Network Observability on a real-time 3D network topology model of devices built via OpenTelemetry distributed tracing

Tech Con 2025 Abstract 147

HPE Edge;

Network Observability on a real-time 3D network topology model of devices built via OpenTelemetry distributed tracing

Abstract

This paper introduces a comprehensive framework for enhancing network observability through the construction of a real-time 3D network service graph topology model of a distributed system of network devices. The model integrates OpenTelemetry traces enriched with device data, facilitating accurate location estimation, and then generating a 3D service graph that provides a detailed and dynamic visualization of the network topology, contributing significantly to the observability and management of complex network infrastructures.

Problem statement

Current network observability tools struggle to provide adequate visibility in complex environments, making it difficult to accurately monitor, manage, and troubleshoot network performance. Existing models often lack the spatial context necessary for precise device location tracking and dynamic visualization.

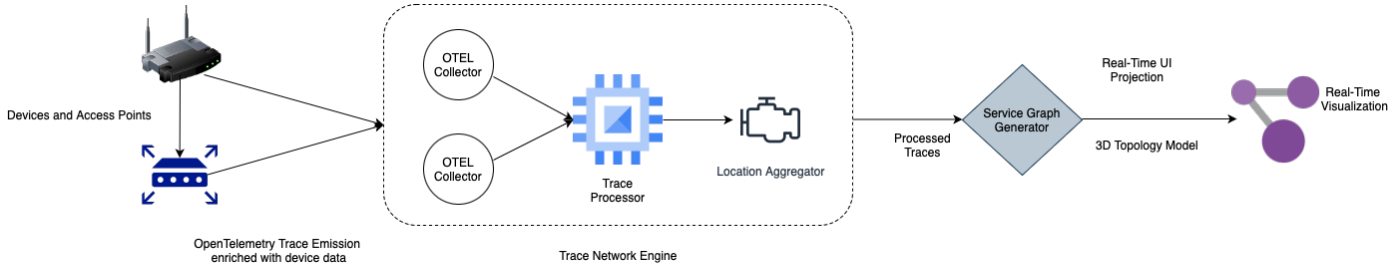
Our solution

This paper addresses these limitations by proposing a real-time 3D network service topology model that integrates OpenTelemetry traces with network device data, enhancing network observability and management. This model allows for faster identification and resolution of issues, reducing costly downtime and minimizing the need for manual troubleshooting efforts. This improved observability can significantly optimize network performance, minimize operational costs, and enhance overall service quality, providing a competitive advantage for businesses.

1. System Architecture

The proposed system architecture is designed to enhance network observability by combining real-time trace data with 3D visualization techniques. It consists of the following key components:

- **Network Devices:** Emit OpenTelemetry traces enriched with device data.
- **Trace Network Engine:** Collects, processes, and enriches traces in real time. It will have a series of open-telemetry collectors. It utilizes device data for calculating the 3D coordinates of devices.
- **Service Graph Generator:** Constructs the 3D network topology model and projects it onto a UI, promoting enhanced network observability.



1.2 Trace Network Engine

The trace network engine plays a critical role in this network observability solution by ingesting and analyzing traces in real time. By aggregating and correlating trace data, the engine provides insights into the operational state of the network, highlighting potential issues such as signal degradation or device disconnection. This engine will comprise of a set of OpenTelemetry collectors, a trace processor, and a location aggregator. The location aggregator enhances
observability by enabling precise tracking of device locations within the network. A triangulation algorithm utilizes device data to calculate the 3D coordinates of devices, which are crucial for understanding spatial relationships within the network.


```
{
 "traceId": "1af3b63c4f3545e4b6782d5a2c1e893f",
 "spanId": "33f5c9e5a5f44b71",
 "parentSpanId": "null",
 "kind": "SPAN_KIND_SERVER",
 "startTimeUnixNano": 1720085185367108600,
 "endTimeUnixNano": 1720085185434272800,
 "name": "OpenTelemetry Trace",
 "timestamp": "2024-08-12T14:48:32.123Z",
 "attributes": {
  "device_id": "device_01",
  "access_point_id": "ap_01",
  "rssi": -45,
  "frequency": 2412,
  "channel": 1,
  "device_mac_address": "00-B0-D0-63-C2-26" }}
```

1.3 Trilateration Algorithm

The trilateration algorithm, based on device data, estimates the position of devices relative to a network reference point (NRP), thus contributing to the network's spatial observability. The rssi data attribute provides information on distance of devices from a common reference point.

Inputs:

1. Positions of NRPs: A set of 3D coordinates $\mathbf{P}_i = (x_i, y_i, z_i)$ for i = 1,2,3,...,n_i where n is the number of NRPs.
2. Distances to NRPs: A corresponding set of distances di from the device to each NRP.

Algorithm Steps:

1. Initialize the Device's Coordinates:
   - Start with an initial guess for the device's coordinates $\mathbf{Q}_0 = (x_0, y_0, z_0)$ typically $\mathbf{Q}_0 = (0, 0, 0)$.
2. Define the Loss Function:
   - For any point $\mathbf{Q} = (x, y, z)$ in space, define the loss function:

$$f(\mathbf{Q}) = \sum_{i=1}^{n} \left(\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} - d_i \right)^2$$

   - This function measures the difference between the actual distances and the estimated distances from the device to the NRPs.
3. **Minimize the Loss Function:**
   - Use an optimization method (e.g., the L-BFGS-B algorithm) to minimize the loss function f(**Q**).
   - The goal is to find the coordinates $\mathbf{Q}^* = (x^*, y^*, z^*)$ that minimizes the sum of squared differences.
4. Output:

The estimated position of the device, $\mathbf{Q}^*$ is the solution to the minimization problem, representing the most likely 3D coordinates of the device based on the given distances.

