ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

STM32-Powered Physical Model of Network Flow

Team #17

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

Sponsor: Pavel Loskot
TA: Yue Yu

May 31, 2024

Contents

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 tune in the constraints affects the optimal solution using mere intuition.

Having a physical model can provide a more intuitive sense of "tuning" the network by assigning directly adjustable parameters 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 builds 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 are computed on a connected computer using the Network Simplex algorithm in Python. Display is controlled using an embedded STM32 microcontroller.

The intermodular communication protocol has gone through several revisions. We moved from a MUX-driven protocol (first iteration) or direct access using Arduino (second iteration) and instead opted for a more flexible peer-to-peer (P2P) communication model (third iteration) where only one node maintains a direct UART connection with the computer, and identical "configuration information" strings are distributed for local parsing and display control.

Specifically, in the final layout, we simulate a simple yet realistic network topology with "forwarding hubs," incorporating eight nodes and ten links in total. We hope 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).

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

- This is a small-scale network with up to four links per node – that is, if the network is ever to be reconfigured.
- The nodes don't have buffers and are 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 is modular, i.e., each node has some "slots" reserved for installing new links.
- The software communicates with all nodes and pipes and update the flows in real-time (within 500ms) in response to changes in setup.
- The algorithm handles and report edge cases, e.g., a network with no feasible flows.

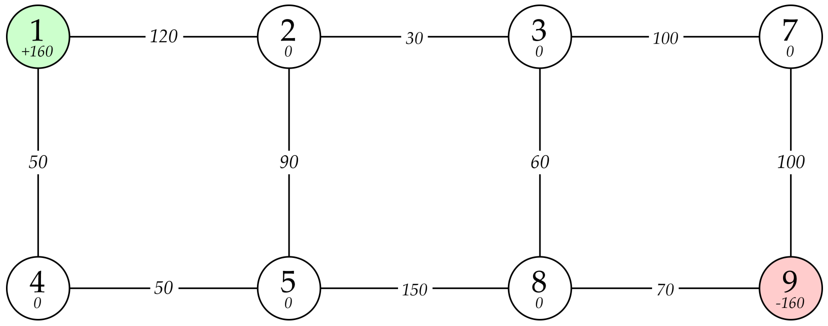


Figure 1: The network topology to be implemented, with sample parameters.
The source is marked in green and the sink in red. Sources and sinks need not be unique.

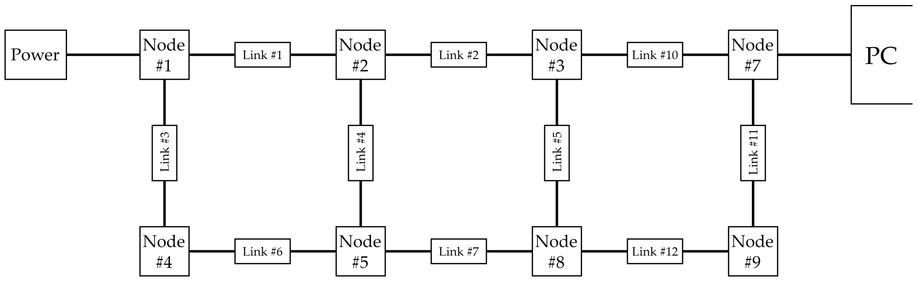


Figure 2: Layout of intermodular connections.
The weird numbering is due to Node #6 malfunctioning.

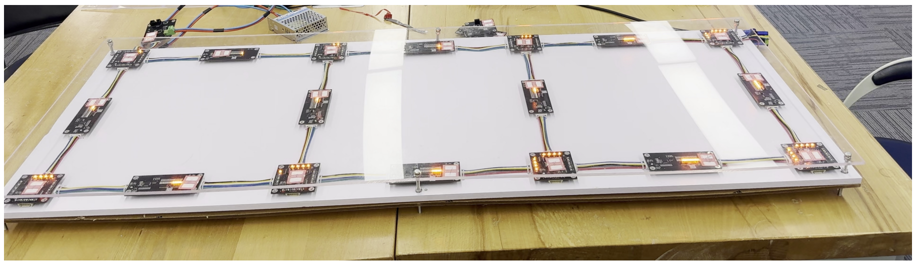


Figure 3: Our final product, connected using the topology above.
Just powered up, the number of LEDs lit up on each PCB indicates its unique ID.

