.

3.2 Node

There are several types of Individe nodes:

- **Mesh nodes** are defined as roaming participants (devices) that do not have a fixed location most of the time and participate in routing directly. Examples of mesh nodes would be smartphones, smart wear, smart cars (while being driven), etc.
- **Router nodes** are defined as nodes that are mostly fixed in space, that participate in routing and can be relied upon as some kind of "pseudo-infrastructure", since they take part in forming mostly permanent links and routes. Examples of router nodes would be smart cars (when stationary and parked), home routers, smart TVs, smart house appliances, etc.
- **Gateway nodes** can be any node type that participates in routing, but also has its IPFS node connected to the Internet.
- **Access nodes** can be any node type that participates in routing, but also offers an endpoint for non-mesh devices, for example via USB, Ethernet cable, Bluetooth PAN, WiFi access point, etc.

Router and mesh nodes are just concepts, there is not much difference between them, router nodes are just characterized as nodes that are also connected with wiring or long-range antennas, acting as a major router in the network "infrastructure", by having greater bandwidth throughput, speed and reliability. Every node in the network is an infrastructure of Individe, but analog to the Internet, router nodes may be considered as an "actual infrastructure", since they shall be most relied upon in everyday non-extreme operation of the network, due to their fixed location, ability to use wires which are faster than wireless and have longer range and ability to use antennas to expand the network range wirelessly when wiring is not an actual or preferred option.

Each node has an exposed IPFS node and automatically adds its mesh neighbors and other participants as its IPFS peers.

Nodes can be connected to the Internet to provide IPFS with Internet access, but are not required to and can limit

the Internet access of their IPFS node as they wish. The Internet and other peer-to-peer networks and protocols can also be routed and bridged to this mesh, but that is not the subject, nor the scope of Individe protocol.

3.3 Optimizations and principles

The important thing to consider while assessing and talking about optimizations of Individe is that it is based on IPFS, a distributed file system and is meant only to route and exchange blocks of data of such system. Optimizations are focused on P2P block exchanges between the nodes and optimizing them as much as possible, to allow for the optimal self-contained system, without a need for an Internet connection to the other IPFS nodes (considering that all the requested data is contained within the Individe network).

3.3.1 Topology-aware P2P communications

IPFS sounds very attractive on top of the WMNs (wireless mesh networks), but unfortunately, it has no regard for the network topology. It works quite well on top of the Internet protocol, which has an enormous infrastructure and supports immense bandwidths and speeds. But, when talking about WMNs, that have significantly smaller capabilities, some optimizations are required in the context of using IPFS to make Individe decent and more robust. So, besides the metrics used by mesh routing algorithms at the routing level, some other metrics should be taken into an account at the level of IPFS.

Making IPFS understand the topology and how data is routed, just as making the routing algorithm understand on top of which topology and what it is routing, would lead to massive improvements in terms of speed and saving of airtime, which would in turn increase bandwidth, since more packets can be sent in the smaller time frame.

These optimizations get really complex and require changes in libp2p's transport, IPFS's peer selection mechanisms and routing algorithms. In the future, Individe plans to explore many possible optimizations with regard to topology-awareness of IPFS and synergy between IPFS and routing algorithms.

ALMswap.

One of the already proposed strategies is ALMswap[12], which includes airtime (time needed to transmit a frame over a specific link or path, considering parameters of the physical layer and the medium of transfer) cost in IPFS's sigmoid function scaled by debt ratio, during the process of peer selection. The proposed strategy takes HWMP's (Hybrid Wireless Mesh Protocol) Airtime Cost Metric (ALM):

$$c_a = [O_{ca} + O_p + \frac{B_t}{r}] * \frac{1}{1 - e_{fr}}$$

where:

- O_{ca} and O_p are constants from the physical layer 1 for the channel access and MAC protocol overhead, respectively
- B_t is fixed virtual test frame size (in HWMP standard defined as 8192-bit frame)
- e_{fr} is the probability of unsuccessful transmission of a test frame on the current link at the rate r (given in MBit/s)

The idea here is to optimize download time and reduce it, by using the best possible link, therefore increasing thought and increasing end-user's experience. Thus, ALMswap combines ALM metrics with BitSwap's debt ratio sigmoid into a single function. But, first, the sigmoid function for ALM must be defined, with a few already existing proposals that could be considered:

$$Sigmoid(C_a) = \frac{1}{1+e^{c_0 - c_a}} \text{ [15]}$$

$$Sigmoid_k(C_a) = \frac{1}{1+e^{k*(c_0 - c_a)}} \text{ [16]}$$

$$tanh(C_a) = \frac{e^{(c_0 - c_a)} - e^{(c_a - c_0)}}{e^{(c_0 - c_a)} + e^{(c_a - c_0)}} \text{ [17]}$$

Now, transmission probability can be redefined to incorporate both ALM and BitSwap's relevant metrics:

$$P(send|(r, c_a)) = (1 - \frac{1}{1+e^{6-3r}}) * \frac{1}{1+e^{c_0 - c_a}} \text{ [18]}$$

where:

- r is the node's debt ratio
- c_a is node's airtime cost
- c_0 is the expected value of the node's airtime cost

It shows the impact of the dept ratio (from IPFS's BitSwap) and the ALM metric (from the 802.11s) on the algorithm for node selection strategy. In certain test scenarios with 5x5 node grid, 15% reduction in download time has been observed[12].

