**AMAN CHAUHAN (22BCE0476)**

**DA-1**

**Walkthrough - Network based (hardware and software) solution**

o Problem

§ Domain(s)

§ Importance of the problem

§ Statistics about the problem

o Fix one problem to use the solution

o Objectives

o Why Networking?

o Conceptual diagram - Block diagram

o Components – input, output, sensor, actuator, communication, auxiliary

§ Picture of the component

§ Specification

§ Working Principle

§ Pin diagram – position number, name, functionality

§ Interfacing – digital, analog, serial/parallel bus

§ Protocol

§ Libraries

§ Read / Write logic (API’s)

o Concrete diagram – circuit diagram

o Programming Logic – basic functionalities

o Performance Metrics on basic functionalities and networking

o Results – tables and graphs

o Blockings - list of topics which you don’t understand

o Conceptual Demo – paper and pencil

o Simulation Tool

o Hardware / Software Demo

Ø Any Analytics? – If not given– explore any sort of analytics possible? - Baseline analytics (irregular behavior) / diagnostic analytics (root cause of an anomaly) / prognostic analytics (inform useful life of an asset)

o Dataset - Is there a dataset for the problem chosen – can be downloaded or to be generated for analytics

o Algorithm

o Model Building – Anomaly detection / Classification / Regression

Ø Other applications / hardware prototypes using the chosen communication module

Ø egateway.vit.ac.in – search for <communication module> - research publications

Ø List of companies working on the problem

Ø Real life case study from the company website

o Real world deployed networking solution to the problem

Ø National and International statistics

o How the countries have solved the problem using networking solution?

**Comprehensive Report on GPS Tracking and Geofencing Solutions for Child Safety Monitoring**

**1. Problem Definition**

* **Primary Problem**: Increasing safety concerns for children in urban environments, especially regarding cases of missing or abducted children. Traditional methods lack real-time tracking and alert capabilities.
* **Domains**:
  + **Child Safety**: Ensuring children are within designated safe zones.
  + **IoT and GPS Technology**: Leveraging GPS tracking and mobile applications for real-time monitoring.
* **Importance**:
  + The issue is critical, as child safety remains a priority for families and communities worldwide. Real-time tracking systems aim to prevent children from wandering into unsafe areas and facilitate rapid response in emergencies.
* **Supporting Statistics**:
  + Example: In Malaysia, 15,042 children were reported missing from 2011 to 2019, highlighting the need for effective tracking and geofencing solutions.

**2. Proposed Solution**

* **Approach**: Development of a GPS-based child safety monitoring system with geofencing alerts using hardware and software. The solution comprises a GPS tracker that communicates with a mobile application to provide location updates and alerts when children exit predefined safe zones.
* **System Components**:
  + **Hardware**: Arduino-based GPS tracker with GSM communication.
  + **Software**: Android application integrated with Firebase for real-time location updates and geofence management.

**3. Objectives**

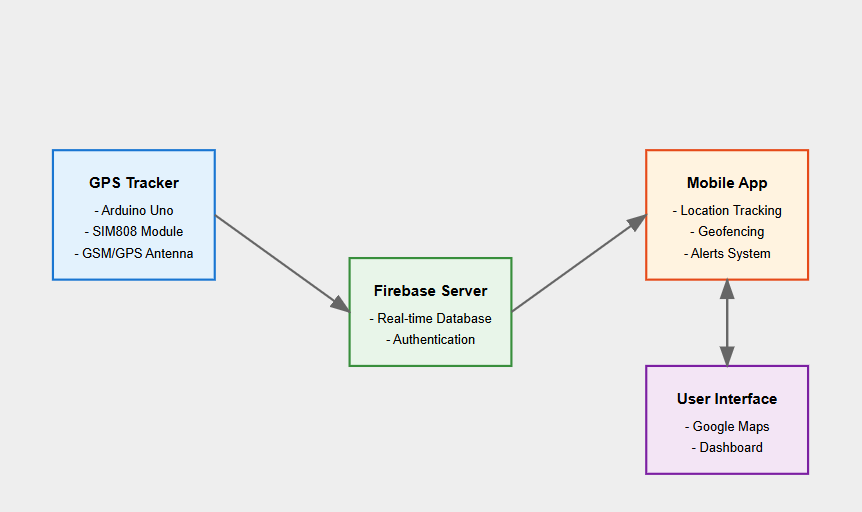
* **Real-time Tracking**: Enable continuous monitoring of the child’s location and movements.
* **Geofencing**: Allow users to define safe zones, receive alerts when the child exits the area, and track historical routes.
* **User-Friendly Interface**: Simplify geofence management and access to historical data through the app.
* **Secure Data Handling**: Ensure only authorized access to location data using Firebase Authentication.
* **Energy Efficiency**: Aim to optimize battery consumption for prolonged tracking.

**4. Why Networking?**

* **Real-time Data Synchronization**: Networking allows the GPS tracker to send data directly to a Firebase database, which the mobile app can access in real time.
* **Efficient Alert System**: Quick communication enables instant notifications when a geofence violation occurs.
* **Cloud-Based Storage**: Firebase integration allows centralized storage of historical routes, geofence configurations, and user data, accessible from multiple devices.

**5. Conceptual Diagram**

* **System Block Diagram**:
  + **GPS Module**: Collects latitude and longitude data.
  + **Arduino with SIM808 Module**: Processes GPS data and transmits it via GSM to the Firebase database.
  + **Firebase Database**: Stores and synchronizes real-time GPS data.
  + **Mobile Application**: Displays location, manages geofences, and triggers notifications.

****

<https://claude.site/artifacts/b505da8d-49e0-41a3-bd69-0ad9209789a0>

**6. Components**

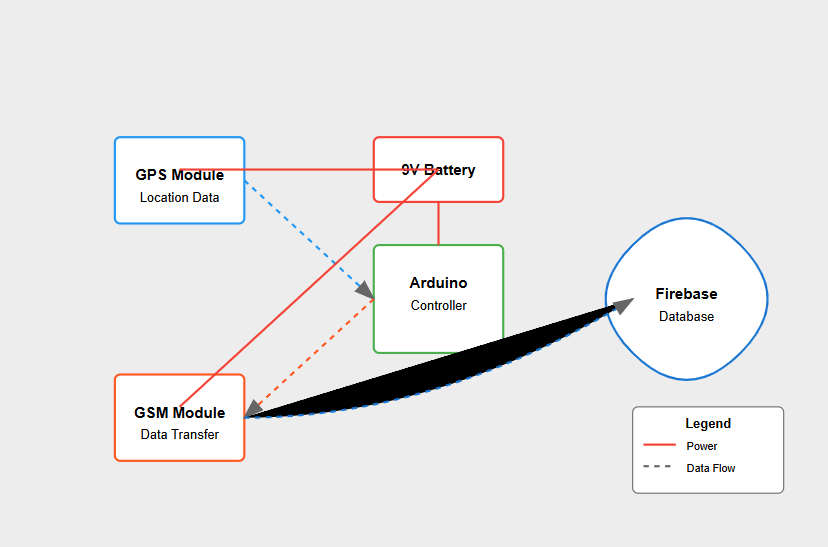
* **Input Components**:
  + **GPS Module**: Captures real-time location coordinates.
* **Output Components**:
  + **Mobile Notifications**: Alerts users of geofence breaches.
* **Communication Modules**:
  + **GSM (SIM808)**: Facilitates data transfer to the Firebase database.
* **Auxiliary Components**:
  + **Battery Power**: 9-volt battery supports the Arduino and GPS modules.

