Cloudless Wide Area Friend-to-Friend Networking Middleware for Smartphones

Jo Inge Arnes and Randi Karlsen
Department of Computer Science, UiT The Arctic University of Norway, Tromsø, Norway

Keywords: Cloudless Peer-to-peer Mobile Communication, Friend-to-friend Networking, Unreachable IP Address.

Abstract: Swirlwave is a middleware that enables peer-to-peer and distributed computing for Internet-connected devices that lack publicly reachable Internet Protocol (IP) addresses, that can be expected to disconnect from the network for periods of time, and that frequently change network locations. This is the typical case for smartphones. The middleware fits into the friend-to-friend subcategory of peer-to-peer systems, meaning that the overlay network is built on top of already existing trust relationships among its users. It is independent of clouds and application servers, it supports encryption for confidentiality and authentication, and it aims to be easily extensible for new applications. The solution described in this paper is implemented for smartphones running the Android operating system, but its principles are not limited to this.

1 INTRODUCTION

Smartphones and their Internet features are used in a wide range of areas in people's lives. Examples are online banking, keeping up with news, education, career, looking up health information, sharing pictures and videos, social networking, navigating in traffic, just to mention a few. In the U.S. 77 % use their phone to share pictures, videos, or comments about events happening in their community, and nearly one-in-three smartphone owners frequently use their phone for navigation (Smith, 2015).

The way a user accesses such services is generally through apps supported by an online service, where data is stored in the service provider's data centers or cloud services. This makes it possible to access data from multiple devices, and share with others, for example by sending a picture to friends through Snapchat[1]. In this case, the picture is uploaded to Snapchat's servers, the recipient friends are notified and can see the picture in their Snapchat app, which downloads the picture from the Snapchat data center.

Smartphone apps that let users share and communicate with others over the Internet are commonly supported by cloud services, where communication passes through the cloud data centers. This can be a source of privacy concerns. In addition to storing data that the user uploads, many other types of data concerning the app are managed there. The provider of the app usually gathers metadata about user activities, such as whom they communicate with, when, where, how often and about what.

The extensive use of cloud services also raises concerns with respect to waste of computing resources. Smartphones today have as much processing power, memory, and storage capacity as a typical desktop PC a decade ago. With the increasing popularity of smartphones (Myers, 2016), it seems sensible to explore ways to harness more of these hardware resources.

An alternative to the cloud-based solutions is to enable smartphones to communicate directly in a peer-to-peer fashion over the Internet. However, this is not a trivial solution, since smartphones usually lack publicly reachable IP addresses and often change networks, which makes it difficult to keep track of the devices' addresses. Most smartphone apps of today, therefore, depend on clouds or application servers as middlemen to enable communication between devices.

In this work, we describe a novel approach to mobile peer-to-peer communication in wide area networks, which allows direct communication between devices that frequently change networks and lack public IP addresses. We introduce *Swirlware*, a middleware that enables wide area peer-to-peer communication for smartphones, without the need for clouds or application servers for storing, processing, or shar-

[1] https://www.snapchat.com

Arnes, J. and Karlsen, R.
Cloudless Wide Area Friend-to-Friend Networking Middleware for Smartphones.
DOI: 10.5220/0006830102550264
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 1: DCNET, ICE-B, OPTICS, SIGMAP and WINSYS*, pages 255-264
ISBN: 978-989-758-319-3

ing data. Our approach also supports incorporating smartphones as nodes in a peer-to-peer or distributed system, so that storage and processing capacities of the smartphones can be utilized as part of a bigger whole.

Experiments show that Swirlwave handles cloudless, mobile peer-to-peer communication well. It enables smartphones to be directly reached and supports continued communication when devices move between networks.

In the following, we first compare Swirlwave to related work. We then give an overview of the proposed system, describing the architecture and underlying communication. Section 4 describes the Swirlwave middleware. Experiments and results are presented in Section 5, while the last sections present discussions and conclusion.

2 RELATED WORK

The Swirlwave middleware provides a novel solution to peer-to-peer mobile communication, where obstacles, such as unreachable IP addresses, disconnections and frequently changing network locations, are handled. We here describe how Swirlwave relates to previous work on mobile communication.

Popescu et al. (Popescu et al., 2006) describe a friend-to-friend (F2F) architecture, called Turtle, for safe sharing of sensitive data. As Swirlwave, it builds an overlay network from pre-existing trust relationships. Turtle differs from Swirlwave by being a theoretical description of a file sharing network architecture, where search queries are flooded through the network. Swirlwave, on the other hand, is a middleware enabling friend-to-friend networking without being tied to specific applications.