This is one of the simpler and already existing steps toward topology-awareness, further research and optimization shall be conducted during the development of Individe.

3.3.2 Multi-link Optimizations

One of the optimizations proposed in B.A.T.M.A.N Advanced is a multi-link optimization[6]. It considers using multiple network interfaces (wireless or wired). Besides the obvious, that node can transmit and receive at the same time with two interfaces or even make multiple transmissions and get multiple receipts simultaneously with multiple mediums (antenna or wired connections) it is possible to use interface alternating and interface bonding.

Interface alternating.

With interface alternating frames are forwarded on a different interface than the one on which the frame was received. The purpose of this alternation is to reduce interference (we can either send or receive on a WiFi interface at one time) and balance the network load better on the available interfaces and eventually increases throughput (Figure 2). Interface alternating is performed by considering the interface where a packet has been received and selecting the best neighbor of the available outgoing interfaces.

Figure 2: Illustrated mechanism for a chain of nodes with two interfaces.

Also, interface decision is can be performed considering the whole mesh network. For example, consider some dual radio mesh nodes where most nodes have both a 2.4 GHz and a 5 GHz link to the next hop, except for the connection between nodes C and D which only has a 2.4 GHz. Based on the information propagated, A will now choose the 2.4 GHz link first to reach node E. This way, it can avoid using the same frequency at node C, compared to starting with 5 GHz (Figure 3).

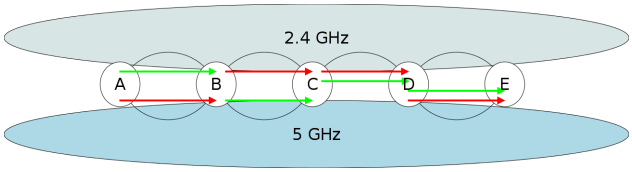

Figure 3: Illustrated interface alteration considering the whole network

Also, with the network-wide multi-interface optimization, the multi-interface node can act as a routing splitting point which can lead to multipath routing. Considering the scenario illustrated (Figure 4) and assuming all paths to be perfect, node B will route the packets via C if they are coming from F (green path) and route the packets via G if they are coming from A (red path) for the destination E.

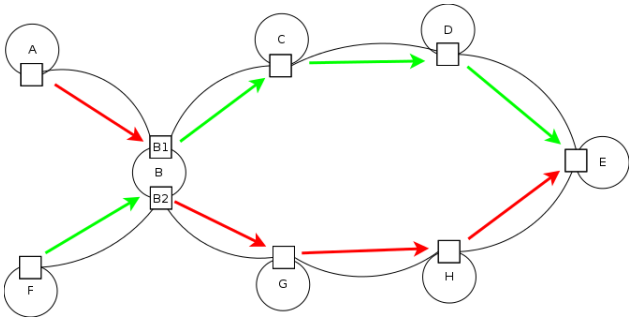

Figure 4: Illustrated interface alteration considering the whole network

Interface bonding.

When multiple paths on different interfaces with similar quality are available, frames may be distributed to be sent over these available paths. The individual frames can be sent over multiple paths in a round-robin fashion (Figure 5). Using this technique, the throughput may be increased by the number of interfaces involved in the bonding. In practical tests over two WiFi links, more than 50% of throughput gain has been observed[6].

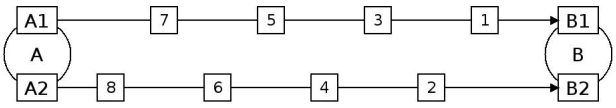

Figure 5: Illustrated interface alteration considering the whole network

However, if the paths have different speeds, the throughput may even decrease due to the slower link slowing down the whole bonding. Therefore, during the implementation, physical layer 1 properties should be considered.

Throughput gain.

At the WirelessBattleMesh in Bracciano, the B.A.T.M.A.N Advanced team performed throughput tests to measure the gain of the various modes (Figure 6).

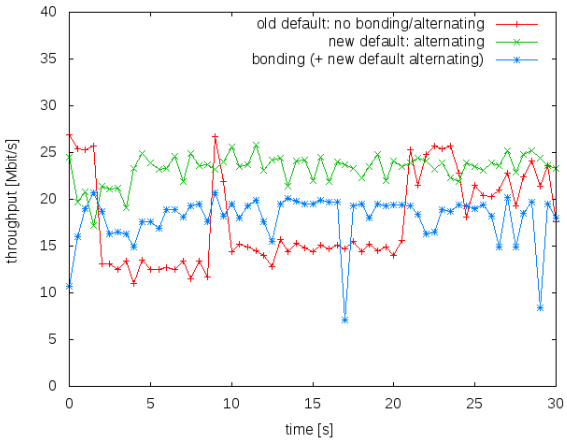

Figure 6: The graph shows a throughput gain due to interface bonding and alternating in a setup of three B.A.T.M.A.N Advanced nodes, each with two 802.11abg wifi interfaces connected to the mesh network (1x2.4GHz and 1x5.8GHz).

