Asterism: Decentralized File Sharing Application for Mobile Devices

Olli-Pekka Heinisuo
Tampere University
Tampere, Finland
op.heinisuo@gmail.com

Valentina Lenarduzzi
Tampere University
Tampere, Finland
valentina.lenarduzzi@tut.fi

Davide Taibi
Tampere University
Tampere, Finland
davide.taibi@tut.fi

***Abstract*—Most applications and services rely on central authorities. This introduces a single point of failure to the system. The central authority must be trusted to have data stored by the application available at any given time. More importantly, the privacy of the user depends on the service provider capability to keep the data safe. A decentralized system could be a solution to remove the dependency from a central authority. Moreover, due to the rapid growth of mobile device usage, the availability of decentralization must not be limited only to desktop computers.**

In this work we aim at studying the possibility to use mobile devices as a decentralized file sharing platform without any central authorities. This was done by implementing Asterism, a peer-to-peer file-sharing mobile application based on the InterPlanetary File System. We validate the results by deploying and measuring the application network usage and power consumption in multiple different devices.

Results show that mobile devices can be used to implement a worldwide distributed file sharing network. However, the file sharing application generated large amounts of network traffic even when no files were shared. This was caused by the chattiness of the protocol of the underlying peer-to-peer network. Consequently, constant network traffic prevented the mobile devices from entering to deep sleep mode. Due to this the battery life of the devices was greatly degraded.

***Index Terms*—decentralized file sharing, peer-to-peer, mobile, Sailfish OS, InterPlanetary File System**

I. Introduction

The usage of mobile devices has increased significantly during the past 10 years. The users of all these network connected devices are producing and consuming more data than ever before

Many of the mobile devices are connected to one or multiple services which are often running on cloud platforms such as Amazon's AWS and Microsoft's Azure. These platforms provide computing resources, for example storage space and processing power, to build scalable web-based software services. To the end users cloud services are usually completely transparent: they make it possible for software applications to share data between different devices everywhere in the world.

Cloud usage has been growing rapidly in recent years and it will continue to grow in the future [1]. Consequently, more and more personal data of the end users is stored in large centralized data centers which are owned and controlled by a few large private enterprises [2]. While centralized storage is proven to work well, it has many implications regarding privacy, security and control of the data. Facebook's and Cambridge Analytica's illegal usage of 50 million user profiles is a recent example of these issues [3].

To avoid the issues of centralization, data can to be stored without any central authorities. Decentralized storage and content distribution can be achieved with peer-to-peer (P2P) networking. In peer-to-peer networking every node of a network is both a client and a server [4]. The nodes then talk to each other without any central service as opposed to centralized systems where all communication goes through dedicated central servers.

Peer-to-peer (P2P networks and file sharing are rather common on desktop environments but they have been rarely used on mobile devices due to more restricted resources. However, the growth of the mobile ecosystems has changed the situation: the processing power of mobile devices is getting better every year [5]. Similarly, the mobile network speeds are growing fast [6]. While mobile devices are used a lot, a considerable amount of processing power and network bandwidth remains still unused. Additionally, mobile devices are almost always on and connected to the Internet. These unused resources could be used to power a worldwide P2P data sharing network.

The goal of this paper is to understand if it is possible to transform mobile devices into fully decentralized content delivery network without any central servers or authorities. We implemented Asterism[1], a file sharing mobile application based on a P2P network and then we evaluate the P2P software affects to the performance of the mobile device.

Based on this goal, we defined the following Research Question (RQ):

RQ1: is it possible to transform mobile devices into decentralized distributed content delivery network without any central servers or authorities?

The remainder of this paper is structured as follows. Section II described related works. Section III introduce the background and the requirements of our proposed decentralized while Section VI reports on the implementation of the system. Section VII reports on the experimental analysis performed to compare the performance of our system. Finally, Section VIII draws conclusions and ideas for further development.

II. Related Work

Matuszewski *et al.* [7] analyzed the user' attitude towards mobile peer-to-peer (P2P) content sharing applications, highlighting the need for such applications and reporting that

[1]Asterism. Source code and Binaries: https://github.com/skvark/Asterism

most of the users they interviewed were willing to run the P2P application only for a short periods of time given that they are familiar with the people they are sharing content with. Heikkinen *et al.* [8] measured the mobile P2P TCP/IP traffic generated by GSM/UMTS networks of three major Finnish operators. Different P2P applications were identified by transport protocol port numbers. As a result, direct P2P traffic was not observed but instead 9-18 % unidentified traffic was detected, probably as a cause of the usage of P2P applications while using phones as wifi routers. In addition to direct network traffic analysis, a panel study was conducted. The study revealed that only one mobile P2P client application, Fring, had significant usage levels across all participants.

The BitTorrent protocol and different implementations based on it has been studied widely in mobile environments in many different contexts.

Nurminen *et al.* [9] proved that P2P content sharing is feasible from power consumption point of view. In the research the power consumption of a mobile BitTorrent client was measured. The power consumption was on the same level as mobile phone voice calls. It must be noted that the measurements were done in 2008: mobile devices and their properties, most notably power usage due to more powerful hardware, has significantly changed since then.

Bori *et al.* [10] studied the reliability of BitTorrent protocol in mobile environment based on a mobile BitTorrent application called DrTorrent with 5000 active users. They concluded that BitTorrent is generally realiable, but the client applications must be capable of handling different anomalies such as failed TCP connections and corrupted data.

A hybrid BitTorrent based content sharing solution has been also studied: Ekler *et al.* [11] proposed a new content sharing solution based on BitTorrent and central servers controlled by mobile operators. The solution supports both mobile and PC clients. The central server and and operator's central unit help to distribute the content more efficiently which in turn helps to reduce service provider costs. Two different BitTorrent client implementations, SymTorrent and MobTorrent, were also addressed in the paper by Ekler *et al.* [11]. When the implementations were compared, SymTorrent was faster than MobTorrent on both 3G and WLAN networks. According to the authors this is due to the fact that MobTorrent was implemented in Java ME and SymTorrent had been written natively in C++. The Java implementation suffers from the overhead of the Java Virtual Machine and socket handling limitations of Java ME.

