OUTLINE

- What is the Ceph messenger
- Messenger API
- Messenger V1 Limitations
- Messenger V2 Protocol
WHAT IS THE CEPH MESSENGER?
WHAT IS THE CEPH MESSENGER?

- It's a wire-protocol specification;
WHAT IS THE CEPH MESSENGER?

- It's a wire-protocol specification;
- and also, the corresponding software implementation
WHAT IS THE CEPH MESSENGER?

- It's a wire-protocol specification;
- and also, the corresponding software implementation
- Invisible to end-users
WHAT IS THE CEPH MESSENGER?

- It's a wire-protocol specification;
- and also, the corresponding software implementation
- Invisible to end-users
  - Unless when it's not working properly
WHAT IS THE CEPH MESSENGER?

- It's a wire-protocol specification;
- and also, the corresponding software implementation
- Invisible to end-users
  - Unless when it's not working properly

The messenger knows nothing about the Ceph distributed algorithms and specific daemons protocols
WHERE CAN WE FIND IT?
CEPH MESSENGER (1/2)
Messenger is used as a "small" communication library by the other Ceph libraries/daemons
CEPH MESSENGER (1/2)

- Messenger is used as a "small" communication library by the other Ceph libraries/daemons
- It can be used as both server and client
  - Ceph daemons (osd, mon, mgr, mds) act as both servers and clients
  - Ceph clients (rbd, rgw) act as clients
CEPH MESSENGER (2/2)
CEPH MESSENGER (2/2)

- Abstracts the transport protocol of the physical connection used between machines
  - Posix Sockets
  - RDMA
  - DPDK
CEPH MESSENGER (2/2)

- Abstracts the transport protocol of the physical connection used between machines
  - Posix Sockets
  - RDMA
  - DPDK
- Reliable delivery of messages with "exactly-once" semantics
CEPH MESSENGER (2/2)

- Abstracts the transport protocol of the physical connection used between machines
  - Posix Sockets
  - RDMA
  - DPDK
- Reliable delivery of messages with "exactly-once" semantics
- Automatic handling of temporary connection failures
class Messenger {
    int start();
    int bind(const entity_addr_t & bind_addr);
    Connection * get_connection(const entity_inst_t & dest);

    // Dispatcher
    void add_dispatcher_head(Dispatcher * d);

    // server address
    entity_addr_t get_myaddr();
    int get_mytipe();

    // Policy
    void set_default_policy(Policy p);
    void set_policy(int type, Policy p);
};

class Connection {
    bool is_connected();
    int send_message(Message * m);
    void send_keepalive();
    void mark_down();
    entity_addr_t get_peer_addr() const;
    int get_peer_type() const;
};
class Messenger {

    Connection *get_connection(const entity_inst_t& dest);

    // Dispatcher
    void add_dispatcher_head(Dispatcher *d);

};

class Connection {

    int send_message(Message *m);

    void mark_down();

};
class Dispatcher {
    // Message handling
    bool ms_can_fast_dispatch(const Message *m) const;
    void ms_fast_dispatch(Message *m);
    bool msDispatch(Message *m);

    // Connection handling
    void ms_handle_connect(Connection *con);
    void ms_handle_fast_connect(Connection *con);
    void ms_handle_accept(Connection *con);
    void ms_handle_fast_accept(Connection *con);
    bool ms_handle_reset(Connection *con);
    void ms_handle_remote_reset(Connection *con);
    bool ms_handle_refused(Connection *con);

    // Authorization handling
    bool ms_get_authorizer(int peer_type, AuthAuthorizer **a);
    bool ms_handle_authentication(Connection *con);
};
class Dispatcher {
    // Message handling

    bool ms_dispatch(Message *m);

    // Connection handling

    void ms_handle_accept(Connection *con);

    // Authorization handling
    bool ms_get_authorizer(int peer_type, AuthAuthorizer **a);
    bool ms_handle_authentication(Connection *con);
};
MESSENGER V1 WIRE PROTOCOL
MESSENGER V1 WIRE PROTOCOL

