POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

"Comparative Performance and Resource Utilization Analysis of Captive Portal Solutions"

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Author: **Alessandro Gabbini**

Student ID: 10671352
Advisor: Prof. Gianni Antichi
Co-advisors: Alberto Pollastro
Academic Year: 2023-2024

Abstract

Captive portals are used by both private and public entities to manage user authentication and to collect data, offering a Wi-Fi service. However, they introduce computational costs that impact both system performance and user experience. This thesis examines the performance of two implementation solutions: CoovaChilli, an open-source software, and a proprietary alternative developed by the Italian company MobiMESH. The goal is to measure the increase in CPU utilization compared to a scenario without a captive portal, as well as to compare the two implementations. The tests separately evaluate the impact of two different factors: consumed bandwidth and number of concurrent users. The analysis was conducted in a controlled test environment with three machines connected via Ethernet: Client, Router, Server. Three scenarios were analyzed: Router without captive portal (baseline), Router with CoovaChilli, Router with enterprise software. Traffic was generated with iperf3, CPU utilization with mpstat, htop, perf. Virtual network interfaces and MACVLANs with network namespace were used to emulate simultaneous users. The results show the prevailing impact of bandwidth over the number of users. It is observed that CPU utilization increases almost linearly with bandwidth up to 800 Mbps, and then shows a non-proportional trend. The introduction of a captive portal introduces significant computational costs, more than doubling CPU utilization compared to the baseline scenario. Comparison of the two solutions reveals significant differences only at a low number of users, with CoovaChilli performing slightly better. As the number of users increases, the difference between the two implementations becomes irrelevant. The study has scalability limitations, especially in increasing concurrent users beyond a certain threshold. However, the results obtained offer concrete and useful data for evaluating the performance of captive portal software in different operational scenarios.

Keywords: Captive Portal, Performance Analysis, CPU Utilization, Bandwidth Limitations, CoovaChilli.

Abstract in lingua italiana

I Captive Portal sono utilizzati sia da aziende sia da enti pubblici per gestire l'autenticazione degli utenti e raccogliere dati, offrendo un servizio di Wi-Fi. Tuttavia, introducono costi computazionali che impattano sia le performance del sistema sia l'esperienza utente. Questa tesi esamina le prestazioni di due soluzioni implementative: CoovaChilli, un software open-source, e un'alternativa proprietaria sviluppata dall'azienda italiana MobiMESH . L'obiettivo è misurare l'aumento dell'utilizzo della CPU rispetto a uno scenario senza captive portal, oltre a confrontare le due implementazioni. I test valutano separatamente l'impatto di due fattori differenti: la banda consumata e il numero di utenti simultanei. L'analisi è stata condotta in un ambiente di test controllato con tre macchine collegate via Ethernet: Client, Router, Server. Sono stati analizzati tre scenari: Router senza captive portal (riferimento), Router con CoovaChilli, Router con il software aziendale. Il traffico è stato generato con iperf3, l'utilizzo della CPU con mpstat, htop, perf. Per emulare più utenti simultanei sono state utilizzate interfacce di rete virtuali e MACVLAN con network namespace. I risultati mostrano l'impatto prevalente della banda rispetto al numero di utenti. Si osserva che l'utilizzo della CPU cresce quasi linearmente con la banda fino a 800 Mbps, per poi mostrare un andamento non proporzionale. L'introduzione di un captive portal introduce costi computazionali significativi, più che raddoppiando l'utilizzo della CPU rispetto allo scenario base. Il confronto tra le due soluzioni rivela differenze significative solamente ad un basso numero di utenti, con prestazioni leggermente migliori di CoovaChilli. Al crescere del numero di utenti, la differenza tra le due implementazioni diventa irrilevante. Lo studio presenta limitazioni di scalabilità, soprattutto nell'aumento di utenti simultanei oltre una certa soglia. Tuttavia, i risultati ottenuti offrono dati concreti e utili per la valutazione delle performance dei software di captive portal in diversi scenari operativi.

Parole Chiave: Captive Portal, Analisi di Performance, Utilizzo della CPU, Limitazioni di Banda, CoovaChilli.

Contents

1 | Introduction

1.1. Goal of the Thesis

Captive portals play a crucial role in managing access to a network, especially in contexts where user authentication and traffic control (speed limits for example) are required. However, these solutions introduce a significant computational cost that impacts both the resources of the system running the service and the user experience. The goal of this thesis is to quantify and analyze the computational cost introduced by the use of a captive portal by answering the following questions:

- **Impact of Bandwidth vs. Number of Clients:** Does CPU usage scale mainly with the number of active users, or is bandwidth consumption the dominant factor?
- **Overhead of Captive Portals:** How much additional CPU usage does a captive portal introduce with respect to a baseline scenario without a captive portal enabled (a simple traffic forwarder)?
- **Comparison between the two solutions:** Are there notable efficiency differences between CoovaChilli and the proprietary solution?

To achieve this, three distinct scenarios are analysed:

1. Baseline measurement on a clean machine, without specific software running, to establish a reference for system resource utilization.
2. Analysis of an open-source solution, CoovaChilli, to assess the computational impact of a widely used captive portal.
3. Evaluation of the company's proprietary software that assesses whether it performs better or worse than CoovaChilli in terms of efficiency and CPU utilization, eventually setting the basis for future optimizations.

1.2. Brief Introduction to the Software Analyzed (Captive Portal Gateway)

Captive portal software provides network authentication mechanisms that restrict connected users from accessing the Internet until they complete a login or authentication procedure. These systems are often used in environments where a Wi-Fi service is offered, such as hotels, hospitals, and airports, but also in corporate settings for security reasons. In addition to verifying authentication, captive portals allow for data and metrics collection of connected users, as well as ensuring security policies, such as restricting browsing to only certain sites or blocking specific ones. Typically, when an unauthenticated user connects to a captive portal network, any attempt to access the Internet is intercepted and redirected to a login page, where credentials (usually username and password) are checked, or another verification method is used. Once the authentication process is completed, access to the Internet is granted, along with any limitations decided by the service provider. Captive portals operate by integrating multiple components to handle authentication, session management, and traffic control. The main functional elements include:

- **Authentication System**: Verifies user credentials via local databases, RADIUS servers, or external identity providers.
- **Network Redirection & Interception**: Captures HTTP(S) requests from unauthenticated users and redirects them to the login page.
- **Traffic Management & Firewall Rules**: Enforces policies to allow, block, or limit access based on authentication status and network policies.
- **Session Handling**: Manages user sessions, ensuring that authenticated users retain access while enforcing session timeouts and restrictions.

From a performance perspective, captive portals introduce additional processing overhead, because they intercept, authenticate, and manage user traffic dynamically. This introduces potential bottlenecks, particularly in high-traffic environments such as airports, where resource efficiency is critical. Understanding how the different implementations handle CPU workload distribution is essential for evaluating their scalability and eventually obtaining better performance.

1.3. Thesis Scope & Methodology

The performance analysis is conducted in a controlled network environment, where the two metrics, bandwidth and number of users, are incrementally increased, observing how CPU utilization scales.

The study focuses primarily on CPU utilization trends, trying to identify any trends, and comparing how two different deployment solutions impact this metric, under different network traffic conditions. The captive portal world contains multiple additional aspects, such as security, user experience, network policies, data profiling, however, these areas are beyond the scope of this of thesis.

Several tools were used to perform the various tests. For generating network traffic iperf3, for monitoring system metrics htop and mpstat. Perf was also used to collect low-level performanca data, and flamegraphs were generated to make possible future more detailed analyses of the inner workings of the two solutions and the computational cost of individual functions. However, the main focus of the thesis remains on CPU utilization rather than low-level functional analysis.

To conduct this analysis, as said, a three-machine test setup has been designed, consisting of:

- A **Client** that generates network traffic in different scenarios involving a growing number of users and bandwidth usage.
- A **Router** running the captive portal software and handling authentication and traffic forwarding. CPU utilization is measured on this machine.
- A **Server** that provides internet access cascadely and acts as the destination for forwarded traffic, acting as if it hosts the required resources requested by the client(s).

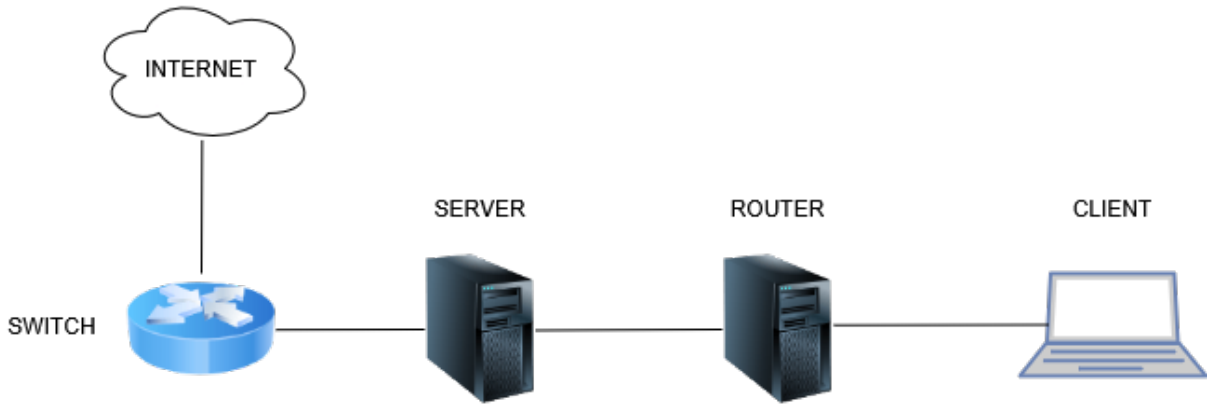

Figure 1.1: Architecture diagram made with draw.io [1]

The remainder of this thesis is structured as follows:

- **Chapter 2** reviews related works and highlights the lack of performance studies on captive portals, stating how this thesis work contributes to the field.
- **Chapter 3** describes the architecture of the two captive portal solutions: CoovaChilli and company's software, providing also a brief comparative summary.
- **Chapter 4** describes the tools and methodologies used for traffic generation and performance measurement, explaining also how the CPU usage was concretely calculated.
- **Chapter 5** contains the core of this work, the results analysis. CPU utilization trends under different bandwidth limits and number of clients are described. It details also the experimental setup and the technological solutions involved.
- **Chapter 6** summarizes the key takeaways and presents the study's limitations.
- **Chapter 7** concludes the thesis with a brief recap and suggests possible future works and directions for more in-depth investigation.

2 | Related Works

In any scientific or technical study, an essential preliminary step is to research the current state of the art and any preexisting studies. Doing so is not only useful to avoid re-executing work that has already been done, but also to clearly identify what the shortcomings and possible limitations of such studies are, as well as being necessary to be able to properly justify the relevance of the new contributions. There is a variety of literature on captive portals for network security and access control contexts, mainly regarding authentication mechanisms, security vulnerabilities, and user experience. However, as discussed more in the next section, there is very little literature regarding performance analysis. This lack underscores the need for this thesis in providing empirical data, assessing how much overhead is introduced by a captive portal, and offering insights or basis for possible optimizations.

2.1. Lack of Existing Performance Studies on Captive Portals

As mentioned, despite the enormous prevalence of captive portals, there is a notable lack of publicly available studies regarding the performance aspect. Existing research focuses primarily on security and authentication issues, with little if any emphasis on the impact these mechanisms bring in terms of performance. Related research in near fields, such as network authentication systems [2], provide some insights into possible bottlenecks in controlled environments, demonstrating how authentication and security mechanisms come with significant computational overhead. However, these studies do not specifically analyze captive portals, leaving as said a research gap.

2.2. How This Thesis Contributes to the Research Gap

Since the literature of direct studies on captive portal performance is very sparse, this thesis represents one of the first empirical analyses of CPU utilization when such software is deployed, providing quantitative data that can serve as a basis for future optimizations and more in-depth analyses. The methodology used, as introduced earlier, involves using a combination of established analysis tools to evaluate system behavior under different conditions and in different scenarios. From a strictly practical point of view, this work provides valuable insights not only for the company of the proprietary solution, but also for the broad community using the open-source solution. Since a direct comparison between the two captive portals is presented, under the same testing conditions, the results can serve as a reference for both internal and external evaluations.

