# Ubiquitous Overlay

## Universal communication using imperfect hardware

Matouš Skála

TU Delft

# Ubiquitous Overlay

## Universal communication using imperfect hardware

by

Matouš Skála

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday January 1, 2013 at 10:00 AM.

| | | |
|---|---|---|
| Student number: | 4893964 | |
| Project duration: | November 15, 2019 – June 30, 2020 | |
| Thesis committee: | Dr.ir. J.A. Pouwelse, | TU Delft, supervisor |
| | ?, | TU Delft |
| | ?, | TU Delft |

An electronic version of this thesis is available at
http://repository.tudelft.nl/.

TU Delft

# Preface

Preface…

*Matouš Skála*
*Delft, April 2020*

# Contents

# 1 Introduction

The Internet was created with the idea that any two computers connected to the shared network should be able to communicate with each other. It has also been built on the principles of decentralization, without any central entity having power to take the network down. 50 years later, we live in the world where most of the services are centralized and user data are stored on the servers owned by a few large profit-oriented companies.

In the recent years, the idea of decentralization has attracted many in the engineering and research community. Since the introduction of cryptocurrencies in the last decade, there have been many discussions on whether we can decentralize other services, such as social media, or web. With the trend of decentralization, applications are shifting from the client-server model to peer-to-peer, which brings many challenges and calls for a new networking stack.

This thesis proposes and implements a protocol for peer to peer communication between any two devices. It is implemented as a Kotlin library which can be used on desktop, smartphones, tablets, and IoT devices. It can be used to deploy a truly ubiquitous network overlay which is available anytime and everywhere. The protocol allows any two devices to establish a direct connection by taking advantage of NAT traversal techniques to connect peers behind different types of NATs. When the Internet connection is not available and peers are located in proximity, the connection can be established using Bluetooth Low Energy. Peers are addressed by their public keys and their physical addresses at lower layers are abstracted away.

The robustness of the NAT traversal mechanism has been tested by conducting a connectivity check between devices using the networks of major mobile network operators and home broadband providers in the Netherlands. The mechanism has been shown to establish connection in all tested network conditions.

To demonstrate the usage of the library, a decentralized messaging application with end-to-end encryption, which allows trustworthy communication insusceptible to the man-in-the-middle attack, has been implemented. To get feedback on the APIs and general usability of the library, 4 teams of MSc students have been asked to develop non-trivial distributed applications on top of it.

Compared to the state of the art solutions, the proposed library combines both nearby and Internet connectivity, does not require any central server, works on a variety of devices under challenging network conditions, and is completely open source.

# 2

# Problem Description

The architecture of the Internet has emerged in an evolutionary process rather than from a carefully designed grand plan. There is no central entity in charge of architectural decisions, or anyone with ability to turn it off. The Internet standards are developed by the working groups of Internet Engineering Task Force (IETF), a non-profit open standards organization composed of volunteers. Its evolution is based on a rough consensus about technical proposals, and on running code. While there are many conflicting opinions on its architecture, the general consensus is that the main goal of the Internet is to provide global connectivity by the *Internet Protocol (IP)*. [2] This ultimate goal requires cooperation of multiple parties, including researchers, developers, and commercial service providers.

## 2.1. End-to-End Principle Challenged

Most decisions related to protocol and system design have for a long time followed the *end-to-end principle*, which states: "The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible." [9]

It implies that the job of the network should be merely to deliver datagrams as efficiently as possible, and the rest of the responsibilities should be pushed to the end points of the communication system. As every application has different requirements, providing additional features on the Internet layer would turn some applications not using those features less efficient. It is also cheaper to upgrade the end points to add new capabilities rather than replace the network infrastructure. A common example for supporting the end-to-end principle are delivery guarantees. In the ARPANET, the receipient would send *Request For Next Message* packet every time a packet was delivered. However, some applications would still need to implement its own acknowledgements on the application level, to indicate that the messages were processed correctly. Similar examples can be found for supporting authentication, encryption, and other guarantees in communication systems.

The end-to-end principle has been challenged by introduction of *middleboxes*, intermediary nodes in the network that perform functions different from functions of a regular IP router. In [3], an extensive catalogue of 22 different middlebox classes

has been analyzed. Network address translators (NATs), packet classifiers, IP firewalls, application layer gateways, or proxies are just a few examples of components hidden in the network. While most of them try to be transparent, the behavior of certain middleboxes can severly impact the end-to-end performance, either by delaying packet retransmission, or aborting user sessions entirely in case they crash. Some of them perform advanced logic on different protocol layers and keep a hard state, which violates the end-to-end principle stating that the network should be kept as simple as possible.

## 2.2. Overcoming Address Exhaustion

In the Internet Protocol, each computer gets assigned an address which is subsequently used for packet routing. The most common version of the protocol, IPv4, uses a 32-byte address space which is not enough to uniquely identify all devices on the planet. To deal with IPv4 address exhaustion, internet providers were forced to deploy different types of NATs which allows a single address to be shared across multiple devices.

Traditionally, each consumer would have a NAT implemented in the router located at the edge of the network. We will refer to this case as a customer premises equipment NAT (CPE NAT). It would serve as a gateway between the local area network (LAN) and the wide area network (WAN), the Internet. This type of NAT usually maps all addresses inside LAN to a single external WAN address. It also provides additional features such as firewall, which only allows incoming traffic on ports that are explicitly open, or on whose the client initiated the communication first.

However, to further conserve the address space, and facilitate transitions to IPv6 with backwards compatibility of IPv4, it has become a trend among internet service providers to implement a NAT in their infrastructure. This topology is called a *carrier-grade NAT* (CGN). This is especially used in mobile networks, where there is no place to implement NAT on customer premises. This allows all subscribers connected to the same gateway to share a pool of private network addresses which are then translated by NAT to an external address. This scenario is also called *NAT444*, as the address is translated twice along the route and the packets pass through 3 different addressing domains: the customer's private network, the carrier's provate network, and the public Internet. Another possible topology is *Dual-Stack Lite* (DS-Lite) which can be used as a transition mechanism from IPv4 to IPv6. In DS-Lite, the carrier's network uses IPv6 addresses which get translated to IPv4 for the public Internet.

