**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

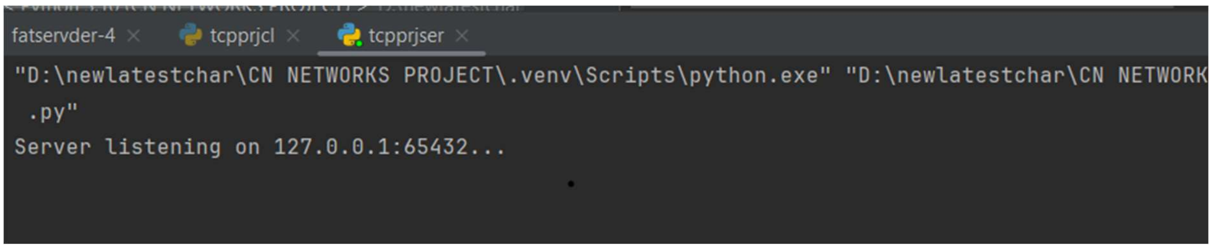
The server listens for incoming connections and processes GPS and alert data sent by the client.

```
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()
```

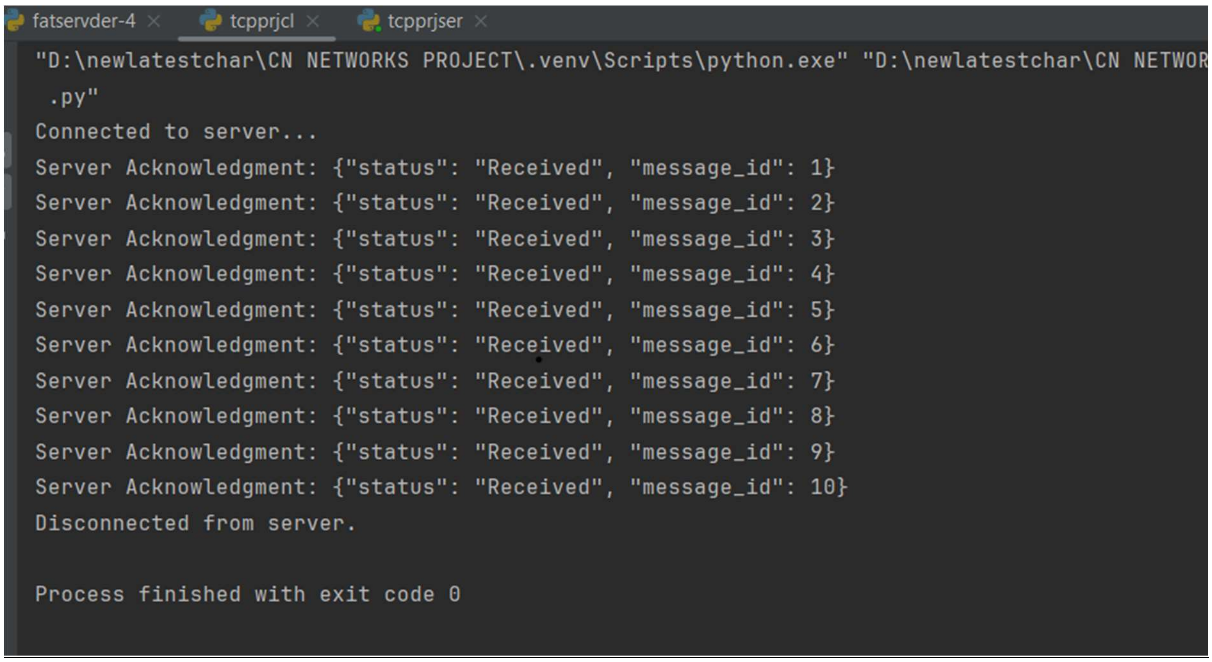
## 2. Client Code

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()
```



```
"D:\newlatestchar\CN NETWORKS PROJECT\.venv\Scripts\python.exe" "D:\newlatestchar\CN NETWORK
 .py"
Server listening on 127.0.0.1:65432...
```



```
"D:\newlatestchar\CN NETWORKS PROJECT\.venv\Scripts\python.exe" "D:\newlatestchar\CN NETWOR
 .py"
Connected to server...
Server Acknowledgment: {"status": "Received", "message_id": 1}
Server Acknowledgment: {"status": "Received", "message_id": 2}
Server Acknowledgment: {"status": "Received", "message_id": 3}
Server Acknowledgment: {"status": "Received", "message_id": 4}
Server Acknowledgment: {"status": "Received", "message_id": 5}
Server Acknowledgment: {"status": "Received", "message_id": 6}
Server Acknowledgment: {"status": "Received", "message_id": 7}
Server Acknowledgment: {"status": "Received", "message_id": 8}
Server Acknowledgment: {"status": "Received", "message_id": 9}
Server Acknowledgment: {"status": "Received", "message_id": 10}
Disconnected from server.

