COMPUTER SCIENCES AND SOFTWARE ENGINEERING
AUBURN UNVERSITY

COMP 5360/6360
Wireless and Mobile Networks
Spring 2017

Project 2

Parallel Flow Connection for Supporting Reliable and Mobile Communication Hub for First Responders

Due: 11:55pm April 3, 2017

1 Objective

The goal of this project is to design and implement Parallel Flow Connections (PFC) for supporting a reliable and mobile communication hub for first responder body sensor networks. PFC will enhance reliability of communication connections using multiple parallel links in order for the body sensor network to reliably transmit critical sensor information from the first responders to the Fire Chiefs, despite failure of some of the communication links. This project will use a WiFi emulation environment that allows transmission of streams of sensor data over the network through the socket interface. In the real environment, the two devices will be connected through different types of wireless networks, including a WiFi link in ad-hoc (IBSS) mode. In this emulation environment, you may emulate the underlying wireless link layer with exchange of UDP datagrams between two neighboring hosts through AF_INET sockets. However, the protocol you will implement can be ported to an actual network layer. In this first project, sensor data packets may be transmitted between two computers that are connected to the Internet using the TCP/IP protocol suite. However, in subsequent projects, the sensor data packets may be transmitted over wireless mobile and ad-hoc networks (MANET) using ad-hoc routing protocols or an emulation of the MANET over a fixed network.

2 Background

By integrating both the sensor hub and communication hub, it would provide the following important capabilities: (i) track streams of real-time sensor data of First Responders, such as location, status, and health, and (ii) reliable transmission of critical information to selected personnel, such as fire chiefs, using parallel communication links. In hazardous environments, such as large buildings on fire, earthquakes, hurricane, or other wide-spread disasters, reliance on only one type of communication link is not advisable. For example, cell phone towers or WiFi access point infrastructures will likely go down or are non-existent. In the absence of these infrastructures, such as cell towers or access points, many wireless devices and sophisticated smartphones will be rendered useless. Instead, we need to provide multiple parallel communication links, including virtual and dynamic networks that can be spontaneously established in areas where no access points or cell tower coverage exists. With parallel links, first responders even in remote mountain areas fighting forest fires will then be able to communicate with each other.

3 Requirements

The objective of this second project is to implement Parallel Flow Connections (PFC) that will support a reliable and mobile communication hub for first responder body sensor networks to transmit the sensor data streams to the Fire Chief. You must enable at least four sensors on the First Responder body sensor network that continuously stream time-varying sensor information for at least five minutes. You must implement at least four of the following sensors: (i) Heart rate sensor, (ii) location sensor, (iii) air-pack oxygen level, and (iv) toxic gas sensor.

Write the sensor hub program that will transmit these four *independent* streams of sensor data over a reliable connection with parallel flows that you will implement. This reliable connection with parallel flows will connect the First Responder to the Fire Chief to enhance communication reliability. For this project, you will implement two parallel flows using Ethernet and Mobile and Ad hoc Network (MANET) using flooding protocol. Figure 1 shows the overview of the sensor hub, communication hub and the reliable connection with parallel flows. When the Ethernet link fails, the transmission will automatically switch to MANET with flooding. However, when the Ethernet link is restored and the MANET route fails, then the transmission will automatically switch to the Ethernet link.

