First 5G deployment of Distributed Artificial Intelligence

Orestis Kanaris
Delft University of Technology
Delft, Netherlands
O.Kanaris@student.tudelft.nl

Johan Pouwelse (MSc Supervisor)
Delft University of Technology
Delft, Netherlands
J.A.Pouwelse@tudelft.nl

***Abstract*—**

***Index Terms*—NAT, CGNAT, 5G, Distributed Machine Learning, Mobile Machine Learning**

I. Introduction

II. Problem Description

A. Background

In recent years, the proliferation of mobile devices has reached unprecedented levels, with smartphones becoming an integral part of everyday life. These devices have increasingly powerful hardware, making them suitable candidates for running complex machine-learning models [1], [2]. Machine learning on mobile devices holds excellent potential for many applications, from personalized recommendations to democratizing big tech. One can imagine a world where every smartphone (or personal computer) holder holds their own portion of "Google's" database (and computation), having all smartphones intercommunication and share information to complete a search result, leading to a democratized distributed peer-to-peer search engine, cleansed from the big tech influence and hidden agendas [3].

However, deploying machine learning models on mobile devices presents numerous challenges, including limited computational resources, memory constraints, and the need for efficient communication between devices. The main struggle this paper focuses on is connectivity between devices since the communication in the context of this research will be handled by the IPv8[1]. IPv8 is a networking layer which offers identities and communication with some robustness and provides hooks for higher layers.

Personal devices, specifically smartphones, communicate through home Wi-Fi and mobile networks like 4/5G. Using these networks, the devices usually end up behind a home NAT or a Carrier-Grade NAT (CGNAT). The existence of these NATs makes it harder for the devices to communicate with each other since they lock their discoverability by hiding the devices behind the NAT's private network, forcing the "NATed" device to initiate the connection first. This is not a particularly impossible problem if one of the two peers has a static IP address and is discoverable. It is particularly bad when both peers are behind NATs (even worse when it is the same NAT, a problem common with CGNATs [4]), then both need to initiate the connection first, but none of them is "visible" to the other.

The STUN protocol (RFC3489 [5]) outlines four types of NATs: Full-cone NAT, Restricted-cone NAT, Port-restricted cone NAT, and Symmetric NAT. These categories are further classified in RFC4787 [4] as "easy" NATs, which employ Endpoint-Independent Mapping (EIM), and "hard" NATs, which utilize Endpoint-Dependent Mapping (EDM). EIM ensures consistency in the external address and port pair if the request originates from the same internal port.

As per V. Paulsamy et al. [6], the specifications for these NAT types are as follows:

- **Full-cone NAT:** This EIM NAT maps all requests from the same internal IP:Port pair to a corresponding public IP:Port pair. Moreover, any internet host can communicate with a LAN host by directing packets to the mapped public IP address and port.
- **Restricted-cone NAT:** Similar to Full-cone NAT, this EIM NAT maps an internal IP:Port pair to an external IP:Port pair. However, communication from an internet host to a machine behind the NAT is only allowed if initiated by that machine.
- **Port-restricted cone NAT:** Also an EIM NAT, similar to Restricted-cone NAT but with additional restrictions on port numbers.
- **Symmetric NAT:** This EDM NAT maps requests from the same internal IP:Port pair to a specific public IP:Port pair. However, it considers the packet's destination as well. Consequently, requests from the same internal pair but to different external hosts result in different mappings.

Symmetric NAT is the most "problematic" in the sense that it is the hardest one to establish a connection with if both peers are behind a NAT. Symmetric NATs behave very similar to a hard firewall; that is, they only allow incoming packets from a specific IP:Port pair only if an outgoing packet went to that destination first. The reason that one might use a symmetric NAT is when the administrator does not want to consume a single IP address per user since they theoretically allow up to 65535 simultaneous users. Symmetric NATs also give the fallacy of security, as in being behind a firewall since they

Identify applicable funding agency here. If none, delete this.

[1] https://github.com/Tribler/py-ipv8

never expose the user to the whole Internet, only to hosts that the user specifically "opted-in" to communicate with. The reason for this need for security is that the Internet lacks any security model. Anybody can freely send you an unlimited amount of data, spam, and malware [7]. A Symmetric NAT, to the average user, will not be an obstacle to their everyday browsing, but it becomes a big problem with peer-to-peer protocols, i.e. BitTorrent —In their 2008 study on fairness for BitTorrent users, J.J.D. Mol et al. [8] discovered that peers behind firewalls encounter greater challenges in achieving equitable sharing ratios. Consequently, they advocated for either puncturing NAT or employing static IP addresses to enhance network performance.

B. Research problem

The central problem of this thesis revolves around the distribution of Machine Learning on 4/5G Networks. To achieve this, one must connect efficiently to other peers through the cellular network.

Specifically, this project introduces the functionality lacking in IPv8 where they have an overlay network and APIs to connect more or less any peer devices, except when a peer is behind a Symmetric NAT. IPv8, as it stands, cannot add in the network peers behind this kind of NAT [9].

To overcome this limitation, this paper introduces a library to improve the proposal of D. Anderson's Birthday Attack blog post [10]. According to that blog post, if both peers send simultaneously $\approx$ 170000 connection-request packets, they have $\approx$ 99.9% probability of connecting. This is not entirely accurate since it doesn't consider the size of the NAT's HashTable nor the timeout time of the NAT. This paper proposes an improvement using data gathered from each provider's cellular data NAT, which is then analyzed to bias the attack to increase its success rate and avoid sending unnecessary packets that would, in turn, sabotage the attack.

The solution is a standalone open-source Kotlin library introduced in the following sections. It is evaluated both as a standalone library and also as part of IPv8, where the machine learning workload of TensorFlow Lite [11] will be distributed on Android mobile phones using the IPv8's ecosystem.

C. Objectives

The primary objectives of this thesis are as follows:

1) Address the NAT puncturing problem to enable seamless connectivity among devices, even when behind NATs or firewalls, by developing a NAT puncturing library in Kotlin.
2) Evaluate the proposed framework's performance, scalability, and resource utilization through experimental validation and benchmarking on Android devices obtained from the Tribler lab[2].

[2]https://www.tribler.org/about.html

III. Methodology

The first step in having peer-to-peer distributed AI applications run on mobile phones using a cellular network is establishing a connection between two (or more) mobile phones. To achieve that communication, the communication parameters need to be known, i.e. the type of NAT used, timeouts and the maximum data that can be transmitted. These parameters are extremely useful in maintaining the communication channel and choosing connectivity strategies, but the telephone carriers do not make them available to the public. The algorithms used to estimate these parameters are in the first three sections of this chapter.