The content lifecycle and client usage in BitTorrent network has been studied by the data provided by MobTorrent mobile application by Csorba *et al.* [12] and Ekler *et al.* [13]. In the papers, 2 and 3 to 4 years of data collected via the MobTorrent client was analyzed. The data suggests that mobile BitTorrent has become more popular every year. The results show also that there were more unfinished downloads than completed downloads caused by multiple simultaneous downloads of the same content. Three largest categories by accessed torrents were videos, applications and audio.

In addition to BitTorrent, also other P2P protocols has been studied from energy consumption point of view in mobile environments. Gurun *et al.* [14] studied Chimera P2P protocol with an application on a embedded device. Most of the energy was consumed by the wireless interface when it was waiting for transmissions in idle state. As a energy conservation technique the wireless card was switched to a low power mode when no network communication was observed. This approach resulted in power savings. As a result, the study suggested that P2P protocols and applications can be used on low-power embedded devices.

The general performance of DHT-based P2P overlays in mobile environments was measured by Chowdhury *et al.* [15]. Five different DTHs were analyzed: Chord, Pastry, Kademlia, Broose and EpiChord. The measurements were done in an simulated environment. According to the results, the best choice for a mobile P2P overlay under heavy churn is Kademlia due to its good 97 % lookup success ratio while consuming 316 bytes/s of bandwidth. Additionally, EpiChord performs also well by achieving a 63 % success ratio while consuming only 70 bytes/s of bandwidth.

Cherbal *et al.* [16] presented an improved Chord-based structured P2P approach to minimize physical distance of the nodes in a mobile environment. The approach is based on two-tier structure where main Chord ring contains supernodes and secondary Chord ring contains cellular nodes. The authors provide a mathematical analysis of the approach which proves that the approach is more efficient in mobile environments due to reduced physical distance of the nodes.

Khan *et al.* [17] introduced a P2P network data store targeted for mobile environments. Their goal was to mitigate the previous issues related to the resource limitations, by structuring the P2P network into redundant clusters of peers and by updating the routing tables with a gossip-based protocol. In their simulations they achieved 12- 48 % better look-up success rate than MR-Chord and Chord based P2P systems. Additionally, due to good churn and workload adaptation capabilities, they also managed to distribute requests more evenly than the two other systems.

In summary, the previous work has studied P2P networking in mobile environments from different perspectives including topics such as P2P usage level measurements, mobile BitTorrent clients, power consumption measurements and different DTH-based systems. This paper aims to extend the previous research by providing a fresh practical approach on P2P networking on mobile devices. Further, by creating an actual mobile application on top of an existing modern P2P network the behavior and performance of the system can be evaluated in real environment.

III. Platform Selection

In this section we first describe the different mobile file sharing systems available and how we selected the most appropriate for this work. Then we describe the different mobile operating systems where the file sharing system can be implemented together with the selection of the most suitable for our purpose.

IV. Modern P2P File Sharing

Many new peer-to-peer (P2P) content sharing networks have emerged in the last few years. Most prominent of these systems are InterPlanetary File System (IPFS) [18], Dat [19] and Swarm [20]. These three systems were selected to this comparison due to their popularity, open source implementations and the large development efforts behind them. Based on this comparison one of them will be selected to be used as the back end in the application implemented in this paper.

InterPlanetary File System is a P2P distributed file system [18]. Its ultimate target is to replace Hypertext Transfer Procotol (HTTP) and build a new decentralized Internet infrastructure. IPFS combines multiple different proven techniques of past successful distributed systems to create a new solution which attempts to connect all computing devices under a single worldwide file system. Among these are for example Kademlia DHT, BitTorrent and distributed version control system Git [21].

Dat is a P2P protocol which is designed for syncing versioned data [19]. It attempts to replace cloud based storage services such as Dropbox and Google Drive due to their centralized nature, high costs, slow transfer speeds and privacy issues. Dat has many similarities with IPFS: it's inspired by projects such as Git and BitTorrent. However, Dat's main idea is narrower: it offers a free and encrypted versioned P2P data syncing system.

Similarly to IPFS and Dat, Swarm is a distributed platform which provides P2P data storage services [20]. Swarm is closely related to the Ethereum blockchain ecosystem [22]. Its primary target is to serve as decentralized and redundant storage system for all data related to Ethereum. Unlike Dat and IPFS, Swarm does not require that the uploaders host the content themselves. This is possible due to a built-in incentive system which rewards nodes that are hosting content.

These three systems are compared from the perspective of the mobile application development. To be able to select the best system for the use case in this thesis and to be future proof, the following properties have been selected to the comparison:

- Initial release year
- Existing protocol implementations by programming language
- Developer community size, roughly estimated by the project's pages at Github
- Maturity of the software

IPFS and Dat have been published a few years before Swarm. Due to this, IPFS and Dat projects have had more time to develop and test the system. Dat is the only project which has been able to achieve a stable release. However, despite IPFS being still in alpha state it is being used all around the world. The IPFS reference implementation, go-ipfs, is mostly functional but it lacks some features of the original specification [23]. On the other hand, Swarm is only at a proof-of-concept phase and is more likely to undergo larger changes in the future.

Swarm is backed both by a very large user-base and developer community because it is tightly related to the Ethereum blockchain ecosystem. IPFS has a medium-sized community behind it and is being actively developed. Dat has the smallest developer community, but the development is active and it is unlikely that the developers would abandon the project at this point.

The main implementations of IPFS and Swarm are written in Go. Go is an open source programming language [26]. It makes it easy to run both IPFS and Swarm almost on any device. In addition to Go, IPFS has also a JavaScript implementation which makes it possible to run IPFS in web browsers. The development efforts of Dat are focused solely on JavaScript. Like IPFS, Dat can be run on web browsers. Outside browsers Dat needs a JavaScript runtime or some browser based environment to be able to run. This is a drawback, since it makes it harder to develop applications especially for mobile devices due to the special runtime environment needs.

While all the three projects are similar on higher level, their approaches to P2P content sharing are very different when observed more closely. Swarm is part of the large Ethereum ecosystem but it means that it requires Ethereum to work properly. Additionally, Swarm is not as mature as the other two projects. Dat has had multiple stable releases, but JavaScript makes it hard to integrate it into resource constrained mobile environments. The remaining candidate is IPFS. Go-ipfs, the reference implementation of IPFS, can be considered stable enough for a proof-of-concept mobile application in this paper.

V. InterPlanetary File System

InterPlanetary File System (IFPS) is a P2P network protocol that integrates multiple techniques to create a distributed and versioned file system with no single point of failure. This section goes through the basics of the architectural stack of IPFS and explains how IPFS works.