A variety of apps for using smartphones as servers exist, for example, web servers for Android, but they only work as part of a local area network. To connect, clients must be on the same local area network as the server. This is a serious restriction when using mobile devices. Swirlwave enables smartphone server apps to be available outside local area networks and continue communicating with clients despite network changes.

Orbot[2] allows smartphones to be reached outside local area networks, but it has no mechanism for changing addresses when the smartphone changes location. This problem is solved in Swirlwave, which handles address changes.

Thali[3] is a Microsoft sponsored experimental platform for building peer web. It is described as an open-source software platform for creating apps that exploit the power of personal devices and put people in control of their data. Thali planned to use the Tor Onion Service protocol[4] (called hidden services at the time), the same protocol as Swirlwave builds on, but abandoned the idea, since onion services were designed for stationary services, not mobile ones.[5] It is clear from the project's homepage that Thali instead communicates over Bluetooth Low Energy (BLE), Bluetooth, and Wi-Fi direct, none of which are wide area communications. In contrast to Thali, Swirlwave provides functionality that enables the use of the Tor Onion Service protocol on mobile devices, can thus support wide area mobile communication.

3 MOBILE PEER-TO-PEER COMMUNICATION WITHOUT PUBLIC IP ADDRESS

This section describes the architecture for Swirlwave and how it builds on Tor and the Tor Onion Service protocol[6]. We also describe the problem of unreachable IP addresses.

3.1 Unreachable Addresses

Usually, when a personal computer is connected to the Internet, other computers cannot directly contact it. This is because of network address translation (NAT). The computer can initiate contact with a server, but it cannot act as a server itself. The same is true for smartphones. The reason is that computers are not directly connected to the Internet, but are part of a local area network (LAN) that communicates with the outside world through a router.

Devices on a LAN are assigned IP addresses that are only valid inside the LAN. In the most common configuration, IPs are assigned by a DHCP-server. A device is given an address when it connects to a LAN, but this address can be different the next time the device connects to the same network.

When a computer connects to a server on the Internet, the server will see the IP of the router. The server sends its replies to the router, which performs network address translation and routes the traffic to the correct computer (Comer, 2014). IPs of computers inside the LAN are not reachable from the Internet. This is also

[2]https://guardianproject.info/apps/orbot

[3]http://thaliproject.org

[4]https://www.torproject.org/docs/onion-services.html.en

[5]http://thaliproject.org/ThaliAndTorHiddenServices

[6]https://www.torproject.org

the case for smartphones connected to the Internet via local Wi-Fi or cellular data such as 4G.

3.2 Architecture

A system based on Swirlwave has a shared-nothing architecture with independent and self-sufficient nodes that do not share memory or disk storage. The nodes will generally run on separate physical devices, most notably smartphones connected to the Internet through Wi-Fi or cellular data. It is a peer-to-peer system that does not rely on cloud computing or application server middleboxes.

3.2.1 Friend-to-Friend Network

Swirlwave is based on friend-to-friend (F2F) networking (Bricklin, 2000), a category of unstructured, private peer-to-peer networking where peers only connect directly to already known peers (friends) (Rogers and Bhatti, 2007). Friendships are commutative, but not automatically transitive.

In Figure 1, peers A, B, and D are all friends and can connect to each other. Peer C is only friends with B, and cannot contact A and D directly. However, it is possible to reach unknown, faraway peers indirectly. Using a far-reaching query, A can ask B to relay the query to its friends, which again could query friends, and so on.

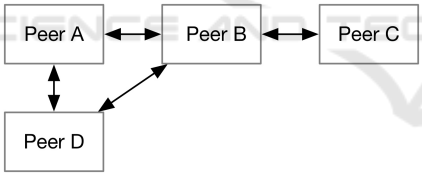

Figure 1: Friend-to-friend network.

Friend-to-friend networks are useful for distributed systems with a predefined set of dedicated nodes that should not be available to everyone. It can, for example, be used for connecting a set of company-owned smartphones or for setting up a social network of friends to chat or share data.

3.2.2 Swirlwave Middleware

Swirlwave is designed as a middleware that facilitates communication between applications on mobile devices. It is located between the application and operating system (including transport layer services, such as TCP/IP). The devices are connected to the Internet, and traffic is routed through the Tor overlay network.

