ECE 445

SENIOR DESIGN LABORATORY

PROJECT PROPOSAL (REVISION)

Arduino-Powered Network Flow Visualization Toolbox

Team #17

BOLIN ZHANG
(bolinz3@illinois.edu)
JIAHAO FANG
(jiahaof3@illinois.edu)
YIYANG HUANG
(yiyangh5@illinois.edu)
ZIYUAN CHEN
(ziyuanc3@illinois.edu)

TA: TBD

March 26, 2024

Contents

1 Introduction **1**
1.1 Problem Statement 1
1.2 Solution Overview 1
1.3 High-Level Requirements 2

2 Design **3**
2.1 Intermodular Protocol 3
2.2 “Node” PCBs 6
2.3 “Link” PCBs 8
2.4 “MUX” PCB 9
2.5 Arduino Controller 9
2.6 Graphical User Interface 9
2.7 Outer Packaging 10

3 Reliability, Ethics, and Safety **11**
3.1 Tolerance Analysis 11
3.2 Ethical Concerns 12
3.3 Safety Concerns 12

References **13**

List of Figures

1 Pictorial representation of the physical model. 1
2 Subsystem organization of top-level design entity. 3
3 Intermodular communication interface signals. 3
4 Subsystem organization of top-level design entity. 5
5 FSM state transition diagram for the Node PCB. 7
6 FSM state transition diagram for the Link PCB. 8

List of Tables

1 Signal flow example for polling network configurations in Figure 4. 5
2 Signal flow example for writing network configurations in Figure 4. 6
3 FSM state actions for the Node PCB. 7
4 FSM state actions for the Link PCB. 8

1 Introduction

1.1 Problem Statement

Many real-world systems involve flows over networks. Logistic systems, transportation networks, and the Internet are all carefully designed to meet the capacity and cost requirements. However, in algorithm courses, flow optimization problems can be hard to imagine. Students often struggle to quantitatively predict how each twerk in the constraints will affect the optimal solution *using only intuition*.

Meanwhile, the existing tools for visualizing network flows are mainly software-based, e.g., the node flux and link capacity values are configured through a computer GUI, and the physical model simply displays the optimal flow. Having a *hardware-oriented* model can provide a more intuitive sense of "twerking" the network by assigning a knob to each parameter and a strip of LEDs to each link. Such a model also has the potential to *dynamically* visualize more complex scenarios in realistic flow management, such as the presence of routing hubs, congestion, and packet delay.

1.2 Solution Overview

Our team aims to build a *modular, reconfigurable* hardware emulator to visualize network flows under capacity constraints on links. Each node can be configured as a source, a sink, or a "transfer station" that holds zero flux. Solutions will be computed using an embedded Arduino microcontroller. This toolset will provide an intuitive visual aid and facilitate the understanding of flow algorithms in a classroom setting, especially where the network in discussion is inherently *dynamic* (e.g., routing packets in the Internet).