Carrier-grade NAT is by definition managed by the network operator and the customer does not have any control over it. CGN usually implement a complex functionality to ensure reliability for thousands subscribers and compliance with legal regulations, which again violates the end-to-end principle. Inability to manage open ports breaks the communication model of many peer-to-peer applications which usually have to reside to using a proxy server for relaying communication. To mitigate this issue, *Port Control Protocol (PCP)* [4] has been designed and recommended by IEFT to enable port forwarding in CGN deployments. However, our experiments have shown that most providers do not have this mechanism deployed at the moment and that some providers have deployed a version of CGN that does not satisfy NAT behavioral requirements [5], which makes P2P communication nearly impossible.

## 2.3. Freedom of Trustworthy Communication

The principle of network neutrality states that ISPs must treat all traffic equally and not discriminate based on its content, addresses of recipients, or methods of communication. However, we can see that this principle is violated in many cases, some of them more worrying than others. There are several countries that enforce Internet censorship either by simple IP address blocking, or advanced deep packet inspection, preventing people from communicating freely.

Even in places with relatively mature infrastructure, there are occasionally connectivity issues at places with high gatherings of people, such as conferences or festivals, causing congestion in the infrastructure networks. The increasing capabilities of smartphones and novel wireless networking technologies open up possibilities to a whole new range of applications that can communicate without the need for the Internet connection. The ideal communication protocol should be able to use those when possible. It has been seen how the Bluetooth technology can be useful in creating a mesh network during the Hong Kong protests in 2018 and 2019, where protesters relied on peer-to-peer communication apps as a communication tactic.

Another problem related to communication is ensuring message authenticity and privacy. Most of the commonly used protocols have adopted *Transport Layer Security (TLS)* protocol to secure all communication between clients and servers. The trust in the system is enforced by using *public key infrastructure (PKI)* with trusted certificate authorities. However, public key infrastructure is not easy to adopt to a peer-to-peer system with self-sovereign identities.

## 2.4. Re-decentralization of Internet Infrastructure

While the Internet is built on the principles of decentralization from the ground up, we cannot say so about the Web and other services built on top of it. The majority of the websites is operated by a few hosting companies and most of the static content is served by a few *Content Delivery Networks (CDNs)*. For example, *Cloudflare* currently powers 10 % of all internet traffic[1]. Since there is a central point of failure, if a provider experiences outage, a large part of the internet goes down.

The inventor of the World Wide Web has proposed a Solid [8] as an architectural pattern for building the next generation of decentralized social applications. It is based on the principle that users own their data and have freedom of choice where to store them, which significantly improves privacy and data reusability. As the application logic is decoupled from the data storage servers, multiple applications can access them same data if allowed by the user.

[1] https://blog.cloudflare.com/cloudflare-traffic/

## 2.5. Research Question

While it is out of scope of this thesis to provide a complete solution for the next generation of web applications, the afore-mentioned problems and challenges lead us to the following question:

- **Can we devise a protocol facilitating trustworthy device-to-device communication between smartphones under challenging network conditions without using a central server?**

We analyze the question in more detail:

- The communication should be **trustworthy**, which means we can verify that we received the message from the person that claims to be its sender. The communication can be optionally encrypted to provide privacy on top of authenticity.
- The communication should be facilitated **device-to-device** without need for any additional **server** infrastructure, with the exception of the initial bootstrap server. The bootstrap server should not be needed if there are peers in proximity which can be used for bootstrapping.
- The protocol should be able to work under **challenging network conditions**, which means it should be robust against the presence of various middleboxes. Mainly, it should be able to establish connection in presence of carrier-grade NATs deployed by mobile network operators.
- The communication protocol should in theory work on both most popular **smartphone** operating systems and support their most commonly used versions. However, supporting only the most commonly used OS would be sufficient for the prototype.

# 3

# State of the Art

## 3.1. Network Address Translation

### 3.1.1. NAT Classification

### 3.1.2. Hairpinning

### 3.1.3. Carrier Grade NAT

## 3.2. NAT Traversal

### 3.2.1. Port Forwarding

### 3.2.2. Session Traversal Utilities for NAT (STUN)

### 3.2.3. NAT Behavior Discovery Using STUN

### 3.2.4. Traversal Using Relays Around NAT (TURN)

### 3.2.5. Interactive Connectivity Establishment (ICE)

### 3.2.6. Symmetric NAT Traversal

## 3.3. Nearby Communication

Modern smartphone devices come equipped with several wireless communication standards that can potentially be used for communication with other nearby devices. It is desirable to use such a technology when multiple devices in proximity want to communicate with each other when there is no reliable Internet infrastructure available. These technologies can be also preferred over the Internet in case of censorship and privacy concerns, as has been shown during numerious occasions such as Hong Kong protests. From the user experience perspective, it is desired that the device discovery and connection establishment does not require any user interaction besides the one required by the application use case. However, this is often difficult to achieve in the security model of smartphone operating systems, which try to protect users by enforcing a system UI for any sensitive operations, as shown in Figure 3.1.

### 3.3.1. Bluetooth

The oldest and the most battle-tested technology for nearby connectivity is Bluetooth, which has been in development for more then 20 years. The common flow for Bluetooth usage is to first force the user to *pair* (or *bond*) two devices. The device