Process finished with exit code 0
```

## 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)

2. 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

3. Simulation Capabilities:

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

4. Key Parameters:

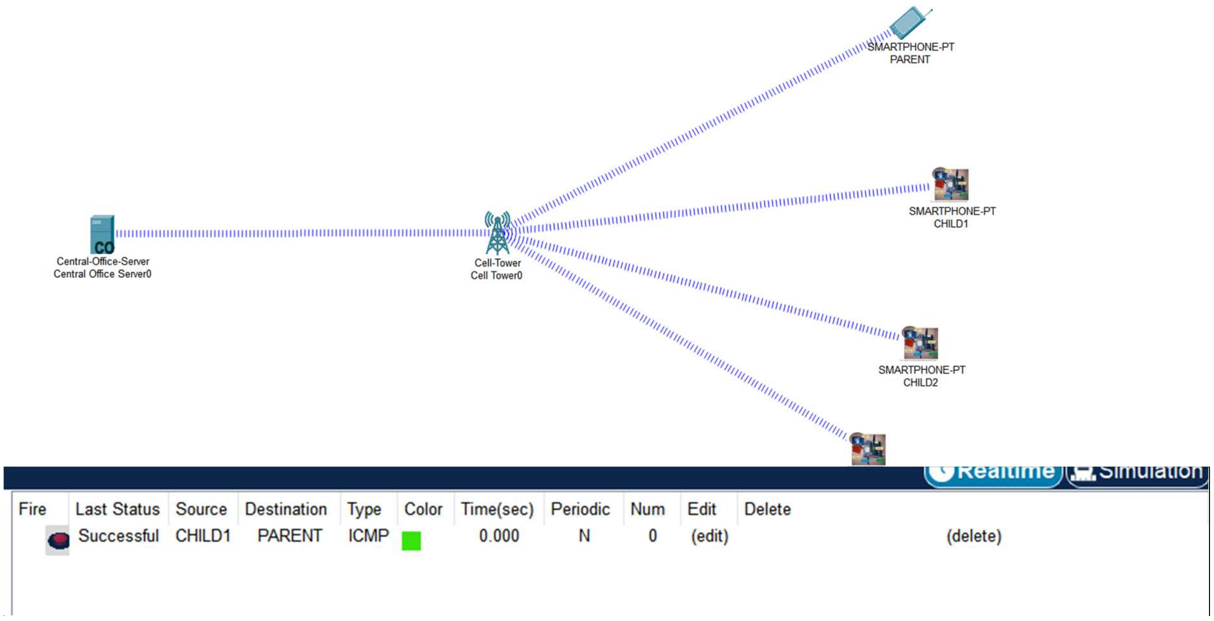
- Configurable number of devices (wearables, mobiles, APs)
- Adjustable simulation area (currently 100m x 100m)
- Variable movement speeds for different device types
- Customizable packet sizes and transmission intervals

The simulation allows you to:

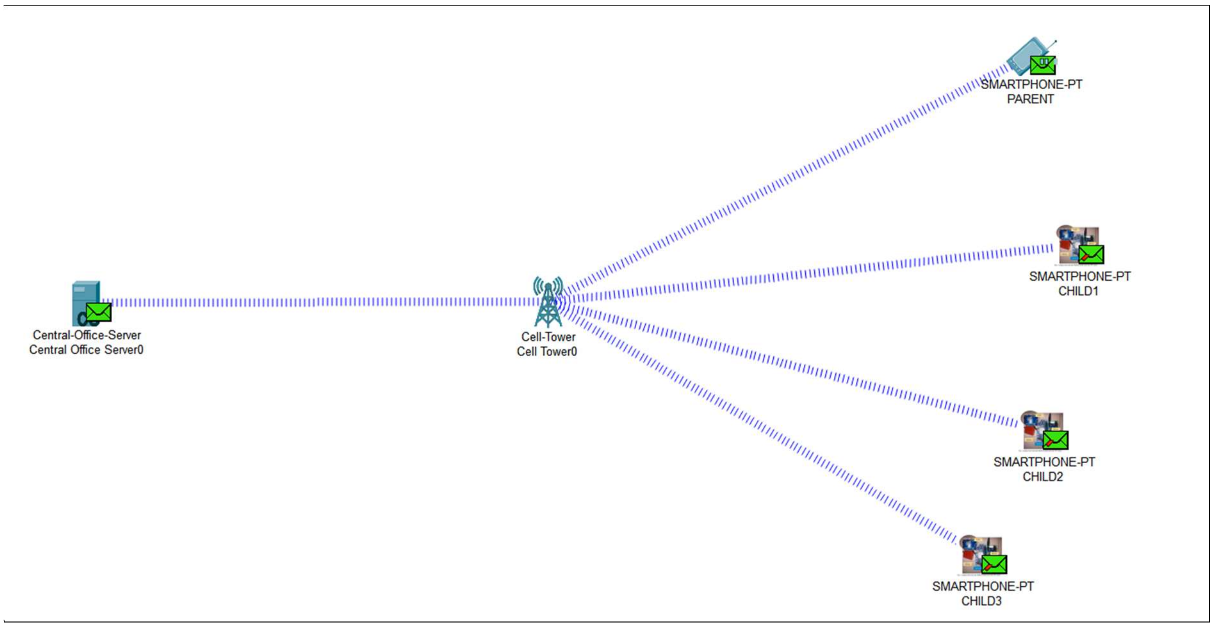
- Test communication reliability under different network conditions
- Analyze network performance metrics
