Fleet Simulator

https://github.com/dmulholl/fleetsim

Overview

Overview

- Three binaries, written in Go:
  - Vehicle Simulator
  - Fleet State Server
  - Subscriber Client
- All communication via UDP packets.

1. Vehicle Simulator

- Simulates an arbitrary number of vehicles.
- Each simulated vehicle runs in its own goroutine.
- Vehicles send location updates to the fleet server.
- One UDP packet per vehicle per second.
- Not a very realistic simulation!
- Generates the right *kind* of data.

2. Fleet State Server

- Listens for incoming UDP packets:
    - Location updates from vehicles.
    - Subscription requests from clients.
- Sends updates to clients:
    - Vehicle location.
    - Vehicle speed.

3. Subscriber Client

- Subscribes to a feed of updates about a particular vehicle.
- Specifies vehicle by VIN.
- Multiple clients can run simultaneously.
- Multiple clients can subscribe to update feeds for the same vehicle.

Design Decisions

Why UDP?

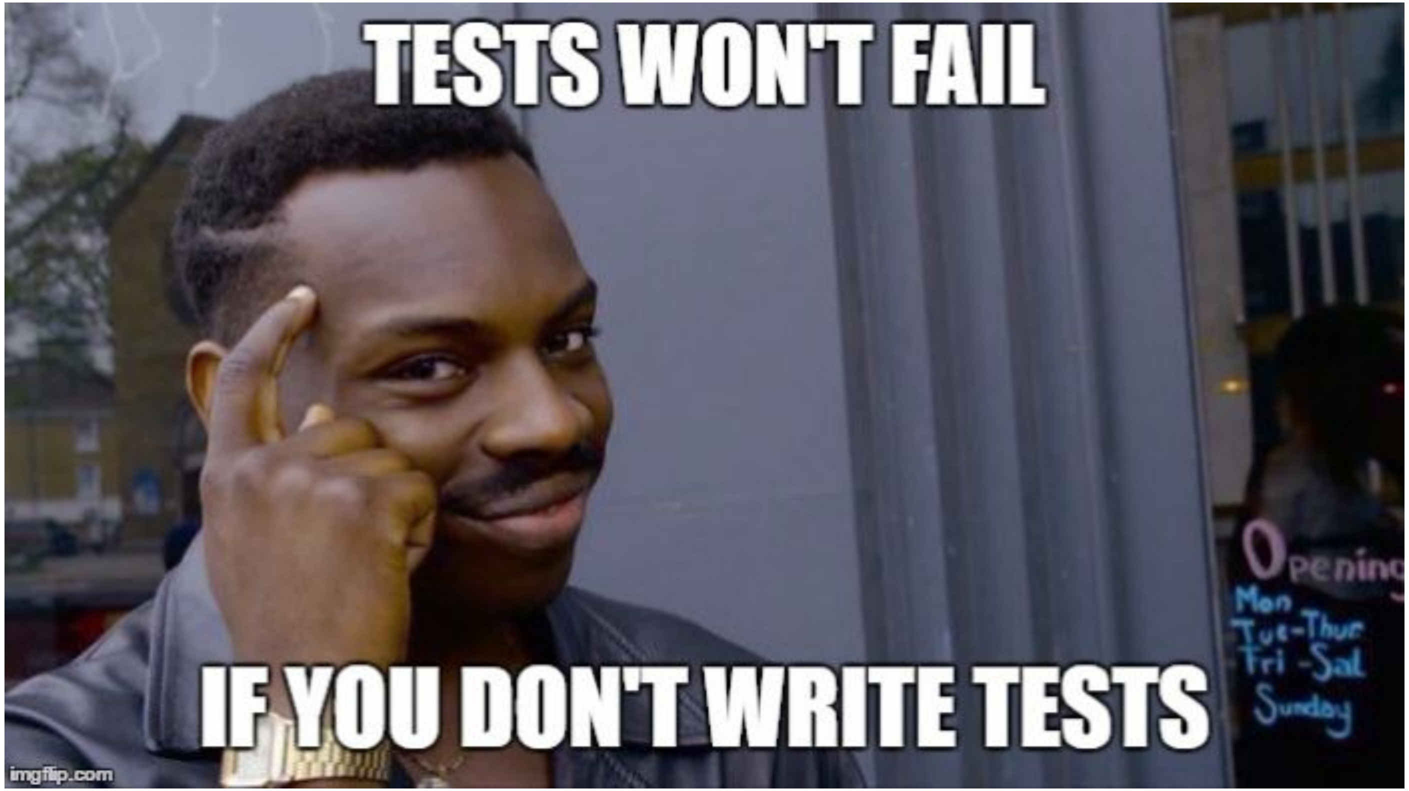
- Lightweight protocol.
- Fast, no TCP handshake overhead.
- Popular choice for real-time systems:
    - Gaming
    - VOIP
    - Audio/Video Streaming

Why UDP?

- No guarantee of delivery.
- No guarantee of order.

UDP is a good choice for time-sensitive applications where dropping packets is preferable to waiting for retransmission.

Testing Strategy


TESTS WON'T FAIL
IF YOU DON'T WRITE TESTS
imgflip.com
Opening
Mon
Tue-Thur
Fri -Sat
Sunday

Issues

Speed Calculation Accuracy

- Depends on accuracy of GPS — approx. 5m radius?
- Significant error margin over short time intervals.
- Calculating speed via GPS is unnecessarily complicated.
- Vehicle knows its own speed accurately.
- Can simply report speed to server.

Security

- All UDP packets in plaintext.
- No authentication of endpoints.
- Easy — can layer encryption on top of UDP.
- Hard — authentication is the same hard problem every communication system faces all the time.

Reliability

- *Unreliable* Datagram Protocol.
- Can design system to be robust in face of dropped packets.
- System needs to be robust in the face of transmission delays anyway.

Scaling

- Difficulty of scaling depends on data processing requirements.
- UDP packet processing unlikely to ever be a serious bottleneck.
- Example: “How to receive a million packets per second”

  `https://blog.cloudflare.com/how-to-receive-a-million-packets/`

Reinventing the wheel?

- Not a unique problem.
- Lots of existing solutions, e.g. Apache Kafka.
- Hard problem is choosing the right dependencies.
- Kafka ⟶ Log4j.