3 | Analysed Solutions

Understanding the architecture of a captive portal, even just at a high level, is critical to being able to analyze its performance. The internal design affects the latency of one of the main processes, the authentication of incoming traffic, as well as clearly other metrics such as CPU and memory utilization and network throughput. By examining both the architecture of CoovaChilli and the company's proprietary solution, structural differences that impact resource utilization and efficiency in handling requests from users can be identified for a more detailed analysis.

3.1. CoovaChilli

3.1.1. Justification for Choosing CoovaChilli

CoovaChilli is one of the most popular open-source captive portal software. It was chosen for this study because it represents one of the best alternatives to the proprietary solutions available, as well as having well-written documentation, a numerous community, and being still maintained. Other alternatives were initially considered, but were discarded because of higher complexity in setup and minor presence of supporting discussions on the Web.

3.1.2. User Connection and Traffic Flow

From an operation point of view, the flow of CoovaChilli can be summarized as follows:

1. **Initial Connection:** When a user connects to the network, CoovaChilli assigns an IP address using its built-in DHCP service.
2. **Redirection:** Any HTTP request from an unauthenticated user is intercepted and redirected to a login portal (Universal Access Mechanism - UAM).
3. **Authentication:** The user submits credentials, which CoovaChilli forwards to an external RADIUS server for the completion of the authentication process.
4. **Post-Authentication:** If authentication succeeds, the user is granted network

access, and network traffic is now forwarded accordingly to the decided policies.

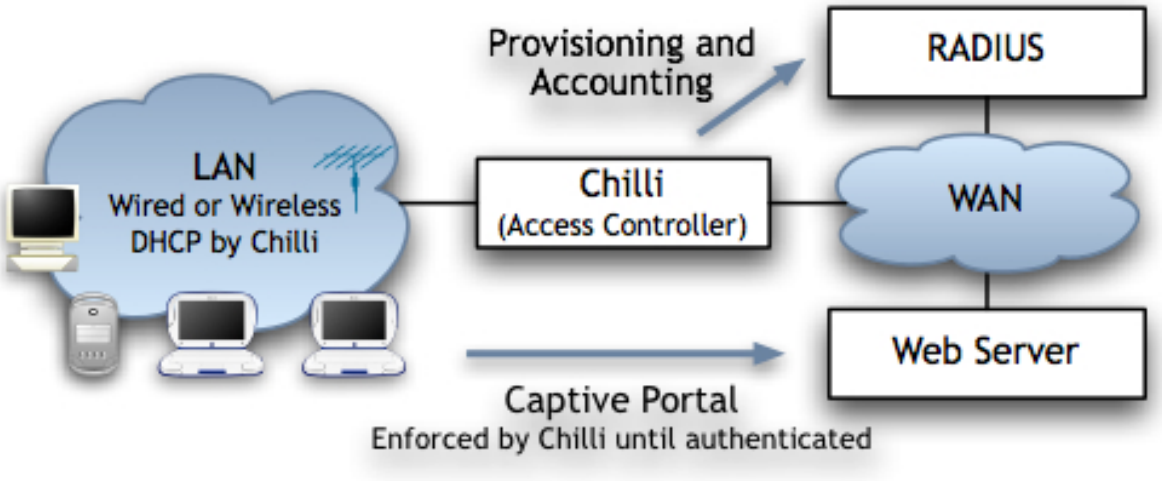

Figure 3.1: CoovaChilli network architecture. Source: OpenWrt Wiki [3]. Licensed under CC BY-SA 4.0.

3.1.3. Internal Architecture

The internal architecture of CoovaChilli can be summarized around the following key functional components:

- **Packet Interception:** CoovaChilli handles all network packets, distinguishing between authenticated and unauthenticated ones.
- **Session Management:** The software maintains session states for connected users, enforcing access control policies.
- **RADIUS Communication:** CoovaChilli interacts with the RADIUS server to process authentication requests and enforce access rules.

When started, CoovaChilli takes control of the internal network interface (facing the LAN) using a raw promiscuous socket and creates a virtual interface (tun or tap) via the vtun kernel module. This module allows CoovaChilli to move IP packets from the kernel to user space, enabling packet processing without requiring custom kernel modifications. The software then provides DHCP, ARP, and HTTP hijacking on the interface connected to the external network (WAN). When a client connects, it is initially restricted to a "walled garden", meaning it can only resolve DNS queries and access predefined websites until authentication is completed. Authentication occurs through one of two mechanisms:

1. **MAC-based authentication:** the device's MAC address is checked against an allowlist.
2. **Universal Access Method (UAM):** based on credentials verification, as explained before.

In the UAM process, when an unauthenticated client requests a webpage over HTTP (port 80), CoovaChilli intercepts the request and redirects it to a splash page, typically a classical username-password-authentication page. This portal presents a login form where user enters credentials. The credentials are then forwarded to an authentication server (e.g., RADIUS) for verification against the information stored in the backend database. Based on the response from the authentication server, the captive portal either: grants access, allowing full network connectivity or denies access, restricting the client to the walled garden or displaying an error message. [4].

For more in-depth details on CoovaChilli's architecture, its official documentation is referenced, as a full technical breakdown is beyond the scope of this thesis [3] [5] [6].

3.2. Company's Captive Portal

The company's captive portal solution follows the same functional model as CoovaChilli for handling user authentication and traffic management. However, its internal architecture differs in key areas, particularly in how it manages DHCP, DNS, traffic classification, and packet forwarding. The operational flow is the following:

1. **Initial Connection:** When a user connects to the network, the captive portal does not provide DHCP services directly but instead relies on an external DHCP server.
2. **Redirection:** Any HTTP request from an unauthenticated user is intercepted and redirected to the login page.
3. **Authentication:** The portal only handles HTTP redirection and classification, delegating authentication processing to an external RADIUS server.
4. **Post-Authentication:** Once authenticated, the user is granted normal network access, and traffic is forwarded accordingly to network policies.

The internal architecture differes from CoovaChilli's:

- **DHCP and DNS Services:** Unlike CoovaChilli, which provides integrated DHCP and DNS functionalities, the company's solution offloads these tasks to external applications.

- **Traffic Filtering:** The system uses netfilter and iptables for packet filtering.
- **Packet Forwarding:** Instead of utilizing TUN/TAP virtual interfaces like CoovaChilli, the company's CP uses nfqueue (a module of netfilter) to forward packets from kernel space to user space for classification.

These architectural choices can influence system performance. Especially the use of nfqueue instead of TUN/TAP can have different implications on packet processing speed and CPU overhead, which will be analyzed in the results analysis chapter. For more details regarding the login and authentication flow, a sequence diagram made with SequenceDiagram.org is provided [7].

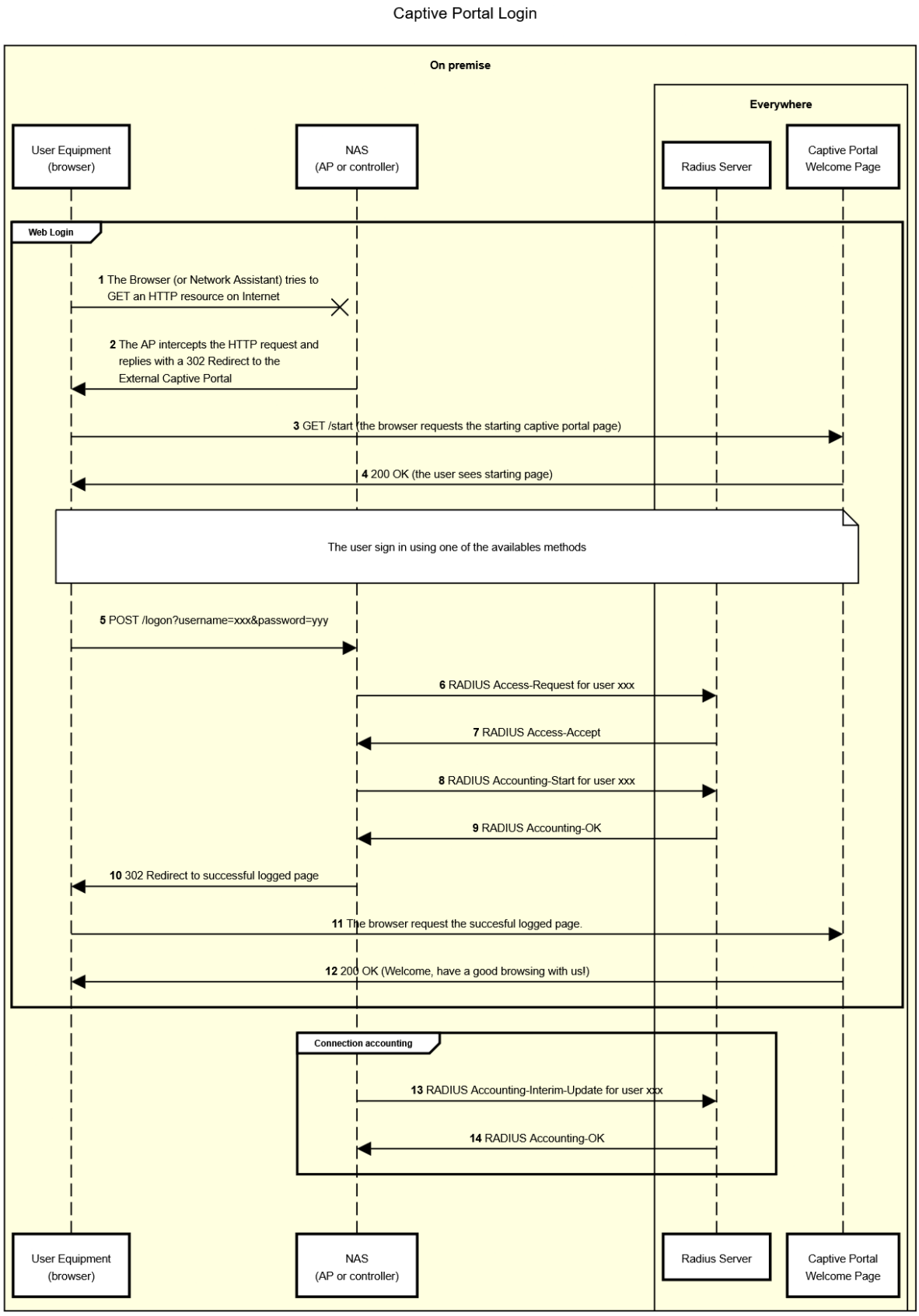

Figure 3.2: Sequence Diagram depicting the login-to-authentication flow.

3.3. Comparative Summary

In summary, both solutions have very similar high-level operational flow; what changes are the architectural choices. CoovaChilli integrates DHCP and DNS resolver services internally, while the proprietary solution delegates both tasks to external applications. Traffic filtering is handled internally by CoovaChilli, while the company's captive-portal uses netfilter and iptables. Finally, with regard to traffic forwarding, CoovaChilli bases its solution on the creation of a virtual network interface, unlike the proprietary solution that leverages the Linux nfqueue module for redirecting packets from kernel-space to user-space. These differences can affect performance in terms of overhead on the CPU, latency, and scalability.

4 | Tools for Traffic Generation and Performance Measurement

4.1. Traffic Generation: iperf3

4.1.1. Introduction to iperf3

Since the thesis work aims to analyse the performances of a software that handles network traffic, it is required a reliable way to generate and measure traffic under controlled conditions. For this purpose, iperf3 has been chosen, since due to its establishment as one of the most widely used tools for customization traffic generation, benchmarking and testing network links. It allows users to generate network traffic, making it particularly useful for network research, infrastructure tuning, and also troubleshooting in general. The versatility of iperf3 makes it ideal for a variety of use cases. One of its primary applications is measuring bandwidth, helping determine the maximum achievable throughput between two network endpoints. It is also frequently used for stress testing, where large volumes of traffic are generated to analyze how a network or system behaves under high loads. Additionally, it supports both TCP and UDP streams, enabling a deeper evaluation of network conditions, from detecting congestion and retransmissions in TCP to assessing jitter and packet loss.

