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 cellular providers 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 cellular providers as discussed in section **??**; thus, an analysis of the inner workings of the NATs used by the cellular providers 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 cellular providers 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 cellular providers 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 provider by running this algorithm each time with different sim cards from different providers. The algorithm is a binary search

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 provider

```
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
```

which 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 providers 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 provider 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 cellular provider pair. As one can see, when Vodaphone was one of the peers, there was always a successful attack. Another fascinating result is that only the Vodaphone connected with itself, i.e., two peers using the same provider achieved connection, 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, although many of the provider 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 provider 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 provider 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,

[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,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 cellular providers (F= No connection, S = Successful connection sometime during the attack)

and the source and destination ports are saved in a CSV file for each packet sent. 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 provider 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 experiments ended in no connection since the CPU threw an exception).

IV. System Design

V. Implementation

VI. Evaluation

A. Inner workings of NATs

This chapter presents the inner workings of cellular data NATs across various cellular providers, emphasising the five main Dutch cellular providers: KPN, Vodafone, LycaMobile, Lebara, and Odido (ex-T-Mobile Netherlands). This chapter presents the unique mechanisms each NAT employs through reverse engineering of the mappings and the behaviours of those NATs, as explained in the previous section. It is shown that each provider uses a different algorithm for the mapping and has different parameters in terms of timeouts and MTUs. By analysing these behaviours, they can be utilised to achieve the goal of a successful connection between different peers on 5G from different networks without having any middleman assisting the connectivity. The steps taken to analyse the mappings are freely available on GitHub [18] and presented in table II.

This chapter first presents and analyses the mapping algorithms of the leading Dutch cellular providers, followed by the timeouts and the MTU of all the tested carriers. It continues with a section explaining some problems encountered because of roaming, resulting in the abandonment of this project's ambitious goal, which measured the time it would take to connect from any provider tested to any other. Finally, the new library for connectivity was tested. The results are analysed to determine whether the new approach for connectivity that utilises the knowledge of the inner workings of the NATs indeed improved the success rate of connecting each Dutch carrier compared to the success rate of the simple birthday attack presented in section III-D.

1) Lebara Netherlands: When analysing Lebara, the initial observation was that many sender ports (the server sees them as return addresses) seemed to follow a linear pattern. Initially, some random port was chosen, then the next port would be the adjacent, 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 multiple times in the same session, making the first port very significant since it would be observed the most during the connection attempt. Still, those specific numbers were not common across runs. For example, if the first port open were port 12800, that and the next x ports would be seen multiple times across the attempt. 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

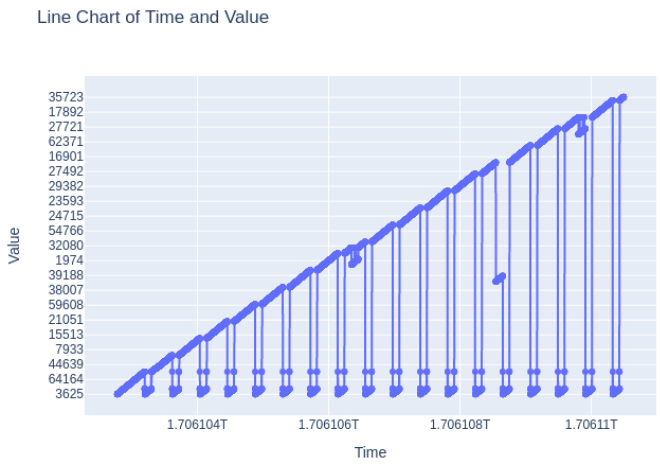

Fig. 1: Port mapping through time on a single Lebara run of $\approx$ 128 minutes

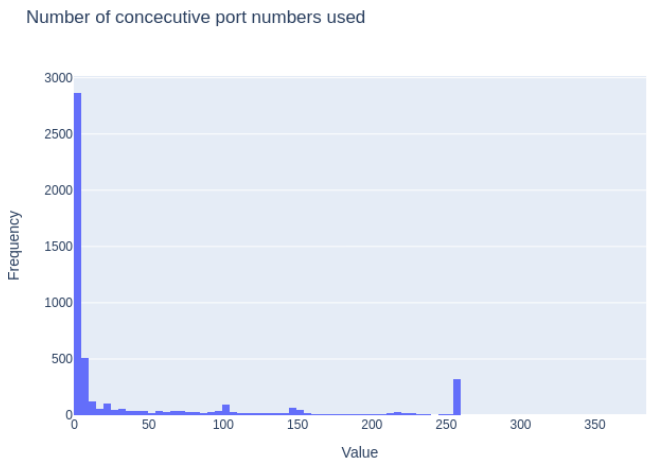

Fig. 2: Frequency of consecutive port numbers used by Lebara

3625 region are more or less constant.

