Mini Project Report

Real-Time Speed Monitoring System

Submitted by:

Sarthak S Kumar PES2UG21CS482
Sathish Kumar G PES2UG21CS484

7th Semester H Section

Prof. Animesh Giri
Associate Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING

1. Introduction

- **Overview:**

  Briefly describe the project, its objectives, and significance.

Project Description:

The Real-Time Speed Monitoring System is designed to monitor the speed of vehicles in a simulated environment using the Cooja simulator and Wireshark. The project aims to emulate real-world scenarios where sensor nodes deployed along a road can detect and report vehicle speeds to a central monitoring system. The system can be used to ensure compliance with speed limits, enhance traffic management, and improve road safety.

Objectives:

Simulation of Sensor Nodes: Use the Cooja simulator to create a network of sensor nodes placed along a virtual roadway. These nodes will be programmed to detect and measure the speed of passing vehicles.

Data Collection and Transmission: Implement protocols for sensor nodes to collect speed data and transmit it to a central base station or monitoring node.

Real-Time Data Analysis: Utilize Wireshark to capture and analyze the transmitted data in real-time. Develop methods to process and visualize the speed data, identifying instances of speeding.

Alert Generation: Set up the system to generate alerts when vehicles exceed predefined speed limits, simulating real-time notifications to traffic authorities.

Significance:

Enhanced Traffic Management: By monitoring vehicle speeds in real-time, traffic authorities can make informed decisions to manage traffic flow and reduce congestion.

Improved Road Safety: The system can help identify and mitigate speeding incidents, contributing to the reduction of road accidents and enhancing overall safety.

Scalable Solution: The project serves as a prototype that can be scaled to monitor larger road networks, making it a versatile solution for smart city applications.

Educational Value: This project provides practical experience in working with Contiki OS, Cooja simulator, and Wireshark, reinforcing theoretical knowledge with hands-on application.

Project Implementation Steps:

Setup Cooja Simulator: Configure the Cooja simulator environment with multiple sensor nodes along a virtual road.

Node Programming: Write Contiki OS code for the sensor nodes to detect and measure vehicle speeds.

Data Transmission Protocol: Implement a communication protocol for nodes to send speed data to the central node.

Wireshark Integration: Use Wireshark to capture the transmitted data, analyze it in real-time, and develop a method to display and process the data.

2. Background

- ## Contiki OS:

  ## Introduction to Contiki OS and its features.

  Contiki OS is an open-source operating system for the Internet of Things (IoT) and embedded systems. It is designed to be lightweight and efficient, suitable for devices with limited memory and processing power.

  Features:

  - Lightweight: Runs on minimal resources (e.g., 2 KB RAM, 40 KB ROM).
  - Event-Driven Kernel: Saves power by remaining idle when no events are processed.
  - Protothreads: Enables multitasking with minimal memory usage.
  - Network Protocols: Supports IPv6, 6LoWPAN, RPL, and CoAP.
  - Simulation and Emulation: Allows testing and development before hardware deployment.
  - Modularity: Customizable to include only necessary components.
  - Wireshark Integration: Captures and analyzes packet data in real-time.
  - Debugging and Visualization: Offers tools to understand and troubleshoot network behavior.
  - Scalability: Simulates large networks efficiently.

-

- ## Cooja Simulator:
  ## Overview of the Cooja simulator and its relevance to the project.

Cooja is a network simulator for Contiki OS, enabling the simulation of large-scale IoT networks, including wireless sensor networks.

Relevance to the Project:

- Realistic Simulation: Mimics real-world scenarios with various hardware platforms and network topologies.
- Sensor Node Emulation: Provides realistic testing conditions.
- Network Analysis: Supports detailed protocol and behavior analysis.

3. Project Description

- ## Problem Statement:

  ## Define the problem your project addresses.

Traffic management and road safety are critical issues in urban areas. Speeding vehicles pose significant risks to pedestrians and other drivers, leading to accidents and fatalities. There is a need for an efficient and real-time system to monitor vehicle speeds, identify speeding incidents, and alert authorities for timely intervention. This project aims to address this need by developing a Real-Time Speed Monitoring System using the Cooja simulator and Wireshark.

Objectives

- Simulate Sensor Nodes: Use the Cooja simulator to create a network of sensor nodes positioned along a virtual roadway to detect vehicle speeds.
- Collect and Transmit Data: Implement protocols for sensor nodes to collect speed data and transmit it to a central monitoring node.
- Real-Time Data Analysis: Use Wireshark to capture and analyze the speed data in real-time, processing it to identify speeding vehicles.
- Enhance Traffic Management: Provide insights for better traffic management and improved road safety by utilizing the speed monitoring data.

