# 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*—**

## 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 Huawei [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.

### 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 [7].

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

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

To overcome this limitation, this paper introduces a library to improve the proposal of D. Anderson's Birthday Attack blog post [8]. 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 [9] 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].

## III. Methodology

Introduce the chapter

### A. Evaluating Naive Birthday Attack

**Algorithm 1** Naive 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** $msgs\_sent < 170000$ and no $ack\_rcvd$ **do**
    $send\_packet(packet)$
  **end while**
  **if** $ack\_rcvd$ **then**
    $maintain\_connection(IP, port\_no)$
  **else**
    Birthday Attack was unsuccessful
  **end if**

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; thus, no one can initiate the communication.

A solution to this is as explained in [7]. 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 space for this is $65535^2$ in a very tight timeframe, which is almost impossible to achieve, let alone it will take a lot of time.

This can be improved using the Birthday Paradox [], a counterintuitive probability theory concept. It states that in a group of just 23 people, there's a better than even 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 calculations of D. Anderson [8], one can get a 50% success rate of a match after sending $\approx$ 54000 packets, and a 99.9% success rate $\approx$ 170000 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 [10] was developed to attempt to connect two mobile peers using 4/5G (which is by default using a NAT) using algorithm 1.

The results of the evaluation of 10 runs per carrier are shown in table I The evaluation of the naive 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, even though half of the trials resulted in at least one successful connection, 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 protocol unusable.

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

| | Odido | Lebara | Lyca | Vodaphone |
|---|---|---|---|---|
| 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 | | |
| Lyca | 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 | |
| Vodaphone | 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 |

TABLE I: Results of 10 consecutive Birthday Attacks for each pair of Carriers (F= No connection, S = Succesful Connection)

| Odido | Vodaphone | LycaMobile | Lebara |
|---|---|---|---|
| - | - | - | - |

TABLE II: Nat types of selected Dutch carriers

### B. Analyzing NAT Timeouts and Types

The NAT timeout is divided into two parts: timeout while waiting for a response and timeout between consecutive sends. I.e., how long will the NAT mapping remain active while waiting for the receiver to respond to an outgoing packet, and how long will the mapping remain active if there are no outgoing packets (all outgoing packets have been responded to)?

Starting off with the timeout time of waiting for a response,

**Algorithm 2** Function to find the timeout in an interval of 20 seconds

```
function WAITTIMEOUTTEST
    delay ← 0
    create UDP Socket
    do
        delay ← delay + 20
        sendUDPPacket(delay)
    while timeoutMsgRcvr(delay) is true
    waitTimeoutBinaryTest(delay − 20, delay)
end function
```

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

```
function WAITTIMEOUTBINARYTEST(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 test for how much time the NAT mapping can remain idle without outgoing packets ...

*1) NAT Types:* The analysis of the types of NATs that the different Dutch Carriers are using can be found in table II. This analysis was performed using an adapted STUN client, stored on GitHub, with the rest of the analysis code used throughout this section [10].

### C. Improving the Birthday Attack

Given the cost, the success rate of the Naive Birthday attack, as shown in table I, is not satisfactory. To improve that, one needs to understand the inner workings of each NAT, i.e., how the mapping works, whether there are any patterns, etc.

To answer these questions, an Android mobile client and a kotlin server were developed [11]. The mobile client sends packets containing a UUID[3] to the server from random mobile ports to random server ports. Each UUID, source and destination port are saved in a CSV file. The server which lies behind an unrestricted network does the same; as soon as a packet is received, it stores the UUID inside the packet, the port that the mobile sent it from and the port that the server received it. The two CSVs are then inner-joining 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 NAT mapping.

How are the data analyzed, what are the results ? Also advertise that anyone can download the app etc?

## IV. System Design

## V. Implementation

## VI. Evaluation

### A. Improved Birthday Attack Evaluation and Findings

...

*1) Lebara:*

*2) Lyca:*

*3) Odido:*

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

## VII. Discussion and Future Work

## VIII. Conclusion

## 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


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

[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] L. Qiaoqiao, “What is nat? what are the nat types?” Sep 2021. [Online]. Available: https://info.support.huawei.com/info-finder/encyclopedia/en/NAT.html
[7] O. Kanaris and J. Pouwelse, “Mass adoption of nats: Survey and experiments on carrier-grade nats,” 2023.
[8] 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/
[9] 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/
[10] O. Kanaris, “NAT measurements gathering with Naive Birthday Attack for connecting smartphones,” Dec. 2023.
[11] ——, “NAT Mapping data Gathering and analysing tool,” Feb. 2023.