TrueDetective: Detecting users behind NAT by analyzing the encrypted traffic

Jaber Daneshamooz
jaber@ucsb.edu
University of California, Santa Barbara
Santa Barbara, California, USA

ABSTRACT

here is the abstract of the paper

1 INTRODUCTION

(1) What is the problem (considering the encryption)
(2) Why it is important
(3) How are we going to address this

Network Address Translation (NAT) aimed originally at resolving the issue of IPv4 address exhaustion. NAT allows multiple devices on a local network to share a single public IP address. This enables efficient utilization of the limited number of public IP addresses and allows devices on a local network to connect to the internet. NAT also provides some privacy benefits.

NAT can lead to communication problems, security vulnerabilities, difficulties in network management, and challenges in identifying network issues. it can also be taken advantage of by malicious users hiding behind NAT devices. Thus, identifying NAT devices and hosts behind them is essential to detect malicious behaviors in traffic and application usage

In the case of security, NAT devices can hide hosts connected to them, thus a host behind a NAT can perform malicious behavior without being detected [7, 8]. Furthermore, a NAT device can be installed in an intranet thus allowing unauthorized hosts access to the network and causing a shadow IT issue

Also, nowadays, ISP, while providing Internet connectivity to users, offers services based on the number of active connections (apart from other bandwidth-based plans). In case of the number of active user-based connections (e.g., ten active uses max), ISP needs to know the number of active users behind NAT to ensure that they do not cross the paid user connection quota. In such scenarios, counting of host is essential. Mobile hotspots implement NAT as well. In the rest of this paper, the term NAT refers to a router, tethering device, or a mobile hotspot while a NATted network refers to the internal network behind the NAT.

How Polynomial Regression Improves DeNATing

other applications like streaming server also require host count to serve its users better, as the server offer limited user connection due to server overload.

To tackle this issue, transfer learning was applied to enhance the model's efficiency and effectiveness

1.1 Current issues

No application level information -> encrypted traffic, privacy etc, also more like OS/Application fingerprinting. Some works just consider the existence of NAT.

we eliminate the issue of highly active user skewing the dataset by down sampling (n flows per user)

Our dataset already consists of wired and wireless users

Consider removig port??? If it creates bias based on NAT implementation

Considering tranalyzer

1.2 Removing shortcuts

more users better data, the accuracy decreases by adding more users cause it reduces the shortcuts effect, trustee, removing the parameters that changes in different network conditions (eg TTL),

seq number can be meaningful specially the last bytes and after that it is src/dst port

we can consider flow start time and end time as part of cic: how to input those values?

TODO: Use dataset with malicious users behind the nat to count the number of hosts

TODO: use tranalyzer, separate the flags, combine tranalyzer with cicflowmeter

TODO: should we consider separate solution for UDP and TCP?

TODO: uses classes for destination port (port outside ucsb)

TODO: Should I remove traffic related to the services at UCSB like the web server?

TODO: What is tenfold cross validation

TODO: Detecting malicoius behaviour based on user behaviour statistics and counting the number of malicious entities

TODO: Compute the training time with different models, compute the accuracy with different models

TODO: NOT dorpping some of the packet fileds like IHL and total length

TODO: Here we are

1.3 Contributions

add it here

2 BACKGROUND

(1) Talk about AQM and Bitag
(2) Explain LibreQoS
(3) Explain and reference Pinot and Netunicorn
(4) Explain how it helps: Contrilling network condition, single vantage point for data collection

3 METHODOLOGY

(1) Taking all the traffic to the box (gre tunnel)
(2) Routing the traffic through the shaper (veth) for upstream
(3) Performing NAT and dissecting NAT
(4) Routing unNATed traffic through shaper for downstream shaping
(5) Routing back traffic to the clients through gre tunnel

(1) Ethernet to namespace 1
(2) Namespace 1 to namespace 2 through shaper
(3) Namespace 2 to the NAT
(4) NAT to the Ethernet (second one)
(5) Reverse of all these steps

Counting the users behind a NAT is a subproblem for detecting the users behind the NAT. When we can classify the users, counting the users is just counting the number of clusters. Same for detecting the existence of NAT, when classification detects only one cluster, we can say there is no NAT in place or we can say there is a 1 to 1 mapping between IP addresses. Therefore, for the rest of paper, we only tackle user classification problem which encompasses the other two use cases.