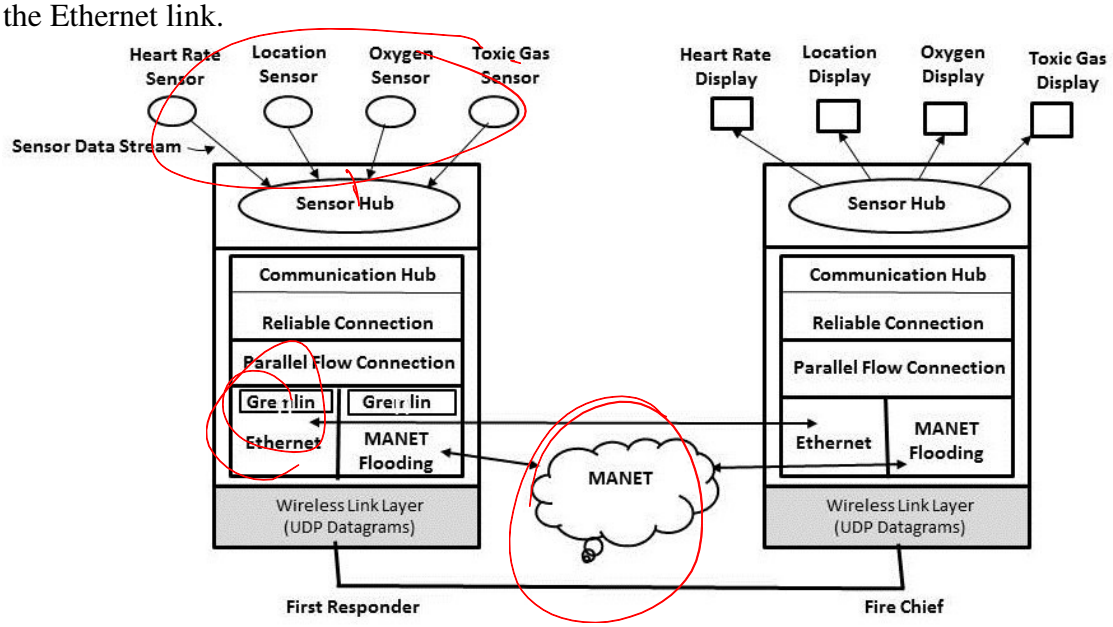

Figure 1. Overview of the Sensor Hub, Communication Hub and Reliable Connection with Parallel Flows

You will implement the reliable connection with parallel flows over an unreliable network that transmits data packets through a UDP socket. All four sensor data streams will use the same connection. When the sensor data streams arrive at the Fire Chief, you must implement an application that will display each sensor data stream on separate windows. You can code these programs in any language of your choice, i.e. C, C++ or Java.

3.1 Sensor Hub Program

You must implement at least four of the following sensors: (i) Heart rate sensor, (ii) location sensor, (iii) air-pack oxygen level, and (iv) toxic gas sensor. Each of these sensors produces independent and separate streams of sensor data that are generated at different rates. The heart rate sensor generate a sensor data every 1 second; the location sensor produce a location information every 0.5 second; air-pack sensor generate the oxygen level reading every 2 seconds, and the toxic gas sensor generates a sensor data every .25 second. Each sensor must be implemented as an independent and separate process or thread. All the sensor processes or threads must generate data continuously for at least *five minutes*. The sensor process or thread can produce the data and transmit to the sensor hub through TCP or UDP sockets or Linux pipes.

The sensor hub program will take the four streams of sensor data from the four sensors attached to the First Responder and transmit them over the reliable connection of the communication hub. The sensor hub at the receiving end (Fire Chief) will then take the sensor data streams and display the real-time sensor stream on four separate windows: (i) Heart rate sensor display, (ii) location sensor display, (iii) Air-pack oxygen level, and (iv) toxic gas level display. You will need to identify the data packets from each of the sensor stream so that the receiving sensor hub can display the sensor data in the correct window.

3.2 Reliable Connection with Parallel Flows

The sensor data streams will be transmitted through a reliable connection with parallel flows between the First Responder and the Fire Chief. You must implement two parallel flows using (i) Ethernet and (ii) Mobile and Ad hoc Network (MANET) with flooding protocol. The parallel flow mechanism must allow transmission through all possible parallel link (could depend on the bandwidth of each link). When the Ethernet link fails, the transmission will automatically switch to MANET with flooding. However, when the Ethernet link is restored and the MANET route fails, then the transmission will automatically switch to the Ethernet link.

The reliable connection is implemented over the parallel communication links for the parallel flows. Each of the parallel links will be based on unreliable network that transmits data packets through a UDP socket. You may use any reliable data transmission protocol, including Stop and Wait protocol, Go Back N protocol and Selective Repeat protocol. You may use any appropriate design parameters for your reliable transmission protocol, including the window size, sequence number size and the packet size. However, in the protocols that you use, the timeout value must not be larger than 40 msec.