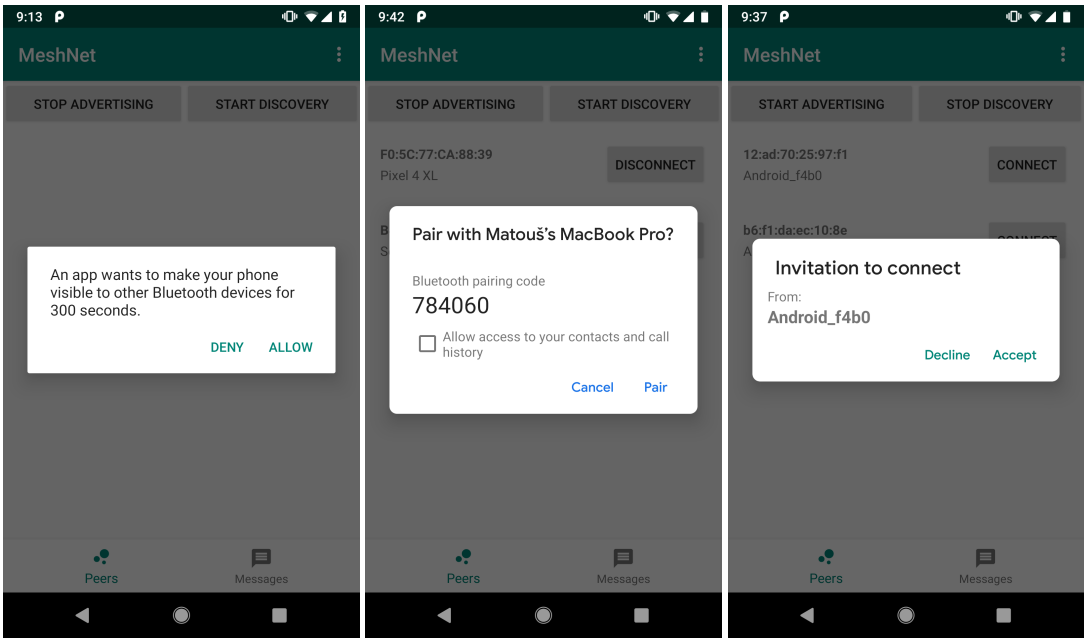


Figure 3.1: The system UI for enabling Bluetooth discovery, accepting an incoming Bluetooth pairing request, and accepting an incoming Wi-Fi Direct connection on Android 9

A first needs to manually be set as *discoverable*, usually for a limited time period. The device B then performs a scan to discover nearby devices. The device B can then send a pairing request to the selected device. Then a pairing code is displayed and once both users accept the pairing request, the devices are paired.

Only after that, a secure *Radio frequency communication (RFCOMM)* channel can be established. The pairing process requires user interaction, which degrades the user experience in certain applications when authentication is performed on the application level. While it is possible to establish an *insecure* RFCOMM channel if the MAC address of the other device is known, the user of the other device still needs to manually set it to be discoverable.

### 3.3.2. Bluetooth Low Energy

*Bluetooth Low Energy (BLE)* [12] was introduced in 2010 as part of the Bluetooth 4.0 specification. It is a completely different communication protocol incompatible with the classic Bluetooth. BLE offers considerably decreased power consumption with a similar communication range and slightly lower bandwidth. It was originally intended to support an infrequent low-power communication with wearables, healthcare accessories, or smart home appliances. However, while it is not a primary use case, it could also be potentially used for low-bandwidth peer-to-peer communication between smartphone devices.

It is notable that once an application is granted a Bluetooth permission, it can fully control BLE APIs without any user interaction, which opens up doors for a range of many different applications.

### 3.3.3. Wi-Fi Direct

There have been many attempts to enable direct communication between IEEE 802.11 radio devices. The 802.11 standard defines two operating modes. Next to the traditional *infrastructure* mode, there is an *ad-hoc* mode which allows device-to-device communication. However, the ad-hoc mode is not supported by Android OS, even though it can be enabled on some devices with a root access.

*Wi-Fi Direct* (also known as Wi-Fi Peer-to-Peer) [**?** ] is a IEEE 802.11 based protocol released by Wi-FI Alliance in 2009 and supported from Android 4.0. With Wi-Fi direct, devices are organized in groups, where one device is the Group Owner (GO) and the rest are Group Members (GM). The roles are not predefined, but are negotiated during the group formation process. Groups are able to support Legacy Clients (LC), which means that even devices without Wi-Fi Direct support can join as group members.

#### Multi-group communication

The standard only defines intra-group communication, but it does not restrict a Group Member from participating in multiple groups simultaneously. Moreover, a device can theoretically connect both to Wi-Fi Direct and an infrastructure network by using multiple virtual MAC entities. However, these functionalities are not defined by the standard and thus depend on the implementation. In Android, the GO always has the same hardcoded IP address (192.168.49.1), while GM are assigned a random address from the same subnet (192.168.49.2-254). As a result, the scenario where the device would act in multiple groups at the same time cannot be directly implemented.

There are several workarounds proposed in [**?** ]. The first solution is to use *time sharing* to allow a device to act as a gateway between multiple groups. It requires a device to periodically disconnect and reconnect to different groups and effectively act as a relay passing messages between multiple groups. Another solution takes advantage of *simultaneous connections* using Wi-Fi Direct and the traditional Wi-Fi. The GO advertises its group using a unique Service Set ID (SSID) that can be used by other clients not supporting the framework to join the group. Experiments have shown that it is possible to create a LC/GM gateway node when communicating using multicast UDP datagram sockets. Due to routing-related issues, this solution does not work with traditional datagram or stream sockets. The authors also propose a more efficient *hybrid* protocol that uses multicast sockets as a control channel that triggers gateway configuration change if needed. However, all solutions rely on undocumented behaviors, which means their reliability can vary across devices or Android OS versions.

### 3.3.4. Wi-Fi Aware

*Wi-Fi Aware*, also known as *Neighbor Awareness Networking* (NAN) is a recent networking standard introduced by Wi-Fi Alliance. [**?** ] It works by forming clusters with nearby devices. The discovery process starts when one device (a *publisher*) publishes a discoverable service. Other devices (*subscribers*) who subscribe to the same service will receive a notification once a matching publisher is discovered. After the subscriber discovers a publisher, it can either send a short message or establish a network connection with the device. A device can be both a subscriber and a publisher

| Technology | Android | iOS | Throughput | Range |
|---|---|---|---|---|
| Bluetooth | 2.0+ | 5.0+ | 2 Mbps | ~40 m |
| BLE Advertising | 4.3+ | 6.0+ | 0.3 Mbps | ~100 m |
| BLE GATT | 5.0+ | 6.0+ | | |
| BLE L2CAP | 10.0+ | 11.0+ | | |
| Wi-Fi Direct | 4.0+ | N/A | 250 Mbps | ~200 m |
| Wi-Fi Aware | 8.0+ | N/A | | |

Table 3.1: The comparison of properties of the most common wireless communication technologies and their support in smarphone operating systems

simultaneously.