Several tools exist for traffic generation and network performance evaluation, yet iperf3 has been chosen due to its simplicity and efficiency. As an open-source project, it is widely supported and actively maintained, ensuring compatibility with modern networking environments. Unlike more advanced benchmarking tools that require complex setup and configuration, iperf3 provides a lightweight powerful solution, accessible via simple command-line interface. Another key advantage is its ability to generate comprehensive performance reports in real-time, allowing users to monitor throughput, retransmissions, and latency fluctuations without the need for external monitoring tools. Since iperf3 is platform-independent, it can be easily deployed on Linux, Windows, and macOS, simplifying the setup on the client machine and allowing also cross-platform experiments.

Given the nature of the study, which involves analyzing the impact of a captive portal on CPU utilization, with different bandwidth values and different number of TCP streams, having a tool that efficiently generates network traffic like iperf3 does was essential.

4.1.2. How iperf3 works

The fundamental operation of iperf3 is based on a client-server model, where one machine acts as the server, passively listening for incoming traffic on a specified port, while the other functions as the client, actively sending traffic to the server for measurement, with specified parameters. This architecture allows for precise control over network conditions, making it possible to measure not only one-way throughput but also bidirectional performance when running tests in reverse mode. One of the reasons iperf3 is so widely adopted is its ability to support both TCP and UDP. TCP-based tests are particularly useful for evaluating bandwidth limitations and congestion control mechanisms, as TCP automatically adjusts its sending rate based on network conditions. On the other hand, UDP tests provide valuable insights into packet loss, jitter, and latency, which are critical for applications where timely delivery is more important than guaranteed packet arrival, such as video conferencing or online gaming. When running an iperf3 test, the tool collects a variety of performance metrics. The throughput, typically expressed in Mbps or Gbps, represents the rate at which data is successfully transmitted over the network. For UDP tests, additional measurements include packet loss percentage and jitter, which help assess network stability. In TCP mode, iperf3 can also tracks retransmissions, providing insight into how often packets had to be resent due to network congestion or errors.

4.1.3. Key Configuration Parameters

To ensure flexibility in network testing, iperf3 provides several key options that allow users to fine-tune traffic generation. The most relevant parameters for this experiment include protocol selection (TCP/UDP), bandwidth control (-b), test duration (-t) and port specification (-p), among other additional options. By default, iperf3 runs TCP tests, UDP mode can be activated using the -u flag. For an exhaustive breakdown of iperf3 options and features, refer to the official documentation at [8].

For example, the following commands make the server listening on port 5201 and make the client generating a TCP stream at 1 Gbps to the server on port 5201, on the loopback interface (127.0.0.1) for 10s.

```
boga411@PC-BOGA:~$ iperf3 -s -p 5201
-----------------------------------------------------------
Server listening on 5201 (test #1)
-----------------------------------------------------------
Accepted connection from 127.0.0.1, port 37266
[  5] local 127.0.0.1 port 5201 connected to 127.0.0.1 port 37278
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec   119 MBytes   999 Mbits/sec
[  5]   1.00-2.01   sec   120 MBytes  1.00 Gbits/sec
[  5]   2.01-3.02   sec   120 MBytes  1000 Mbits/sec
[  5]   3.02-4.01   sec   118 MBytes  1000 Mbits/sec
[  5]   4.01-5.00   sec   118 MBytes  1000 Mbits/sec
[  5]   5.00-6.04   sec   124 MBytes  1.00 Gbits/sec
[  5]   6.04-7.00   sec   115 MBytes  1000 Mbits/sec
[  5]   7.00-8.00   sec   119 MBytes  1.00 Gbits/sec
[  5]   8.00-9.00   sec   119 MBytes  1.00 Gbits/sec
[  5]   9.00-10.00  sec   119 MBytes   999 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.00  sec  1.16 GBytes  1000 Mbits/sec                  receiver
```

Figure 4.1: iperf3 command for server mode and test output

```
boga411@PC-BOGA:~$ iperf3 -c 127.0.0.1 -p 5201 -b 1G -t 10
Connecting to host 127.0.0.1, port 5201
[  5] local 127.0.0.1 port 37278 connected to 127.0.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec   119 MBytes   999 Mbits/sec    0    639 KBytes
[  5]   1.00-2.00   sec   119 MBytes  1.00 Gbits/sec    0    639 KBytes
[  5]   2.00-3.00   sec   119 MBytes  1.00 Gbits/sec    0    639 KBytes
[  5]   3.00-4.00   sec   119 MBytes   999 Mbits/sec    0    639 KBytes
[  5]   4.00-5.00   sec   119 MBytes  1.00 Gbits/sec    0    639 KBytes
[  5]   5.00-6.00   sec   119 MBytes  1.00 Gbits/sec    0    639 KBytes
[  5]   6.00-7.00   sec   119 MBytes   999 Mbits/sec    0    639 KBytes
[  5]   7.00-8.00   sec   119 MBytes  1.00 Gbits/sec    0    639 KBytes
[  5]   8.00-9.00   sec   119 MBytes  1.00 Gbits/sec    0    639 KBytes
[  5]   9.00-10.00  sec   119 MBytes   999 Mbits/sec    0    639 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  1.16 GBytes  1000 Mbits/sec    0             sender
[  5]   0.00-10.00  sec  1.16 GBytes  1000 Mbits/sec                  receiver

iperf Done.
```

Figure 4.2: iperf3 command for client mode and test output

4.2. Performance Measurement: perf and other Linux Tools

4.2.1. Introduction to Performance Monitoring

In any system that processes and forwards network traffic, understanding how resources are utilized is essential to evaluate efficiency, performance bottlenecks, and overall system behavior. In this experiment, the focus is on analyzing the CPU utilization of the machine forwarding the packets while handling different levels of traffic, to quantify its impact under different workloads, with and without the captive portal enabled.

The original intent of the performance analysis was to obtain a comprehensive view of system behavior using additional profiling and monitoring tools. In practice, this study primarily focused on CPU utilization trends, with additional profiling data collected for potential future investigations. The key aspects that were effectively analyzed include:

- **CPU utilization**: Measured across different scenarios to quantify the additional processing overhead introduced by captive portals.
- **Process-level behavior**: CPU consumption of specific processes was monitored to assess how much computational load the captive portal software imposed.
- **Flamegraph generation**: perf was used to collect low-level data and to generate flamegraphs for deeper profiling, however, these were not extensively analyzed in this thesis but remains available for potential future work.

The following sections describe the specific tools used, explaining their purpose, rationale for selection, and application within this study.

4.2.2. perf: Linux Profiling Tool

When analyzing system performance at a low level, it is essential to go beyond simple CPU and memory usage metrics. While tools like htop, mpstat and vmstat provide real-time monitoring, they do not reveal where CPU cycles are spent, which functions consume the most processing time, or whether performance bottlenecks exist at the kernel level. This is where perf, the Linux Performance Profiler, becomes invaluable. perf is a powerful, low-overhead profiling tool that provides detailed insights into CPU performance, cache behavior, system calls, and interrupts. Unlike general monitoring utilities, perf can break down performance metrics at the function level, allowing users to analyze not only which processes are consuming CPU time but also which functions within those processes

contribute most to resource consumption. In this experiment, perf was used to gather additional data on:

- **Which functions the CPU spends the more clock cycles on**, while forwarding packets.
- **How many system calls and interrupts are generated**, which can indicate kernel-level inefficiencies or bottlenecks.
- **Cache Misses and Memory Bottlenecks**: identifying whether poor memory access patterns are slowing down packet processing.
- **System Calls and Kernel Overhead**: helps determine if user-space processes spend too much time in kernel operations.
- **Interrupt Rate and Context Switching**: for monitoring how frequently the system switches between tasks, which can impact performance.

Different perf commands were used to collect performance data under different network loads. The most relevant ones include:

- perf record: for data collection
- perf report: for visualizing collected data

While perf provides detailed numerical data, visually analyzing function call stacks is often more effective. This is where Flame Graphs, developed by Brendan Gregg, become useful. Flame Graphs aggregate data collected with perf into an interactive visualization, making it easy to identify: which functions consume the most processing time; how execution flows through different system components; whether CPU time is spent in user-space applications or kernel functions. For flamegraphs generation, the following commands are used, available from Brendan Gregg github repository [9]:

```
# Run perf script and generate the flamegraph
perf script --kallsyms=/proc/kallsyms > out.perf
./stackcollapse-perf.pl out.perf > out.folded
./flamegraph.pl out.folded > "${OUTPUT_FILE}.svg"
```

Figure 4.3: Bash commands for flamegraph generation.

The final result is similar to the following image:

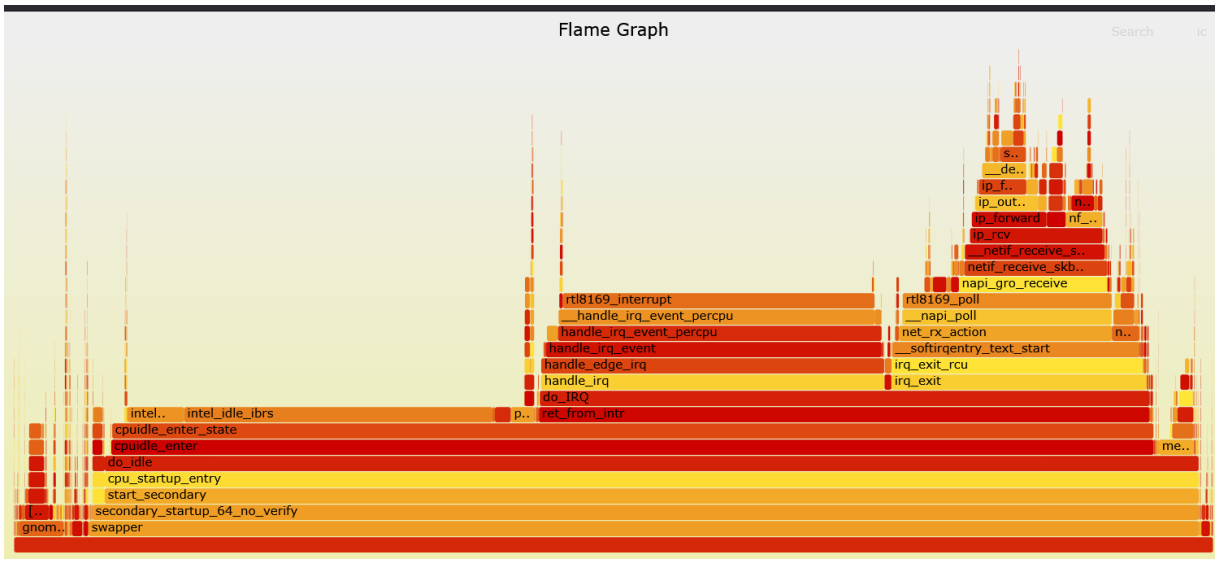

Figure 4.4: An example of a flamegraph.

It is important to note that flamegraphs are not just simple images, they are interactive files that can be further explored by opening directly with a web browser.

Additional information, including a more in-depth explaination of flamegraphs and other useful tools, is directly available from Brendan Gregg's website [10]

4.2.3. Linux System Monitoring Tools

While perf provides low-level profiling and function-level CPU analysis, it is also important to monitor system-wide performance in real time. To achieve this, various Linux system monitoring tools were used in this study, with a particular focus on htop and mpstat. These tools provide a broader overview of how CPU and system resources are utilized under different traffic conditions, giving a quicker and easier to read comparison between different scenarios. Each tool plays a distinct role in performance monitoring:

- **htop** was used to track process activity.
- **mpstat** provided detailed CPU load distribution across multiple cores, with a detailed break-down of cpu usaage.

htop: One of the most widely used real-time monitoring tools, htop, is an enhanced version of top, it offers a user-friendly interface with detailed process statistics. Unlike top, which displays system metrics in a fixed way, htop provides a dynamic and a customization view, making it easier to analyze system load in real time. It can be simply launched via command line with "htop". This command displays a real-time graphical representation of

CPU and memory usage, alongside a list of the running processes, their CPU and Memory consumption and execution states. For more details: [11].