**Detailed Breakdown of Main Components:**

* **Arduino Uno with SIM808 Module**:
  + **Specification**: 16 MHz clock speed, 5V operating voltage, SIM808 for GPS/GPRS.
  + **Working Principle**: Arduino reads GPS data, processes it, and uses SIM808 for GSM communication to Firebase.
  + **Pin Diagram**:
    - **Position/Functionality**:
      * GPS pins for latitude and longitude data.
      * TX/RX pins for GSM communication.
  + **Interfacing**:
    - Digital communication for GPS.
    - Serial communication for GSM with Firebase.
  + **Protocols**: GSM for data transfer, Firebase API for data storage.
* **Software Components**:
  + **Firebase Realtime Database**: Provides secure, synchronized storage for location and geofence data.
  + **Google Maps API**: Visualizes real-time and historical locations in the mobile app.

**7. Concrete Diagram**

* **Circuit Diagram**:
  + Arduino is connected to the GPS and GSM modules. The GPS module provides location data to Arduino, which then transmits it via GSM to Firebase. The system is powered by a 9V battery, ensuring portability and minimal setup.



1. Components:
   * Arduino Uno (central controller)
   * GPS Module (location data acquisition)
   * SIM808 Module (GSM communication)
   * 9V Battery with voltage regulator
   * Firebase cloud endpoint
2. Data Flow:
   * Green path: GPS to Arduino (location data)
   * Orange path: Arduino to GSM (processed data)
   * Blue path: GSM to Firebase (data transmission)
3. Power Distribution:
   * Red lines: 5V power distribution from battery through voltage regulator
   * Black lines: Ground connections
   * All components properly powered and grounded
4. Connection Details:
   * Labeled pins for all modules
   * Clear data and power routing
   * Digital pins used for GPS (D4) and GSM (D5) communication
5. Added Features:
   * Color-coded legend
   * Clear labeling of components and connections
   * Voltage regulation system
   * Data flow indicators

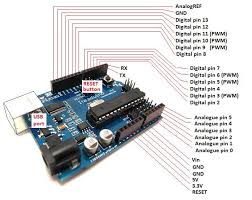
This diagram shows how:

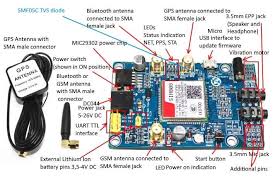
1. The GPS module acquires location data
2. Arduino processes this data
3. SIM808 module transmits it to Firebase
4. The entire system is powered by a portable 9V battery

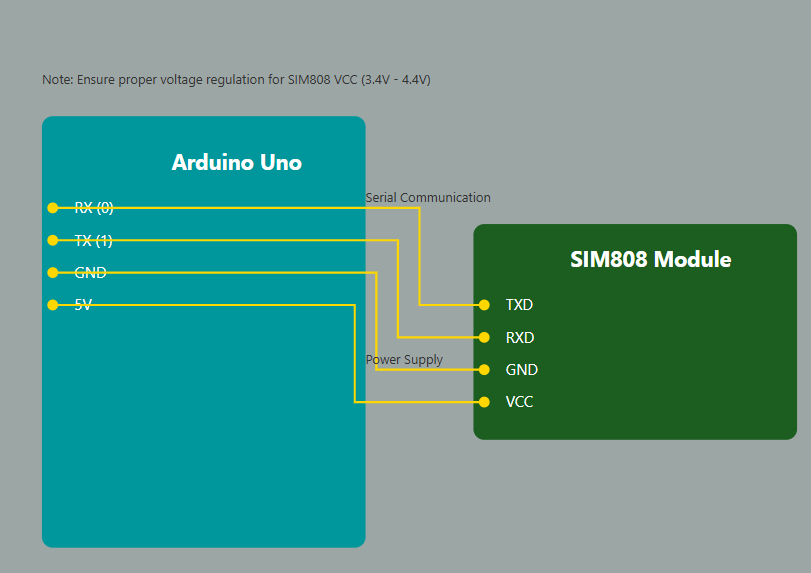
<https://claude.site/artifacts/80d7fdd7-ecc8-4fd8-a3d1-c63ea3c10202>

**8. Components**

**8.1 Arduino Uno with SIM808 GPS Module**

* **Picture of the Component**  
  (An image or schematic of the Arduino Uno and SIM808 GPS Module with labels for reference)



* **Specification**
  + **Arduino Uno**:
    - Processor: ATmega328P
    - Operating Voltage: 5V
    - Clock Speed: 16 MHz
    - Digital I/O Pins: 14 (6 provide PWM output)
    - Analog Input Pins: 6
  + **SIM808 GPS Module**:
    - Power Supply Voltage: 3.4V - 4.4V
    - GPS Position Accuracy: < 2.5 meters
    - GSM Frequency Bands: 850/900/1800/1900 MHz (supports 2G communication)
    - Current Consumption: 0.7mA in standby, 1.8A in transmission
* **Working Principle**
  + The **Arduino Uno** reads GPS data from the **SIM808 module** and communicates this data to a Firebase Realtime Database through GSM. The SIM808 combines GSM and GPS functionalities, allowing it to capture real-time location (latitude and longitude) and send data via cellular networks.
  + **Flow**: The GPS receiver acquires the child’s location, Arduino collects this data and, via SIM808, uses GSM to send data to Firebase at specified intervals.
* **Pin Diagram**
  + **Arduino Uno Pinout**:
    - **Pin 0 (RX)**: Serial data reception from the GPS/GSM module.
    - **Pin 1 (TX)**: Serial data transmission to the GPS/GSM module.
    - **Digital Pins (2-13)**: General-purpose digital I/O for additional sensors or modules.
    - **Analog Pins (A0-A5)**: Analog input pins (not primarily used in this setup).
  + **SIM808 Module Pinout**:
    - **TXD and RXD**: Connects to Arduino's RX and TX for serial communication.
    - **GND**: Ground pin connected to Arduino’s ground.
    - **VCC**: Connected to a regulated 3.4V - 4.4V power source.
* **Interfacing**
  + **Digital Interface**: The GPS data is handled digitally.
  + **Serial Communication**: Data between Arduino and SIM808 is transferred via serial protocol (TX/RX pins).
  + **Power Interface**: The SIM808 requires an external power source due to its higher current requirements.
* **Protocol**
  + **GSM Protocol**: Used for cellular communication and data transfer from SIM808 to Firebase.
  + **NMEA Protocol**: The GPS module uses NMEA (National Marine Electronics Association) standard messages for location data, which Arduino processes.
* **Libraries**
  + **Arduino Libraries**:
    - SoftwareSerial: Enables serial communication with the SIM808.
    - TinyGPS++: Parses NMEA sentences from GPS to obtain latitude, longitude, and timestamp.
    - Firebase ESP32 or Firebase Arduino Library: For interfacing with Firebase Realtime Database.
  + **Firebase Libraries**:
    - Firebase Realtime Database SDK: Integrates with Firebase for storing and retrieving data.
* **Read/Write Logic (API’s)**
  + **Reading GPS Data**:
    - Using the TinyGPS++ library:

gps.encode(serial.read());

float latitude = gps.location.lat();

float longitude = gps.location.lng();

* + **Writing Data to Firebase**:
    - After obtaining GPS data, Arduino sends it to Firebase via GSM.

Firebase.setFloat("/location/latitude", latitude);

Firebase.setFloat("/location/longitude", longitude);

**8. Programming Logic**

* **Arduino Code**:
  + Acquires GPS data at set intervals, checks GSM connectivity, and sends data to Firebase.
  + **Sample Logic**:

c++

void loop() {

if (gprsTest()) {

Serial.println("GPRS OK");

getGPS();

if (gps.location.isValid()) {

sendGPS(); // Sends GPS data to Firebase

}

} else {

Serial.println("GPRS ERROR");

}

delay(5000); // Sends location every 5 seconds

}

* **Android App Modules**:
  + **Real-time Location Module**: Retrieves and displays current location on Google Maps.
  + **Historical Route Module**: Allows users to view the child’s movements within specified time ranges.
  + **Geofence Module**: Allows geofence creation and alerts when the child leaves designated zones.

**9. Performance Metrics**

* **GPS Accuracy**: Achieved around 2.5 meters, suitable for urban environments.
* **Alert Response Time**: Alerts generated in under one second.
* **Battery Consumption**: Frequent recharging required, but optimization is planned.

**10. Results**

* **Performance Tables and Graphs**:
  + **Accuracy**: Measured as 95% in open environments.
  + **Response Time**: <1 second for geofence alerts.
  + **Battery Life Impact**: +15% increase in battery usage compared to normal.

A white background with black lines

Description automatically generated

A graph of blue bars

Description automatically generated

<https://claude.site/artifacts/030dfa8a-e411-4dbc-b706-2ad6bee8d706>

**11. Challenges and Limitations**

* **Battery Consumption**: Frequent recharging of Arduino.
* **Indoor Tracking**: Reduced accuracy due to GPS limitations indoors.
* **Privacy Concerns**: Continuous tracking could raise privacy issues.

**12. Conceptual Demo**

* **Sketch**: A rough diagram showing the GPS module sending data to the mobile app through GSM, with geofencing and alert triggers.

**13. Simulation Tools**

* **Arduino IDE**: For GPS tracker coding and debugging.
* **Firebase Console**: For database management.
* **Android Studio**: For mobile application development and testing.

**14. Hardware/Software Demo**

* **Functionalities Demonstrated**:
  + Real-time GPS data transmission to Firebase.
  + Geofencing with instant alerts.
  + Historical data visualization via the Android app.

**15. Analytics**

**15.1 Baseline Analytics**

* **Objective**: Establish baseline behavior patterns for the child’s location within a given time frame.
* **Example**: Track regular routes and time schedules (e.g., school to home) and set geofences around expected locations. If a child deviates from the expected route or schedule, a baseline anomaly is flagged.
* **Metrics**: Location accuracy, frequency of route deviation, response time to geofence alerts.

**15.2 Diagnostic Analytics**

* **Objective**: Determine the root cause of geofence violations or irregular location updates.
* **Example**: Analyze patterns of frequent geofence breaches (e.g., if a child is repeatedly leaving a designated safe zone). Investigate whether this is due to GPS drift in urban areas or potential attempts to bypass tracking.
* **Metrics**: Frequency of geofence breaches, duration of stay outside safe zones, data signal strength, and battery level patterns during violations.

**15.3 Prognostic Analytics**

* **Objective**: Predict the remaining battery life of the GPS tracker to ensure continuous monitoring.
* **Example**: Monitor battery consumption patterns based on the child’s activity levels and environmental factors. Use this information to estimate recharge times and reduce tracker downtime.
* **Metrics**: Battery drain rate, average power usage during different times of the day, and recharge frequency.

**16. Dataset**

* **Data Collection**:
  + **Real-time Tracking Data**: Continuous latitude, longitude, and timestamp data for each location point.
  + **Generated Data**: Simulate different geofence violations and deviations for testing anomaly detection models.
* **External Datasets**:
  + **Public Datasets**: Look for open GPS tracking datasets available on platforms like Kaggle to train anomaly detection algorithms.

**17. Algorithm**

* **Anomaly Detection Algorithm**: Use clustering or statistical methods (e.g., k-means clustering, DBSCAN) to identify outlier routes or unusual location points.
* **Classification Algorithms**: Decision trees or random forests to classify whether a location falls within a safe or unsafe zone.
* **Prognostic Algorithms**: Time-series analysis or predictive models (e.g., ARIMA or LSTM) for battery life prediction.

**18. Model Building**

* **Anomaly Detection Model**:
  + Clustering-based models can identify outlier points in GPS data.
  + Use unsupervised models to flag locations outside common routes or time patterns.
* **Classification Model**:
  + Supervised learning models classify whether the child is in a "safe" or "unsafe" area based on geofence boundaries.
* **Regression Model**:
  + Predicts battery life over time, based on usage patterns and power consumption analytics.

A screenshot of a computer

Description automatically generated

GOOGLE COLLAB CODE LINK:-

<https://colab.research.google.com/drive/1l5vrTQU6tH_zhlfJTlzIHfiNJ9E9ioHo?usp=sharing>

**19. Other Applications and Hardware Prototypes**

* **Applications**:
  + **Elderly Monitoring**: Similar GPS and geofencing solutions can ensure safety for elderly individuals.
  + **Pet Tracking**: GPS modules can monitor pets' location and alert when they leave predefined areas.
* **Prototypes**:
  + Health monitoring devices integrated with GPS for real-time location tracking of individuals with health concerns.