### 3.3.5. Support in Operating Systems

## 3.4. P2P Communication Libraries

### 3.4.1. libp2p

### 3.4.2. Nearby Connections API

### 3.4.3. MultipeerConnectivity

### 3.4.4. Bridgefy SDK

### 3.4.5. Berty Protocol

### 3.4.6. IPv8

# 4

# Design

## 4.1. Goals and Considerations

### 4.1.1. Reliable vs. Unreliable Transport

### 4.1.2. UDP Socket Multiplexing

### 4.1.3. Authentication and Encryption

### 4.1.4. Peer Discovery

## 4.2. NAT Traversal with Peer Introductions

One of the traditional NAT traversal methods called *UDP hole punching* allows to establish UDP connectivity between two peers when both of them are hidden behind a NAT. It is based on the concept that both peers fire a UDP packet targeted at each other at the same time. This first packet creates a mapping entry in the sender's NAT and allows subsequent incoming packets to be delivered. This process can also be seen as opening a *hole* in the firewall, which explains the term *hole punching*.

IPv8 implements a decentralized variant of UDP hole punching mechanism integrated with the peer discovery process, which has been previously described in [6] and [14]. The complete peer discovery protocol consists of 4 messages and it is visualized in Figure 4.1.

After reviewing the py-ipv8 implementation, it has been discovered that it only works reliably when all nodes are behind different NATs. This is because of the fact that the tracker only stores WAN addresses which are then used for peer introductions, and peers always use their WAN address for communication. When multiple nodes are located behind the same NAT, they can still commnicate in case their NAT supports hairpinning, but this is not always the case. There is no mechanism to discover other peers on the same LAN without using a tracker, and there is no way to use LAN addresses for communication. We will fix this flaw by reusing some additional fields which are already present in the IPv8 packet format probably for legacy reasons, but are not currently used. Thanks to that, the updated protocol is still backwards compatible and can be used to connect to peers using older versions of the protocol. The updated protocol works as follows:

1. When Alice wants to connect to a new peer in a specific community, it can send an *introduction request* to a tracker, or any other peer present in the commu-

nity. The introduction request should contain the community ID representing the ID of the community in which Alice wants to find a new peer, and her LAN and WAN address.

2. We assume Alice sends an introduciton request to Bob. Bob selects a random peer (Charlie) from the requested community, and sends a *puncture request* to him. The puncture request contains the WAN address of Alice.

3. At the same time, Bob sends an *introduction response* back to Alice. The introducion response contains LAN and WAN address of Charlie.

4. As soon as Charlie receives a puncture request, he should send out a packet to the WAN address of Alice. That will create a mapping in his NAT, so Alice would be able to contact him.

5. Alice then keeps sending introduction requests to Charlie's WAN address for a specified interval. After Charlie sends out a puncture packet, the next introduction request should reach him. She can also try to contact him over LAN if they reside in the same subnet.

6. Charlie should respond to the introduction request with an introduction response, in which he include another peer for introduction. Upon receiving an introduction request or response, the receiver should add the sender to their verified peer list.

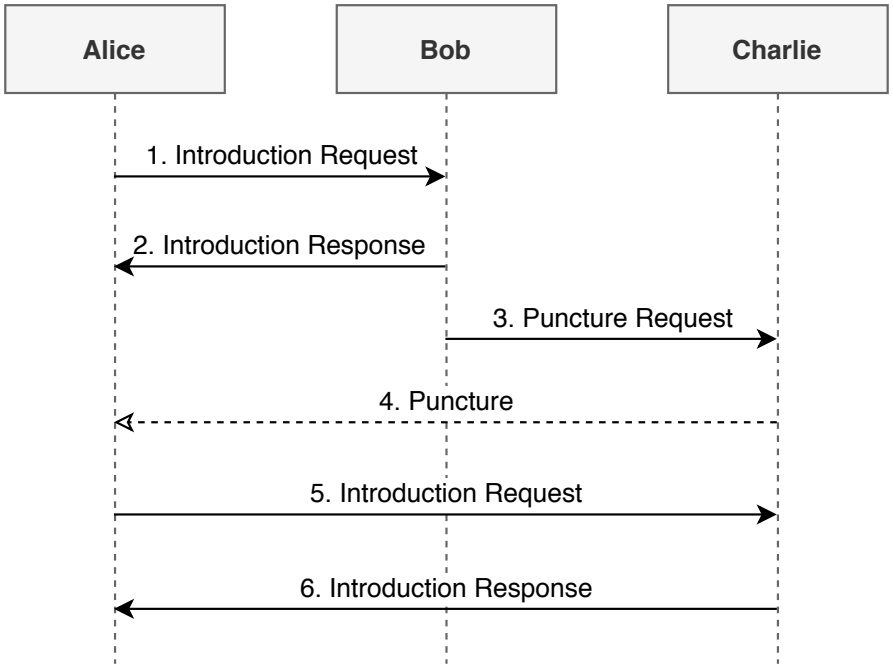


Figure 4.1: Peer discovery mechanism with NAT traversal

## 4.3. Symmetric NAT Traversal

Our method provided in the previous section allows us to traverse around NATs that perform endpoint-independent mapping. However, as previously discussed in Section 3.1.1, peers behind NAT with address-dependent mapping would only be able to connect to public peers that do not require NAT traversal. This poses a major threat to our goal of global connectivity.

### 4.3.1. Topological Assumptions

Inspired by the previously discussed symmetric NAT traversal methods, we have extended our peer introduction protocol to work with some additional NAT behaviors that we encountered during our experimental evaluation of CGN deployments. For the further discussions, we consider three different cases with respect to NAT topology between two different peers A and B:

**Case 1: One peer behind a symmetric NAT**

1. Peer A is behind a NAT with endpoint-independent mapping.

2. Peer B is behind a NAT with endpoint-independent IP address mapping, but an arbitrary port mapping behavior.

3. Peer B can estimate a session limit implemented by their NAT. By default, the algorithm first assumes there is no limit. In case the first traversal attempt fails, then it falls back to the default limit of 1.000 sessions with a 30-second timeout. This limit has been empirically shown to be successful in all scenarios considered in our experiments. However, the peer should have an option to tweak this parameter if they believe their NAT implements more permissive or restricted behavior.

