CERIAS

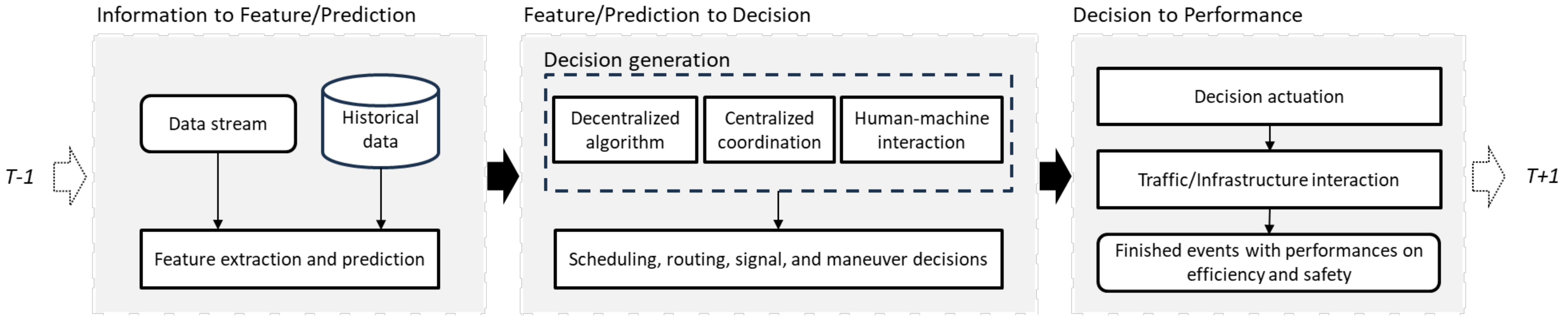
The Center for Education and Research in Information Assurance and Security

A Cybersecurity Testbed for Connected and Autonomous Vehicle Systems

Zengxiang Lei [1], Ruichen Tan [1], Satish V. Ukkusuri [1]
[1] Lyles School of Civil Engineering, Purdue University

Overview

- The Transportation Cybersecurity and Resilience Center (TraCR) is developing a cybersecurity testbed for connected and autonomous vehicle (CAV) systems.
- The goal of this testbed is to comprehensively evaluate the impact of cyberattacks on CAV systems and assess the efficacy of defense algorithms.
- To achieve this, the testbed incorporates the full life cycles of transportation services (including private vehicles, ride-hailing, and public transit) with realistic data streams.