- Evaluate scalability with different numbers of devices

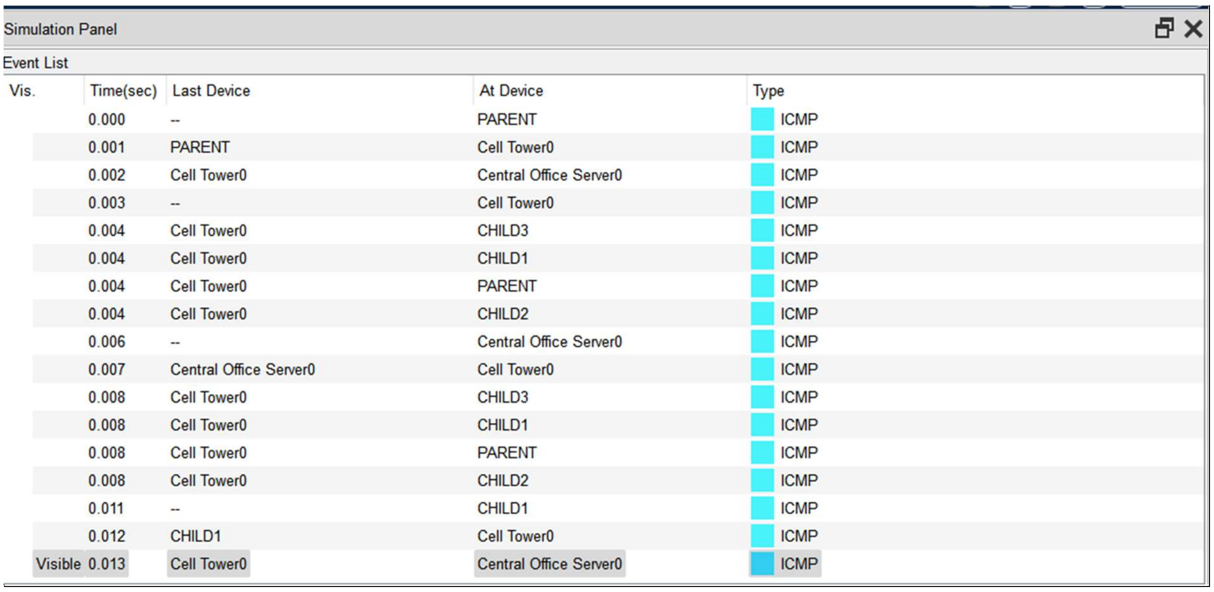
- Study the impact of mobility patterns on connection stability

**CPT GSM SIMULATION:-**



| Fire | Last Status | Source | Destination | Type | Color | Time(sec) | Periodic | Num | Edit | Delete |
|---|---|---|---|---|---|---|---|---|---|---|
| ● | Successful | CHILD1 | PARENT | ICMP | 🟩 | 0.000 | N | 0 | (edit) | (delete) |



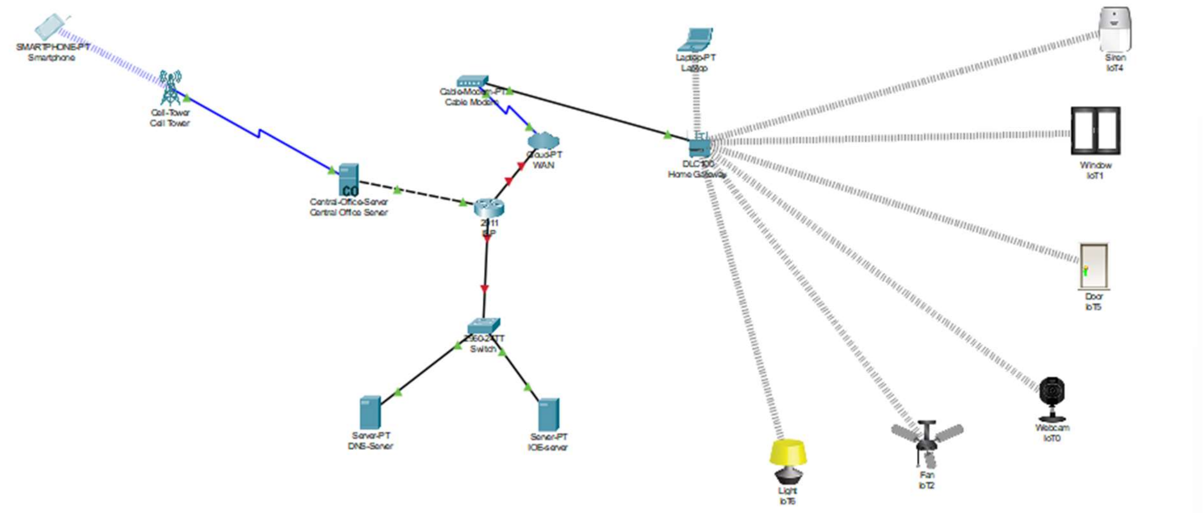


Simulation Panel

Event List

| Vis. | Time(sec) | Last Device | At Device | Type |
|---|---|---|---|---|
| | 0.000 | -- | PARENT | 🟦 ICMP |
| | 0.001 | PARENT | Cell Tower0 | 🟦 ICMP |
| | 0.002 | Cell Tower0 | Central Office Server0 | 🟦 ICMP |
| | 0.003 | -- | Cell Tower0 | 🟦 ICMP |
| | 0.004 | Cell Tower0 | CHILD3 | 🟦 ICMP |
| | 0.004 | Cell Tower0 | CHILD1 | 🟦 ICMP |
| | 0.004 | Cell Tower0 | PARENT | 🟦 ICMP |
| | 0.004 | Cell Tower0 | CHILD2 | 🟦 ICMP |
| | 0.006 | -- | Central Office Server0 | 🟦 ICMP |
| | 0.007 | Central Office Server0 | Cell Tower0 | 🟦 ICMP |
| | 0.008 | Cell Tower0 | CHILD3 | 🟦 ICMP |
| | 0.008 | Cell Tower0 | CHILD1 | 🟦 ICMP |
| | 0.008 | Cell Tower0 | PARENT | 🟦 ICMP |
| | 0.008 | Cell Tower0 | CHILD2 | 🟦 ICMP |
| | 0.011 | -- | CHILD1 | 🟦 ICMP |
| | 0.012 | CHILD1 | Cell Tower0 | 🟦 ICMP |
| Visible | 0.013 | Cell Tower0 | Central Office Server0 | 🟦 ICMP |

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



