AegisLights: Self-Adaptive Signal Control Project Proposal

Group 8: Prabhjyot Singh, Yixing Zhang

I. Abstract

Urban cities still rely largely on fixed-time or locally actuated traffic signal plans that struggle under non-stationary demand and unexpected disturbances (incidents, lane closures, events). Recent self-adaptive examples such as ATRP and TRAPP demonstrate adaptation through vehicle local decisions (individual or decentralized route choice), achieving lower travel time by steering drivers through the network. However, this project targets the infrastructure side of the problem. We treat traffic intersections themselves as the managed system and adapt traffic signals at runtime using a MAPE-K loop.

Our system is designed to add a managing system to a simulated section of a real city. We will model intersections as nodes in a graph and commuter routes as edges. The managing system will monitor queue lengths and spill back such that no intersections are blocked (in addition to other indicators) and update timing plans at cycle boundaries with rollback if performance indicators regress. This project extends beyond routine adaptation by being incident-aware. When a crash or blockage is detected, signals are adapted in real time to divert the flow of traffic, effectively routing flow away without controlling driver behavior. Through simulation and data-driven adaptation, our goal is to demonstrate measurable reductions in overall commute time and waiting time per intersection.

We hypothesize that infrastructure-level adaptation can cut average and p95 trip time versus fixed-time and actuated baselines. Additionally, incident-aware planning reduces commute time to clear the impacted region and how often spill backs occur. Lastly, we want to demonstrate that decentralized planners at each intersection with some knowledge sharing achieve better performance than a centralized controller. We will evaluate a modified SUMO/CityFlow micro-network (4–16 intersections) across scripted rush-hour and incident scenarios, reporting average commute time on varying starts. By focusing on signals rather than routes, AegisLights is complementary to ATRP/TRAPP and highlights infrastructure based self-adaptation with runtime safety.

II. Problem Description

Every city experiences highly variable demand on its roads during commute peaks, variable weather and incidents such as collisions or stalled vehicles. Fixed time coordination only performs well only near the load it was designed for, even reactive control is reactive and local. This design system often fails to prevent long queues at signals and queue spillback into intersections that further results in traffic problems. To solve this issue and achieve travel time reduction without costly infrastructure expansion, a self-adaptive approach may be the best option. It can sense emerging conditions, plan adjustments that follow safety/legal rules and continuously learn better responses to situations. The traffic rules for each intersection will vary according to local laws and regulations. These intersection-specific traffic rules, together with the directed graph representing the area map, will serve as inputs to our program.

A. Managed System and Adaptation Controls

The managed system for this project is an urban network of intersections (4–16 nodes), using a local set of intersections in Waterloo. Adaptation will act on green splits, cycle length, and inter-signal offsets. This choice of control mechanisms are deliberately different from vehicle-centric adaptation (e.g., ATRP/TRAPP) as we do not assign routes to cars and rather aim to reshape capacities and priorities at junctions.

B. Adaptation Goals

The primary adaptation goal for the system is to minimize the average trip time and the p95 (95th percentile for outlier consideration) time for commuters. Additionally, we want to minimize the time it takes to clear incidents during varying loads. Lastly we want to minimize spillback events over safety concerns and stop count for vehicles.

C. Control & Safety Constraints

To avoid violating any laws and ensuring the safety of people when this system is active, some control and safety constraints must be put in place. Normal traffic laws should apply and any given intersection should not have all green or red signals. Changes to the signal timings need to apply at cycle boundaries to avoid sudden changes. Lastly, there needs to be a sense to revert to a verified plan if performance indications show a degradation over a few cycles.

D. Research Questions

There are two main questions we want to answer through this project:

- **RQ1**: Can signal-level self-adaptation (MAPE-K) outperform fixed-time baselines on avg/p95 trip time under varying demand?
- **RQ2**: How much does incident-aware planning reduce commute time versus incident-unaware adaptation?

E. Assumptions & Scope

For this project, to limit the scope and to be able to answer the research questions proposed some assumptions are needed. Firstly, no direct vehicle control will be applied and benefits in commute time are based on prioritization and capacity changes only. Additionally, evaluation will use multiple random seeds at the same load to ensure sufficient statistical coverage. Lastly, the traffic represented in the simulator will be limited to cars and pedestrians with pedestrians having an intersection to intersection route with cars having entrance to exit route generation.