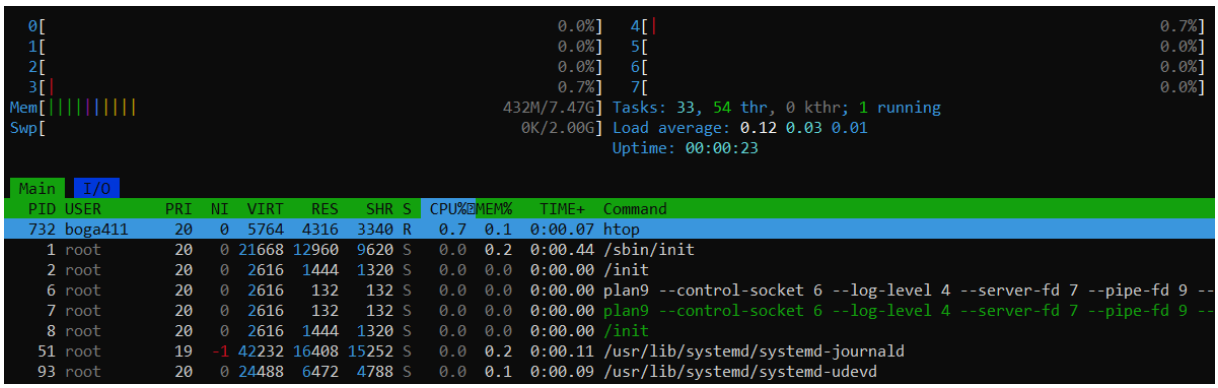

Figure 4.5: htop output

mpstat: It is part of the sysstat package and provides detailed per-core CPU usage statistics, making it ideal for analyzing how workload is distributed across multiple processors, with also a break-down of the CPU usage. For more details: [12].

To capture per-core CPU usage at 1-second intervals: “mpstat -P ALL 1”.

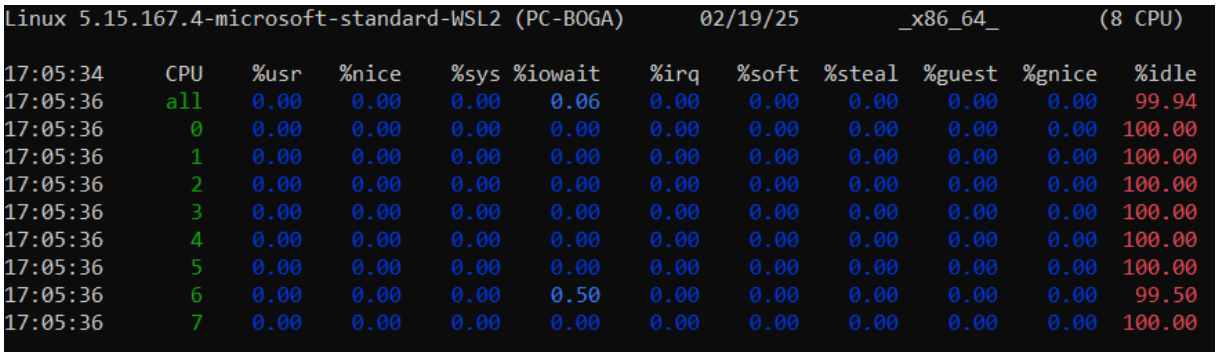

Figure 4.6: mpstat -P ALL 1 output

In addition to these tools, accurately measuring the actual bandwidth consumption during the experiments was crucial to ensure proper data normalization and fair comparisons. Given the need for minimal measurement overhead to avoid influencing the results, a lightweight approach was adopted. A custom Python script was developed to directly sample CPU utilization from /proc/stat and network bandwidth usage from /proc/net/dev/ at regular intervals. By leveraging these low-level system files, it was possible to extract real-time performance metrics without introducing computational overhead (although minimal), ensuring a more precise measurement.

4.3. System Performance by Brendan Gregg

When analyzing system performance, it is essential to follow a structured methodology to ensure accurate measurements and meaningful interpretations. System Performance: Enterprise and the Cloud by Brendan Gregg [13] is widely recognized as a foundational resource for performance analysis, offering methodologies and best practices for monitoring and optimizing system behavior, particularly in Linux-based environments. This study primarily focused on CPU utilization trends, but Gregg's work provided valuable insights into performance monitoring strategies. His layered approach to system analysis highlights the importance of combining multiple tools to gain a holistic and comprehensive view while carrying out a performance analysis. While tools such as perf and flamegraphs were used in this study, their full analytical potential was not deeply explored; however, Gregg's methodologies remain highly relevant for future work in identifying bottlenecks and inefficiencies within network-processing systems. By referencing System Performance, this study ensures that performance measurements align with established best practices and that the foundations for further investigation are well-structured.

5 | Results Analysis

In this chapter, firstly are explained the technological solutions adopted for trying replicating real-world conditions where clients attempt to reach resources on a server while a machine in the middle routes traffic, alongside with a detailed description of the experimental setup. Then the core of the thesis work, the test results, is presented.

5.1. Technological Solutions

5.1.1. Virtual Network Interface Card (vNIC)

In Linux-based environments, virtual network interfaces cards (vNICs) provide a way to create multiple logical network interfaces within a single physical machine, allowing the simulation of multiple independent clients without requiring additional hardware.

To achieve this, virtual Ethernet (veth) pairs are used. A veth pair consists of two connected interfaces—one is assigned an IP and used for traffic generation, while the other remains active but unused. This allows the system to simulate multiple distinct clients from a single machine [14].

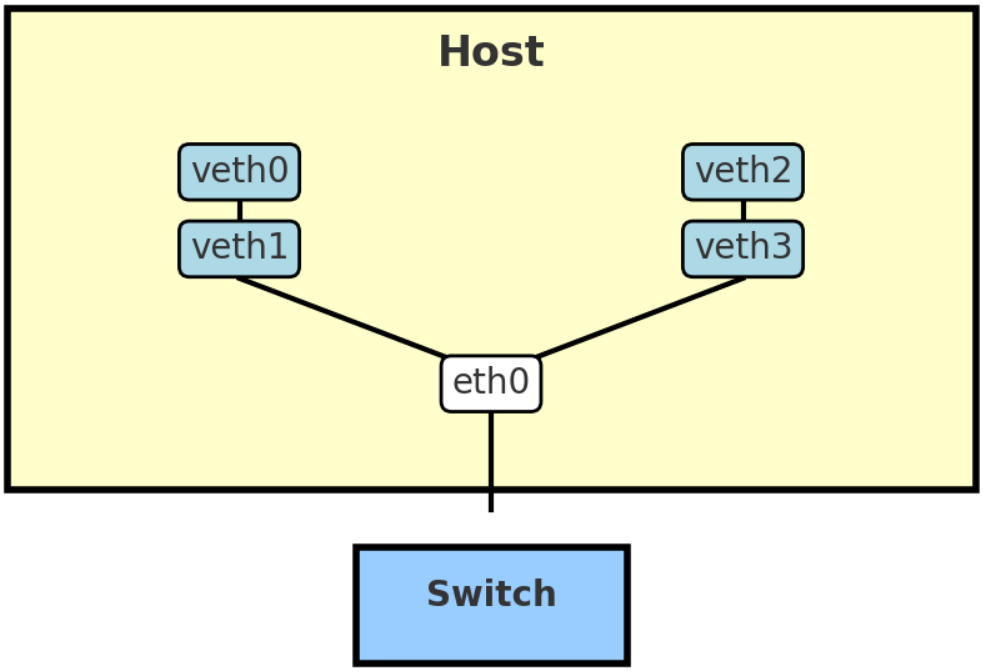

Figure 5.1: Diagram illustrating the use of veth pairs to simulate multiple clients on a single machine.

The following commands illustrate how veth pairs are created and assigned IPs:

```
# Create the virtual interface pair and assign the IP
ip link add "$interface_name" type veth peer name "$peer_name"
ip addr add "$new_ip/24" dev "$peer_name"

# Bring up the interfaces
ip link set "$interface_name" up
ip link set "$peer_name" up

# Delete conflicting routes
ip route del "$subnet" dev "$peer_name" proto kernel scope link src "$new_ip"
```

Figure 5.2: Snippet of the bash script that: adds a veth peer NIC, assigns a static IP to the vNIC, sets both NICs UP, removes the auto-generated conflicting route.

Each virtual interface is manually assigned a static IP within the same subnet as the ROUTER, ensuring that the SERVER perceives multiple distinct clients, even though they originate from the same physical machine. This method is a lightweight solution for

controlled experiments, avoiding the need for additional machines or complex networking configurations.

Next, we explore network namespaces and MacVLAN, which are used in scenarios where dynamic IP assignment via DHCP is required.

5.1.2. Namespace and MACVLAN

A Dynamic Host Configuration Protocol (DHCP) server assigns IP addresses to network clients, a fundamental mechanism used in all captive portal gateways [15]. Virtual NICs, as explained, rely on manually assigned IPs, making vNIC-based solutions impractical in captive portal environments, since a client connecting to a network with a captive portal has to receives dynamically an IP through DHCP. Therefore, a different technological solution is therefore needed. This solution is based on the combination of two Linux networking features: network namespaces (netns) and MACVLANs. In Linux, network namespaces provide isolated environments that allow multiple virtual networking instances to coexist independently within a single system. Each netns has its own network stack, including interfaces, routing tables, and firewall rules, effectively acting as a separate system from the main network [16] [17]. A MACVLAN is a virtual network interface that allows multiple virtual interfaces to exist on a single physical NIC while each having its own unique MAC address. This enables multiple simulated clients to receive individual DHCP leases and behave as distinct entities within the network [14].

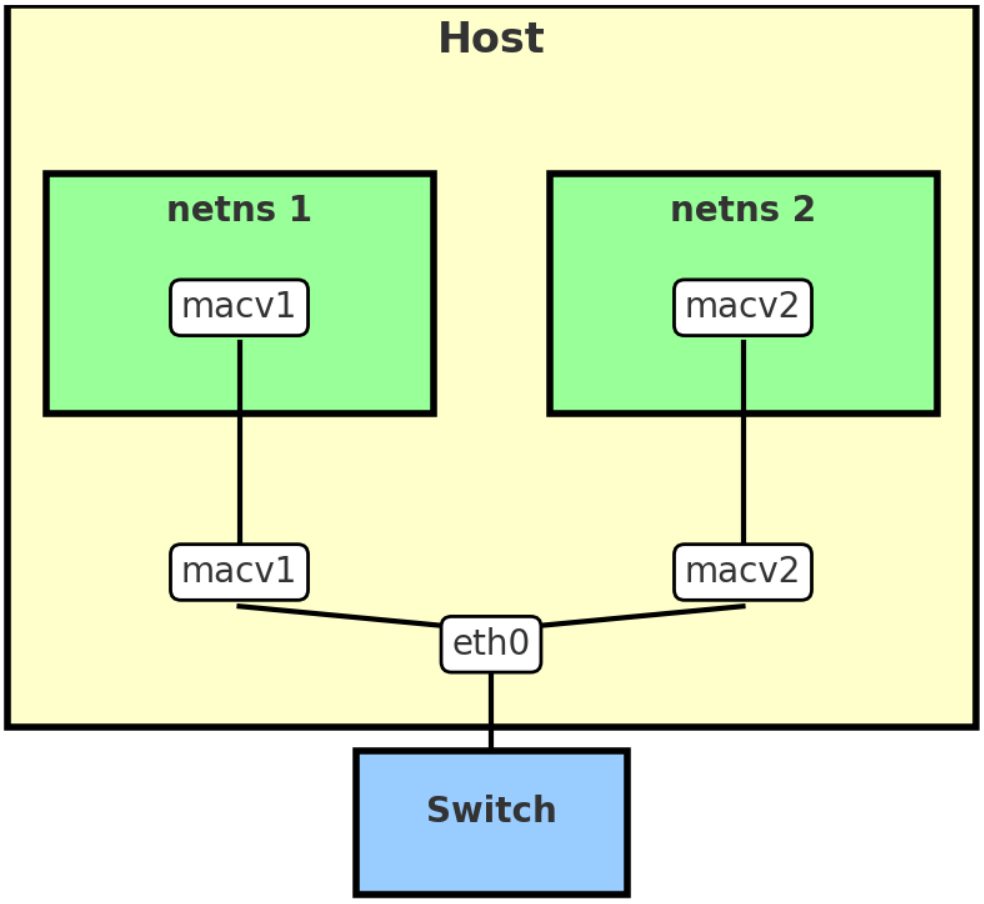

Figure 5.3: Diagram illustrating the use of network namespaces and MACVLAN interfaces to simulate multiple clients on a single machine.

