4. Project 2. Manage Multiple Access

Important

Please read the following instructions carefully:

1. This project is to be completed by each group individually. Use your own code to complete the main parts of the tasks.
2. This project contains 7 tasks. 2 of them are optional (12 points + 6 points).
3. The task score does not reflect the implementation difficulty but rather the recommended level.
4. Suggested workload is 6~8 FULL days. Manage your time.
5. Submit your code through Blackboard. The submission due date is **Nov. 17, 2024**.
6. Each group needs to submit the code once and only once. Immediately after TAs' checking. The submission is performed by one of the group members.
7. Tasks with "Optional" tag are optional tasks. Their due date is the end of the semester. Optional tasks are graded based on performance criteria and the results of the code review.
8. Unless otherwise mentioned, the instructor is responsible for grading optional tasks. Contact the instructor to check if you have finished one or more of them.
9. *Task 5 and 6* are graded by TAs.
10. Tasks are graded according to their hierarchy. The hierarchy of this project is *Task 1* < *Task 2* < *Task 3* < *Task 6*, and *Task 1* < *Task 2* < *Task 3* < *Task 4*. A full score of one task automatically guarantees the full score of non-overdue preceding tasks (left side of "<").
11. Free to use any tools for debugging, but only the provided toolkit is permitted in performance assessment.
12. During the performance assessment, any task can only be attempted up to 5 times.

4.1. Overview

This project aims to enable multiple devices to share the audio medium for relatively reliable data transmission, building upon the results of Project 1. To simplify the physical layer implementation, network devices will no longer be connected through sound but via audio cables. These cables carry analog electrical signals driving the loudspeakers, acting as an "electromagnetic mirror" of the actual sound. It is worth noting that such wired connections are similar to Ethernet, making the CSMA/CD mechanism an good reference for this project.

However, unlike Ethernet, there is a significant latency (~10 ms) when using the sound card for carrier sensing. This latency arises because the sound card is not designed for this purpose, and we have to process carrier information in the digital domain. This delay reduces the efficiency of CSMA and, more importantly, prevents the implementation of an effective collision detection mechanism. Therefore, we will use ACK to handle error frames. In this sense, the access mechanism is more akin to what is adopted in wireless situations like Wi-Fi. Figure 4.1 illustrates the program architecture of this project.