Information to Feature/Prediction

T-1

Data stream

Historical data

Feature extraction and prediction

Feature/Prediction to Decision

Decision generation

Decentralized algorithm

Centralized coordination

Human-machine interaction

Scheduling, routing, signal, and maneuver decisions

Decision to Performance

Decision actuation

Traffic/Infrastructure interaction

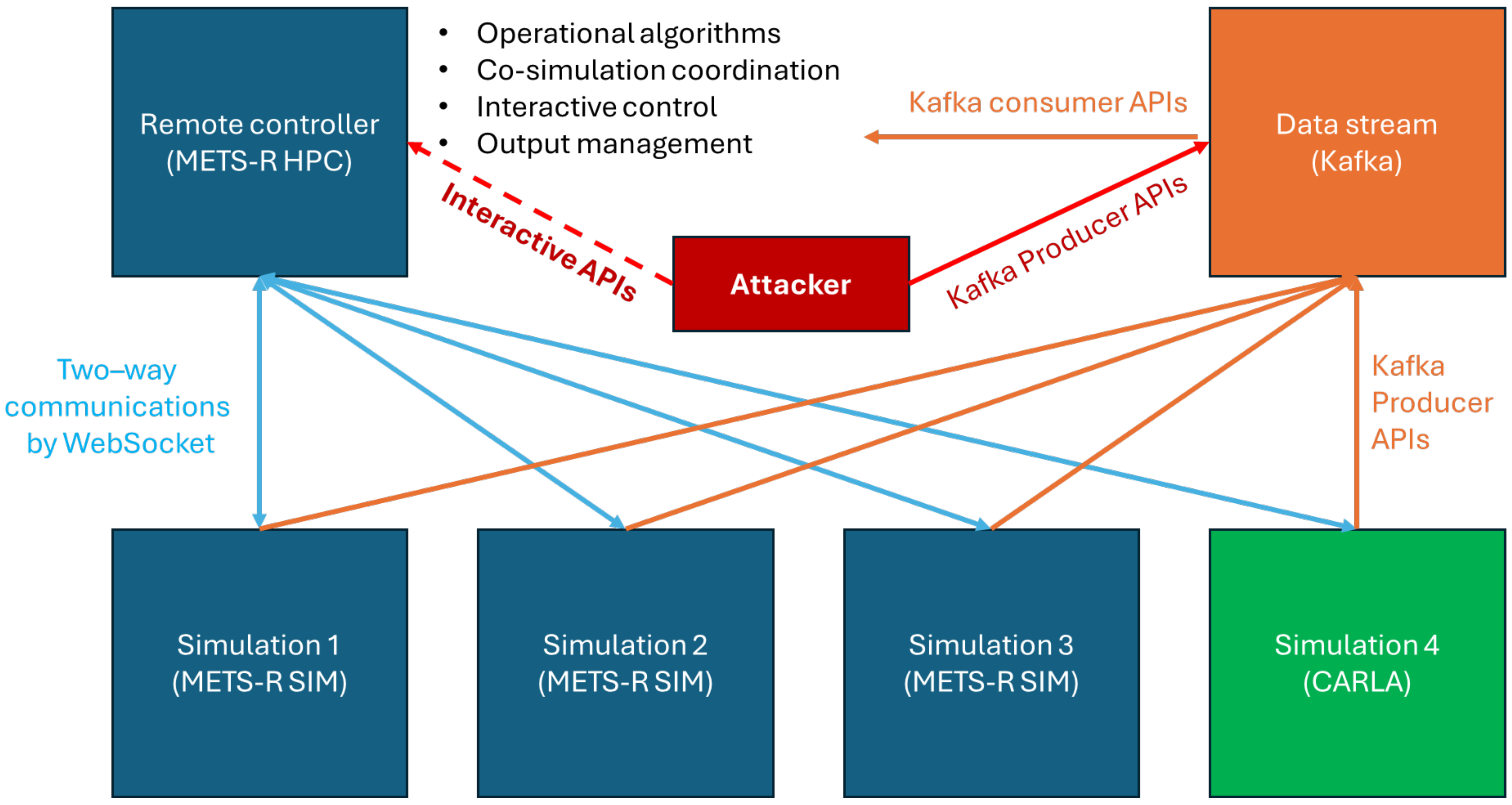
Finished events with performances on efficiency and safety

T+1

- This poster shows the testbed design and current development progress.

Testbed design

- Our framework comprises three key components: simulator, data stream, controller.
- **Simulator**: We use METS-R SIM for large-scale multi-modal traffic simulation and CARLA for photorealistic sensor and driving simulation. This combination ensures comprehensive testing environments.
- **Data Stream**: This Kafka-based component can model the data stream explicitly and expose it to potential attacks like DoS and data spoofing. This also allows us to separate the logics of data processing, decision generation, traffic simulation, and attacks.
- **Controller**: The controller serves as the intermediary between multiple simulation instances and the data stream, this framework guarantees strict synchronization. By broadcasting simulation time among the simulator, data processor, and attacker, it enables sensible testing of attack algorithms with any level of computational costs.
- Open source:
  METS-R SIM: https://github.com/umnilab/METS-R_SIM.git
  METS-R HPC: https://github.com/umnilab/METS-R_HPC.git
  Visualization: https://engineering.purdue.edu/HSEES/METSRVis/



Remote controller (METS-R HPC)

- Operational algorithms
- Co-simulation coordination
- Interactive control
- Output management

Kafka consumer APIs

Data stream (Kafka)

Interactive APIs

Attacker

Kafka Producer APIs

Two–way communications by WebSocket

Kafka Producer APIs

Simulation 1 (METS-R SIM)