3.2.1 Connection through Ethernet Link

The first of the parallel communication link will be using Ethernet, similar to what you did in the first project. Assume that the communication hubs are emulated using the tux computer and are connected through the Ethernet LAN.

3.2.2 Connection through MANET with Flooding

The second of the parallel communication link will be using Mobile and Ad hoc Network (MANET) with flooding protocol. The MANET Flooding protocol is implemented in an emulated environment that consists of tux computers that represent each node in the

MANET. The following describes how you can implement the emulation of the MANET flooding protocol.

3.2.2.1 Flooding Protocol for MANETs

You will design and implement the flooding protocol on top of the simple UDP socket interface that emulates the link layer represented by the shaded portion of Figure 1. The shaded portion is not part of this project, but represents protocols that could conceptually be added or substituted at the layer where it is shown.

Initially, a source node will generate a sensor data packet from the sensor hub and forwards it to all its neighbor nodes through its Flooding Protocol (FP) layer. Eventually the packet will be forwarded through multiple hops and reach the destination address. The FP layer of the intermediate nodes performs the task of forwarding packets from one node to another. The packet header must contain at least the following fields:

1. Sequence number
2. Source address
3. Destination address
4. Previous hop

One issue in FP forwarding is the addresses in the packet header. Every wireless node in the ad-hoc network has a globally unique 16-bit address (not IP address), which you can define. Since we will not be focusing on unicast or multicast forwarding, the voice packet's destination address will contain a special broadcast address. Although typically the higher-half of the address space is reserved for distant network broadcast addresses, you will use only a single network in this project, in which case, all the address bits of the broadcast destination address is set to 1.

The FP packets are limited to 128 bytes, including a header. For longer messages, the higher layer protocols or applications must perform the corresponding segmentation and reassembly of the message to/from different packets.

You must calculate the packet drop rate for each link which is proportionate to the square of the distance. Develop a function to compute the packet drop rate. When the error rate is above a certain threshold value, consider the node is out of range. Set your packet loss function so that the range of each node's transmission is about 100 meters. The error rate should capture the propagation loss (optional to include the effect of slow fading and fast fading). You can calculate the distance between nodes based on the current location.

Each router will run the flooding protocol as follows. When a node with address j receives a packet, it first checks the source address i of that packet. If $i = j$, then the node will discard the packet, since it is the source of the packet. Otherwise it checks in its *cache table* the largest sequence number of broadcast packets that it has received from source i. (Each FP layer maintains a *cache table of broadcast packets* that it has received from each source with the largest sequence number of packets that it has received from it up to that point, i.e. the cache table contains information on flooding packets that the node has forwarded previously.) If that sequence number in the cache table is smaller

than the sequence number in the new packet, then that packet is new and *needs to be forwarded.* The router *j* then updates the cache table with the largest sequence number that it has received from *i*. It then forwards a copy of the packet to *each* of its neighbors that is not the neighbor *k* that it received the packet from. The neighbor *k* can be determined from the "Previous Hop" field specified in the packet header. Eventually the packet is sent to all reachable wireless nodes, including the destination of the voice packets.

Some of the design issues for FP that you must address are as follows:

1. FP packet header
2. Forwarding mechanism for flooding packets
3. Structure of cache tables
4. Memory management of packet buffers

3.2.2.2 Network Configuration for MANET

Each network node will be emulated by a separate execution of your program in one of the tux computers. Nodes will run on a subset of the COE instructional Linux workstations in Shelby 2125, using the computers tux199 – tux209 with each node corresponding to a (machine IP address, UDP port) pair. Use only port numbers that are assigned to you. Therefore, each execution of your program will represent a virtual node communicating with other virtual nodes, each through its own UDP port. There may be many such virtual nodes running on a single workstation or running on several workstations.