In order to have peer-to-peer distributed AI applications run on mobile phones using a cellular network, one needs to first "connect" these mobile phones. This part builds on top of the approach proposed by D. Anderson [10], which was analysed further in a previous study by the same authors of this study [9], which suggests a method for peer-to-peer communication through the randomized exchange of packets until a successful "match" is achieved.

Anderson's approach of performing a Birthday attack to reduce the number of packet exchanges performed yielded an underwhelming success rate on the Dutch carriers as discussed in section **??**; thus, an analysis of the inner workings of the NATs used by the carriers was performed to utilize that knowledge and potentially increase the connectivity (success) rate.

The implementation of all algorithms found in these chapters are available in GitHub [12]

A. NAT Types

As already mentioned, Symmetric NAT can severely restrict P2P connectivity, which is the main NAT type that requires an alternative connectivity method. All other NAT types can get away with having some "middle-man" (another peer in the case of full distribution) to keep track of the NAT mapping (of the new peer) and communicate it to the peers wanting to connect to them. Connecting to a NAT by trying all possible combinations of ports is a very costly operation and thus should be avoided whenever necessary (no peer is behind a Symmetric NAT). A problem arises when there are no other peers to relay information; thus, the NAT needs to be attacked for a connection to be established.

The NAT types, determined from algorithm 1 are particularly useful in the case that there is a network already establish and information about peers can be passed around.

Algorithm 1 is based on RFC3489 [5] where the client (in this case the mobile phone) sends a Binding Request —over UDP— to a STUN server in order to determine the bindings allocated by the NATs. The STUN server will respond with a message containing the IP address and port that the request came from. The client will then send more Binding Requests to different ports and different STUN servers.

With the responses of these requests the client can then determine the NAT type that they are behind by analysing how the responses of the STUN servers changed.

Algorithm 1 STUN Test, NAT Type Detection, and Getting IP Information

```
function STUNTEST(sock, host, port, sendData)
    Initialize response data structure
    Convert sendData to hex byte array with headers
    Send byte array to (host, port)
    Receive and decode response packet
    if response matches and transaction ID correct then
        Parse attributes like Mapped Address, Source Address, etc.
    end if
    return response
end function
function GETNATTYPE(s, sourceIp, stunHost, stunPort)
    Attempt STUN test with provided or default server
    if initial test fails then
        for all server in STUN_SERVERS do
            Attempt STUN test with server
        end for
    end if
    Determine NAT type based on test results
    Perform additional tests for refining NAT type
    return NAT type
end function
function GETIPINFO(sourceIp, sourcePort, stunHost, stunPort)
    Create socket with specified source IP and port
    Determine NAT type using GETNATTYPE
    Close socket
    return NAT type, external IP, and external port
end function
```

The type of NAT used by the carriers tested are shown in table III.

B. Determining NAT Timeouts

To get a clear idea of how the NAT mappings over UDP work, one can imagine the first outgoing packet as both a regular packet and a connection initiation message. When this first packet is sent, the NAT that the packet was sent from starts a timer as soon as the packet leaves. That timer waits for a response from the receiving client, meaning that the packet was received/accepted, and regular communication will follow. This timer will be referred to as `connection initiation timeout` throughout this section. Knowing this parameter is very useful for the case of a fully collaborative distributed network since the connection initiation timeout is the time that the peers have to collaborate and connect the new joiner based on the NAT mapping that the new joiner advertised.

The second type of timeout is called `session timeout`, meaning how long will the mapping remain active while there are no outgoing or incoming packet flows? Knowing how long the session can remain active while idle is used to determine how often "connection maintenance" packets need to be sent to keep the connection alive. Once a connection is established, it is preferred to be maintained since maintaining a connection is much "cheaper" than re-establishing one.

The reason that the two are separated is because usually the connection initiation timeout is much smaller than the session timeout.

Starting with determining the connection initiation timeout, initially, algorithm 2 establishes a lower and an upper bound on the time that the mapping will remain active while waiting for a response. This is achieved by sending a packet to the server, which the server waits a fixed amount of time before sending a response. The time the server waits is incremented by a fixed number after each packet is received. When no response is received by the mobile phone —meaning that the NAT mapping disappeared— the time that the server waited to send the response is the upper bound of the timeout. The wait time of the last received packet will be the lower bound.

When the bounds are established, a binary search (algorithm 3 is performed on those bounds to find the precise —down to the second— timeout of the NAT.

Algorithm 2 Function to find the connection initiation timeout upper and lower bounds

```
function CONNECTIONINITIATIONTIMEOUTBOUNDS
    delay ← 0
    INC ← IncrementationConstant
    create UDP Socket
    do
        delay ← delay + INC
        sendUDPPacket(delay)
    while timeoutMsgRcvr(delay) is true
    ConnectionInitTimeBinary(delay − INC, delay)
end function
```

Algorithm 3 Binary search on the timeout interval to get accuracy to the second

```
function CONNECTIONINITTIMEBINARY(l, r)
    while l ≤ r do
        delay ← (l + r)/2
        sendUDPPacket(delay)
        responseRcvd ← timeoutMsgRcvr(delay)
        if responseRcvd then
            l ← delay + 1
        else
            r ← delay − 1
        end if
    end while
return l, r
end function
```

The algorithm for determining the session timeout, is very similar to the one for connection initiation timeout; Initially, algorithm 4 establishes a lower and upper bound on the idleness time of a connection. The algorithm works as follows: The client sends a packet to the server, and the server responds with the port number from which the client sends it. Then,

the client waits a fixed amount of time until it sends the next packet. The wait time is incremented by a fixed number after each packet is sent. The client compares the port in the body of the server's response i.e. the port that server believes that the client sent the message from. If the port in the latest response is not the same with the the one in the previous means that the mapping timed out and a new one was created.

When the bounds are determined, a binary search is run within those bounds to precisely determine the expiration time down to the second as shown in algorithm 5.

Algorithm 4 Function to find how the lower and upper bound of how long a NAT mapping is active while there is no incoming or outgoing packets

```
1:  function SessionTimeoutBounds
2:      delay ← 0
3:      INC ← IncrementationConstant
4:      create UDP Socket
5:      prev_port ← null
6:      do
7:          wait(delay × 1000)
8:          sendUDPPacket("TIMEOUT − TEST")
9:          resp ← timeoutMsgRcvr()
10:         port ← extract_port(resp)
11:         if prev_port = null then
12:             prev_port ← port
13:         end if
14:         delay ← delay + INC
15:     while prev_port = port
16:     l ← delay − (2 × INC)
17:     r ← delay − INC
18:     SessionTimeoutBinary(l, r)
19: end function
```

The results of multiple runs of these algorithms on different telecom carriers can be seen in section VI-C.

