# Yen-Ju Tseng

tyj850916@gmail.com | +1(858) 729-3110 | LinkedIn Profile| Personal Website

## EDUCATION

**University of California, San Diego** **San Diego, USA**
*M.S. in Electrical and Computer Engineering (GPA: 3.5/4.0)* Sep 2021 – Jun 2023
- Coursework: Software Foundations, Operating Systems, Computer Networks, Front-end Development, Graduate Networked System, Advanced Data Structure, Principles of Programming Languages

**National Taipei University** **New Taipei City, Taiwan**
*B.S. in Communication Engineering (GPA: 3.46/4.0)* Sep 2015 – Jun 2019
- Coursework: Data Structure, Advanced Computer Programming, Database System

## SKILLS

**Programming:** C/C++, Golang, Java, Python, Kotlin, JavaScript, HTML/CSS, MATLAB
**Tools:** Git, Visual Studio, Visual Studio Code, Android Studio, Zookeeper, Kafka, MongoDB

## PERSONAL PROJECTS (code available upon request)

**MySQL-like Relational Database System in C++17** Apr 2022 – June 2022
- Designed and constructed a robust relational database akin to MySQL using C++17, showcasing flawless performance with seamless handling of **15000+** data entries.
- This system entailed **interpreting**, **manipulating**, **querying**, and **presenting table data results**.
- Developed the database system based on the **MVC (Model-View-Controller)** application design pattern.
- Employed **scanning**, **tokenizing**, and **parsing** techniques proficiently to manage user input.
- Implemented the **chain-of-responsibility** design pattern to efficiently process user-provided commands.
- Employed the **factory** design pattern to seamlessly handle statements.
- Optimized database performance with **indexes** and **LRU** Cache, achieving approximately a **20%** improvement.

**Distribute Systems Development (Java, Kafka, Zookeeper, MongoDB, Google Cloud Platform)** June 2023 – July 2023
- Established and deployed a distributed system with **Java** on **Google Cloud Platform**, achieving **scalability** and **fault tolerance**.
- Utilized **Kafka** as message brokers with **Zookeeper** for **scalability** and enhanced the **fault tolerance** by leader algorithm.
- Optimized network communication by leveraging **HTTP** and handling data serialization and deserialization with **protocol buffer**.
- Enhanced system performance and reliability by incorporating **load balancers** to avoid bottlenecks and ensure higher **availability**.
- Launched **MongoDB** with **replication set (master/slave architecture)** for high availability and data **sharding** for scalability.

**Fault-tolerance Scalable Cloud-Based File Storage service in Golang** Feb 2023 – Mar 2023
- Developed a **Dropbox-like**, **scalable**, **networked** file storage application, facilitating **concurrent** connections from multiple clients to access a shared set of files on the server. Utilized **gRPC** for client interaction and managed **100+** files.
- Divided files on the server into an ordered sequence of one or more blocks. Employed the **SHA-256 hash function** for each block, creating a hash list that represented the file.
- Utilized **protocol buffers** for **gRPC** and incorporated **versioning** and hash list techniques to handle update conflicts.
- Created and maintained an **index.db** file in the base directory of the client program to streamline **synchronization** operations.
- Implemented a mapping approach based on **consistent hashing** for efficient block storage and to ensure server **scalability**.
- Enhanced server reliability by integrating **fault tolerance** mechanisms based on the **RAFT distributed consensus protocol**.

**Sliding Window Protocol in C** Jan 2023 – Feb 2023
- Implemented communication between two or more hosts using a **sliding window protocol** that employed **selective repeat/retransmission** and **cumulative ACK** to guarantee reliable in-order delivery of frames between hosts. (Window size = 8).
- Partitioned messages larger than MAX_FRAME_SIZE (i.e., 64 bytes) into frames.
- Reconstructed frames at the receivers' end, accurately retrieved the original messages from the sender, and produced the output.
- Integrated a robust **error detection mechanism** utilizing **CRC-8** on both senders and receivers. Each sender could only communicate with one receiver at a time, while a receiver must be able to handle frames from multiple senders **concurrently**.

**Simple Router in C** Feb 2023 – Mar 2023
- Constructed a streamlined router capable of receiving raw Ethernet frames and efficiently handling various packet types, including **ARP requests**, **ARP replies**, **ARP caching**, **ICMP** (returning messages to the sending host), **switching**, **longest prefix matching**, **IP sanity-check** (ensuring minimum length and checksum), and other vital IP forwarding functionalities.
- Implemented **ping** and **traceroute** operations, and enabled file downloads using HTTP from designated application servers.
- Implemented **Trie-based Longest Prefix Match**, achieving **90%** improvement over brute force method for **1000+** IPv4 addresses.

**Nachos Operating System Implementation in Java** Sep 2022 – Nov 2022
- Executed the development of the **Alarm class**, implementing **waitUntil**, **timerInterrupt**, and **cancel**, as well as **KThread.join**.
- Employed **interrupt disable and restore techniques** to ensure atomicity while implementing **condition variables**.
- Implemented advanced memory management techniques such as **Demand Paging, Lazy Loading**, and **Page Pinning**.