3.3.3 *Network Coding Optimizations*

Yet another optimization proposed in B.A.T.M.A.N Advanced, that could be implemented on routing protocol is Network Coding[4][7]. Network Coding can enable a relay (a medium or a path between two or more nodes) node to combine two packets into a single transmission, thus saving airtime.

The most common and simplest example of network coding requires a setup of three nodes (Figure 7).

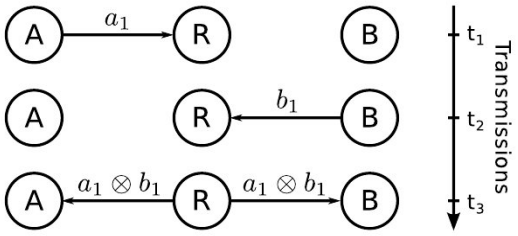

Figure 7: A network coding example illustration, the repeater R can save one transmission by sending the combined messages of A and B. A and B can calculate the message they want to receive by subtracting their own sent message.

Another scenario, in which network coding can help to save air time, is the X-topology, where two sets of nodes

communicate through the same relay (8). In this example, Node C and Node D both receive the same network-coded packet, and they both use the overheard packet (from Node A and Node B, respectively) to decode the received packet.

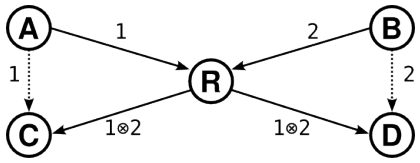

Figure 8: A network coding example illustration, depicting an X-topology.

In certain scenarios (e.g. heavy load traffic intersects at a relay), network coding can give up to 1.6 times gain in total throughput. Under less load, the relay might hold back packets up to 10 ms before forwarding these, as it tries to get packets to combine.

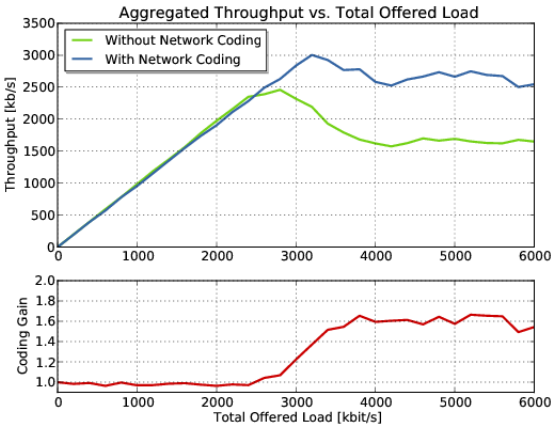

Figure 9: Graph illustrating the throughput in kb/s with and without network coding (blue and green line respectively) as well as the throughput gain (red line) achieved by network coding on a chain of 3 routers (Figure 7) with clients attached to each end.

Since relays defer packet forwarding to wait for opportunities to network code packets, a delay of up to 10ms is introduced at each hop in the network. If the traffic load increases enough, more opportunities to network code should appear and delay can be decreased by combining packets.

Network coding with IPFS.

Network coding, though being an optimization for routing towards Internet gateway nodes in the mesh, is much more powerful considering Individe's architecture and IPFS usage. A relay node can combine incoming packets that are being routed in the same direction and save airtime by combining them into a single transmission (Figure 10). So, unlike the abovementioned principles, packets can be grouped even if some of them will be forwarded further (with more hops) and maybe grouped into another coded transmission. This approach on a large scale gets used very much, since Individe routes traffic in a peer-to-peer manner and many packages are sent and can be combined if their direction is the same.

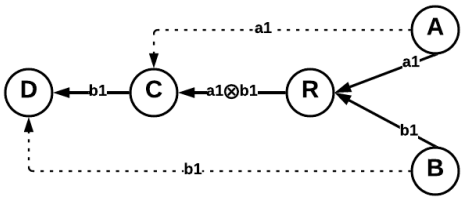

Figure 10: Graph illustrating the network coding in the IPFS. Node A is sending packet a_1 to node C and node B is sending packet b_1 to node D. Both packets are coded into a single transmission $a_1 x b_1$ and transmitted through relay node R. Node C receives both packages, takes only the one requested (a_1) and forwards the rest (b_1) down the route.