C. Maximum Transmission Unit

The maximum transmission unit (MTU) denotes the maximum size of a single data unit that can be sent in a network layer transaction. MTU is related to the maximum frame size at the data link layer (such as an Ethernet frame).

A larger MTU is linked with reduced overhead, allowing more data to be transmitted in each packet. Conversely, smaller MTU values can help decrease network delay by facilitating quicker processing and transmission of smaller packets.The determination of the appropriate MTU often hinges on the capabilities of the underlying network and may require manual or automatic adjustment to ensure that outgoing packets don't exceed these capabilities.

A jumbo frame is an Ethernet frame with a payload greater than the standard maximum transmission unit (MTU) of 1,500 bytes.

Algorithm 6 is used to determine the MTU of each carrier by running this algorithm each time with different sim cards from different providers. The algorithm is a binary search which

Algorithm 5 Function to find exactly how long a NAT mapping is active while there are no incoming or outgoing packets

```
1:  function SessionTimeoutBinary(l, r)
2:      sendUDPPacket("TIMEOUT-TEST")
3:      response ← timeoutMessageReceiver()
4:      latestPort ← extract_port(response)
5:      while l ≤ r do
6:          midpoint ← floor((l + r)/2)
7:          delay(midpoint ∗ 1000)
8:          sendUDPPacket("TIMEOUT-TEST")
9:          response ← timeoutMessageReceiver()
10:         port ← extract_port(response)
11:         if latestPort = port then
12:             l ← midpoint + 1
13:         else
14:             r ← midpoint − 1
15:             latestPort ← port
16:         end if
17:     end while
18:     return r, l
19: end function
```

Algorithm 6 Function to find the Maximum transmission unit of a carrier

```
1:  function findMTU
2:      icmp ← new Icmp4a()
3:      left ← 0
4:      right ← 65507
5:      while left < right do
6:          midPoint ← floor((left + right)/2)
7:          result ← icmp.ping(packetSize = midPoint)
8:          switch result do
9:              case Success
10:                 left ← midPoint + 1
11:             case Failed
12:                 right ← midPoint − 1
13:     end while
14:     return right
15: end function
```

tries to find the precise number of bytes, where one more byte will cause the packet to be split into two. Table V shows the MTU of the different providers tested and whether they support Jumbo frames.

D. Simple Birthday Attack

The rule for communicating in a NATed network is that the person behind the NAT must initiate communication first. The assumption is that the Internet works mainly in a Client-Server fashion where the Server is discoverable (has a *Public Static IP address*). This assumption breaks in the case of peer-to-peer communication between two clients behind a NAT since none are discoverable, no one can initiate the communication.

Algorithm 7 Simple Birthday Attack

Require: On packet received, send an ACK
Require: On packet received, $ack\_rcvd \leftarrow True$
Require: On packet received, store senders port
  $ack\_rcvd \leftarrow False$
  open UDP socket
  $msgs\_sent \leftarrow 0$
  $UUID \leftarrow generate\_UUID()$
  $packet \leftarrow create\_packet(UUID)$
  while $msgs\_sent < 243587$ and no ack_rcvd **do**
    $port \leftarrow get_random_port()$
    $send\_packet(port, packet)$
  end while
  if ack_rcvd **then**
    $maintain\_connection(IP, port\_no)$
  else
    Birthday Attack was unsuccessful
  end if

A solution to this is as explained in [9], [10]. Both peers should send packets to random ports until a "match" is achieved. A match is when peer A sends a packet from port X to port Y, and peer B sends a packet from port Y to port X in a timeframe smaller than their NAT's timeout. One can understand that the probability of this match is $\frac{1}{65535^2}$, which is almost impossible to achieve given restrictions that will be imposed by the carriers when a huge amount of rapid requests will be fired towards the NAT, let alone it will take a lot of time.

This can be improved using a Birthday Attack, which is an attack built on the Birthday Paradox [13], a counterintuitive probability theory concept that states that in a group of just 23 people, there's a better than 50% chance that two people share the same birthday. This might seem surprising, as intuition might lead one to think that with 365 days in a year, it would require many more people to have such a high probability of a shared birthday. The paradox arises because we're not just looking for a specific birthday match but any pair of people with matching birthdays. The probability of any two people not sharing a birthday decreases as more people are added to the group, and the opposite, the probability of at least one pair sharing a birthday increases rapidly.

The birthday paradox can be used to reduce the number of combinations of $sender\_port, receiver\_port$ while maintaining a satisfactory match probability. From the Birthday Paradox calculator [14], one can get a 50% success rate of a match after sending 77162 packets, and for a 99.9% success rate, 243587 packets are needed. Due to the nature of NATs (timeouts and a limited number of mapping maintained), these probabilities are unlikely to occur, but this would be the case even if all combinations are attempted.

Using the numbers above, an Android application [12] was developed to attempt to connect two mobile peers using 4/5G (which is by default using a NAT) using algorithm 7.

The results of the evaluation of 10 runs per carrier are shown in table I. A green **S** signifies that sometime during the Birthday Attack (a set of 243587 random requests), one went through, and communication was established. An **F** means no attempt went through; thus, the birthday attack failed. The evaluation of the simple birthday attack did not show auspicious results. The first conclusion that can be derived is that whether the attack will lead to a connection is very dependent on the telecommunication carrier pair. As one can see, when Vodaphone was one of the peers, there was always a successful attack. Another fascinating result is that only Vodaphone connected with a carrier of the same type, which was also the trial with the most successful connections.

These aside, a very small part of the trials resulted in at least one successful connection, let alone that many of the carrier combinations never succeeded. Even once they manage to eventually succeed at connecting, it is not satisfactory since one successful attempt out of ten makes this protocol costly in terms of cellular data used and time inefficient since coordinating two users to start attacking at the same time is already hard and error-prone on its own, doing it multiple times to achieve a single connection will deem the algorithm not very useful.

E. Improving the Birthday Attack

Given the complexity and the cost of cellular data and time, the success rate of the simple birthday attack, as shown in table I, is unsatisfactory; after multiple hypotheses on how to improve the connectivity based on time of connection, area, etc. The most prominent idea that the research team came up with is to understand the inner workings of each NAT, i.e., discover how the mapping works, whether there are any consistent patterns followed, etc. and then use that to predict what will be the next ports that each NAT will map the requests to. So instead of trying to connect to random ports of the peer, make a prediction —partially based on the NAT's inner workings, partially random, to enable exploration and exploitation— of what port the peer's NAT will map to and attempt that. The peer will do the same, thus potentially increasing the probability of connecting. This will still be attempted based on the Birthday Paradox 99.9% probability of success, i.e. 243587 attempts.