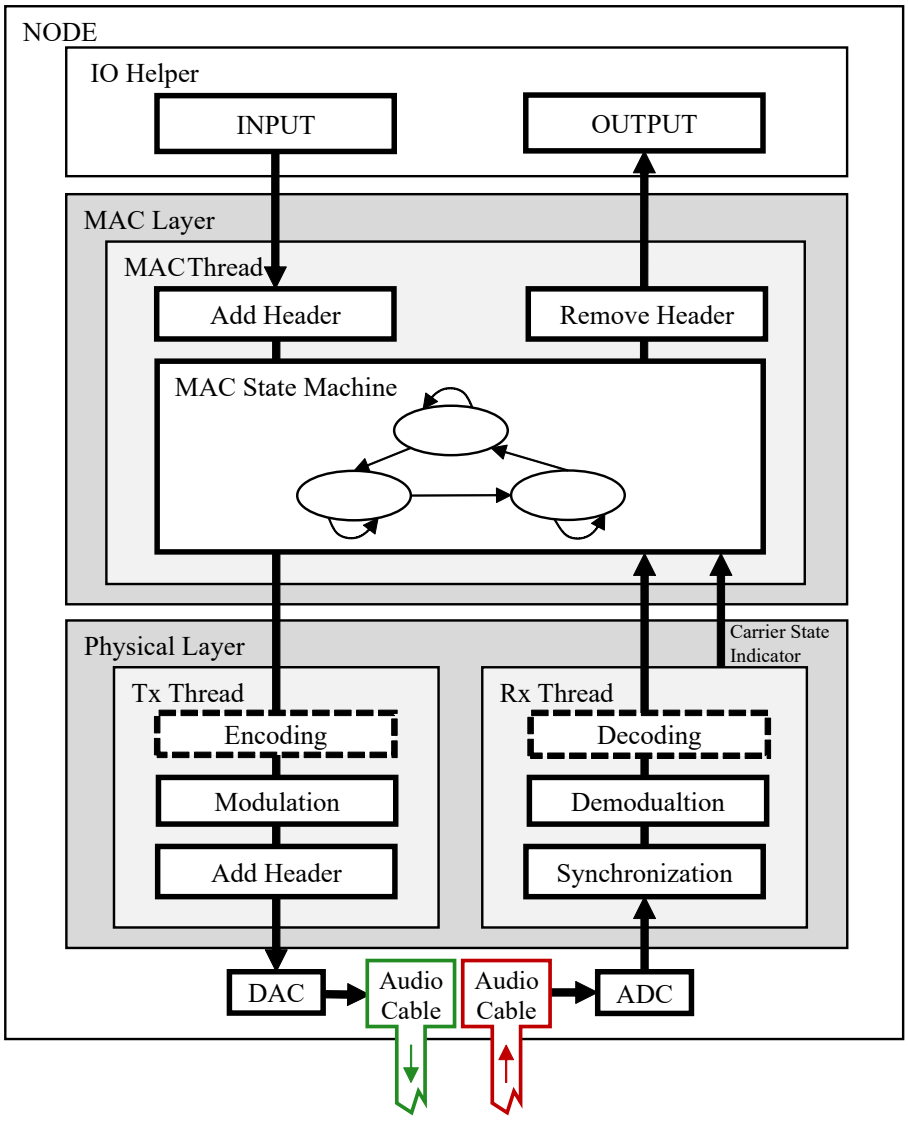

Figure 4.1. Project 2 Overview.

4.2. Task 1: (4 points) Cable Connection

The ADC and DAC of the USB sound card are directly connected to the input and output connectors, so the signals flowing between connected computers are analog electrical signals (observable via an oscilloscope). The signal intensity corresponds to the intensity of the sound waveform. Compared to actual sound, electrical signals are quiet, quick, and clean. They undergo minimal attenuation, deformation, multipath, ring effects, etc., factors that can lead to noise in the ideal waveform during cable transmission (sampling frequency offset may still exist). This means a faster and more stable physical layer should be achievable.

This task revisits Project1.Task3. While there are equivalent connection topology, please prioritize using the one in Figure 4.2, as it can smoothly transition to other tasks. The mixer functions like an Ethernet hub, allowing all connected devices to hear each other. Connecting through hubs provides better scalability and convenience. The provided audio mixer can connect 4 computers together.

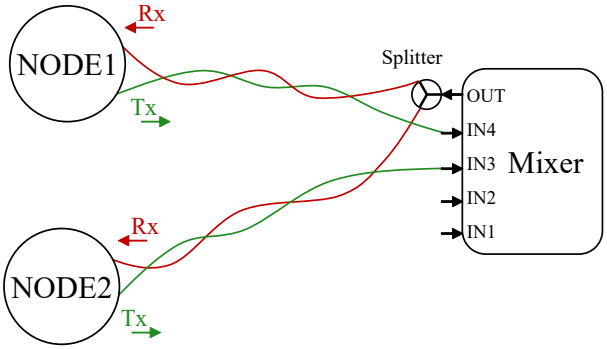

Figure 4.2. Connecting Two Nodes.

Tip

- Project 1 uses carrier waves to convey information over the air, but it is no longer necessary for cable connections. You might want to simplify and speed up the physical layer with line coding. A raw rate of at least 12 Kbit/s is suggested to complete the projects with ease.
- You can design a shorter preamble to reduce the header overhead.
- It is recommended to unplug the power adapters of your computers when connecting them. The reason is that different AC-to-DC adapters, especially those without grounding, may cause non-zero voltages in the device's "ground". In some cases, this issue can lead to biased DC levels in the received signals. Occasionally, it may even cause permanent damages. Be very careful if you have to connect two devices with different "ground", such as a charging laptop and a desktop computer.
- Signals in the cables may be susceptible to electromagnetic interference from other electronic devices or even adjacent cables. Please maintain a tidy testing environment and keep the cables away from potential high-powered electromagnetic devices, e.g., power adapters.