III. Proposed Solution

To develop a system to target infrastructure self-adaptation rather than telling cars how to route (as in ATRP and TRAPP, which optimize vehicle decisions), we adapt the signal network itself. Effectively, we model a small urban grid of signals and roads as a directed graph where nodes are intersections and edges are approaches, stream relevant data from a simulator (queues, delays, spill back, incidents) to then reweigh the graph each cycle to reflect congestion risk. These steps then are used to plan safe signal updates for green splits and offset coordination along dynamically discovered bypasses. This shifts the adaptation controls from drivers to signals under legal/safety constraints. Our hypothesis is that infrastructure adaptation can reduce average and p95 trip time and time needed to clear incidents without controlling driver behavior which was the case for ATRP/TRAPP.

A. Graph Representation

To enable the system to be applied to any set of intersections, we are aiming to use a directed graph to represent the intersections and roads. We can model the road network as a directed graph

$$G = (V, E).$$

Each node represents an intersection and each directed edge represents a road or approach with the following properties that can be published to the managing system:

- **Capacity** (veh/s): approximate discharge rate when the approach is served.
- **Free-flow travel time** (s): time along the road with no congestion.
- **Current queue length** (veh) and the related **delay** (s/veh).
- **Spillback flag**: boolean indicating the queue is blocking an upstream intersection.
- **Incident flag**: boolean indicating a lane closure, crash, or obstruction on the edge.

Given a small city area, we will pick a bounding box on the map and create a node for every signalized intersection inside it and an edge for every directed road between adjacent intersections that we can attach the above attributes to. Each node would then be controlled such that the signal timings at the intersection can be modified for maximum throughput. Looking at Figure 2 as an example, it would be a representation of the map shown in Figure 1 and how intersections with signals can be mapped to a directed graph.

We calculate the total travel time of commuter k by doing the following:

$$T^k = \sum_{(v_i, v_j) \in \text{route}_k} \left(t_{ij}^x + t_{x_{\text{delay}}}(v_i) \right)$$

where $x \in \{c, p\}$ indicates whether the commuter is a car or pedestrian, t_{ij}^x is the average travel time, and $t_{x_{\text{delay}}}(v_i)$ is the waiting delay at intersection i.

The system goal is to minimize the average commute time:

$$T_{\text{avg}} = \frac{1}{N} \sum_{k=1}^{N} T^k$$

where N is the total number of commuters in the system.

To expose traffic to the algorithms without heavy notation, we assign every edge a dynamic cost:

$$w_e(t) = a \cdot \text{delay}_e(t) + b \cdot \text{queue}_e(t) + c \cdot 1[\text{spill back}_e(t)] + d \cdot 1[\text{incident}_e(t)].$$

where $a, b, c, d > 0$ are simple tunable coefficients. More delay/queue implies a higher cost with a spill back or incident immediately inflating the cost.

At each control cycle the managing system re-computes $\{w_e(t)\}$ and:

1) Finds risky areas by locating edges with large $w_e(t)$ (often the ones spilling back).
2) Suggests bypasses by running a k-shortest-paths search on the re-calculated graph (low total cost means a likely faster path).
3) Coordinates signals by reducing green splits on high-cost edges feeding a hotspot, and favouring greens along low-cost edges to create a temporary green wave.

B. MAPE-K Overview

We treat the simulated network of traffic as the managed system and implement a MAPE-K controller as a separate managing system. The simulator continuously outputs data regarding queue length, delay, throughput, spillback and incident flags. The controller will also be route independent, never seeing individual driver routes or intentions, only what the sensors report. Therefore all decisions are based on aggregated edge-level signals.