2 Design

In our physical model, a complex network is abstracted into a series of pipes, representing the communication **links** characterized by their designated capacities ("link capacity"). The embedded LEDs depict the dynamic flow ("link flux") of data packets coursing through the network.

Each **node** within this network, also represented by a dedicated PCB, is associated with a "node flux" value, offering the flexibility to characterize each node as a source with outgoing flow, a sink with incoming flow, or a neutral transfer station with no net flow. This modular approach underpins the system's design, ensuring adaptability and ease of modification as the network model evolves.

The transition to the STM32 microcontroller platform on each PCB has substantially increased the system's capabilities. Tasked with collecting network configuration, the microcontroller reads from the network configuration string, control the LEDs correspondingly, and forwards it to the next neighbor.

The Python code running on an external computer implements the Ford-Fulkerson algorithm that 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.

The following diagram details the UART interface between modules. The links are *not* directly connected to the computer, and thus the capacity values must be passed indirectly through the nodes.

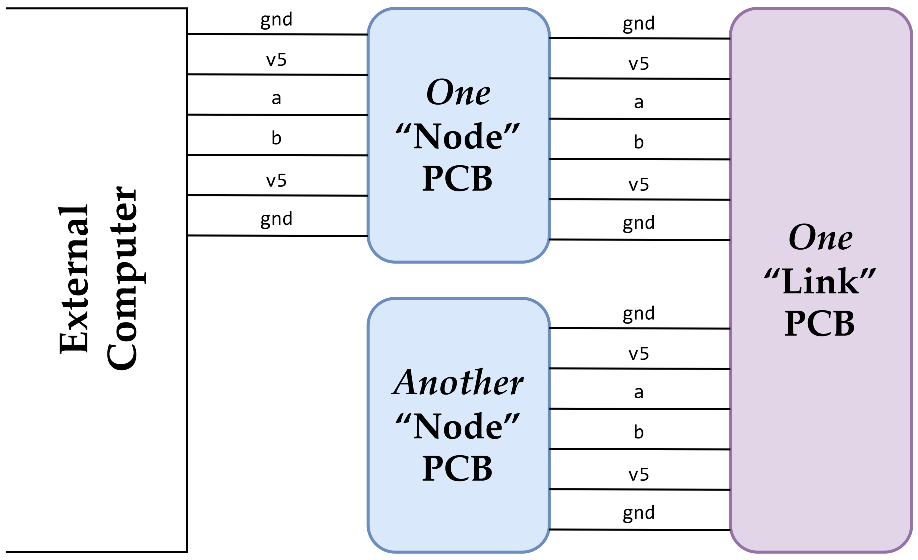


Figure 4: Intermodular communication interface signals.

2.1 “Node” PCBs

We now investigate the circuit components and finite-state machine design in the modules, except for the external computer 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 STM32 **microcontroller** that drives the display and signals,
- sixteen **LEDs** that indicates the “node flux”, and
- four groups of UART serial **interfaces** with link PCBs, the computer, or power.
- *We considered integrating potentiometers (knobs) onto the PCBs, but space was insufficient.*

The Node PCB forms a critical part of our toolbox, serving as the visualization point for network traffic at each node within the system. Central to the Node PCB is the integration of the `74hc595` series shift registers. These components enable the expansion of output ports through serial-to-parallel conversion, allowing the microcontroller to control a larger array of LEDs while utilizing fewer I/O pins.

In the schematic, the connection of resistors `R1` to `R22` in series with the LEDs facilitates the display of network traffic. Each resistor-LED pair acts as an individual indicator. The binary states managed by the shift register correspond to the on/off states of LEDs.