- The first wire-protocol of Ceph
MESSENGER V1 WIRE PROTOCOL

- The first wire-protocol of Ceph
- No extensibility at an early stage of the protocol
MESSENGER V1 WIRE PROTOCOL

- The first wire-protocol of Ceph
- No extensibility at an early stage of the protocol
- No data authenticity supported
MESSENGER V1 WIRE PROTOCOL

- The first wire-protocol of Ceph
- No extensibility at an early stage of the protocol
- No data authenticity supported
- No data encryption supported
MESSENGER V1 WIRE PROTOCOL

- The first wire-protocol of Ceph
- No extensibility at an early stage of the protocol
- No data authenticity supported
- No data encryption supported
- Limited support for different authentication protocols
MESSENGER V1 WIRE PROTOCOL

- The first wire-protocol of Ceph
- No extensibility at an early stage of the protocol
- No data authenticity supported
- No data encryption supported
- Limited support for different authentication protocols
- No strict structure for protocol internal messages
MESSENGER V2 WIRE PROTOCOL (1/2)

• By default is available on the IANA port 3300 in Ceph Monitors
  ■ Messenger V1 will still be available through port 6789
MESSENGER V2 WIRE PROTOCOL (1/2)

- By default is available on the IANA port 3300 in Ceph Monitors
  - Messenger V1 will still be available through port 6789
- Only Ceph Nautilus userspace libraries support V2
  - Ceph kernel modules still talk V1
MESSENGER V2 WIRE PROTOCOL (1/2)

- By default is available on the IANA port 3300 in Ceph Monitors
  - Messenger V1 will still be available through port 6789
- Only Ceph Nautilus userspace libraries support V2
  - Ceph kernel modules still talk V1
- Still in development as Nautilus has not been released yet
MESSENGER V2 WIRE PROTOCOL (2/2)
MESSENGER V2 WIRE PROTOCOL (2/2)

• Complete redesign and implementation
MESSENGER V2 WIRE PROTOCOL (2/2)

- Complete redesign and implementation
- Extensible protocol
  - A different path can be taken in a very early stage of the protocol
MESSENGER V2 WIRE PROTOCOL (2/2)

- Complete redesign and implementation
- Extensible protocol
  - A different path can be taken in a very early stage of the protocol
- No limitations on the authentication protocols used
MESSENGER V2 WIRE PROTOCOL (2/2)

- Complete redesign and implementation
- Extensible protocol
  - A different path can be taken in a very early stage of the protocol
- No limitations on the authentication protocols used
- Encryption-on-the-wire support
MESSENGER V2 SPECIFICATION
• Actors:
  ▪ Connector
  ▪ Acceptor
MESSENGER V2 SPECIFICATION

• Actors:
  ■ Connector
  ■ Accepter

• Phases
  1. Banner Exchange
  2. Authentication
  3. Session Handshake
  4. Message Exchange
struct frame {
    uint32_t frame_len;       // 4 bytes
    uint32_t tag;             // 4 bytes
    char payload[frame_len - 4];
};

struct encrypted_frame {
    uint32_t frame_len;
    uint32_t tag;
    char encrypted_payload[frame_len - 4];
};
1. BANNER EXCHANGE

We can change the behavior of the protocol at this point based on the supported/required features.

```c
struct banner {
    char banner[8]; // "ceph v2\n"
    uint16_t payload_len;
    struct banner_payload payload;
};

struct banner_payload {
    uint64_t supported_features;
    uint64_t required_features;
}

struct hello {
    uint8_t entity_type;
    entity_addr_t peer_address;
}
```
2. AUTHENTICATION

```c
struct auth_request {
    uint32_t method;
    uint32_t preferred_modes[num_modes];
    char auth_payload[payload_len];
};

struct auth_bad_method {
    uint32_t method;
    int result;
    uint32_t allowed_methods[num_methods];
    uint32_t allowed_modes[num_modes];
};

struct auth_reply_more {
    char auth_payload[payload_len];
};

struct auth_request_more {
    char auth_payload[payload_len];
};

struct auth_done {
    uint64_t global_id;
    uint32_t mode;
    char auth_payload[payload_len];
};
```

From this point message frames can be encrypted.

