# Computer Networks (CS3530) Project Report

# TCP connection monitoring for an application deployed as microservice in a Kubernetes cluster

December 2, 2024

## Objective

The objective of this project is to design and implement a TCP Connection Monitoring Framework for an application deployed as a microservice in a Kubernetes cluster. The framework collects detailed TCP connection statistics, enabling analysis of network performance. Additionally, it evaluates the impact of monitoring on the application's throughput and latency, providing insights into trade-offs between performance and observability. Furthermore, an evaluation study is conducted to compare system performance with and without connection monitoring.

## System Design

### Components

1. **Data Aggregator:**
    - A Flask-based microservice collects and aggregates TCP connection statistics from nodes in the Kubernetes cluster.
    - Provides a `/collect` endpoint to receive data and a `/aggregate` endpoint to retrieve aggregated logs.
    - Implements logging for debugging and classifies TCP connection states (e.g., `ESTABLISHED`, `TIME_WAIT`).
2. **TCP Monitor:**
    - A Python-based monitoring script runs on each Kubernetes node as a DaemonSet.
    - Extracts TCP connection statistics using `psutil` and sends the data to the aggregator at regular intervals.
3. **Kubernetes Configuration:**

- The Data Aggregator runs as a `Deployment` with one replica and exposes its services via a `ClusterIP` Service.
- The TCP Monitor runs as a `DaemonSet` ensuring one instance runs on each cluster node.

### Workflow

1. Each node collects TCP statistics, including connection state, local and remote addresses.
2. Node data is sent to the Data Aggregator at regular intervals (every 5 seconds).
3. Aggregated data is classified for TCP issues and stored in memory for retrieval.

## Deployment Details

### Data Aggregator

- **Deployment:**
  - Python 3.8-based container.
  - Configured via a Kubernetes `ConfigMap` containing the aggregator script.
- **Service:**
  - Exposes the Flask application on port `5000` using a `ClusterIP` Service.

### TCP Monitor

- **DaemonSet:**
  - Python 3.9-based container.
  - Collects TCP statistics on each node.
  - Configured via a Kubernetes `ConfigMap` containing the monitoring script.

## Testing and Results

### Environment

- Kubernetes cluster deployed using `Minikube` with:
  - Data Aggregator deployed as a `Deployment`.
  - TCP Monitor deployed as a `DaemonSet`.

## Steps

1. Deployed the Data Aggregator and TCP Monitor in the cluster.
2. Simulated TCP connections to generate data.
3. Verified data aggregation and classification by querying the `/aggregate` endpoint.
4. Conducted a comparative evaluation with and without connection monitoring.
5. Measured and compared application throughput and latency in both scenarios.

## Observations

- The system successfully collected and classified TCP connection statistics.
- Logs included details on `ESTABLISHED` connections and potential issues.
- Data aggregation introduced minimal latency, ensuring timely analysis.
- Without monitoring, the application achieved higher throughput and lower latency due to the absence of monitoring overhead.
- With monitoring, slight increases in latency and reduced throughput were observed, attributed to the additional resource usage for collecting and transmitting TCP statistics.