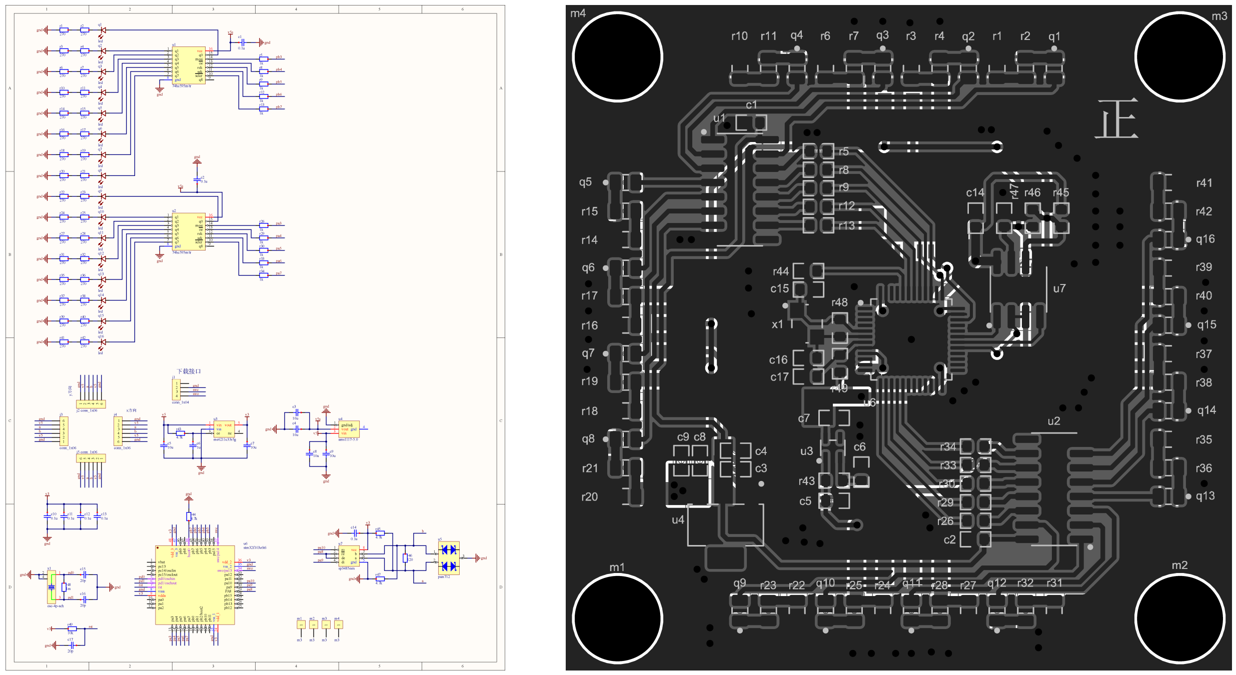


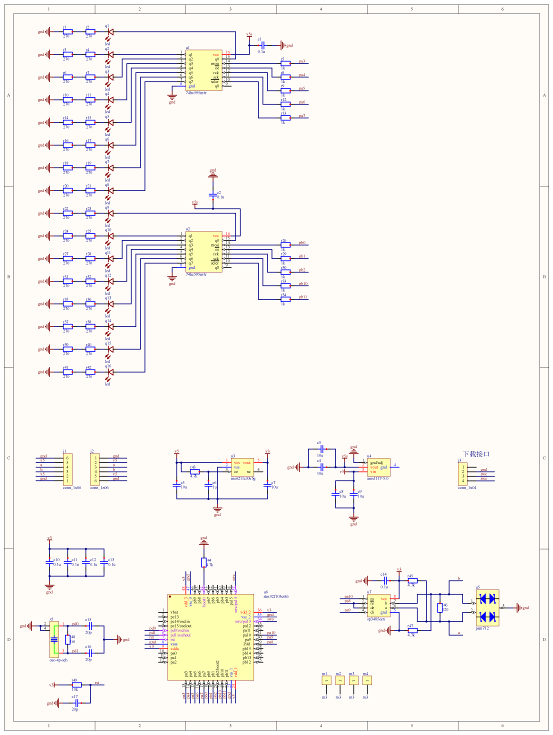
Figure 5: Circuit schematic and PCB layout for the nodes

2.2 "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

- one STM32 **microcontroller** that drives the display and signals,
- sixteen **LEDs** that indicates the "link flux" (exactly one LED is on at any time, and the "rolling" speed indicates the flux amount), and
- two groups of UART serial **interfaces** with node PCBs.