- connector
- accepter
- auth_request
- auth_bad_method
- auth_request
- auth_reply_more
- auth_request_more
- auth_done
- several rounds
3. SESSION HANDSHAKE (NEW SESSION)

```c
struct client_ident {
    entity_addrvec_t addrs;
    int64_t global_id;
    uint64_t global_seq;
    uint64_t supported_features;
    uint64_t required_features;
    uint64_t flags;
};

struct server_ident {
    entity_addrvec_t addrs;
    int64_t global_id;
    uint64_t global_seq;
    uint64_t supported_features;
    uint64_t required_features;
    uint64_t flags;
    uint64_t cookie;
};
```
3. SESSION HANDSHAKE (RECONNECT)

```
struct reconnect {
    entity_addrvec_t addr;
    uint64_t cookie;
    uint64_t global_seq;
    uint64_t connect_seq;
    uint64_t msg_seq;
};

struct reconnect_ok {
    uint64_t msg_seq;
};
```
4. MESSAGE EXCHANGE

```
struct message {
    __u8 tag;
    // includes last seen msg seq
    ceph_msg_header2 header;
    char payload[front_len + middle_len]
};

// TAGS
CLOSE       6 // closing pipe
MSG         7 // message
ACK         8 // message ack
KEEPALIVE2  14 // keepalive 2
KEEPALIVE2_ACK 15 // keepalive 2 reply
```
FRAME INTEGRITY, AUTHENTICITY, AND CONFIDENTIALITY
FRAME INTEGRITY, AUTHTENTICITY, AND CONFIDENTIALITY

- Integrity:
FRAME INTEGRITY, AUTHENTICITY, AND CONFIDENTIALITY

- Integrity:
  - CRC in frame header (length + tag)
FRAME INTEGRITY, AUTHENTICITY, AND CONFIDENTIALITY

- Integrity:
  - CRC in frame header (length + tag)
  - CRC in messages payload (same as in V1)
FRAME INTEGRITY, AUTHENTICITY, AND CONFIDENTIALITY

• Integrity:
  ■ CRC in frame header (length + tag)
  ■ CRC in messages payload (same as in V1)

• Authenticity and Confidentiality:
FRAME INTEGRITY, AUTHENTICITY, AND CONFIDENTIALITY

- Integrity:
  - CRC in frame header (length + tag)
  - CRC in messages payload (same as in V1)

- Authenticity and Confidentiality:
  - Frame payload only
FRAME INTEGRITY, AUTHENTICITY, AND CONFIDENTIALITY

• Integrity:
  - CRC in frame header (length + tag)
  - CRC in messages payload (same as in V1)

• Authenticity and Confidentiality:
  - Frame payload only
  - Authenticity with SHA256 HMAC
FRAME INTEGRITY, AUTHENTICITY, AND CONFIDENTIALITY

- Integrity:
  - CRC in frame header (length + tag)
  - CRC in messages payload (same as in V1)

- Authenticity and Confidentiality:
  - Frame payload only
  - Authenticity with SHA256 HMAC
  - Confidentiality with AES encryption
WHERE CAN I FIND THE CODE?
WHERE CAN I FIND THE CODE?

- Source code location:
  src/msg/async/ProtocolV2.cc
WHERE CAN I FIND THE CODE?

• Source code location:
  src/msg/async/ProtocolV2.cc

• Specification draft:
  http://docs.ceph.com/docs/master/dev/msg
FUTURE FEATURES
FUTURE FEATURES

- More authentication protocols: Kerberos, ...
FUTURE FEATURES

- More authentication protocols: Kerberos, ...
- Connection multiplexing
FUTURE FEATURES

- More authentication protocols: Kerberos, ...
- Connection multiplexing
- New ideas and contributions are welcome