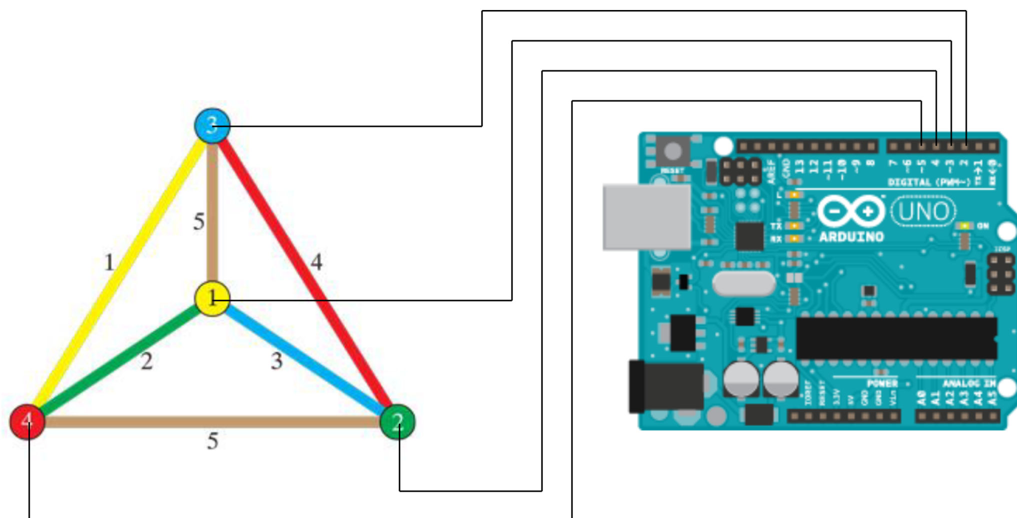

Figure 1: Pictorial representation of the physical model.

Meanwhile, we are making several assumptions that simplify the problem:

- This is a small-scale network with a maximum of 6 nodes and up to 3 links per node.
- The nodes don't have buffers and is able to respond instantaneously to changes in capacity constraints.
- All links are bidirectional, and both directions have the same capacity.

1.3 High-Level Requirements

- The physical model should be modular, i.e., each node has a number of "slots" reserved for installing new links. We aim to serve 6 partially connected nodes.
- The Arduino software should communicate with all nodes and pipes and update the flows in real-time (within 500ms) in response to changes in setup.
- The algorithm should handle and report edge cases such as a network with zero or multiple feasible flows.

2 Design

In the physical network model, pipes represent links attributed with **link capacity**, and the LEDs within show the real-time **link flux** of "data packets." Each node is a PCB board with a knob that specifies the **node flux**, configuring the node as source (positive), sink (negative), or "transfer station" (zero). We use a scalable design where components are easily replaceable to account for network expansion.

The embedded Arduino board implements the Ford-Fulkerson algorithm that efficiently computes network flows while considering all constraints (node flux and link capacity). A software GUI displays the solution alongside the physical model due to limited space (number of LEDs and pins for interconnection) in each node and link.

Figure 2: Subsystem organization of top-level design entity.

2.1 Intermodular Protocol

The following diagram details the interface between the Arduino controller, the node PCBs, and the link PCBs. Note that links are *not* directly connected to the Arduino board, and thus the capacity values configured by the knobs must be passed indirectly through the nodes. Moreover, we incorporate a MUX as an extra layer of abstraction to account for the limited number of I/O pins on Arduino.