Performance Assessment

The group provides two devices: `NODE1` and `NODE2`, and connects them with the toolkit according to Figure 4.2 (the mixer's connecter IDs are interchangeable).

- Objective (4 points). TAs provide a binary file `INPUT.bin`, which contains 6,250 Bytes. `NODE1` sends the bits from the file to `NODE2`. `NODE2` stores the received bits in `OUTPUT.bin`.

  Transmission must be completed within 20 seconds:

Completion Time	Percentage Earned
<10 s	100%
<15 s	75%
<20 s	50%
>20 s	0%

  TAs use `diff` tool to compare `INPUT.bin` and `OUTPUT.bin`:

Similarity	Percentage Earned
<80%	0%
<95%	80%
>95%	100%

4.3. Task 2: (5 points) Acknowledgement

The cable-based physical layer can dramatically reduce transmission errors, but errors cannot be completely avoided. For example, non-real-time scheduling of the ASIO process by the operating system may occasionally lead to audio samples missing. More importantly, when multiple nodes share the same cable, decoding errors caused by collisions are almost inevitable. This task will utilize the unreliable data link posed by the physical layer to achieve a much higher level of transmission reliability by using ACKs.

The logic of ACK is straightforward: the sender keeps re-/transmitting until it receives the receiver's acknowledgement of correct reception. The state machine of a simple ACK protocol is shown in Figure 4.3, where each circle represents the working state of the MAC thread. The arrows between states highlight events triggering state transitions (uppercase events are from the physical layer) and the associating operations (`>>`). In the `Rx Frame` and `Tx Frame` states, the MAC thread invokes the data transmission and reception functions provided by the physical layer.

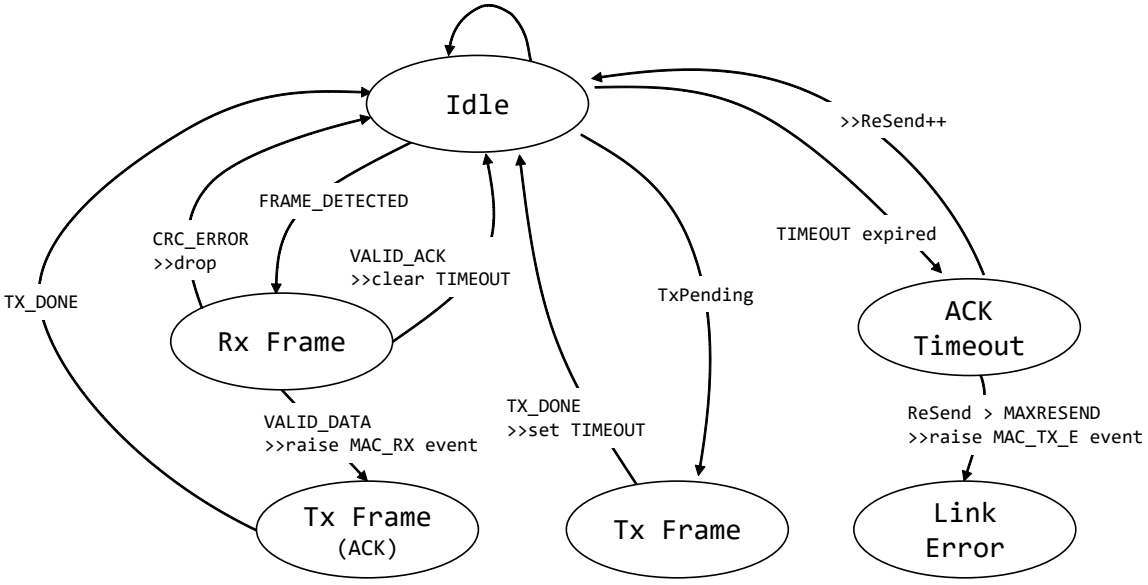

Figure 4.3. Example: State Machine of ACK Protocol.