3.3.4 *Multicasting blocks*

Since Individe's aim is data distribution, at some point, when data is distributed enough, it would be safe to assume that multiple neighboring peers in the network would have one or more same block in their wantlists. The current approach is to just send these blocks by n number of transmissions, with n being a number of neighboring nodes wanting the same block. In the context of the Internet, this is completely fine, since bandwidth is extremely huge and speeds groundbreaking, but in mesh networks, we are looking to save airtime and lower the amount of bandwidth we are sending. Thus, sending blocks via multicast to multiple neighbors that want that same block, would save airtime and bandwidth, since we are transmitting that block in a single transmission, compared to the multiple transmissions that would occur when multiple peers want the same block.

3.3.5 *Hop reduction due to content-addressed data distribution*

Also, one interesting behavior or principle Individe observes is that increased data distribution optimizes traffic in mesh due to the content-addressed nature of IPFS and also, that a certain portion of content tends to be distributed where it is needed.

In BitSwap, any block can be retrieved via its hash, it doesn't matter if it is a part of some data structure or not, as long as someone has the hash and it is seeded, it can be retrieved. Now, this is interesting, since the location of the block or where it is coming from is irrelevant, it can be received from anywhere. Currently, with mesh networks relying on Internet bridges, whatever the action, the packet is being sent via the shortest path to the node connected to the Internet. But, in Individe, Internet can be avoided altogether, if any node closer than the Internet gateway node has the requested block. With larger use, the probability of the requested block being closer to the requestor than the Internet gateway is increased, thus need for the Internet infrastructure and number of hops is decreased, therefore network gets self-contained and the speed is increased.

Also, interestingly, a certain amount of content tends to get distributed where it is needed. It can be observed with trending content (music videos, movies, TV shows, memes, social media posts, etc). For example a music video trending in a certain city or country it gets more distributed within the region where it is more likely to get consumed, thus

number of hops is likely to get reduced and the network gets more self-contained, without the need for Internet access. The same goes for files shared in the workplace, college, school, etc, content is there, where it is needed. And content shared within small proximity (e.g. sending pictures and messages to friends and family) is routed without relying on the Internet connection.

3.3.6 Hop and traffic reduction due to data deduplication

The interesting behavior of IPFS's usage of Merkle DAG is a possibility for data deduplication. Since data structures are split into smaller objects, they can be referenced multiple times in their representation via links, thus if the structure has repeating sequences, less data is needed to represent it with blocks. Splitting, i.e. chunking, of data structures is done by chunking algorithms. The chunking algorithm currently in use is Rabin-Karp. By optimizing chunking algorithms to increase data deduplication, fewer chunks/blocks are produced and thus bandwidth is reduced.

Content-defined chunking - CDC.

To increase data deduplication, content-defined chunking (CDC) algorithms could be used, so that they understand what data structure is being chunked. CDC increases deduplication, in a way that if the data structure gets changed, not all of the underlying chunks will get changed, only a small portion, thus, more chunks get reused, optimistically, chunks that are already distributed and in circulation, therefore decreasing the bandwidth and hops, if older chunks are decently distributed.

Common bytes as means of increasing deduplication.

Common bytes in terms of data deduplication, here, are defined as bytes that are reused in the same data format across different data structures. For example, reusable file headers, paddings, magic numbers, certain sequences, code snippets, etc. One example would be the common begining of an HTML document:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta
        name="viewport"
        content="width=device-width,
        initial-scale=1.0">
    <meta
        http-equiv="X-UA-Compatible"
        content="ie=edge">
```

This code is present in almost every HTML page on the Internet in this exact form and the same goes for the HTML footer. Another example would include the beginning of a bash scripts:

```
#!/bin/sh
#!/bin/bash
#!/bin/bin/pwsh
#!/bin/bin/env python3
#!/bin/false...
```

More examples are configuration files for software, libraries, and transpilers that often have some exactly the same portions. Also, interpreted, formatting, markup, styling and some other language types could have the same code snippets.

By identifying reusable chunks across different data of the same format, a certain number of chunks would be referenced in a lot of distributed data structures. This would mean that often-used chunks would be distributed all across the network and when requesting and retrieving content that references them, for that small portion of data, the network will be offloaded by that number of bytes, thus reducing traffic and increasing bandwidth through.

For a simple HTML file header, saving would be 187 bytes, if no spaces or tabs are used, 194 bytes if tabs are used, 201 bytes if double spaces are used and 215 bytes if four spaces are used. Let's take 201 bytes as a reference, if we have one million nodes in the network and each requests a different HTML page with the same header (probabilistically very high) and we consider that each node has that header stored, savings would be 201 megabytes. And if some node does not have some common bytes, the number of hops needed to retrieve that portion of a data structure is reduced from the actual number of hops to likely just a single hop, since one of the neighbors will most likely have them.

One of the ways to implement common bytes deduplication would be defining common bytes in advance and initially, it would be a way to go and would provide a nice baseline. But, machine learning could be employed across larger data sets to identify common bytes and also, end-users, nodes and routers could opt-in to run resource-restricted model training to identify new and yet unidentified common bytes, that can be included in the future. And considering the sheer amount of data that a single person possesses and shares nowadays and the amount of data that would pass through nodes in the network the possibility of training models with unforeseen common bytes decent for deduplication is very tempting. Also, training models in the actual network provides statistics about how often such common bytes would be used and would be useful while considering breaking previous deduplications to favor common bytes in a certain data format.

3.3.7 Reducing amount of traffic using compression

As previously mentioned, decreasing the size of traffic, i.e. bandwidth is crucial for increasing the bandwidth and speed of WMN. One of the easiest solutions would be using lightweight real-time (fast) compression algorithms to compress data chunks before routing them. This makes sense, since after IPFS finishes chunking data structure(s), we are left with blocks of data larger than a few kilobytes, so deduplication done by IPFS still leaves room for further compression, because chunker has a restriction with respect to minimal block size. Some more notable and used real-time compression algorithms would be Brotli, Deflate, Zopfli, LZMA, LZHAM, bzip2, Zlibm LZF, lz4, lzo1x, snappy, quicklz and even ZSTD by Facebook. Their compression ratios are as great as ≈ 3 and can compress gigabyte files within a second[8]. But without any doubt, ZSTD by Facebook outperforms all the others, taking into account speed and compression ratio (ratio of uncompressed to compressed file). So, for now, let's discuss using ZSTD compression on Individe, i.e. compressing IPFS blocks via libp2p transport.

We could easily get the number of seconds required to transport a block of data with certain bandwidth:

$$t = \frac{B_s}{BW}$$

where:

- t is a time of exchange or transfer in seconds (s)
- B_s is block size in bytes (B)
- BW is bandwidth in bytes per second (B/s)

We could do the same for the compressed block, including the time required for compressing and decompressing block:

$$t = \frac{B_s}{C_s} + \frac{B_s}{C_r} * \frac{1}{BW} + \frac{B_s}{D_s}$$

where:

- t is a time of exchange or transfer in seconds (s)
- B_s is block size in bytes (B)
- BW is bandwidth in bytes per second $(bytes/s)$
- C_s is compression speed in bytes per second (B/s)
- D_s is decompression speed in bytes per second (B/s)
- C_r is compression rate $C_r = \frac{filesize}{compressedFilesize}$

Now, we can create a function that represents the time difference between sending a compressed block compared to the uncompressed block.

$$f(BW, B_s, B_n) = (\frac{B_s}{BW} - (\frac{B_s}{C_s} + \frac{B_s}{C_r * BW} + \frac{B_s}{D_s})) * B_n$$

where:

- B_n is the number of blocks transferred, since realistically we won't be transferring only one block, but many blocks via IPFS

Now, let's substitute values C_s, D_s and C_r with ZSTD's benchmarks for the highest compression ratio setting:

```
Cs = 500 MB/s,
Ds = 1660 MB/s,
Cr = 2.884
```

and take IPFS's block size at 512 KB:

```
Bs = 0.512 MB
```

and for now, remove B_n, and we get hyperbolic, uncompressed to compressed exchange time difference function:

$$f(BW) = \frac{0.512}{BW} - (\frac{0.512}{530} + \frac{0.512}{2.887 * BW} + \frac{0.512}{1700})$$

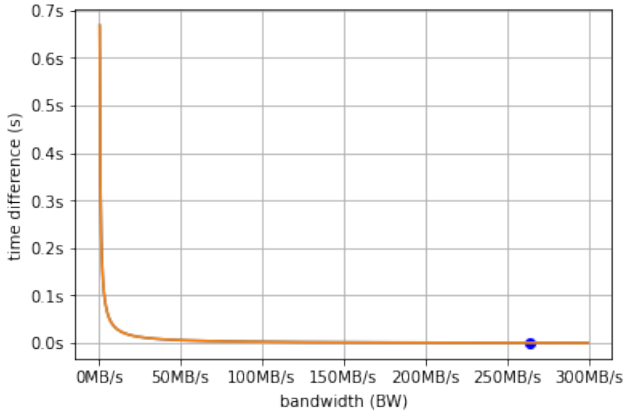

Figure 11: Uncompressed to compressed exchange time difference function

Now, the total bandwidth is reduced by $C_r = 2.887$ times and also, there are time improvements. Subtle at greater speeds and large at smaller speeds. Function becomes negative, i.e. time difference becomes negative at the speed of 264.086 MB/s, meaning that at that speed, the compressed exchange becomes slower than the uncompressed, but since the function is hyperbolic, if we find the limit of the function:

$$\lim_{BW \to \infty} (\frac{0.512}{BW} - (\frac{0.512}{530} + \frac{0.512}{2.887 * BW} + \frac{0.512}{1700})) = 0.00126721$$

we can conclude that a time difference bigger than $126721 * 10^{-3}$ seconds is impossible. But, if B_n is plugged in, as it is increased, although there are even better time improvements, after the function becomes negative at 264.086 MB/s, time gets worse and worse, since the limit gets multiplied by the B_n. So, since the idea of the network that everyone seeds almost all the time, $B_n \to \infty$, therefore B_s also tends to infinity and transport should stop compression if the speed exceeds 264.086 MB/s thresholds and just send the uncompressed block, but realistically it won't happen that often. Or some kind of tolerable negative time difference could be introduced since cables and strong antennas can achieve such speeds.

Another possible optimization to reduce compression time to zero for some blocks could be caching mechanism. Smart or A.I. caching mechanisms could recognize blocks that are in high demand and cache them in compressed form, further reducing the time of exchange and reducing CPU usage. Also, applications and services could programmatically cache compressed blocks, to save on CPU usage and improve user experience

More in-depth research and analysis could be conducted in the future to possibly find better algorithms, settings and calculations to achieve even better bandwidth optimization.

So far, this was all about the optimization at the libp2p's streaming, but services and applications could also compress the content before putting it up on the IPFS and reference compression used via some form of metadata, producing even better bandwidth improvements, since IPFS, unlike libp2p can read all the blocks as a whole, therefore compress it with respects to the file format and its content.

3.3.8 Comparison to traditional mesh

Now, we can make a comparison between a mesh network deployment with the existing routing protocol and the one with Individe. Considering that both networks have a gateway to the Internet, we could make some statements:

- number of hops gets reduced over time as the content is distributed and the network used
- number of hops gets drastically reduced, if the content is in high demand
- number of hops gets reduced for a small portion of static content that is highly deduplicated (common bytes)
- impact of content-defined chunkers is not yet measured, but potentially significant, in theory saving airtime
- Individe's compression on the transport layer makes up to 2.887 increase in the bandwidth throughput compared to traditional mesh
- impact of network coding and multi-link optimization is not yet measured, but is potentially significant
- topology-awareness with ALMswap, showed up to 15% reduction in download times
- Individe can operate completely off-grid with no Internet access and is resistant to Internet outages, since the applicative layer conforms to the distributed P2P model
- dynamic data exchange is optimized slightly using compression and topology-awareness, but on longer ranges still relies on some kind of infrastructure, but completely avoids Internet on shorter ranges

In total, we cannot measure the exact improvement in terms of bandwidth and speed of Individe compared to existing WMN solutions, until the prototype. But, measured optimizations like compression and topology-awareness already offer almost 3 times more bandwidth for the same mesh deployment and drastically improved speeds. All that with the ability to function completely without an Internet backbone. More analysis and tests shall be conducted to establish improvements brought by proposed optimizations. But, it's very important to remember that these optimizations even when measured shall not be considered separately, since when combined, they actually optimize each other, e.g:

- deduplication and distribution -> increased bandwidth and better airtime
- compression -> better airtime and increase in bandwidth
- better airtime -> more efficient interface alternating and bonding
- interface alternating and bonding -> increased bandwidth
- topology-awareness -> better airtime
- increased bandwidth -> better airtime

A bit crude, but there is a strong relation between airtime, bandwidth and proposed optimizations, thus, one optimization, further optimizes the others and so on.

4. USAGE

Individe, relying on the IPFS, requires applications to follow specific architecture to get the most out of it and to function properly. This section will outline how applications can leverage Individe and several different architectures that applications can use. More generally, Individe will pursue the following principles and is meant to be used and designed as follows:

- as a simplistic general-purpose networking stack, implementing only the necessary functions, leaving the rest to the applicative layer
- as a standalone solution and protocol for general-purpose networking (with or without some sort of backbone infrastructure wired or wireless, depending on the location, usage, needs, etc.)
- as a support for Internet backbone or a standalone solution in regions where it is hard to set up Internet infrastructure efficiently

4.1 Connecting to Individe

Just using IPFS would be enough to open a gateway to the Individe mesh, since inevitably one of your peers or your peers' peers would be part of the Individe mesh. But to guarantee the most direct connection to the Individe, if the user's device has Individe node, we specify an endpoint that ensures applications usage of Individe's IPFS node. Individe defines *individe.local* mDNS hostname, which resolves to the IP address of the user's Individe node, whether running locally on the same device or on the LAN. Using HTTP to query GET */ipfs/info* route on that address returns the multiaddresses required to establish a connection to that Individe node's IPFS endpoint and start peering. This endpoint is designed to allow for the creation of usable production Individe nodes. Ideally, Individe node would be integrated into the device and the endpoint would not be required in this context, but due to the logistics and cost of getting vendors to integrate it, just as the problems with early adoption, an external node presents the best opportunity for connecting and participating.

4.2 Application architecture

Individe, focuses on IPFS applications and does not guarantee Internet connection and DNS resolutions. Therefore, the only thing that is guaranteed is a connection to the reachable nodes in the network and working and exposed IPFS node connected to the other reachable nodes in the network, meaning that applications need to approach Individe in specific ways. Also, applications should consider the underlying mechanics of the Individe and other underlying protocols, leverage optimizations proposed and try to offload as much bandwidth as possible of the network.

4.2.1 Content (static data) distribution

Individe forces applications to use IPFS for content and data storage and exchange mechanisms. Applications can use pinning services with their IPFS nodes to guarantee data persistence. Also, the lack of Internet connectivity should be considered, since not all users could have Internet access, thus offline and online app modes could be considered or just as any other currently existing app, it can just not work

without the Internet. Encryption may be applied to sensitive data, IPFS supports object-level encryption, so data can be encrypted at the lowest level of the IPFS and encryption can be described at the object level. Also, applications should always consider compression (for the sake of the network) for static content that is not time-sensitive or if the time impact is not that great due to the compression.

4.2.2 Static names

Since IPFS is content-addressed, addresses change unpredictably as the content is changed. That can be quite challenging to manage in applications and systems and could quite negatively impact user experience. Thankfully, that is easily solved with one of the integral parts of the IPFS, the Interplanetary Name System (IPNS)[11]. It provides static names for changing content addresses, by using asymmetric cryptography. Content hash can be signed with a private key and a record (consisting of signature, public key and hash) propagated via DHT so that anyone with the hash of a public key can easily retrieve a record (static hash, i.e. name) from DHT and verify that it is valid and then easily resolve it to the actual content address, i.e. hash.

Proquint human-readable names.

IPNS produces somewhat unfriendly names for humans, such as:

```
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/.
```

Of course, such a name is to be expected since it is just encoded random binary data. To make them more human-readable, it is possible to use the Proquints[9], identifiers that are readable, spellable, and pronounceable by humans. In essence, it encodes binary data into human-readable words, thus:

```
# this proquint phrase
/ipns/dahih-dolij-sozuk-vosah-luvar-fuluh

# will resolve to corresponding
/ipns/KhAwNprxYVxKqpDZ
```

4.2.3 Dynamic data exchange

The important thing to lay out, which has not yet been mentioned, is that networking is not only directed towards content retrieval (static data exchange), rather networking implies any form of data exchange, including content and data retrieval, but also real-time communication and dynamic data exchange. So far, we discussed only the content and data retrieval mechanisms. This section outlines and discusses real-time message exchange in Individe.

Message exchange on Individe is possible but is not as efficient and scalable as content retrieval without infrastructure. Individe message exchange and real-time communication relies on IPFS and libp2p pubsub. It is based on publish/subscribe model and functions as P2P. Since message exchange does not usually happen just between nodes and users on short distances, e.g. in client-server communication, long-distance P2P communications, etc, it is not more optimized than current approaches, besides proposed stream compression, and therefore requires infrastructure (on longer distances). But, for example, private chats on short distances do not require infrastructure and would mostly have direct or relatively short routes that would be used for message exchange and communication, just as group chats, applications like forums, games, etc, on small or a bit longer distances.

Applications should consider that dynamic data exchange may not happen instantaneously in the deployments of Individe that lack Internet connectivity, but should again consider offline and online app modes and should be aware that dynamic data exchange on smaller distances is possible even without the Internet connection.

4.2.4 Servers

Individe allows the creation and usage of servers. Servers can use any protocol of their choosing, i.e. TCP, UDP, HTTP, etc, but are encouraged to limit the client-server communication to a pub-sub, since Individe is optimized for libp2p and IPFS. But, usage of servers should be employed only when necessary and it would always be a good idea to consider serverless architectures if possible, to allow for applications and services to run even if disconnected from the global mesh and the Internet.

4.2.5 Centralized databases

Individe allows the creation and usage of centralized databases. As previously described, servers can be used, therefore, putting a database behind (within) such a server would be a way to go for the usage of such databases.

4.2.6 Distributed databases

One very interesting feature that Individe enables through IPFS are CRDTs[10]. Conflict-free replicated data types, the data structures or databases that are suitable for being distributed across the network, may be used in many cases, to provide applications with data store solutions, without central servers and databases. One interesting implementation worth mentioning is OrbitDB.

5. FUTURE

This section discusses future possible ideas, principles and features worth mentioning.

5.1 Custom mesh routing protocol

As aforementioned, the currently picked routing protocol (batman-adv) is not that efficient and is chosen because of its wide range of features that can enable fast prototype development for measuring Individe's performance compared to the traditional mesh approaches. In the future, the development of a custom mesh routing protocol for the Individe is a must, fusing all the best techniques from its predecessors and using PeerIDs for addressing, instead of IP and MAC addresses. It will most likely operate in the kernel.

5.2 Satellites

At the time of writing, Apple has just recently announced its built-in satellite communication modem in their new Apple iPhone 14. Also, we have SpaceX with its network of Starlink satellites. Though the idea of satellite Internet communications has been around for a while, it is just as of recently getting mainstream traction. One of the possible interesting integrations for Individe would be the usage of satellite communications as a transport mechanism. Since the whole idea of Individe is based around retrieving as much data as possible from the nearest sources, it decreases the