All peers can be clients and servers at the same time; they can expose several services for peers to consume, and they can be clients to services published by other peers. To contact a server, the client application need not know the address of the server peer. Neither does it need to know how to connect to the underlying communication service Tor.

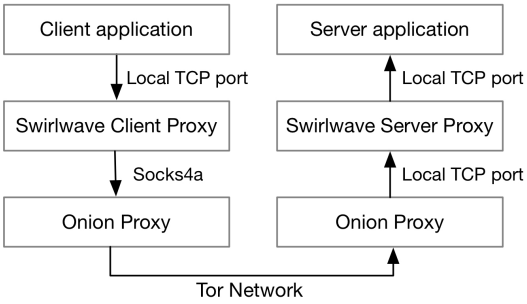

Figure 2: Proxying from client to server.

Applications built on top of Swirlwave communicate over TCP and register as plug-ins with Swirlwave along with their capabilities, such as properties of the application. Swirlwave is designed so that existing applications and libraries easily can make use of it. This is achieved by creating two proxies, a client- and a server-side proxy.

Figure 2 shows two peers, one acting as client, the other as server. Any client application using TCP connections can connect to the locally running Swirlwave proxy, and the middleware will automatically route the traffic to the correct peer. The client proxy listens to connections from local clients on a range of ports. When a client application connects, the port number is used to find the correct peer and requested service. The Swirlwave client proxy connects to the locally running onion proxy through the SOCKS4a protocol (Lee, 2012), which sends the traffic through the Tor network.

On the other end, a Swirlwave server proxy receives data from the onion proxy and directs the traffic via ordinary TCP to the correct service running locally on the receiving peer.

3.3 Tor and Tor Onion Services

The main objective of Tor[7] is anonymity, not connectivity. It is designed to conceal online traffic from surveillance and monitoring by relaying through several nested proxies, compared to the layers of an onion. Per February 2018 the network consisted of over 6000 volunteer relays[8].

Tor is a public overlay network where encrypted traffic is routed through at least three onion routers before reaching its destination. On each end of the net-

[7]https://www.torproject.org
[8]https://metrics.torproject.org

work, there are onion proxies. The client-side onion proxy has access to a directory of onion routers, and when choosing the preferred onion routers, a circuit is built. Each onion router in a circuit knows only its predecessor and successor.

At the destination end of the circuit, an onion proxy receives the traffic and sends it to the destination server, which is just an ordinary server reachable from the Internet. The server is not aware of the use of Tor. The Tor network protects the privacy of the client by hiding the client identity from the server.

If a server wants to hide its location, the Tor Onion Service protocol can be used. In this protocol, an onion proxy on the server side will register an onion service in the Tor network. It then gets a special type of address, called an onion address, which is valid inside the Tor network. Clients can reach the server via Tor by using the onion address. The protocol thus makes it possible for a client to reach a server without letting the client know the server's real location.

As a side effect of hiding the server's location, the server becomes available without a public IP-address. We also note that the onion proxy that registers the onion service, initiates a connection from the server side to the Tor overlay network. This means that the server can be behind a NAT, since NAT only prevents connections from the Internet to the server, not in the opposite direction (Comer, 2014). An unintended consequence of the protocol is therefore that onion services can be used to reach servers behind NAT. This is useful for Swirlwave, and the reason for building the Swirlwave communication on Tor.

The use of the Tor Onion Service protocol makes it possible to reach smartphones outside local area networks. However, Tor does not include any means to announce new addresses to clients, so clients are not able to connect anymore when a smartphone changes location. Also, there is no protocol transparency, so a client connecting to the smartphone server app must understand the protocol used by Tor. The Swirlwave system is designed to solve these problems transparently as a middleware.

4 SWIRLWAVE

Swirlwave builds on the Tor Onion Service protocol, not because of the anonymity provided by the protocol, but because of the onion addresses assigned to participating devices. Thus, Swirlwave uses onion addresses to reach devices that are lacking publicly visible IP addresses. However, since the Tor Onion Service protocol is not designed for mobile devices, but rather devices that never change location, Swirlwave adds functionality for locating devices and keeping peers up to date with correct addresses. As anonymity is not required in Swirlwave, as opposed to in Tor, authentication of peers is also added to Swirlwave.