Another observation was when observing the length of the linear increases. During low traffic hours, what was strongly observed (more than peak network traffic hours) 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 explained later) are also usage-based and not only time-based. This means that a user will remain on that port block until either the port timeout or the user consumes all the ports available in the block, including the whole block, in low-traffic hours.

The assumption is that consecutive ports are grouped in groups of size 256. The 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 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 of who is assigned to which block cannot yet be inferred. The two theories are based on the number of consumers in the block or based on the number of free ports (not currently used) in the block. Both of these strategies are reasonable because the same ranges are consumed repeatedly since they time out, and the block gets full again, and no one has been 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 block, which may be either because of the number of free ports in the block or because no one else was consuming —since the test was performed in the morning hours, in a residential area. For example, the phone can often get many consecutive ports during peak traffic hours on the university campus. This means there is some strategy on the NAT to give the user access to as many ports as possible in case they need it, again, either through the number of consumers on the specific block or based on the number of available ports on that block. No theory can be ruled out or verified since the number of users in each tower and their mapping behaviour are unknown. A future test to get a definite answer is presented in section VII-A

By the end of the data gathering and analysis of the NAT parameters of Lebara's NAT, the SIM used was deprovisioned with no warnings. After trying to use the SIM card a week later (which also had some credit left), and after multiple attempts on different days, it could not find the Lebara network to connect to, and attempts to connect to any other network, such as KPN's, were rejected. Attempting to call service numbers or run USSD codes led to a message saying the card was not activated. Attempts to reactivate the SIM card led to errors.

2) KPN: KPN behaves exactly like Lebara in that 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 with the model of mobile virtual network operator (MVNO)— 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 because of the difference in the number of users per IP address.

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

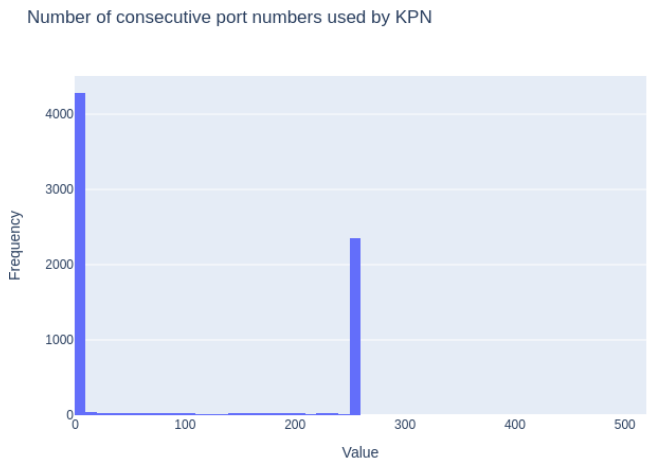

Fig. 3: Frequency of consecutive port numbers used by KPN

divisible by 256). This observation gives birth to a strategy of trying port numbers divisible by 256. This may significantly increase the probability of a collision since in the time needed for the phone to attempt all 256 port numbers divisible by 256, if no interruption occurs, all mappings will remain active simultaneously.

3) LycaMobile Netherlands: LycaMobile, although utilizing the KPN network, employs a different strategy for its address mapping than KPN. After analyzing ≈ 288000 mappings, the mapping algorithm is estimated to be random. The smallest port number used was 2048, and the biggest was 65535; hence, it uses the whole available space, with the first 2048 being reserved. Out of the 288000 mappings, there were ≈ 51000 unique mappings. Most of the time, the same port number was opened 22 times across 3 different runs. Analysing the frequency of when the port appeared again showed no consistency since the intervals in which the port was reopened ranged from 85 seconds (the earliest) to 5980 seconds (the latest), meaning no mapping or port number was algorithmically reused. This lack of consistency and no linear incrementation on the mappings shows that the address mapping strategy of the NATs of LycaMobile Netherlands is First Come, First Serve on available ports.

