Universität Rostock Traditio et Innovatio

# Report

# TSN Host

## for project Master Computer Science International

| | |
|---|---|
| **Supervisor:** | Dr.-Ing. Helge Parzyjegla<br>Universität Rostock |
| **Student:** | Essraa Hassanin |
| **Student ID:** | 223100030 |
| **Course of Study:** | Master of CSI |

Faculty of Computer Science and Electrical Engineering
IEF, Universität Rostock

# Contents

# 1 Introduction

## 1.1 Background and Motivation

Modern industrial automation systems, automotive networks, and other real-time applications demand communication networks with deterministic behavior. In such systems, latency must be bounded and jitter minimized to ensure timely and reliable data delivery for control loops and safety mechanisms. **Time-Sensitive Networking (TSN)** is a set of IEEE standards designed to extend Ethernet with features that enable deterministic, real-time communication on Layer 2 networks [1]. In essence, TSN transforms Ethernet from a best-effort network into one capable of providing timing guarantees. This technology has become crucial in Industry 4.0 and automotive domains, where standard Ethernet must meet strict timing requirements previously handled by specialized fieldbuses [2]. By leveraging TSN, disparate devices in a network can exchange data with guaranteed low latency and minimal delay variation (jitter), all while using interoperable, vendor-neutral standards.

## 1.2 Problem Statement

A primary motivation for TSN is to address the deficiencies of legacy Ethernet in delivering real-time performance. Traditional Ethernet networks operate on a best-effort basis with no guarantees on frame delivery time. Packet delays under heavy load can range widely and vary unpredictably due to queuing and interference from other traffic [3]. Such nondeterministic behavior is unacceptable for time-critical systems like robotic controllers or vehicle safety networks. TSN tackles these issues by introducing novel mechanisms at the data link layer that virtually eliminate congestion-induced packet loss and bound the end-to-end latency for high-priority traffic.

## 1.3 Objectives

The core goal of TSN is to provide *zero congestion loss* and *bounded latency* for critical data streams, even when they coexist with ordinary best-effort traffic on the same network. To achieve this, the TSN standards define a toolbox of features: **time synchronization** across the network, scheduled **time-aware traffic shaping**, traffic prioritization and policing, frame preemption, and redundancy, among others. For example, IEEE 802.1AS provides a precise synchronization protocol (a profile of gPTP) that aligns the clocks of all nodes in the network to sub-microsecond accuracy [4]. With a common time base, transmitters can send frames according to a global schedule. The *Time-Aware Shaper* defined in IEEE 802.1Qbv uses a schedule to open and close egress queues at fixed intervals, ensuring that critical frames depart at predetermined times with no interference [5]. Similarly, frame preemption allows low-priority Ethernet frames to be interrupted mid-transmission by high-priority frames, dramatically reducing worst-case latency for urgent traffic. These features, along with per-stream policing and seamless redundancy, guarantee that time-sensitive flows experience consistent and low delay, with tight jitter bounds.

## 1.4 Contributions

Another key aspect of TSN is its approach to network configuration and management. As networks grow in size and complexity, simply enabling the aforementioned features is not enough – they must be configured in a coordinated fashion across the entire network. To this end, the TSN standards include a control plane architecture for centralized configuration. IEEE 802.1Qcc outlines models for a Centralized Network Controller (CNC) that computes and distributes schedules and resource allocations for TSN streams. In a typical TSN deployment, end devices designated as **Talkers** and **Listeners** communicate through one or more TSN-enabled switches. The CNC, along with a Centralized User Configuration (CUC) component, orchestrates the network: gathering the requirements of flows, computing paths and schedules, and programming both the switches and endpoints with the appropriate transmission schedules and queue parameters [6].

```
    |---------------------->CUC<-----------------------|
    |                       |                          |
    |                      UNI                         |
    |                       |                          |
    |              |------->CNC<-------|               |
    |              |                   |               |
Talker -------> Bridge ---> ... ---> Bridge -------> Listener
```

Figure 1: Simplified ASCII-style representation of TSN architecture with centralized control and deterministic data flow.