4.1 Contacts

Keeping track of peer addresses is a central feature of Swirlwave. This is achieved without external directory services or single points of failure. Each peer in Swirlwave keeps its own, locally stored, contact list of the known peers. New contacts are added out-of-band, for example through near-field communication (NFC) when friends meet face-to-face.

An entry in the contact list contains data that is needed to communicate with that specific peer. It also contains information about services offered by the peer. An entry includes the peer ID, onion address, services offered by the peer, phone number (used as an alternative address in an SMS fallback protocol) and its public-key. See Table 1 for the complete contact list information.

To conduct meaningful communication, client and server must use the same protocol. Swirlwave allows applications to use whichever protocol that is suitable. This flexibility is possible by representing protocols as universally unique identifiers (UUID) (Leach et al., 2005). More generally, they are identifiers of contracts or agreements that server and client must comply to in order to properly communicate. Swirlwave does not care about the details of this contract, but simply uses the identifier to match clients and servers.

For example, to send a message to a friend, a user selects the friend from the Swirlwave contact list. Based on protocol UUIDs registered for this friend, Swirlwave presents a list of available communication types. If the user has an application that can be used as client, Swirlwave detects it by matching the identifiers of the locally installed applications with the identifier of the friend's messaging service.

4.2 Authentication and Confidentiality

Each peer is equipped with its own key-pair for public-key encryption (Goodrich and Tamassia, 2014). This is used for authentication purposes. It is also used for ensuring confidentiality, integrity, and non-repudiation of data when communicating over other channels. Tor Onion Service communication is end-to-end encrypted, which provides communication confidentiality.

The Tor protocol is designed for anonymity. Consequently, the onion proxy on the server side will not know the origin of incoming connections. In our ap-

Table 1: Information in a contact list record.

Field name	*Description*
Name	A human-readable name of the friend
Peer ID	An ID that is unique across all installations
Address	The friend's onion-address
Address Version	Each time a peer changes its address, it will increment the address version number.
Secondary Address	The phone number used when sending SMS-messages to the peer
Public-key	The public-key from the friend's asymmetric keys
Online Status	If the last attempt to reach the friend was unsuccessful, this status is set to offline, otherwise online
Last Contact Time	The last time contact was made with the peer
Known Friends	A list of peer IDs for mutual friends
Capabilities	A list of capabilities supported by this friend. Such as available services and protocol UUIDs
Awaiting Answer	A flag indicating if an answer from the SMS fallback protocol is pending

proach, this anonymity hinders the identification of incoming requests from friends.

Swirlwave solves this by providing an authentication mechanism for validating the identity of incoming connections. This functionality is part of the Swirlwave client and server proxies and is based on public-key cryptography.

To establish a new connection, the client-side proxy sends a system message encrypted with the client's private key. To validate the identity of the sender, the server-side proxy decrypts the message with the client's public key. The connection request is refused if the claimed identity of the client cannot be authenticated.

4.3 Establishing Connections

To establish a connection, a request is sent from a Swirlwave client proxy through the onion proxy. The message header contains (among others) the friend's onion-address. If the onion proxy returns a positive response code (0x5A) telling that it successfully connected to the remote onion service, the client proxy also receives a four-byte number. This number is later returned to the server as part of the connection message. If the server proxy accepts the connection (after evaluating the connection message from the client) it responds with a success code (0x10). The client proxy then starts reading and writing bytes between the incoming socket from the application-layer client and the outbound onion proxy socket. Figure 3 illustrates this communication.

In the connection message, everything except the client ID is encrypted with the client's private-key. The server proxy looks up the peer ID in the contact list and rejects the connection if the peer is unknown. If the ID is found in the contact list, the registered public-key is used to decrypt the message. If the message is successfully decrypted, and the returned number equals the one that was sent to the client, the client is authenticated and the connection is accepted.

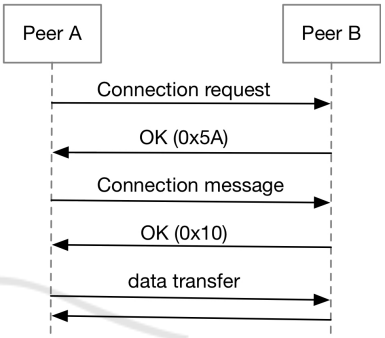

Figure 3: Establishing connection.