The IPFS stack can be divided into five sections [23] (see Figure 1 for more details).

Applications can be implemented on top of the IPFS protocol. Go-ipfs is the main reference implementation of the IPFS protocol [23]. Besides being able to run a full IPFS node, go-ipfs can used also as an application to control the node.

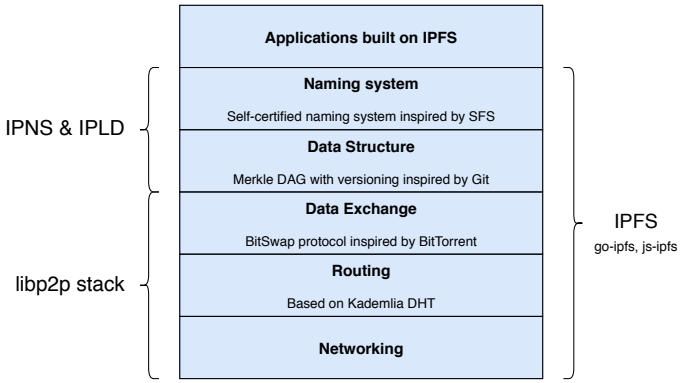

Fig. 1. Overview of the IPFS architectural stack. [18], [23]

TABLE I
COMPARISON OF MODERN P2P CONTENT SHARING SYSTEMS. [23]–[25]

Property	IPFS	Dat	Swarm
Initial Release	2014	2013	2016
Protocol Implementations	Go (reference implementation), JavaScript	JavaScript	Go
Community Size	Medium, 100+ contributors	Small, about 100 contributors	Large, hundreds of contributors
Implementation Maturity	Alpha, latest reference implementation version 0.4.17	Stable, latest version 13.11.4	Alpha (proof-of-concept), latest version 0.3

A. Existing IPFS Mobile Applications

For iOS or Sailfish OS there are no known IPFS applications while there is an Android IPFS mobile application called IPFSDroid [27], targeted primarily for developers. IPFSDroid ships with a full go-ipfs binary and runs go-ipfs as a separate daemon next to the application.

The software architecture used in the IPFSDroid application has many drawbacks. Handling the IPFS daemon as a separate process on a mobile device is complicated as the main application becomes responsible for the daemon process. In addition, the daemon starts a separate HTTP server. The application uses the HTTP interface provided by this server to access the IPFS. This adds extra overhead to all operations since the files must be transferred over HTTP to the IPFS daemon. On a mobile device this approach is sub-optimal and makes the application architecture complicated.

B. Operating System Selection

The operating system for which the application is implemented may limit the functionality of the application. This is true especially for the background execution requirement. More importantly, also the the flexibility of the platform must be considered from the application development point of view. In this comparison it is assumed that an interface to IPFS is provided via a shared library. Due to these reasons, three different mobile operating systems were selected as candidates: Android, iOS and Sailfish OS.

Java is the main programming language of Android applications. Android allows also low level application development in C++. This makes it possible to use external shared libraries via the C++ code. However, in the end a C++ application would need a bridge to Java code to be able use the normal Android application programming interfaces.

On iOS applications can be developed in Objective-C which makes it possible to link a shared library directly with the application [28]. Running long or indefinite background tasks is not allowed in iOS. Consequently, application with a IPFS node would most likely lead to rejection from the application store.

Sailfish OS uses C++ as the main programming language [29]. Therefore, shared libraries are supported natively by the environment. Sailfish OS allows indefinite execution of background tasks: all started applications run in the background unless explicitly closed.

Both Android and iOS could be used to implement the IPFS application but the application development and distribution would likely require much more work than on Sailfish OS. Additionally, Sailfish OS has a special focus on openness and user privacy [29]. These aspects align better with the mission of IPFS than the other two systems which depend heavily on massive centralized organizations.

C. Sailfish OS

Sailfish OS is a mobile operating system developed by Jolla [29]. Version 1.0 of the operating system was released in 2013. Sailfish OS has been actively developed since: the latest version of the operating system is 2.2.1.18 released in September 2018.

Sailfish OS is based on Linux kernel and resembles many desktop GNU/Linux distributions [29]. Despite being a mobile operating system it can be run on a variety of devices including desktop systems. The Linux kernel of different Sailfish OS mobile devices is usually based on a device specific Android kernel. In practice, most of the mobile devices which run Sailfish OS have been originally Android devices. Sailfish OS has been ported on these devices by hardware adaptation work done by Jolla or community members. A separate glue-layer between Android-specific device drivers and Sailfish OS makes it possible the reuse the Android drivers in Sailfish OS. Sailfish OS can also run Android applications through a separate proprietary Android runtime. On top of these stacks there is a proprietary Sailfish user interface layer. The native applications and user interfaces are developed with cross-platform framework Qt by combining C++ and Qt modeling language (QML).

For the purposes of this work, Sailfish OS can be seen to be the best platform for the IPFS application implementation. The following aspects of the operating system and the application development environment were considered to be better on Sailfish OS than on Android and iOS:

- True multitasking support. Running applications are killed only in the rare case of out-of-memory event.
- Applications are programmed with C++. A go-ipfs based shared library can be used directly from C++ without extra software layers.
- Native access to full GNU/Linux system and terminal without rooting or jailbreaking the device. This makes system metrics monitoring straightforward.
- Access to many command line applications which would be hard to come by in the other two operating systems.

VI. Implementation

In this Section, we describe the implementation of the mobile file sharing application and its architecture.

A. Requirements

We identified six requirements for the system to be implemented:

1) The application can start/stop a fully working IPFS node.
2) The application is able to run in the background while other applications are running.
3) The application can display basic information about the node and its status in the user interface.
4) The application user interface can be used to change basic settings of the node.
5) The application user interface allows adding content to IPFS and retrieving content from IPFS.
6) The application provides a file system style interface for managing the added files.

B. Overview of the Architecture

The file sharing application architecture was designed keeping in mind the fact that the application will be running on a mobile device with restricted resources. Running a separate go-ipfs based on daemon is suboptimal and impractical approach in mobile environments. Additionally, shipping multiple executables in an application package to the official Sailfish OS application store is not allowed.