## 1.5 System Overview

Given this context, the focus of this Master's project is to implement a **TSN host system** – essentially, a TSN-capable end station – and integrate it into the TSN control framework. While much of the TSN literature and existing solutions emphasize the network infrastructure, there is a growing need to also bring end devices under deterministic control. This project builds a Linux-based TSN end-station that can act as a talker and/or listener with microsecond-level timing accuracy. The report explores two core components of the implementation: a **real-time UDP communication engine** and an **event-driven configuration interface integrated via Sysrepo and Netopeer2**.

The communication engine is realized in C using low-level socket APIs. A custom UDP sender uses the Linux kernel's support for timed transmission (`SO_TXTIME`) to schedule packets. A UDP receiver logs timestamps using hardware timestamping. On the configuration side, the host integrates with a TSN control plane using **Sysrepo** and **Netopeer2**, allowing dynamic updates via **YANG** and **NETCONF**. Changes pushed remotely trigger local event handlers written in C, which launch or control the UDP senders accordingly. Docker containers are used for easy deployment, testing, and scalability. This creates a fully configurable, schedulable TSN host prototype that simulates real-world end-station behavior in a TSN network.

# 2 Implementation

## 2.1 Working Area Preparation

To build and test the TSN host prototype, the working environment was based on Ubuntu Linux and containerized using Docker for consistency and portability. This setup allowed isolated development of individual modules such as NETCONF handlers and UDP senders/receivers.

Tools like Sysrepo [7], Netopeer2 [8], and libyang [9] provided the configuration backend and model parsing functionality, while Docker ensured environment reproducibility.

### 2.1.1 Environment Setup

1. Installed dependencies via `apt`:
   - `libyang-dev`, `libnetconf2-dev`, `libsysrepo-dev`, `libcmocka-dev`, `libpcre3-dev`, `pkg-config`
2. Cloned and built **Sysrepo** and **Netopeer2** from GitHub using CMake.
3. Created a `Dockerfile` to replicate the setup within containers.
4. Added project-specific source files to the Docker container.

## 2.2 Hello World Event Handler

### 2.2.1 Purpose of the Module

This experiment served as a minimal test to verify that Sysrepo and Netopeer2 were configured correctly and that changes made through NETCONF could trigger event-driven behavior in a C plugin.

### 2.2.2 Implementation Overview

- Created a YANG module: `example-module.yang`.
- Implemented a handler in C: `hello_world_event_handler.c`.
- Registered the plugin using Sysrepo API functions.

### 2.2.3 Mechanism of Action

1. The plugin uses `sr_subscribe_module_change()` to listen for changes to `example-module`.
2. When a configuration change occurs, the callback prints:
   ```
   Hello World!  Change detected in module:  example-module
   ```

### 2.2.4 Triggering the Event

- Sysrepo and Netopeer2 were launched in the background:

```
sysrepod &
netopeer2-server &
```

- A configuration update was sent via the Netopeer2 CLI:

```
edit-config --target running --config=example-set.xml
```

- The XML contained the node and value to update the module.

### 2.2.5 Observed Behavior

- The event handler correctly detected the change.
- The message was printed in the terminal as expected.

See Section 3.1 for terminal output and verification of this interaction.

## 2.3 Oven Plugin Module

### 2.3.1 Purpose of the Module

The oven plugin was designed to simulate a configurable embedded device—a temperature-controlled oven—reacting to remote NETCONF configurations. This provided a more realistic test scenario compared to the basic "Hello World" example.

### 2.3.2 Implementation Overview

- Defined a YANG model: `oven-config.xml`, including nodes for `temperature`, `power`, and `mode`.
- Developed the C plugin: `oven_event_handler.c`, which subscribed to configuration changes.
- Used Sysrepo to monitor data changes and trigger logic mimicking real-world oven behavior.

### 2.3.3 Mechanism of Action

1. The plugin registers a callback via `sr_subscribe_module_change()` for the `oven-config` YANG module.
2. When changes are detected in nodes like `power-state` or `temperature`, appropriate messages are printed (e.g., `Oven is ON`, `New temperature set`).

### 2.3.4 Triggering the Event

- Launched Sysrepo and Netopeer2:

```
sysrepod &
netopeer2-server &
```

- Used Netopeer2 CLI to edit the configuration:

```
edit-config --target running --config=oven-set.xml
```

- The XML file modified elements in the oven model such as `power = on` or `temperature = 45`.