amount of data that would go through satellite links, going first to the wireless and wired points that are near for most data needs. This principle would theoretically make data transport via satellite more robust, since only a certain percentage of data would be routed through satellite links and would therefore use it less, which is good, since satellite communication is somewhat slow and would deload satellites, making them faster when needed.

5.3 Post-quantum cryptography

IPFS and libp2p currently use RSA cryptography. Due to a recent number of breakthroughs in quantum computing, many cryptographic algorithms are in danger of being broken. Thus, it is a good idea to lay out that there are many post-quantum cryptographic algorithms in existence and that in case of RSA being broken, it will be easy to integrate some novel post-quantum cryptography.

6. ACKNOWLEDGMENTS

Individe is the synthesis of many great ideas and systems. It would be impossible to think about it, if not for the existence and creation of protocols, ideas and concepts such as P2P networks, BitTorrent, BitSwap, IPFS, Wireless Mesh Networks, mesh routing protocols (B.A.T.M.A.N, BMX, OLSR, AODV and many more), DHT, etc. Therefore, special thanks to the brilliant minds whose works, papers, ideas, concepts, knwoledhe and protocols are mentioned in this writing, for their immense contribution to science. But, one special thanks to Juan Benet, for his remarkable work on IPFS and for being an inspiration for this paper and to my career as a software architect and engineer.

7. REFERENCES

[1] D. Murray, M. Dixon and T. Koziniec, "An experimental comparison of routing protocols in multi hop ad hoc networks "2010 Australasian Telecommunication Networks and Applications Conference, 2010, pp. 159-164, doi: 10.1109/ATNAC.2010.5680190.

[2] M. Abolhasan, B. Hagelstein and J. C. . -P. Wang, "Real-world performance of current proactive multi-hop mesh protocols" 2009 15th Asia-Pacific Conference on Communications, 2009, pp. 44-47, doi: 10.1109/APCC.2009.5375690.

[3] Hachtkemper, Manuel & Rademacher, Michael & Jonas, Karl. (2017). Real-World Performance of current Mesh Protocols in a small-scale Dual-Radio Multi-Link Environment.

[4] HundebÃÿll, Martin & Ledet-Pedersen, Jeppe (2011), Inter-Flow Network Coding for Wireless Mesh Networks martin. Master Thesis in Networks and Distributed Systems, Aalborg University

[5] https://libremesh.org/howitworks.html

[6] https://www.open-mesh.org/projects/batman-adv/wiki/Multi-link-optimize

[7] https://www.open-mesh.org/projects/batman-adv/wiki/NetworkCoding

[8] Jyrki Alakuijala, Evgenii Kliuchnikov, Zoltan Szabadka, and Lode Vandevenne, "Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms", Google, inc.

[9] Wilkerson, Daniel. (2009). A Proposal for Proquints: Identifiers that are Readable, Spellable, and Pronounceable., arXiv:0901.4016v2 [cs.SE] 26 Jan 2009

[10] Hector Sanjuan, Samuli Poyhtari, Pedro Teixeira, Ioannis Psaras. "Merkle-CRDTs: Merkle-DAGs meet CRDTs", arXiv:2004.00107 [cs.NI]

[11] Juan Benet, "IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)"

[12] Guanyu Wu, Liang Qian, Changyi Wang, Lianghui Ding, and Feng Yang. 2019. A novel cross-layer P2P mechanism for dynamic wireless mesh networks. In Proceedings of the 5th International Conference on Communication and Information Processing (ICCIP '19). Association for Computing Machinery, New York, NY, USA, 237âĂŞ241. https://doi.org/10.1145/3369985.3370018

[13] L. Wang and J. Kangasharju. Measuring large-scale distributed systems: case of bittorrent mainline dht. In Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on, pages 1âĂŞ10. IEEE, 2013.

[14] M. J. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with coral. In NSDI, volume 4, pages 18âĂŞ18, 2004.

[15] Sbai, M.K., Barakat, C., Choi, J., Hamra, A.A., Turletti, T. (2008). Adapting BitTorrent to Wireless Ad Hoc Networks. In: Coudert, D., Simplot-Ryl, D., Stojmenovic, I. (eds) Ad-hoc, Mobile and Wireless Networks. ADHOC-NOW 2008. Lecture Notes in Computer Science, vol 5198. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-85209-4_15

[16] S. M. S. Bari, F. Anwar and M. H. Masud, "Performance study of hybrid Wireless Mesh Protocol (HWMP) for IEEE 802.11s WLAN mesh networks," 2012 International Conference on Computer and Communication Engineering (ICCCE), 2012, pp. 712-716, doi: 10.1109/ICCCE.2012.6271309.

[17] Michiardi, Pietro & Urvoy-Keller, Guillaume. (2007). Performance Analysis of Cooperative Content Distribution in Wireless Ad Hoc Networks. 22 - 29. 10.1109/WONS.2007.340468.

[18] M. Rethfeldt, P. Danielis, B. Konieczek, F. Uster and D. Timmermann, "Integration of QoS Parameters from IEEE 802.11s WLAN Mesh Networks into Logical P2P Overlays," 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, 2015, pp. 1170-1177, doi: 10.1109/CIT/IUCC/DASC/PICOM.2015.175.