Instead of interfacing with a separate daemon process we interface the application directly with go-ipfs via a separate wrapper library. In addition to easier application packaging, the direct integration also removes any overhead caused by the HTTP server interface of the go-ipfs daemon. The resulting library is not Sailfish OS specific: if needed, the library can be used also in other operating systems in the future.

The high level overview of the architecture can be seen in Figure 2. The resulting IPFS client application was named as Asterism. The go-ipfs wrapper library is called as Lipipfs and it is embedded to the application package. Together, these two programs form the full application package.

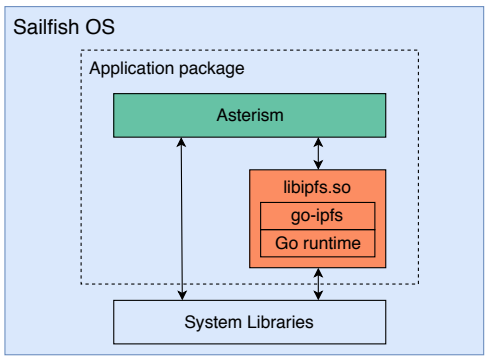

Fig. 2. Overview of the file sharing application architecture.

C. Mobile Client Application

The application utilizes Libipfs [2] to provide a user interface for sharing files and interacting with the IPFS. Asterism was written for Sailfish OS in C++ and QML with the Qt framework based on the system requirements identified before.

The user interface of Asterism consists of four main views. The first view allows to view basic information of the IPFS node and makes it possible to start and stop the node. The second view provides an interface to the mutable file system (MFS) with the possibility to add files to the IPFS. The added files are pinned automatically. The MFS can be browsed like a regular file system. Additionally, new directories can be created. Screenshots of these views can be seen in Figure 4.

The third view includes a listing of the currently connected peers. The list includes the node identifier and address for each peer. Peer count could be limited via go-ipfs configuration file. Asterism does not provide user interface for changing the peer limit values.

The last view is the settings view. In the settings view user can change the mode of the IPFS node from full DHT mode to client only mode. This mode determines how the DHT works. In the client only mode the node does not serve requests to the rest of the network and should limit the network usage of the node to some extent.

In the settings view there are also settings for maximum repository size and manual garbage collection. The repository size limits the disk space used by the IPFS. Garbage collection allows to remove all unused or cached resources from the repository manually. An example of this kind of content are unpinned files. The garbage collections is also run periodically by the go-ipfs and when the maximum disk usage limit is approaching. Screenshots of the peers and settings views can be seen in Figure 5.

Special actions, such as stopping or starting the node, can be accessed via the pulley menu at the top of the screen. It is a part of Sailfish OS user interface paradigm and can be accessed by pulling the screen down.

Asterism uses the exported functions of Libipfs to interact with the IPFS. An example of the file add sequence is visualized in the Figure 3. The add function and most of the other exported functions are run in the background as goroutines. These asynchronous operations are presented in the sequence diagram by the goroutine bounding box.

To add a file to the IPFS, Asterism passes a file path to Libipfs add function which will in turn add the file to the IPFS with the help of go-ipfs. The return value of the file add operation is a hash identifier of the file contents. Effectively this means that the filename information has been lost during the operation.

Asterism uses the MFS implementation of go-ipfs to preserve the filenames and to provide a file system like interface to the added files. In practice, an entry by the original name of the file is created to the MFS by copying the hash which was returned from the add operation to the MFS. After the copy

[2]https://github.com/skvark/libipfs

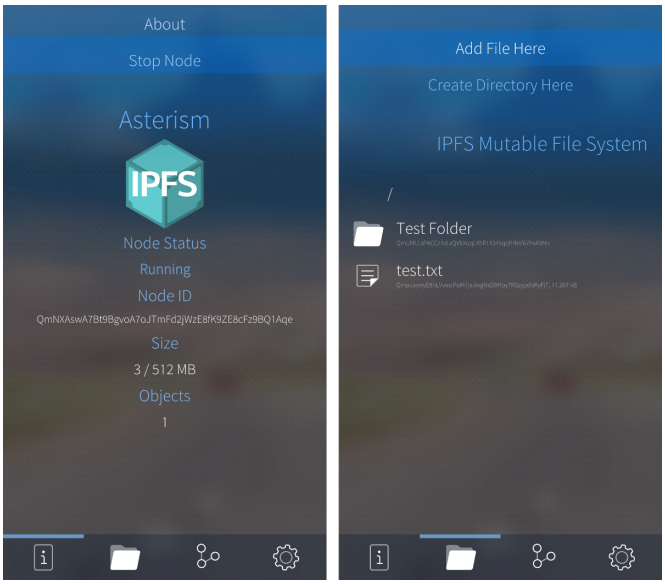

Fig. 4. Asterism info and MFS views.

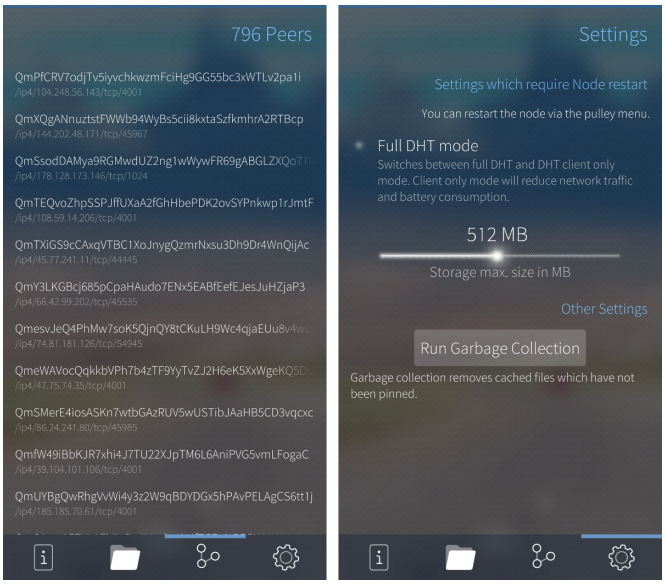

Fig. 5. Asterism peers and settings views.