Specifically,

- The upper-layer application appends a `DATA` frame to the transmission buffer and raise the `TxPending` flag. When the MAC thread is idle, it sends the `DATA` frame and sets a timer with a duration of `TIMEOUT`.

  If no `ACK` frame is received before `TIMEOUT` expires, the sender retransmits the frame.

If the maximum number of retransmissions is reached, the MAC terminates and raises an error.

- The MAC thread continues to receive/detect frames while idle.

  If the received frame is corrupted, it is discarded.

  If the received frame is error-free, then

  If it is an `ACK` frame, indicating the previously sent `DATA` frame was received correctly, the `TIMEOUT` timer is cleared to prevent retransmission.

  If it is a `DATA` frame, the received content is appended to the reception buffer. The MAC thread reports to the upper-layer, and sends an `ACK` frame immediately.

In the above protocol, it is important for the receiver to be able to distinguish between `DATA` frames and `ACK` frames. This is because, although `ACK` frames are sent by the receiver itself, due to driver delays (typical round-trip time for a USB sound card using ASIO with 128 buffer size is 15 to 20 ms), the sender might receive its own `ACK` while it is in the idle state. Therefore, the protocol requires adding auxiliary information to the frames, i.e., the MAC layer header. For example, in Figure 4.4, the `Type` field can be used to incorporate a flag to differentiate between `ACK` and `DATA`. Note that the aforementioned issue can also be resolved by specifying the receiver ID, i.e., the MAC address, in the frames. MAC addresses provide other essential functionalities, which will become apparent in the subsequent tasks.

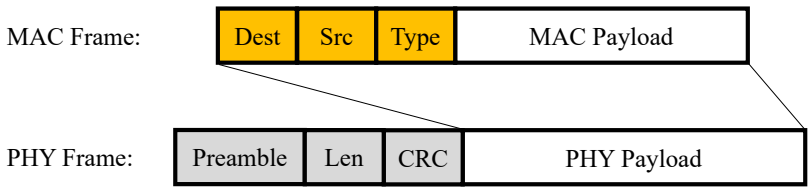

Figure 4.4. Example: MAC Layer Frame Structure.

Tip

- The length of the MAC address depends on the number of devices using the protocol. It can be shortened to fit the actual scale and reduce protocol overhead.

Performance Assessment

The group provides two devices: `NODE1` and `NODE2`, and connects them with the toolkit according to Figure 4.2 (the mixer's connecter IDs are interchangeable).

- Objective 1 (4 points). TAs provide a binary file `INPUT.bin`, which contains 6,250 Bytes. `NODE1` sends the bits from the file to `NODE2`. `NODE2` stores the received bits in `OUTPUT.bin`. This part's assessment criteria and procedures are similar to Task 1, with modifications made to the completion time and accuracy considering the protocol overhead and effectiveness of the ACK mechanism.

  Transmission must be completed within 40 seconds:

Completion Time	Percentage Earned
<20 s	100%
<30 s	75%
<40 s	50%
>40 s	0%

TAs use `diff` tool to compare `INPUT.bin` and `OUTPUT.bin`:

Similarity	Percentage Earned
<100%	0%
100%	100%

- Objective 2 (1 point). Repeat the above file transfer with the same implementation. The TAs will unplug one of the cables and start timing until the transmitter raises the "link error" warning. This delay is recorded as the detection delay.

Detection Delay	Percentage Earned
<5 s	100%
<8 s	50%
>8 s	0%

4.4. Task 3: (2 points) Carrier Sense Multiple Access

If both nodes in Task 2 want to send data to each other, the ACK protocol alone will no longer be sufficient. This is because both nodes will directly send packets at will, leading to transmission collisions. To allow multiple nodes to share the transmission medium, MAC (Medium Access Control) protocols can be used. This task will implement the CSMA (Carrier Sense Multiple Access) protocol, which has been adopted in early Ethernet and many current wireless networks. CSMA can be described as listen before transmit (carrier sensing) and wait for a random period (backoff) after a collision occurs before attempting to retransmit.