To understand the inner workings of the NATs, it was first assumed that no carrier uses the same NAT (in terms of configuration) since it was observed from different experiments, i.e. NAT timeouts, that even though there are some standards on how a NAT should be configured such as RFC 2663 and 4787 [4], [15] it is not necessarily followed; thus an Android mobile client and a Kotlin server were developed [16] to gather data on each carrier that a SIM card could be easily acquired by visiting the country, buying SIM cards and gathering data on their local network. The mobile client sends packets containing a UUID[3] to the server from random mobile ports to random server ports. The UUID, a timestamp, and the source and destination ports are saved in a CSV file for each packet sent.

[3]https://docs.oracle.com/javase/8/docs/api/java/util/UUID.html

	Odido	Lebara	LycaMobile	VodaFone	KPN
Odido	F,F,F,F,F,F,F,F,F,F	-	-	-	-
Lebara	F,F,F,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F	-	-	-
LycaMobile	F,F,F,F,S,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F	-	-
VodaFone	F,F,F,S,F,S,F,F,F,F	F,F,F,F,S,F,F,F,F,F	F,F,F,F,F,S,F,F,F	F,F,F,S,S,F,F,S,F,S	-
KPN	F,F,F,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F

TABLE I: Results of 10 consecutive simple Birthday Attacks for each pair of Carriers (F= No connection, S = Successful connection sometime during the attack)

The server, which lies behind an unrestricted network having its own static IP address, does the same; once a packet is received, it stores the UUID (which is in the body of the packet), the port that the mobile sent it from (NAT mapping), and the port that the server received it from together with a timestamp on when the packet was received. The two CSVs are then inner-joined on the UUID column, resulting in two crucial columns: the port the mobile believes it sent the packet from and the port the packet came from, i.e. the exact NAT mapping.

To figure out the algorithm behind each mapping, i.e. what drives the decision-making on which port maps to which and when, a manual Exploratory Data Analysis (EDA) [17] is performed [18] to uncover the hidden inner workings of each NAT. The questions that the EDA aims to answer are:

1) Is the first port mapping completely random?
2) Is the mapping following some pattern?
3) Does the pattern, if it exists, depend on the port choices, sender or receiver? Is it time-based?
4) For how long is the pattern being followed, and if it changes at some point, why?

To answer these, different tests are performed while trying to make sense by visualizing the data or analysing time or population-based windows. The results of this EDA are explained in section VI-A.

Using the findings of the EDA, a new connectivity library was developed. The algorithm is very similar to algorithm 7 used for the simple birthday attack, with two major differences. First, instead of choosing a random port to send to, it chooses the port based on the peer's port-choosing algorithm shown in table II. This means that to have a higher probability of connecting to some peer; one needs to know the carrier from which the peer is connected. The second difference is that the phone opens one hundred random ports and listens to all of them simultaneously —this, though, has the side effect of sometimes overloading the phone's CPU and using a lot of memory. Thus, some of the experiments ended in no connection since an exception was thrown by the CPU).

IV. System Design

V. Implementation

VI. Evaluation

A. Inner workings of NATs

In this chapter, we delve into the inner workings of Cellular Data NAT across various service providers. Through reverse engineering efforts targeting multiple providers, we unveil the nuanced mechanisms NAT systems employ. Each provider has a different implementation of address mapping strategies, which hinders connectivity on peer-to-peer protocols. By dissecting these NAT architectures, we gain a deeper understanding of their address-mapping strategies, which can be used to one's advantage when connecting to another peer using cellular data. The analysis is freely available on GitHub [18].

The data gathered using a simple app developed for this research [16] utilises a server with a static open IP address and a phone sending packets to the server containing a UUID. The phone stores the UUID, a timestamp, the port from which it sends the packet, and the port to which it sends it. The server stores the UUID, a timestamp, the phone's port, and the received port. Then, the two files are joined on the UUID, showing the port the phone opened and the port the NAT mapped it to. This relationship is then analyzed across runs to understand the address-mapping strategy of each NAT.

1) Lebara Netherlands: The initial observation was that many sender ports (what the server sees as return addresses) seemed to follow a linear pattern. Initially, some random port was chosen, then the next port's number would be the one of the previous +1, and so on, until a condition was met that would cause it to choose a new random port to start with and then get the consecutive ones and so on.

It was also observed that the initial random ports were often reused across sessions, but those specific numbers were not common across runs. For example, if the first port open was port 12800 and the next x ports, this sequence would be seen multiple times across the run. It is the same with the second, third, random, and consecutive ports, but not as frequently.

One can see from figure 1 that the port mapping follows a pattern. It starts from the 3625 region —region since it is not the actual starting port—, stays in that region for a bit (incrementing the port numbers by 1), chooses other random regions and then goes back to the 3625 region. One can also notice that the intervals between NAT defaults back to the 3625 region are more or less constant.

Another observation was when observing the length of the linear increases. During low traffic hours, what was observed was that the initial "random" port's number was a multiple of 256 (2^8), then the next 255 consecutive ports will be used, and then another random starting port (again multiple of 256) will be used and so on as can be seen in figure 2.

This NAT behaviour is consistent with the findings of Microsoft in 2011 [19] but builds on top of it, discovering that address blocks in the case of Lebara (and KPN, as is

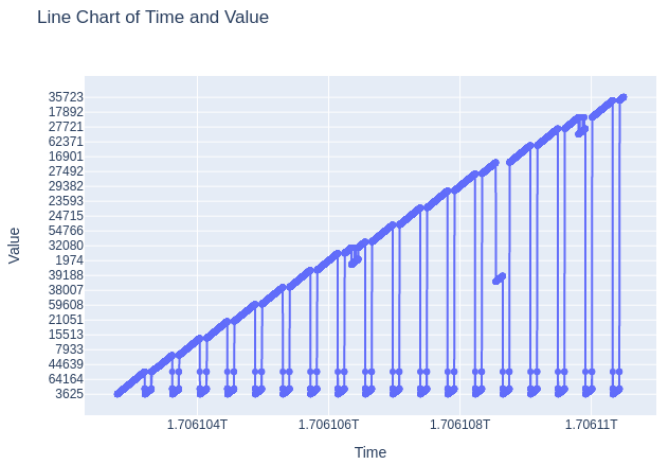

Fig. 1: Port mapping through time on a single Lebara run of $\approx$ 128 minutes

explained later) are also usage-based and not only time-based. Meaning that a user will remain on that port block until either the ports timeout or until the user consumes the whole block.