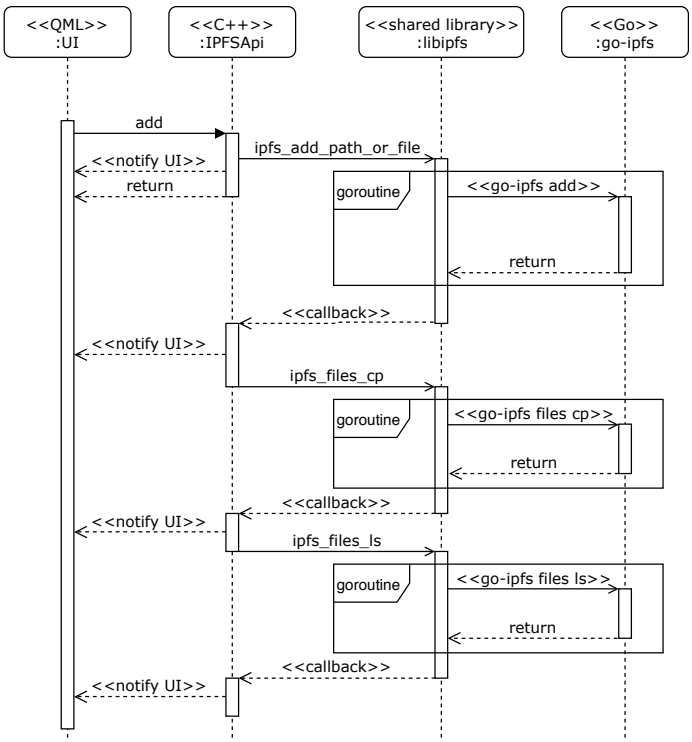

Fig. 3. File sharing application file add sequence diagram.

operation has been finished the MFS contents are refreshed by fetching the latest file listing from go-ipfs.

VII. Experimental Evaluation

In this Section, we focus on the evaluation of the file sharing application.

To answer to the research question, the application performance was measured on several Sailfish OS based mobile devices. The performance measurements include battery consumption and network usage measurements. Additionally, the general functionality and usability of the application is evaluated.

In the next sub-section, we fist introduce the devices with the performance measurement setup, and then we present the results.

A. Measurements

Capabilities of mobile devices are much more limited than normal desktop computers. P2P applications have a continuous need for data exchange caused by the data lookups and changes in the network routing. This will have an effect on the network usage: a P2P application will use network more actively than a normal centralized network architecture based application.

Consequently, active network usage will most likely have a negative effect on the battery life of a device. Therefore, both the battery consumption and network usage were measured while the file sharing application was running on a mobile device. With these measurements it can be evaluated is it worthwhile to run a P2P application on a mobile device.

The measurements were done on official Sailfish OS version 2.2.1.18. It was the latest available version when the measurements were performed. The go-ipfs version used by the Asterism file sharing application was 0.14.7. During measurements the Android support layer of the Sailfish OS was turned off to avoid possible measurement bias caused by the additional power consumption of the layer.

The battery consumption measurements were done on three different devices. As reported in Table II, the devices fall into different categories, also considering their price point and features. All three devices have been released during a timespan of one year in 2015 and 2016. Two of the devices are mobile phones and one of them is a tablet.

The first device, Sony Xperia X, is the latest and most powerful of the devices. It can be seen as a mid-range device

TABLE II
THE PROPERTIES OF THE DEVICES USED IN MEASUREMENTS. [30], [31]

Property	**Sony Xperia X**	**Jolla C**	**Jolla Tablet**
Release Date	2016, February	2016, May	2015, July
Chipset	Qualcomm MSM8956 Snapdragon 650	Qualcomm MSM8909v2 Snapdragon 212	Intel Atom Z3735F
Central Processing Unit (CPU)	4 x 1,4 GHz Cortex-A53 & 2 x 1,8 GHz Cortex-A72	4 x 1,3 GHz Cortex-A7	4 x 1,8 GHz
Battery	2620 mAh	2500 mAh	4450 mAh
Network Interfaces	GSM / HSPA / LTE & WLAN (Wi-Fi 802.11 a/b/g/n/ac, dual-band)	GSM / HSPA / LTE & WLAN (Wi-Fi 802.11 b/g/n)	WLAN (Wi-Fi 802.11 a/b/g/n, dual-band)
Talk Time	19 h	20 h	N/A

which represents most of the commonly used mobile phones. The second device is the Jolla Tablet. The tablet was selected for measurement to get samples also from a device which is not a mobile phone. On the tablet measurements were done only on WLAN interface because the tablet has no cellular connectivity. The last device, Jolla C, is a low-end mobile phone with a less powerful processor than the two other devices. Jolla C is also known as rebranded Intex Aqua Fish device.

1) Power Consumption: The power consumption measurements were executed in multiple different configurations. IPFS node was run in full DHT mode or in client only mode. Additionally, the IPFS modes were combined with different network combinations: both 4G and WLAN enabled, only WLAN enabled and only 4G enabled. The idle power consumption was measured for all devices while all network interfaces were enabled and no applications were running. During all measurements the screen of the device was turned off. No content was added to the IPFS and no content was downloaded from the IPFS before or during the measurements.

The power consumption was measured with Charge Monitor application available in the official Sailfish OS applications store. The application is able to log the battery level of the device into a log file. All measurements were started from a battery level of 100 %. Battery levels were logged for 4 hours. A linear regression line was fitted to the obtained data points. With the fitted line the battery life of the device can be estimated with a reasonable accuracy. The estimated battery life numbers in the next paragraphs have been rounded to 10 minute accuracy and the percentage differences in 5 % accuracy.

B. Results

The firsts two figures, 7 and 8, show measurements which were done on the mobile phones when all network interfaces were enabled. The idle consumption of the devices is also included in these two figures. The battery of the Xperia X is estimated to last about 780 minutes in full DHT mode and about 850 minutes in client only mode. For Jolla C the same numbers are 620 minutes and 710 minutes.

Both devices have similar power consumption trends. The idle power consumption is low which is also expected because no applications were running. The difference between the battery life of the devices is large. On full DHT mode the battery of the Xperia X lasts significantly longer than Jolla C's. Similarly, Xperia X performance is better than Jolla C's when the IPFS node is run in the client only mode.

The difference in power consumption between the DHT modes on the both devices is noticeable. On Xperia X the client only mode adds about 10 % more battery life when compared to the full DHT mode. On Jolla C the change is a bit larger, about 15 %. As the estimated battery lives are better than in the 4G only figures and worse than in WLAN only figures, IPFS node has probably defaulted to the WLAN interface during the measurements.