1) **Monitor.** At each control cycle t, the Monitor ingests data from every directed road edge. It timestamps and normalizes units and computes short rolling aggregates (e.g., 4–8 recent steps) to smooth noise. The Monitor then updates the runtime graph state G_t by writing these values as edge attributes, ensuring the rest of the loop always reasons over a recent snapshot of network conditions.
2) **Analyze.** Using the graphical representation G_t, the Analyze stage computes a simple, re-weighted cost

Fig. 1. Real life intersections

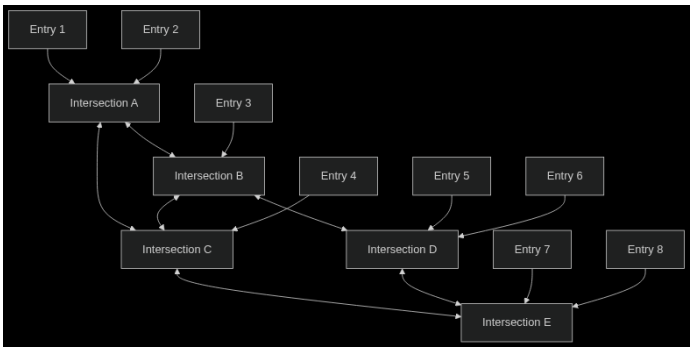

Fig. 2. Figure 2 mapped as a directed graph

$w_e(t)$ per edge to identify congestion and disruption. With these weights it can highlight risky edges and potential cuts where queues may spill back. Additionally, running a k-shortest-paths search on the re-weighted graph to determine low-cost routes around hot spots. A lightweight trend estimate (e.g., exponential smoothing) provides $\Delta Q_e(t)$ to anticipate expected growth. The output is a compact set of targets which includes edges to throttle, edges to favor and groups to coordinate.

3) **Plan.** The Plan stage converts targets into safe timing decisions. Each intersection v has a set of plans that have been verified, $\mathcal{P}_v$ (legal movements, min/max greens, pedestrian minimums). Our primary method uses contextual bandits that, per cycle, select a plan $p \in \mathcal{P}_v$ maximizing expected reward (negative local delay with spill back penalties) given the local context and route membership. A lightweight bypass coordinator proposes offsets for progression. When an incident is active, the planner switches to an incident mode and it reduces splits feeding blocked edges and increases splits and align offsets along selected edges to clear areas of high traffic. Objectives trade off average travel time, p95 travel time, and spill back count.
4) **Execute.** The Execute stage applies changes only at cycle boundaries to prevent sudden changes. Before acting, it validates constraints such as no conflicting greens, amber/all-red clearance, pedestrian minimums, and rate-of-change limits on splits/offsets to avoid causing further traffic. A watchdog evaluates rolling performance indicators. If utility degrades beyond a threshold for N consecutive cycles, Execute rolls back to the last-known-good configuration. All actions and results are logged for analysis and reproducibility.
5) **Knowledge.** The Knowledge base persists the current graph configuration and attributes, phase libraries and safety rules per intersection, goal weights and performance thresholds and snapshots of last-known-good signal settings. This store makes adaptation explainable and progressively better over time.

IV. Block Diagram

The block diagram covers all of the components mentioned as part of the problem solution.

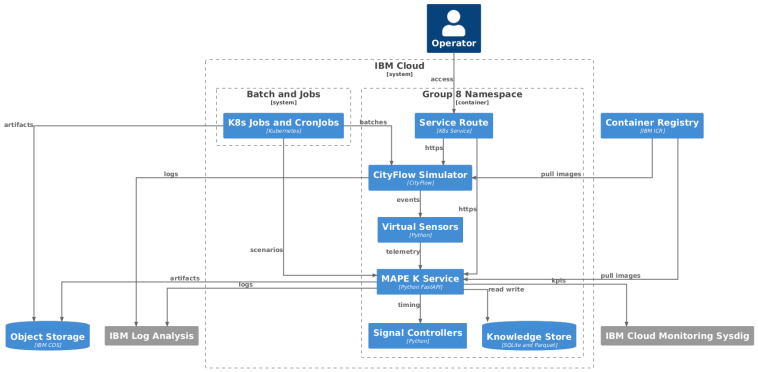

Fig. 3. Architectural Diagram

V. Used Tools / Technologies

To implement this project, we will need to utilize a variety of tools and technologies that will help us simulate traffic, implement and run the MAPE-K loop, and be able to host this in a cloud environment to test in a extensive manner. These tools and their use cases are outlined in Table 1.

