Cairo University - Faculty of Engineering

Computer Engineering Department

Computer Networks Project

Team One

Mohamed Shawky Zaky
SEC:2, BN:15

Remonda Talaat Eskarous
SEC:1, BN:19

Mohamed Ahmed Mohamed Ahmed
SEC:2, BN:10

Mohamed Ramzy Helmy
SEC:2, BN:13

Contents

List of Figures

1 Overall System

Our system is a **Centralized Network**, consisting of N nodes connected to a hub, the number of nodes N are given from external file. This topology is, also, referred to as *star network*. Since, we use *full-duplex* communication, then each **node** acts as a sender and a receiver at the same time, sending data and acknowledges *(piggybacked)*.

The **hub** has the following functionalities:

- Allocate sessions between nodes.
- Navigate packets and acknowledges between peers.
- It's considered the control device of access medium, so it's responsible for the different types of channel noise.
- Gather the required statistics for each node.

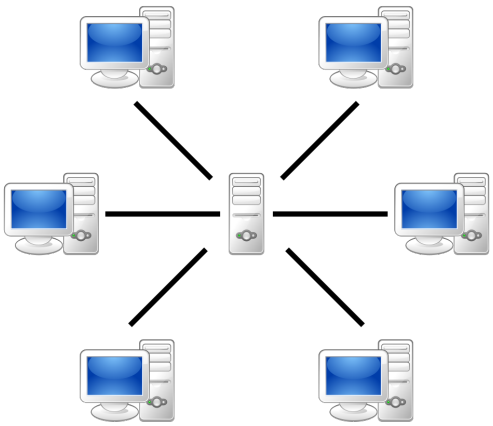

Figure 1: Centralized Star Network.

2 Implemented Functionalities

2.1 Hamming Code (Error Detection/Correction)

Hamming code is implemented to handle $1-bit$ error detection and correction. Since the message modification can only be of 1 bit, the usage of hamming code ensures that **no corrupted frames** are received. The algorithm implementation follows exactly the hamming code algorithm. *Hamming code* is applied on the message payload and the result is padded with zeros to be sent as a **string of characters**.

The implementation of the *hamming code* algorithm can be found in `src/Node.cc` under two functions :

- `string Node::computeHamming(string s, int &to_pad);`
- `string Node::decodeHamming(string s, int padding);`

2.2 Character Count (Framing)

Character Count is implemented as a **framing method** of the transmitted message. It is applied to the **message payload** to count characters of the string and **prepend** the count as one byte to the beginning of the **message payload**. The *decoding* is done in the same manner. *Character count* framing is applied **before** hamming code.

The implementation of the *character count* algorithm can be found in `src/Node.cc` under two functions :

- `string Node::addCharCount(string msg);`
- `bool Node::checkCharCount(string &msg);`

2.3 Go Back N (Sliding Window Protocol)

The used **data link protocol** is *Go Back N*, where the sender re-transmits the whole current window upon **acknowledge timeout**. The node can receive a message *(of kind 4)* from the hub, so that it can start talking to another node. It can, also, send and receive messages *(of kind 3)* to and from the other node, which contains both data and acknowledges *(piggybacked)*. Finally, the node has two kinds of self-messages. *One* is used for marking **acknowledge timeout**, so that the **window pointer** is reset. *Two* is

used for sending **piggybacked messages** at certain intervals. So, we can summarize our *Go Back N* implementation as follows :

- The node receives a message *(of kind 4)* from the hub, so that it can start transmission.
- The node sends its first **piggybacked message** with acknowledge of -1 and sends a self-message *(of kind 2)* to schedule **next message** to be sent and a self-message *(of kind 1)* to set **acknowledge timeout**.
- When the node receives a message from the other node *(of kind 3)*, it decodes the **incoming frame** to advance its **excepted frame count**. Also, it decodes the **incoming acknowledge** to advance **its window**.
- When the node receives self-message *(of kind 1)*, it checks whether the window is advanced. Accordingly, it can reset **window pointer**.
- When the node receives self-message *(of kind 2)*, it sends the next message to the hub.
- When the node finishes its **message buffer**, it sends an *"end session"* node to the hub.

The implementation of the *Go Back N* protocol can be found in `src/Node.cc` under functions :

- `void Node::handleMessage(cMessage *msg);`
- `void Node::sendMsg();`
- `void Node::post_receive_ack(cMessage *msg);`
- `void Node::post_receive_frame(cMessage *msg);`

2.4 Transmission Channel Noise Modelling

As mentioned, the hub acts as a **medium controller**, that is why the hub is responsible for adding **noise** and **delay** to the transmitted messages. The implemented noise types are *1-bit* **modification** *(on message payload)*, **delay**, **drop** and **duplication**. The noise is added with **random probability** and **multiple types** might be applied on a single message at once.

The implementation of the *noise modelling* can be found in `src/Hub.cc` under function :

- `int Hub::applyNoise(Imessage_Base *msg);`

2.5 Centralized Network Architecture

Since we are implementing a **centralized network**, we use a **hub** to communicate between nodes. The hub is, mainly, responsible for allocating **sessions** in the following way :

- Generate a **table of pairs** at the beginning of the simulation that includes all the node pairs to communicate.
- At *each session time*, the hub starts a new session through sending the two nodes a message *(of kind 4)* to start transmission.
- The hub schedules a self-message *(of kind 1)*, as well, in order to mark the **beginning** of a new session.
- The hub continues to *re-direct* the messages, until one node sends an *"end session"* message or the session *times out*.
- The hub starts a *new session* between two new nodes and **ignores** any other messages.

The implementation of the *centralized network* can be found in `src/Hub.cc` under function :

- `void handleMessage(cMessage *msg);`
- `void generatePairs();`
- `void startSession();`
- `void parseMessage(Imessage_Base *msg);`

2.6 Statistics Gathering

The **statistics gathering** function is implemented inside the hub, since the hub is the one controlling the **whole transmission process**. Also, the **transmission noise** is created in the hub. So, the hub keeps track of the **generated**, **lost**, **re-transmitted** and **duplicated** message for every node. It sets up a self-message *(of kind 2)* to schedule the **statistics print**. **Statistics** can be printed for each node *separately* or *collective* for all nodes. So, the following statistics are printed :

1. The total number of **generated** frames.
2. The total number of **dropped** frames.

3. The total number of **re-transmitted** frames.

4. The **percentage** of useful transmitted data *(Efficiency of the system)*.

The implementation of the *statistics gathering* can be found in `src/Hub.cc` under function :

- `void Hub::initializeStats();`
- `void Hub::updateStats(int node_idx, int seq_num, int frame_size, bool is_dropped, bool is_duplicated);`
- `void Hub::printStats(bool collective);`

3 Workload Division

Name	Work
Remonda Talaat Eskarous	- Centralized Network *(Hub)*. - Transmission Channel Noise Modelling. - Code Integration.
Mohamed Shawky Zaky	- Character Count. - Statistics Gathering. - Code Integration. - Final Report.
Mohamed Ahmed Mohamed Ahmed	- Go Back N.
Mohamed Ramzy Helmy	- Hamming Code.