The connection message also specifies if the connection is for transmitting system messages or application-layer data. For an application-layer connection, the server proxy will use the identifier in the destination field, to match a local service endpoint, and set up a connection from the client to the service. For a system message, the content of the message field will be dispatched to an internal module in the Swirlwave middleware that handles system messages. See Table 2 for a complete list of information included in the connection message.

If the onion proxy fails to connect to the remote onion service, the client proxy marks the peer as being offline. It then starts the process of obtaining an updated address to the peer, either by asking a mutual friend, or using an SMS fallback protocol that contacts the peer directly. The client proxy will not try to establish new connections to the friend until an updated address is obtained. The friend will then be marked as online again.

4.4 Address Changes

When a smartphone moves from one network to another, for instance from Wi-Fi to cellular data, its access point is not the same as before. The IP address will most likely be different, and the route to the device will most certainly be different.

Table 2: Connection message information.

Field name	*Description*
Sender ID	The peer ID of the client
Random Number	A random number initially generated and sent by the server proxy
Message Type	Whether this is a system message or an application-layer connection
Destination	An identifier of a capability representing a service that the client wishes to consume. This will only be set for application-layer connections.
System Message	A system message that will be dispatched to a module that handles system messages. This will only be set for system message types.

Address changes must be announced to friends. A device that has been offline, or has changed its location, will contact its friends as soon as it is online again. The new address is passed with a version number. This version number is increased every time a peer changes its address, and is used to determine the newest address when comparing registered addresses across peers.

If the peer has been offline for a while, it is not unlikely that some of its friends have changed addresses. The peer will not have received the updated address, and will not be able to reach them. It must then either contact a mutual friend to obtain a peer's new address, or contact the peer directly using an SMS fallback protocol.

As an example, assume peer B, in Figure 4, has changed address. It sends a system message (marked 1) with the new address to its three friends. Peer A successfully receives the address and updates its contact list, while the two other peers cannot be reached. When C and D later tries to contact B, they discover that it cannot be reached, and they will request an updated address. Peer A is a mutual friend of D and B, and D can therefore ask A for B's address (marked 2a). Peer C, on the other hand, does not have any other friends to ask. Instead, it uses an alternative channel to ask B directly for its address (marked 2b). As phone numbers represent stable addresses where peers can always be reached, Swirlwave uses SMS as the alternative channel.

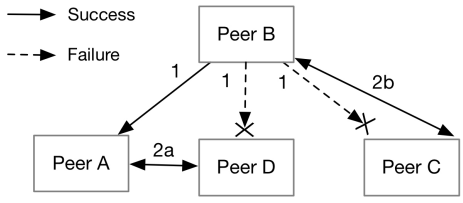

Figure 4: Peer B changes address.

Swirlwave uses onion-addresses that make it possible to connect to peers, even if their IP addresses are unreachable from the Internet. Nevertheless, routing is dependent on IP addresses under the hood, just as everything else on the Internet. It is possible to reuse an onion-address so that it resolves to a new access point. However, there is no support in the Tor control protocol for letting the client refresh the route to an onion service.

The Swirlwave solution is to monitor network changes on the device and register a new onion service when the smartphone connects to a new network. If a network and access point address is recognized from earlier, the onion service and onion-address from last time is reused.

If a connection is broken while being used by an application, Swirlwave does not try to reconnect automatically. In the current implementation of Swirlwave, it is the responsibility of the application to reconnect and resume transfers. To support continuous communication when devices change network, the Swirlwave proxies can be improved so that the connection is kept open between application-layer and Swirlwave proxy-layer. This will allow peers to continue communication when the new address is available.

4.5 SMS Fallback Protocol

The SMS fallback protocol is used to request new addresses from unresponsive peers. When the client proxy of peer *C* discovers that it cannot connect to a friend, *B*, and there are no other friends to ask for the address, it will send a data SMS to *B*. In contrast to an ordinary text SMS, a data SMS will not be visible to the user. Instead it will be received directly by Swirlwave.

The fallback protocol starts by *C* sending *B* an SMS, including *C*'s address and a secret one-time code. This is encrypted with *C*'s private key. The one-time code has several purposes; it enables duplicate message detection, and it is a combined message-ID and anti-forgery token sent back to *C*.

On the receiving side, *B* looks up *C*'s phone number in the contact list, to confirm that the SMS is from a friend. *B* decrypts the message and updates the contact list with *C*'s address. If Swirlwave is running and connected, *C* is answered immediately over the Internet with *B*'s current address. Otherwise *C* will receive the answer as soon as Swirlwave connects again.