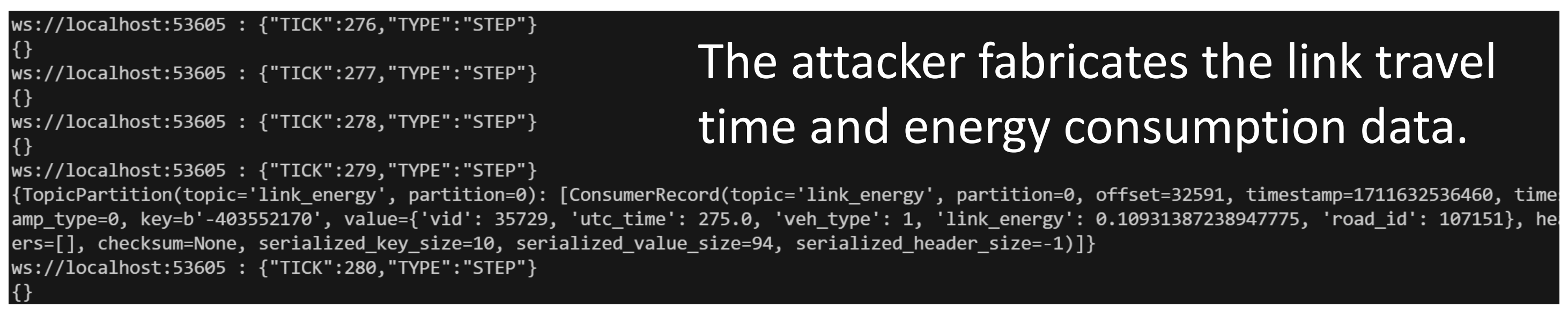
Simulation 2 (METS-R SIM)

Simulation 3 (METS-R SIM)

Simulation 4 (CARLA)

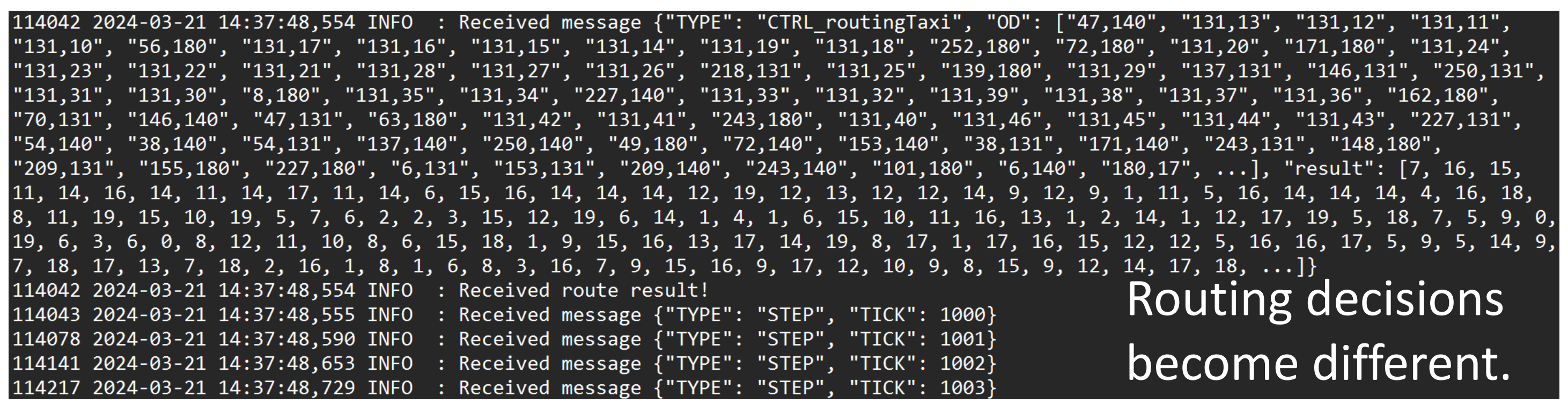
Use case examples

- Testing attacks against the online routing algorithm for CAVs.



```
ws://localhost:53605 : {"TICK":276,"TYPE":"STEP"}
{}
ws://localhost:53605 : {"TICK":277,"TYPE":"STEP"}
{}
ws://localhost:53605 : {"TICK":278,"TYPE":"STEP"}
{}
ws://localhost:53605 : {"TICK":279,"TYPE":"STEP"}
{TopicPartition(topic='link_energy', partition=0): [ConsumerRecord(topic='link_energy', partition=0, offset=32591, timestamp=1711632536460, time
amp_type=0, key=b'-403552170', value={'vid': 35729, 'utc_time': 275.0, 'veh_type': 1, 'link_energy': 0.10931387238947775, 'road_id': 107151}, he
ers=[], checksum=None, serialized_key_size=10, serialized_value_size=94, serialized_header_size=-1)]}
ws://localhost:53605 : {"TICK":280,"TYPE":"STEP"}
{}
```