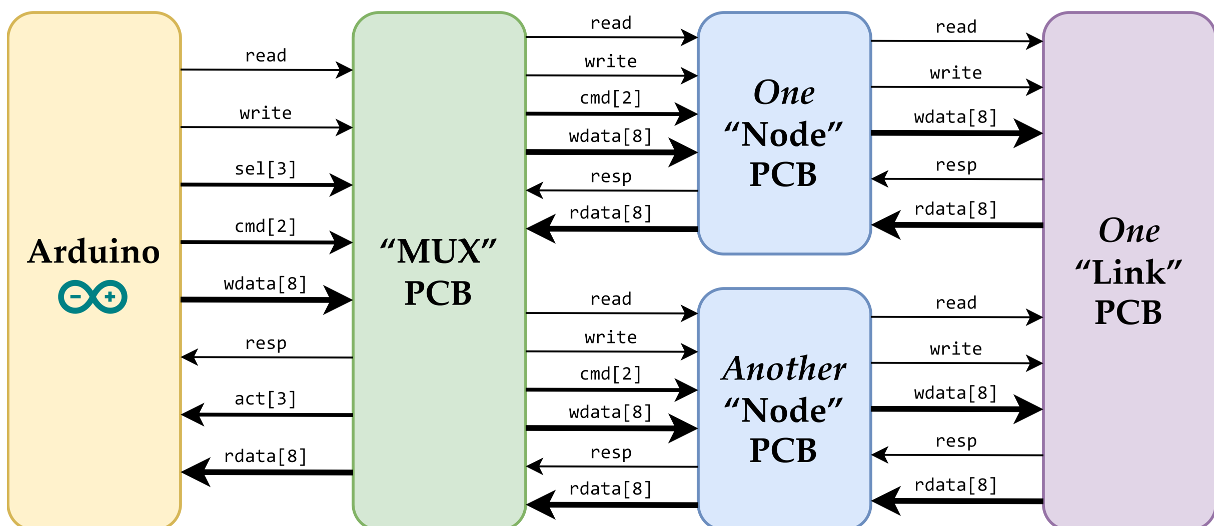

Figure 3: Intermodular communication interface signals.

Each signal functions as follows.

- `read`: master-to-slave flag to read node flux or link capacity
- `write`: master-to-slave flag to set link flux
- `resp`: slave-to-master flag to acknowledge the completion of read/write
- `sel[3]`: Arduino-to-MUX "select" signal of the operation target among 6 nodes
- `act[3]`: MUX-to-Arduino "active" signal of the message source among 6 nodes
- `cmd[2]`: master-to-slave "command" signal to specify the nature of a read/write operation (`00`, `01`, `10` for accessing one of 3 links, `11` for accessing the node itself)
- `wdata[8]`: master-to-slave data bus (signed 8-bit int) for link flux
- `rdata[8]`: slave-to-master data bus (signed 8-bit int) for node flux or link capacity

The following signal flow reads the node flux of Node 0.

- Arduino to MUX: `read`, `sel` Node 0, `cmd` NODE
- MUX to Node 0: `read`, `cmd` NODE
- Node 0 to MUX: `resp`, `rdata` flux
- MUX to Arduino: `resp`, `act` Node 0, `rdata` flux

The following signal flow reads the link capacity between Nodes 0 and 1, given that Link 1-2 is connected to Port A of Node 0. Note that we initially don't know what's on the other side, thus a Link PCB receives command from a port and responses *to the other port*.

- Arduino to MUX: `read`, `sel` Node 0, `cmd` LINKA
- MUX to Node 0: `read`, `cmd` LINKA
- Node 0 to Link 0-1: `read`
- Link 0-1 to Node 1: `resp`, `rdata` cap
- Node 1 to MUX: `resp`, `rdata` cap
- MUX to Arduino: `resp`, `act` Node 1, `rdata` cap

The following signal flow writes the link flux between Nodes 0 and 1. Note that never "write" a node flux, since this is a hard requirement failing which should produce an error message on the GUI.

- Arduino to MUX: `write`, `sel` Node 0, `cmd` LINKA, `wdata` flux
- MUX to Node 0: `write`, `cmd` LINKA, `wdata` flux
- Node 0 to Link 0-1: `write`, `wdata` flux
- Link 0-1 to Node 1: `resp`
- Node 1 to MUX: `resp`
- MUX to Arduino: `resp`, `act` Node 1

Demonstrative example. We illustrate the interaction of these protocols in a practical scenario where the following network is implemented.

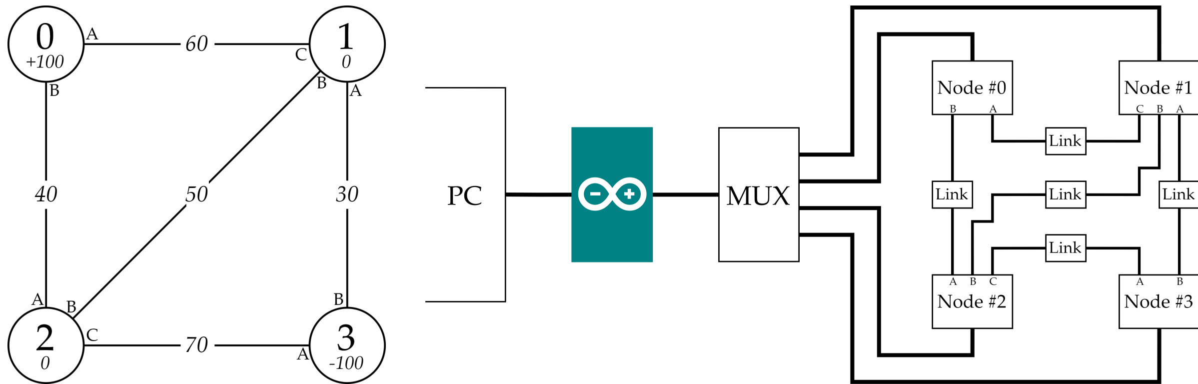

Figure 4: Subsystem organization of top-level design entity.

