A Simple Messenger Application: Client-Side Implementation

EE5150: Communication Networks, Jan-May 2025

1 Introduction

This document outlines the server protocol and requirements for developing the client-side application of a simple e-messenger system. The server will manage sessions, store messages, and handle delivery. Students are expected to implement the client application to run in their browsers, adhering to the protocol described herein.

2 Requirements

The client must implement the following functionality:

- **Session Management**: Establish a session with the server before any communication. The session may expire after a specified period of inactivity.
- **Message Exchange**: Send and receive data packets through the server using the following message types:
  - `ASSOCIATE`: Initiate a session with the server.
  - `GET`: Retrieve messages stored on the server for the client.
  - `PUSH`: Send messages to the server for other clients to access.
- **Handling Server Responses**: Process server responses, including status messages regarding session establishment, buffer status, and other error scenarios.

3 Communication and Responses

The server responds to specific communications or events as described below:

- **Association request:** Packet type `MANAGEMENT` with message `ASSOCIATE`
  - The client sends a `MANAGEMENT` type packet to the server to initiate a session with the message `ASSOCIATE`.
  - The server responds with a `MANAGEMENT` type packet containing one of the following messages:
    - `ASSOCIATIONSUCCESS`: Indicates that the client is associated successfully and a session is established, allowing the client to send and receive data messages from the server.
    - `ASSOCIATIONFAILED`: Indicates that the association failed, possibly due to resource constraints.
    - `UNKNOWNERROR`: Indicates an unforeseen situation.
- **Get data from the server:** Packet type `CONTROL` with message `GET`
  - The client sends a `CONTROL` type packet with the message `GET` to retrieve one data message intended for it from the relevant stack at the server.
  - The server responds with one of the following messages:

    * Data with `GETRESPONSE`: A data packet (payload) is sent to the client from its data stack at the server using a `DATA` type packet with message type `GETRESPONSE`, provided there are data messages stored for the client. The messages are served and subsequently removed from the finite buffer in FIFO manner.
    * `BUFFEREMPTY`: A `CONTROL` type packet with message `BUFFEREMPTY` is sent if the client's buffer queue is empty.
    * `ASSOCIATIONFAILED`: A `MANAGEMENT` type packet with message `ASSOCIATIONFAILED` is sent if the association with the client has not been successfully established yet.
    * `UNKNOWNERROR`: A `MANAGEMENT` type packet with message `UNKNOWNERROR` is sent in other scenarios.

- **Push data to the server:** Packet type `DATA` with message `PUSH`
  - The client sends a `DATA` type packet to the server with the message `PUSH` and a data payload, to forward data messages to other users in the system. The messages are expected to be stored in the relevant buffer of the `receiver_id` client at the server.
  - The server responds with one of the following messages:
    * Positive acknowledgement: A `CONTROL` type packet with message `POSITIVEACK` is sent if the data is correctly received and stored in the buffer at the server.
    * Negative acknowledgement: A `CONTROL` type packet with message `BUFFERFULL` indicates that the buffer for the intended receiver at the server is full, and the packet is not stored.
    * `ASSOCIATIONFAILED`: A `MANAGEMENT` type packet with message `ASSOCIATIONFAILED` is sent if the association with the client has not been successfully established yet.
    * `UNKNOWNERROR`: A `MANAGEMENT` type packet with message `UNKNOWNERROR` is sent in other scenarios.

4 Packet Frame Format

All communication between the client and server is conducted using packets. Each packet consists of a header and may include a payload. All fields except the payload are 1-byte unsigned integers and stored as binary values. The payload has a variable length, maximum size is 254 bytes, and shall be encoded in UTF-8 format.

- **Management packets:** 3 bytes

```
{
    "type": MANAGEMENT (0),
    "message": ASSOCIATE (0) or ASSOCIATIONSUCCESS (1)
    or ASSOCIATIONFAILED (2) or UNKNOWNERROR (3),
    "id": client_id
}
```

- **Control Packets:** 3 bytes

```
{
    "type": CONTROL (1),
    "message": GET (0) or BUFFEREMPTY (1) or POSITIVEACK (2) or BUFFERFULL (3),
    "id": client_id
}
```

- **Data Packets:** 5 bytes header and variable-length data

```
{
    "type": DATA (2),
    "message": GETRESPONSE (0) or PUSH (1),
```

```
    "id": client_id,
    "id2": sender_id or receiver_id,
    "length": length,
    "payload": data (up to 254 bytes)
}
```

The `id2` corresponds to the `client_id` of the sender of the message in the GETRESPONSE message. The `id2` corresponds to the `client_id` of the intended receiver in the PUSH message.

5 Conclusion

Students must implement the client-side application to adhere to the protocol described in this document. The server will handle session management, message storage, and delivery. Ensure that all packets are formatted correctly and that the handshake mechanism is followed.