NETWORK PROTOCOL

This file describes the network protocol used by Ceph. In order to understand the way the structures are defined it is recommended to read the introduction of Network Encoding first.

HELLO

The protocol starts with a handshake that confirms that both nodes are talking ceph and shares some basic information.

BANNER

The first action is the server sending banner to the client. The banner is defined in `CEPH_BANNER` from `src/include/msgr.h`. This is followed by the server's then client's address each encoded as a `entity_addr_t`.

Once the client verifies that the servers banner matches its own it replies with its banner and its address.

CONNECT

Once the banners have been verified and the addresses exchanged the connection negotiation begins. First the client sends a `ceph_msg_connect` structure with its information.

```
// From src/include/msgr.h
struct ceph_msg_connect {
        u64le features;             // Supported features (CEPH_FEATURE_*)
        u32le host_type;            // CEPH_ENTITY_TYPE_*
        u32le global_seq;           // Number of connections initiated by this host.
        u32le connect_seq;          // Number of connections initiated in this session.
        u32le protocol_version;
        u32le authorizer_protocol;
        u32le authorizer_len;
        u8    flags;                // CEPH_MSG_CONNECT_*
        u8    authorizer[authorizer_len];
}
```

CONNECT REPLY

Once the connect has been sent the connection has effectively been opened, however the first message the server sends must be a connect reply message.

```
struct ceph_msg_connect_reply {
        u8    tag; // Tag indicating response code.
        u64le features;
        u32le global_seq;
        u32le connect_seq;
        u32le protocol_version;
        u32le authorizer_len;
        u8    flags;
        u8    authorizer[authorizer_len];
}
```

MSGR PROTOCOL

This is a low level protocol over which messages are delivered. The messages at this level consist of a tag byte, identifying the type of message, followed by the message data.

```
// Virtual structure.
struct {
        u8 tag; // CEPH_MSGR_TAG_*
        u8 data[]; // Length depends on tag and data.
```

```
}
```

The length of `data` is determined by the tag byte and depending on the message type via information in the `data` array itself.

> **Note:** There is no way to determine the length of the message if you do not understand the type of message.

The message tags are defined in `src/include/msgr.h` and the current ones are listed below along with the data they include. Note that the defined structures don't exist in the source and are merely for representing the protocol.

CEPH_MSGR_TAG_CLOSE (0X06)

```
struct ceph_msgr_close {
        u8 tag = 0x06;
        u8 data[0]; // No data.
}
```

The close message indicates that the connection is being closed.

CEPH_MSGR_TAG_MSG (0X07)

```
struct ceph_msgr_msg {
        u8 tag = 0x07;
        ceph_msg_header header;
        u8 front [header.front_len ];
        u8 middle[header.middle_len];
        u8 data  [header.data_len  ];
        ceph_msg_footer footer;
}

// From src/include/msgr.h
struct ceph_msg_header {
        u64le seq;       // Sequence number.
        u64le tid;       // Transaction ID.
        u16le type;      // Message type (CEPH_MSG_* or MSG_*).
        u16le priority;  // Priority (higher is more important).
        u16le version;   // Version of message encoding.

        u32le front_len;  // The size of the front section.
        u32le middle_len; // The size of the middle section.
        u32le data_len;   // The size of the data section.
        u16le data_off;   // The way data should be aligned by the reciever.

        ceph_entity_name src; // Information about the sender.

        u16le compat_version; // Oldest compatible encoding version.
        u16le reserved;       // Unused.
        u32le crc;            // CRC of header.
}

// From src/include/msgr.h
struct ceph_msg_footer {
        u32le front_crc;  // Checksums of the various sections.
        u32le middle_crc; //
        u32le data_crc;   //
        u64le sig; // Crypographic signature.
        u8    flags;
}
```

Messages are the business logic of Ceph. They are what is used to send data and requests between nodes. The message header contains the length of the message so unknown messages can be handled gracefully.

There are two names for the message type constants `CEPH_MSG_*` and `MSG_*`. The only difference between the two is that the first are considered "public" while the second is for internal use only. There is no protocol-level difference.

CEPH_MSGR_TAG_ACK (0X08)

```
struct ceph_msgr_ack {
        u8    tag = 0x08;
        u64le seq; // The sequence number of the message being acknowledged.
}
```

CEPH_MSGR_TAG_KEEPALIVE (0X09)

```
struct ceph_msgr_keepalive {
        u8 tag = 0x09;
        u8 data[0]; // No data.
}
```

CEPH_MSGR_TAG_KEEPALIVE2 (0X0E)

```
struct ceph_msgr_keepalive2 {
        u8      tag = 0x0E;
        utime_t timestamp;
}
```

CEPH_MSGR_TAG_KEEPALIVE2_ACK (0X0F)

```
struct ceph_msgr_keepalive2_ack {
        u8      tag = 0x0F;
        utime_t timestamp;
}
```