**Case 2: Both peers behind symmetric NATs without a session limit**

1. Both peers are behind a NAT with endpoint-independent IP address mapping, but an arbitrary port mapping behavior.

2. There is no session limit implemented by any of the NATs.

**Case 3: Both peers behind symmetric NATs with a session limit**

1. Both peers are behind a NAT with endpoint-independent IP address mapping, but an arbitrary port mapping behavior.

2. There is a session limit implemented by one or both of the NATs.

In this case, we need to send packets at the frequency restricted by the session limit. However, as port mappings can frequently change on both sides due to a binding timeout, there is no guarantee when both peers will find find the port mapping. We can keep trying opening ports at random, hoping both peers will meet eventually. It is questionable whether it makes sense to support this case, as the effort of establishing the connection might not meet the benefits. It would make sense for long-lived connections that are kept alive for a long period. However, we expect mobile

connections to be mostly temporary, so this case has not been implemented while it is theoretically feasible.

Even in the extreme case when the NAT assigns ports completely at random, our NAT traversal mechanism should still work.

The problem is that when a peer A is connecting to peer B behind a symmetric NAT, the peer A does not have a way to find out the public port of peer B.

### 4.3.2. NAT Type Detection

We would like to have a fully automated NAT traversal. For that, we need a mechanism to decide whether to perform a simple UDP hole punching, or there is a need for an extended multiple UDP hole punching method. We try to detect the NAT type based on its behavioral properties. We keep a log of triples consisting of our LAN address, our WAN address, and peer address. When we detect our LAN address has been mapped to multiple WAN addresses as reported by different remote peers, we can conclude we are located behind a symmetric NAT. Otherwise, we report our NAT type as *unknown*, as it is not necessary to know the exact NAT type for the purposes of UDP hole punching.

### 4.3.3. Extended Peer Introduction Protocol

## 4.4. Relay Protocol with Bandwidth Accounting

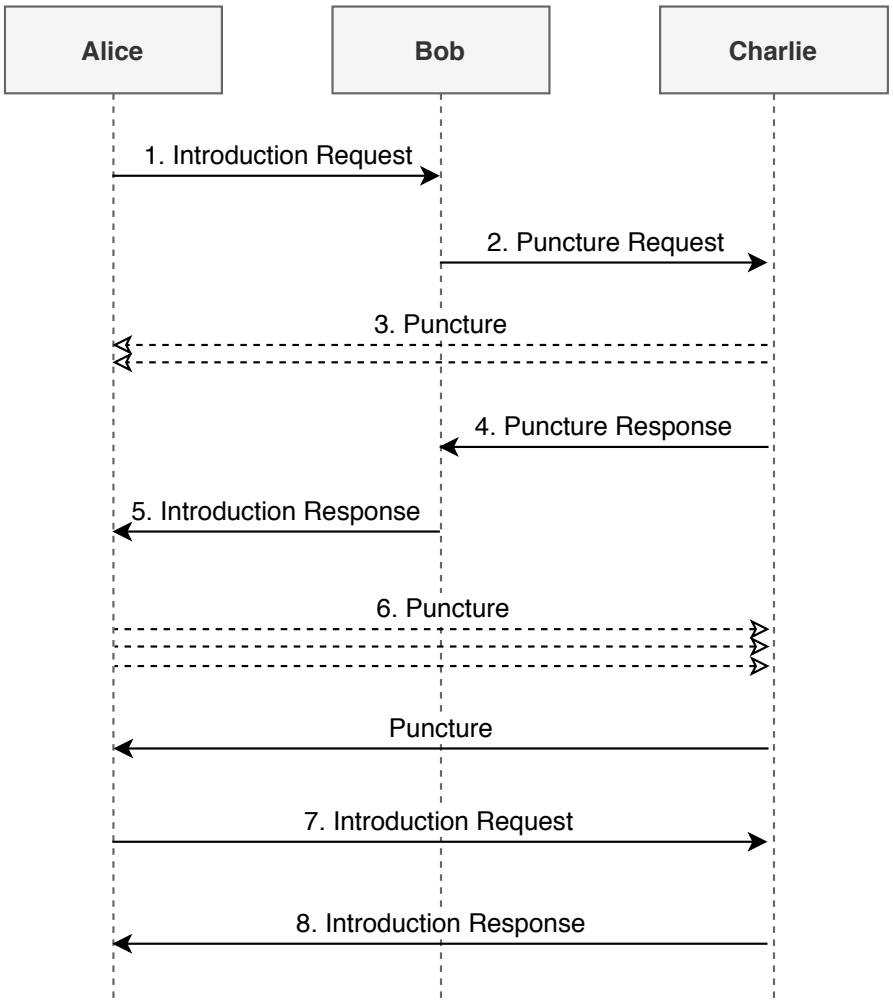


Figure 4.2: The flow diagram of the symmetric NAT traversal protocol

## 4.5. P2P Communication with Nearby Devices

### 4.5.1. Introduction to Bluetooth Low Energy

In this section, we introduce the most important Bluetooth Low Energy concepts defined in the Bluetooth specification [1]. This background knowledge is fundamental for understanding the subsequent sections. In principle, there are two methods that BLE devices can use for communication: using connectionless *broadcasting*, or by establishing *connections*.

#### Broadcasting

BLE advertising packets are used to broadcast data to multiple peers at the same time. Other devices can run a *scanning* procedure to read advertisement packets. Each advertisement packet can carry 31-byte payload to describe its capabilities and or any other custom information. Optionally, the scanning device can request a *scan response* from the advertiser, in which the advertiser can send an additional 31-byte payload. That means 62 bytes in total can be transmitted using the broadcasting mechanism. It is important to note that this is unidirectional data transfer and the broadcaster has no way to specify who can receive those packets, or receive any acknowledgements.

#### Connections

An advertising packet can be marked as *connectable*. In that case, if data have to be transferred bidirectionally or more than 62 bytes are required, a connection between two devices can be established.