Arduino-MUX	MUX-Node	Node-Link
`read`, `sel 000`, `cmd 11`		
	(N0) `read`, `cmd 11`	
	(N0) `resp`, `rdata` 100	
`resp`, `act 000`, `rdata` 100		
`read`, `sel 000`, `cmd 00`		
	(N0) `read`, `cmd 00`	
		(N0-N1) `read`
		(N0-N1) `resp`, `rdata` 60
	(N1) `resp`, `rdata` 60	
`resp`, `act 01`, `rdata` 60		
...		
`read`, `sel 011`, `cmd 01`		
	(N3) `read`, `cmd 01`	
		(N3-N1) `read`
		(N3-N1) `resp`, `rdata` 30
	(N1) `resp`, `rdata` 30	
`resp`, `act 01`, `rdata` 30		

Table 1: Signal flow example for polling network configurations in Figure 4.

Arduino-MUX	MUX-Node	Node-Link
write, sel 000, cmd 00, wdata 60		
	(N0) write, cmd 00, wdata 60	
		(N0-N1) write, wdata 60
		(N0-N1) resp
	(N1) resp	
resp, act 01		
...		
write, sel 001, cmd 01, wdata 30		
	(N1) write, cmd 01, wdata 30	
		(N1-N2) write, wdata 30
		(N1-N2) resp
	(N2) resp	
resp, act 10		
...		

Table 2: Signal flow example for writing network configurations in Figure 4.

2.2 "Node" PCBs

We now investigate the circuit components and finite-state machine design in the modules, except for the ready-made Arduino controller which is primarily concerned with the software subsystem. Circuit diagrams are also provided wherever necessary.

Each node is a customized PCB board that includes

- one 8-bit **register** that stores the node flux `flux`, i.e., the amount of simulated flow (for sources) originating from or (for sinks) terminating at the node,
- one **knob** that adjusts `flux`,
- four digits of **7-segment display** that indicates `flux` (-128 to 127),
- one group of node-to-MUX **interface** `mux_*` (as specified in Section 2.1),
- three groups of node-to-link **interfaces** `linka_*`, `linkb_*`, `linkc_*`, and
- one **microcontroller** that implements a FSM and drives the display and signals.

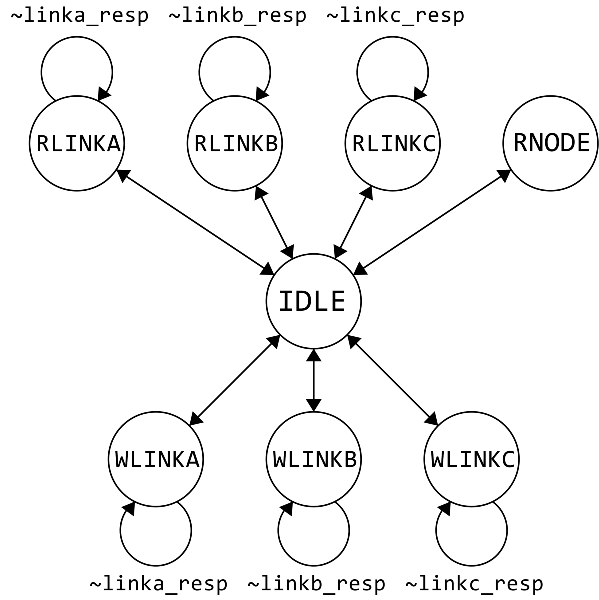

Figure 5: FSM state transition diagram for the Node PCB.

