CS 2620 Custom Wire Protocol

We designed a custom wire protocol that encodes each transmission object into an efficient binary representation.

By packing data into binary representations with fixed-size headers and explicit lengths, the protocol minimizes overhead compared to text-based encodings (such as JSON) while still supporting complex, variable-length messages.

There are 2 different transmission types where the custom wire protocol happens:

1. Client sends request to server (Server handles the client request)
2. Server sends response to client (Client handles the server response)

Client sends request to server

Common Header

Each client request begins with a **2-byte header**:

- **Version (1 byte):**

  The protocol version (currently set to `1`). This ensures that both client and server agree on the message format.

- **Opcode (1 byte):**

  A numerical code that identifies the specific action being requested. For example, opcodes for actions such as LOGIN, CREATE_ACCOUNT, or SEND_MESSAGE are defined in a dictionary (e.g., `"LOGIN": 1, "CREATE_ACCOUNT": 2, ..., "QUIT": 9`).

All multi-byte fields in the payload are encoded in network byte order (big-endian) using the `struct` module's format strings (prefixed with `!`).

Payload Formats by Request Type

After the header, the remainder of the message (if any) is the **payload**, whose format depends on the action being performed:

1. **LOGIN & CREATE_ACCOUNT**

   (Both requests use the same payload structure.)

   - **Payload Fields:**
     - **Username Length (2 ytes):**An unsigned short indicating the length of the username.
     - **Username (variable bytes):**The username encoded in UTF-8.
     - **Password Length (2 bytes):**An unsigned short indicating the length of the password hash.
     - **Password Hash (variable bytes):**The SHA-256 hash of the password (in hexadecimal form) encoded in UTF-8.
   - **Example Format String:***(Here, `BB` covers the header: version and opcode.)*

     ```
     f"!BBH{len(username_bytes)}sH{len(password_bytes)}s"
     ```

2. **DELETE_ACCOUNT**

   - **Payload:**No additional data is required. The message consists solely of the header (`VERSION` and the DELETE_ACCOUNT opcode).

3. **LIST_ACCOUNTS**

   - **Payload Fields:**
     - **Page Size (2 bytes):**An unsigned short indicating how many account entries to return.
     - **Page Number (2 bytes):**An unsigned short indicating which page of results is being requested.
     - **Pattern Length (2 bytes):**An unsigned short representing the length of the account name pattern.
     - **Pattern (variable bytes):**A UTF-8 encoded string that specifies a search pattern (or wildcard, e.g., `"*"`).
   - **Example Format String:**

```
f"!BBHHH{len(pattern_bytes)}s"
```

4. **SEND_MESSAGE**
   - **Payload Fields:**
     - **Recipient Length (2 bytes):**An unsigned short specifying the length of the recipient’s username.
     - **Recipient (variable bytes):**The recipient’s username encoded in UTF-8.
     - **Message Length (2 bytes):**An unsigned short indicating the length of the message text.
     - **Message (variable bytes):**The message content encoded in UTF-8.
   - **Example Format String:**

     ```
     f"!BBH{len(recipient_bytes)}sH{len(message_bytes)}s"
     ```

5. **READ_MESSAGES**
   - **Payload Fields:**
     - **Page Size (2 bytes):**An unsigned short for the number of messages per page.
     - **Page Number (2 bytes):**An unsigned short for the requested page.
     - **Optional Chat Partner:**To optionally filter messages by a specific chat partner:
       - **Flag (1 byte):**A byte set to `1` if a chat partner is provided, or `0` if not.
       - **(If flag is `1`) Partner Length (2 bytes):**The length of the chat partner’s username.
       - **(If flag is `1`) Chat Partner (variable bytes):**The chat partner’s username in UTF-8.

- **Encoding Note:**The message is first encoded with the fixed fields. If a chat partner is specified, the flag and additional partner fields are appended; otherwise, only a flag byte of `0` is added.

