Data types proposal

Hello. This document shows and explains various proposed changes to the main data types we will deal with for our project.

If you have the patience to read through the whole thing, your opinion will be of great value to our discussion in the next working group meeting.

Our purpose is to find the best solution (hopefully in time to code the thing).

You are very welcome to criticize the following.

1. Message

Proposed changes:

- Omission of **<M>** and **m: M** as MessageType is too big. The purpose of this is to avoid HUGE pattern matching, in favor of nested match-es. More on this at point 2.
- Omission of **fragment_header**: it is not necessary as the fragments already have a FragmentHeader with a session id. Note that **message_header** already identifies the source of the message.

```
struct Message{
    message_header: MessageHeader,
    message_content: MessageContent,
    source_routing_header: SourceRoutingHeader
}
```

2. MessageContent

Instead of the MessageType enum, we use nested enums that divide the messages among the server thay are interacting with, and the kind of message itself (request or response).

All example Client-Server interactions are accounted for. Additionally, this structure makes it easier to add requests or responses if necessary.

- **message_size** was omitted in more than one case in the quest to keep only what's needed.
- In file? request: **list_length, list_of_media_ids** were removed.
- **list_of_file_ids** was deliberately set to [u64, 20].
- **media_id** was changed from &str to u64.
- media type and server type enums are not yet defined, so they are indicated with ServerKind and MediaKind.
- **MediaRequest** was made into an enum with a single possible value for consistency, even though it could be a struct.

- **error_no_media** was eliminated due to the redundancy with **error_no_files** : since the files list is the same, one response for “no items in the list” is enough. Differently, **error_media_not_found** was kept to distinguish file request and media request.

```
enum MessageContent{
    ChatMessage(ChatMessage), // chat == communication server
    TextMessage(TextMessage), // text == content server
    MediaMessage(MediaMessage)// media == content server
}

enum ChatMessage{
    ChatRequest(ChatRequest),
    ChatResponse(ChatResponse)
}
enum ChatRequest{ //(chat == communication server)
    ClientList, // => C -> S : client_list?
    MessageFor { // => C -> S : message_for?(client_id, message)
        // note: message_size omitted!
        client: u64,
        message: String,
    },
}
enum ChatResponse{
    ClientList(u64, Vec<u64>), //=> S -> C : client_list!(list_length, list_of_c
    MessageFrom{ // => S -> C : message_from!(client_id, message)
        // note: message_size omitted!
        client: u64,
        message: String
    },
    ErrWrongClient // => S -> C : error_wrong_client_id!
}
enum TextMessage{// (text == media server with text)
    TextRequest(TextRequest),
    TextResponse(TextResponse)
}
enum TextRequest{
        ServerType, //C -> S : server_type?
        FilesList, //C -> S : files_list?
        File(file_id),// C -> S : file?(file_id)    note: additional params omit
}
enum TextResponse{
        ServerType(ServerKind), //S -> C : server_type!(type)
        FilesListResponse { //S -> C : files_list!(list_length, list_of_file_ids
        list_length: char,
        list_of_file_ids: [u64,20],
    },
      ErrorNoFiles, // S -> C : error_no_files!
      File{ //S -> C : file!(file_size, file)
        file_size: u64,
        file: String,
    },
    ErrorFileNotFound, //S -> C : error_file_not_found!
}
enum MediaMessage{
    MediaRequest(MediaRequest),
    MediaResponse(MediaResponse)
}
struct MediaRequest{
         Media{  //C -> S : media?(media_id +media type+ )
                 media_id: u64,
                 media_type: MediaKind
         }
}
enum MediaResponse{
    MediaResponse { //S -> C : media!(media_size, media, +media type+)
                media_id: u64,
        media_size: u64,
        media: std::fs::File,
    },
    ErrNotFound, //S -> C : error_media_not_found!
}
```

3. Fragment

In the original protocol file, Message had both high-level components (the MessageType) and low-level components (the fragment_header). That did not work well for us because we need a clearer distinction between levels. Hence we divide a message into lower level Fragments. This is also more similar to what's done in reality, which is kinda cool.