4. Implementation

- ## Development Environment:
  ## Detail the development setup.

Contiki OS: Version 3.0

Cooja Simulator

Programming Language: C

Tools: Wireshark for network analysis

Mobility in Cooja

Simulation:

Models Used: Cooja supports mobility models such as Random Waypoint and Random Walk for emulating mobile nodes.

Configuration: Nodes can be configured with movement patterns and speeds to simulate realistic mobility scenarios.

Wireshark Integration

Network Monitoring:

Packet Capture: Wireshark captures packets transmitted between nodes in the simulated network.

Analysis: Filters and dissectors in Wireshark are used to analyze network protocols and behaviors in real-time.

- **Modules and Functions:**

  Summarize the key modules and functions with code snippets.

```
#include "contiki.h"
#include "net/ip/uip.h"
#include "net/ipv6/uip-ds6.h"
#include "net/ip/uip-udp-packet.h"
#include "net/rpl/rpl.h"
#include "dev/serial-line.h"
#if CONTIKI_TARGET_Z1
#include "dev/uart0.h"
#else
#include "dev/uart1.h"
#endif
#include "collect-common.h"
#include "collect-view.h"

#include <stdio.h>
#include <string.h>
#include <stdlib.h> // For random number generation

#define UDP_CLIENT_PORT 8775
```

```
#define UDP_SERVER_PORT 5688

#define DEBUG DEBUG_PRINT
#include "net/ip/uip-debug.h"

static struct uip_udp_conn *client_conn;
static uip_ipaddr_t server_ipaddr;

/---------------------------------------------------------------------------/
PROCESS(udp_client_process, "UDP client process");
AUTOSTART_PROCESSES(&udp_client_process, &collect_common_process);
/---------------------------------------------------------------------------/
void
collect_common_set_sink(void)
{
  /* A udp client can never become sink */
}
/---------------------------------------------------------------------------/

void
collect_common_net_print(void)
{
```

```
  rpl_dag_t *dag;
  uip_ds6_route_t *r;

  /* Let's suppose we have only one instance */
  dag = rpl_get_any_dag();
  if(dag->preferred_parent != NULL) {
    PRINTF("Preferred parent: ");
    PRINT6ADDR(rpl_get_parent_ipaddr(dag->preferred_parent));
    PRINTF("\n");
  }
  for(r = uip_ds6_route_head();
      r != NULL;
      r = uip_ds6_route_next(r)) {
    PRINT6ADDR(&r->ipaddr);
  }
  PRINTF("---\n");
}
/---------------------------------------------------------------------------/
static void
tcpip_handler(void)
{
  if(uip_newdata()) {
```

```
    /* Ignore incoming data */
  }
}
/---------------------------------------------------------------------------/
void
collect_common_send(void)
{
  static uint8_t seqno;
  struct {
    uint8_t seqno;
    uint8_t for_alignment;
    char vehicle_name[10];
    char registration_number[15];
    int speed[20];
  } msg;
  uint16_t parent_etx;
  uint16_t rtmetric;
  uint16_t num_neighbors;
  uint16_t beacon_interval;
  rpl_parent_t *preferred_parent;
  linkaddr_t parent;
  rpl_dag_t *dag;
```

```
if(client_conn == NULL) {
  /* Not setup yet */
  return;
}
memset(&msg, 0, sizeof(msg));
seqno++;
if(seqno == 0) {
  /* Wrap to 128 to identify restarts */
  seqno = 128;
}
msg.seqno = seqno;

int speed_current = rand() % 5;

// Add random vehicle details
snprintf(msg.vehicle_name, sizeof(msg.vehicle_name), "Veh-%d", rand() % 100);
snprintf(msg.registration_number, sizeof(msg.registration_number), "SPE%d", rand() %
100);
snprintf(msg.speed, sizeof(msg.speed), "ID:%d", (speed_current < 0 ? 0 : speed_current));

linkaddr_copy(&parent, &linkaddr_null);
parent_etx = 0;
```

```
/* Let's suppose we have only one instance */
dag = rpl_get_any_dag();
if(dag != NULL) {
  preferred_parent = dag->preferred_parent;
  if(preferred_parent != NULL) {
    uip_ds6_nbr_t *nbr;
    nbr = uip_ds6_nbr_lookup(rpl_get_parent_ipaddr(preferred_parent));
    if(nbr != NULL) {
      /* Use parts of the IPv6 address as the parent address, in reversed byte order. */
      parent.u8[LINKADDR_SIZE - 1] = nbr->ipaddr.u8[sizeof(uip_ipaddr_t) - 2];
      parent.u8[LINKADDR_SIZE - 2] = nbr->ipaddr.u8[sizeof(uip_ipaddr_t) - 1];
      parent_etx = rpl_get_parent_rank((uip_lladdr_t *) uip_ds6_nbr_get_ll(nbr)) / 2;
    }
  }
  rtmetric = dag->rank;
  beacon_interval = (uint16_t) ((2L << dag->instance->dio_intcurrent) / 1000);
  num_neighbors = uip_ds6_nbr_num();
} else {
  rtmetric = 0;
  beacon_interval = 0;
  num_neighbors = 0;
```

```
  }

  uip_udp_packet_sendto(client_conn, &msg, sizeof(msg),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
}
/---------------------------------------------------------------------------/
void
collect_common_net_init(void)
{
#if CONTIKI_TARGET_Z1
  uart0_set_input(serial_line_input_byte);
#else
  uart1_set_input(serial_line_input_byte);
#endif
  serial_line_init();
}
/---------------------------------------------------------------------------/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;
```

```
  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
      /* hack to make address "final" */
      if (state == ADDR_TENTATIVE) {
        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
      }
    }
  }
}
/------------------------------------------------------------------------/
static void
set_global_address(void)
{
  uip_ipaddr_t ipaddr;

  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
```

```
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);

  /* set server address */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 1);

}
/---------------------------------------------------------------------------/
PROCESS_THREAD(udp_client_process, ev, data)
{
  static struct etimer periodic_timer;

  PROCESS_BEGIN();

  PROCESS_PAUSE();

  set_global_address();

  PRINTF("UDP client process started\n");

  print_local_addresses();
```

```
/* new connection with remote host */
client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

PRINTF("Created a connection with the server ");
PRINT6ADDR(&client_conn->ripaddr);
PRINTF(" local/remote port %u/%u\n",
     UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));

etimer_set(&periodic_timer, CLOCK_SECOND);

while(1) {
  PROCESS_YIELD();
  if(ev == tcpip_event) {
    tcpip_handler();
  }
  if(etimer_expired(&periodic_timer)) {
    collect_common_send();
    etimer_reset(&periodic_timer);
  }
}
```

```
  PROCESS_END();
}
/---------------------------------------------------------------------------/
```

