Introducing DUST: a Dataset of real-time UDP Sound packet Traces

Leonardo Severi, Matteo Sacchetto, Andrea Bianco, Cristina Rottondi
Department of Electronics and Telecommunications, Politecnico di Torino, Turin, Italy
{name.surname}@polito.it

***Abstract*—This paper introduces DUST, a dataset composed of UDP audio packet traces generated by a real-time Networked Music Performance system. The dataset includes a variety of files collected under different network and device configurations. Each file combines traces of incoming audio packets with local playback status information. The dataset aims to support offline simulations of real-time audio streaming systems, aiding the development of algorithms to address challenges such as packet loss concealment, network jitter resilience, and clock drift compensation.**

***Index Terms*—Network Traffic Traces, Real-Time Audio Streaming, Network Jitter, Clock Drift, Sound Cards.**

I. Introduction

Real-time audio streaming over the internet is employed by a wide range of applications, such as videoconferencing and Networked Music Performance (NMP) tools. While deferred audio streaming, as adopted in music on-demand services, primarily focuses on preserving the quality of the received data, real-time audio streaming imposes the additional constraint of maintaining a low latency. This constraint arises from the necessity to ensure a seamless and responsive user experience. The reference metric for this purpose is the Mouth-To-Ear (M2E) latency, defined as the time intervening from the audio acquisition by a recording device to its reproduction by a remote playout device. The recommended M2E latency for traditional videoconferencing applications is around 150 ms [2]. In the case of NMP it should instead be kept below 30 ms [6] to ensure realistic interaction among musicians and to avoid potential timing issues, such as tempo deceleration, during the performance. Therefore, designing a real-time audio streaming system poses additional challenges compared to implementing a deferred streaming system.

The basic setup of a real-time audio streaming application involves an audio source device at one location and an audio destination device at another (remote) location, interconnected via a telecommunication network such as the Internet. The best-effort design of the Internet Protocol (IP) does not permit to concurrently achieve timely and error-free data delivery, thus introducing a trade-off between these two objectives.

More in detail, factors such as network configuration, link-layer connection type (e.g., Ethernet, Wi-Fi), network congestion, and routing changes influence packet delivery timing, creating observable *network jitter*. Network jitter, defined as the variation in packet inter-arrival times, can significantly impact the performance of real-time applications. To mitigate such impact, the receiver end of a real-time audio streaming system usually features a playout buffer, an in-memory buffer designed primarily to smooth out the jitter effect.

Moreover, real-time audio streaming applications typically rely on the User Datagram Protocol (UDP) as a transport layer protocol, as it avoids retransmission and reordering delays introduced by the Transmission Control Protocol (TCP). Therefore, no guarantees are provided regarding the actual packet delivery, leading to potential late or lost packets, whose audio content cannot be reproduced in due time. Such *packet losses* deteriorate the quality of the audio playback by introducing audible artifacts. Therefore, Packet Loss Concealment (PLC) techniques aimed at mitigating the perceptual impact of missing data portions in a multimedia stream, due to either delayed or lost packets, need to be adopted.

Finally, since communication occurs between two distinct machines, two separate sound cards are involved for audio capture and playback. Despite operating at the same nominal frequency, the clocks of the two sound cards are never exactly identical. Since time synchronization strategies typically cannot be applied, this leads to the necessity of tackling the problem of clock *drift* [3].

Even though these three factors can be modelled independently, in a real scenario they influence each other. Indeed, a high jitter has the potential to cause local buffer underruns. Additionally, drift must be corrected to prevent unrecoverable overruns and underruns. Finally, packet loss can only be handled if promptly detected, i.e., if the buffer is large enough to permit the application of any recovery strategy in due time.