The next three figures, 9, 10 and 6, are WLAN only measurements. For Xperia X, the estimated battery life in full DHT mode is 880 minutes and in client only mode 1080 minutes. For Jolla C the estimations are 720 minutes in full DHT mode and 1020 in client only mode. For the Jolla Tablet the numbers are 720 minutes and 820 minutes.

The battery lives of the Xperia X and Jolla C are longer on WLAN only when compared to the measurements where both WLAN and 4G were enabled. The percentage changes between the DHT modes are about 20 % on Xperia X, 40 % on Jolla C and 15 % on Jolla Tablet. The Tablet results are similar with the two mobile phones. However, the results indicate that the WLAN interface of the Tablet is less energy efficient because it should last longer given the capacity of the battery.

The last set of power consumption measurements were done on 4G interface only. The results of these measurements are in figures 11 and 12. For the 4G only measurements, the estimated battery life for Xperia X is 420 minutes in full DHT mode and 430 minutes in client only mode. For Jolla C the same numbers are 420 minutes and 440 minutes.

The 4G only results have close to none difference between the different DHT modes. Running IPFS node on 4G clearly consumes much more power than when using only WLAN. Both devices perform very similarly on these measurements: battery life on both devices is approximately the same. The

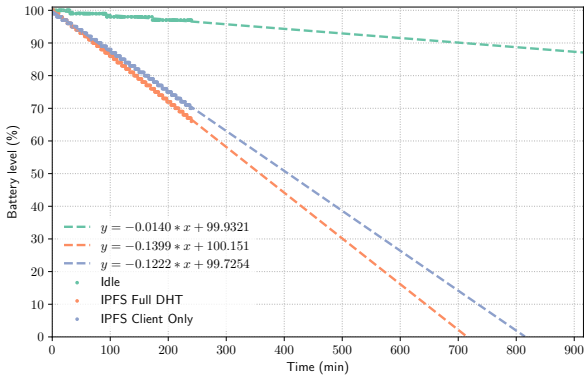

Fig. 6. Jolla Tablet WLAN battery consumption.

WLAN interface is far more energy efficient than 4G on both devices.

1) Network Usage: The network usage measurements were done only on Xperia X. Network usage was measured in full DHT mode and client only mode with Nethogs [32] traffic monitoring tool. Both networking interfaces, WLAN and 4G, were turned on because it is the most likely default setup during common mobile phone use. After Asterism had been started Nethogs was run for 140 minutes. The total sent and received data was logged to a log file in MB every 10 seconds

The results of the measurements are presented in Figure 13 and 14. Asterism downloaded in both modes approximately the same amount of data. Client only mode reduced the sent data about 30 %. The amount of the downloaded and uploaded data is very large. The application downloaded close to 800 MB in 140 minutes in both modes. Upload numbers were about 440 MB full DHT mode and about 300 MB in client only mode.

The networking layer used in go-ipfs was able to bypass Network Access Translation (NAT) and possible firewalls of the cellular network operators. This was confirmed by adding previously unknown content to the IPFS while running only on 4G. After this the content could be downloaded from a public IPFS gateway indicating that go-ipfs can upload content through cellular networks to the other nodes of the P2P network.

C. Discussion

The analysis and discussion in this section is based on the results in Section VII-A. The results indicate that running a full IPFS P2P network node on mobile devices is possible. However, the node causes large amounts of network traffic and affects negatively to the battery life of the device.

It is important to notice that no content was downloaded or uploaded during the measurements. All network traffic was caused by the chattiness of the IPFS protocol. DHT routing updates cause some of this traffic and it is likely that the IPFS Bitswap protocol is also responsible partly for the traffic. The amount of network traffic increases if a node downloads or adds some popular content and thus participates in the distribution of that content. This behavior is caused by the data exchange protocol which is similar to BitTorrent.

The go-ipfs implementation is not mature and there is a lot of room for optimization. On a mobile device the amount of traffic caused by the IPFS node is not acceptable in most cases due to the monthly upper traffic limits on mobile network subscriptions. Setting the DHT mode to client only has only minimal effect to the amount of the sent data. Therefore, traffic limit setting should be built to the IPFS protocol implementations to avoid excessive bandwidth usage.

Due to the constant network traffic the device can never enter to deep sleep mode. In the idle measurements the deep sleep behavior could be observed. The battery level logs contained longer gaps because the values were logged only when the device periodically briefly woke up to perform some tasks. The constant network traffic caused by the IPFS node prevents the device from entering to the deep sleep mode. Consequently, this behavior also prevents constant churn caused by the disconnections despite affecting negatively to the battery life.

The measurements were done on a stable environment. The devices were located close to a cellular tower and a WLAN router. Device movement and connection quality would most likely have an additional negative effect to the battery life. Additionally, mobile devices are usually used also for other tasks. Browsing the web or playing games while the IPFS node is running would consume the battery even faster.

The mobile BitTorrent measurements done by Nurminen *et al.* [9] had similar results when compared to our results. Client-only mode reduced the power consumption and the cellular network interface was less energy efficient than WLAN. The power consumption was on the same level with phone calls when run on 3G. In contrast, Asterism consumed more power than phone calls: the results indicate that on 4G the battery life is about one third of the talk time of the Xperia X and Jolla C. However, on 3G connection Nurminen *et al.* performed the measurements only for content download since the BitTorrent implementation could not bypass NAT and firewalls of cellular operators. Due to this, the results are not directly comparable.

Asterism was able to upload previously unknown content through the cellular network connection. This is an important finding: if the mobile node is not able to upload new content on cellular networks, the node cannot act as a full peer and only consumes the resources of the rest of the network. While the client only mode can be desirable due to the bandwidth and battery life savings the negative effect must be also considered. If a large amount of mobile client only IPFS nodes were to suddenly to join the IPFS, the network would not likely perform as well as before.

VIII. Conclusion

In this work we examined the possibility to transform mobile devices into decentralized content delivery network without any central servers or authorities by implementing a peer-to-peer mobile file sharing application and measuring how the application affects to the performance of the mobile device. We implemented the application based on the Sailfish operating system and on the InterPlanetary File System.

The file sharing application was developed in two phases. In the first phase a wrapper library was written for the reference implementation of the InterPlanetary File System. With this

approach a separate daemon process was avoided. In the second phase the shared library was utilized in the implementation of the mobile file sharing application. The application provides an easy to use user interface to the peer-to-peer file system. The user interface makes it possible to manage the added content in a way similar to common file system browsers. Additionally, the application can show basic information about the node and peer nodes as well as to control some settings of the node.