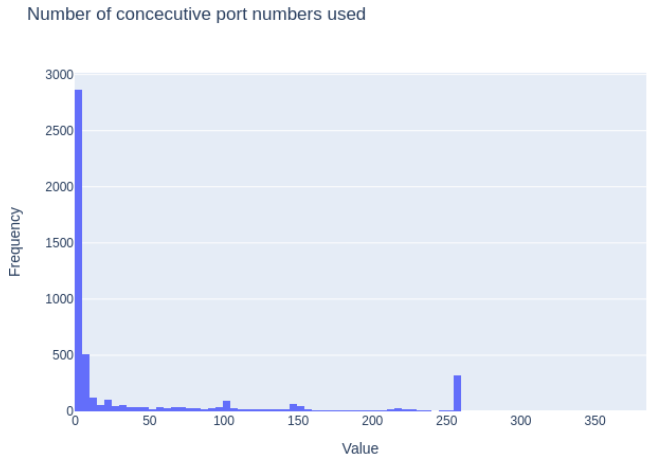

Fig. 2: Frequency of consecutive port numbers used by Lebara

The assumption is that consecutive ports are grouped together in groups of size 256. The exact number of groups cannot be inferred since not all ports were observed, but it seems to span the whole space of 65535 ports. Thus, it is assumed that there are 256 groups of 256 ports. The ports are probably grouped in queues, and then users are assigned to queues. They consume port mappings until the queue runs out of available ports; then, they are assigned to another queue. When the ports are freed or timed out, they return to their queue.

The strategy on who is assigned to which queue cannot yet be inferred, but it is probably either based on the number of consumers in the queue or the queue size. Both of these strategies are reasonable because the same ranges are consumed repeatedly since they timeout and the queue gets full again, and no one is currently using it since it was empty.

Both strategies are also validated throughout the day. On low traffic hours, the test phone was assigned a full queue, which may be either because of the queue size or because no one else was consuming (morning hours in a residential area). Similar to peak traffic hours on the university campus, many times, the phone achieved a significant amount of consecutive ports. This means that there is some strategy on the NAT to give the user as many ports as possible, again, either through the number of consumers on the specific port range or based on the number of available ports on that range.

2) KPN: KPN behaves exactly like Lebara, that is ports are grouped in groups of 256 with consecutive port numbers. The main difference between KPN and Lebara is that KPN has more infrastructure than Lebara —since Lebara is renting infrastructure from KPN— thus, as one can see in figure 3, the test phone managed to consume much more groups of 256 consecutive ports in its entirety than on Lebara. This is likely the case because of the difference in the number of users per infrastructure.

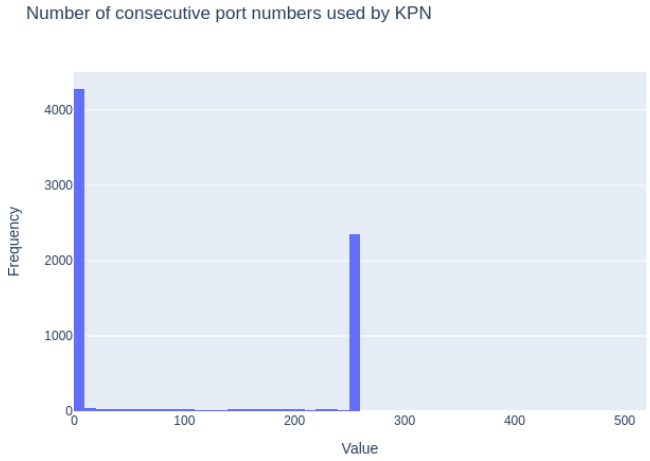

Fig. 3: Frequency of consecutive port numbers used by KPN

The number of subscribers on a single KPN hardware/ IP address makes KPN significantly more predictable than Lebara. During the testing period, the test phone consumed a port group in its entirety 32.3% of the time. On top of that 36.9% of the time, the phone got assigned to a group where the initial port was available (the port number was divisible by 256). This observation gives birth to a strategy of trying port numbers divisible by 256. This may significantly increase the probability of achieving a collision since the phone can perform a request every $\approx 15ms$, meaning that it can try all ports divisible by 256 in under 4 seconds.

3) LycaMobile Netherlands: LycaMobile, although utilizing the network of KPN, employs a different strategy for their

address mapping. After analyzing ≈ 288000 mappings, there seems to be complete randomness. No mapping is reused (very few are and in different runs; thus, they are assumed to be a coincidence), and there is no linearity on the mappings, making their address mapping strategy a First Come, First Serve on available ports.

Regarding efficiency, the theory on the inner workings is a FIFO Queue of available ports that all network subscribers subscribe to and "consume" free ports in the range $[2048, 65535]$. When a port is freed or time-out, it returns to the Queue. There is no indication of the port numbers being sorted, or eventually sorted, since consecutive ports were consumed so rarely, even on low traffic hours, that it can just be written off as a coincidence.

4) Vodafone Netherlands: For Vodaphone, more than 900 thousand mappings were collected and analysed. It is shown that Vodaphone is not following the KPN, Lebara model of dividing ports into blocks of 256, and it did not seem to be similar to LycaMobile, which assigns the user a random port.

This led to a series of tests since the graph of frequency mappings resembled a normal distribution. Performing a Shapiro-Wilk test [20] showed that it is not a normal distribution; after further examination, it was determined that it was possible to fit a beta distribution. One can see in figure 4 how well the beta distribution (orange) fits on the frequency of mappings of Vodafone (blue).

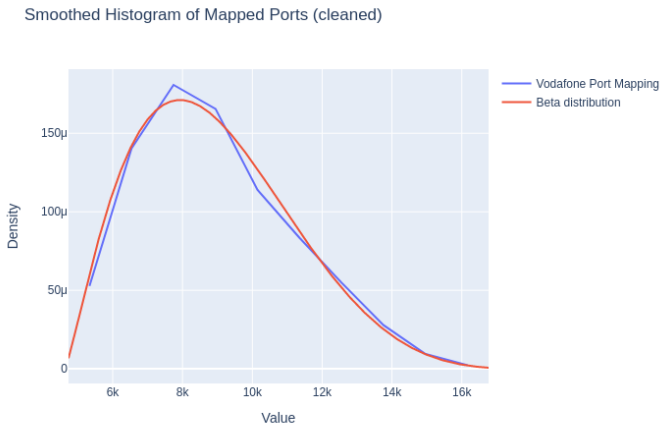

Fig. 4: Beta Distribution fitting on Vodafone's mappings

The beta distribution is a continuous probability distribution defined on the interval [0, 1]. Two shape parameters characterize the distribution, typically denoted as α, β; these parameters control the shape of the distribution. Alpha influences the shape of the distribution towards higher values, and beta influences the shape of the distribution towards lower values [21]. Two more parameters can be used, which in this case were very useful, i.e. the location parameter, which specifies the location or shift of the distribution along the x-axis and the scale, which determines the scale or spread of the distribution along the x-axis.