To the best of our knowledge, currently publicly available datasets consider multimedia streaming over HTTP of pre-recorded material (e.g., [4]) or videoconferencing applications (e.g., [5]), instead of NMP frameworks, and predominantly focus on the analysis of one of the three above-mentioned aspects (e.g., [1]), but do not permit to evaluate the intrinsic interconnection among them. Moreover, packet traces are typically constructed under the hypothesis that every packet can be either lost or correctly received, under the common assumption that a late packet is treated as a lost packet. However, this assumption does not necessarily hold in a real-time audio streaming scenario. A packet that arrives too late w.r.t. the due playout time of the first audio sample in its payload may still contain a portion of samples whose playout

time has not yet expired. In such a case, the packet content should not be entirely discarded, as part of the carried audio signal could still be reproduced in due time.

To address such shortcomings, this paper presents DUST: a dataset of network packet traces transmitted by a NMP application (i.e., MEVO [7]), which leverages UDP as a transport layer protocol. The traces were collected under varying network conditions using different audio devices, and were captured and stored at the receiver side. DUST has been created with the purpose of enabling simulations of the behavior of a real-time audio streaming application in realistic network conditions. The dataset is designed to integrate information about the playout process with audio packet arrivals captured in successive snapshots. This enables the simultaneous simulation of the reception or loss of a number of samples smaller than the packet size and an accurate quantification of the clock drift. In turn, such features permit a more thorough evaluation of the performance of state-of-the-art PLC techniques.

The remainder of the manuscript is organized as follows: Section II provides details about the dataset, the data collection method, and a description of its structure and content. Section III provides some details on how to process the dataset to quantify drift and sample losses. Finally, Section IV concludes the paper.

II. The Dataset

A. Data collection method

Data has been collected leveraging a modified version of the MEVO system. The system features a Raspberry Pi 4B (RPI) as computing device and an off-the-shelf professional sound card. The RPI is connected to the Internet by means of a wired Ethernet connection. The RPI runs the Raspberry Pi OS with the Linux PREEMPT_RT kernel. The MEVO software running on it controls the sound card through the in-kernel ALSA driver and sets the hardware parameters of both the capture and playback interface to the lowest possible hardware buffer size, and the sampling rate to 44100 Hz. Since sender and receiver features can be independently set, in the following we will describe the behavior of the two processes separately, from a software perspective.

1) Sender process: As soon as a chunk of audio is ready to be captured, it is read and stored in a small memory buffer by the capture thread `TC`. According to a given policy, after a sufficient number of samples (usually between 30 and 132) have been collected, a second thread, sender thread `TS`, stores them in a UDP packet, using a simple custom application layer protocol, and sends it to the remote receiver process.

2) Receiver process: Upon reception of a packet from the network, network thread `TN` extracts audio samples from it and enqueues them in the playout buffer. Playout thread `TP` constantly reads audio samples from the playout buffer and outputs them through the ALSA driver. For the sake of the data collection, the behavior of `TN` has been modified to collect a snapshot of the playout buffer state and of the incoming packet when enqueueing it. The snapshot collection process has been designed to be computationally lightweight, to minimize its impact on the measured quantities. `TN` writes the snapshot to memory, and another (logger) thread `TL` is in charge of batch writing the snapshots to mass storage as lines of a CSV file (see Fig. 1). During the data collection process, the audio has been monitored at regular intervals to guarantee it was correctly audible at both sides.