CSMA implementations of different level of aggressiveness differ in the backoff strategy. One simple strategy is shown in Figure 4.5: the node performs medium sensing before each transmission attempt. If the medium is found busy or if a transmission error occurs, the node waits for a random period before attempting to transmit again.

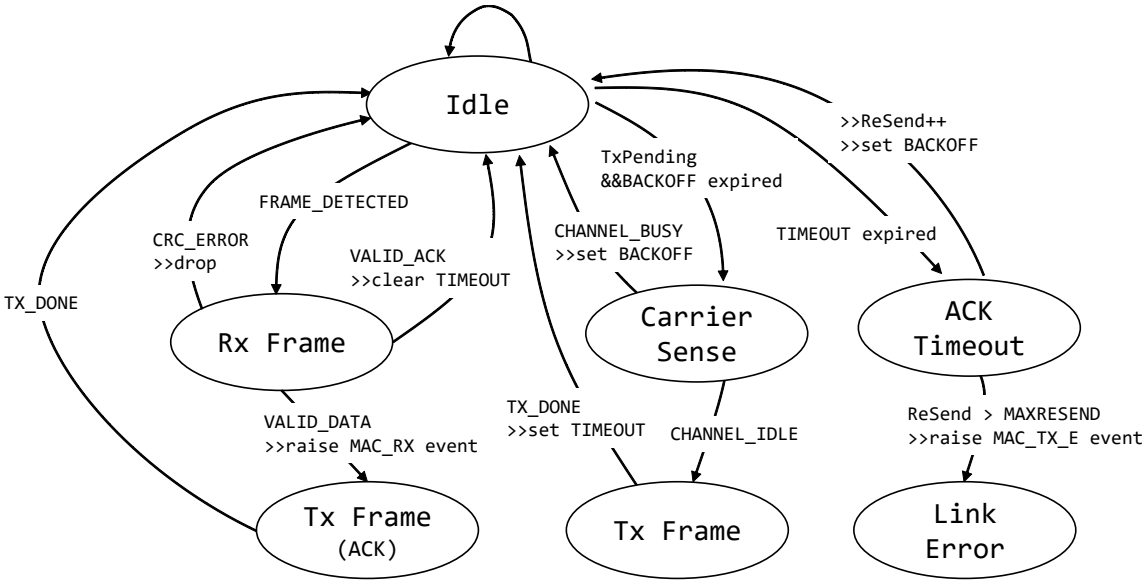

Figure 4.5. Example: State Machine of CSMA Protocol.

Specifically,

- After sending `DATA` and not receiving an `ACK` before the `TIMEOUT`, set the `BACKOFF` timer to a random value.
- When the `BACKOFF` timer reaches zero, sense the medium (generally, at least longer than the frame interval between a `DATA` and the corresponding `ACK`).

  If the medium is idle, proceed with immediate transmission.

  If there are other signals on the medium, set the backoff timer to a random value and attempt transmission again.

The WARP CSMAMAC provides a detailed and clear description of a CSMA protocol (with slight differences from the above) from the implementation's perspective, which should be considered an important reference for this task.

Performance Assessment

The group provides two devices: `NODE1` and `NODE2`, and connects them with the toolkit according to Figure 4.2 (the mixer's connecter IDs are interchangeable).

- Objective (2 points). TAs provide a binary file `INPUT1to2.bin`, which contains 6,250 Bytes. `NODE1` sends the bits from the file to `NODE2`. `NODE2` stores the received bits in `OUTPUT1to2.bin`. TAs provide a binary file `INPUT2to1.bin`, which contains 6,250 Bytes. `NODE2` sends the bits from the file to `NODE1`. `NODE1` stores the received bits in `OUTPUT2to1.bin`. `NODE1` and `NODE2` start transmission simultaneously. TAs record the completion times for `NODE1` and `NODE2`'s transmissions as T1 and T2.