Devices in BLE can act in two roles: *centrals* and *peripherals*. A central repatedly scans for advertising packets broadcasted by peripherals and when needed, it initiates a connection. A peripheral then periodically broadcasts advertising packets and accepts incoming connections. There are no restrictions on connection limits imposed by the specification. Since Bluetooth 4.1, a single device can act both as a central and peripheral at the same time, and it can also be connected to multiple centrals/peripherals.

#### Address Types

The BLE protocol stack differentiates between two types of addresses. The *public address* is a standard IEEE-assigned MAC address that uniquely identifies the hardware device. Since all BLE packets include a device address, it would be possible to track device movement by adversial scanners. BLE addresses this issue by using a *random address* for any communication. This address is generated using the combination of a device *identity resolving key* and a random number, and it can be changed often, even during the lifetime of a connection.

### 4.5.2. BLE Protocol Stack

The BLE standard is composed of several protocols which form the BLE stack visualized in Figure 4.3. The stack is split in two parts: a *controller* and a *host*. The controller is usually implemented in hardware, while the host is a part of the operating system. Both parts communicate over the *Host Controller Interface (HCI)*. Even though the

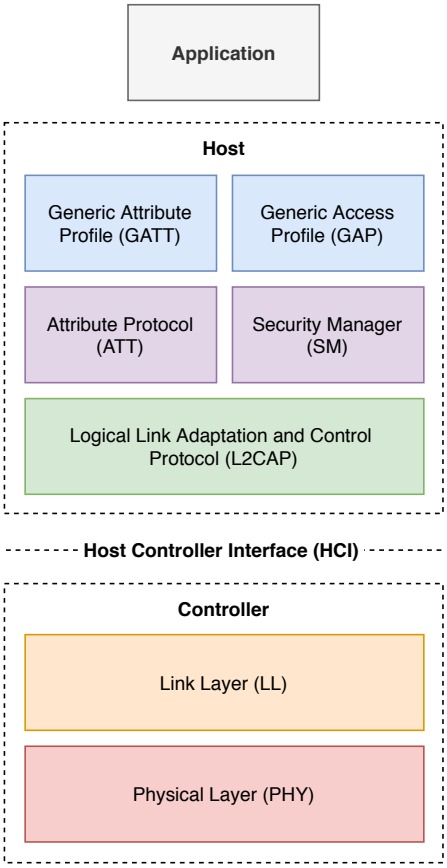


Figure 4.3: Bluetooth Low Energy Protocol Stack [12]

applications usually communicate only with the highest layers, it is worth to understand how all layers of the stack fit together. In this section, we describe the responsibilities and mechanics of each protocol.

**Logical Link Control and Adaptation Protocol (L2CAP)**
On the lowest layer in the host, L2CAP is responsible for multiplexing and splitting data from higher-level protocols (ATT and GAP) into 27-byte data packets. These are then forwarded to the Link Layer and transmitted over the Physical Layer implemented in the Bluetooth radio. Starting with Bluetooth 4.1, the applications can communicate over L2CAP directly. User-defines channels allow for higher-throughput data transfer without the additional complexity added by ATT.

**Generic Access Profile (GAP)**
// TODO

**Attribute Protocol (ATT)**
ATT is a client-server protocol for reading and writing attributes. It is strict about sequential operation. If a request is pending, no further requests should be sent until the response is received, otherwise they are discarded. Attributes are stored under attribute handles defined by UUIDs. Various operations are defined by the protocol. Apart from the standard configuraiton, read and write, queued write is supporte to write data longer than a single packet. If a client wants to be notified about handle

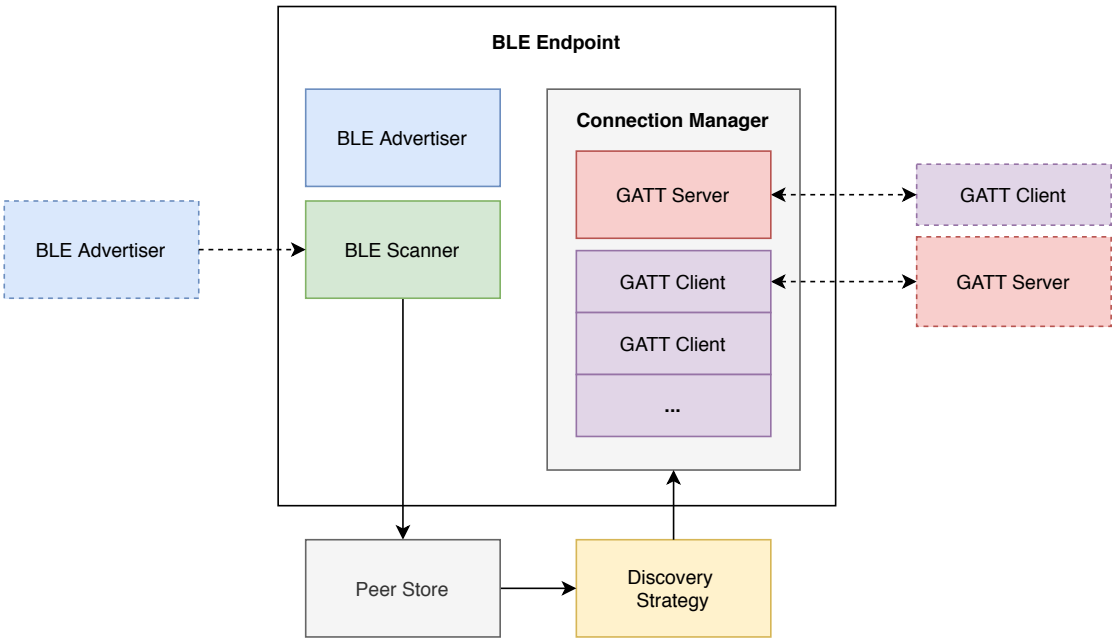


Figure 4.4: Bluetooth Low Energy Communication Architecture

changes, it can subscribe to handle value notifications or indications. Both allow the server to notify the clients whenever a value changes, but in case of indication, the client should respond with ack packet to confirm it has received the indication.