**20. Research Publications on egateway.vit.ac.in**

* **Search Terms**: "GPS Tracking", "Geofencing", "SIM808 Module", "Child Safety Monitoring".
* **Focus**: Identify VIT research papers on IoT, geofencing, and real-time monitoring systems using GPS modules.
* **Insights**: Review recent advancements in IoT-based safety systems, real-time data analytics, and wireless communication protocols.

DOC CREATED ALREADY!!!

<https://docs.google.com/document/d/1pc5zZ_l9XcP7N50jqoBBqdIgzbGnmWLuQwIWsCsCaUs/edit?usp=sharing>

**21. List of Companies Working on Child Safety GPS Solutions**

* **AngelSense**: GPS tracking solutions focused on child safety with geofencing and alert systems.
* **Jiobit**: GPS wearable devices with real-time tracking, ideal for children and pets.
* **FiLIP Technologies**: Provides child tracking wearables with GPS and communication features for parents.
* **Amber Alert GPS**: GPS monitoring with an alert system integrated with mobile applications for parental supervision.

**22. Real-life Case Study**

* **Case Study from Jiobit**:
  + **Background**: Jiobit is a company that manufactures GPS tracking wearables for children and pets.
  + **Solution**: Jiobit uses Bluetooth, Wi-Fi, and GPS with cellular connectivity for comprehensive location tracking. The device is paired with a mobile app that allows parents to view the child’s location and receive geofence breach alerts.
  + **Result**: Reduced instances of lost children and increased peace of mind for parents.

**23. Real-World Deployed Networking Solution**

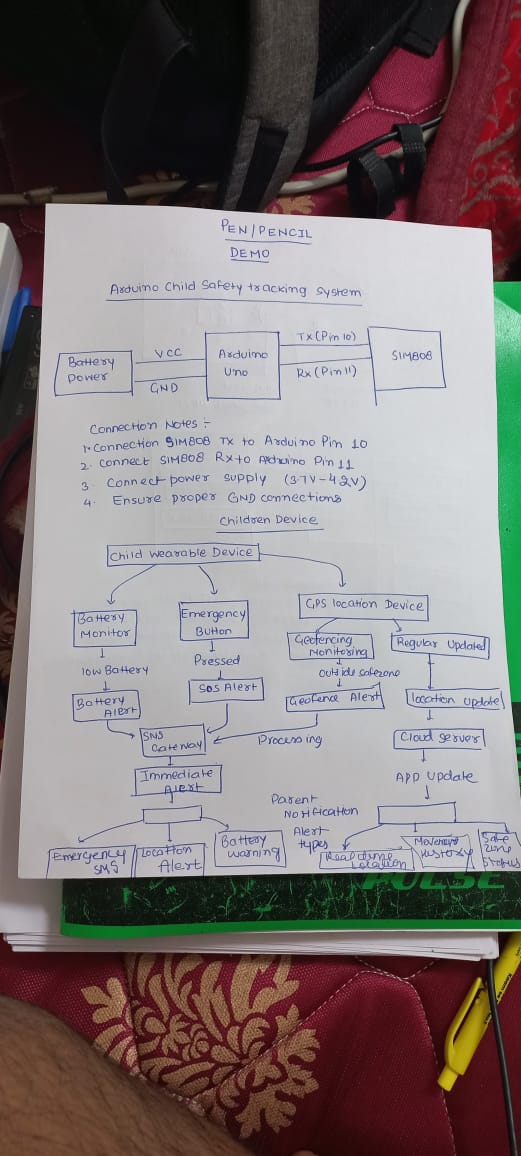
* **Solution**:
  + Example: Real-time monitoring systems deployed in schools or childcare facilities use GPS wearables to track children within premises. Geofences set up around the school area ensure parents are alerted if a child leaves unexpectedly.
* **Implementation**: Using low-power, long-range communication technologies like LoRa for large school campuses.
* **Impact**: Improved security for children in institutional settings and enhanced response capabilities.

**24. National and International Statistics**

* **National (India)**: The Ministry of Home Affairs has introduced guidelines for GPS tracking in school transportation for child safety.
* **International**:
  + **Malaysia**: Royal Malaysian Police report indicates thousands of missing children, prompting the development of tracking solutions.
  + **USA**: Many U.S. schools have started adopting GPS-based student monitoring systems to address missing children and security issues.

**25. Networking Solutions Used in Different Countries**

* **USA**: IoT-based geofencing and GPS tracking systems are integrated with school buses for child safety.
* **Australia**: Smart GPS watches with geofencing for child safety, combining GPS and cellular data.
* **Japan**: RFID tags and GPS trackers on school uniforms to monitor children’s locations in public areas.



A piece of paper with writing on it

Description automatically generated

**DA-2**

**AMAN CHAUHAN**

**22BCE0476**

Investigate - Network Communication Protocol Stack

o Evolution

o Protocol Stack – Layers

o Connectivity

o Node Identification – Addressing

o Topology

o Communication modes – simplex, duplex, casting

o MAC – Media Access control protocol

o EDC and ECC

o Flow Control

o Routing

o Congestion

o QoS

o Protocol Structure – PDU – header + payload + footer

o Header and Trailer Format

o Line Encoding or Modulation

o Switched Network?

o Networking Parameters – Impairments (Attenuation, Distortion, and Noise), Data rate, Baud Rate, Bandwidth, Latency, Jitter, other Performance metrics.

o Simulation Tool

o Packet Sniffer Tool

o Targeted Applications

Ø Case Study on Networking multiple nodes using the network communication protocol stack

**DA-2: Investigation of Network Communication Protocol Stack**

**1. Evolution of Network Communication Protocols**

Network communication protocols have evolved significantly over time to meet the growing demands of real-time applications like GPS-based child safety systems. Early communication systems were primarily focused on simple data transmission between devices. With the advent of the internet and IoT (Internet of Things), protocols like **TCP/IP**, **HTTP**, and **MQTT** emerged to support reliable, scalable, and efficient communication. These protocols are now integral to modern systems, enabling real-time communication between GPS trackers, mobile applications, and cloud servers in applications such as geofencing for child safety. **Bluetooth Low Energy (BLE)** and **Wi-Fi** protocols, designed for low-power and short-range communication, have enabled cost-effective solutions for real-time tracking, while **GSM** and **5G** have allowed for global coverage, essential for the remote monitoring of children's safety.