Paring both namespaces and MACVLANs allow each simulated client to receive an IP address dynamically via DHCP. The commands used for setting up a namespace with MACVLAN and requesting a DHCP lease are the following:

```
# Create network namespace
ip netns add net$n

# Create macvlan and bridge it to the physical NIC
ip link add macvlan$n link enp2s0 type macvlan mode bridge

# Assign the macvlan to the namespace
ip link set macvlan$n netns net$n

# Request the IP through DHCP
ip netns exec net$n dhclient macvlan$n
```

Figure 5.4: Snippet of the bash script that: creates a network namespace, adds a MACVLAN interface in bridge mode, moves it to the namespace, requests a DHCP lease.

Each network namespace (netns), along with its MACVLAN interface, functions as an independent client that must authenticate before sending traffic. To automate authentication, a script launches a curl request to the captive portal gateway, handling authentication for each namespace dynamically. Additionally, other scripts are available to handle logout procedures and clean up network namespaces (netns) and MACVLAN interfaces by releasing DHCP leases and removing virtual interfaces

5.2. Experimental Setup

5.2.1. Setup Overview

The experimental setup consists of three physical machines located in the same environment, representing a simplified client-router-server architecture. The first machine, referred to as CLIENT, generates traffic by initiating iperf -c requests to a server whose IP address it knows but has no direct route to beyond the router. The second machine, ROUTER, directly receives all traffic from CLIENT and forwards it to the third machine, SERVER. In scenarios involving a Captive Portal, ROUTER also authenticates and controls CLIENT's access before forwarding traffic. In the Clean Machine scenario, ROUTER acts as a simple forwarder, allowing unrestricted access. The SERVER machine represents the destination resources a real-world client would request, such as a website or other network services.

5.2.2. Machine Configuration

The CLIENT machine can operate with any OS; however, a Linux-based system is preferable due to the technological solutions used to simulate multiple clients. It requires only a single Ethernet NIC and must have sufficient hardware resources to handle multiple client simulations. The exact CPU and RAM requirements per single client remain undetermined. However, it has been observed that when running 100 simultaneous iperf3 -c instances, the client machine, featuring a AMD Ryzen 5 2500U processor [18], reached nearly 100% of CPU utilization. This issue, including its impact on scalability, will be further analyzed in Chapter 6, Section 6.2 (Limitations of the Study).The CLIENT's role is solely to generate traffic and authenticate simulated clients in the captive portal scenarios.

The ROUTER machine runs Rocky 8 Linux, ensuring compatibility with the company's Captive Portal software. It requires two Ethernet NICs to manage traffic between CLIENT and SERVER and features as CPU a Intel Core i7-6700 3.40 GHz (4 cores, 8 threads) [19] and 16GB of RAM. However, these are not hardware requirements, as the study focuses on CPU utilization percentage rather than absolute performance. The role of the machine is only handling and forwarding incoming traffic.

The SERVER machine serves only as a traffic endpoint and has no performance relevance in this experiment. For consistency and simplicity in configuration, it also runs Rocky 8 Linux, though its OS choice is otherwise inconsequential.

5.2.3. Connection Details

The physical network configuration remains constant across all test scenarios. The CLIENT is connected via Ethernet to the ROUTER, which in turn connects via Ethernet to the SERVER. The SERVER is also connected via Ethernet to the Office LAN, providing internet access cascaded down to the other two machines. All physical links operate at 1 Gbps bandwidth. The SERVER maintains a static IP configuration on both its NIC for every scenario:

- **NIC facing ROUTER:** 192.168.89.1/24
- **NIC facing Office LAN:** Receives a local IP via DHCP from the Office LAN network.

CLIENT always launches iperf3 -c requests using the static IP of the SERVER, simplifying the controlled test environment.

Clean Machine scenario

CLIENT simultes multiple clients by creating multiple virtual NIC pairs and assigning statically to each one of them an IP in the **192.168.137.0/24** subnet. A default route pointing to the ROUTER's IP is also addded.

The ROUTER has a default route pointing to the SERVER's IP and maintains two static IP configurations:

- **NIC facing CLIENT (enp2s0):** 192.168.137.254/24
- **NIC facing SERVER (enp3s0):** 192.168.89.99/24

SERVER adds a route for the **192.168.137.0/24** subnet, pointing to ROUTER's IP (192.168.89.99/24), to answer CLIENT back.

CoovaChilli scenario

In this scenario, the CLIENT simulates multiple clients by creating network namespaces and MACVLAN. Each simulated client receives a unique IP address via DHCP within the **10.1.0.0/24** subnet from the ROUTER, along with the default route pointing to the static IP of the virtual NIC of the ROUTER 10.1.0.1/24. Then, CLIENT must authenticate each instance before initiating multiple iperf -c requests to the SERVER simultaneously. The authentication process and mechanisms will be detailed in the next subsection.

The ROUTER has a default route pointing to the SERVER's IP and maintains two static IP configurations:

- **NIC facing CLIENT (enp2s0):** 192.168.137.254/24
- **NIC facing SERVER (enp3s0):** 192.168.89.99/24

However, when running CoovaChilli Captive Portal, the ROUTER automatically assigns itself the IP **10.1.0.1/24** on its virtual interface created by CoovaChilli.

SERVER adds a route for the **10.1.0.0/24** subnet, pointing to ROUTER's IP (192.168.89.99/24), to answer CLIENT back.

Company's Captive Portal Scenario

CLIENT, similarly to CoovaChilli' scenario, receives on each instance, through DHCP, an IP in the **192.168.137.0/24** subnet from the ROUTER, with a default route pointing to the static IP of the ROUTER. Authentication process is required before launching iperf3.

Both ROUTER and SERVER keep the same configuration as in the Clean Machine scenario.

This setup ensures that the CLIENT must pass through the ROUTER to access the SERVER, enabling controlled evaluation of the impact of the Captive Portal on network performance. With these configurations, all traffic flows through the ROUTER, and its hardware performance varies depending on whether it operates in a clean forwarding mode or actively processes authentication in the Captive Portal scenario.

5.2.4. Authentication Process

Before being able to launch iperf3 from the clients, each simulated client needs to be authenticated by the captive portal. Authentication is handled by a Bash script that iterates through all simulated clients (each in its own network namespace with a MACVLAN interface). For each namespace, the script executes a curl command to send an HTTP POST request containing the username and password required by the captive portal for authentication.

Additionally, a similar logout script properly handle session termination, ensuring that each client can properly disconnect from the captive portal when required.

These automation scripts streamline the authentication process, allowing all simulated clients to authenticate efficiently before generating traffic.

5.3. CPU Utilization vs Bandwidth

5.3.1. Theoretical Expectations

In networked systems, CPU utilization is expected to increase with available bandwidth because a higher bandwidth allocation directly translates to an increased volume of network traffic. When bandwidth increases, a larger number of packets can be transmitted per unit of time, which in turn requires more processing power from the machine handling the traffic. Considering the following factors:

- **Packets as the Fundamental Unit of Network Processing:** Data transmission occurs in the form of packets, each of which must be individually processed, routed, and forwarded by network devices. A higher bandwidth allocation means that a greater number of packets can be sent and received per second.
- **CPU Involvement in Packet Processing:** Each packet must undergo multiple

operations such as header parsing, routing decision-making, queuing, and forwarding, all of which impose a computational burden on the system. The greater the packet arrival rate, the more CPU cycles are consumed to keep up with the increased workload.

- **Captive Portal Overhead:** In environments where a captive portal is active, additional processing steps, such as authentication, logging, and policy enforcement, introduce even greater CPU load compared to a clean machine scenario. These overheads scale with packet volume, reinforcing the expectation that CPU usage will rise with bandwidth.

A key question in this analysis is whether the increase in CPU utilization follows a linear pattern with respect to bandwidth. In an ideal scenario, assuming an evenly distributed packet size and a stable processing cost per packet, one would expect CPU usage to increase linearly with bandwidth. However, real-world networking often introduces additional complexities, such as varying packet sizes, interrupt coalescing at the NIC level, and optimizations in TCP congestion control mechanisms. These factors could cause slight deviations from perfect linearity, making it important to verify the actual relationship through empirical measurement. Given these principles, the hypothesis to validate is: CPU usage should increase proportionally with bandwidth usage, with potential deviations caused by implementation details and OS-level optimizations.

5.3.2. Experimental Procedure

To empirically evaluate the relationship between CPU utilization and bandwidth usage, the number of clients was fixed at 100, ensuring a consistent network load in terms of active TCP connections. Each client received an equal share of the total available bandwidth, allowing for uniform distribution of network traffic across all active connections. This approach isolates the effect of bandwidth variation while eliminating the factor related to client count. The bandwidth was progressively limited across five predefined values: 100 Mbps, 200 Mbps, 400 Mbps, 800 Mbps, and the maximum available of 1 Gbps. To ensure result reliability, each test was executed multiple times and the values averaged to obtain more reliable data. Additionally, to account for any discrepancies in actual bandwidth consumption during each test, the CPU utilization values were normalized using the following formula:

$$\text{CPU\_Utilization\_Normalized} = \frac{\text{CPU\_Utilization\_Sampled} \times \text{Bandwidth\_Limit}}{\text{Bandwidth\_Sampled}} \tag{5.1}$$

This normalization ensures that if a test exceeded the intended bandwidth limit due to network variations, the corresponding CPU utilization value is adjusted accordingly. This step improves the accuracy of the CPU-to-bandwidth correlation analysis. The collected data was then plotted to verify the hypothesis.