6. **DELETE_MESSAGE**
   - **Payload Fields:**
     - **Count (1 byte):**A single byte indicating how many message IDs follow.
     - **Message IDs (each 4 bytes):**For each message to be deleted, an unsigned integer (4 bytes) is provided.
   - **Example Format String:***(Here, the first two `B`'s are for the version and opcode, followed by the count and then each message ID.)*

     ```
     f"!BBB{count}I"
     ```

7. **CHEC_USERNAME**
   - **Payload Fields:**
     - **Username Length (2 bytes):**An unsigned short for the length of the username.
     - **Username (variable bytes):**The username encoded in UTF-8.
   - **Example Format String:**

     ```
     f"!BBH{len(username_bytes)}s"
     ```

8. **QUIT**
   - **Payload:**No payload is necessary. The client simply sends the header with the opcode corresponding to QUIT.

Server sends response to client

1. Common Header

Every server-to-client message starts with a 4-byte fixed header. The header fields are as follows:

- **Version (1 byte):**

  This field specifies the protocol version (currently set to 1). The client will verify that the version matches the expected value. If there is a mismatch, the message is considered invalid.

- **Opcode (1 byte):**

  The opcode indicates the type of response or event. For standard responses, the opcode is chosen based on the operation's name using a mapping (for example, responses related to listing accounts, reading messages from a conversation, or reading general unread messages are encoded with opcodes defined in a dedicated response dictionary). For push events, the server typically uses opcode 0.

- **Payload Length (2 bytes):**

  This unsigned short field tells the client how many bytes follow the header to form the payload. It ensures that the client waits for the full payload before attempting to decode the message.

2. Payload Structures

The content of the payload depends on whether the message is a **standard response** or a **push event**. Below are the details for each type:

A. Standard Responses

Standard responses are created (typically via the dedicated binary encoding function) to acknowledge a client's request. Their payload has the following structure:

1. **Success Flag (1 byte):**
   - Indicates whether the operation was successful.
   - A value of `1` means success; `0` indicates failure.

2. **Message Field:**
    - **Message Length (2 bytes):** An unsigned short specifying the length (in bytes) of the human-readable status or error message that follows.
    - **Message (variable bytes):** A UTF-8 encoded text string that may contain details such as “Login successful” or an error explanation.
3. **Data Field:**
    - **Data Length (2 bytes):** An unsigned short that indicates the size of the additional binary data that follows.
    - **Data (variable bytes):** This block is used to send extra structured information when needed. Its format depends on the type of response:
        - **For Listing Accounts:** The data includes:
            - A 4-byte unsigned integer representing the total number of accounts.
            - A 2-byte unsigned short that provides the count of account names in the current page.
            - For each account, a 2-byte unsigned short (the length of the account name) followed by the UTF-8 encoded account name bytes.
        - **For Reading Conversation Data (when a chat partner is specified):** The data includes:
            - The chat partner’s identifier, which is encoded as a 2-byte length field followed by the corresponding UTF-8 bytes.
            - Paging information (page number and page size as 2-byte unsigned shorts).
            - Total message count (4-byte unsigned integer) and remaining message count (4-byte unsigned integer).
            - A 2-byte unsigned short for the number of messages in the current page.
            - For each message:

            - A 4-byte unsigned integer message ID.
            - A 2-byte length field followed by the message content (UTF-8 encoded).
            - A 1-byte read flag (`1` for read, `0` for unread).
    - **For Reading Unread Messages (general case):** The data includes:
        - Two 4-byte unsigned integers representing the total unread count and the number of remaining unread messages.
        - A 2-byte unsigned short for the number of messages included.
        - For each message:
            - A 4-byte unsigned integer message ID.
            - A sender field that consists of a 2-byte length followed by the sender's name (UTF-8).
            - A content field, which is a 2-byte length followed by the message content (UTF-8).
            - A 1-byte read flag.

The server encodes these responses using a dedicated routine that first builds the payload (by packing the success flag, message details, and the additional data block) and then prepends the header (using the correct version, opcode, and computed payload length).

B. Push Events

Push events are used to notify clients of asynchronous events (for example, new messages). Their encoding differs slightly from standard responses:

- **Header:**

  Although the header still consists of the version (1 byte), opcode (typically 0), and payload length (2 bytes), the meaning here is that the payload carries an event notification.

- **Payload:**

Instead of the multi-field binary structure used for responses, push events are encoded as a JSON-formatted string. This JSON object typically contains:

- An `"event"` key that specifies the event type (for instance, `"NEW_MESSAGE"`).
- A `"data"` key that contains the event-specific information.

The JSON string is then converted into UTF-8 bytes, and its length is included in the header's payload length field.