```
/*
 * 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
```

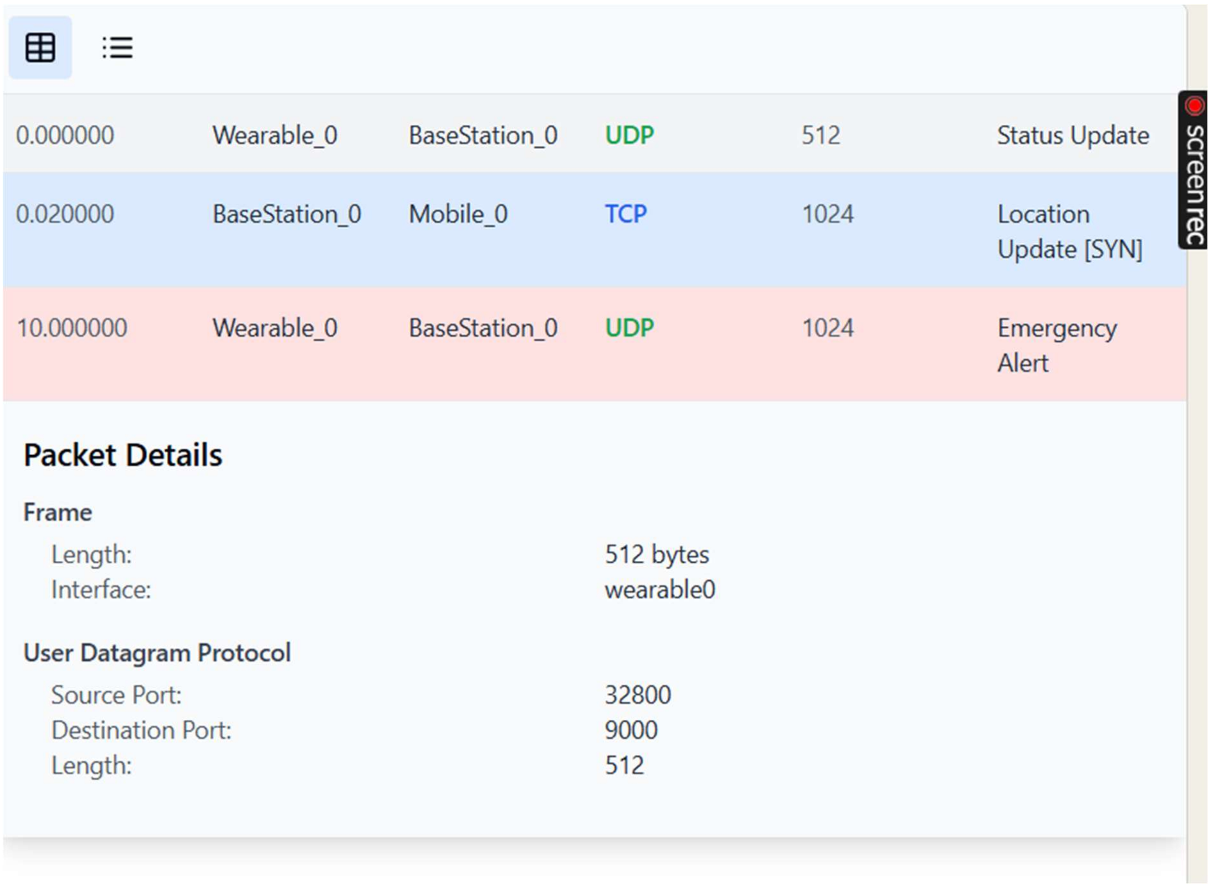
```
# Open NAM trace file
set namtrace [open child_safety_nam.nam w]
$ns_ namtrace-all-wireless $namtrace 500 500