The conditions of unlabelled transitions to/from IDLE are specified as follows.

- IDLE → RLINKA: mux_read AND mux_cmd == LINKA (00)
- IDLE → RLINKB: mux_read AND mux_cmd == LINKB (01)
- IDLE → RLINKC: mux_read AND mux_cmd == LINKC (10)
- IDLE → RNODE : mux_read AND mux_cmd == NODE (11)
- IDLE → WLINKA: mux_write AND mux_cmd == LINKA (00)
- IDLE → WLINKB: mux_write AND mux_cmd == LINKB (01)
- IDLE → WLINKC: mux_write AND mux_cmd == LINKC (10)
- RLINKA → IDLE, WLINKA → IDLE: linka_resp
- RLINKB → IDLE, WLINKB → IDLE: linkb_resp
- RLINKC → IDLE, WLINKC → IDLE: linkc_resp
- RNODE → IDLE: Unconditional

State	Encoding	Signals
RLINKA	000	linka_read
RLINKB	001	linkb_read
RLINKC	010	linkc_read
RNODE	011	mux_resp, mux_rdata = flux
WLINKA	100	linka_write, linka_wdata = mux_wdata
WLINKB	101	linkb_write, linkb_wdata = mux_wdata
WLINKC	110	linkc_write, linkc_wdata = mux_wdata
IDLE	111	None

Table 3: FSM state actions for the Node PCB.

2.3 "Link" PCBs

We considered treating the links as mere LED strips and leave all capacity configurations to the node PCBs, but this appears counterintuitive and would require a complex communication protocol to set up the network topology. Therefore, each link is also a customized PCB board that includes

- two 8-bit **registers** that stores the link capacity `cap` and the actual link flux `flux`,
- one **knob** that adjusts `cap`,
- three digits of **7-segment display** that indicates `cap` (0 to 127),
- a string of ten **LEDs** that shows `flux` (10% capacity used per LED),
- two groups of link-to-node **interfaces** `node0_*`, `node1_*`, and
- one **microcontroller** that implements a FSM and drives the display and signals.

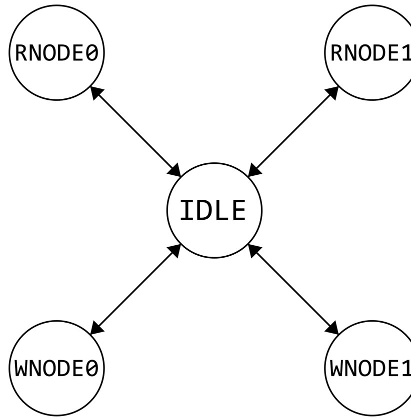

Figure 6: FSM state transition diagram for the Link PCB.

The conditions of unlabelled transitions to/from `IDLE` are specified as follows.

- `IDLE` → `RNODE0`: `node0_read`
- `IDLE` → `RNODE1`: `node1_read`
- `IDLE` → `WNODE0`: `node0_write`
- `IDLE` → `WNODE1`: `node1_write`
- `RNODE0`, `RNODE1`, `WNODE0`, `WNODE1` → `IDLE`: Unconditional

State	Encoding	Signals
RNODE0	000	node1_resp, node1_rdata = cap
RNODE1	001	node0_resp, node0_rdata = cap
WNODE0	100	node1_resp
WNODE1	101	node0_resp
IDLE	111	None

Table 4: FSM state actions for the Link PCB.

Pay special attention that as mentioned before, a Link PCB receives command from a port and raises the response signal *on the other port* such that both the link capacity and the destination node ID are passed back to the Arduino controller.

2.4 "MUX" PCB

The MUX PCB (1) helps simplify the interface to Arduino and (2) supplies power, clock, and reset signals to all nodes and links. It includes

- six groups of MUX-to-node **interfaces** `node0_*`, `node1_*`, ..., `node5_*`,
- one group of MUX-to-Arduino **interface** `arduino_*`, and
- one **microcontroller** that drives the signals.