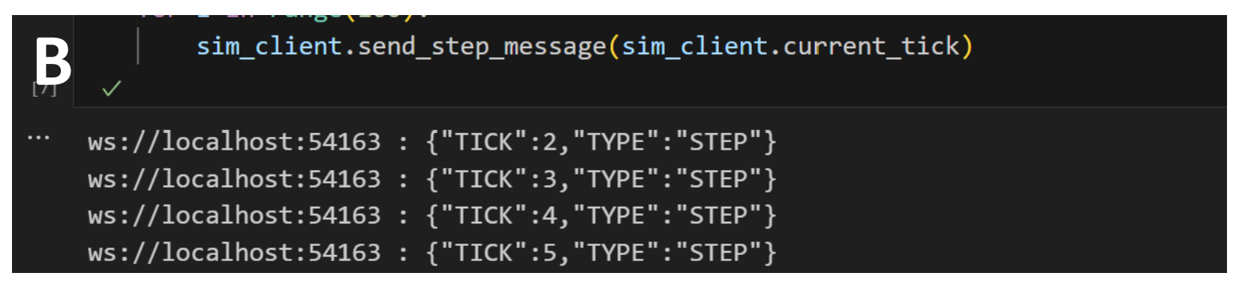
The attacker fabricates the link travel time and energy consumption data.



```
114042 2024-03-21 14:37:48,554 INFO  : Received message {"TYPE": "CTRL_routingTaxi", "OD": ["47,140", "131,13", "131,12", "131,11",
"131,10", "56,180", "131,17", "131,16", "131,15", "131,14", "131,19", "131,18", "252,180", "72,180", "131,20", "171,180", "131,24",
"131,23", "131,22", "131,21", "131,28", "131,27", "131,26", "218,131", "131,25", "139,180", "131,29", "137,131", "146,131", "250,131",
"131,31", "131,30", "8,180", "131,35", "131,34", "227,140", "131,33", "131,32", "131,39", "131,38", "131,37", "131,36", "162,180",
"70,131", "146,140", "47,131", "63,180", "131,42", "131,41", "243,180", "131,40", "131,46", "131,45", "131,44", "131,43", "227,131",
"54,140", "38,140", "54,131", "137,140", "250,140", "49,180", "72,140", "153,140", "38,131", "171,140", "243,131", "148,180",
"209,131", "155,180", "227,180", "6,131", "153,131", "209,140", "243,140", "101,180", "6,140", "180,17", ...], "result": [7, 16, 15,
11, 14, 16, 14, 11, 14, 17, 11, 14, 6, 15, 16, 14, 14, 14, 12, 19, 12, 13, 12, 12, 14, 9, 12, 9, 1, 11, 5, 16, 14, 14, 14, 4, 16, 18,
8, 11, 19, 15, 10, 19, 5, 7, 6, 2, 2, 3, 15, 12, 19, 6, 14, 1, 4, 1, 6, 15, 10, 11, 16, 13, 1, 2, 14, 1, 12, 17, 19, 5, 18, 7, 5, 9, 0,
19, 6, 3, 6, 0, 8, 12, 11, 10, 8, 6, 15, 18, 1, 9, 15, 16, 13, 17, 14, 19, 8, 17, 1, 17, 16, 15, 12, 12, 5, 16, 16, 17, 5, 9, 5, 14, 9,
7, 18, 17, 13, 7, 18, 2, 16, 1, 8, 1, 6, 8, 3, 16, 7, 9, 15, 16, 9, 17, 12, 10, 9, 8, 15, 9, 12, 14, 17, 18, ...]}
114042 2024-03-21 14:37:48,554 INFO  : Received route result!
114043 2024-03-21 14:37:48,555 INFO  : Received message {"TYPE": "STEP", "TICK": 1000}
114078 2024-03-21 14:37:48,590 INFO  : Received message {"TYPE": "STEP", "TICK": 1001}
114141 2024-03-21 14:37:48,653 INFO  : Received message {"TYPE": "STEP", "TICK": 1002}
114217 2024-03-21 14:37:48,729 INFO  : Received message {"TYPE": "STEP", "TICK": 1003}
```

Routing decisions become different.