**Generic Attribute Profile (GATT)**
The Generic Attribute Profile defines how to exchange data between devices. It uses ATT as a transport protocol. The data are organized hierarchically in *services* which contain groups of related data called *characteristics*, which can be further specified using *descriptors*.

GATT is a client–server protocol and it follows the same principles as ATT. A client sends requests to the server and receives responses or server-initiated updates. The server is responsible for storing data written by the client and responding to read requests. It is important to mention that the client and server roles in GATT are independent of the roles defined by GAP. Both a peripheral and a central can take the role of a client or a server, or even both at the same time.

### 4.5.3. BLE Communication Architecture

As mentioned earlier, the primary purpose of BLE was to enable exchange of information with peripheral devices. However, as it is currently the most universal way for nearby communication on mobile devices that does not require any user interaction, it is potentially suitable for any type of communication. Since Android 5.0, it is possible to create a custom GATT server which allows two Android devices to communicate with each other over BLE. We now proceed to designing a system architecture for P2P communication using BLE. The overall high-level architecture of data flow is shown in Figure 6.1.

The BLE module should be composed of several submodules with clearly separated responsibilities. The communication begins by the device A broadcasting connectable advertising packets using **BLE Advertiser**. The advertising packet contains:

- a *service UUID* which identifies our application,
- a *transmission power level* in dB which can be used by the receiver to calculate a path loss and estimate the distance between devices, and
- a *peer ID* which identifies the broadcasting device.

The device B then scans for advertising packets using a **BLE Scanner**. It should filter packets by service UUID to receive only packets relevant to our application. The BLE scan is a power-intensive operation, so it should be performed only when the user actually wants to connect to a new device. It could be done e.g. only when the application is in the foreground. In case we are designing a long-running service that should run in the background, a scan should be run periodically. We should have an option to specify a scan window duration and an interval between individual scans.

Once the scanner receives an advertising packet, it creates a *peer candidate* which consists of a peer ID, a Bluetooth device address which can be used to initiate a connection, a transmission power level in dB, and a received signal strength (RSSI) in dBm. It stores the peer candidate into the **Peer Store**.

The **Discovery Strategy** is responsible for selecting which peer we should connect to. The strategy should be application-specific and can e.g. prefer to connect to devices with the largest RSSI value, or to connect only to known peers based on their peer ID. Once the strategy selects a peer it wants to contact, it is sent to the **Connection Manager**.

The connection manager contains all GATT-related communication logic. Each device has exactly one GATT server and an arbitrary number of GATT client instances, one for each device it is connected to. The GATT server implements a single GATT service containing two characteristics which are shown in Figure 4.5. The **Public Key** characteristic has a *readable* permission and simply contains a public key of the peer. It is used to determine the identity of the device when initializing the connection and can be used for authentication and encryption in the further communication. The **Writer** characteristic with a *writable* permission is then used for sending data from the client to the server. As we want to support bidirectional communication, every two devices need to have a pair of client–server and server–client connections. This can be implemented in a way that every time a GATT server receives an incoming connection, the connection manager initiates an outgoing connection to the GATT server of the connecting device. Only after both links are established, the connection is considered ready.

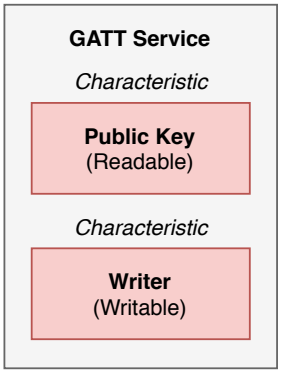


Figure 4.5: Our GATT Server Architecture

# 5

# Implementation

# 6

# Experiment

## 6.1. Analysis and Puncturing of Carrier Grade NAT

According to the report by Statista [10], there were three major mobile phone operators providing services in the Netherlands in Q4 2018, as listed in Table 6.1. In total, these represent up to 85 % of the market share. The rest of the market is shared by Mobile Virtual Network Operators who sell services over existing networks of those three operators.

We have purchased pre-paid SIM cards for all three major mobile network operators to investigate whether they are suitable for peer-to-peer communication. First, we tried to infer the characteristics of their Carrier Grade NAT deployments.

We used the STUN protocol and NAT behavior discovery mechanisms described in [7]. They have shown that all networks appear to use *Endpoint-Independent Mapping (EIM)* and *Address and Port-Dependent Filtering* (also known as *port-restricted cone NAT*). EIM is a sufficient condition for our NAT traversal mechanism to be successful, so this would make all these NATs suitable for P2P communication.

However, we also observed that NAT behavior of CGN can change over time, so STUN behavior discovery mechanism is not sufficient to correctly classify CGN behavior. We performed some more tests to verify if the behavior is consistent over time. We attempted to connect to 50 different peers over the interval of 5 minutes. We verified that KPN and T-Mobile networks are consistent with EIM behavior. However, the CGNAT used by Vodafone changes the port mapping for new connections approximately every 60 seconds, even when connecting to the same IP address and a different port. While not strictly correct due to temporal dependency, this behavior

| **Operator** | **Market share** |
|---|---|
| **KPN** | 35% |
| **Vodafone** | 25% |
| **Mobile Virtual Network Operators** | 25% |
| **T-Mobile** | 20% |

Table 6.1: Market share of mobile network operators in the Netherlands in Q4 2018. The shares do not sum up to 100% as they are rounded up within five percent ranges in the original report. [10]

could be classified as *Address and Port-Dependent Mapping (APDM)* which is characteristic for a *symmetric NAT*.