  Transmission must be completed within 80 seconds:

MAX(T1,T2)	Percentage Earned
<40 s	100%
<60 s	75%
<80 s	50%

MAX(T1,T2)	Percentage Earned
>80 s	0%

Their completion times should be close enough:

ABS(T1-T2)	Percentage Earned
<10 s	100%
>10 s	0%

TAs use `diff` tool to compare `INPUT*.bin` and `OUTPUT*.bin`:

Similarity	Percentage Earned
<100%	0%
100%	100%

4.5. Task 4: (1 point) CSMA with Interference

This task will apply CSMA in a more realistic environment:

- There might be more than two nodes participating in the network transmission.
- Hidden terminals could exist in the network.
- Outdated, selfish, and poorly-implemented nodes might also be present in the network.

To understand the performance of CSMA in these scenarios, a jamming source, denoted as the `Jammer`, is connected to the network, as shown in Figure 4.6. The `Jammer` does not adhere to CSMA and emits interfering signals randomly. However, the `Jammer` is not always active, allowing data transmission during its silent periods.

The `Jammer`'s behavior can be described as follows: its jamming duration follows a uniform distribution between [50, 100) ms. During the jamming period, the `Jammer` emits a white noise signal. After each jamming period, it switches to silent before initiating the next jamming, with each silent period following a uniform distribution between [100, 200) ms.

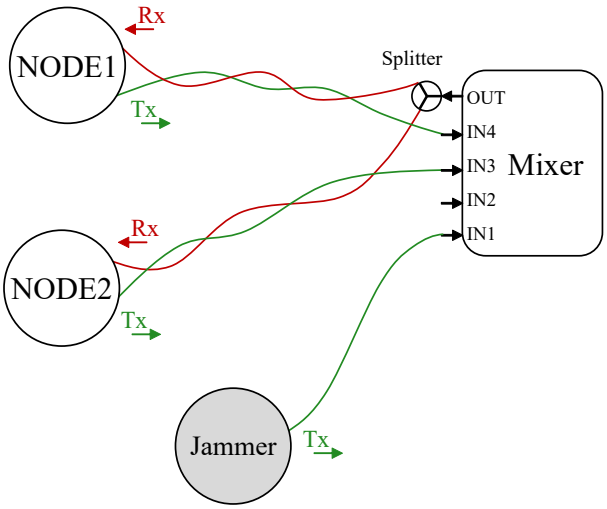

Figure 4.6. Two Nodes and a Jammer.

Performance Assessment

The group provides two devices: `NODE1` and `NODE2` and TAs provide a `Jammer`. They are connected according to Figure 4.6. The `Jammer` uses the USB sound card from the toolkit to emit jamming signals. The script for generating the jamming signal is uploaded to Blackboard. The output gains/volumes of the `Jammer` and `NODE*` operating systems are set to similar levels (± 10%).

> Transmission must be completed within 120 seconds, other assessment criteria and procedures are the same as in Task 3:

MAX(T1,T2)	Percentage Earned
<80 s	100%
<100 s	75%
<120 s	50%
>120 s	0%

Any group intending to finish Task 5 must inform the TAs explicitly during this task. TAs will repeat the transmission task five times and record the mean of MAX(T1, T2) to rank the performance.

4.6. Task 5: (Optional, 3 points) Performance Rank

A primary design objective of network systems is to enhance data transmission performance, but achieving high performance is often challenging. This task is to reward the best-performing implementation in this project.

Performance Assessment

This task is automatically graded according to the completion time of Task 4. When determining ranks, round the completion time to the nearest 5 seconds, e.g., 23.4 seconds and 20.1 seconds would be considered the same rank.

Rank	Percentage Earned
1st	100%
2nd	66%
3rd	33%

4.7. Task 6: (Optional, 3 points) X

Let's try with more nodes.