The empirical Beta distribution derived is: $Y \sim \mathrm{B}(\alpha, \beta, loc, scale) = \mathrm{B}(2.242, 5.008, 4630, 13937)$

5) Odido: .....

A high-level description of the inner workings of each NAT analysed in a mathematical notation can be seen in table II

B. Nat Types

Knowing the NAT type of the carrier that one is using and also of the peer they want to connect allows one to adapt their connectivity strategy to increase the chance of connecting. Different strategies should be adopted based on the types, i.e. a Symmetric NAT requires a Birthday Attack, while one can easily connect with a peer behind a Full-Cone NAT using a STUN server. The types of the NATs of various carriers are presented in table III.

C. Timeout of NATs

Understanding a NAT's timeout is crucial for efficient network management and troubleshooting. First, knowing the timeout period for waiting for a response ensures that THE protocol administrators can optimize their network configurations for timely communication. Second, it was useful in understanding patterns of the NAT's behaviour, such as why some port numbers are being reused repeatedly in similar time frames. Thus, aligning the NAT puncturing strategies with the expected timeout duration increases the probability of a puncture. These timeouts are shown in the left side of table IV

Secondly, NATs also have an idleness timeout, i.e., a mapping is deleted if it is not used. Establishing a connection is costly; thus, maintaining it, even if not needed at some particular instance, is the way to go. To achieve that, one needs to send connection maintenance packets, i.e. empty packets, that will trick the NAT into thinking that some communication is still happening. One can use the precise timeout of the NAT to send these packets at intervals that will not flood the network unnecessarily while also ensuring that the connection stays active. These timeouts are shown in the right side of table IV.

D. Maximum Transmission Unit

Knowing a carrier network's Maximum Transmission Unit (MTU) offers several advantages. Firstly, it helps optimize network performance by determining the largest packet size that can be transmitted without fragmentation, reducing overhead and latency. Additionally, understanding the MTU enables efficient bandwidth utilisation, as smaller packets may lead to increased overhead and decreased throughput. Knowledge of the MTU facilitates troubleshooting network issues, allowing for more effective diagnosis and resolution.

As for jumbo frames, their presence further enhances network efficiency by supporting larger packet sizes than standard MTU, thereby reducing the overhead of transmitting data. However, it's important to ensure compatibility with all devices and networks involved to fully leverage the benefits of jumbo frames.

Country	Name	Algorithm	Infrastructure Owner	ID Required
Netherlands	KPN	Let B_i represent a block of 256 port numbers [$\mathrm{B}_i = 256 \times i, 256 \times i + 1, \ldots, 256 \times i + 255] for i = 0, 1, \ldots, 255$ The user is randomly assigned to a block B_i, which has available port numbers When B_i has no more available ports, the user is assigned to B_j, etc.	✓	✓
Netherlands	Lebara	Let B_i represent a block of 256 port numbers $[B_i = \{256 \times i, 256 \times i + 1, \ldots, 256 \times i + 255\}] for i = 0, 1, \ldots, 255$ The user is randomly assigned to a block B_i, which has available port numbers When B_i has no more available ports, the user is assigned to B_j, etc.	KPN	×
Netherlands	LycaMobile	Random Sampling from the block [2048, 65535]	KPN	×
Netherlands	Vodafone	Beta Distribution: $Y \sim \mathrm{B}(\alpha, \beta, loc, scale) = \mathrm{B}(2.242, 5.008, 4630, 13937)$	✓	×
Netherlands	Odido		✓	×
France	Orange		✓	✓
France	SFR		✓	✓
Belgium	Orange		✓	✓
Belgium	LycaMobile		TeleNet	✓
Norway	Telia		✓	✓
Norway	MyCall		Telia	✓
Cyprus	Epic		✓	×
Cyprus	Cyta		✓	
Cyprus	Primetel		✓	
Cyprus	Cablenet		✓	

TABLE II: The algorithm each carrier uses, in mathematical notation, and the ease of obtaining a SIM card from them.

Provider	Type	Area
Lyca NL 4G	Full Cone	Echo Tu delft
Lyca NL 5g	Full Cone	Echo Tu delft
Vodaphone 4G	Restrict NAT	Echo Tu delft
Vodaphone 5G	Restrict NAT	Echo Tu delft
KPN 4G	Symmetric NAT	Echo Tu delft
Lebara 4G	Restrict NAT	Echo Tu delft
Orange Belgium 4G	Symmetric NAT	Spiti tu giorgou Bg
Lyca Mobile Belgium 4G	Restrict NAT	Spiti tu giorgou Bg
MyCall Norway 4G	Full Cone NAT	Oslo Airport
Telia Norway 5G	Restrict NAT	Oslo Airport
Telia Norway 4G	Restrict NAT	Oslo Airport

TABLE III: Nat Types of all the carriers tested and the location of the test

The MTU of various carriers and whether their network supports jumbo frames is presented in table V.

E. Roaming

Roaming seems to be significantly affecting the birthday attack. No conclusive evidence is derived on a universal change in the behaviour of carriers while roaming since it seems that every carrier behaves differently. Some anecdotal evidence of the change are:

1) **Telia and MyCall Norway:** Telia and MyCall both had a 120 second timeout on waiting for a response on the first packet send —measured in the Oslo airport. When measure in Delft, Netherlands (both tunneling through LycaMobile) the timeout fell to 5 seconds and 19 seconds respectively thus indicating that roaming affects the behaviour of the timeout.
2) **LycaMobile Belgium:** LycaMobile Belgium was never tested in Belgium due to technical difficulties but when measured in Delft, Netherlands no response reached the phone. Indicating that the timeout is so small, even the smallest delay will lead to a timeout. This does not make sense to be the standard behaviour since it can lead to a very bad user experience. Also, since no carrier tested in their local country has such a tiny delay, it hints that, potentially, the timeout changes when roaming. However, for LycaMobile, this has not been proven beyond doubt.
3) **Vodaphone Netherlands:** A birthday attack between two phones on the Vodaphone NL network where each phone is choosing random ports (each attack was comprised of 170000 attempts) led to a success rate of 4 out of 10 attempts (an attempt is a full birthday attack, all 170k requests and a success means that in those 170k packets sent, one was received). Upon attempting to re-measure this from Cyprus (thus, the two phones would be roaming to the Vodaphone NL network through Cyprus' CytaMobile-Vodaphone network) using exactly the same code at a low traffic hour, there were zero successful birthday attacks in ≈ 40 attempts. Although not a definite conclusion, this also hints that the network infrastructure behaves differently to some extent when roaming. Note that when Vodaphone was tested while roaming, the timeout and NAT types remained the same.