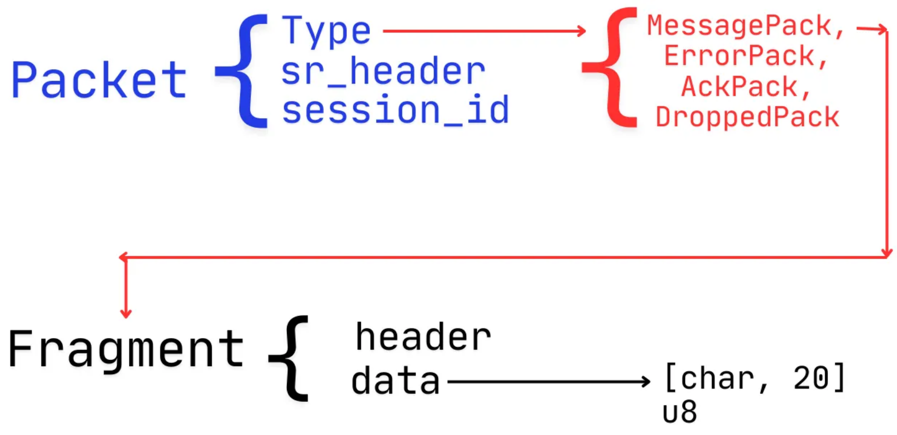
Packet is the main entity exchanged by the drones.

It has a type and a source routing header. The types are Message, Error, Ack, Dropped.

Only the Packet containing a Message is subject to fragmentation.

Packet contains Fragment. The fragmenter will take a message and give back fragments, the assembler will take fragments and give back a message.

Fragment is a piece of a message, with FragmentData holding our array of chars



- In **FragmentData,** [char, 20] could be changed into [u8, 80] to make the encoding of media easier. This is PERFECTLY EQUIVALENT to using char, since in Rust a char is made of 4 u8-s, which is the reason why we might use 80.
- Regarding **FragmentHeader**, **next_fragment** was removed and **fragment_index** was changed into a u64.
- main thing exchanged is the Packet struct
- Fragmentheader is included only for message since all other kind of packets will be only a singular Packet.
- **session_id** of error converted from char to u64.
- **id_not_neighbour** omitted from Error because not necessary: the missing node that is not a neighbour can be deduced from the route taken to return the Error. So we would confront the inital send request and the routing header of the Error
- **source_routing_header** removed from Ack, Dropped, Error because present in Packet.
- **next_fragment** was removed from FragmentHeader since we already have the current fragment index and the total number of fragments, which are enough to decide which comes first.

```
struct Packet{ //fragment defined as entity exchanged by the drones.
    pt: PacketType,
    source_routing_header: SourceRoutingHeader,
    session_id: u64
    //sourcerouting header is inverted if necessary.
}

struct Fragment{ // fragment defined as part of a message.
    header: FragmentHeader,
    data: FragmentData,
}
enum PacketType {
    MsgPack(Fragment), ErrorPack(Error), AckPack(Ack), DroppedPack(Dropped)
}
struct FragmentData{
    data: [u8; 80], //it's possible to use .into_bytes() so that images
    //can also be encoded->[u8, 80]
    length: u8 // assembler will fragment/defragment data into bytes.
}
pub struct FragmentHeader {
    /// Identifies the session to which this fragment belongs.
    session_id: u64,
    /// Total number of fragments, must be equal or greater than 1.
    total_n_fragments: u64,
    /// Index of the packet, from 0 up to total_n_fragments - 1.
    fragment_index: u64,
}
struct Error {
    session_id: u64,
    id_not_neighbor: String,
    ttl: u32,
}
struct Dropped {
        session_id: u64,
}
pub struct Ack(AckInner);
```

```
struct AckInner {
    session_id: u64,
    when: std::time::Instant,
}
```