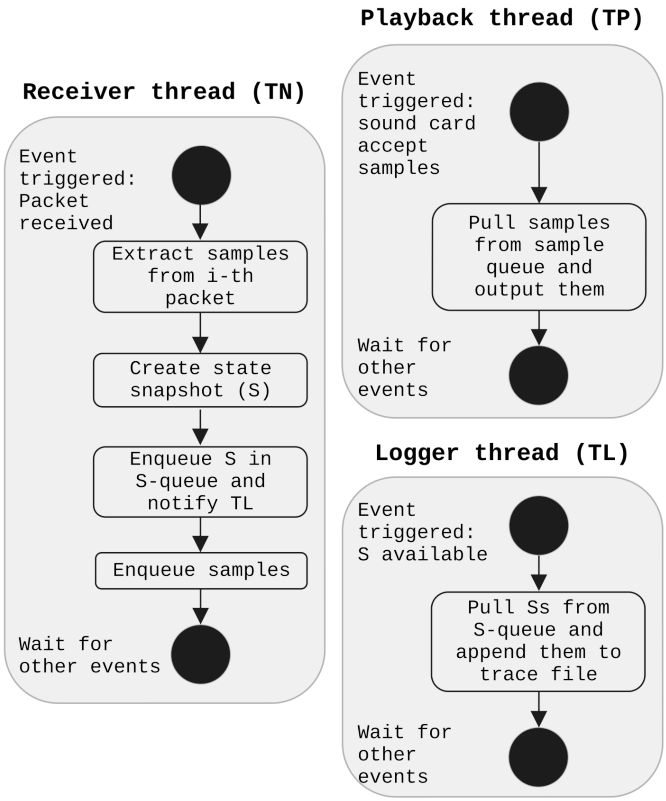

Fig. 1. Data collection diagram

B. Dataset Structure and Content

The dataset is a collection of CSV files containing one record for each received packet. Each line consists of three columns:

- **Read index**: the number of samples read (consumed) from the buffer by `TP`.
- **Send index**: the index of the first sample in the packet being enqueued by `TN`.
- **Timestamp**: a local timestamp measured in nanoseconds, according to the RPI real-time-clock, relative to the first reception event by `TN`.

No data cleaning has been performed. All initial values are set to 0. For each experiment, three files are generated: a pair of CSV files, adhering the aforementioned format, produced by the two communicating devices, where each device acts both as sender and receiver, plus a third file that offers a high-level description of the experiment by providing the following details:

- The models of the sound cards used

- The locations of the experiment
- The duration of the experiment, expressed in hours (ranging from 1 to 9)
- The type of the *last mile* Internet connection
- Further comments

At the time of writing, three different sound cards were used:

- Behringer UMC404HD
- Scarlett 2i2 (1st generation)
- Audient ID14 MKII

Experiments have been performed through non-exhaustive combinations of network conditions. We combined a Fiber-To-The-Home (FTTH), Fiber-To-The-Cabinet (FTTC) and an enterprise network. Additionally, we collected data on a direct Gigabit-Ethernet connection between the two RPIs, using the same model of sound card at both sides to provide a baseline of the expected behavior. Some traces present behaviors which are strongly influenced by a third agent (e.g., another device concurrently sharing the same network access link).

III. Information extraction and utilization

A. Clock Drift and Jitter Characterization

Each file enables the extraction of the drift between the devices' clocks and the characterization of jitter. The drift d is generally expressed in parts-per-million (ppm). For audio devices, drift can be measured using the number of samples as a time indicator. Given a file of length N with remote device D_R and local device D_L, a possible simple definition of the average drift of the clock of D_R w.r.t. the clock of D_L can be given by the following formula:

$$d = 10^6 \left(\frac{S_{N-1}}{R_{N-1}} - 1 \right) \tag{1}$$

with S and R being respectively the *send index* and the *read index*, and their subscripts indicating the packet number (starting from 0). Assuming a certain behavior of d (e.g., supposing that it remains constant during the experiment), it is possible to define a function f such that $\overline{S_i} = f(R_i)$ with $\overline{S_i}$ being the estimated value of S_i considering the drift. If d and f are representative of the behavior of the two devices, then it is possible to use the value $\epsilon_i = S_i - \overline{S_i}$ to characterize the jitter observed by the receiver.

B. Sample Loss Simulation

Identifying lost samples in the audio playout is pivotal for the assessment of PLC methods. When referring to audio streams, sample losses can be generated by three possible causes:

- *Hardware/Software (machine) faults* at sender side, for instance caused by a buffer overrun/underrun due to processor overload;
- *Packet drops* introduced by intermediate routers in the telecommunication infrastructure, when using a non-reliable transport protocol, i.e., UDP;
- *Latency spikes*, meaning that a chunk of audio samples, encapsulated in one or more consecutive packets, arrives at the destination significantly later than expected.