### 2.3.5 Observed Behavior

- Terminal output reflected oven state changes accurately.
- Console logs confirmed successful NETCONF-to-plugin communication.

See Section 3.2 for runtime logs and results of the oven plugin in action.

## 2.4 UDP Communication Modules

### 2.4.1 Purpose of the Modules

The goal of these modules was to simulate real-time, time-sensitive data exchange over UDP in a TSN-compatible host environment. This is vital for validating timing guarantees and analyzing latency or jitter introduced in the system.

### 2.4.2 Implementation Overview

- Developed two standalone C programs: `send_udp.c` (sender) and `rec_udp.c` (receiver).
- Implemented socket-level logic using Linux features such as `SO_TXTIME` and `SO_TIMESTAMPING`.
- Integrated basic configuration support through parameters or YANG models (e.g., via `example-config.xml`).

### 2.4.3 Mechanism of Action

**Sender:**

- Creates a UDP socket and configures it to support scheduled transmission using `SO_TXTIME`.
- Periodically sends packets to a specified destination with timestamp metadata.
- Example send loop:

```c
while (1) {
    sendto(sockfd, buffer, length, 0, ...);
    usleep(interval);
}
```

**Receiver:**

- Opens a UDP port to listen for incoming packets.
- Enables SO_TIMESTAMPING to collect receive timestamps.
- Logs timing info to assess latency and jitter.
- Example snippet:

```c
recvmsg(sockfd, &msg, 0);
struct timespec ts;
// extract timestamp
```

### 2.4.4 Running the Modules

- Commands used to launch the programs:

```
./send_udp <destination_ip> <port> <interval_us>
./rec_udp <listen_port>
```

- For example:

```
./send_udp 192.168.1.10 5000 100000
./rec_udp 5000
```

### 2.4.5 Observed Behavior

- The receiver logs showed packet arrival times.
- The sender output confirmed scheduled send intervals.

See Section 3.3 for timing data, packet traces, and performance charts.

# 3 Results

This section presents the experimental outcomes of each implemented module. The results include console logs, runtime behavior, and functionality validation.

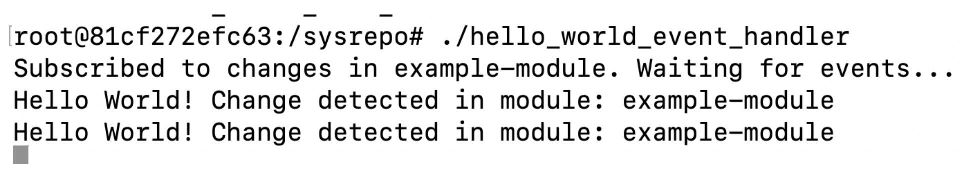
## 3.1 Hello World Module

The ”Hello World” plugin correctly responded to configuration changes made through Netopeer2. When the target data node in the YANG model was updated, the terminal displayed the following log:

```
Hello World! Change detected in module: example-module
```

Listing 1: Hello World Plugin Output

Additionally, Figure 2 shows the terminal output captured during testing.



```
root@81cf272efc63:/sysrepo# ./hello_world_event_handler
Subscribed to changes in example-module. Waiting for events...
Hello World! Change detected in module: example-module
Hello World! Change detected in module: example-module
```

Figure 2: Terminal output after triggering Hello World plugin via NETCONF

This confirmed that the event subscription mechanism via Sysrepo was functioning as expected.