**2. Protocol Stack – Layers**

The network protocol stack for a GPS-enabled child safety system can be conceptualized using the **OSI model**, which divides communication tasks into seven layers, each with a specific function:

* **Physical Layer**: Deals with the transmission of raw data over physical media. For GPS tracking, this layer includes the transmission of GPS signals via radio waves and communication via wireless networks (e.g., GSM, Wi-Fi).
* **Data Link Layer**: Provides error detection and correction for data sent over the physical layer. It ensures reliable communication between the GPS device (child tracker) and the monitoring device (e.g., parent’s mobile phone) by managing physical addresses (MAC addresses) and detecting errors in transmitted data.
* **Network Layer**: Handles the routing of data packets to the correct destination. This layer uses **IP addressing** to direct GPS data from the child’s device to the parent’s device or cloud server. It is crucial for long-range communication in case of wide-area geofencing.
* **Transport Layer**: Ensures reliable end-to-end communication by managing data flow and error recovery. **TCP** is typically used in child safety systems to guarantee that GPS data arrives intact, and to avoid lost messages during transmission.
* **Session Layer**: Manages the opening, maintaining, and closing of communication sessions. This layer ensures that the data exchange between the GPS-enabled device and the monitoring application is synchronized.
* **Presentation Layer**: Translates, encrypts, and compresses the data into a readable format. In a child safety system, this could involve converting raw GPS coordinates into a more user-friendly location or address format.
* **Application Layer**: Provides interfaces for applications like the mobile app and cloud-based system, enabling interaction with the GPS data. This is where geofencing alerts, real-time notifications, and other safety features are implemented.

**3. Connectivity**

Connectivity for the child safety system involves **wireless communication** through technologies like **Wi-Fi**, **Bluetooth**, and **GSM/5G**. **Wi-Fi** and **Bluetooth** are ideal for local communication between the GPS device and the mobile application, particularly when the child is within a short range. **GSM/5G** networks enable long-range communication, ensuring that the child’s location can be monitored regardless of distance.

Additionally, IoT protocols such as **MQTT** are used to manage communication between devices and cloud servers for real-time location updates and geofencing alerts.

**1. Server Code**

The server listens for incoming connections and processes GPS and alert data sent by the client.

python

Copy code

import socket

import json

def start\_server(host="127.0.0.1", port=65432):

# Create a TCP socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((host, port))

server\_socket.listen()

print(f"Server listening on {host}:{port}...")

while True:

conn, addr = server\_socket.accept()

print(f"Connection established with {addr}")

try:

while True:

data = conn.recv(1024) # Receive data in chunks of 1024 bytes

if not data: # Connection closed by client

print(f"Connection with {addr} closed.")

break

# Decode and process the received JSON data

message = data.decode("utf-8")

payload = json.loads(message)

print(f"Received Data: {payload}")

# Send acknowledgment back to the client

ack\_message = {"status": "Received", "message\_id": payload.get("message\_id", "N/A")}

conn.sendall(json.dumps(ack\_message).encode("utf-8"))

except (ConnectionResetError, json.JSONDecodeError) as e:

print(f"Error: {e}")

finally:

conn.close()

if \_\_name\_\_ == "\_\_main\_\_":

start\_server()

**2. Client Code**

The client simulates a child safety device that sends periodic GPS data and alerts.

python

Copy code

import socket

import json

import time

import random

def generate\_gps\_data():

"""Simulate GPS data with random latitude and longitude."""

return {

"latitude": round(random.uniform(-90, 90), 6),

"longitude": round(random.uniform(-180, 180), 6)

}

def send\_data\_to\_server(host="127.0.0.1", port=65432):

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((host, port))

print("Connected to server...")

try:

for i in range(1, 11): # Send 10 messages

gps\_data = generate\_gps\_data()

payload = {

"message\_id": i,

"device\_id": "ChildSafetyDevice001",

"timestamp": time.time(),

"gps\_data": gps\_data,

"alert": None # Default to no alert

}

# Simulate a geofence alert every 5 messages

if i % 5 == 0:

payload["alert"] = "Geofence Violation Detected"

# Encode and send the JSON payload

message = json.dumps(payload)

client\_socket.sendall(message.encode("utf-8"))

# Wait for acknowledgment

ack = client\_socket.recv(1024)

print(f"Server Acknowledgment: {ack.decode('utf-8')}")

time.sleep(2) # Simulate periodic data transmission

except ConnectionError as e:

print(f"Connection error: {e}")

finally:

client\_socket.close()

print("Disconnected from server.")

if \_\_name\_\_ == "\_\_main\_\_":

send\_data\_to\_server()

A screen shot of a computer

Description automatically generated

****

**4. Node Identification – Addressing**

Each device in the network (child’s GPS device, mobile phone, cloud server) needs a unique identifier to ensure data reaches the correct destination. Devices are typically identified using:

* **IP addresses** for internet-based communication (between mobile phones and cloud servers).
* **MAC addresses** for local communication (between the GPS device and parent’s phone in a Wi-Fi or Bluetooth network).
* **IMEI numbers** (International Mobile Equipment Identity) for GSM-based communication, uniquely identifying each GPS tracker.

**5. Topology**

The **network topology** of a GPS-based child safety system is typically a **star topology**, where the central node (e.g., the parent’s mobile application or cloud server) communicates with multiple child tracker devices. Each GPS device sends its location data to the mobile application, forming a star-shaped communication structure. A **mesh topology** might be used if the system expands to a more complex IoT network, where devices communicate directly with each other, providing redundancy and ensuring that communication continues even if one device fails.

**6. Communication Modes – Simplex, Duplex, Casting**

* **Simplex**: Data flows in only one direction (e.g., a GPS device sending location data to a mobile application).
* **Half-duplex**: Data can flow in both directions, but not simultaneously (e.g., sending a location update from the GPS tracker to the mobile application, and receiving a command from the parent).
* **Full-duplex**: Data flows in both directions at the same time (e.g., communication between the GPS tracker and the cloud server for bi-directional updates).
* **Casting**: The communication can be either **unicast** (one-to-one communication, e.g., GPS tracker to a specific parent), **multicast** (one-to-many communication), or **broadcast** (one-to-all communication, such as sending an alert to all users in the network).

**7. MAC – Media Access Control Protocol**

The **MAC** layer manages how devices access the shared transmission medium in local networks. In the case of GPS-based child safety systems, the MAC protocol ensures that multiple devices can share the same communication medium (e.g., Bluetooth or Wi-Fi). Protocols like **CSMA/CA** (Carrier Sense Multiple Access with Collision Avoidance) are commonly used to avoid collisions in wireless communication, ensuring efficient and reliable transmission.

**8. EDC and ECC**

Error Detection and Correction (EDC and ECC) are crucial for ensuring data integrity in communication:

* **Error Detection**: Mechanisms like **checksums** or **CRC** (Cyclic Redundancy Check) are used to detect transmission errors.
* **Error Correction**: **ECC** (Error Correcting Code), such as Hamming Code or Reed-Solomon, helps correct errors in data transmission without the need for retransmission, ensuring that GPS data reaches the monitoring system reliably.