Another interesting finding is that the port was reopened after 85 seconds, which is not consistent with Lyca's port timeout, as explained in the following section. This behaviour hints that a port can timeout earlier if there is a need for it, either because of a lack of available ports or because the subscriber utilised more ports than what their share is.

The analysis showed that LycaMobile employs a pool of all available ports that all network subscribers subscribe to and "consume" free ports in the $[2048, 65535]$. When a port is freed, it returns to the pool. There is no indication of the port numbers being sorted, or eventually sorted in that pool, since consecutive ports were rarely consumed, even on low traffic hours, making it 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 was not 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; further examination determined that it could 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).

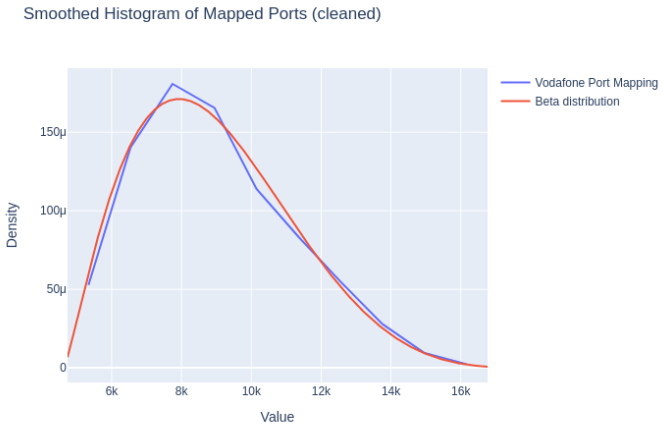

Fig. 4: Beta Distribution fitting on Vodafone's mappings

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: Odido's NAT randomly selects the port to map to. The unique property of Odido's mapping is that after analysing 493982 mappings, only 9920 ports were used by the NAT. The problem is that these mappings are scattered around the possible ports space, contrary to Vodafone only using consecutive ports. The histogram of the mappings of Odido is shown in figure 5.

Analyzing the hits of Odido, an algorithm suggestion for penetrating Odido's NAT is first to merge consecutive bins with a frequency of more than 30 (which is in the middle of the frequencies). Merging is performed to increase the space and not over-specify the algorithm since many results are time-

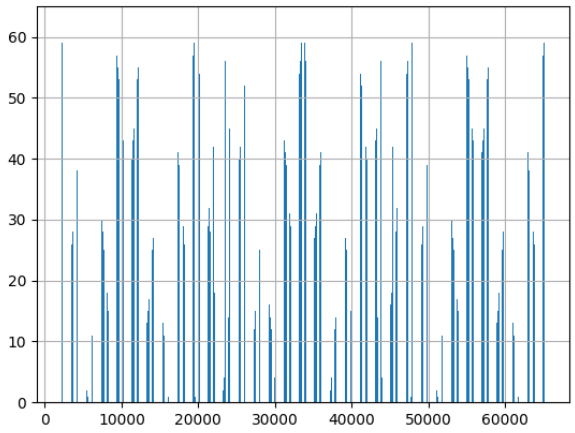

Fig. 5: Histogram of the output of Odido's mapping strategy

dependent. After merging the bins and creating the ranges, an exploitation exploration strategy will be used where, based on some weights —Odido demonstrated good results with a 40-60 weight allocation— the next port will be chosen 40% of the time, the weight will come from those established ranges and 60% of the time it will be a random port from the range $[2048, 65535]$[4]. It was tested once it was decided that the next port should come from one of the bins and whether it was better to choose a bin based on its frequency on the histogram. This strategy gave worse results due to added delay because of the number of bins or because the algorithm is over-fitted; thus, the decision to allow the bin choice to be random.

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 one is using, and the one of the peer they want to connect to 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 to connect. In contrast, one can easily connect with a peer behind a Full-Cone NAT using a STUN server or some peer acting as a middleman relaying information to the rest of the network. The types of the NATs of various carriers are presented in table III.

C. Timeout of NATs

Analyzing the NAT timeouts showed two crucial things. First, roaming highly influences the timeout, as seen from the results of the Norwegian cellular carriers and Belgium's LycaMobile. Second, the timeouts are multiples of a minute, a behaviour that is to be expected. Still, it also shows that designers did not optimize the timeouts to the full extent since it is highly unlikely that the optimal timeout is nicely rounded to the minute.