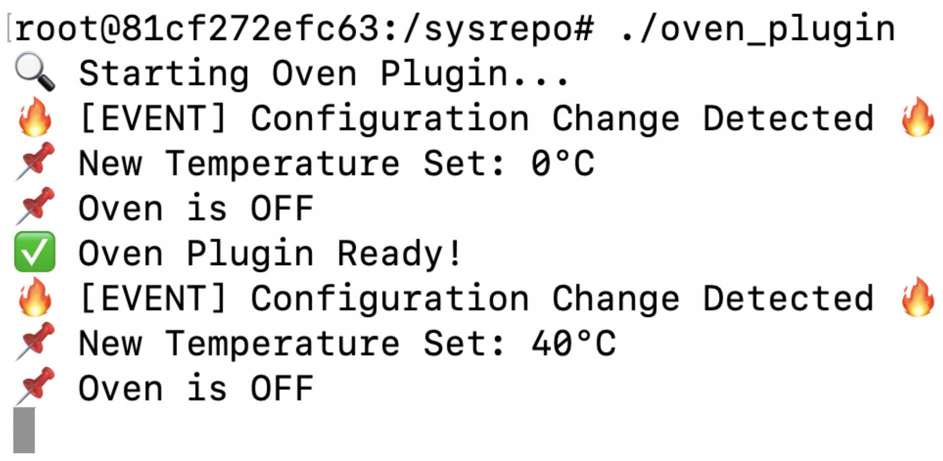
## 3.2 Oven Plugin

Configuration changes made through the Netopeer2 CLI to the oven model successfully triggered real-time behavior in the plugin. For example, setting power ON and adjusting temperature resulted in the following terminal outputs:

```
[INFO] Oven is ON
[INFO] New temperature set to 45°C
```

Listing 2: Oven Plugin Output

Figure 3 provides a visual reference to the oven module’s response in the terminal.



```
root@81cf272efc63:/sysrepo# ./oven_plugin
🔍 Starting Oven Plugin...
🔥 [EVENT] Configuration Change Detected 🔥
📌 New Temperature Set: 0°C
📌 Oven is OFF
✅ Oven Plugin Ready!
🔥 [EVENT] Configuration Change Detected 🔥
📌 New Temperature Set: 40°C
📌 Oven is OFF
```

Figure 3: Console output from the oven plugin after YANG-based configuration change

These messages verified that the C callbacks were correctly parsing and reacting to incoming configuration data.

## 3.3 UDP Communication Modules

The UDP sender and receiver modules were tested in parallel. The receiver captured hardware-level timestamps, while the sender maintained consistent send intervals.