The network usage and the power consumption of the application was measured on three different devices. Two of the devices were mobile phones and one of them was a tablet. No content was added or fetched from the network during the measurements. All devices were able to run the application without problems. The application was able to upload content not only via WLAN but also through cellular connection.

The results show that the application consumed large amounts of network bandwidth. The traffic was caused by the chattiness of the peer-to-peer network protocol. Distributed hash table routing updates and the BitSwap protocol of the InterPlanetary File System are the most likely causes for the excessive bandwidth usage. When the distributed hash table routing mode was changed to client only mode the amount of the sent data was reduced by about 30 %.

Due to the constant network traffic the devices could never enter to deep sleep mode. This led to large power consumption. On WLAN interface the battery of the measured devices was estimated to last about 720 to 880 minutes depending on device. When the distributed hash table routing mode was changed to client only mode the devices gained 15 to 40 % more battery life. When run on 4G interface only the estimated battery life dropped down to 420 minutes on both mobile phones which is about one third of the estimated talk time of the phones. Client only routing mode had negligible effect on the battery life when the application was run only on 4G connection.

The amount of bandwidth consumed by the application is too excessive for continuous usage due to the data limits in usual mobile network data subscriptions. With a unlimited data subscription the application could be run constantly given that a user accepts greatly degraded battery life. The application could be also highly beneficial for phones used as edge devices, always connected to a power source.

Therefore, decentralized solutions could be potentially used in the future to replace centralized clouds and servers on mobile devices. However, for mainstream use cases the InterPlanetary File System needs network usage optimizations or bandwidth limits.

REFERENCES

[1] "Cisco global cloud index: Forecast and methodology, 20162021," Cisco, White Paper, 2018. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf

[2] P. De Filippi and S. McCarthy, "Cloud computing: Centralization and data sovereignty," *European Journal for Law and Technology*, vol. 3, no. 2, 2012. [Online]. Available: http://ejlt.org/article/view/101/234

[3] M. Rosenberg, N. Confessore, and C. Cadwalladr, "How trump consultants exploited the facebook data of millions," New York Times, 2018. [Online]. Available: https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html

[4] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Int. Conf on Peer-to-Peer Computing*. IEEE, August 27–29 2001, pp. 101–102.

[5] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction," in *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, March 12–16 2016, pp. 64–76.

[6] G. Fettweis and S. Alamouti, "5g: Personal mobile internet beyond what cellular did to telephony," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 140–145, 2014.

[7] M. Matuszewski, N. Beijar, J. Lehtinen, and T. Hyyrylainen, "Understanding attitudes towards mobile peer-to-peer content sharing services," in *Int. Conf. on Portable Information Devices*. IEEE, 2007, pp. 1–5.

[8] M. V. Heikkinen, A. Kivi, and H. Verkasalo, "Measuring mobile peer-to-peer usage: Case finland 2007," in *Int.Conf. on Passive and Active Network Measurement*. Springer, 2009, pp. 165–174.

[9] J. Nurminen and J. Nyrnen, "Energy-consumption in mobile peer-to-peer-quantitative results from file sharing," in *Consumer Communications and Networking Conf.* IEEE, 2008, pp. 729–733.

[10] A. Bori and P. Ekler, "The analysis of bittorrent protocol reliability in modern mobile environment," in *Eastern European Regional Conf. on the Engineering of Computer Based Systems*. IEEE, 2013, pp. 120–126.

[11] P. Ekler, I. Kelényi, I. Dévai, B. Bakos, and A. Kiss, "Hybrid peer-to-peer content sharing in mobile networks." *JNW*, vol. 4, no. 2, pp. 119–132, 2009.

[12] K. Csorba, P. Ekler, and I. Kelényi, "Analyzing content life cycle in mobile content sharing environment," in *East. European Conf. on the Engineering of Computer Based Systems (ECBS-EERC)*. IEEE, 2011.

[13] P. Ekler and K. Csorba, "The usage and behavior patterns of mobile bittorrent clients," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 18, no. 3, pp. 320–323, 2014.

[14] S. Gurun, P. Nagpurkar, and B. Y. Zhao, "Energy consumption and conservation in mobile peer-to-peer systems," in *International workshop on Decentralized resource sharing in mobile computing and networking*. ACM, 2006, pp. 18–23.

[15] F. Chowdhury, J. Furness, and M. Kolberg, "Performance analysis of structured peer-to-peer overlays for mobile networks," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 32, no. 5, pp. 522–548, 2017.

[16] S. Cherbal, A. Boukerram, and A. Boubetra, "An improvement of mobile chord protocol using locality awareness on top of cellular networks," in *5th Int. Conf on Electrical Engineering-Boumerdes (ICEE-B)*. IEEE, 2017, pp. 1–6.

[17] M. A. Khan, L. Yeh, K. Zeitouni, and C. Borcea, "Mobistore: A system for efficient mobile p2p data sharing," *Peer-to-Peer Networking and Applications*, vol. 10, no. 4, pp. 910–924, 2017.

[18] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[19] M. Ogden, "Dat - distributed dataset synchronization and versioning," 2017.

[20] "Swarm," Swarm, 2018. [Online]. Available: http://swarm-gateways.net/bzz:/theswarm.eth/

[21] "Git," Git, 2018. [Online]. Available: https://git-scm.com/

[22] "Ethereum project," Ethereum, 2018. [Online]. Available: https://www.ethereum.org/

[23] "Ipfs github organization," 2018. [Online]. Available: https://github.com/ipfs/

[24] "Dat project github organization," 2018. [Online]. Available: https://github.com/datproject

[25] "Ethereum github organization," 2018. [Online]. Available: https://github.com/ethereum

[26] "The go programming language," 2018. [Online]. Available: https://golang.org/

[27] "Ipfsdroid," 2018. [Online]. Available: https://github.com/ligi/IPFSDroid

[28] "Apple documentation archive," 2018. [Online]. Available: https://developer.apple.com

[29] "Sailfish os," Jolla, 2018. [Online]. Available: https://sailfishos.org/

[30] "Gsmarena," GSMArena, 2018. [Online]. Available: https://www.gsmarena.com/

[31] "Devicespecifications," DeviceSpecifications, 2018. [Online]. Available: https://www.devicespecifications.com/