Table IV can be interpreted as `LB` being a lower bound and `UB` being an upper bound. Bounds are used since, due to network delays, it is hard to know which of the two is the actual timeout. On the left side is the timeout for how long the mapping will remain active when the recipient does not receive an initial response. On the right side is the communication timeout, i.e., communication is established (initial response is received) but is currently idle. The timeouts of each cellular provider don't differ significantly except for the two extremes, which are Lyca of Belgium being unable to establish a connection while roaming and CytaMobile of Cyprus having an enormous session timeout of half an hour.

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.

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 leverage jumbo frames' benefits fully.

The MTU of various carriers and whether their network supports jumbo frames is presented in table V. Notable findings are the Dutch Lebara and the Norwegians Telia and MyCall, which have an MTU of 65507, a big difference from other providers, even if they support jumbo frames. Findings regarding MTU across cellular providers show no consistency across carriers, especially those who chose to allow Jumbo Frames, since the three who happen to have the same jumbo frames (65507, which is also the maximum possible PING payload), the ones who do not allow Jumbo Frames are arbitrarily close to the threshold of 1500 bytes.

E. Roaming

Roaming is significantly hindering the success of a connection attempt between two peers. No definite reason is derived on why roaming hinders connectivity since many parameters of the NATs change when roaming. Some observations of behavioural and parameter changes are:

1) **Telia and MyCall Norway:** Telia and MyCall of Norway both had a 120-second timeout for connection initiation —measured in the Oslo airport. When measured in Delft, Netherlands (both tunnelling through LycaMobile), the timeout fell to 5 seconds and 19 seconds, respectively, showing that roaming changes the timeout configuration.
2) **LycaMobile Belgium:** LycaMobile Belgium's connection initiation timeout measurement failed when mea-

[4]The bins and the full implementation of the algorithm can be found in the library presented section VI-F

Country	Name	Algorithm	Infrastructure Owner	ID Required
Netherlands	KPN	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.	✓	✓
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	Semi-Random Sampling from the block $[2048, 65535]$ Sampling strategy is analysed in VI-A5	✓	×
France	Orange	Random Sampling from the block $[1, 65500]$	✓	✓
France	SFR	Random Sampling from the block $[1025, 65535]$	✓	✓
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
Odido NL 4G	Symmetric	Echo Tu Delft NL
LycaMobile NL 4G	Full Cone	Echo Tu Delft NL
LycaMobile NL 5g	Full Cone	Echo Tu Delft NL
Vodaphone NL 4G	Restrict	Echo Tu Delft NL
Vodaphone NL 5G	Restrict	Echo Tu Delft NL
KPN NL 4G	Symmetric	Echo Tu Delft NL
Lebara NL 4G	Restrict	Echo Tu Delft NL
Orange BG 4G	Symmetric	Schaarbeek Brussels BG
LycaMobile BG 4G	Restrict	Schaarbeek Brussels BG
MyCall NO 4G	Full Cone	Oslo Airport NO
Telia NO 5G	Restrict	Oslo Airport NO
Telia NO 4G	Restrict	Oslo Airport NO
Cyta CY 4G	Restrict	Aglantzia, CY
Epic CY 4G	Symmetric	Aglantzia, CY

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

sured in Delft, Netherlands, since no response reached the phone. After multiple attempts on different days to measure the timeout with no success, it's concluded that the timeout is so small that even the slightest network 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 provider tested in their local network has such a tiny delay, it is concluded that some providers' timeout changes when roaming.

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 243587 requests and a success means that in those 243587 packets sent, at least 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 the same code at a low traffic hour, there were zero successful birthday attacks in ≈ 4 hours worth of attempts (each attempt takes between one and two minutes). This is a total decrease in the success rate of connection initiation. Such a decrease shows that although on the surface, when roaming, everything looks the same, in actuality, there are significant differences. 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 is required for each European cellular provider tested to connect. As soon as at least one phone is roaming, the success rate of a birthday attack falls almost to zero. There was no successful connection between roaming peers, although theoretically possible.

Multiple attempts were made (with proven working software), with various roaming provider combinations, including one of the two phones running on WiFi with no one leading to a successful connection. However, instant successful attempts exist when both phones are on local networks.

These experiments ran for ≈ 1 week, leading to no successful connection between roaming cellular providers, showing that some roaming feature(s) hinder entirely the connectivity. No conclusion is made on whether the connectivity is impossible, but it is not possible with the current implementation in a reasonable timeframe; thus, the vision of a cross-cellular-provider time-to-connect matrix is postponed.

