**CSCI 492: Senior Project I**

**Cloud Based Telemetry System for SDNs**

**Initial Project Proposal Document**

**Team Members:**

**1) John Aulich, CSCI**

**2) Chase Bacon, CSCI**

**3) Cole Hardy, CSCI**

**4) Cheydan Hauf, CSCI**

**5) Adam Roy, CSCI**

**Project Advisor: Dr. Jun Liu**

**September 20th, 2024**

# 

# Table of Contents

[**1. Project Objectives 3**](#_heading=h.1fob9te)

[**2. Background 3**](#_heading=h.3znysh7)

[**3. Contributions 3**](#_heading=h.2et92p0)

[3.1. Development of a Dynamic Load Balancer 3](#_heading=h.tyjcwt)

[3.2. Real-Time Traffic Management and Dynamic Reconfiguration 3](#_heading=h.3dy6vkm)

[3.3. Scalability & Flexibility 3](#_heading=h.1t3h5sf)

[3.4. Capstone Worthiness 4](#_heading=h.4d34og8)

[**4. Broader Impacts & Relevance 4**](#_heading=h.2s8eyo1)

[**5. Approach & Methodology 4**](#_heading=h.17dp8vu)

[5.1. Design, Architecture, and Implementation 4](#_heading=h.3rdcrjn)

[5.1.1. Load Balancing Logic. 4](#_heading=h.26in1rg)

[5.1.2. Deployment & Testing. 4](#_heading=h.lnxbz9)

[5.2. Resources and Tools 5](#_heading=h.35nkun2)

[5.3. Testing and Evaluation 5](#_heading=h.1ksv4uv)

# Project Objectives

The goal of this project is to advance research into cloud-based network telemetry infrastructure by designing and developing a prototype load balancer. This load balancer will dynamically adjust traffic paths based on real-time network conditions to increase data throughput and reduce congestion. Integrated with existing SDN architecture, it will continuously monitor traffic paths, utilizing telemetry data to identify performance issues and implement dynamic traffic flow adjustments to enhance scalability and overall network efficiency.

# Background and Existing Work

Last year’s students made significant progress in building a cloud-based telemetry system at UND, integrating open-source tools like Kafka and GoFlow2 to work in a Software Defined Networking (SDN) system where Open vSwitch (OVS) switches are adopted to forward traffic flows under the control of the OpenFlow protocol. Each OVS switch has been configured to periodically report its observations about traffic flows passing through this switch. The reports are formatted with respect to the specification of the IPFIX protocol. Each OVS switch is configured to send its report to a GoFlow2 instance which is included in a cluster of collectors. A GoFlow2 instance primarily performs the conversion from the IPFIX format to another format which can be suitably further processed in the Kafka subsystem. A cluster of Kafka consumers are deployed in a Kafka subsystem to collaboratively construct the global events based on the local observations reported by the OVS switches. Each Kafka consumer is designated to form a partial result of a global event based on its limited knowledge regarding a global event. The collaborative construction of global events among Kafka consumers relies on the message-passing mechanism provided by the Kafka framework. The partial results are sent through the Kafka message-passing mechanism to the suitable Kafka consumers for continued construction. The network telemetry system outputs fully constructed global events.

# Contributions

## Development of a Dynamic Load Balancer

The primary contribution of this project is the creation of a dynamic load balancer that integrates SDN and Kafka technologies. This addresses a gap in the market for network management by enabling real-time traffic management without the need for costly proprietary software or manual intervention. By leveraging Kafka’s real-time data streaming capabilities, which provide continuous updates on network traffic and resource usage, the load balancer enhances decision-making and traffic optimization.

## Real-Time Traffic Management and Dynamic Reconfiguration

Current off-the-shelf solutions often lack scalability and flexibility, struggling to manage real-time traffic effectively. By integrating SDN's centralized control with Kafka's streaming features, this load balancer will dynamically adjust traffic paths based on live network conditions, reducing latency and ensuring efficient resource allocation. The combination of these technologies allows for dynamic reconfiguration of the network, ensuring the system can adapt in real-time to optimize performance.

## Scalability & Flexibility

The integration of SDN and Kafka offers a scalable and flexible load balancing solution that can be applied across a range of environments, from small business networks to large-scale cloud systems. Unlike proprietary load balancers, this open-source solution can be customized with specific algorithms tailored to unique network demands, providing a cost-effective alternative for businesses seeking adaptive traffic management.

## Capstone Worthiness

This project’s complexity, involving the integration of Kafka, SDN controllers, and custom load balancing algorithms, makes it a strong candidate for a capstone project. It requires a deep understanding of network systems, event-driven architectures, and real-time decision-making processes. By addressing these challenges, the project aims to push the boundaries of traditional load balancing and introduce innovative approaches to network management.

# Broader Impacts & Relevance

The successful implementation of this project offers environmental, social, and economic benefits while advancing network management technology. Dynamic load balancing improves resource efficiency, reducing energy consumption and operational costs by minimizing the waste caused by idle or overloaded servers, aligning with green computing principles. Socially, the load balancer can enhance the reliability of critical infrastructure, including emergency response systems and hospitals, ensuring these networks remain operational during high demand. Economically, the open-source solution makes advanced network management accessible to small and medium-sized enterprises (SMEs), enabling them to scale efficiently without costly proprietary software. This project contributes to the current state of the art by offering a scalable, real-time load balancing solution that optimizes performance in distributed networks, providing a cost-effective alternative to existing proprietary systems for cloud-based architectures.

# Approach & Methodology

## Design, Architecture, and Implementation

### *Load Balancing Logic* - The load balancer, developed in Go, will leverage its concurrency model to handle multiple telemetry data streams simultaneously. Several algorithms will be implemented, including Round Robin, Least Connection, and Source IP Hash, which will optimize traffic flow based on real-time telemetry data. These algorithms will allow the system to adjust traffic paths dynamically, improving network performance under varying conditions. The collected telemetry data will inform the load balancing decisions, ensuring that the network adapts to changes in traffic patterns in real time.

### *Deployment & Testing* - The system will be containerized using Docker, making it easy to deploy and test in different environments. This containerized approach ensures consistency and flexibility across platforms, allowing the load balancer and related services like Kafka and GoFlow2 to be quickly set up. Testing will be conducted in a simulated SDN environment using Mininet, where various traffic loads will be generated. Key metrics such as latency, packet loss, and link utilization will be tracked to evaluate the effectiveness of the load balancing algorithms. The system’s ability to handle real-time telemetry data processing and make dynamic traffic adjustments will also be thoroughly assessed.

## Resources and Tools

* *Go Language*: The primary programming language used for developing the load balancer and processing telemetry data.
* *Kafka*: A distributed streaming platform used for handling real-time telemetry data.
* *GoFlow2*: A telemetry tool used for collecting flow records from Open vSwitch (OVS) instances.
* *Docker*: Containerization technology for deploying the load balancer and other microservices, enabling flexible testing environments.
* *Mininet*: A network simulator used to emulate the SDN environment and generate test traffic.
* *OpenFlow*: The protocol used by the SDN controller to manage and modify flow tables in OVS instances.
* *Virtual Machines (VMs)*: Cloud-based VMs will host the OVS instances and SDN controllers, simulating a distributed cloud environment.

## Testing and Evaluation

The system will be deployed in a simulated SDN environment using Mininet and Docker containers. Network traffic will be generated to simulate different loads, testing the load balancer's ability to redistribute traffic dynamically. Key metrics such as latency, link utilization, and packet loss will be tracked to evaluate system performance and validate the effectiveness of real-time telemetry-driven traffic management.