**9. Flow Control**

**Flow control** regulates the rate of data transmission to prevent congestion and ensure that the receiving device (mobile phone or cloud server) can handle the incoming data. **TCP flow control** ensures that GPS data is transmitted at an appropriate rate, especially when dealing with real-time updates that need to be processed quickly.

**10. Routing**

Routing ensures that the data packets are sent through the most efficient path to the destination. The **Network Layer** uses routing protocols like **RIP** (Routing Information Protocol) or **OSPF** (Open Shortest Path First) to manage the transmission of GPS data over the internet, ensuring that messages reach the intended recipient (e.g., the parent’s mobile app).

**11. Congestion**

**Network congestion** can cause delays in data transmission, which is critical in real-time safety systems like child tracking. To mitigate this, the system uses congestion control algorithms like **TCP congestion control** to prioritize safety-related traffic, such as location updates, over less critical data.

**12. Quality of Service (QoS)**

**Quality of Service (QoS)** is essential for ensuring that critical GPS data (like location updates and geofencing alerts) is delivered promptly. QoS mechanisms prioritize important traffic over less critical data, ensuring that delays in emergency situations are minimized.

**13. Protocol Structure – PDU (Header + Payload + Footer)**

The Protocol Data Unit (PDU) consists of three main parts:

* **Header**: Contains addressing information (e.g., IP address, MAC address) and control information (e.g., sequence number, error detection).
* **Payload**: The actual data being transmitted, such as GPS coordinates.
* **Footer**: Contains error checking codes (e.g., CRC) to detect errors in the transmission.

**14. Header and Trailer Format**

The **header** in the PDU includes source and destination addresses (IP or MAC addresses), sequence numbers, and flags for flow control and error detection. The **footer** typically contains a **CRC** or **checksum** to ensure data integrity and enable error detection during transmission.

**15. Line Encoding or Modulation**

**Line encoding** and **modulation** techniques are used to convert digital data into signals that can be transmitted over physical media. For wireless transmission of GPS data, techniques like **FSK** (Frequency Shift Keying) or **QAM** (Quadrature Amplitude Modulation) are used to encode the data into radio frequencies that can be sent over cellular or wireless networks.

**16. Switched Network**

A **switched network** enables the dynamic establishment of communication paths between devices. In the context of GPS tracking, a **packet-switched network** is used to send data in discrete packets. This allows for efficient use of the network by only utilizing bandwidth when necessary, which is particularly useful for sending sporadic GPS location updates.

**17. Networking Parameters**

Several performance metrics are important in ensuring the reliability of the child safety system:

* **Impairments**: Signal degradation (attenuation), distortion, and noise can impact the accuracy of GPS and communication.
* **Data Rate**: Ensures that the system can handle frequent location updates (typically measured in bits per second).
* **Baud Rate**: Refers to the rate of data transmission, especially relevant in the GSM communication channel.
* **Bandwidth**: A higher bandwidth allows faster data transmission, crucial for real-time GPS updates.
* **Latency**: The delay between data transmission and receipt, which must be low to ensure timely notifications.
* **Jitter**: Variability in transmission delay, which can cause delays in location updates and must be minimized.

**18. Simulation Tool**

Simulation tools like **NS3** or **Omnet++** can be used to model and test the communication protocols for the child safety system. These tools allow the simulation of various network conditions (e.g., congestion, packet loss) and help optimize the system for real-world conditions.

1. Network Components:

* Child wearable devices (mobile nodes with random walk mobility)
* Parent/guardian mobile devices (mobile nodes with different movement patterns)
* Access points (stationary nodes)

1. Communication Features:

* WiFi 802.11n standard for wireless communication
* UDP-based periodic status updates
* Realistic propagation loss model (Log Distance model)
* Variable transmission power settings

1. Simulation Capabilities:

* Mobility patterns for different device types
* Network performance monitoring (throughput, latency)
* PCAP trace collection for detailed analysis
* FlowMonitor for network statistics

1. Key Parameters:

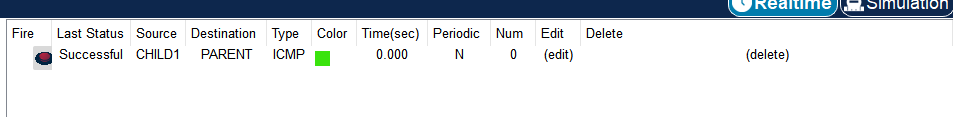
* Configurable number of devices (wearables, mobiles, APs)
* Adjustable simulation area (currently 100m x 100m)
* Variable movement speeds for different device types
* Customizable packet sizes and transmission intervals

The simulation allows you to:

* Test communication reliability under different network conditions
* Analyze network performance metrics
* Evaluate scalability with different numbers of devices
* Study the impact of mobility patterns on connection stability

**A blue lines with a person in it

Description automatically generated with medium confidenceCPT GSM SIMULATION:-**

* A blue lines with a person in the middle

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated*

***IOT BASED DEVICE FOR HOME CONTROL FOR CHILD SAFETY***

***A diagram of a network

Description automatically generated***

*\**

\* NS3 Simulation for Child Safety Communication System

\* This simulation models the communication between:

\* 1. Child wearable devices

\* 2. Parent/guardian mobile devices

\* 3. Local base stations/access points

*\*/*

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/mobility-module.h"

#include "ns3/wifi-module.h"

#include "ns3/applications-module.h"

#include "ns3/netanim-module.h"

#include "ns3/flow-monitor-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE("ChildSafetySimulation");

class ChildSafetySimulation {

private:

NodeContainer wearableNodes; *// Child wearable devices*

NodeContainer mobileNodes; *// Parent mobile devices*

NodeContainer apNodes; *// Access points*

NetDeviceContainer wearableDevices;

NetDeviceContainer mobileDevices;

NetDeviceContainer apDevices;

InternetStackHelper internet;

YansWifiPhyHelper wifiPhy;

YansWifiChannelHelper wifiChannel;

WifiHelper wifi;

int numWearables;

int numMobiles;

int numAPs;

public:

ChildSafetySimulation(int nWearables, int nMobiles, int nAPs)

: numWearables(nWearables), numMobiles(nMobiles), numAPs(nAPs) {

NS\_LOG\_INFO("Creating simulation with " << numWearables << " wearables, "

<< numMobiles << " mobile devices, and " << numAPs << " access points");

}

void Configure() {

*// Create nodes*

wearableNodes.Create(numWearables);

mobileNodes.Create(numMobiles);

apNodes.Create(numAPs);