1.4 Graph Construction Algorithm

The graph construction algorithm maps the network's spatial and logical topology, enriching it with metadata such as device types, connection quality, and signal strength, all of which are vital for network observability.

Inputs:

1. Node Positions: 3D coordinates for each node (e.g., access points and devices).
2. Node Types: The type of each node, such as "AP" for access points and "Device" for connected devices.
3. Edges and Attributes: Connections between nodes with associated trace data, such as latency, packet loss, bandwidth, or other metrics.

Algorithm Steps:

1. Initialize the Graph:
   - Create an empty graph G.
2. Add Nodes to the Graph:
   - For each node i with position $\mathbf{P}_i = (x_i, y_i, z_i)$ and type T_i:
     - Add the node n_i to the graph G with attributes:
       - Position: $\mathbf{P}_i = (x_i, y_i, z_i)$
       - Type: T_i(e.g., "AP" or "Device").
3. Add Edges between Nodes:
   - For each pair of connected nodes (n_i, n_j) with associated trace data:
     - Add an edge between n_i and n_j in the graph G.
     - Store trace attributes as edge attributes, such as:
       - Latency (l_{ij}): Time delay between the nodes.
       - Packet Loss (p_{ij}): Percentage of packets lost during transmission.
       - Bandwidth (b_{ij}): Maximum data transfer rate.
       - Other Metrics: Additional data points like jitter, throughput, etc.
4. Retrieve Node Positions:
   - Extract the positions of all nodes from the graph G for visualization.
5. Visualize the 3D Graph:
   - Initialize a 3D plotting environment using Matplotlib.
   - For each node n_i:
     - Plot the node's position $\mathbf{P}_i$ in 3D space and label the node with its identifier.
   - For each edge (n_i, n_j):
     - Draw a line representing the connection between nodes n_i and n_j.
     - Optionally, colour or style the edge based on trace attributes (e.g., latency colour gradients).
6. Display Trace Information:
   - Annotate or provide visual cues (like color-coded lines) to represent trace attributes such as high latency, significant packet loss, or other critical metrics.
7. Display the 3D Graph:
   - Render the 3D graph using Matplotlib, displaying spatial relationships and connections between access points and devices with trace attributes.

1.5 Real-Time UI (User Interface) Projection Algorithm

The 3D service graph is projected onto a user interface (UI), providing a dynamic and interactive platform for network observability in real-time. By leveraging WebSockets, the UI is capable of rendering network changes as they happen, ensuring that the network administrator has up-to-the-minute insights into the network's state.

Inputs:

1. WebSocket Server: URL of the server providing real-time network data.
2. Data Format:
    - Nodes: Array of node objects with **id**, **position**, and **type** attributes.
    - Edges: Array of edge objects with **id** and **positions** attributes.

Algorithm Steps:

1. Initialize State:
    - Nodes: Create an empty state variable **nodes** to store node data.
    - Edges: Create an empty state variable **edges** to store edge data.
2. Setup WebSocket Connection:
    - Connect: Establish a WebSocket connection to the server using the provided URL.
    - Handle Data: Listen for **network-update** events from the server.
        - Update State: Update **nodes** and **edges** state variables with the new data received.
3. Cleanup:
    - Disconnect: Ensure the WebSocket connection is properly closed when the component is unmounted to prevent memory leaks.
4. Render 3D Graph:
    - Create 3D Scene: Use **@react-three/fiber** to render a 3D scene.
        - Render Nodes: For each node in the **nodes** array:
            - Mesh: Create a mesh using **sphereGeometry** to represent the node.
            - Material: Apply **meshStandardMaterial** with colours based on node type (e.g., "AP" in blue, "Device" in green).
        - Render Edges: For each edge in the **edges** array:
            - Line: Create a line using **bufferGeometry** to represent the connection between nodes.
            - Material: Apply **lineBasicMaterial** to style the edge.
5. Display the 3D Graph:
    - Canvas: Use **Canvas** from **@react-three/fiber** to display the rendered 3D graph.

1.5.1 Summary of Real-Time Aspects

The real-time aspects of this network observability solution are critical for maintaining an up-to-date view of the network where devices often connect or disconnect, or where the signal strength may vary due to environmental factors. Key Considerations for Real-Time Implementation include **OpenTelemetry data streaming, WebSocket integration** and **efficient rendering.**

Evidence the solution works

The proposed solution's effectiveness will be substantiated through empirical data obtained from simulations, real-world experiments, and benchmark analyses. While the theoretical foundations of the approach are sound, efforts are currently underway to develop a proof of concept (POC) to implement the same.

Competitive approaches

This paper presents a novel approach to real-time network monitoring by integrating device data into OpenTelemetry traces and visualizing the real-time 3D network topology.

Current status

The solution has been designed, and preparations are underway to conduct experimental evaluations.

Next steps

We could also integrate machine learning techniques for predicting device movement and optimizing the placement of network devices to enhance coverage and reduce interference.