Packet.h/cpp

Purpose: Packet defines the packet class, which is contained inside the buffer queue of each router. Packets contain a string Header, and two ints, destination and source. The packet class simulates a simplified version of the network layer packet.

Functions:

```
string getHeader();
void setHeader(string header);
```

These functions return and set the value of the string Header to header.

```
Packet(string header);
Packet();
```

These are the Packet class constructors. The first constructor accepts a string and sets the Header to the header variable, the second constructor creates a packet without setting any of the packet variables.

```
void setDestination(int dest);
int getDestination();
```

These functions get and set the value for the packets int destination variable. This variable is important, because it is used by the Graph class to determine if a packet has reached its intended destination.

```
void setSource(int src);
int getSource();
```

These functions get and set the int source variable. The source variable indicates which router the packet originates from.

Router.h/cpp

Purpose: Router defines the router class, which stores the amount of packets into the router. The router class simulates a network layer and transposes the packets. Router contains queue<Packet> packet_queue, which simulates the buffer queue, bool buffer_flag which signals if the router is getting full, and int ID, which identifies each router in the graph.

Functions:

```
Router();
```

This constructor creates a router. None of the variables inside the Router class are set in this constructor.

```
bool add_packet(Packet packet);
```

This function adds a packet to a router. There are several if statements to check if the buffer queue is getting full and if the packets are at their intended destination.

If the buffer queue is not getting full (there are less than 7 packets in the queue), then we add the packet to the routers buffer queue and return true, to indicate that the packet was successfully “sent” to the router.

If the buffer queue is getting full (greater than 7) but not yet full (greater than 10) then set_flag is called, setting the buffer_flag to indicate that the router is getting congested. The packet is added to the buffer queue and the function returns true.

If the buffer queue is full and the packets are at their intended destination, then the packets are “acknowledged” with a cout statement, and the buffer queue is completely emptied. The set_flag function is called to set the buffer flag to 0, indicating that the router is no longer congested. The function returns false to indicate that the router was full and the packet was dropped.

If the buffer queue is full (greater than 10) and the packets are not at their intended destination, the buffer queue is cleared of half of all its packets, simulating that they are dropped. After clearing the buffer queue, the buffer flag is set to 0 to indicate that the buffer queue is no longer full and the router is no longer congested. The function returns false to indicate that the packet was dropped because the router was full.

If none of the conditionals are met, the function returns false.

```
int getPacketTotal();
```

This function returns the number of packets in a router.

```
void remove_packet();
```

This function removes the packet from the front of the buffer queue.

```
Packet getPacket();
```

This function returns the packet at the front of a router's buffer_queue.

```
void print_queue();
```

This function prints the packets in the buffer flag queue.

```
void set_flag(bool buffer_flag);
bool get_flag();
```

These functions get and set the value of the buffer flag.

```
int getID();
void setID(int ID);
```

These functions get and set the ID from each router. The ID variable is important, because we check a packet's destination to see if it matches the ID of the current router.

```
void depop();
```

This function deletes the packets inside of the router and sets the buffer flag to false once it is done.

Graph.h/cpp

Purpose: Graph creates the mapping for the router nodes in the simulated network, as well as the traversal of the packets between the nodes. Graph contains the primary variable of int router_num; which designates the number of routers in the graph, as well as a vector called vector<Router> line, which contains all of the routers in the graph, and an adjacency_matrix of ints which stores how each router is connected to each other.

Functions:

```
Graph(int N);
```

The constructor function accepts an int N, that determines how many routers will be inside the graph. For every new router created, we add the router to the line vector, set the buffer flag to false, and give each router a unique ID. The adjacency matrix is set by default to 0 since we don't know if the routers will be connected or not.

```
void add_edges(int i, int j);
```

This function connects the routers to each other in the adjacency matrix.

```
bool is_connected(int i, int j);
```

This function shows if the routers are connected in the graph.

```
bool packet_path(Packet packet, int src, int dest, int packets);
```

This sits at the core of our implementation.The first part of packet_path sets the traversal path for the packets in the graph. The second part of this function is what sends the packets along this traversal path.

This function accepts a Packet variable, as well as ints source, dest, and packets.
The packet itself is what will be traversing through the graph, the src is the router where the packet is being sent from, dest is the intended destination router, and int packets is the total number of packets that will be sent from the src to dest routers.

Lines 68-72 create auxiliary variables that are used to find the path between the source and destination routers, and lines 86 through 127 actually find the path between the two routers. A breadth first search based on the minimum number of edges between the two routers in the graph is used to determine the path.

Once the path is determined, line 138 initiates a while loop which checks that the variable total_packets is greater than 0. The total_packets variable represents how many packets are left to send from src to dest, and for each successful packet that is sent from src to dest, this variable is decremented. Initially, total_packets is set to equal int packets.

Lines 142 - 230 detail the for loop that traverses through the path between src and dest. Because this for loop is nested inside the while loop at 138, the path is traversed from beginning to end multiple times, until all packets are successfully sent from src to dest.

Every iteration of the for loop at line 142 contains a while loop, beginning on line 149. This while loop keeps sending packets from one router in the path to the next router in the path until the receiving routers buffer flag is set to true, and it checks to make sure that total_packets has not reached 0. If either of these conditions are met the while loop is terminated.

The if-statement at line 154 will check if the currently selected router is the final destination. If it is, it will send a packet then check if the total number of packets to be sent is still greater than zero. If this is the case, then it will check if the packet can be sent to the router (to check in the event that the buffer queue for the router is full), decrement the total number of packets needed,

and print out the number of packets left. If the buffer queue is full and a packet cannot be sent to its intended destination, then it will track the amount of times this fails and reattempt to send a packet again. If it fails to send a packet three times, it will prompt the user that the packet traversal failed, then returns false out of the function.

Line 187 provides the else of the previous if-statement, in where if the destination router is not the intended final destination router for the packets. Here, it will attempt to send a packet, like in the previous condition. If it can, it will print to the user the total amount of packets left to be sent, if it cannot, then it will attempt to resend the packets up to three times before prompting the user that the packet traversal had failed. If it fails to send the packet after three attempts, it will return false out of the function.

Once the while loop at line 149 terminates, we check to see if the router we are at is the destination router. If it is, then the packets are cleared from the destination router buffer queue, which simulates the packets being sent up to the next layer. The break statement breaks out of the for loop at line 142 and the path is traversed again if there are still more packets to send from source to destination. If we are not at the destination, the buffer queue of the receiving router is cleared, again to simulate that the packets are now being sent to the next layer. Instead of breaking out of the for loop, we continue with the next iteration.

Finally, once there are no more packets to send to the destination router, the function returns true, to signify all of the packets were successfully sent to the destination router from the source.

```
bool send_packet(Packet packet, int src, int dest);
```

Send_packet is what actually simulates the traversal of a packet between two routers connected in the graph.

This function takes three variables, a Packet variable, and two integers for src and dest. The first if-statement checks if the total packets at the source is zero. If it is, it will try to add a packet into the source. If it cannot add a packet, it returns false.

The second if-statement checks to see if a packet can be added into the destination, if it cannot it will return false.

Lastly, if the function has not returned out of the previous conditions, it will remove a packet from the source then return true.

The adding of the packet to the source and removing the packet from the source once its reached the destination simulates the packet leaving the buffer queue of the source router and being added to the destination router buffer queue.

```
Router getRouter(int router);
```

This function accepts an int, and returns the specific router in the graph where the ID matches the router int.

Main.cpp

Purpose: This file creates a graph with a specified number of routers, sets a random source and destination in the graph, and calls the packet_path function.