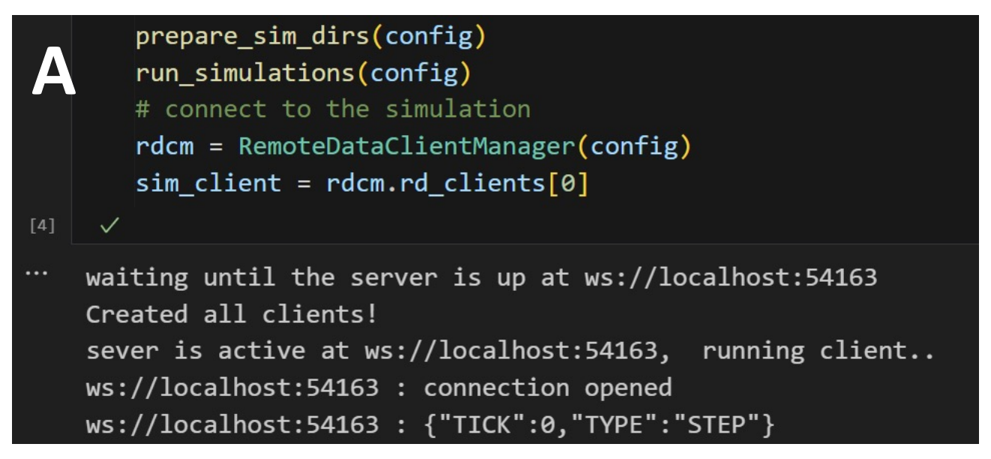
- Running in an interactive environment.



B

```
        sim_client.send_step_message(sim_client.current_tick)
```

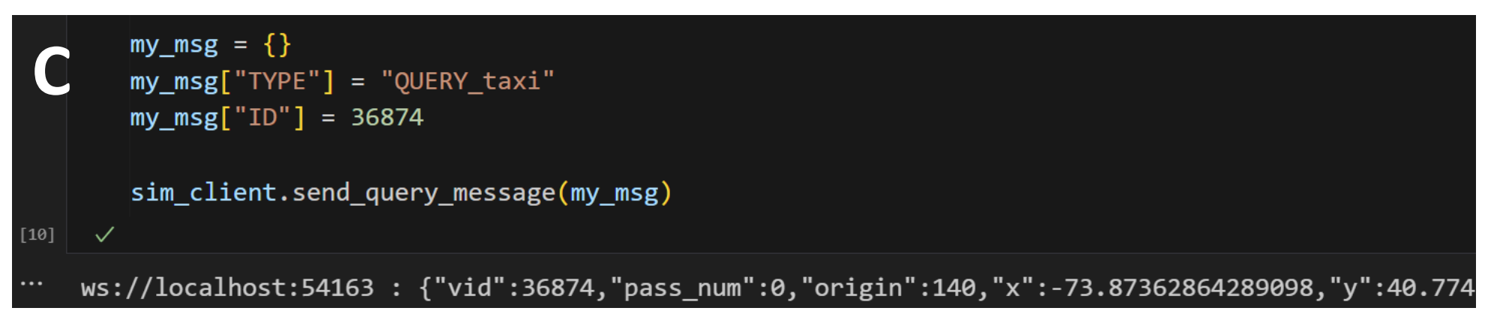
```
ws://localhost:54163 : {"TICK":2,"TYPE":"STEP"}
ws://localhost:54163 : {"TICK":3,"TYPE":"STEP"}
ws://localhost:54163 : {"TICK":4,"TYPE":"STEP"}
ws://localhost:54163 : {"TICK":5,"TYPE":"STEP"}
```



A

```
prepare_sim_dirs(config)
run_simulations(config)
# connect to the simulation
rdcm = RemoteDataClientManager(config)
sim_client = rdcm.rd_clients[0]
```

```
waiting until the server is up at ws://localhost:54163
Created all clients!
sever is active at ws://localhost:54163,  running client..
ws://localhost:54163 : connection opened
ws://localhost:54163 : {"TICK":0,"TYPE":"STEP"}
```



C

```
my_msg = {}
my_msg["TYPE"] = "QUERY_taxi"
my_msg["ID"] = 36874

sim_client.send_query_message(my_msg)
```

```
ws://localhost:54163 : {"vid":36874,"pass_num":0,"origin":140,"x":-73.87362864289098,"y":40.774
```

What’s next?

- Currently, we are working on developing an extensive set of APIs for query/control the simulators.
- The next step is to leverage the platform to test typical attacks (e.g., GPS spoofing) against CAV systems and applications.