# 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:



| | | | | | |
|---|---|---|---|---|---|
| 0.000000 | Wearable_0 | BaseStation_0 | UDP | 512 | Status Update |
| 0.020000 | BaseStation_0 | Mobile_0 | TCP | 1024 | Location Update [SYN] |
| 10.000000 | Wearable_0 | BaseStation_0 | UDP | 1024 | Emergency Alert |

**Packet Details**

**Frame**

| | |
|---|---|
| Length: | 512 bytes |
| Interface: | wearable0 |

**User Datagram Protocol**

| | |
|---|---|
| Source Port: | 32800 |
| Destination Port: | 9000 |
| Length: | 512 |

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)

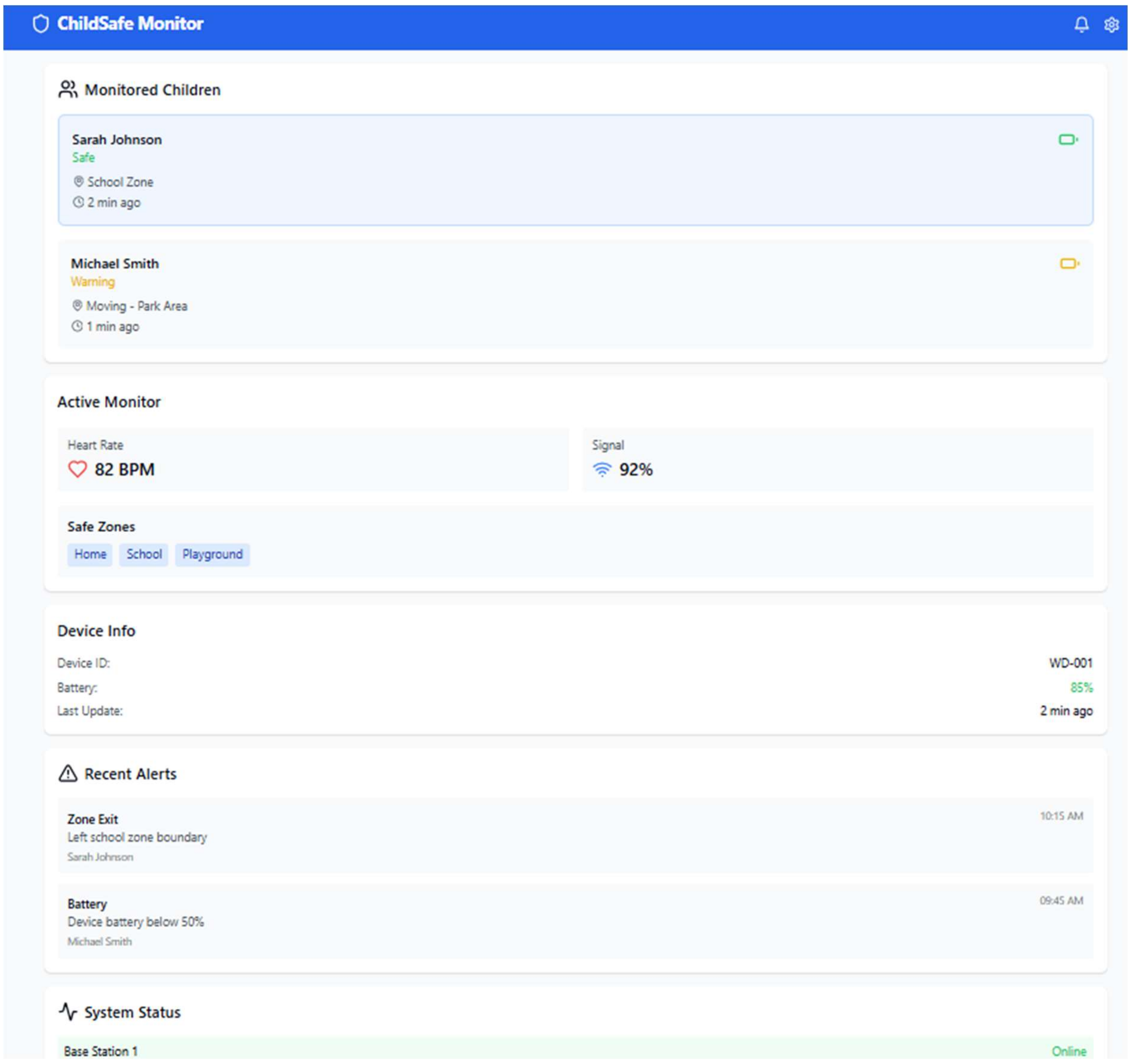
## 20. Targeted Applications

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



ChildSafe Monitor

Monitored Children

Sarah Johnson
Safe
School Zone
2 min ago

Michael Smith
Warning
Moving - Park Area
1 min ago

Active Monitor

| | |
|---|---|
| Heart Rate | Signal |
| 82 BPM | 92% |

Safe Zones
Home School Playground

Device Info

| | |
|---|---|
| Device ID: | WD-001 |
| Battery: | 85% |
| Last Update: | 2 min ago |

Recent Alerts

Zone Exit — 10:15 AM
Left school zone boundary
Sarah Johnson

Battery — 09:45 AM
Device battery below 50%
Michael Smith

System Status

Base Station 1 — Online