Unlike the node and link PCBs, the MUX does not have input (knob) or output (display) components. In fact, even an FSM is unnecessary for performing the "forwarding" function. Specifically, the microcontroller implements an encoder/decoder that

- decodes `arduino_read` to `node*_read` according to `arduino_sel`,
- decodes `arduino_write` to `node*_write` according to `arduino_sel`,
- decodes `arduino_cmd` to `node*_cmd` according to `arduino_sel`,
- decodes `arduino_wdata` to `node*_wdata` according to `arduino_sel`,
- encodes `node*_resp` to `arduino_act` with priority given to lower node IDs, and
- multiplexes `node*_rdata` to `arduino_rdata` according to `arduino_act`.

2.5 Arduino Controller

The emulator has a central Arduino controller that talks to each node (but *not* the links) to display the capacities and actual flow amounts. The microcontroller is expected to

- read the node configurations and link capacities from the knobs,
- computes the maximal flow using an optimized Ford-Fulkerson algorithm, and
- updates the flow display on each node and link in real-time.

Still, the limited number of ports on Arduino inherently limits the scale of network – this set of protocol can support **up to 6 nodes and 3 links per node**.

2.6 Graphical User Interface

The graphical user interface (GUI) receives the flow solution from Arduino and visualizes the network flow data. *Note that all parameter configurations are done on the physical knobs,*

not in the GUI. The design and development of the GUI are guided by user-centric principles to ensure intuitiveness, ease of use, and functionality. The GUI should implement the following key functions:

- **Real-time flow visualization.** The GUI should display the real-time flow of data packets within the network, preferably assisted by *animated* changes in the network diagram, correlating with the actual flow of data through the nodes and links.
- **Algorithm control and monitoring.** The user should be able to initiate, pause, or stop the flow computation algorithm and monitor its progress. The GUI should provide a console or log view to observe the real-time output from the algorithm, including any alerts or error messages.
- **Error handling and feedback.** Prompt and clear feedback should be given for any invalid actions or errors in configuration, e.g., when there are no *or multiple* feasible solutions, or when the conservation principle is violated. This includes the visualization of flow states that are not permissible due to the current network setup.
- **Responsiveness and scalability.** The GUI should be responsive to different screen sizes and resolutions, ensuring usability across various devices.

The GUI will be developed in a modular fashion, allowing for future enhancements and features to be added with minimal disruption to the existing system. By focusing on these core functions, the GUI will facilitate an effective and educational experience.

2.7 Outer Packaging

The entire toolbox will *reside on a vertical surface* for convenient display on whiteboards. We aim to make the outer packaging structure and overall appearance of the toolbox both aesthetically pleasing and functional. The following considerations guide the design of our product's exterior:

- **Acrylic casing.** The node and link PCBs will be encased in high-quality acrylic panels, allowing for the visibility of the internal components and LED indicators.
- **LED indicators.** The flow of data through the network will be represented by LED lights housed in clear, durable tubes that not only protect the electronics but also distribute light evenly, making the flow visually discernible from all angles.
- **Modularity and expandability.** The modular design will allow for the network to be expanded or reconfigured. This includes detachable nodes and links, which can be securely attached or removed without the need for specialized tools. *Note that the interface has a large number of signals, which may require a revision for over 6 nodes.*
- **Environmentally conscious.** The design process will incorporate environmentally friendly materials and practices, including recyclable plastics and efficient LED lighting, to minimize the ecological footprint of our product.