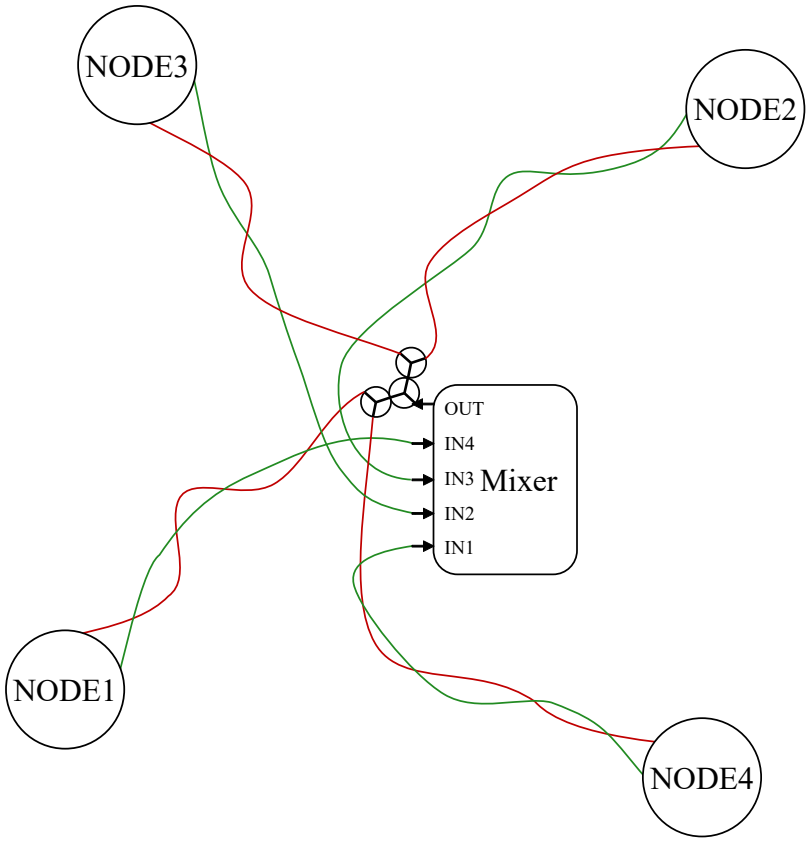

Figure 4.7. Connecting Four Nodes.

Tip

- The provided mixer is a passive mixer. Its output power depends on the number of connected input and output connectors. When connecting more or fewer devices, such as monitoring headphones, to it, the signal strength at the receiver should be reevaluated.
- You may need additional cables.

Performance Assessment

The group provides **four** devices: `NODE1`, `NODE2`, `NODE3`, and `NODE4`. They are connected according to Figure 4.7.

TAs provide a binary file `INPUT1to2.bin`, which contains 6,250 Bytes. `NODE1` sends the bits from the file to `NODE2`. `NODE2` stores the received bits in `OUTPUT1to2.bin`. TAs provide a binary file `INPUT2to1.bin`, which contains 6,250 Bytes. `NODE2` sends the bits from the file to `NODE1`. `NODE1` stores the received bits in `OUTPUT2to1.bin`.

TAs provide a binary file `INPUT3to4.bin`, which contains 6,250 Bytes. `NODE3` sends the bits from the file to `NODE4`. `NODE4` stores the received bits in `OUTPUT3to4.bin`. TAs provide a binary file `INPUT4to3.bin`, which contains 6,250 Bytes. `NODE4` sends the bits from the file to `NODE3`. `NODE3` stores the received bits in `OUTPUT4to3.bin`.

`NODE1`, `NODE2`, `NODE3`, and `NODE4` start transmission simultaneously. TAs record their completion times as T1, T2, T3, and T4.

> Transmission must be completed within 160 seconds:

MAX(Ti)	Percentage Earned
<120 s	100%
<140 s	75%
<160 s	50%
>160 s	0%

Their completion times should be close enough:

MAX(Ti)- MIN(Ti)	Percentage Earned
<15 s	100%
>15 s	0%

TAs use `diff` tool to compare `INPUT*.bin` and `OUTPUT*.bin`:

Similarity	Percentage Earned
<100%	0%
100%	100%