1) Roaming makes birthday attack-based connectivity almost impossible: This section quantifies the difference in time needed to connect two peers roaming versus two peers on the local Telia network in Norway. Starting with the roaming case, assume the best-case scenario in which the person roaming is the only one using the roaming tower for

	Connection Initiation Timeout					Session Timeout				
Provider	***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	120	121	2000	Echo TuD	-	119	120	2000	Echo TuD	-
Vodaphone NL	302	303	2000	Echo TuD	-	299	300	2000	Echo TuD	-
KPN NL	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	F	F	2000	Echo TuD	Lyca NL	F	F	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
CytaMobile CY	59	60	2000	Aglantzia, CY	-	1800	1801	2000	Aglantzia, CY	-
Epic CY	239	240	2001	Aglantzia, CY	-	240	241	2001	Aglantzia, CY	-

TABLE IV: Timeouts of various carriers in seconds. On the left side are timeouts for waiting for communication to be established.

Provider	**MTU (BYTES)**	**Allows Jumbo Frames?**	**Area**
Odido NL 4G	3972	Yes	Echo Tu Delft NL
Lebara NL 4G	65507	Yes	Echo Tu Delft NL
LycaMobile NL 4G	1473	No	Echo Tu Delft NL
Vodaphone NL 4G	1437	No	Echo Tu Delft NL
KPN NL 4G	1445	No	Echo Tu Delft NL
Orange BG 4G	1472	No	Schaarbeek Brussels BG
LycaMobile BG 4G	42987	Yes	Schaarbeek Brussels BG
MyCall NO 4G	65507	Yes	Oslo Airport NO
Telia NO 5G	65507	Yes	Oslo Airport NO
Telia NO 4G	65507	Yes	Oslo Airport NO
Cyta CY 4G	1473	No	Aglantzia, CY
Epic CY 4G	4433	Yes	Aglantzia, CY

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

the whole duration. The roaming peers are assumed to be in Cyprus for the experiment using a Samsung Galaxy A53 5G (SM-A536B/DS).

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 each phone sends a packet, the NAT each is subscribed to, i.e., NAT A and NAT B, creates a NAT mapping; let those mappings be X_A, X_B, respectively. In the case of roaming Telia, X_A, X_B have a connection initiation timeout of 5 seconds.

The algorithm tries different ports of the peer's NAT by sending packets to random ports of the peer. The packets are sent at a rate of one every p milliseconds. Let the ports that the packets are sent to be Y_B if the packet is sent to phone B or Y_A if the packet is sent to phone A. The algorithm aims to cause a collision while mappings remain active, i.e., attempt $(X_A, Y_B) = (Y_A, X_B)$ in the 5-second window that the mappings remain active. The probability of such collision is: $P((X_a, Y_a) == (Y_b, X_b)) = \frac{1}{64511^2} \approx 0.00000000024$.

As mentioned, each mapping has a time-to-live of 5 seconds minus the latency. In that 5-second minus latency window, the phone 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), this time, peers are assumed to be physically in Norway, and the three variables become:

- $p \approx 2.98$**:** 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. Cellular Provider Aware NAT Puncturing

After gathering the NAT mapping data from various cellular providers and converting them into algorithms, they were implemented in Kotlin and integrated into a connectivity library [24], which has the capability of connecting to multiple peers at once by continuously sending them `CONNECTION-INIT` packages from multiple open ports and waiting for a response to be received. The connectivity attempts are made asynchronously; thus, it does not block the main frame of the app that the library will be integrated with, allowing the user to continue using the app until the connections are established. If a response is received, the ID, IP address and port of the connected peer and the source port (where the connection was established) are returned —again asynchronously. The app implementing the library can then use it to send any message, depending on the use case. The library periodically sends `CONNECTION-MAINTENANCE` packets based on the peer's connection timeout to keep the connection active until the implementing app uses it. Those maintenance packets will continue to be sent until the implementing app stops the functionality if it has some implementation that monitors and maintains active peer connections. A demo app[5] was developed to test and evaluate the success rate of the new library compared to a simple random birthday attack. The evaluation results of 10 runs per Dutch cellular provider using the provider-aware NAT puncturing are shown in table VI. From the table, one can deduce that four more combinations succeeded in at least one successful connection establishment compared with the random birthday attack: Odido-Odido, Odido-Lebara, Lebara-Lebara and VodaFone-KPN. Odido-Vodafone resulted in no successful connections even though two successful connections were achieved using the random birthday attack, and VodaFone-VodaFone resulted in one less successful connection. Although the connectivity rate improved, a hypothesis test was developed for each cellular provider combination to determine whether the improvements were statistically significant.

