MSGR2 PROTOCOL

This is a revision of the legacy Ceph on-wire protocol that was implemented by the SimpleMessenger. It addresses performance and security issues.

GOALS

This protocol revision has several goals relative to the original protocol:

- *Multiplexing*. We will have multiple server entities (e.g., multiple OSDs and clients) coexisting in the same process. We would like to share the transport connection (e.g., TCP socket) whenever possible.
- *Signing*. We will allow for traffic to be signed (but not necessarily encrypted).
- *Encryption*. We will incorporate encryption over the wire.
- *Flexible handshaking*. The original protocol did not have a sufficiently flexible protocol negotiation that allows for features that were not required.
- *Performance*. We would like to provide for protocol features (e.g., padding) that keep computation and memory copies out of the fast path where possible.

DEFINITIONS

- *client* (C): the party initiating a (TCP) connection
- *server* (S): the party accepting a (TCP) connection
- *connection*: an instance of a (TCP) connection between two processes.
- *entity*: a ceph entity instantiation, e.g. ‘osd.0’. each entity has one or more unique entity_addr_t’s by virtue of the ‘nonce’ field, which is typically a pid or random value.
- *stream*: an exchange, passed over a connection, between two unique entities. in the future multiple entities may coexist within the same process.
- *session*: a stateful session between two entities in which message exchange is ordered and lossless. A session might span multiple connections (and streams) if there is an interruption (TCP connection disconnect).
- *frame*: a discrete message sent between the peers. Each frame consists of a tag (type code), stream id, payload, and (if signing or encryption is enabled) some other fields. See below for the structure.
- *stream id*: a 32-bit value that uniquely identifies a stream within a given connection. the stream id implicitly instantiated when the send sends a frame using that id.
- *tag*: a single-byte type code associated with a frame. The tag determines the structure of the payload.

PHASES

A connection has two distinct phases:

1. banner
2. frame exchange for one or more strams

A stream has three distinct phases:

1. authentication
2. message flow handshake
3. message exchange

BANNER

Both the client and server, upon connecting, send a banner:

```
"ceph %x %x\n", protocol_features_suppored, protocol_features_required
```

The protocol features are a new, distinct namespace. Initially no features are defined or required, so this will be “ceph 0 0n”.

If the remote party advertises required features we don’t support, we can disconnect.

FRAME FORMAT

All further data sent or received is contained by a frame. Each frame has the form:

```
stream_id (le32)
frame_len (le32)
tag (TAG_* byte)
payload
[payload padding -- only present after stream auth phase]
[signature -- only present after stream auth phase]
```

- frame_len includes everything after the frame_len le32 up to the end of the frame (all payloads, signatures, and padding).
- The payload format and length is determined by the tag.
- The signature portion is only present in a given stream if the authentication phase has completed (TAG_AUTH_DONE has been sent) and signatures are enabled.

AUTHENTICATION

- TAG_AUTH_METHODS (server only): list authentication methods (none, cephx, ...):

  ```
  __le32 num_methods;
  __le32 methods[num_methods];   // CEPH_AUTH_{NONE, CEPHX}
  ```

- TAG_AUTH_SET_METHOD (client only): set auth method for this connection:

  ```
  __le32 method;
  ```

  - The selected auth method determines the sig_size and block_size in any subsequent messages (TAG_AUTH_DONE and non-auth messages).
- TAG_AUTH_BAD_METHOD (server only): reject client-selected auth method:

  ```
  __le32 method
  ```

- TAG_AUTH: client->server or server->client auth message:

  ```
  __le32 len;
  method specific payload
  ```

- TAG_AUTH_DONE:

  ```
  confounder (block_size bytes of random garbage)
  __le64 flags
    FLAG_ENCRYPTED  1
    FLAG_SIGNED     2
  signature
  ```

  - The client first says AUTH_DONE, and the server replies to acknowledge it.

MESSAGE FRAME FORMAT

The frame format is fixed (see above), but can take three different forms, depending on the AUTH_DONE flags:

- If neither FLAG_SIGNED or FLAG_ENCRYPTED is specified, things are simple:

  ```
  stream_id
  frame_len
  ```

```
tag
payload
payload_padding (out to auth block_size)
```

- If FLAG_SIGNED has been specified:

```
stream_id
frame_len
tag
payload
payload_padding (out to auth block_size)
signature (sig_size bytes)
```

  Here the padding just makes life easier for the signature. It can be random data to add additional confounder. Note also that the signature input must include some state from the session key and the previous message.

- If FLAG_ENCRYPTED has been specified:

```
stream_id
frame_len
{
  payload_sig_length
  payload
  payload_padding (out to auth block_size)
} ^ stream cipher
```

  Note that the padding ensures that the total frame is a multiple of the auth method's block_size so that the message can be sent out over the wire without waiting for the next frame in the stream.

MESSAGE FLOW HANDSHAKE

In this phase the peers identify each other and (if desired) reconnect to an established session.

- TAG_IDENT: identify ourselves:

```
entity_addrvec_t addr(s)
__u8   my type (CEPH_ENTITY_TYPE_*)
__le32 protocol version
__le64 features supported (CEPH_FEATURE_* bitmask)
__le64 features required (CEPH_FEATURE_* bitmask)
__le64 flags (CEPH_MSG_CONNECT_* bitmask)
__le64 cookie (a client identifier, assigned by the sender. unique on the sender.)
```

  - client will send first, server will reply with same.
- TAG_IDENT_MISSING_FEATURES (server only): complain about a TAG_IDENT with too few features:

```
__le64 features we require that peer didn't advertise
```

- TAG_IDENT_BAD_PROTOCOL (server only): complain about an old protocol version:

```
__le32 protocol_version (our protocol version)
```

- TAG_RECONNECT (client only): reconnect to an established session:

```
__le64 cookie
__le64 global_seq
__le64 connect_seq
__le64 msg_seq (the last msg seq received)
```

- TAG_RECONNECT_OK (server only): acknowledge a reconnect attempt:

```
__le64 msg_seq (last msg seq received)
```

- TAG_RECONNECT_RETRY_SESSION (server only): fail reconnect due to stale connect_seq
- TAG_RECONNECT_RETRY_GLOBAL (server only): fail reconnect due to stale global_seq
- TAG_RECONNECT_WAIT (server only): fail reconnect due to connect race.
    - Indicates that the server is already connecting to the client, and that direction should win the race. The client should wait for that connection to complete.

MESSAGE EXCHANGE

Once a session is stablished, we can exchange messages.

- TAG_MSG: a message:

```
ceph_msg_header2
front
middle
data
```

    - The ceph_msg_header is modified in ceph_msg_header2 to include an ack_seq. This avoids the need for a TAG_ACK message most of the time.
- TAG_ACK: acknowledge receipt of message(s):

```
__le64 seq
```

    - This is only used for stateful sessions.
- TAG_KEEPALIVE2: check for connection liveness:

```
ceph_timespec stamp
```

    - Time stamp is local to sender.
- TAG_KEEPALIVE2_ACK: reply to a keepalive2:

```
ceph_timestamp stamp
```

    - Time stamp is from the TAG_KEEPALIVE2 we are responding to.
- TAG_CLOSE: terminate a stream

  Indicates that a stream should be terminated. This is equivalent to a hangup or reset (i.e., should trigger ms_handle_reset). It isn't strictly necessary or useful if there is only a single stream as we could just disconnect the TCP connection, although one could certainly use it creatively (e.g., reset the stream state and retry an authentication handshake).