- **Challenges and Solutions:**
  Discuss major challenges faced and how they were resolved.

Challenges:

Node Synchronization: Ensuring synchronized data transmission from multiple sensor nodes.

Realistic Mobility: Emulating realistic vehicle movements in Cooja.

Protocol Analysis: Understanding and interpreting network protocols in Wireshark.

Solutions:

Node Coordination: Implemented a time-slotted protocol for synchronized data transmission.

Mobility Modeling: Adjusted movement parameters to simulate varying vehicle speeds and routes.

Protocol Filters: Used Wireshark filters and protocol dissectors to isolate and analyze relevant packet data.

5. Results and Discussion

Results

Summary of Project Outcomes:

1. **Simulation of Sensor Nodes**: Successfully simulated sensor nodes using Cooja simulator, positioned along a virtual roadway.
2. **Data Collection and Transmission**: Implemented protocols for sensor nodes to collect vehicle speed data and transmit it to a central monitoring node.
3. **Real-Time Data Analysis**: Utilized Wireshark for real-time capture and analysis of speed data packets transmitted within the simulated network.
4. **Alert Generation**: Developed a system to generate alerts when vehicles exceeded predefined speed limits, simulating real-time notifications.
5. **Visualization**: Generated visualizations of speed data trends and alerts using graphs and charts.

Discussion

Significance of Results and Insights Gained:

1. **Enhanced Traffic Management**: The project demonstrated the feasibility of using IoT and real-time data analysis to monitor vehicle speeds, which can contribute to better traffic flow and reduced congestion.
2. **Improved Road Safety**: By promptly identifying speeding vehicles, the system can assist in preventing accidents and improving overall road safety.

3. **Scalability**: The modular design and scalability of the system make it adaptable for monitoring larger road networks and smart city applications.
4. **Educational Value**: Participants gained practical experience in Contiki OS, Cooja simulator, and Wireshark, reinforcing their understanding of IoT system development and network monitoring.
5. **Challenges Overcome**: Addressed challenges such as node synchronization, realistic mobility modeling, and protocol analysis, showcasing problem-solving skills and technical proficiency.

6. Conclusion

- **Summary:**
  Summarize key findings and achievements.
- **Future Work:**
  Suggest potential future improvements or extensions.