The Link PCBs function as conduits for demonstrating the flow of data between nodes, with LED arrays illustrating the link status and activity within the network.



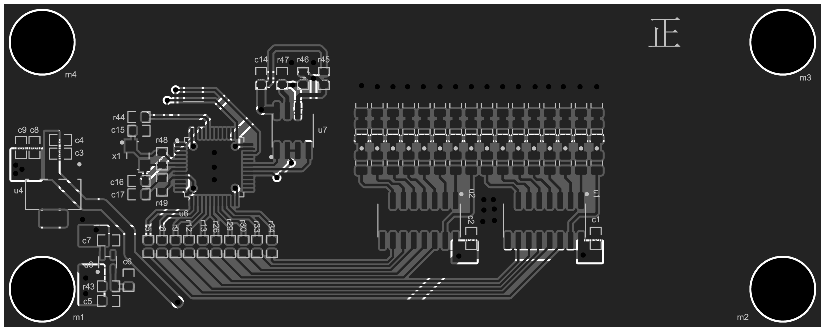


Figure 6: Circuit schematic and PCB layout for the links.

In the detailed schematic provided, the `74hc595` shift registers are used to control arrays of LEDs, each corresponding to a particular link state, illuminated to represent the presence of data flow. The shift registers enable the control of multiple LEDs while minimizing the GPIO usage of the controlling microcontroller. Each LED is connected in series with a resistor, which limits the current to prevent damage to the LEDs.

During operation, the shift register receives serial data from the microcontroller. Upon receiving a clock pulse, the register shifts this data through, setting each LED's state accordingly. Then a latch signal is applied, which updates the output pins and changes the LEDs' states simultaneously. This design allows for real-time updates to the network flow visualization without perceptible lag to the user.

2.3 Power PCB

We used to have a MUX PCB that (1) helps simplify the interface to computer while (2) supplying power, clock, and reset signals to all nodes and links. However, due to the exceedingly large number of I/O ports, this part of design has been refactored to a dedicated power PCB that supplies 5V power, ground, and clock signals to all PCBs.

Similar to the P2P communication network in which the protocol implies that only one node needs to be connected to the external computer, we design the power supply such that only one PCB needs to be connected to the power source.

Power management on the board is handled by the `MEC6211` voltage regulator, ensuring that the STM32 and other components receive a stable voltage, critical for maintaining reliable operation. Surrounding are various passive components like resistors and capacitors, which stabilize the signals and power supply. The capacitors closer to the power inputs, marked as `C4` through `C7`, are for decoupling purposes, filtering out noise from the power supply to the microcontroller and other sensitive components.