An initial vision of this project was to derive and present a matrix showing how much time (measured in Birthday Attack attempts) is required for each European carrier to connect. The main birthday attack attempts were made by choosing random ports with Dutch carriers in the Netherlands. The results of tableI, although not having outstanding results, show that birthday attack and connectivity between carriers is possible (especially if one phone is on WiFi, leading to a 100% success rate which each birthday attack attempt achieving connectivity); this is not the case as soon as at least one is roaming. As soon as a carrier is roaming, no birthday attack is successful.

Multiple attempts were made (with proven working soft-

Connection Initiation Timeout						Session Timeout				
	LB(s)	UB(s)	Server Port	Location	Tunnel	LB(s)	UB(s)	Server Port	Location	Tunnel
Lebara NL	120	121	2000	Echo TuD	-	240	241	2000	Echo TuD	-
Lyca NL	120	121	2000	Echo TuD	-	120	121	2000	Echo TuD	-
Odido NL					-					-
Vodaphone NL	302	303	2000	Echo TuD	-	299	300	2000	Echo TuD	-
KPN	120	121	2000	Echo TuD	-	239	240	2000	Echo TuD	-
Orange BG	58	59	2000	Echo TuD	Odido	60	61	2000	Echo TuD	Odido
Orange BG	59	60	2000	Echo TuD	Lyca NL	57	58	2000	Echo TuD	Lyca NL
Lyca BG	-	-	2000	Echo TuD	Lyca NL			2000	Echo TuD	Lyca NL
Telia NO	120	121	2000	Oslo airport	-					
Telia NO	5	6	2000	Echo TuD	Lyca NL	300	301	2000	Echo TuD	Lyca NL
MyCall NO	120	121	2001	Oslo airport	-					
MyCall NO	19	20	2001	Echo TuD	Lyca NL	299	300	2001	Echo TuD	Lyca NL

TABLE IV: Timeouts of various carriers in seconds. On the left side are timeouts for waiting for communication establishment (i.e., the initial packet is sent, but no response has yet been received from the server). On the right side is the communication timeout, i.e., communication is established, but no communication occurs. (UB=Upper Bound, LB= Lower Bound. The two-letter code next to the carrier is the ISO country code.)

Provider	MTU (BYTES)	Allows Jumbo Frames?	Area
T-Mobile			
Lebara 4G	65507	Yes	Echo Tu delft
Lyca 4G	1473	No	Echo tu delft
Vodaphone 4G	1437	No	Echo Tu Delft
KPN 4G	1445	No	Echo Tu delft
Orange Belgium 4G	1472	No	Spiti tou giorgou
Lyca Mobile Belgium 4G	42987	Yes	Spiti tou giorgou Belgium
MyCall Norway 4G	65507	Yes	Oslo Airport
Telia Norway 5G	65507	Yes	Oslo Airport
Telia Norway 4G	65507	Yes	Oslo Airport

TABLE V: The MTU of various carriers and whether they accept Jumbo frames at the location of testing

ware), with various different roaming carrier combinations, including one of the two phones running on WiFi with no one leading to a successful connection. However, as soon as both phones are on local networks, there are successful attempts.

These experiments ran for $\approx$ 1 week, leading to no successful connection between roaming carriers, hinting that some roaming feature completely hinders the connectivity. No conclusion is made on whether the connectivity is impossible, but it is not possible with the current data in a reasonable timeframe; thus, the vision of a cross-carrier time-to-connect matrix is postponed.

These failures all occurred in Cyprus. It is not impossible that roaming from Cyprus is the problem. Still, there is not enough information on the precise behaviour of roaming to either support or refute the claim that the Cypriot network is the problem.

1) Roaming completely hinders the Birthday Attack: Let's take a best-case scenario for Telia Norway roaming from Cyprus on a Samsung Galaxy A53 5G (SM-A536B/DS) where the person roaming is the only one using the roaming tower for the whole duration.

There are three significant variables, i.e.

- $p \approx 2.98$**:** processing time
- $l \approx 79.20ms$**:** average network latency Limassol to Oslo [22]
- $P = 64511$**:** number of available ports, all ports except the first 1024 are available

When they send a packet, each phone, i.e., phone A and phone B, creates a NAT mapping (opens a port in the NAT); those mappings are X_A, X_B, respectively. X_A, X_B, in this case, expire every 5 seconds.

The algorithm tries different ports of the target's NAT every $(1+t)ms$ by sending a packet to that port. If the packet originated from phone A, the port is Y_A; if it originated from phone B, it is Y_B.

The algorithm aims to cause a collision while mappings remain active, i.e., attempt $(X_a, Y_a) = (Y_b, X_b)$ in a 5-second window.

The probability of such collision is: $P((X_a, Y_a) == (Y_b, X_b)) = \frac{1}{64511^2}$.

As mentioned, each mapping has a time-to-live of 5 seconds minus the latency.

In that 5-second minus latency window, it can attempt $Y = \frac{5000}{p} \approx 1678$ ports.

Thus, the probability of a successful collision in a single window is $P(CollisionInWindow_{roaming}) = \frac{Y}{64511^2} = \frac{1678}{4161669121} = 0.0000004032$.

A new window is created every l milliseconds, meaning that worst case is $\frac{1}{0.0000004032} = 2480159$ windows are needed, meaning $(2480159 * l)$ ms which is approximately $54.6hours$.

On the contrary, if there was no roaming involved (still with Telia Norway) there are again three significant variables, i.e.

- $p \approx 2.98$**:** processing time
- $l \approx 23ms$**:** average network latency of Telia Norge in Norway [23]

- $P = 64511$**:** number of available ports remain the same

The probability of a single collision remains the same, but the time-to-live of a message is larger (≈ 120 seconds); thus, more attempts can be made in a single window. In that 120-second minus latency window, it can attempt $Y = \frac{120000}{p} \approx 40269$ ports.