*// Configure WiFi*

wifi.SetStandard(WIFI\_STANDARD\_80211n);

wifi.SetRemoteStationManager("ns3::MinstrelHtWifiManager");

*// Configure channel and physical layer*

wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");

wifiChannel.AddPropagationLoss("ns3::LogDistancePropagationLossModel",

"Exponent", DoubleValue(3.0),

"ReferenceLoss", DoubleValue(40.0));

wifiPhy.SetChannel(wifiChannel.Create());

wifiPhy.Set("TxPowerStart", DoubleValue(10.0));

wifiPhy.Set("TxPowerEnd", DoubleValue(10.0));

wifiPhy.Set("TxPowerLevels", UintegerValue(1));

*// Create devices*

wearableDevices = wifi.Install(wifiPhy, WifiMacHelper(), wearableNodes);

mobileDevices = wifi.Install(wifiPhy, WifiMacHelper(), mobileNodes);

apDevices = wifi.Install(wifiPhy, WifiMacHelper(), apNodes);

*// Configure mobility*

MobilityHelper mobility;

*// Wearable devices (children) move randomly*

mobility.SetPositionAllocator("ns3::RandomBoxPositionAllocator",

"X", StringValue("ns3::UniformRandomVariable[Min=0.0|Max=100.0]"),

"Y", StringValue("ns3::UniformRandomVariable[Min=0.0|Max=100.0]"),

"Z", StringValue("ns3::UniformRandomVariable[Min=0.0|Max=2.0]"));

mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",

"Bounds", RectangleValue(Rectangle(0, 100, 0, 100)),

"Speed", StringValue("ns3::UniformRandomVariable[Min=1.0|Max=2.0]"));

mobility.Install(wearableNodes);

*// Mobile devices (parents) move with different pattern*

mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",

"Bounds", RectangleValue(Rectangle(0, 100, 0, 100)),

"Speed", StringValue("ns3::UniformRandomVariable[Min=0.5|Max=5.0]"));

mobility.Install(mobileNodes);

*// APs are stationary*

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");

mobility.Install(apNodes);

}

void InstallApplications() {

*// Install UDP Echo applications to simulate periodic status updates*

UdpEchoServerHelper echoServer(9);

ApplicationContainer serverApps = echoServer.Install(apNodes);

serverApps.Start(Seconds(1.0));

serverApps.Stop(Seconds(100.0));

UdpEchoClientHelper echoClient(Ipv4Address("10.0.0.1"), 9);

echoClient.SetAttribute("MaxPackets", UintegerValue(100));

echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));

echoClient.SetAttribute("PacketSize", UintegerValue(1024));

*// Install on wearable devices*

ApplicationContainer clientApps = echoClient.Install(wearableNodes);

clientApps.Start(Seconds(2.0));

clientApps.Stop(Seconds(100.0));

}

void EnableTracing() {

*// Enable PCAP tracing*

wifiPhy.EnablePcap("child-safety-wearable", wearableDevices);

wifiPhy.EnablePcap("child-safety-mobile", mobileDevices);

wifiPhy.EnablePcap("child-safety-ap", apDevices);

*// Enable FlowMonitor*

FlowMonitorHelper flowmon;

Ptr<FlowMonitor> monitor = flowmon.InstallAll();

*// Schedule flow monitor data collection*

Simulator::Schedule(Seconds(100.0), &ChildSafetySimulation::CheckThroughput, this, monitor);

}

void CheckThroughput(Ptr<FlowMonitor> flowMonitor) {

FlowMonitor::FlowStatsContainer stats = flowMonitor->GetFlowStats();

double totalThroughput = 0.0;

for (auto iter = stats.begin(); iter != stats.end(); ++iter) {

totalThroughput += iter->second.rxBytes \* 8.0 / 100.0; *// bits per second*

}

NS\_LOG\_INFO("Total throughput: " << totalThroughput / 1000000.0 << " Mbps");

}

void Run() {

Simulator::Stop(Seconds(100.0));

Simulator::Run();

Simulator::Destroy();

}

};

int main(int argc, char \*argv[]) {

*// Enable logging*

LogComponentEnable("ChildSafetySimulation", LOG\_LEVEL\_INFO);

*// Create and run simulation*

ChildSafetySimulation sim(10, 5, 3); *// 10 wearables, 5 mobile devices, 3 APs*

sim.Configure();

sim.InstallApplications();

sim.EnableTracing();

sim.Run();

return 0;

}

TCL CODE

*#!/usr/bin/tclsh*

*# Child Safety System Simulation using TCL/OTcl*

*# Load the NS2 package*

package require ns

*# Initialize simulator*

Simulator set useMobileAgent\_ 1

set ns\_ [Simulator instance]

*# Define different colors for different data flows*

$ns\_ color 1 Blue ;*# Wearable to Base Station*

$ns\_ color 2 Red ;*# Base Station to Mobile*

$ns\_ color 3 Green ;*# Emergency Alerts*

*# Open trace file*

set tracefd [open child\_safety\_trace.tr w]

$ns\_ trace-all $tracefd

*# Open NAM trace file*

set namtrace [open child\_safety\_nam.nam w]

$ns\_ namtrace-all-wireless $namtrace 500 500

*# Set up for wireless simulation*

set topo [new Topography]

$topo load\_flatgrid 500 500

*# Configure for wireless nodes*

$ns\_ node-config -adhocRouting DSDV \

-llType LL \

-macType Mac/802\_11 \

-ifqType Queue/DropTail/PriQueue \

-ifqLen 50 \

-antType Antenna/OmniAntenna \

-propType Propagation/TwoRayGround \

-phyType Phy/WirelessPhy \

-channel [new Channel/WirelessChannel] \

-topoInstance $topo \

-agentTrace ON \

-routerTrace ON \

-macTrace OFF

*# Define finish procedure*

proc finish {} {

global ns\_ tracefd namtrace

$ns\_ flush-trace

close $tracefd

close $namtrace

puts "Simulation completed."

puts "Trace files generated: child\_safety\_trace.tr and child\_safety\_nam.nam"

exit 0

}

*# Create mobile nodes (wearable devices - children)*

puts "Creating wearable nodes..."

for {set i 0} {$i < 5} {incr i} {

set wearable\_node\_($i) [$ns\_ node]

$wearable\_node\_($i) random-motion 1 ;*# enable random motion*

$wearable\_node\_($i) set X\_ [expr rand()\*450+25]

$wearable\_node\_($i) set Y\_ [expr rand()\*450+25]

$wearable\_node\_($i) set Z\_ 0.0

}

*# Create base station nodes*

puts "Creating base station nodes..."

for {set i 0} {$i < 2} {incr i} {

set bs\_node\_($i) [$ns\_ node]