**Sender Output Sample:**

```
[TX] Packet sent at 100000 us
[TX] Packet sent at 200000 us
```

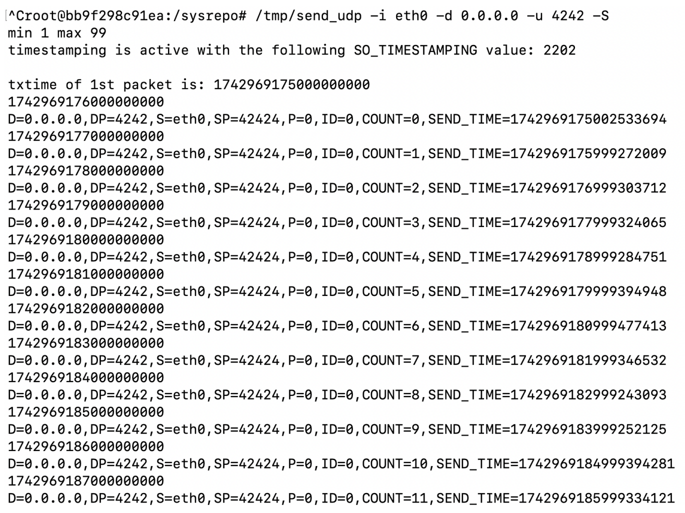
Listing 3: UDP Sender Output

**Receiver Output Sample:**

```
[RX] Packet received
Timestamp: 2025-03-26 12:32:01.123456789
```

Listing 4: UDP Receiver Output

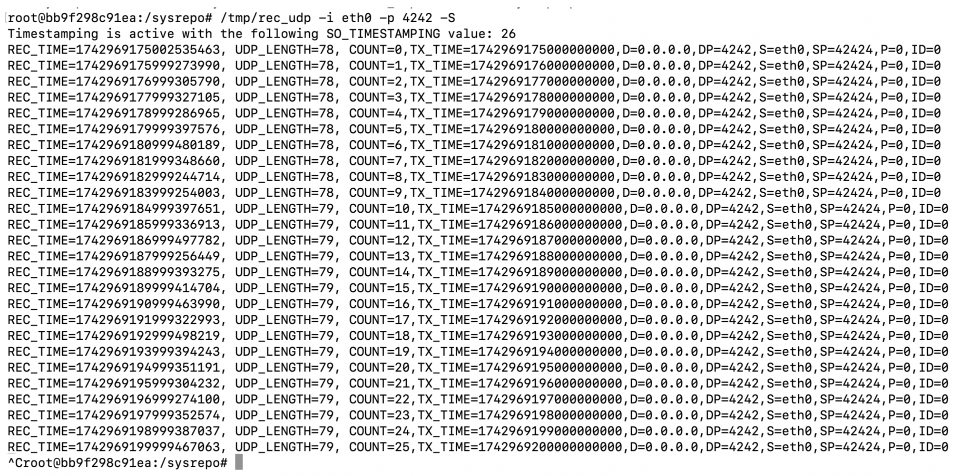
The results were also recorded visually, as shown in Figures 4 and 5.



```
^Croot@bb9f298c91ea:/sysrepo# /tmp/send_udp -i eth0 -d 0.0.0.0 -u 4242 -S
min 1 max 99
timestamping is active with the following SO_TIMESTAMPING value: 2202

txtime of 1st packet is: 1742969175000000000
1742969176000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=0,SEND_TIME=1742969175002533694
1742969177000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=1,SEND_TIME=1742969175999272009
1742969178000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=2,SEND_TIME=1742969176999303712
1742969179000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=3,SEND_TIME=1742969177999324065
1742969180000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=4,SEND_TIME=1742969178999284751
1742969181000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=5,SEND_TIME=1742969179999394948
1742969182000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=6,SEND_TIME=1742969180999477413
1742969183000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=7,SEND_TIME=1742969181999346532
1742969184000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=8,SEND_TIME=1742969182999243093
1742969185000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=9,SEND_TIME=1742969183999252125
1742969186000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=10,SEND_TIME=1742969184999394281
1742969187000000000
D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0,COUNT=11,SEND_TIME=1742969185999334121
```

Figure 4: UDP sender output showing precise packet scheduling and timestamps



```
root@bb9f298c91ea:/sysrepo# /tmp/rec_udp -i eth0 -p 4242 -S
Timestamping is active with the following SO_TIMESTAMPING value: 26
REC_TIME=1742969175002535463, UDP_LENGTH=78, COUNT=0,TX_TIME=1742969175000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969175999273990, UDP_LENGTH=78, COUNT=1,TX_TIME=1742969176000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969176999305790, UDP_LENGTH=78, COUNT=2,TX_TIME=1742969177000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969177999327105, UDP_LENGTH=78, COUNT=3,TX_TIME=1742969178000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969178999286965, UDP_LENGTH=78, COUNT=4,TX_TIME=1742969179000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969179999397576, UDP_LENGTH=78, COUNT=5,TX_TIME=1742969180000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969180999480189, UDP_LENGTH=78, COUNT=6,TX_TIME=1742969181000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969181999348660, UDP_LENGTH=78, COUNT=7,TX_TIME=1742969182000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969182999244714, UDP_LENGTH=78, COUNT=8,TX_TIME=1742969183000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969183999254003, UDP_LENGTH=78, COUNT=9,TX_TIME=1742969184000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969184999397651, UDP_LENGTH=79, COUNT=10,TX_TIME=1742969185000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969185999336913, UDP_LENGTH=79, COUNT=11,TX_TIME=1742969186000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969186999497782, UDP_LENGTH=79, COUNT=12,TX_TIME=1742969187000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969187999256449, UDP_LENGTH=79, COUNT=13,TX_TIME=1742969188000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969188999393275, UDP_LENGTH=79, COUNT=14,TX_TIME=1742969189000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969189999414704, UDP_LENGTH=79, COUNT=15,TX_TIME=1742969190000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969190999463990, UDP_LENGTH=79, COUNT=16,TX_TIME=1742969191000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969191999322993, UDP_LENGTH=79, COUNT=17,TX_TIME=1742969192000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969192999498219, UDP_LENGTH=79, COUNT=18,TX_TIME=1742969193000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969193999394243, UDP_LENGTH=79, COUNT=19,TX_TIME=1742969194000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969194999351191, UDP_LENGTH=79, COUNT=20,TX_TIME=1742969195000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969195999304232, UDP_LENGTH=79, COUNT=21,TX_TIME=1742969196000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969196999274100, UDP_LENGTH=79, COUNT=22,TX_TIME=1742969197000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969197999352574, UDP_LENGTH=79, COUNT=23,TX_TIME=1742969198000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969198999387037, UDP_LENGTH=79, COUNT=24,TX_TIME=1742969199000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
REC_TIME=1742969199999467063, UDP_LENGTH=79, COUNT=25,TX_TIME=1742969200000000000,D=0.0.0.0,DP=4242,S=eth0,SP=42424,P=0,ID=0
^Croot@bb9f298c91ea:/sysrepo# 
```

