Web Traffic Visualization Project Report

Ildar Rakiev

April 9, 2025

1 Introduction

This report documents the implementation of a web traffic visualization system that displays network packages on a 3D globe. The project consists of three main components:

- A Python script for sending packages from a CSV file
- A Flask server for receiving and managing packages
- A Three.js based frontend for interactive visualization

All components are containerized using Docker for easy deployment and reproducibility.

2 Implementation

2.1 Data Generation (Sender)

The sender component reads package data from a CSV file and sends it to the server with proper timing intervals.

Key features:

- Reads CSV with IP addresses, coordinates, timestamps, and suspicious flags
- Maintains original timing between packages using Python's `time.sleep()`
- Sends data as JSON via HTTP POST requests
- Includes an "end marker" package to signal transmission completion

```python
def read_and_send_packages(csv_file: str) -> None:
    with open(csv_file, mode='r') as file:
        reader = csv.DictReader(file)
        previous_timestamp = None

        for row in reader:
            unix_timestamp = int(row['Timestamp'])
            current_timestamp = datetime.fromtimestamp(unix_timestamp)

            if previous_timestamp is not None:
                delay = (current_timestamp - previous_timestamp).total_seconds()
                time.sleep(delay)

            package = {
                'ip': row['ip address'],
                'latitude': float(row['Latitude']),
                'longitude': float(row['Longitude']),
                'timestamp': current_timestamp.strftime('%Y-%m-%d %H:%M:%S'),
                'suspicious': int(float(row['suspicious']))
            }

            send_package(package)
```

Listing 1: Sender Implementation Highlights

2.2 Data Receiving (Server)

The Flask server handles incoming packages and serves them to the frontend.

Key features:

- Thread-safe storage using `deque` with a maximum capacity
- Two endpoints: POST for receiving and GET for frontend access
- CORS enabled for frontend communication
- Transmission state management
- Serves static files including the frontend

```python
@app.route('/api/packages', methods=['POST'])
def receive_package():
    data = request.get_json()

    if data.get('is_last'):
        with lock:
            global transmission_active
            transmission_active = False
        return jsonify({'status': 'transmission_complete'}), 200

    if not transmission_active:
        with lock:
            packages.clear()
            transmission_active = True

    with lock:
        packages.append(data)

    return jsonify({'status': 'success'}), 200
```

Listing 2: Server Implementation Highlights

2.3 Visualization (Frontend)

The frontend visualization was the most challenging and rewarding part of this project. My goal was to create an interactive 3D globe that not only displays traffic locations but also tells a story about the data patterns. Here's a detailed breakdown of my implementation decisions:

2.3.1 Globe Implementation

I chose Three.js for its robust 3D capabilities and active community support. The globe consists of:

- A high-resolution sphere (128x128 segments) with NASA's Blue Marble texture for realism
- Three light sources to create proper shading:
  - Ambient light (0.6 intensity) for base illumination
  - Two directional lights (0.8 and 0.4 intensity) from different angles
- Phong material with specular highlights to simulate atmospheric glow

```javascript
const earthGeometry = new THREE.SphereGeometry(5, 128, 128);
const earthTexture = textureLoader.load('earth_atmos_2048.jpg');
const earthMaterial = new THREE.MeshPhongMaterial({
    map: earthTexture,
    specular: 0x111111,
    emissive: 0x112244  // Subtle blue glow
});
```

Listing 3: Globe Initialization

2.3.2 Data Points Representation

Each network package is represented as a colored sphere on the globe's surface. I implemented several visual distinctions:

- **Size Differentiation**: Suspicious packages (0.15 units) are 50% larger than normal ones (0.1 units)
- **Color Coding**:
  - Normal traffic: #7bed9f (soft green)
  - Suspicious traffic: #ff6b6b (alerting red)
- **Fade-out Effect**: Points gradually become transparent over 15 seconds before disappearing

The coordinate conversion from latitude/longitude to 3D space required careful mathematical handling:

```
function latLongToVector3(lat, lon, radius) {
    const phi = (90 - lat) * (Math.PI / 180);
    const theta = (lon + 180) * (Math.PI / 180);

    return new THREE.Vector3(
        -radius * Math.sin(phi) * Math.cos(theta),
        radius * Math.cos(phi),
        radius * Math.sin(phi) * Math.sin(theta)
    );
}
```

Listing 4: Coordinate Conversion

2.3.3 User Interaction Features

I implemented five key interaction modes to facilitate data exploration:

1. **Rotation Control**:
   - Auto-rotation (adjustable speed)
   - Manual rotation via mouse drag (using OrbitControls)
2. **Point Inspection**:
   - Raycasting for mouse-over detection
   - Dynamic tooltips showing IP, location, and timestamp
   - Visual highlighting (scale increase + emissive glow)
3. **View Management**:
   - Toggle points visibility
   - Reset camera position
4. **Real-time Statistics**:
   - Package counter with suspicious traffic highlight
   - Top 5 locations by request volume
5. **Performance Optimization**:
   - Limited point lifetime (15 seconds)
   - Maximum of 5000 points stored server-side
   - Efficient rendering loop with selective updates

3 Deployment

The system is containerized using Docker Compose with two services:

- **Server**: Flask application serving on port 5000
- **Sender**: Python script that sends packages from CSV

```yaml
version: '3.8'

services:
  server:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    volumes:
      - ./static:/app/static

  sender:
    build:
      context: .
      dockerfile: Dockerfile.sender
    volumes:
      - ./ip_addresses.csv:/app/ip_addresses.csv
    depends_on:
      - server
```

Listing 5: Docker Compose Configuration

4 Conclusion

The implemented system successfully meets all project requirements:

- Correctly sends packages with proper timing intervals
- Efficiently manages incoming packages on the server
- Provides an interactive and informative visualization
- Offers multiple user interaction features
- Is fully containerized for easy deployment

The visualization quite good represents the geographic distribution of network traffic while distinguishing suspicious activity. The Docker implementation ensures the system can be easily reproduced and deployed in various environments.