The mapped ports seem to be assigned at random from the range of 10,000 ports, which makes it infeasible to use any known symmetric NAT traversal techniques such as port prediction or multiple hole punching [13][11].

| Operator | Mapping behavior | Filtering behavior | Binding lifetime |
|---|---|---|---|
| **KPN** | EIM | APDF | ? |
| **Vodafone** | APDM | APDF | ? |
| **T-Mobile** | EIM | APDF | ? |

Table 6.2: Characteristics of CGNATs deployed by Dutch mobile network operators

## 6.2. Connectivity Test

### 6.2.1. Experimental Setup

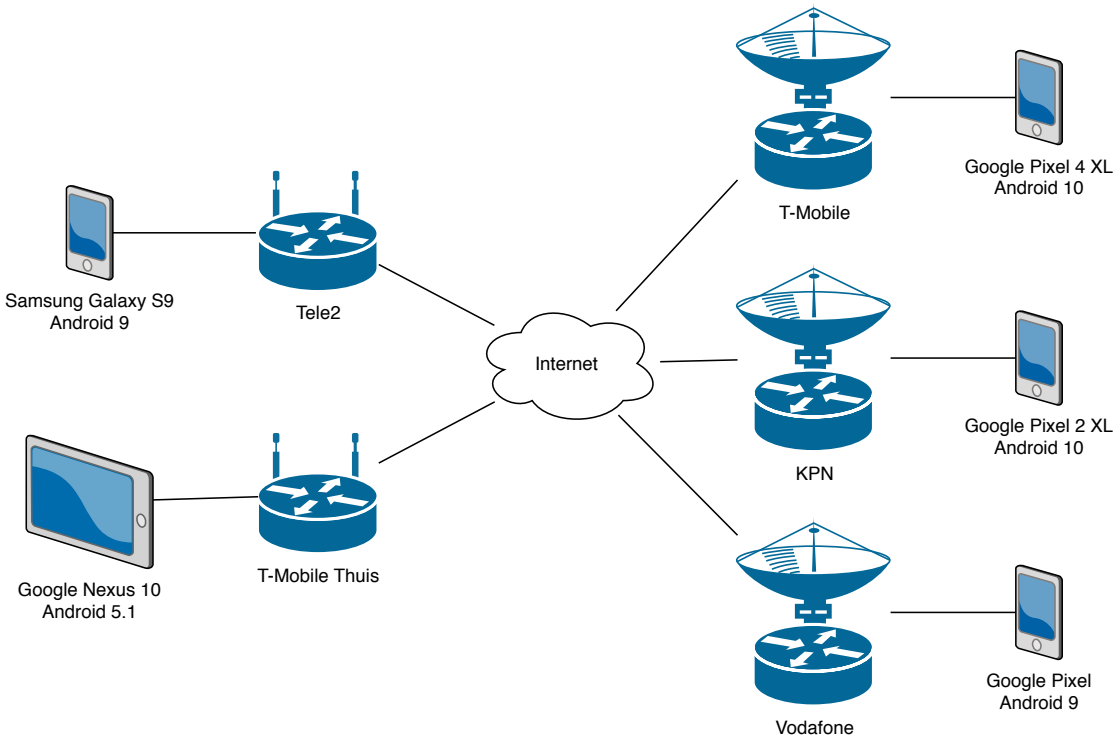


Figure 6.1: The experimental setup for the connectivity test using devices connected to different mobile (T-Mobile, KPN, Vodafone) and home broadband (Tele2, T-Mobile Thuis) ISP networks.

### 6.2.2. Results

## 6.3. Bootstrap Performance Evaluation

## 6.4. Stress Test

| | T-Mobile | KPN | Vodafone | Tele2 | T-Mobile Thuis |
|---|---|---|---|---|---|
| **T-Mobile** | ? | | | | |
| **KPN** | 1 | ? | | | |
| **Vodafone** | 1 | 1 | ? | | |
| **Tele2** | 1 | 1 | 1 | 1 | |
| **T-Mobile Thuis** | 1 | 1 | 1 | 1 | 1 |

Table 6.3: The connectivity matrix representing the success rate of connection establishment between devices connected via different ISPs.

# 7

# Conclusion

## 7.1. Future Work

# Bibliography


[1] Bluetooth SIG. Bluetooth core specification version 5.1. Accessed: Oct. 26, 2019, January 2019. URL https://www.bluetooth.com/specifications/bluetooth-core-specification/.

[2] B. Carpenter. Architectural principles of the internet. Technical report, 1996. URL https://tools.ietf.org/html/rfc1958.

[3] B. Carpenter. Middleboxes: Taxonomy and issues, 2002. URL https://tools.ietf.org/html/rfc3234.

[4] Ed. D. Wing, S. Cheshire, M. Boucadair, R. Penno, and P. Selkirk. Port Control Protocol (PCP). Technical report, 2013. URL https://tools.ietf.org/html/rfc6887.

[5] Ed. F. Audet and C. Jennings. Network address translation (nat) behavioral requirements for unicast udp. Technical report, 2007. URL https://tools.ietf.org/html/rfc4787.

[6] Gertjan Halkes and Johan Pouwelse. UDP NAT and firewall puncturing in the wild. In *NETWORKING 2011*, pages 1–12, Berlin, Heidelberg, 2011. ISBN 978-3-642-20798-3.

[7] D. MacDonald and B. Lowekamp. NAT behavior discovery using Session Traversal Utilities for NAT (STUN). RFC 5780, May 2010. URL https://tools.ietf.org/html/rfc5780.

[8] The Solid Project. Solid. Accessed: May. 13, 2020. URL https://solid.mit.edu/.

[9] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984. ISSN 0734-2071. doi: 10.1145/357401.357402. URL https://doi-org.tudelft.idm.oclc.org/10.1145/357401.357402.

[10] Statista. Distribution of mobile network connections in the netherlands in the fourth quarter of 2018, by operator. Accessed: Mar. 11, 2020. URL https://www.statista.com/statistics/765491/distribution-mobile-network-connections-in-the-netherlands-by-operator/.

[11] Y. Takeda. Symmetric NAT traversal using STUN. Technical report, June 2003. URL https://tools.ietf.org/id/draft-takeda-symmetric-nat-traversal-00.txt.

[12] Kevin Townsend, Carles Cufí, Akiba, and Robert Davidson. *Getting Started with Bluetooth Low Energy*. O'Reilly Media, Inc., 2014.

[13] Yuan Wei, Daisuke Yamada, Suguru Yoshida, and Shigeki Goto. A new method for symmetric NAT traversal in UDP and TCP. 2008.

[14] Niels Zeilemaker, Boudewijn Schoon, and Johan A. Pouwelse. Dispersy bundle synchronization. 2013.