There are two ways to tackle user problem. One method is giving the entire traffic and flows as input to the system and then classifying the users based on the entire flows/packets belonging to the same users. The other way is that comparing to flows one by one and deciding whether these two flows belong to the same user or not. We argue that the second approach is better in terms of required training dataset size. Imagine we captured data for 1 hours of n users, each have on average k flows. In the former case, we have only n samples in our dataset and each sample is labeled with a unique ID which would be the user ID; In the latter case, we would have the following number of samples in the dataset:

- Samples of flows belonging to the same user: $n * \binom{k}{2}$
- Samples of flows belonging to the different users: $\frac{n*(n-1)}{2} * k^2$

Even if we decide to have a balanced dataset and have the same number of samples from both, our dataset size would be almost $n*k^2$ which is much larger than n and gives better opportunity to the model to learn the patterns based on the same raw input data. Once we have realized which flows are from the same user and which flows are not, we can cluster the flows into different groups where each group represents a user.

3.1 Data preprocessing

We had access to UCSB gateway data where the user traffic was not NATted due to big range of IP addresses we have at UCSB. So, it was straightrait forwoard for us to capture huge dataset which is already labeled. We just needed to separate the users based on the internal IP address of the packets (either src or destiantion belong to a specific user). For this paper, we used pcapsplitter to separate all the flows from each other. Then, we used CICFlowMetere to get the data related to the flows like the number of packets, bytes, flow duration, inter arrival time etc. After that, we grouped the flows (both CICFlowMetere and the pcaps) based on the internal IP address of the packets.

One thing we did before identifying the flows was removing every flow with less than 3 packets. These short flows are not useful for our classification problem as they do not have enough information to be classified. Besides that, they are usually bots scanning the network or some other malicious activities or unsuccesful connections.

After that, we created another column called userLabel which is the internal IP address of the flows and remvoed the source/destination IP address columns. We also removed the columns which are not useful for our classification problem like checksum and IP version. After that, we chose the number of flows per user based on the mean and median of the number of flows per user. We sampled 20 flows per user and then combined them with each other and with the flows from other users. The two flows belong to the same user is labelled as one, and it would be zero otherwise. We used weighted random selection to select the flows from other users.

We considered 3 different representations for the flows:

- Using cicFlowMeter only
- Using npring only (first 5 packets of flow)
- Using cicFlowMeter and several first packets of the flow
- Maybe: packet level information, flow level information and user behvior

3.2 TDUserMeter

We developed a tool call True Detective User Meter (TDUserMeter) which analyases the user behavior based on the flows. This tool takes all the CICFlowmeter output for all of the flows of the user in a period of time (eg one hour) and calculates differnet features which would help distinguish the users from each other. The features we used are as follows:

Feature	Description
TotalFlowDuration	Total duration of the flows
MinFlowDuration	Minimum duration of the flows
MaxFlowDuration	Maximum duration of the flows
AvgFlowDuration	Average duration of the flows
StdFlowDuration	Standard deviation of the duration of the flows
totalPackets	Total number of packets in the flows
minPackets	Minimum number of packets in the flows
maxPackets	Maximum number of packets in the flows
avgPackets	Average number of packets in the flows
stdPackets	Standard deviation of the number of packets in the flows
totalBwdPackets	Total number of packets in the backward direction of the flows
minBwdPackets	Minimum number of packets in the backward direction of the flows
maxBwdPackets	Maximum number of packets in the backward direction of the flows
avgBwdPackets	Average number of packets in the backward direction of the flows
stdBwdPackets	Standard deviation of the number of packets in the backward direction of the flows
totalFwdBytes	Total number of bytes in the forward direction of the flows
minFwdBytes	Minimum number of bytes in the forward direction of the flows
maxFwdBytes	Maximum number of bytes in the forward direction of the flows
avgFwdBytes	Average number of bytes in the forward direction of the flows
stdFwdBytes	Standard deviation of the number of bytes in the forward direction of the flows
totalBwdBytes	Total number of bytes in the backward direction of the flows
minBwdBytes	Minimum number of bytes in the backward direction of the flows
maxBwdBytes	Maximum number of bytes in the backward direction of the flows
avgBwdBytes	Average number of bytes in the backward direction of the flows
stdBwdBytes	Standard deviation of the number of bytes in the backward direction of the flows
totalBytes	Total number of bytes in the flows
distinctSrcPorts	Number of distinct source ports in the flows
distinctDstPorts	Number of distinct destination ports in the flows
distinctDstIP	Number of distinct destination IP addresses in the flows
totalFlows	Total number of flows
minFwdPacketLen	Minimum length of the packets in the forward direction of the flows
maxFwdPacketLen	Maximum length of the packets in the forward direction of the flows
avgFwdPacketLen	Average length of the packets in the forward direction of the flows
stdFwdPacketLen	Standard deviation of the length of the packets in the forward direction of the flows
minBwdPacketLen	Minimum length of the packets in the backward direction of the flows
maxBwdPacketLen	Maximum length of the packets in the backward direction of the flows
avgBwdPacketLen	Average length of the packets in the backward direction of the flows
stdBwdPacketLen	Standard deviation of the length of the packets in the backward direction of the flows
minDownUpRatio	Minimum down/up ratio of the flows
maxDownUpRatio	Maximum down/up ratio of the flows
avgDownUpRatio	Average down/up ratio of the flows
stdDownUpRatio	Standard deviation of the down/up ratio of the flows
minFlowIAT	Minimum inter arrival time of the flows
maxFlowIAT	Maximum inter arrival time of the flows
avgFlowIAT	Average inter arrival time of the flows
stdFlowIAT	Standard deviation of the inter arrival time of the flows
minIdleTime	Minimum idle time of the flows
maxIdleTime	Maximum idle time of the flows
totalIdleTime	Total idle time of the flows
stdIdleTime	Standard deviation of the idle time of the flows
avgIdleTime	Average idle time of the flows
totalTCPFlows	Total number of TCP flows
totalUDPFlows	Total number of UDP flows
UserIP	Internal IP address of the user