The probability of a successful collision in a window thus is $P(CollisionInWindow_{local}) = \frac{40269}{64511^2} = 0.00000967616$.

In the worst case,≈ 103347 windows are needed, and since a window is created every $\approx l$ milliseconds, then the worst case time needed for a birthday attack on Telia Norge in their local network in Norway is $103347 * l = 39.6$ minutes.

These simplified calculations of two best-case scenarios for the time needed to penetrate the NAT while on the local network or roaming demonstrate that roaming requires at least $82\times$ more time than running it on the local network.

F. Improved Birthday Attack Evaluation and Findings

TALK about the new library etc and analyse the results blabla

The evaluation results of 10 runs per carrier using the improved Birthday attack are shown in table VI. The success rate difference is shown in table **??**

VII. Discussion and Future Work

This exploratory study on the inner behaviour of NATs showed various interesting properties. First of all, no NAT implementation is exactly the same, although carriers that operate a mobile virtual network operator (MVNO) model like Lebara Netherlands, which uses KPN's network, have an implementation exactly the same as KPN, hinting that potentially they allow KPN to handle everything and route the traffic to them. LycaMobile Netherlands, on the other hand, implemented a different infrastructure even though they are also using KPN's infrastructure.

Birthday attacks are inherently unpredictable; even with complete knowledge of a carrier's NAT mapping function, there is some randomness and outside influences that affect the success of the attack. For example, attempting a birthday attack during peak network usage hours will most probably lead to a lower success rate than during peak hours. This is due to the carriers experiencing congestion on their network and employing some fairness protocols to allow all users to be connected, thus limiting a user that requires high network usage (one that performs a birthday attack, which constantly opens up sockets in a "robotic" way).

Another limiting factor is roaming. Roaming as explained in section VI-E significantly hindered this research since for reasons that have not yet been fully identified dims the connectivity through birthday attack almost impossible.

The main takeaway from this research is that connectivity through birthday attacks in a fully remote setting is possible in principle, but very hard to fully quantify the success rate and its reliability. There are so many factors that may jeopardize it such as NAT type of carrier, combination of carriers, time of the day, congestion of the network and roaming.

A. Future Work

VIII. Conclusion

Appendix

1) Orange Belgium:

2) LycaMobile Belgium:

3) Orange France:

4) SFR France:

5) Telia Norway:

6) MyCall Norway:

References

[1] M. S. Louis, Z. Azad, L. Delshadtehrani, S. Gupta, P. Warden, V. J. Reddi, and A. Joshi, "Towards deep learning using tensorflow lite on risc-v," in *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*, vol. 1, 2019, p. 6.

[2] J. Dai, "Real-time and accurate object detection on edge device with tensorflow lite," in *Journal of Physics: Conference Series*, vol. 1651, no. 1. IOP Publishing, 2020, p. 012114.

[3] Tribler, "msc placeholder: "swarming llm": decentralised artificial intelligence · issue 7633 · tribler/tribler." [Online]. Available: https://github.com/Tribler/tribler/issues/7633

[4] C. F. Jennings and F. Audet, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," RFC 4787, Jan. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc4787

[5] J. Rosenberg, C. Huitema, R. Mahy, and J. Weinberger, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489, Mar. 2003. [Online]. Available: https://www.rfc-editor.org/info/rfc3489

[6] V. Paulsamy and S. Chatterjee, "Network convergence and the nat/firewall problems," 2003.

[7] M. Zolotykh, "Comprehensive classification of internet background noise," 2020.

[8] J. Mol, J. Pouwelse, D. Epema, and H. Sips, "Free-riding, fairness, and firewalls in p2p file-sharing," 2008.

[9] O. Kanaris and J. Pouwelse, "Mass adoption of nats: Survey and experiments on carrier-grade nats," 2023.

[10] D. Anderson, "How nat traversal works - nat notes for nerds," Apr 2022. [Online]. Available: https://blog.apnic.net/2022/04/26/how-nat-traversal-works-nat-notes-for-nerds/

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[12] O. Kanaris, "NAT measurements gathering with Naive Birthday Attack for connecting smartphones," Dec. 2023.

[13] K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota, "Birthday paradox for multi-collisions," in *Information Security and Cryptology – ICISC 2006*, M. S. Rhee and B. Lee, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 29–40.

[14] Fast-Reflexes, "Fast-reflexes/birthdayproblem-python: Implementation of a solver of the generalized birthday problem in python." [Online]. Available: https://github.com/fast-reflexes/BirthdayProblem-Python

[15] M. Holdrege and P. Srisuresh, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663, Aug. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2663

[16] O. Kanaris, "NAT Mapping data Gathering and analysing tool," Feb. 2023.

[17] Oct 2021. [Online]. Available: https://www.ibm.com/topics/exploratory-data-analysis

[18] O. Kanaris, "Cellular Network NAT Reverse Engineering and Exploration," Apr. 2024. [Online]. Available: https://https://github.com/OrestisKan/telecom-analysis

	Odido	Lebara	LycaMobile	VodaFone	KPN
Odido	F,F,S,F,F,S,F,S,F,S				
Lebara	F,F,S,F,S,S,S,S,F,S	F,F,F,F,F,F,F,F,S,F			
LycaMobile	F,F,S,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F		
VodaFone	F,F,F,F,F,F,F,F,F,F	S,F,S,S,S,F,F,F,S,F	F,F,F,F,F,F,F,F,S,S	F,S,S,F,F,F,F,S,F,F	
KPN	F,F,F,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F	F,F,F,F,F,F,F,F,F,F	F,S,S,F,F,F,S,F,S,F	F,F,F,F,F,F,F,F,F,F

TABLE VI: Results of 10 consecutive Birthday Attacks for each pair of carriers, where the port choice is based on the carriers NAT mapping function (F= No connection, S = Successful connection sometime during the attack)

[19] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang, "An untold story of middleboxes in cellular networks," *Proceedings of the ACM SIGCOMM 2011 conference*, Aug 2011.

[20] [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html

[21] J. B. McDonald and Y. J. Xu, "A generalization of the beta distribution with applications," *Journal of Econometrics*, vol. 66, no. 1, pp. 133–152, 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304407694016124

[22] Jun 2024. [Online]. Available: https://wondernetwork.com/pings/Limassol

[23] S. Ltd., "Telia norge as speed test," Jun 2024. [Online]. Available: https://www.broadbandspeedchecker.co.uk/isp-directory/Norway/telia-norge-as.html