TABLE I
AEGISLIGHTS TOOLS & TECHNOLOGIES

Category	Purpose	Technologies / Notes
Traffic Simulator	Traffic simulation, per-edge data, external signal updates	**CityFlow** (Python API): hooks for queues, delays, throughput; spillback/incident flags, incident injector, cycle-boundary actuation
Managing System (MAPE-K)	Implements Monitor/Analyze/Plan/Execute + Knowledge store	**Python + FastAPI**: endpoints `/ingest`, `/analyze`, `/plan`, `/execute`, `/metrics`
Graph & Analytics	Graph abstraction, bypass discovery, cut detection, trends	**NetworkX** (k-shortest paths, clustering), NumPy/Pandas, smoothing / simple linear models
Planning Algorithms	Safe timing selection and bypass offset coordination	Primary: **Contextual Bandits** (UCB/TS) over phase-plan libraries
Knowledge & Storage	Persist models, libraries, outcomes, metrics	**SQLite**: metadata, data/results
Visualization	Plots and minimal UI for replays	**Plotly**/Matplotlib for easy integration
Containerization	Package simulator and manager	**Docker** images for both the simulator and manger that can be deployed
IBM Cloud (Compute)	Host services run large simulation batches	**OpenShift/Kubernetes**: always-on MAPE-K
IBM Cloud (Registry & Storage)	Store images and outputs	**IBM Cloud Container Registry (ICR)**; **IBM Cloud Object Storage (COS)** for logs, Parquet, plots

VI. PROJECT SCHEDULE

The project schedule is detail in Table 2, with focus dates, Key Tasks, and Deliverables detailed.

TABLE II
7-WEEK DEVELOPMENT PLAN

Week	Focus Area	Key Tasks and Deliverables
Week 1 (Oct 20 – Oct 26)	**Project Setup & Design**	• Complete Project Proposal • Finalize detailed design of intersection graph model and simulation parameters • Setup CityFlow simulation environment and initial test network
Week 2 (Oct 27 – Nov 2)	**MAPE-K Architecture**	• Create MAPE component interfaces (Monitor, Analyze, Plan, Execute) • Implement Monitor & Analyze parts of our self-adaptive system (data ingestion, utility function)
Week 3 (Nov 3 – Nov 9)	**Planning & Execution Module**	• Implement Planner (contextual bandits or heuristic logic) • Implement Executor with rollback & safety constraints
Week 4 (Nov 10 – Nov 16)	**Integration & Simulation & Testing**	• Integrate CityFlow with MAPE-K controller • Run baseline scenarios (fixed-time, adaptive) • Record results, queue statistics, and performance baselines
Week 5 (Nov 17 – Nov 23)	**Results Analysis & Optimization**	• Analyze results (compare adaptive vs. baseline) • Visualize results with plots • Optimize model parameters and control logic
Week 6 (Nov 24 – Dec 1)	**Final Presentation**	• Prepare Final Presentation • Record demonstration video or simulation replay (if required)
Week 7 (Dec 2 – Dec 8)	**Final Report**	• Prepare Final Report

REFERENCES

[1] I. Gerostathopoulos and E. Pournaras, "TRAPPed in Traffic? A Self-Adaptive Framework for Decentralized Traffic Optimization," in *Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2019, pp. 32–38. [Online]. Available: https://ieeexplore.ieee.org/document/8787057

[2] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy, "Traffic routing for evaluating self-adaptation," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '12)*, Zurich, Switzerland, 2012, pp. 27–32. [Online]. Available: https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/model-problem-atrp/

[3] "CityFlow: A Multi-Agent Reinforcement Learning Environment for Large-Scale City Traffic Scenario," [Online]. Available: https://github.com/cityflow-project/CityFlow/

[4] "How to monitor openshift with Sysdig Monitor." Sysdig. [Online]. Available: https://sysdig.com/blog/how-to-monitor-openshift/

[5] "Introducing CRC." CRC Documentation. [Online]. Available: https://crc.dev/docs/introducing/

[6] "OpenShift container platform. Chapter 2. OpenShift CLI (oc) — CLI tools — OpenShift Container Platform — 4.12 — Red Hat Documentation." [Online]. Available: https://docs.redhat.com/en/documentation/openshift_container_platform/4.12/html/cli_tools/openshift-cli-oc#cli-getting-started