Table 1: *Features extracted by TDUserMeter*

4 NETREPLICA

Changing section name and position in future:

- netreplica architecture
- controlling the bandwidth
- realistic background traffic
- nodes with different distance
- controlling the bottleneck in access link

In order to create realistic network conditions, we need to be able to control the bandwidth and the background traffic plus running controlled applications. For traffic shaping, we use the LibreQoS open-source framework. LibreQoS is employed by many ISPs to control and monitor their networks, manage subscription plans, and provide optimal internet services to customers. It is scalable, having been tested with over 1000 users and 25+ Gbps, and is known for its low overhead. LibreQoS performs shaping and policing by relying on Linux's TC command and utilizing CAKE/FQ-CoDel active queue management. With LibreQoS, you can set minimum and maximum download/upload bandwidth for individual users or subnets in the network. We utilize this methodology to effectively shape and manage network traffic

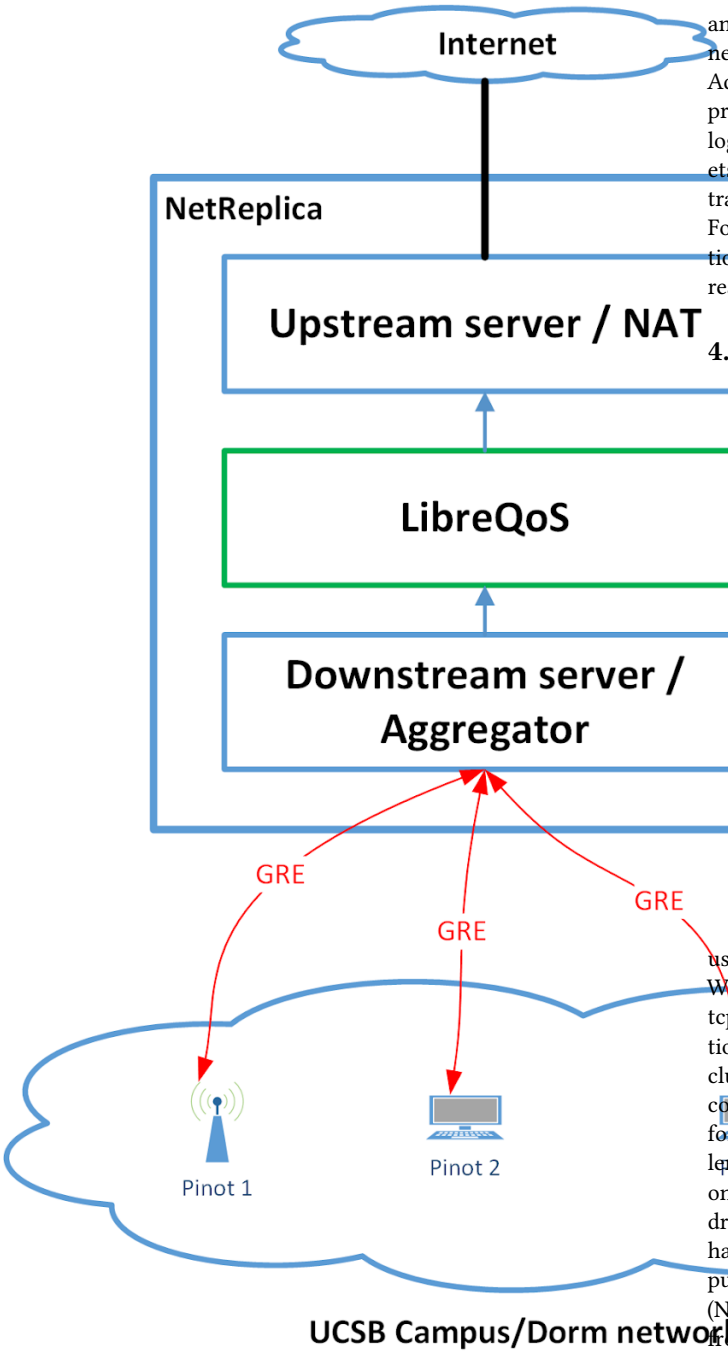

Figure 1: *netReplica Architecture*

Controlling available bandwidth alone does not accurately replicate real network conditions. Each hop from source to destination encounters background traffic from other users sharing network resources. For instance, in a home network, family members using the internet simultaneously can limit bandwidth and increase latency. The same applies to enterprise networks and ISPs, where multiple users access services concurrently, affecting application behavior and packet traces. Realistic background traffic is crucial for accurate network condition simulation. In applications like video streaming, Adaptive Bitrate (ABR) algorithms may choose lower bitrates to prevent rebuffering under heavy network load. Even if application logic remains unchanged, packet traces are still affected; some packets experience more delay, interarrival times and the number of retransmissions may vary, and other metrics can also be impacted. For these reasons, a realistic platform controlling network conditions must incorporate authentic background traffic that mimics real-world scenarios.

4.1 Existing Approaches

(1) No Background Traffic: Some studies neglect background traffic, resulting in datasets that fail to capture realistic traces and experiments.
(2) Steady Stream of Flows: Some studies use tools like iperf to create a steady stream of background traffic with specific throughput. This method fails to represent the bursty nature of real network traffic.
(3) Probabilistic Models: Using Gaussian, Laplacian, or other distributions to generate bursty background traffic is an improvement. However, these models do not reflect real user behavior, as the bursts are not based on actual network traffic distributions.