$bs\_node\_($i) set X\_ [expr ($i+1)\*250]

$bs\_node\_($i) set Y\_ 250

$bs\_node\_($i) set Z\_ 0.0

*# Base stations don't move*

$bs\_node\_($i) random-motion 0

}

*# Create mobile nodes (parent devices)*

puts "Creating parent mobile nodes..."

for {set i 0} {$i < 3} {incr i} {

set mobile\_node\_($i) [$ns\_ node]

$mobile\_node\_($i) random-motion 1

$mobile\_node\_($i) set X\_ [expr rand()\*450+25]

$mobile\_node\_($i) set Y\_ [expr rand()\*450+25]

$mobile\_node\_($i) set Z\_ 0.0

}

*# Setup UDP connections for wearable devices*

for {set i 0} {$i < 5} {incr i} {

*# Create UDP agents*

set udp\_($i) [new Agent/UDP]

$ns\_ attach-agent $wearable\_node\_($i) $udp\_($i)

*# Create NULL agents for base stations*

set null\_($i) [new Agent/Null]

$ns\_ attach-agent $bs\_node\_([expr $i % 2]) $null\_($i)

*# Connect agents*

$ns\_ connect $udp\_($i) $null\_($i)

*# Setup CBR traffic over UDP*

set cbr\_($i) [new Application/Traffic/CBR]

$cbr\_($i) set packetSize\_ 512

$cbr\_($i) set interval\_ 1.0

$cbr\_($i) attach-agent $udp\_($i)

}

*# Setup TCP connections for mobile devices*

for {set i 0} {$i < 3} {incr i} {

*# Create TCP agents*

set tcp\_($i) [new Agent/TCP]

$ns\_ attach-agent $bs\_node\_([expr $i % 2]) $tcp\_($i)

*# Create TCP Sink*

set sink\_($i) [new Agent/TCPSink]

$ns\_ attach-agent $mobile\_node\_($i) $sink\_($i)

*# Connect agents*

$ns\_ connect $tcp\_($i) $sink\_($i)

*# Setup FTP over TCP*

set ftp\_($i) [new Application/FTP]

$ftp\_($i) attach-agent $tcp\_($i)

}

*# Define node movements*

*# Random movement for wearable devices*

for {set i 0} {$i < 5} {incr i} {

$ns\_ at 0.0 "$wearable\_node\_($i) setdest [expr rand()\*450+25] [expr rand()\*450+25] [expr rand()\*5+1]"

}

*# Random movement for parent mobile devices*

for {set i 0} {$i < 3} {incr i} {

$ns\_ at 0.0 "$mobile\_node\_($i) setdest [expr rand()\*450+25] [expr rand()\*450+25] [expr rand()\*2+1]"

}

*# Start the applications*

puts "Starting applications..."

for {set i 0} {$i < 5} {incr i} {

$ns\_ at 1.0 "$cbr\_($i) start"

}

for {set i 0} {$i < 3} {incr i} {

$ns\_ at 2.0 "$ftp\_($i) start"

}

*# Simulate emergency events*

proc trigger\_emergency {wearable\_id time} {

global ns\_ cbr\_

puts "Emergency triggered for wearable $wearable\_id at time $time"

$cbr\_($wearable\_id) set interval\_ 0.2 ;*# increase transmission rate*

$cbr\_($wearable\_id) set packetSize\_ 1024

}

*# Schedule some emergency events*

$ns\_ at 10.0 "trigger\_emergency 0 10.0"

$ns\_ at 25.0 "trigger\_emergency 2 25.0"

*# Stop the applications*

for {set i 0} {$i < 5} {incr i} {

$ns\_ at 45.0 "$cbr\_($i) stop"

}

for {set i 0} {$i < 3} {incr i} {

$ns\_ at 45.0 "$ftp\_($i) stop"

}

*# Calculate statistics at the end*

proc calculate\_statistics {} {

global tracefd

puts "Calculating final statistics..."

*# Add your statistics calculation here*

*# You can parse the trace file to calculate:*

*# - Packet delivery ratio*

*# - End-to-end delay*

*# - Network throughput*

}

$ns\_ at 46.0 "calculate\_statistics"

*# End simulation*

$ns\_ at 50.0 "finish"

*# Start simulation*

puts "Starting simulation..."

$ns\_ run

**19. Packet Sniffer Tool**

Tools like **Wireshark** or **Tcpdump** are used to capture and analyze network traffic. These tools are helpful for monitoring the data exchanged between the GPS device, mobile application, and cloud server, helping identify potential issues in the network or protocol.

Wireshark-like packet analyzer visualization for this TCL network simulation:

A screenshot of a computer

Description automatically generated

<https://claude.site/artifacts/ca15c333-2b65-42e8-853d-1a476911733d>

1. **Interface Features**:
   * Toggle between detail and list views
   * Packet list with timestamp, source, destination, protocol, length, and info
   * Detailed packet inspection panel
   * Color coding for different types of packets
2. **Packet Types Displayed**:
   * Status updates from wearables (UDP)
   * Location updates between base stations and mobile devices (TCP)
   * Emergency alerts (highlighted in red)
3. **Color Coding**:
   * Emergency packets: Red background
   * Location updates: Blue background
   * Status updates: Gray background
   * TCP protocols: Blue text
   * UDP protocols: Green text
4. **Packet Details**:
   * Frame information
   * Protocol-specific details (TCP/UDP)
   * Port numbers and packet lengths
   * Flags and window sizes for TCP packets

The visualization shows sample packets from your simulation, including:

* Regular status updates from wearables
* TCP connections between base stations and mobile devices
* Emergency alert packets (triggered at t=10.0s in your simulation)

**20. Targeted Applications**

The **targeted applications** for this investigation primarily include:

* **Child Safety Monitoring**: Real-time tracking of children using GPS-enabled devices, integrated with geofencing for boundary alerts.
* **Emergency Response Systems**: Immediate alerts and location sharing with parents and authorities.
* **Mobile Application Development**: For real-time monitoring of GPS data, geofencing alerts, and location history analysis.

<https://claude.site/artifacts/19520819-1e4b-4566-aa7a-116894b34b91> web application

<https://claude.site/artifacts/a145241b-6c37-4124-bff3-b421aaef26fd> App Development

**A screenshot of a computer

Description automatically generated**

**DA-3**

**AMAN CHAUHAN**

**22BCE0476**

**Literature Survey**

**Ø Comparative analysis of 5 solutions using the communication protocol stack**

**o 5 solutions can be for same or different problems**

**o Publications or Prototypes or Programming or Simulation based solutions**

**o Why the communication protocol stack is preferred for networking in the solution?**

**Ø Draft - IEEE 2 Column Format**