Publicly-available datasets used for packet loss simulation typically identify packet loss by assigning a boolean value to each packet to indicate whether it was correctly received. Differently, the information collected in DUST makes it possible to simulate more refined scenarios, e.g., where a portion of the data carried by a late packet can still be used for audio playout. For instance, suppose a packet carrying 128 samples being 1 ms late w.r.t. its expected playout time. In this scenario, supposing that the sampling rate is 44.1 kHz, the application could still be able to reproduce $128 - \lceil 1 \cdot 44.1 \rceil = 128 - 45 = 83$ samples contained in such packet. This may have a non-negligible impact on the performance of a PLC technique.

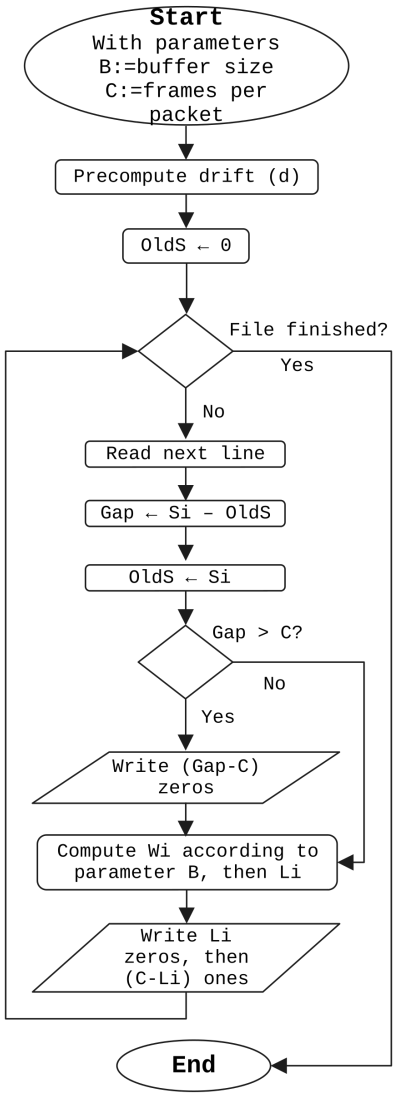

Fig. 2. Example flow-chart for a sample loss simulator, given a file of DUST and parameters B (buffer size) and C (samples per packet). An output value of zero marks a lost sample.

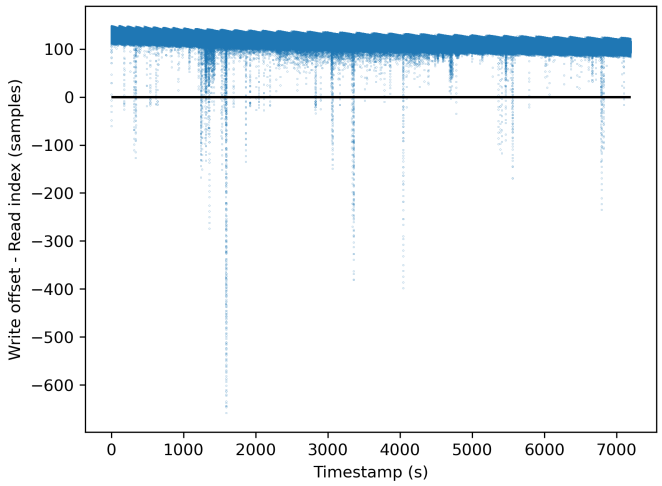

Fig. 3. Example of the evolution of $W_i - R_i$ over time, assuming $B = 128$. Negative values indicate the presence of lost samples.