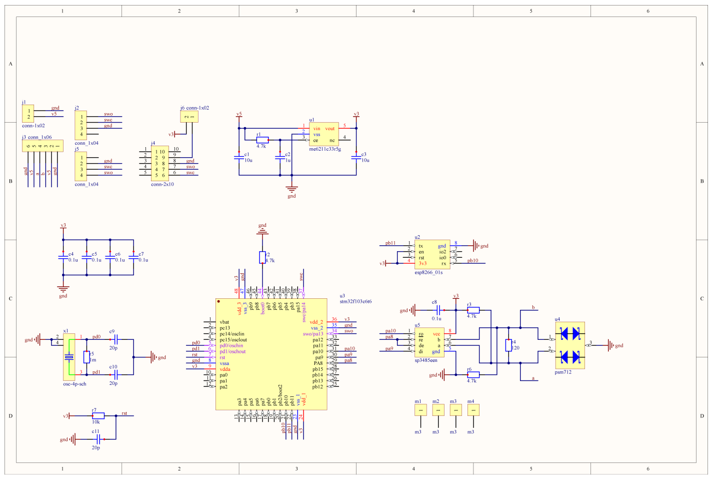
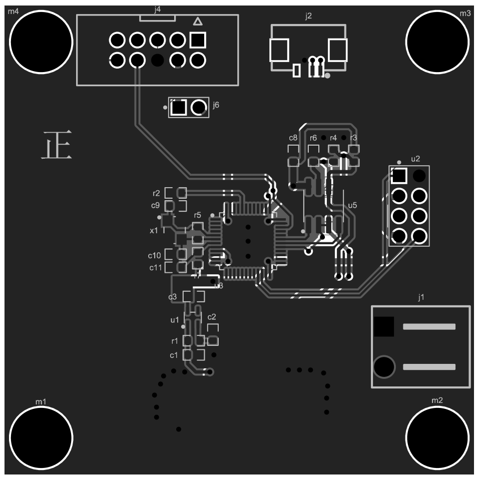



Figure 7: Circuit schematic and PCB layout for the power.

The communication lines between the STM32 and other peripherals are safeguarded by the `psm712` diodes, which protect against voltage spikes that could otherwise damage the microcontroller. There is also a crystal oscillator circuit, fundamental for providing a precise clock signal and ensuring accurate timing for all operations.

2.4 Intermodular Communication

In our project, the communication between the Python application and the physical hardware, specifically the printed circuit board (PCB), is facilitated through PySerial [1]. This is a Python library that provides a convenient interface to the serial ports, enabling the sending and receiving of data over serial communication lines with minimal effort. The choice of PySerial is driven by its simplicity and robustness, allowing developers to manage serial communication through a few lines of code.

The underlying communication interface employed in the project is RS-485 [2], [3], known for its reliability and efficiency in enabling long-distance and high-speed data transmission. The embedded SP3485EEN chip ("low power half-duplex RS-485 transceiver") [4] handles the electrical aspects of RS-485 communication, ensuring robust data transmission even over long distances and in electrically noisy environments. The STM32 sends and receives data to/from the SP3485EEN and controls the transceiver's operation by managing the DE/RE pins to switch between sending and receiving modes.

2.5 Software Controller

We incorporate the `networkx` Python library [5] in our solution and therefore inherit its graph representation, problem formulation, and the Network Simplex Algorithm [6], [7] working under the hood. The basic idea is to maintain a spanning tree of the graph with as high "potential" (available flux) as possible, and repeatedly pivot at the root and send flows along the paths found.

The Python software is expected to

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

2.6 Graphical User Interface

We integrate the components and implement a graphical user interface (GUI) using `tkinter` [8]. The GUI receives user inputs, compute the flow solution and visualizes the network flow data. Its development is guided by user-centric principles to ensure intuitiveness and ease of use. The key functions are:

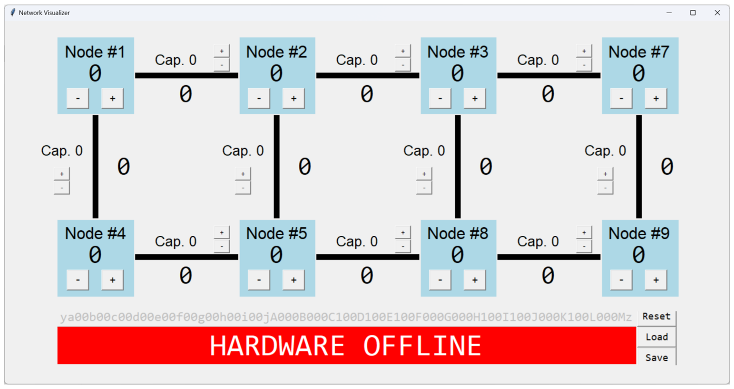
- **Real-time flow visualization.** The GUI displays the real-time flow of data packets within the network, correlating with the actual flow of data through the nodes and links.
- **Algorithm control and monitoring.** The user is able to initiate, pause, or stop the flow computation algorithm and monitor its progress. The GUI provides 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 are given for any invalid actions, e.g., when there are no feasible solutions, or when the conservation principle is violated.