5.3.3. Results

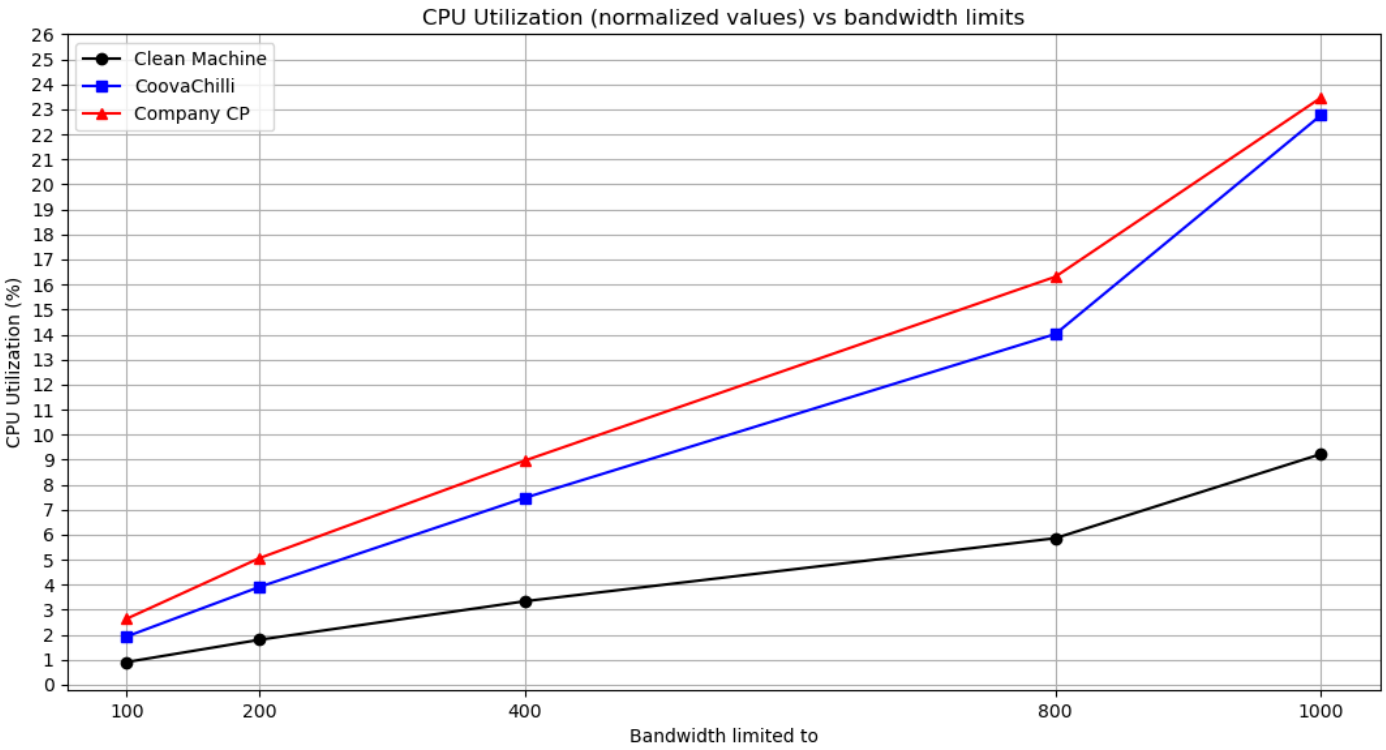

Figure 5.5: CPU Utilization normalized vs Bandwidth.

The chart illustrates the relationship between CPU utilization and bandwidth limits.

- The **x-axis** represents the five tested bandwidth limits: 100 Mbps, 200 Mbps, 400 Mbps, 800 Mbps, and 1 Gbps.
- The **y-axis** shows CPU utilization (%) with a granularity of 1%.
- The three test scenarios are represented as follows:
  - **Black** for the **clean machine** (baseline).
  - **Blue** for **CoovaChilli**.

  - **Red** for the **company's captive portal**.

- This visualization provides a clear comparison of CPU usage trends across different bandwidth conditions and captive portal solutions.

Band. Limits	CPU Clean	CPU Chilli	CPU Company	Chilli Incr.	Company Incr.
100	0.91	1.93	2.64	112.09	190.11
200	1.80	3.91	5.06	117.22	181.11
400	3.34	7.47	8.96	123.65	168.26
800	5.86	14.02	16.31	139.25	178.33
1000	9.22	22.76	23.46	146.85	154.45

Table 5.1: CPU Utilization vs Bandwidth for Different Scenario

The table presents the numerical values of CPU utilization corresponding to the data plotted in the chart, with additional insights.

- **Columns 2-4** report the CPU utilization (%) for each tested bandwidth limit across the three scenarios (the values plotted in the chart):
  - **CPU Clean**: Baseline values for the clean machine.
  - **CPU Chilli**: CPU utilization with CoovaChilli active.
  - **CPU Company**: CPU utilization with the company's captive portal active.
- **Columns 5-6** provide additional insights by reporting the relative CPU overhead (in percentage) compared to the clean machine:
  - **Chilli Incr.**: Percentage increase in CPU utilization when using CoovaChilli relative to the clean machine.
  - **Company Incr.**: Percentage increase in CPU utilization when using the company's captive portal relative to the clean machine.
- This table allows for a direct numerical comparison of CPU utilization trends and highlights the increasing overhead introduced by captive portals as bandwidth grows.

The following tables report, for each scenario, the relative increase in CPU utilization between subsequent bandwidth limits. This highlights how the rate of CPU consumption evolves as bandwidth increases.

Bandwidth Limit (Mbps)	CPU (%)	Net Increase (%)	Relative Increase (%)
100	0.91	-	-
200	1.80	0.89	97.80
400	3.34	1.54	85.56
800	5.86	2.52	75.45
1000	9.22	3.36	57.34

Table 5.2: Clean Machine - CPU utilization and incremental increase at different bandwidth limits.

Bandwidth Limit (Mbps)	CPU (%)	Net Increase (%)	Relative Increase (%)
100	1.93	-	-
200	3.91	1.98	102.59
400	7.47	3.56	91.05
800	14.02	6.55	87.68
1000	22.76	8.74	62.34

Table 5.3: CoovaChilli - CPU utilization and incremental increase at different bandwidth limits.

Bandwidth Limit (Mbps)	CPU (%)	Net Increase (%)	Relative Increase (%)
100	2.64	-	-
200	5.06	2.42	91.67
400	8.96	3.90	77.08
800	16.31	7.35	82.03
1000	23.46	7.15	43.84

Table 5.4: Company's CP - CPU utilization and incremental increase at different bandwidth limits.

Key observation:

- For every bandwidth value tested, the two **captive portal scenarios consistently exhibit higher CPU utilization** compared to the clean machine.

- **The expected theoretical relationship, higher bandwidth leading to increased CPU utilization, is empirically confirmed.** The increase in packet processing directly translates to higher CPU workload across all configurations.
- **The correlation between bandwidth and CPU utilization follows an almost perfect linear trend up to 800 Mbps.** However, at 1 Gbps, CPU utilization does not increase proportionally, suggesting potential system-level optimizations or hardware limits affecting processing efficiency at full capacity.

From these results it is clear that:

- **Empirical data aligns with theoretical expectations**, reinforcing the well-known principle that increased bandwidth leads to higher CPU consumption due to increased packet processing demands.
- **Captive portals introduce a measurable CPU overhead**, in every scenario.

For completeness, the chart and the table with sampled values before normalization are provided in Appendix A (Figure A.1 and Table A.2-A.4). This behavior aligns with theoretical expectations and provides a strong foundation for understanding how bandwidth constraints impact CPU load across different configurations.

5.4. CPU Utilization vs Number of Clients

5.4.1. Theoretical Expectations

This section addresses the relationship between the **number of clients and CPU utilization**. From a theoretical standpoint, we expect that:

- **More clients generate more TCP connections**. Each client initiates a separate TCP connection, requiring the system to manage multiple independent network flows, including tracking state, scheduling packets, and processing acknowledgments.
- **Captive portals should further amplify CPU utilization**, as they introduce additional per-connection processing overhead for the authentication processes.

The primary goals of this analysis are:

1. Analyze the correlation between the number of clients and CPU usage, determining its impact when the aggregated bandwidth is fixed.
2. Quantify the CPU overhead introduced by captive portals compared to a clean machine scenario.

3. Determine whether the company's proprietary captive portal performs better or worse than CoovaChilli in handling multiple client connections.

The following subsections detail the experimental setup, results, and analysis of the obtained data.

5.4.2. Experimental Procedure

To evaluate how CPU utilization scales with the number of clients, the total available bandwidth wasfixed at 1 Gbps. Each client received an equal share of the bandwidth, ensuring fair distribution and eliminating potential confounding factors related to uneven traffic allocation. The number of clients was incrementally increased in four predefined steps: 1, 10, 50, and 100, to analyze CPU utilization trends as connection load increased, allowing to observe how CPU usage evolved as more independent TCP connections were introduced. To ensure the reliability of the measurements, as for the previous experiment, multiple test runs were performed for each configuration, and the average CPU utilization was computed across all trials. Despite this averaging process, it is important to note that a confidence interval was not analytically estimated, meaning that some level of variability remains in the results. As in the bandwidth-focused experiment, to improve result accuracy, CPU utilization values were normalized against actual bandwidth consumption to mitigate any inconsistencies caused by network variations. This adjustment ensures that if more bandwidth was consumed during a test, the corresponding CPU utilization value is fairly corrected.

5.4.3. Results

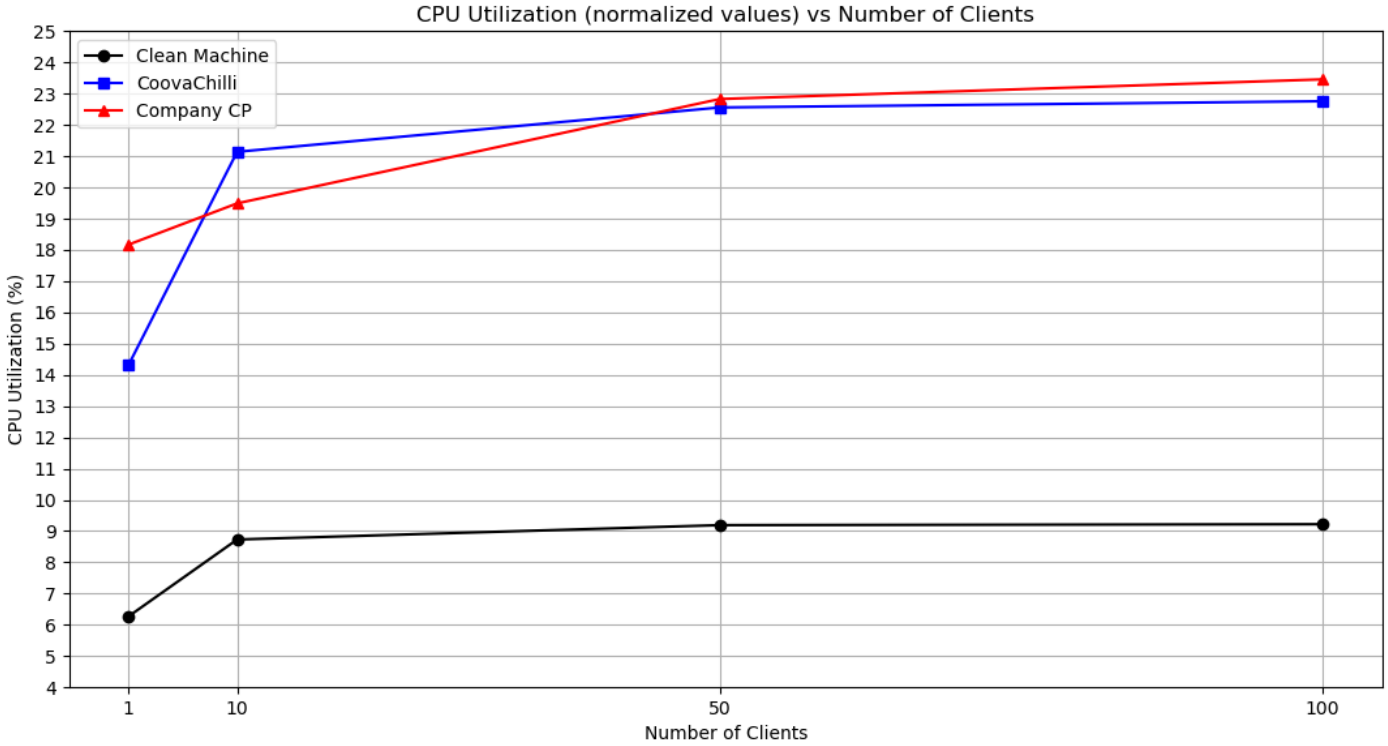

Figure 5.6: CPU Utilization normalized vs Number of Clients.

The chart illustrates the relationship between CPU utilization and the number of clients.