To use DUST for sample loss simulation, a dedicated simulator must be created. The simulator would produce a bit-stream (a stream of boolean values) associated to a given file of DUST. Each value of the bit-stream would mark the correct/wrong reception of the corresponding sample. Fig. 2 provides an illustrative example of how the simulator could produce the bit-stream. We highlight that the simulation output is not unique, as it is strongly dependent on the buffering and drift-correction policies implemented in the simulator. We can safely assume that there exists a controller that applies a correction to the packet's *send index* S_i, incorporating the buffering policy and the drift correction. A simple approach to implement such controller may assume a fixed user-defined target value for the buffer size (in number of samples) and the presence of an oracle that provides a bias B to be applied to S_i based on the elapsed time and the prior knowledge of the clock drift d between the two sound cards. In Eq. 2, we show how to obtain a value W_i, defined as the *write index* of the incoming i-th packet, from S_i with the parameters B and d (expressed in ppm).

$$W_i = B + \frac{S_i}{(1 + \frac{d}{10^6})} \tag{2}$$

W_i is directly comparable to R_i such that

$$L_i = \begin{cases} 0, & \text{if } W_i \geq R_i \\ \min(R_i - W_i, C), & \text{otherwise} \end{cases} \tag{3}$$

with C being the number of samples per packet and L_i the number of initial lost samples of i-th packet. Fig. 3 shows an example of the behavior of $W_i - R_i$ assuming $B = 128$. We also assume that there are no out-of-order packets; otherwise, the file must be sorted in ascending order by the *send index* column before being fed into the simulator.

It is worth mentioning that the above described approach does not take into consideration potential machine faults, therefore, even though applicable to the files included in DUST, it does not have general validity.

IV. Conclusions

This paper presented DUST, a dataset containing UDP audio traces collected from the communication between two devices running a NMP system over an IP network. The dataset encompasses different scenarios, considering various network configurations and audio devices. DUST aims to be a support tool for research on packet loss concealment, drift estimation and jitter characterization. Even though the dataset has been created with NMP as reference application, we acknowledge its potential for broader utilization, even under less restrictive latency requirements (as, e.g., in the case of videoconferencing systems). Therefore, in the future we aim to further expand DUST by incorporating new traces collected in wireless network scenarios, including 5G networks. The dataset is available at https://doi.org/10.5281/zenodo.12726670.

Acknowledgment

This work has been partially supported by the Italian Ministry for University and Research under the PRIN program (grant n. 2022CZWWKP). Leonardo Severi's PhD Programme is funded by the European Union in the framework of the Resiliency and Recovery Plan (RRP), within the NextGenerationEU initiative.

References

[1] Lorenz Diener, Solomiya Branets, Ando Saabas, and Ross Cutler. The icassp 2024 audio deep packet loss concealment grand challenge. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing*, April 2024.

[2] ITU-T Rec. G.114. One-way transmission time. ITU, May 2003.

[3] Christoffer Lauri and Johan Malmgren. Synchronization of streamed audio between multiple playback devices over an unmanaged ip network. *Department of Electrical and Information Technology, Faculty of Engineering, LTH, Lund University, Sweden*, 2015.

[4] Stefan Lederer, Christopher Müller, and Christian Timmerer. Dynamic adaptive streaming over http dataset. In *Proceedings of the 3rd multimedia systems conference*, pages 89–94, 2012.

[5] Babak Naderi, Ross Cutler, Nabakumar Singh Khongbantabam, Yasaman Hosseinkashi, Henrik Turbell, Albert Sadovnikov, and Quan Zou. Vcd: A video conferencing dataset for video compression. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3970–3974. IEEE, 2024.

[6] Cristina Rottondi, Chris Chafe, Claudio Allocchio, and Augusto Sarti. An overview on networked music performance technologies. *IEEE Access*, 4:8823–8843, 2016.

[7] Leonardo Severi, Matteo Sacchetto, Andrea Bianco, Cristina Rottondi, Aleksandra Knapinska, and Piotr Lechowicz. Demonstration of a networked music performance experience with mevo, 2024.