In the response, *B* sends its new address together with the one-time code originally send by *C*. The message is encrypted with *B*'s private key. When received, *C* updates the contact list with *B*'s new address.

5 EXPERIMENTS

We have conducted several experiments using the Swirlwave middleware. Testing included both functionality of the system and the ability to support location transparent communication, and performance measurements, for startup time, connection establishment time, throughput, transmission time and latency. This section describes our results.

5.1 Peer-to-Peer Communication

We have tested peer-to-peer communication using Swirlwave for establishing connections to the network, connecting to peers and transferring data. In these experiments, we have in particular tested how the system handles network changes and the ability to stay connected despite location changes.

In one of the experiments, we use two smartphones, directly connected using Swirlwave. A web-cam app was installed on one phone, and the Swirlwave connection enabled the user of the other phone to receive live streaming from the web-cam in the phone's browser. Because of Swirlwave, this is possible while both phones are connected to 4G. We also demonstrated that the phones can change network between 4G and Wi-Fi during streaming. In that case, the phones will update each other's addresses and the streaming from the web-cam to the browser continues.

The address used by the browser, is a port on localhost. Thus, from the browser's point of view, the web-cam seems to be on the same phone. However, Swirlwave keeps the current address to the peer and routes the traffic to the correct smartphone. This means the browser can continue to use the localhost-address and can be kept unaware of network changes. This demonstrates location transparency and that we can stream from anywhere, without being connected to Wi-Fi.

5.2 Performance

When evaluating the performance of Swirlwave, we compared with two alternative configurations; one where Tor is used without Swirlwave, and another using a plain Internet connection.

We used two smartphones during the experiments; a Huawei P9 Lite, used as client and connected to cellular data (4G), and a Samsung Galaxy Note 4, used as server and connected to Wi-Fi. Experiments that collected measurements that are compared, were carried out within a short time-frame.

Orbot, which is the official version of the Tor onion routing service on Android, was used as to test communication with Tor without Swirlwave. Orbot enables smartphones to be reached outside local area networks, but has no mechanism for changing addresses when a smartphone changes location.

To enable direct Internet connection between smartphones, an Internet subscription with a static, public IP address was used in the experiments. For this environment, a wireless router was manually configured to forward from a specific port to one of the smartphones. This means that the server smartphone could be contacted directly from the Internet. The limitation of this approach, compared to Swirlwave, is that the server smartphone cannot be reached if it leaves the manually configured Wi-Fi. Also, the smartphone cannot have the role as server when connected to cellular data.

5.2.1 Starting Onion Proxy

This experiment measures how long it takes from the onion proxy is started to the onion service is registered and ready for use. The difference between starting a new onion service and reusing an existing one is compared.

Table 3: Onion proxy start-up times.

	Median	90^{th} *Percentile*	*Num. Trials*
New onion service	18.080s	43.941s	10
Reused onion service	8.401s	8.855s	10

The implementation of Swirlwave reuses an already registered onion service when reconnecting to a previously seen network location. As seen in Table 3, there is a clear difference in onion proxy start-up times between registering a new onion service and reconnecting to one that is already registered (and the registration process is avoided). Registering a new onion service took about twice as long. Also, the start-up time varied much more when registering new onion services than for reusing. When comparing the 90th percentiles, the start-up time for registering a new onion service was approximately five times slower than reusing.

5.2.2 Establishing Connections

We here measure how long it takes to establish a connection between client and server in the three different cases:

- Connecting via Swirlwave. This includes the time it takes to authenticate the client.
- Both client and server use Orbot.
- Connecting directly over the Internet.

Table 4: Times for establishing connection.

	Median	95^{th} *Percentile*	*Num. Trials*
Connecting via Swirlwave	1.829s	3.557s	100
Onion service w/Orbot	1.384s	2.931s	100
Directly over Internet	1.827s	3.597s	100

According to our experiments, the time it took to establish connections is nearly identical for connecting via Swirlwave and directly via Internet. This is a bit surprising, since the connections in case of Swirlwave must be made through the Tor overlay network. Additionally, the Swirlwave connection times include authenticating the client. Connecting via Tor using Orbot, which does not include any authentication, is faster than connecting directly via a plain Internet connection. This may suggest that the difference between connecting via Tor and via the Internet roughly equals the time Swirlwave uses to authenticate the client.

