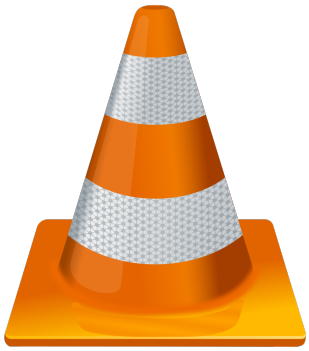

Project 4.6

Add back netsync module

VideoLAN

Google Summer of Code 2026

Arshidha S

Indian Institute of Technology, India

me24b089@smail.iitm.ac.in

arshidha.817@gmail.com

GitHub: arshidha817

LinkedIn: arshidha-s

IRC: Arshidha

1 Project Synopsis

VLC Media Player's legacy network synchronization module (`netsync`) allowed multiple instances to synchronize media playback over a network. It relied on custom 16-byte raw UDP packets and shifted playback time by manipulating the Program Clock Reference (PCR) via functions like `GetPcrSystem()` and `input_ModifyPcrSystem()`.

In VLC 4.0, the core clock is completely re-architected into a Master-Slave model. This new system replaces direct PCR manipulation with an affine function (`system = ts * coeff / rate + offset`) to map stream time to system time. In this design, a single Master Clock acts as the "setter" defining the function's slope, while Slave Clocks] act as "getters".

This project aims to restore netsync to VLC 4.0 by executing the following core tasks:

1. **Protocol Upgrade:** Replacing the legacy UDP protocol with RTP MIDI protocol (RFC 6295) for robust clock drift handling.
2. **Core API Extension:** Extending the `vlc_clock_main_t` API within `src/clock/clock.c` to safely support Master Clock overriding.
3. **Module Integration:** Plugging this time-sync logic into a new `netsync` control module capable of acting as the Master Clock.

2 Benefits to the VideoLAN community

Restoring the `netsync` module fulfills a crucial use case for modern media consumption: multi-device synchronization (e.g., routing audio to a separate laptop connected to a surround-sound system while watching video on a main projector).

Furthermore, by moving away from custom UDP packets to RTP MIDI (RFC 6295), VLC will adopt a robust, RFC-standardized way of handling network clock drift. Finally, by correctly implementing `netsync` as an external "Master Clock", this project will validate and finalize the clock architecture for VLC 4.0, paving the way for other external synchronization protocols in the future.

About Me

I am a second-year undergraduate student at the Indian Institute of Technology, Madras. My journey into software engineering began with a strong fascination for problem-solving, which quickly evolved into a passion for building robust, efficient systems. I have dedicated myself to mastering computer science fundamentals, firmly believing that analytical engineering skills translate perfectly into writing innovative, real-world software.

Regarding this project, I have got strong foundation in C, Linux and C++. Working extensively with C and C++ has taught me the power of systems programming, memory management, and the Standard Template Library (STL). I am deeply curious about how low-level software interacts with hardware and operating systems. As a regular user of VLC, the opportunity to dive into its core architecture and network synchronization is a primary reason I chose this project. Furthermore, I have hands-on experience with tools like Docker, Git, and Linux environments which may come handy along this project.

While I am eager to make my first step in open-source development through Google Summer of Code, I have actively honed my technical and debugging skills through various challenges. For instance, tackling the OverTheWire Bandit Wargame significantly strengthened my Linux command-line skills and my systematic problem-solving abilities. I also actively participate in competitive programming, such as Codeforce contests and the Advent of Code.

During my academic journey, I have built several projects heavily focused on performance optimization and search algorithms. I developed "Poozle," a C++ based word-frequency analysis tool where I utilized regex for parsing and implemented both single-threaded and multithreaded versions to actively compare runtime and memory usage. Additionally, I have experimented with Gale-Shapley algorithms to solve complex Stable Matching models. These experiences have given me a practical understanding of project architecture and performance profiling.

Feel free to check my GitHub to explore my projects in more detail: arshidha817

3 Deliverables

Guided by my potential mentor Thomas Guillem, the following are the deliverables that need to be done for the completion of the subject:

https://code.videolan.org/videolan/vlc/-/issues/27577

3.1 Implement Protocol Parser: RTP MIDI

Using VLC's internal networking helpers, the module will establish a network listener to ingest incoming RTP payloads. It will parse the RTP headers to extract 32-bit timestamps and apply standard NTP-style mathematical formulas to compute precise network latency and hardware clock offsets.