The GUI is developed in a modular fashion, allowing for future enhancements and features to be added with minimal disruption to the existing system.

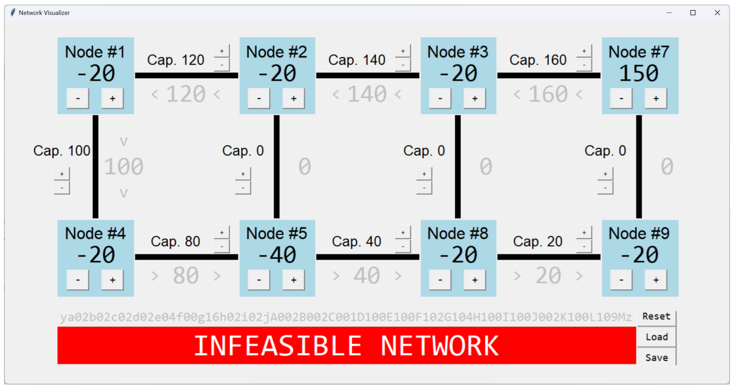
2.7 Outer Packaging

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:

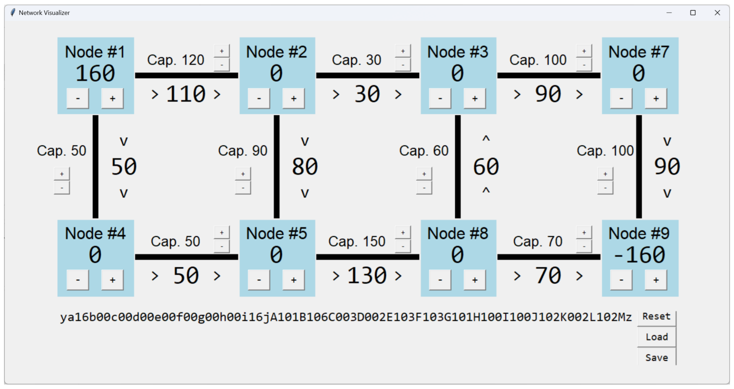
- **Acrylic casing.** The node and link PCBs will be encased in high-quality acrylic panels, allowing for the visibility of the internal components and LED indicators.
- **LED indicators.** The flow of data through the network will be represented by LED lights housed in clear, durable tubes that not 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.



```
ya00b00c00d00e00f00g00h00i00jA000B000C100D100E100F000G000H100I100J000K100L000Mz
HARDWARE OFFLINE
```



```
ya02b02c02d02e04f00g16h02i02jA002B002C001D100E100F102G104H100I100J002K100L109Mz
INFEASIBLE NETWORK
```



```
ya16b00c00d00e00f00g00h00i16jA101B106C003D002E103F103G101H100I100J102K002L102Mz
```

Figure 8: Three GUI screenshots showing different flow patterns and edge cases.

(Top) shows the initial configuration where no flux is present, and the GUI application has just been started without connecting to the serial port. A "Hardware offline" warning message is displayed.

(Middle) shows a single-source multiple-sink "distributed" flow pattern, but the flux supply from Node #7 is nudged from 160 to 150 to mismatch the total demand. An "Infeasible network" error message is displayed.

(Bottom) shows a valid single-source single-sink "diagonal" flow pattern as shown in **Figure 1**. The shown message is broadcasted to the hardware.