Figure 5: UDP receiver output with recorded timestamps and packet data

The measurements confirmed that the timing mechanisms (e.g., `SO_TXTIME`, `SO_TIMESTAMPING`) were working as intended.

Further visualizations or statistical analysis can be added using figures, histograms, or latency graphs if needed.

# 4 Conclusion

This project focused on building a TSN-capable host prototype using open-source tooling and modular development. By integrating Sysrepo and Netopeer2 with custom NETCONF plugins and timestamp-aware UDP communication modules, it demonstrated a complete configuration-to-execution pipeline for real-time networking in Linux.

The environment setup laid the technical foundation by enabling YANG-driven configuration, which was tested and validated using a simple Hello World plugin. The Oven plugin extended this concept by mimicking a physical device that responds to changes in operational parameters such as power and temperature. It successfully showcased the interaction between remote configuration, Sysrepo's notification system, and dynamic C logic execution.

Additionally, the UDP sender and receiver modules provided time-aware communication capabilities using kernel-level socket options. These were tested to confirm the transmission of timestamped packets and logging of receive-side timing, emulating a lightweight TSN data flow scenario.

The combination of NETCONF/YANG configuration, real-time reaction, and time-sensitive communication built a credible TSN prototype that aligns with industrial standards and serves as a valuable starting point for future experimentation.

## 4.1 Future Work

While this project laid a robust foundation, several areas can be expanded or improved:

- **TSN Scheduling and QoS Policies:** Integration with actual TSN switch scheduling (e.g., IEEE 802.1Qbv) using traffic shaping or Linux's Qdisc could bring real traffic control to the prototype.
- **Automation via CNC/CUC:** Extending the control plane to include a Centralized Network Controller (CNC) and Centralized User Configuration (CUC) logic would demonstrate end-to-end orchestration of flow scheduling.
- **Scalability Testing:** Deploying the system across multiple virtual or physical nodes to evaluate jitter, packet loss, and synchronization under network load.
- **GUI for Configuration:** A web or graphical frontend to interact with the YANG model and trigger configuration changes would improve usability and visualization.
- **TSN Feature Simulation:** Incorporating other TSN mechanisms like frame preemption or redundancy to enrich simulation realism.

Overall, the project achieved its core objectives while remaining modular and extensible. It provides a reusable framework for anyone aiming to explore deterministic networking using open and standards-based platforms.

## References

[1] J. L. Messenger, “Time-sensitive networking: An introduction,” *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 29–33, 2018.

[2] P. Varis and T. Leyrer, “Time-sensitive networking for industrial automation,” Texas Instruments, Tech. Rep., 2020, white Paper.

[3] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, “Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research,” *IEEE Communications Surveys and Tutorials*, vol. 21, no. 1, pp. 88–145, 2019.

[4] *IEEE Standard for Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*, IEEE Std. IEEE Std 802.1AS-2020, 2020.

[5] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks Amendment 25: Enhancements for Scheduled Traffic*, IEEE Std. IEEE Std 802.1Qbv-2015, 2015.

[6] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*, IEEE Std. IEEE Std 802.1Qcc-2018, 2018.

[7] Sysrepo Project, “Sysrepo - yang-based configuration and management framework,” https://www.sysrepo.org/, 2024, accessed: 2025-03-29.

[8] CESNET, “Netopeer2 - netconf server and cli,” https://github.com/CESNET/netopeer2, 2024, accessed: 2025-03-29.

[9] ——, “libyang - yang data modelling language parser and toolkit,” https://github.com/CESNET/libyang, 2024, accessed: 2025-03-29.