3.2 Core API Extension: Master Clock Override

To apply the calculated offsets, the module must safely interact with the core engine. The `netsync` module will natively claim the Master Clock role within the core engine. It will act as the authoritative "setter" of the synchronization affine function. By operating as the master, the module ensures that all local audio and video streams function as slaves, adjusting their playback timings to match the network master without disrupting the core synchronization loop.

Currently, `vlc_clock_main_CreateMaster()` asserts `main_clock->master == NULL`, causing a crash if the local audio output has already initialized as the master. To resolve this, I propose implementing a safe override function in `src/clock/clock.c` that safely demotes the existing master to priority 1 (slave priority) and promotes the new `netsync` clock. A draft of this implementation is as follows:

```
1 vlc_clock_t *vlc_clock_main_OverrideMaster(vlc_clock_main_t *main_clock
    , const struct vlc_clock_cbs *cbs, void *cbs_data) {
2     /* The new master has priority 0 */
```

```
    vlc_clock_t *clock = vlc_clock_main_Create(main_clock, 0, cbs,
    cbs_data);
    if (!clock) return NULL;

    vlc_mutex_lock(&main_clock->lock);

    if (main_clock->master != NULL) {
        /* Demote existing master */
        vlc_clock_set_slave_callbacks(main_clock->master);
        main_clock->master->priority = 1;
    }

    vlc_clock_set_master_callbacks(clock);
    main_clock->master = clock;
    main_clock->rc++;

    vlc_mutex_unlock(&main_clock->lock);
    return clock;
}
```

Exposing the API and Linker Symbols
This function prototype will be exposed in `include/vlc_clock.h` and exported in `src/libvlccore.sym` to allow dynamic control modules to link against it at runtime.

3.3 Module Integration: The netsync Loop

The `modules/control/netsync.c` initialization sequence will invoke `vlc_clock_main_OverrideMaster()` to claim control. A dedicated background thread will then feed the computed drift offsets (from Task 1) into the main clock's update callbacks, ensuring seamless, continuous synchronization.

4 Tentative Timeline

- **Before March 31st (Proposal Deadline):**
  - Continually improve my proposal based on feedback from my mentor to refine the presentation of my subject.
- **April 1st - April 30th (Application Review Period):**
  - Increase my familiarity with RFC 6295 (RTP MIDI) and VLC's RTP access modules.
  - As local compilation is done, I'll deeply study the new clock architecture (`src/clock/clock.c`), documentation (`doc/clock.md`), and the previous `netsync.c` implementations.
  - Work on and provide patches to VLC Media Player to get accustomed to its Git workflow and rules of contribution.
- **May 1st - May 24th (Community Bonding Period):**
  - Communicate with my mentor and finalize the deliverables' goals, feasibility, and scope.

  - Prepare a detailed work breakdown structure according to the deadlines assigned by my mentor.
  - Interact with the VLC community and continue sending minor patches.

- **May 25th - June 7th (Coding officially begins - Task 1):**
  - Communicate with my mentor about the plan to start working on the RTP MIDI protocol implementation.
  - Write the network listener loop and RTP header parsing logic for the future module.
  - Test payload extraction and NTP-style offset calculations.
  - Provide updates to my mentor regarding the progress.

- **June 8th - June 21st (Task 2):**
  - Inform my mentor regarding the completion of the network parsing implementations.
  - Start modifying the core engine API in `src/clock/clock.c` to implement `vlc_clock_main_OverrideMaster`.
  - Expose the API in `include/vlc_clock.h` and update `libvlccore.sym`.
  - Test for core regressions and document the core API changes.

- **June 22nd - July 12th (Task 3):**
  - Begin work on the final deliverable: plugging the core APIs into `modules/control/netsync.c`.
  - Implement the background threads to continually update the new master clock's affine function based on the incoming RTP payloads.
  - Test the synchronization extensively between two local VLC instances.
  - Continue the modifications with tests and documentation.

- **July 13th - July 26th:**
  - Finish the task on the new module and notify my mentor about the completion.
  - Fix synchronization edge cases (e.g., pausing, seeking, or massive network jitter).

- **July 27th - August 2nd:**
  - Provide a final update to my mentor and ask for feedback.
  - According to feedback, work on any other remaining tasks, bugs, or documentation related to the subject.

Note: I have spared an extra 2-3 weeks for tasks that may take longer than estimated (the GSoC 2026 timeline officially allows extensions up to November). Also, my college semester exams are scheduled for May for 2 weeks. During that period, I may work for fewer hours. Apart from that, I am fully available to contribute daily for the required number of hours.