[32] "Nethogs," 2018. [Online]. Available: https://github.com/raboof/nethogs

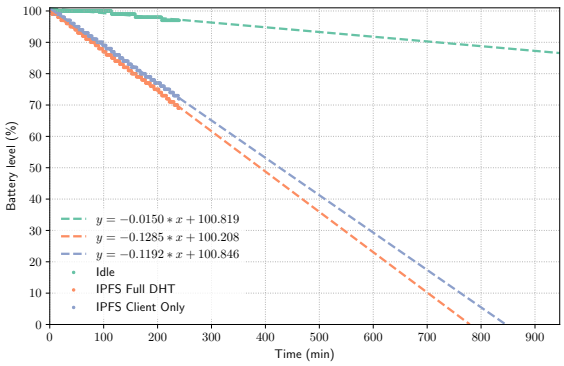

Fig. 7. Battery Consumption, Xperia X, 4G and WLAN

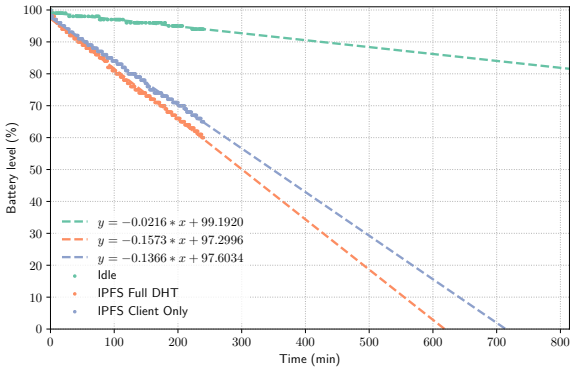

Fig. 8. Battery Consumption, Jolla C, 4G and WLAN

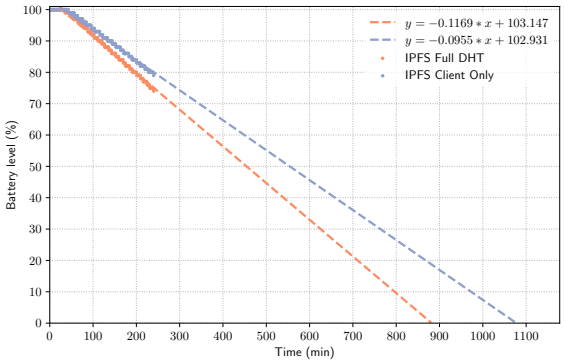

Fig. 9. Battery Consumption, Xperia X, WLAN

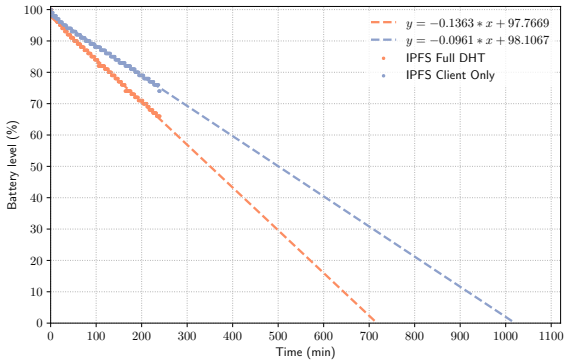

Fig. 10. Battery Consumption, Jolla C, WLAN

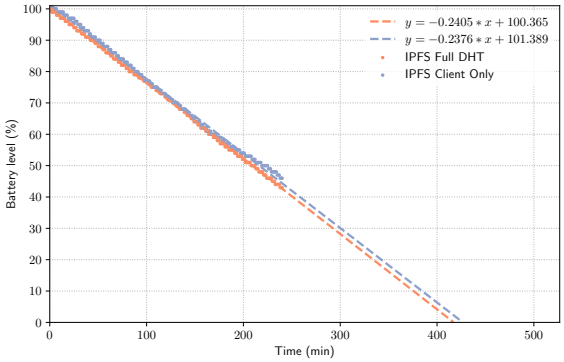

Fig. 11. Battery Consumption, Xperia X, 4G

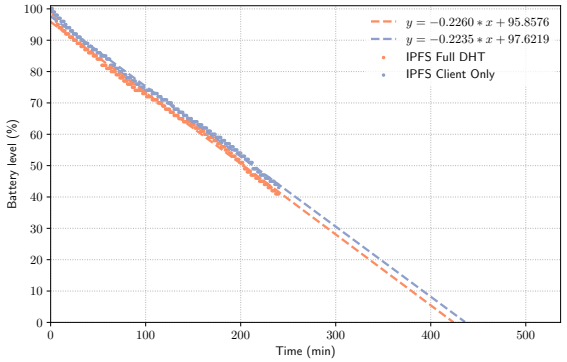

Fig. 12. Battery Consumption, Jolla C, 4G

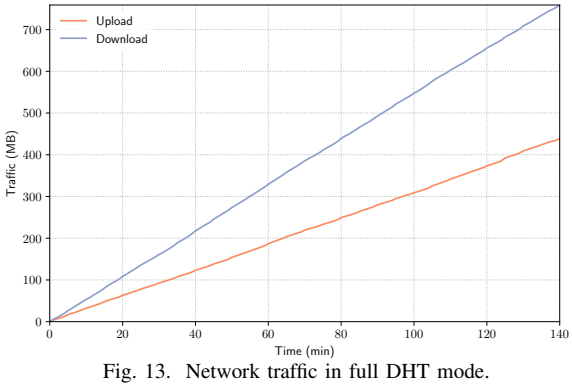

Fig. 13. Network traffic in full DHT mode.

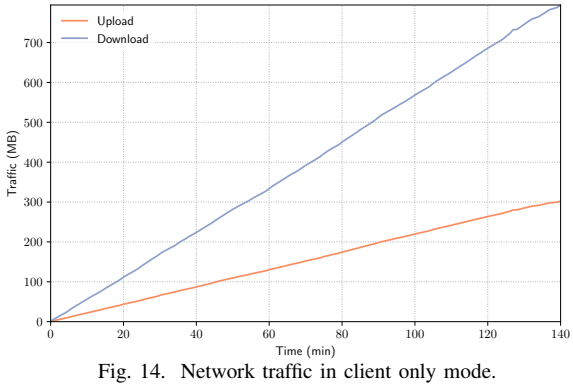

Fig. 14. Network traffic in client only mode.