The Hypothesis testing will be performed using Fisher's exact test due to the small sample size of connectivity attempts [25]. The hypotheses are as follows: the null hypothesis will be that the proportion of successful NAT punctures using cellular aware NAT puncturing compared to random birthday attack has not increased $H_0 = P_{random} \leq P_{CellularProviderAware}$. The alternative hypothesis is that the proportion of successful NAT punctures has increased using the cellular provider aware method compared to the random birthday attack $H_1 = P_{random} < P_{CellularProviderAware}$. The significance level that will be used is 0.05. If the p-value derived by Fisher's exact test is less than the significance level, the null hypothesis (H_0) is rejected. Otherwise, the alternative hypothesis is rejected. Table VII shows the p-value results of the Fisher's Exact test, with the four cells highlighted in Green being the combinations where the alternative hypothesis is accepted, meaning that the cellular provider aware NAT puncturing performed significantly better compared to the random birthday attack based NAT puncturing.

The results of cellular provider-aware NAT puncturing, although a few times showed that it improved the connectivity rate statistically significantly, did not make it worse. Since the inverse, Fisher's exact test (H_1 being that random birthday attack has a higher proportion of successful results) would not be accepted in any provider combination of the ones tested.

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: Orange France utilizes few reused ports, with no linear increments, effectively spanning the entire port space from 1 to 65,500. The distribution of port choices is highly diverse, with a maximum of 65,407 unique ports. The most frequent port was used a maximum of 20 times, with subsequent reuse counts decreasing. The total dataset

[5] https://github.com/OrestisKan/bday-demo-app

	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,FS	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,S,F,S,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)

	Odido	Lebara	LycaMobile	VodaFone	KPN
Odido	$p = 0.0433$ $p < 0.05$ H_1 is accepted				
Lebara	$p = 0.0054$ $p < 0.05$ H_1 is accepted	$p = 0.5$ $p > 0.05$ H_1 is rejected			
LycaMobile	$p = 0.7632$ $p > 0.05$ H_1 is rejected	$p = 1.0$ $p > 0.05$ H_1 is rejected	$p = 1.0$ $p > 0.05$ H_1 is rejected		
VodaFone	$p = 1.0$ $p > 0.05$ H_1 is rejected	$p = 0.0286$ $p < 0.05$ H_1 is accepted	$p = 0.5$ $p > 0.05$ H_1 is rejected	$p = 0.8251$ $p > 0.05$ H_1 is rejected	
KPN	$p = 1.0$ $p > 0.05$ H_1 is rejected	$p = 1.0$ $p > 0.05$ H_1 is rejected	$p = 1.0$ $p > 0.05$ H_1 is rejected	p=0.0433 $p < 0.05$ H_1 is accepted	$p = 1.0$ $p > 0.05$ H_1 is rejected

TABLE VII: Results of the hypothesis testing on Random Birthday Attack versus Cellular Provider Aware NAT Puncturing

comprises 433,528 entries, illustrating a broad and varied use of ports, thus concluding that the NAT mapping is random

4) SFR France: SFR demonstrates minimal reuse of ports, without any linear increments, covering the entire port space from 1,025 to 65,535. The distribution of port choices shows significant diversity, with 64,509 unique ports. The highest frequency for any single port is 8, occurring only once, with the reuse count decreasing to 5 for subsequent ports. The dataset contains 257,913 entries, highlighting a broad and varied utilization of ports, thus concluding that the NAT mapping is random.

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

[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
[24] O. Kanaris, "Birthday-Attack-based smartphone connectivity kotlin library," May 2023.
[25] A. Edwards, "Chapter 67 - r.a. fischer, statistical methods for research workers, first edition (1925)," in *Landmark Writings in Western Mathematics 1640-1940*, I. Grattan-Guinness, R. Cooke, L. Corry, P. Crépel, and N. Guicciardini, Eds. Amsterdam: Elsevier Science, 2005, pp. 856–870. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444508713501480