(4) Trace-based Methods: Some approaches derive traffic distribution from input traces captured from real users using the internet and attempt to mimic it. While better, these methods often rely on simulation environments like MahiMahi, which have their own limitations. These limitations are discussed in subsequent sections.

To create realistic background traffic that accurately mimics real user behavior, we use UCSB network traffic as background traffic. We select a subset of the captured traffic and replay it using the tcpreplay tool in our system while running the controlled application. We first captured traffic from the UCSB campus network, including dormitories, at the gateway of UCSB AS for 15 minutes. We collected MAC, IP, and transport layer headers, truncating payloads for privacy and performance while retaining the original packet length. Using PacketSplitter, we separated the captured data based on flows identified by 5-tuples and grouped them by internal IP address. Given that UCSB is one of the early hubs of the internet and has a large block of public IPv4 addresses, most users have unique public addresses, and there is limited network address translation (NAT) in place. Next, we used CICFlowMeter to extract features from each flow pcap. Our tool, UserActivityAnalyzer (UAA), then analyzed user behavior based on CICFlowMeter output for all flows associated with each user. The extracted metrics included total forward/backward bytes, throughput, number of unique ports used, and distribution of active and idle times, among others.

4.2 Experiment Design

We are considering 3 different scenarios to evaluate the performance of this system using netReplica.

- Scenario 1: Wired vs. Wireless nodes
- Scenario 2: Nodes with different distances to the bottleneck

- Scenario 3: Limiting the bandwidth vs. not limiting the bandwidth
- Scenario 4: With/without background traffic
- scenario 5: Low burst background traffic vs. High burst background traffic
- scenario 6: Creating congestion in the access link
- scenario 7: Utilizing different AQM algorithms at the bottleneck link
- scenario 8: Separating shaping bottleneck and background traffic bottleneck

5 MEASUREMENT STUDY

We conduct a study where we characterize the location of the bottleneck link as measured by popular speed test tools. We focus on two such tools, namely Ookla Speed test and M-Lab Network Diagnostic Test (NDT). In addition, we focus on the question whether the bottleneck link is within the last-mile ISP or upstream. We use the Netrics platform for these measurements.

Data Collection: Describe Netrics deployment, including number of tests, time period of the study, number of vantage points.

Testing setup. Why we are measuring from the client side. What are the implications.

5.1 Results

The data reveals interesting differences between the two tools. The last-mile ISP is the bottleneck for xx% tests in Ookla, while this number is yy% in MLab. Further dissecting them by the speed tests, the difference between the two tools increases.