3 Reliability and Verifiability

3.1 Requirements and Verification

Requirement	Verification
Node PCBs. The Node PCB should parse the command message correctly and all the LEDs are functional.	1. Power up a Node PCB. 2. Identify its ID, and drive multiple manually encoded commands with the Flux Indicator ranging from `99`, `00`, `01`, . . . , to `20`. 3. Observe that • {ID} number of LEDs light up for `99`, • all LEDs turn off for `00`, • one LED lights up for `01`, . . . • sixteen LEDs light up for `16`, and • all LEDs turn off for `17` onwards.
Link PCBs. The Link PCB should parse the command message correctly and all the LEDs are functional.	1. Power up a Node PCB. 2. Identify its ID, and drive commands with the Direction Indicator (DI) choosing from `0`, `1`, `2`, and the Speed Indicator (SI) ranging from `99`, `00`, `01`, . . . , to `40`. 3. Observe that • all LEDs turn off for DI=`2`; else, • {ID} number of LEDs light up statically with no animation for SI=`99`, • no two LEDs light up simultaneously for SI values other than `99`, • all LEDs turn off for SI=`00`, • one LED moves swiftly for SI=`01`, • it turns around when DI flips, • it moves slower for SI=`02`, . . . • it moves the slowest for SI=`36`, and • all LEDs turn off for SI=`37` onwards.

Table 1: Technical requirements and verification procedures.

Requirement	**Verification**
Algorithm. The algorithm should respond to changes in the network setup within 500ms.	1. Power up the system. 2. On the GUI, designate the source and sink by deviating some "node flux" values from the neutral position. Ensure the displayed node flux values sum to zero. 3. Enable all links except for leaving the last leap on *one of* the feasible paths at zero capacity (e.g., Link #12 in **Figure 2**). 4. Loosen the "bottleneck" and observe that the flow is redistributed within 500ms.
GUI. The system should report an infeasible network when no feasible flow exists.	1. Power up the system. 2. Set all node and link attributes to the neutral position except for one single source and a matching sink. 3. Set the capacities of all links to a sufficiently large value, except for leaving those entering the sink at zero capacity. 4. Observe that the GUI reports an unsolvable case while the LEDs maintain the last feasible solution.
GUI. The system should report an infeasible network when the flow is not conserved.	1. Starting from the last experiment, slightly tune the sink flux to mismatch the source. 2. Observe that the GUI reports an unsolvable case while the LEDs maintain the last feasible solution.
Packaging. The system should attach firmly to a acrylic whiteboard and be visible from a distance of 3 meters.	1. Configure the network such that the solution uses over 90% capacity for all links. 2. Observe that all LEDs and segment digits are clearly visible from 3 meters away.

Table 2: Technical requirements and verification procedures (cont'd).

4 Cost Analysis

Estimated labor costs [9]: 4 people$\times 10$ hrs/week$\times 8$ weeks$\times$RMB 200/hr $=$ **RMB 64,000**

Component	Quantity	Cost
Magnetic Isolation USB 485 Bidirectional Serial Converter	1	RMB 59.00
Chip Resistor	1 set	RMB 25.00
Switching Mode Power Supply	1	RMB 51.00
High-power Two Pin Plug	2	RMB 7.20
Spacing Plug-in Terminal Block	3	RMB 1.92
Curved Pin Socket	3	RMB 0.36
FFC/FPC Connector	30	RMB 8.40
2.54mm PIN	3	RMB 6.60
FC Double Headed Ribbon Cable	3	RMB 7.20
ST - LINK V2 STM 32	1	RMB 19.80
High Brightness SMT LED	500	RMB 11.00
XH 2.54 Terminal Wire	4	RMB 90.00
XH 6P Recumbent Patch	1	RMB 30.48
PCB Source	5	RMB 46.60
PCB Node	20	RMB 58.92
PCB Link	20	RMB 63.07
AMS1117	50	RMB 12.74
LDO	50	RMB 22.10
TVS/ESD	50	RMB 15.15
RS-485	45	RMB 66.35
Shift Register	85	RMB 38.16
304 Stainless Steel Bolt Set M5*70	1 set	RMB 6.9
304 Stainless Steel Bolt Set M5*60	1 set	RMB 6.9
304 Stainless Steel Nut M5	50	RMB 2.9
M3 Diamond Pointed Round File	1	RMB 11.00
Chevrolet Board 1200*450	1	RMB 60.00
Acrylic Board 1200*450	2	RMB 260
Cable ties	1000	RMB 5.36
Software Copyright Fee	/	RMB 580.00
Total		**RMB 1574.11**

Table 3: Cost breakdown of all circuit components.