3 Reliability, Ethics, and Safety

3.1 Tolerance Analysis

- (Node and Link PCBs) Tolerance of Resistors and Capacitors
  - Impact: They affect the timing and signal shaping within the node's circuitry, possibly leading to misinterpretation of signals or timing mismatches.
  - Analysis: Consider how the RC time constants change due to variations in resistor and capacitor values, and how they could affect the signal levels and timing, especially for signals interfacing with the MUX and links.
  - Simulation: We may use RC response curves to illustrate how different RC time constants affect the rise and fall times of the signal waveforms.
- (Node and Link PCBs) Accuracy of Knob Potentiometers
  - Impact: It affects the precision with which node flux can be set, potentially leading to inaccurate flow visualization.
  - Analysis: Determine the range of actual values for a supposed set value and how this affects the node's status as a source, sink, or transfer station.
  - Simulation: We may display a range of actual flux values corresponding to a set position on the knob to highlight the variability due to tolerance.
- (Link PCBs) Variability of LED Display
  - Impact: Variability in LED brightness and color could lead to inconsistent flow visualization across different links.
  - Analysis: Consider the variation in LED brightness and color due to current and voltage tolerances and its impact on visual accuracy.
  - Simulation: We may use a comparison chart to show the expected brightness range of LEDs under different current and voltage conditions due to tolerances.
- (Intermodular Protocol) Voltage Levels and Noise Margins
  - Impact: Fluctuations in voltage levels can affect the logic levels interpreted by the microcontrollers and introduce errors in the protocol.
  - Analysis: Examine how variations in voltage levels (due to power supply tolerances or signal integrity issues) might impact the detection of high and low states in the communication protocol.
  - Simulation: We may show the acceptable voltage levels for logic `0` and logic `1` for the microcontrollers and how variations might lead to incorrect logic level interpretation.

3.2 Ethical Concerns

To avoid ethical breaches in the development and deployment of our toolbox, we commit to adhering closely to the principles outlined in both the IEEE Code of Ethics [1] and the ACM Code of Ethics [2]. Key considerations include but are not limited to

- Respecting *intellectual honesty* (ACM, Clause 1.5), acknowledging contributions accurately, adopt secure coding practices, and avoiding plagiarism in development.
- Committing to *inclusivity and accessibility* (ACM, Clause 1.4). For example, both the model and the GUI should be designed to be usable by a broad spectrum of individuals and accommodate users with diverse technical backgrounds.
- Supporting *sustainable development* (ACM, Clause 3.4; IEEE, Clause 1). This includes choosing recyclable and sustainable materials for hardware components and designing the embedded electrical system for energy efficiency.
- Mitigating the *risk of overreliance* by positioning our tool as a supplementary, instead of replacement, of traditional educational resources, in compliance with the IEEE's commitment to continuous learning (IEEE, Clause 6).

By fostering an environment of transparency, responsibility, and respect for user rights, we aim to not only comply with professional ethical standards but also contribute positively to the educational and technological communities.

3.3 Safety Concerns

This project involves the use of diodes, microcontrollers, and light bulbs to simulate the network information transmission flow. Recognizing the associated electrical, fire, mechanical, chemical, and operational hazards both in the development and deployment stages, we will abide by the IEEE National Electrical Safety Code [3] through

- Implementing comprehensive safety measures including protection against *electric* shocks (e.g., insulated tools) and *fire* precautions (e.g., circuit breakers, fuses) to prevent overheating and short circuits.
- Securely mounting all PCB board components to ensure *mechanical* robustness.
- Using protective gears during assembly involving batteries and soldering operations, and safely disposing hazardous *chemical* waste.
- Ensuring the safety of users (ACM, Clause 2.9; IEEE, Clause 1) by rigorously testing the system to prevent any *operational* hazards. For instance, the number of small parts in the physical model should be minimized to prevent choking hazards.
- Training in safe handling practices and emergency procedures.

References

[1] IEEE. “IEEE Code of Ethics.” (2016), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 03/06/2024).
[2] ACM. “ACM Code of Ethics.” (2018), [Online]. Available: https://www.acm.org/code-of-ethics (visited on 03/06/2024).
[3] “2023 National Electrical Safety Code® (NESC®),” *2023 National Electrical Safety Code(R) (NESC(R))*, pp. 1–365, 2022. DOI: 10.1109/IEEESTD.2022.9825487.