6 RELATED WORK

6.1 Identifcation and Counting of Hosts Behind NAT Using Machine Learning [5]

In this, we use ICS data set to train our model which contains traffic flow from 1116 hosts. We identify each host with the help of source IP address and mark them as a unique class. Once classes are assigned to each fow, we train our model with the help of ICS data set. After training our data model, we have used diferent data sets which contains NATted trafc flow from multiple hosts to test our trained model. Our trained model then classifes each fow to a class with the help of all the selected features. Unique classes are then counted to count number of hosts.

How long the duraiong of ICS dataset? is 400MB enough? It had 460K flows, not natted

The second dataset, MTA: 143 hosts, natted, and non natted The third dataset, their own, 73 PC hosts 6 hours, natted and non natted, only web, running a script The test

The paper relies heavily on reference 17 for choice of flow analayzer, choice of filtering method

Much different featurs than us, we are getting mostly the interarrival time and flow based stats while the paper gets mostliy thibngs related to window size and ACks. ALso, they only not consider src port, but destination port is considered.

Do they consider UDP packets? Their most important features are tcp

Why not considering all the features and then eliminating shortcuts? They also had to reduce the number of features and it improved the preformance –> maybe they did not have large enough dataset –> their dataset size is equal to number of flows but ours can be as big is n chose 2 (half of n squared): They use multiclass calssifier

It redifines some ML stuff to just fill out the paper

They got 98 percent accuracy when test and train performed on ICS dataset, it is a shortcut though, accuracy dropped to 89 on lab dataset

IMPORTANT: Tranalayser considers the IAT

6.2 Identifying NAT Devices to Detect Shadow IT: A Machine Learning Approach [3]

6.3 A Generalizable Machine Learning Model for NAT Detection [4]

Based on their previous work, it is just detecting the existences of NAT, not complete set of features, users transfer learning, not a good dataset

6.4 Exploring nat detection and host identification using machine learning [2]

check this paper: Integrating machine learning with off-the-shelf traffic flow features for http/https traffic classification,

Also based on this paper, tranalyzer had the best preformance

6.5 How Polynomial Regression Improves DeNATing [1]

The refrences one to 9: check them all

The main ID of using IPID sucks: One is because of concurrent connections, it weould not get increamented but it would increase a lot in time, the other thing is DNS resovler in the network behind NAT and also DNS caching

It is TCP/IP based, what about UDP

Identifying packets belong to the same flow using TCP timestamp: Isn't it already solved by 4 tuples using pcapsplitter or cicflow?

Assumption about IP IP assignmet by increameing it for each packet either per flow or globally

They use TTL that can be a shortcut, but what about different devices use differnt Initial TTL: This can be a good info

Check the TCP window Scale size

OS fingerprinting using TCP/IP header fileds

Communication with speficif domains for OS fingerprinting (ref 15)

check netflow records

Check Timestamp filed and the two studies based on timestamp

In old versions of Windows OS, the IP-ID field was implemented as a simple counter. Newer versions use separate counters per destination address. IOS assigns a random number to IP-ID, and some versions of Linux always set it to zero.

The IP-ID method requires long time of catpuring data cause for creating sequence I guess they need more than several DNS requests. Also, how precies is that?

7 DISCUSSION

Discuss the limitations and future work here.

Explaining errors

Generalizability of methodology to different tests and device platforms

Generalizability of the measurement study

8 CONCLUSION

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES

[1] Ari Adler, Lior Bass, Yuval Elovici, and Rami Puzis. 2023. How Polynomial Regression Improves DeNATing. IEEE Transactions on Network and Service Management, IEEE.

[2] Ali Safari Khatouni, Lan Zhang, Khurram Aziz, Ibrahim Zincir, and Nur Zincir-Heywood. 2019. Exploring NAT Detection and Host Identification Using Machine Learning. Presented at the 2019 15th International Conference on Network and Service Management (CNSM), IEEE. , 8 pages.

[3] Reem Nassar, Imad Elhajj, Ayman Kayssi, and Samer Salam. 2021. Identifying NAT Devices to Detect Shadow IT: A Machine Learning Approach. Presented at the 2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA), IEEE. , 7 pages.

[4] Reem Nassar, Imad H Elhajj, Ayman Kayssi, and Samer Salam. 2023. A Generalizable Machine Learning Model for NAT Detection. Presented at the 2023 International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS), IEEE. , 44–49 pages.

[5] Sanjeev Shukla and Himanshu Gupta. 2022. Identification and Counting of Hosts Behind NAT Using Machine Learning. SN Computer Science, Volume 3, Issue 2, Page 126, Springer.