5.2.3 Throughput

Throughput is the rate of successful data delivery over a communication channel (Forouzan, 2013). Given that a connection is established and the client authenticated, we measure how long it takes from the client starts reading the first byte until 12.5MB has been read. The rate is subsequently calculated.

Table 5: Throughput.

	Median	95^{th} *Percentile*	*Num. Trials*
Swirlwave	2.510Mbps	1.380Mbps	74
Onion service w/Orbot	1.950Mbps	0.910Mbps	100
Directly over Internet	18.58Mbps	11.95Mbps	100

The throughput was lower when routing via Tor than directly over the Internet. This was true for both Swirlwave and Orbot. The throughput for Swirlwave was in this case higher than Orbot. Transmitting directly over the Internet without Tor was 7.4 times faster than Swirlwave, and 9.5 times faster than Orbot.

5.2.4 Transmission Time

Transmission time is the time it takes from the first bit till the last bit of a message is sent from a node. Transmission time is depending on message size and bandwidth (Forouzan, 2013), as shown in (1).

$$\textit{Transmission time} = \textit{MessageSize / Bandwidth} \quad (1)$$

To estimate transmission time for our system, the median throughput is used in place of the bandwidth, and the message size is set to 8 bits.

Table 6: Transmission times.

	Transmission Time 1 Byte (8 bits)
Swirlwave	3.200×10^{-6}s (3.200μs)
Onion Service w/Orbot	4.103×10^{-6}s (4.103μs)
Directly over Internet	4.306×10^{-7}s (0.4306μs)

The results show that transmission time for all three alternatives are very low, with the Internet-connection having the lowest result, followed by Swirlwave and Orbot.

5.2.5 Latency

Network latency specifies how long it takes for a bit of data to travel across the network from one node to another (Forouzan, 2013). Latency depends on several components, as shown in (2).

$$\textit{Latency} = \textit{Propagation Time} + \textit{Transmission Time} + \textit{Queuing Time} + \textit{Processing Time} \quad (2)$$

We first measure round-trip time (RTT), which is the time it takes from the client sends a byte until it receives a response byte from the server. This has the advantage that start and end times can be measured at the same smartphone. RTT is described in (3)

$$\textit{RTT} = 2 \times \textit{Latency} + \textit{Processing Delay} \quad (3)$$

The extra processing delay represents the time from the byte is read by the server until it sends a response byte to the client.

We estimate latency based on the RTT measures, using the simplified calculation in (4).

$$\textit{Latency} = \textit{RTT} / 2 \quad (4)$$

Latencies were almost similar for Swirlwave and Orbot, at about three tenths of a second, while latency

Table 7: Round-Trip Times.

	Median	95^{th} *Percentile*	*Num. Trials*
Swirlwave	0.637s	0.816s	100
Onion service w/Orbot	0.639s	1.554s	100
Directly over Internet	0.106s	1.039s	100

Table 8: Latency.

	RTT median	*Latency*
Swirlwave	0.637s	0.3185s
Onion service w/Orbot	0.639s	0.3195s
Directly over Internet	0.106s	0.053s

when transmitting directly over the Internet were approximately six times less.

From Table 6 we have that transmission time is very low, and thus negligible when considering latency. Latency in Table 8 therefore depend on propagation time, and the time used for processing and queuing in the nodes.

6 DISCUSSIONS

To establish connections from the Internet to hosts behind NAT, a technique, known as NAT Traversal (Hu, 2005), is needed for circumventing the problems associated with address translations and private IP addresses.

Before choosing Tor as a basis for Swirlwave, we considered other approaches to NAT traversal, including Virtual Private Network (VPN) (Comer, 2014), UDP hole punching (Hu, 2005) and SSH[9]. These alternatives had several drawbacks that made them unsuitable for Swirlwave. Setting up VPN servers requires public IPs and an amount of manual work for configuration and management. Also, for clients to act as servents, they need reserved IP addresses or some other mechanism for locating peers. UDP hole punching only supports UDP communication, and needs a server middlebox to establish peer-to-peer communications, while SSH requires a server with a public IP address.

Another choice we made, was using SMS as fallback protocol. Gossiping, hand-offs, and other techniques (Tanenbaum and Steen, 2014) were considered, but they were all regarded more complicated, less secure, less reliable, and they require the involvement of more than two participants. We consider the SMS fallback protocol to have several advantages. SMS is available on all smartphones, the phone number is a stable address, it works when only two peers exist, and the protocol does not require extra hardware, servers or software.