- The **x-axis** represents the number of clients in the test: 1, 10, 50, and 100.
- The **y-axis** shows the CPU utilization (%) with a granularity of 1%
- The three test scenarios are represented as follows:
  - **Black** for the **clean machine** (baseline).
  - **Blue** for **CoovaChilli**.
  - **Red** for the **company's captive portal**.
- This visualization allows for a direct comparison of how CPU utilization scales with increasing numbers of clients in the three scenarios.

Clients	CPU Clean	CPU Chilli	CPU Company	Chilli Incr.	Company Incr.
1	6.27	14.32	18.17	128.39	189.79
10	8.73	21.14	19.49	142.15	123.25
50	9.19	22.56	22.83	145.48	148.42
100	9.22	22.76	23.46	146.85	154.45

Table 5.5: CPU Utilization vs Number of Clients for Different Scenario

The table presents the numerical values of CPU utilization corresponding to the data plotted in the chart, with additional insights.

- **Columns 2-4** report the CPU utilization (%) for each tested number of clients across the three scenarios (the values plotted in the chart):
  - **CPU Clean**: Baseline values for the clean machine.
  - **CPU Chilli**: CPU utilization with CoovaChilli active.
  - **CPU Company**: CPU utilization with the company's captive portal active.
- **Columns 5-6** provide additional insights by reporting the relative CPU overhead (in %) compared to the clean machine:
  - **Chilli Incr.**: Percentage increase in CPU utilization when using CoovaChilli.
  - **Company Incr.**: Percentage increase in CPU utilization when using the company's captive portal.
- These values allow for an immediate visualisation of the computational cost introduced by captive portals at different client scales.
- Additionally, they provide a direct comparison between CoovaChilli and the company's captive portal.

The following tables report, for each scenario, the relative increase in CPU utilization between subsequent number of clients. This highlights how the rate of CPU consumption evolves as the number of clients increases

Clients	CPU (%)	Net Increase (%)	Relative Increase (%)
1	6.27	-	-
10	8.73	2.46	39.23
50	9.19	0.46	5.27
100	9.22	0.03	0.33

Table 5.6: Clean - CPU Utilization and incremental increase at different number of clients.

Clients	CPU (%)	Net Increase (%)	Relative Increase (%)
1	14.32	-	-
10	21.14	6.82	47.63
50	22.56	1.42	6.72
100	22.76	0.20	0.89

Table 5.7: CoovaChilli - CPU Utilization and incremental increase at different number of clients.

Clients	CPU (%)	Net Increase (%)	Relative Increase (%)
1	18.17	-	-
10	19.49	1.32	7.26
50	22.83	3.34	17.14
100	23.46	0.63	2.76

Table 5.8: Company's CP - CPU Utilization and incremental increase at different number of clients.

The key observations derived from the data are as follows:

- **Quantification of CPU Overhead Introduced by Captive Portals:**
  - The results confirm that both **CoovaChilli and the company's captive portal introduce significant CPU overhead** compared to the clean machine.
  - The **relative CPU increase exceeds 100% in all cases**, meaning that captive portals more than double CPU consumption.

  - At **100 clients**, the overhead reaches **146.85% for CoovaChilli and 154.45% for the company's CP**, demonstrating the computational burden imposed by these solutions.

- **Scaling Behavior of CPU Usage with Increasing Clients:**
  - The **largest CPU utilization increase occurs between 1 and 10 clients**, across both scenarios.
  - Beyond **10 clients, the incremental increase slows down considerably**, suggesting that after a certain threshold, additional connections do not linearly translate into CPU load increments.
  - The **clean machine scenario** exhibits this behavior most clearly, with **only a 0.03% CPU increase from 50 to 100 clients**, which is **well within a margin error**. This highlights that **handling new TCP connections is not the dominant factor in CPU usage** at higher loads, even without a Captive Portal solution enabled, and the system likely reaches a point where additional connections contribute negligible computational overhead.
  - It remains to be verified whether this result changes when using a different approach for simulating a high number of clients, or if, at very low per-client bandwidth allocations (in this case, 10 Mbps per client), the processing overhead becomes inherently negligible.
- **Non-Linear Growth in CPU Utilization:**
  - Unlike bandwidth tests (which followed a near-linear trend up to 800 Mbps), the CPU utilization **does not increase proportionally with the number of clients**.
  - This suggests that the overhead introduced by additional TCP connections **is not a simple function of the client count** but is influenced by factors such as **kernel optimizations, queue handling, and system-level efficiencies**.
  - The additional tables provide **incremental percentage increases**, reinforcing this observation: after an initial sharp rise, the relative CPU increase per additional client significantly diminishes.
- **Comparison Between CoovaChilli and the Company's Captive Portal:**
  - At **low client counts (1 and 10 clients)**, CoovaChilli **consumes more CPU than the company's CP**, possibly due to differences in connection

management overhead.

- As the client count increases, **the performance gap narrows**, with both captive portals reaching similar CPU utilization levels at **100 clients**.
- This confirms that the **difference between the two solutions is most pronounced at low loads**, and any performance gap diminishes as the number of clients increases.

Also for this experiment, for completeness, the chart and the table with sampled values before normalization are provided in Appendix A (Figure A.2 and Table A.4-A.6).

These results indicate that while the number of clients influences CPU utilization, **the impact diminishes as the number of concurrent connections grows beyond a certain point**. The clean machine scenario establishes a **clear reference baseline**, while the captive portal configurations consistently add overhead. The company's captive portal presents different scaling behavior compared to CoovaChilli, especially at lower client counts. This analysis confirms that **the primary CPU utilization driver is bandwidth, rather than just the number of independent connections**. Further discussion on experimental constraints and limitations will be addressed in a later section.

6 | Key Takeaways

This chapter presents a concise summary of the main results derived from the analysis of the various tests, explained in detail in the previous chapter. What was found in the analysis provides interesting insight into how CPU utilization varies in the three different scenarios and conditions analyzed. By critically reviewing what was found, it is possible to suggest some trends and highlight broader implications, with some necessary clarifications. In addition to summarizing the results, a critical discussion of the limitations of the entire study is presented . Although the methodology adopted and described is designed to provide as much information as possible, several factors must be taken into account when interpreting the results. Explaining these limitations in depth is essential to contextualize the work done and to determine how far the suggested trends can be extended and generalized beyond the controlled testing environment.

6.1. Summary of Findings and Possible Trends

The performance evaluation revealed some key takeaways:

- **Bandwidth is the dominant factor on CPU utilization.** It was observed that the relationship between bandwidth and CPU utilization is essentially linear up to 800 Mbps, a threshold beyond which there is an increase with a higher rate. Further increases in bandwidth are then assumed to lead to an even greater relative increase, indicating an major deviation from a linear trend.
- **The use of captive portals brings with it significant overhead, especially at high bandwidth.** Even at 100 Mbps, a significant gap is observable between the clean scenario and those with captive portals. This difference becomes more significant as the bandwidth increases, culminating at 1 Gbps with CPU utilization in the clean scenario at 9% versus about 23% in the captive portal scenarios.
- **The number of concurrent users has a significant impact especially at low numbers.** A significant increase in CPU utilization is observed in the transition from 1 to 10 clients, beyond this point the impact of this factor decreases (easily ob-

servable from Tables 5.6 - 5.8), becoming statistically insignificant in the transition from 50 to 100 users. This suggests two hypotheses:

1. Optimizations at Operating System level or limitations in the technological solutions adopted put a limit on a further increase in CPU utilization.
2. Since the total bandwidth is always 1 Gbps, the bandwidth per user allocated is so low that the overhead introduced per client becomes irrelevant.

- **Comparison between CoovaChilli and the proprietary solution provides variable results.** The company's software performs worse than CoovaChilli with a single user, slightly better with 10 users, and basically equal when the number of users exceeds 50. It has to be noted that the delta between the two solutions becomes very small as the number of users increases, potentially falling within a margin of error.

These findings provide empirical and valuable basis for understanding and quantifying the impact of these two relevant factors when deploying captive portal solutions.

6.2. Limitations of The Study

As anticipated, despite the various insights and takeaways obtained for the captive portal performance domain, several limitations must be acknowledged:

1. **Limited number of simultaneous users.** The study was conducted with a relatively small number of clients. Considering that in certain real-world scenarios captive portals are deployed with numbers of users involved reaching tens of thousands simultaneously, the case of a large airport. Having conducted reliable tests with a maximum number of users of 100 places limitations on generalizing the results obtained to large-scale deployments.
2. **Limitations of the technological solutions adopted in simulating multiple users.** The methods used to simulate multiple clients with a single physical machine result in inherent limitations in scalability. Having to create a virtual interface or a network namespace - MACVLAN pair for each individual user brings with it a considerable cost in terms of hardware resources for the machine used; this cost limited the maximum number of users to 100. Tests were conducted with 250 users but the results obtained were judged unreliable.
3. **The clients' TCP streams come from the same physical machine.** Although the router treats each TCP stream as coming from a different machine, all network

traffic is generated from the same physical machine. This could have led to system-level optimizations in queue management, reduction in individual stream overhead, or other kernel-level efficiencies that would not be present if the users were actually separate physical machines.

4. **Network namespace constraints.** Although network namespaces in Linux provide logical isolation at the network level, all traffic goes through the one true Ethernet network interface. This may lead to different behavior that deviates from a real-world scenario.

5. **Use of a single tool to generate traffic.** This study is based solely on the use of iperf3 to generate traffic. Despite its strengths and the simplicity of its use, it cannot replicate the complexity of network traffic that a real captive portal handles.

6. **Impact of system-specific optimizations.** The results obtained could be affected by specific optimizations at the Linux network stack level, which are not necessarily representative of other operating systems or hardware configurations. Different kernel versions or NIC drivers could have produced different results.

7. **Default network configuration settings.** Router network system settings were kept in their default state: no traffic offloading mechanisms were applied and no OS-level parameters were tuned. In a real-world scenario, network administrators often fine-tune several parameters or enable/disable certain options.

The limitations presented highlight areas where future work and research could make a significant contribution in understanding captive portal performance under more realistic large-scale conditions.

7 | Conclusions

7.1. Summary of the Work

This thesis work analyzes the performance impact introduced by the use of a captive portal, with a main emphasis on CPU utilization under different conditions and with different solutions. The study was conducted using a controlled test environment, where CPU consumption was measured while the Router machine, responsible for traffic management, was forwarding network packets and handling authentication processes. In particular, it was analyzed how two crucial factors in this area, bandwidth consumption and the number of concurrent users, affect this metric.

Through a series of structured experiments, a comparison was made between an open-source solution, CoovaChilli, and a proprietary implementation from a private company.

Technological solutions such as virtual network interfaces and Linux network namespaces with MACVLAN were used to simulate multiple users, within a single physical machine, allowing scalable testing up to 100 concurrent clients. iperf3 was used for the generation of network traffic while both CPU usage and consumed bandwidth were extracted and parsed with a script from the kernel.

In summary, this analysis seeks to provide a quantitative understanding of the overhead introduced by captive portals, with respect to a clean scenario, with particular emphasis on how two key factors such as bandwidth and number of users impact the scalability of resource utilization.

7.2. Future Works

- **Testing with higher Bandwidth:** Bandwidth values beyond the tested limits would provide additional insights into whether CPU utilization keeps deviating from linear scaling or returns to a more predictable trend. Increasing bandwidth could also lead to higher per-client allocation, allowing to assess whether the impact of a high number of clients became negligible due to the small per-client bandwidth.

- **Testing with a higher number of clients:** an increment of the number of concurrent clients would allow for a more accurate evaluation of the impact of this factor. For doing this, more powerful hardware for client machine is required for sure. It may be necessary also exploring alternative technological solutions in order to find a more scalable approach to simulate multiple clients.
- **Repeating tests with two clients machines:** improving the experimental setup by adding a second clients machine, and retaking all the tests by splitting the number of client between the two machines, would make possible ruling out wether some results were influenced by the fact that a single machine was used for generating all the TCP streams or not.
- **Different traffic generation tools:** Since this study relied only on iperf3 for traffic generation, trying other tools could offer additional perspective on performance behaviour.
- **Analysis in-depth of flamegraphs:** A more detailed analysis of flamegraphs could provide greater insights into where CPU clock cycles are exactly spent, helping to identify potential bottlenecks and understanding in details what functions of the captive portals are the most responsible for the increment in CPU utilization.
- **Tuning parameters and OS optimizations:** Leveraging on system tuning techniques such as offloading, adjusting kernel parameter or utilizing more efficient packet processing mechanisms could reduce performance overhead and improve overall efficiency.