A configuration file will specify the virtual nodes in the network and the links between them. Your program will take the configuration file as a command line argument. Then, the configuration file will have to be parsed and a copy of your program will be executed in each virtual node. The format of the network configuration file is shown next, while the topology of the network represented by that file is shown in Figure 2. The x- and y-coordinates of each node is specified preceding the link keyword. For example, the coordinate of Node 1 is (50, 120).

```
Node 1 tux175, 10010 50 120 links 2
Node 2 tux175, 10011 80 120 links 1 3 4
Node 3 tux180, 10010 180 170 links 2 4
Node 4 tux180, 10011 150 60 links 2 3
```

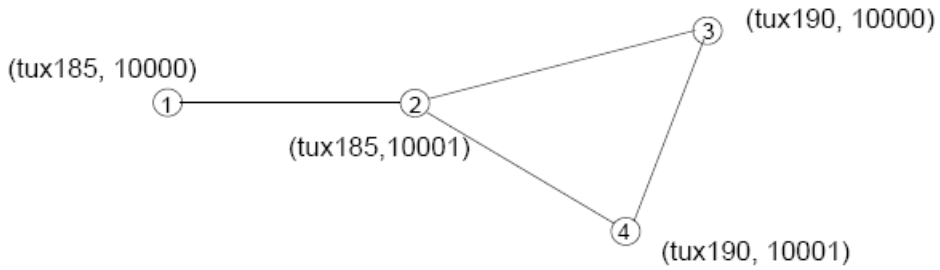

Figure 2. Network Topology

To emulate mobility of hosts, your program running in each virtual node must constantly monitor any change in the configuration file. When a mobile host moves to another location, its link to the fixed hosts in the previous location is disconnected while a new link is established to fixed hosts in the new location. The change in the configuration file can be manually done or automatically updated by a randomized program. The configuration file may also contain location information of each node. Each node periodically read the configuration file to determine the set of neighbors that it can reach through a wireless links and their new locations. The connectivity of the fixed hosts should not change.

3.3 Gremlin Function

In order to test your reliable transmission, implement a Gremlin Function that may lose packets at a specified percentage. There are two Gremlin Functions, one placed before the UDP socket for the Ethernet link and another placed before the UDP socket representing the source node in the MANET. Before your reliable connection protocol sends that packets to the UDP socket, either for the Ethernet link or MANET link, it will first send them to the Gremlin function which may randomly lose some of the packets. Packets that are not selected to be lost will be sent by the Gremlin function through the UDP socket interface.

You must be able to set the *loss percentage* though *two arguments* when the program is executed, the first is the loss percentage for the Ethernet link and the second is the loss percentage for the MANET link. For example, if the loss percentage is 0.3, then 30% of the packet will be randomly dropped by the Gremlin Function. When the loss percentage of the Ethernet link is 100%, this indicates that the Ethernet link has failed and the Parallel Flow Connection will automatically switch all its traffic to the MANET link.

4 Testing

Your reliable communication hub application programs must execute correctly in the College of Engineering Linux tux computers in Shelby 2125. The communication hub application must allow the First Responder to communicate with the Fire Chief, whereby each user must use a different tux computer. You will verify that your program executes correctly with all the sensor data streams transmitted reliably even when the Gremlin Function may lose packets at any specified loss rate. *Your program must continuously monitor the throughput of the data traffic passing through each of the parallel links. Show that when the Ethernet link goes down, the MANET will transmit 100% of the*

sensor data traffic. When the MANET goes down, the Ethernet will transmit 100% of the traffic.

6 What to do and turn in

After you have implemented, debugged, and thoroughly verified your reliable communication hub implementation, submit the following in Canvas on or before the due date.

a. Complete communication hub and sensor hub application codes (which should include comments for full credit)
b. A brief (2 pages) report that describes your system architecture, design and implementation issues
c. Capture some execution trace of the programs that show your program executes correctly and is reliable. In Linux, use the `script` command to capture the trace of the execution of the programs. The trace must contain information when packets and ACK/NAK are sent or received and when packets are lost. Sequence numbers and other relevant information on the packets must be printed.

You will demonstrate your program to the TA or me on April 5, 2017 in Shelby 2125 on the COE network of Linux computers.