The functionality and performance of Swirlwave has been tested in several experiments. We find that Swirlwave handles peer-to-peer communication between smartphones well. Phones can act as both client and servers, and Swirlwave enables continued communication also when phones move between networks.

From the experiments, the most prominent downside with Swirlwave communication is the lower throughput caused by the use of Tor. The extra round trips and processing involved in authenticating the client does not seem to affect the performance much. Neither does the processing done by the Swirlwave proxies. The experiments also show that establishing a connection between peers takes time. It will therefore be beneficial to keep connections open, instead of closing when a session is finished.

The use of *IP* here, refers to IPv4 (Postel et al., 1981), which is by far the most widespread IP version as of today. However, with the use of the more recent IPv6 (Deering, 1998), each device will be given an address that is public, and NAT will no longer be needed. An addition to this protocol, called mobile IPv6, is designed to let devices keep their address even when changing networks (Perkins et al., 2011). In this scenario, Swirlwave would not need Tor for connectivity, but would rather build on IPv6. However, the adoption of IPv6 is still low in most countries. Per February 2018, it is estimated to about 12.8% in the U.K., 16.8% in Norway and 40.4% in the U.S.[10] Also, even with a full adoption of IPv6, NAT may still be used for security reasons, since it shields devices from direct access from the Internet. We therefore believe that Swirlwave (or similar types of middleware) will continue to be useful in the future.

7 CONCLUSION

The goal of this work has been to design and implement an alternative to the traditional, cloud-centric approach to smartphone communication. We have described a novel approach to mobile friend-to-friend communication in wide area networks, that allows direct communication between devices that lack public

[9]https://www.ssh.com/ssh

[10]https://www.akamai.com/uk/en/about/our-thinking/state-of-the-internet-report/state-of-the-internet-ipv6-adoption-visualization.jsp

IP addresses, and often disconnect or change network locations. This functionality is implemented as a middleware, called Swirlwave, on which applications can be built.

The system is friend-to-friend based, and only devices that are explicitly registered as friends can communicate directly as peers. This makes it possible to define smaller overlay networks of trusted devices.

Swirlwave is based on the Tor Onion Service protocol, which provides onion addresses for participating devices. Swirlwave extends the Tor service by providing authentication of connection requests and location transparency by handling network changes and updates of addresses.

Testing shows that Swirlwave handles peer-to-peer communication between smartphones well. It provides an interface that makes it easy for various applications to access the communication service. Swirlwave also supports incorporation of new devices, which makes it easy to expand a network.

REFERENCES

Bricklin, D. (2000). Friend-to-friend networks. http://www.bricklin.com/f2f.htm.

Comer, D. (2014). *Internetworking with TCP/IP*. Pearson, 6th edition.

Deering, S. E. (1998). Internet protocol, version 6 (ipv6) specification.

Forouzan, B. (2013). *Data communications and networking*. McGraw-Hill, New York, 5th edition.

Goodrich, M. and Tamassia, R. (2014). *Introduction to computer security*. Pearson, 1st edition.

Hu, Z. (2005). Nat traversal techniques and peer-to-peer applications.

Leach, P. J., Mealling, M., and Salz, R. (2005). A universally unique identifier (uuid) urn namespace.

Lee, Y. (2012). Socks 4a: A simple extension to socks 4 protocol. https://www.openssh.com/txt/socks4a.protocol.

Myers, J. (2016). World economic forum: 4 charts that explain the decline of the pc. https://www.weforum.org/agenda/2016/04/4-charts-that-explain-the-decline-of-the-pc/.

Perkins, C., Johnson, D., and Arkko, J. (2011). Mobility support in ipv6. Technical report.

Popescu, B. C., Crispo, B., and Tanenbaum, A. S. (2006). Safe and private data sharing with turtle: Friends team-up and beat the system. In Christianson, B., Crispo, B., Malcolm, J. A., and Roe, M., editors, *Security Protocols*, pages 213–220, Berlin, Heidelberg. Springer Berlin Heidelberg.

Postel, J. et al. (1981). Rfc 791: Internet protocol.

Rogers, M. and Bhatti, S. (2007). How to disappear completely: A survey of private peer-to-peer networks. *RN*, 7(13):1.

Smith, A. (2015). U.s. smartphone use in 2015. http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/.

Tanenbaum, A. and Steen, M. (2014). *Distributed Systems, Principles and Paradigms*. Pearson, 2nd edition.