Link Layer Emulator

In this assignment we will develop a link layer emulator to a fictitious network called DCCNET. The emulator will handle framing, sequencing, error detection and data retransmission.

Framing

The format of a DCCNET frame is shown in Diagram 1. The sequence used for synchronization between end points (`SYNC` , 32 bits) is always `0xDCC023C2` . The error detection field (`chksum` , 16 bits) uses the Internet checksum algorithm for error detection. The size field (`length` , 16 bits) counts the number of bytes of data transmitted in the frame; the `length` field does *not* count frame header bytes. The `ID` field (16 bits) identifies the transmitted or acknowledged frame. The `length` and `ID` fields must be transmitted using network byte order (big-endian).

```
DIAGRAM 1: DCCNET Frame Format

0        32       64         80         96     104   112
112+length
+---/----+---/----+---------+---------+-----+-----+------ ... ---+
|  SYNC  |  SYNC  | chksum  | length  | ID  |flags| DATA         |
+---/----+---/----+---------+---------+-----+-----+------ ... ---+
```

Sequencing

We will emulate the DCCNET link layer using the stop-and-wait flow control algorithm, that is, both the transmission and reception windows will hold one frame only. Consequently, the values of the `ID` field are restricted to zero or one. Below we detail the functioning of a node to transmit and receive data. Note that, as DCCNET links are full-duplex (data can be transmitted in both directions), each node functions as a transmitter and receiver simultaneously.

Transmitter

To transmit data, the transmitting node creates a frame as specified in Diagram 1. Data transmission frames have a value greater than zero in the `length` field; in other words, at least one byte of data must be transmitted in each data frame (unless it is the termination frame, see below). The `ID` field contains the frame identifier. While the transmitted frame is not confirmed, it must be retransmitted periodically every second. Upon receipt of the acknowledgement, the transmitter must change the value of the `ID` field (from zero to one, or from one to zero) before transmitting the next frame. After initialization, the first transmitted frame must have `ID` zero.

Receiver

A data frame can only be accepted if it has an identifier (`ID`) different from the identifier of the last frame received and no error is detected (see below). Upon accepting a data frame, the receiver must respond with an acknowledgement frame. Acknowledgement frames do not carry data, have the `ACK` flag set and `length` equal to zero. Diagram 2 describes the meaning of each flag bit and indicates the `ACK` bit. The acknowledgement frame's `ID` field must be identical to that of the acknowledged data frame.

```
DIAGRAM 2: Flag control bit meanings

Bitmask (hex)    | Bit | Name | Function
1000 0000 (0x80) |  7  | ACK  | Data reception acknowledgement
0100 0000 (0x40) |  6  | END  | End-of-transmission bit
0011 1111 (0x3f) | 5-0 | ---  | Reserved. Should be set to zero.
```

The receiver must keep in memory the identifier (`ID`) and the checksum (`chksum`) of the last received data frame. If one received frame is a retransmission of the last received frame, the receiver must re-send the acknowledgement frame. A frame is considered a retransmission of the last received frame if it has the same identifier (`ID`) and the same checksum (`chksum`).

Error Detection

Transmission errors are detected using the checksum present in the frame header. The checksum is the same as that used on the

Internet and is calculated over the entire frame, including header and data. During checksum calculation, the header bits reserved for the checksum must be considered as zero. Frames with error must not be accepted by the destination nor confirmed.

To detect the beginning of each frame, your emulator must wait for two occurrences of the synchronization pattern (`SYNC`) that mark the beginning of a frame. Detecting the beginning of each frame is important and more challenging after a transmission error, particularly if the transmission error corrupts the `length` field. When the `length` field is corrupted, your emulator will lose track of where the current frame ends and the next frame starts. Your emulator should continuously wait for two occurrences of the synchronization pattern (`SYNC`) and attempt to receive frames to recover from errors.

End of Transmission

When the transmitter sends the last data frame it must set the `END` bit of the `flags` field to indicate that there will be no more data frames after this frame, or send an empty frame with the `END` flag on and a new sequence number. The receiver must acknowledge the frame that has the `END` flag set normally and keep the information that the other side no longer has data to transmit. Note that only the last frame of data must have the `END` flag set; in particular, acknowledgement frames (with `length` zero and the `ACK` flag set) should never have the `END` flag set. When a process receives and acknowledges the last data frame with the `END` flag set and has nothing else to transmit, it must close the connection and terminate execution.