5 Ethics and Safety

5.1 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 [10] and the ACM Code of Ethics [11]. 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.

5.2 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 [12] 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.

6 Conclusion

In this project, we design a modular hardware emulator to visualize network flows under capacity constraints on links. In addition, we reviewed the design requirements and verification process from the deprecated Arduino-based solution to match the current STM32-based specification. We hope that this education tool will provide a more intuitive experience involving the logistics network functioning in real time in response to changing constraints, and therefore facilitate the understanding of network flow optimization problems in the classroom.

Future work. Our solution may be further improved in the following aspects:

- The knob potentiometers and digit displays in our original design have been optimized away from the Node and Link PCBs due to space constraints. Yet this decision leads to suboptimal user experience, as the capacity and flux on links cannot be displayed quantitively. A physical interface could be valuable in providing a more realistic experience of “tuning” the network flow.
- The RS-485/UART transmission latency could easily exceed 500ms if the maximum number of leaps between nodes further increases (this value is 5 at this point). The protocol could be optimized if we switch from an ASCII-based message encoding.
- We make several strong assumptions involving the simulated network, including uniform weights on links and the absence of “forwarding buffers” on nodes. A more realistic model could be developed to better reflect real-world logistics.
- The physical model is quite large (120cm $\times$ 45cm $\times$ 10cm) and expensive, since it involves a large number of interconnected PCBs. A more compact, cost-effective, and mechanically robust design is necessary to make our solution accessible to a wider audience.

References

[1] Chris Liechti. "pySerial 3.0 Documentation." (2015), [Online]. Available: https://pythonhosted.org/pyserial (visited on 05/16/2024).

[2] Texas Instruments. "The RS-485 Design Guide." (2021), [Online]. Available: https://ecs.grainger.illinois.edu/student-resources/offers/salary (visited on 05/05/2024).

[3] Analog Devices. "RS485 Quick Guide." (), [Online]. Available: https://www.analog.com/media/en/technical-documentation/product-selector-card/rs485fe.pdf (visited on 05/05/2024).

[4] Mouser Electronics. "+3.3V Low Power Half-Duplex RS-485 Transceiver with 10Mbps Data Rate." (2017), [Online]. Available: https://www.mouser.com/datasheet/2/146/SP3485-1222763.pdf (visited on 05/07/2024).

[5] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[6] J. B. Orlin, "A polynomial time primal network simplex algorithm for minimum cost flows," *Mathematical Programming*, vol. 78, no. 2, pp. 109–129, 1997, ISSN: 1436-4646. DOI: 10.1007/BF02614365. [Online]. Available: https://doi.org/10.1007/BF02614365.

[7] R. E. Tarjan, "Dynamic trees as search trees via euler tours, applied to the network simplex algorithm," *Mathematical Programming*, vol. 78, no. 2, pp. 169–177, 1997, ISSN: 1436-4646. DOI: 10.1007/BF02614369. [Online]. Available: https://doi.org/10.1007/BF02614369.

[8] Python Software Foundation. "tkinter — Python interface to Tcl/Tk." (2024), [Online]. Available: https://docs.python.org/3/library/tkinter.html (visited on 05/27/2024).

[9] Grainger Engineering Career Services. "2021-2022 Graduates Salary Data." (2022), [Online]. Available: https://ecs.grainger.illinois.edu/student-resources/offers/salary (visited on 03/27/2024).

[10] IEEE. "IEEE Code of Ethics." (2016), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 03/06/2024).

[11] ACM. "ACM Code of Ethics." (2018), [Online]. Available: https://www.acm.org/code-of-ethics (visited on 03/06/2024).

[12] "2023 National Electrical Safety Code® (NESC®)," *2023 National Electrical Safety Code(R) (NESC(R))*, pp. 1–365, 2022. DOI: 10.1109/IEEESTD.2022.9825487.