# With Monitoring

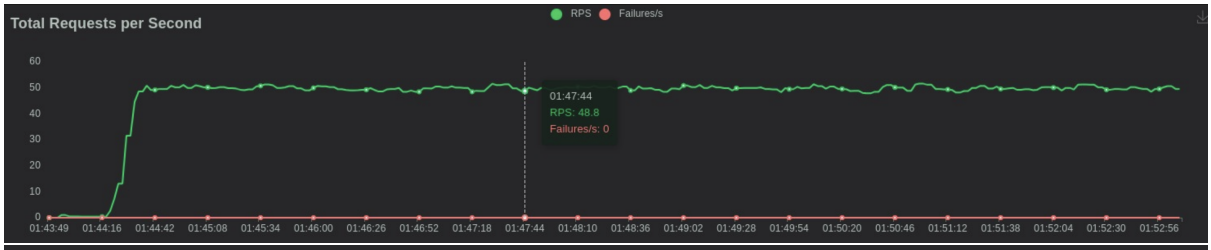

## Request Statistics

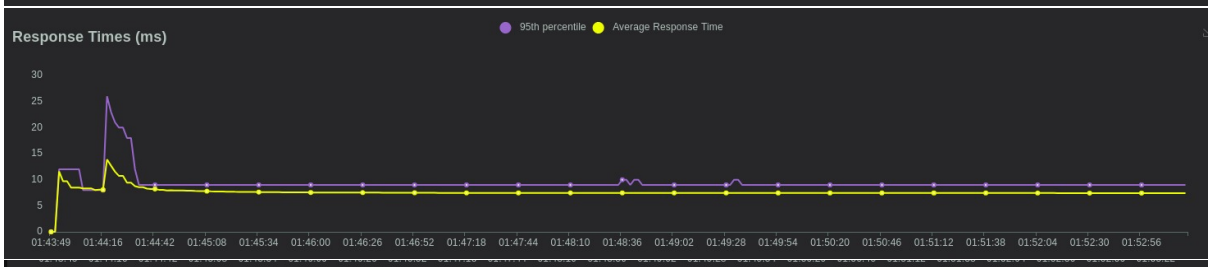
| Type | Name | # Requests | # Fails | Median (ms) | Average (ms) | Min (ms) | Max (ms) | Average size (bytes) | Current RPS | Current Failures/s |
|---|---|---|---|---|---|---|---|---|---|---|
| POST | /collect | 150257 | 0 | 7 | 8.54 | 5 | 122 | 21 | 48.9 | 0 |
| | Aggregated | 150257 | 0 | 7 | 8.54 | 5 | 122 | 21 | 48.9 | 0 |

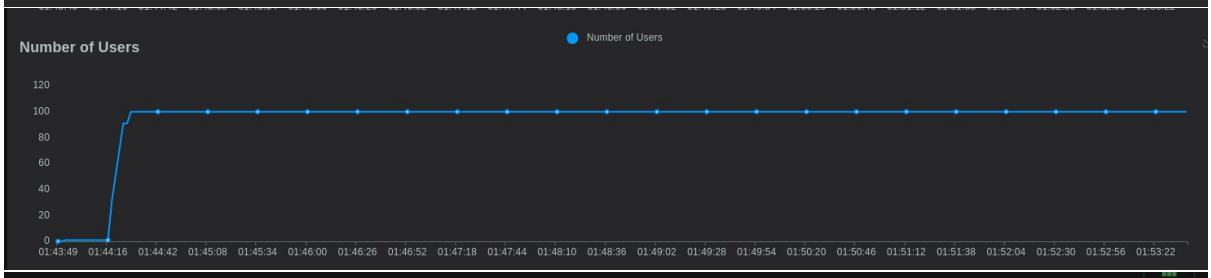
## Response Time Statistics

| Method | Name | 50%ile (ms) | 60%ile (ms) | 70%ile (ms) | 80%ile (ms) | 90%ile (ms) | 95%ile (ms) | 99%ile (ms) | 100%ile (ms) |
|---|---|---|---|---|---|---|---|---|---|
| POST | /collect | 7 | 8 | 8 | 8 | 14 | 17 | 22 | 120 |
| | Aggregated | 7 | 8 | 8 | 8 | 14 | 17 | 22 | 120 |

## Without Monitoring




| Type | Name | # Requests | # Fails | Median (ms) | 95%ile (ms) | 99%ile (ms) | Average (ms) | Min (ms) | Max (ms) | Average size (bytes) | Current RPS | Current Failures/s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| POST | /collect | 29195 | 0 | 7 | 9 | 11 | 7.44 | 5 | 27 | 21 | 51.1 | 0 |
| | Aggregated | 29195 | 0 | 7 | 9 | 11 | 7.44 | 5 | 27 | 21 | 51.1 | 0 |

# Conclusion

The TCP monitoring framework successfully collects and aggregates real-time insights into cluster-wide TCP connections, demonstrating its effectiveness in identifying connection states and potential network issues. The modular design ensures flexibility, enabling easy integration of additional metrics such as retransmissions and RTT in the future. Enhanced classification logic can further improve the system's diagnostic capabilities.

A comparative evaluation highlighted the trade-offs between monitoring and system performance. While monitoring introduced minor overhead, it provided valuable visibility into network behavior, making it a scalable and extensible solution for Kubernetes-based microservice environments.