Implementation Details

You must implement DCCNET over a TCP connection. Your DCCNET emulator must be able to function as transmitter and receiver simultaneously. Your emulator must interoperate with other emulators (test with colleagues' emulators), including the reference emulator implemented by the instructor.

Grading Application 1

To test your implementation of DCCNET, you should also create an application that will exchange data with the grading server. Your app should establish a DCCNET connection with the grading server (information posted on the course website), and compute MD5 checksums for each line of ASCII text received from the server. More precisely, the server will continuously send `\n` -terminated lines of ASCII text to your program. After each line of text is received, your program should compute the MD5 checksum of the line of text (including the `\n` byte). After computing the MD5 checksum, your program should transmit it to the server as a hexadecimal string (containing only the characters `a-f` and `0-9`); transmit one linebreak (`\n`) after each MD5 checksum. The program should terminate after all lines are received and the MD5 checksums acknowledged.

Grading Application 2

As another test, you should also create an application that will send data from a file to the remote endpoint, and save all data received from the remote endpoint into a file. When sending data, the program should terminate the transfer after sending the complete file. When the files finish transmission in both directions, the program should terminate.

Grading

This assignment can be implemented in Python, C, C++, Rust, or Java but must interoperate with emulators written in other languages.

Any inconsistency or ambiguity in the specification must be reported to the instructor; depending on the inconsistency or ambiguity, the student who first report it will receive extra credit.

Submit your documentation in PDF format with up to 4 pages (two pages), using font size 10. Documentation should discuss project challenges, difficulties and unforeseen events, as well as the solutions adopted for problems. In particular, your documentation should:

- Discuss the mechanism implemented to recover framing after errors;
- Discuss how it handles transmission and reception in parallel;
- Discuss how the applications interface with the DCCNET implementation.

User Interface

The file transfer application should be executable as a passive server (wait for connections) or active client (establish a connection). The passive server should be executed as follows:

```
./dccnet-xfer -s <PORT> <INPUT> <OUTPUT>
```

Where `PORT` number where the program will listen for a connection. All data exchanged through this port must follow the DCCNET protocol. The program must wait for connections on all IPs of the machine it is running on. `INPUT` is the name of a file with the data that will be sent to the remote end of the link, and `OUTPUT` is the name of a file where the data received from the remote end of the link will be stored.

The active client should be executed as follows:

```
./dccnet-xfer -c <IP>:<PORT> <INPUT> <OUTPUT>
```

Where the `IP` parameter is the IP address of the machine where the passive server is running. The other parameters have the same meaning as above. (Note that `IP` can be `127.0.0.1` if the programs are running on the same machine, but it can also be the address of another machine accessible over the network.)

Your MD5 application using DCCNET must be executed as follows:

```
./dccnet-md5 -c <IP>:<PORT>
```

Where the `IP:PORT` are the IP and port number provided in the course website for the automated grading server.

Tips

Below is an example of a DCCNET frame transmitting four bytes with values 1, 2, 3 and 4 (in this order). The following bytes, in hexadecimal notation, will have to be transmitted at the link layer. This example assumes that the frame identifier (`ID`) is zero and that more data needs to be transmitted (so the `END` flag is not set):

```
dcc023c2dcc023c2faef0004000001020304
```

A Note on the Tests with Transmission Errors

The version of the program to be submitted must work correctly even if there is an error in any of the received frames. As the programs will run over the TCP protocol, there will never be wrong bytes upon arrival, as the TCP connection already performs error checking. Bytes received with `recv` will match exactly to the bytes sent with `send`. However, in testing, we can use programs that deliberately, purposefully generate and `send` corrupted frames, causing the remote end's `recv` to return corrupted frames and triggering the code to detect synchronization errors, compute checksums, etc.

To test your program in these cases you can create modified versions of the program that intentionally generates and sends corrupted frames to see how the remote program behaves. It is not necessary to hand over these programs used for testing, only the programs that works according to this specification. The instructor has a test program that can inject these errors in the connection during testing. The grading server will also generate corrupted frames to exercise during the MD5 checksum tests.