Bibliography

[1] diagrams.net, "diagrams.net - Online Diagram Software," 2025.

[2] Divya Singla, Neetu Verma, "Performance analysis of authentication system: A systematic literature review," *ResearchGate*, 2023.

[3] CoovaChilli Developers, "Coovachilli - open-source captive portal access controller," 2024.

[4] Nils Petersohn, "Coovachilli wifidocs," 2013.

[5] CoovaChilli Developers and Contributors, "Coovachilli github repository," 2024.

[6] OpenWrt Community, "Coovachilli setup on openwrt," 2020.

[7] SequenceDiagram.org, "Online sequence diagram tool," 2024.

[8] ESnet, "iPerf - The ultimate speed test tool for TCP, UDP and SCTP," 2024.

[9] B. Gregg, "Flamegraph," 2024.

[10] Brendan Gregg, "Brendan Gregg Website," 2025.

[11] H. Muhammad and Contributors, "htop - an interactive process viewer," 2024.

[12] S. Team, "mpstat - report processors related statistics," 2023.

[13] B. Gregg, *Systems Performance: Enterprise and the Cloud*. Addison-Wesley Professional, 2nd ed., 2020.

[14] Hangbin Liu, "Introduction to linux interfaces for virtual networking," 2018.

[15] R. Droms, "RFC 2131: Dynamic Host Configuration Protocol." Internet Engineering Task Force (IETF), 1997.

[16] M. Kerrisk, "namespaces(7) - linux manual page," 2024.

[17] N. D. Eric W. Biederman, "ip-netns(8) - linux manual page," 2025.

[18] TechPowerUp, "AMD Ryzen 5 2500U Specifications," 2024.

[19] Intel Corporation, “Intel® Core™ i7-6700 CPU Specifications,” 2022.

A | Appendix A

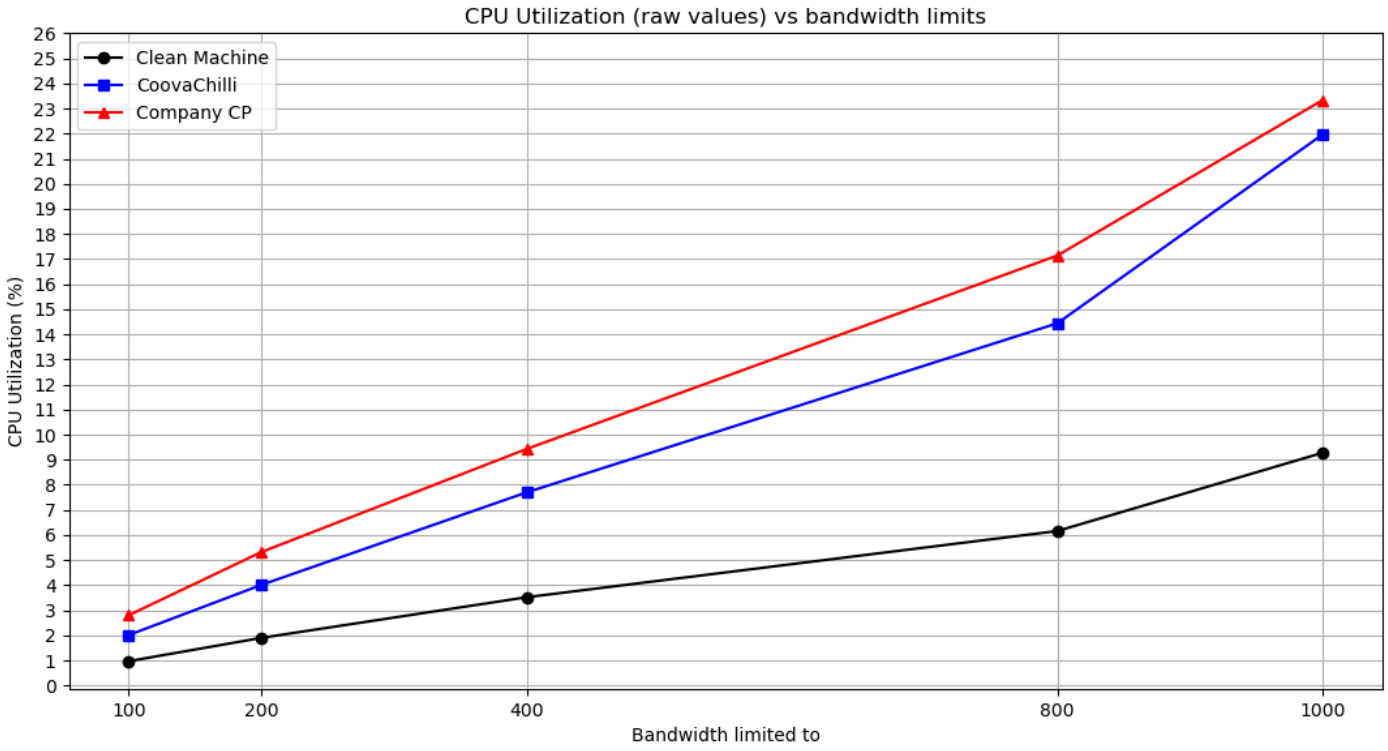

Figure A.1: CPU Utilization vs Bandwidth not normalized.

Band. Limit	Band. Sampled (Mbps)	CPU Sampled	CPU Normalized
100	106.19	0.97	0.91
200	210.65	1.9	1.8
400	421.13	3.52	3.34
800	840.54	6.16	5.86
1000	1006.99	9.28	9.22

Table A.1: CLEAN Machine - CPU utilization values sampled and normalized.

Band. Limit	Band. Sampled (Mbps)	CPU Sampled	CPU Normalized
100	104.05	2.01	1.93
200	204.87	4.01	3.91
400	412.32	7.7	7.47
800	823.86	14.44	14.02
1000	965.19	21.97	22.76

Table A.2: CoovaChilli - CPU utilization values sampled and normalized.

Band. Limit	Band. Sampled (Mbps)	CPU Sampled	CPU Normalized
100	106.19	2.8	2.64
200	210.36	5.32	5.06
400	420.94	9.43	8.96
800	840.47	17.14	16.31
1000	994.75	23.34	23.46

Table A.3: Company's CP - CPU utilization values sampled and normalized.

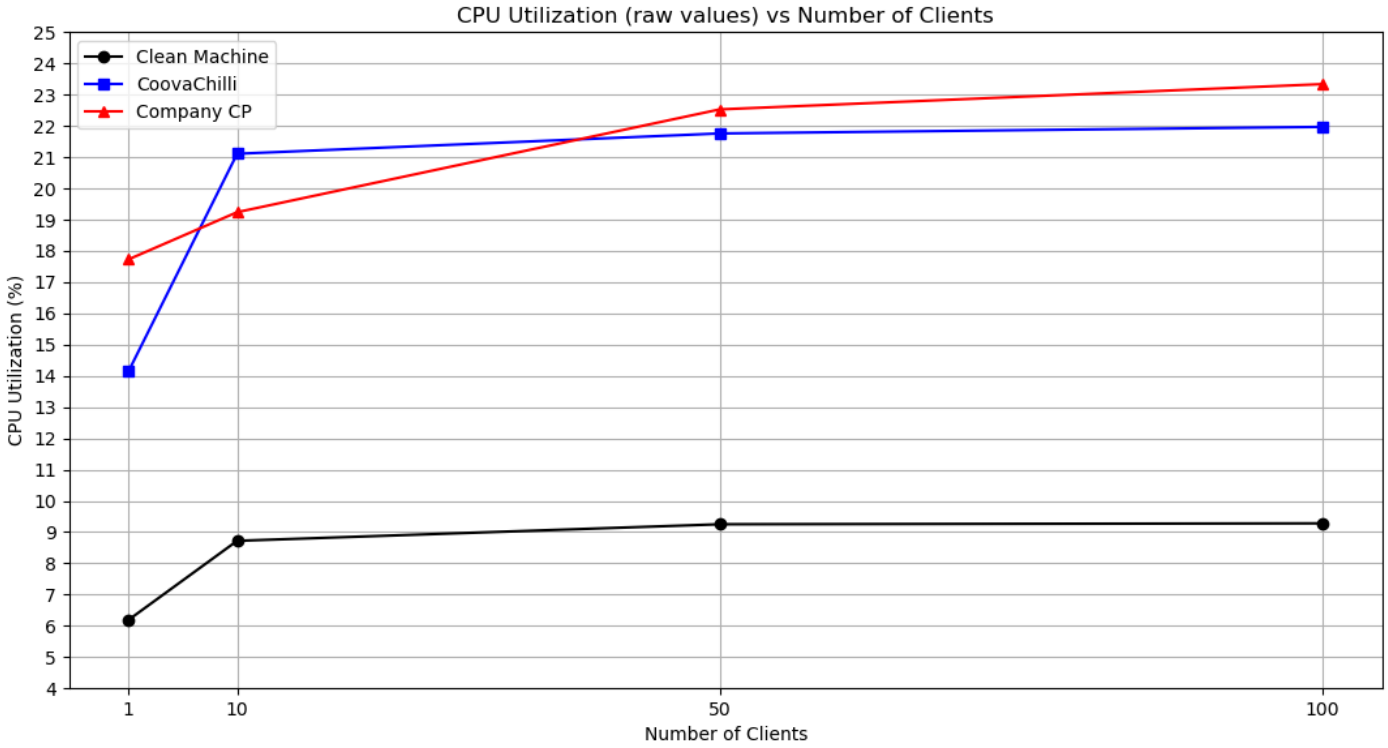

Figure A.2: CPU Utilization vs Number of Clients not normalized.

Number of Clients	Band. Sampled (Mbps)	CPU Sampled	CPU Normalized
1	987.61	6.19	6.27
10	998.62	8.72	8.73
50	1006.2	9.25	9.19
100	1006.99	9.28	9.22

Table A.4: CLEAN Machine - CPU utilization values sampled and normalized.

Number of Clients	Band. Sampled (Mbps)	CPU Sampled	CPU Normalized
1	987.85	14.15	14.32
10	998.8	21.11	21.14
50	964.75	21.76	22.56
100	965.19	21.97	22.76

Table A.5: CoovaChilli - CPU utilization values sampled and normalized.

Number of Clients	Band. Sampled (Mbps)	CPU Sampled	CPU Normalized
1	975.98	17.73	18.17
10	986.98	19.24	19.49
50	986.73	22.53	22.83
100	994.75	23.34	23.46

Table A.6: Company's CP - CPU utilization values sampled and normalized.

List of Figures

List of Tables

Acknowledgments

La prima persona che voglio ringraziare con tutto il cuore è mia mamma: senza i suoi sacrifici e il costante supporto che mi ha sempre dato, niente di tutto questo sarebbe stato possibile. La seconda è mio fratello Matteo, la persona più importante nella mia vita. Un grazie di cuore va anche a tutto il resto della mia famiglia, che mi ha sempre sostenuto sin da quando ero piccolo, senza mai farmi sentire sotto pressione. Un ringraziamento speciale a tutti i miei amici e amiche: siete persone straordinarie e sono orgoglioso di avervi al mio fianco. Doverosi ringraziamenti vanno al mio advisor aziendale, Alberto Pollastro, per avermi seguito lungo tutto il progetto di tesi, così come ai colleghi che ho avuto il piacere di conoscere e da cui ho ricevuto preziosi consigli.. Infine, un sentito grazie al mio relatore, Gianni Antichi, che, nonostante la tesi sia stata svolta in azienda, è sempre stato più che disponibile a fornirmi supporto e suggerimenti.