CHAPTER 5 OVERLAY MESSAGE FORMATS 1

5.1 Overview 1

5.2 Overlay Message 2

5.3 Message Extensions 3

5.3.1 Format of Payload Extension Header 5

5.3.2 Finite State Machine (FSM) Extension 5

5.3.3 Route Record Extension 8

5.3.4 Security Extension 9

5.3.5 Jumbo Payload Extension 11

5.4 Appendix A: Implementation Information 12

updated: August 2006, v5

This is an unfinished draft. If you have comments or corrections, please mark this document up and send it to jorg@comm.utoronto.ca. If you send your comments in plain text, please include the date (see upper left corner), the page number and the paragraph number. If you find discrepancies between this document and the most recent version of the HyperCast software, please give a detailed description of the problem.

Thank you,

Jörg Liebeherr

CHAPTER 5 Overlay Message Formats

5.1 OVERVIEW

Data between overlay sockets is exchanged as formatted messages. An overlay socket provides two endpoints for communications. The node adapter exchanges Protocol Messages with node adapters at other overlay sockets. The socket adapter exchanges overlay messages with socket adapters at other overlay sockets. The handling of protocol and overlay messages is completely separate. This chapter is about the format of overlay messages.

- **Protocol Messages**: These are messages that are exchanged between overlay nodes. Protocol messages are mostly sent to neighbors in the overlay network. Some overlay protocols may define protocol messages that are broadcast to all nodes in the overlay network (e.g., Beacon messages in the HC protocol). The format of protocol messages are specific to a given overlay protocol, and are completely defined by the overlay node).

- **Overlay Messages:** These are messages that are exchanged between socket adapters of overlay sockets that are neighbors in the overlay network, and are forwarded by the forwarding engine. All application data is transmitted as overlay messages. Overlay messages can be sent to all overlay sockets in an overlay network via flooding or multicast, or unicast to one specific overlay socket. In addition to transferring application data, overlay messages can be used for monitoring and control functions, e.g., to perform functions similar to traceroute or provide error reporting similar to ICMP.

The socket adapter of the overlay socket is responsible interfacing to the protocol used in the underlay network. The address(es) used by the socket adapter to access the underlay network are maintained at the overlay node as part of the physical address of the socket and the socket adapter. For the Internet as underlay network, socket adapters exist to transmit overlay messages using TCP, UDP unicast, or UDP multicast. It is emphasized once more that using non-Internet protocols or new versions of the Internet Protocol merely requires to upgrade the adapters.

The message formats of overlay messages are described in the sections, which contain the overlay protocols.

5.2 OVERLAY MESSAGE

An overlay messages have a header and a sequence of extensions. The header of an overlay message specifies, the version, the delivery mode, the source address, the logical address, etc. All addresses in the overlay message header are logical addresses.

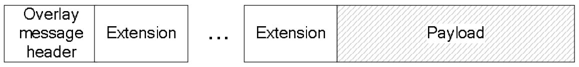

Figure 5.1. Types of Overlay Messages.

Figure 5.2 shows the structure of an overlay message. It consists of a common overlay message header followed by a sequence of extension headers. Application payload is part of an extension header. The concept of extension headers and the ability to concatenate multiple extension headers in a single overlay message follows the design of the IPv6 packet format.

All overlay messages have a header that is at least 14 bytes long. Multiple payloads can be concatenated by using extension headers, as indicated in the Next Header field.

Version *(4 bits)*	DMode *(4 bits)*	Traffic Class *(8 bits)*	Flow Label *(16 bits)*
Overlay Message Length *(16 bits)*			Hop Limit *(16 bits)*
Extension Type *(8 bits)*		LA Size *(8 bits)*	Source Logical Address *(variable)*
Previous Hop Logical Address *(variable)*			Destination Logical Address *(only if DMode=0x3, variable)*

Figure 22. Overlay message header.

Version (4 bits):	Version of the protocol. The current version is 0x3.
LA Size (8 bits):	Size of the logical address field.
DMode (4 bits)	Delivery mode. Currently defined delivery modes are: 0x1 Multicast 0x2 Flood 0x3 Unicast
Traffic Class (8 bits):	Specifies traffic class for service differentiation *(ignored in 3.0).*
Flow Label (16 bits)	Specifies flow for service differentiation *(ignored in 3.0).*
Overlay Message Length (16 bits)	Length of the overlay message, not including the header. If the field is set to zero, and the extension Type is set to Jumbo Payload, then the length of the message is contained in the Jumbo Payload extension header.
Extension Type (8 bits)	Specifies the type of the first extension following this header. The available extension types are explained in the next section.
Hop Limit (16 bits):	Number of logical links traversed before the message is dropped. This corresponds to Time-to-Live field in the IP (version 4) protocol.
Source Logical Address (*LA Size* bytes)	Logical address of the source of the overlay message.
Previous Hop Logical Address (*LA Size* bytes)	Logical address of the neighbor from which this message was received.

Note: In version 2.0 this field only existed for the Flood delivery mode. In 3.0 it is a required field.

Destination Logical Address (*LA Size* bytes)	Logical address of the destination of the overlay message. This field is present only in messages with delivery mode set to unicast. (Dmode = 0x2)

5.3 MESSAGE EXTENSIONS

Following the overlay message header is a set of extensions that carry control information or application data. There can be any number of extensions, but there are some restrictions regarding the occurrences and positions of some extensions. For example, there can be at most one extension with application data. The general structure of an extension is shown in Figure 4.

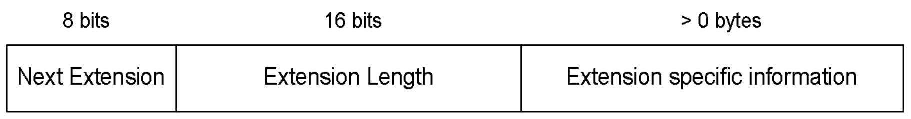

Figure 2. Structure of an extension in an overlay message.

The Next Extension field (8 bits long) specifies the type of the following extension. If there is no extension following, then the field is set to zero (0x00). The header length field (16 bits long) specifies the total length of the extension, including the Next Extension field. The remainder of the extension header is type specific.

With this organization, the type of the first extension header is determined by the next header field in the overlay message header. The type of the following extension headers is specified by the next header field of the previous extension headers. The next header field of the last extension field is set to zero.

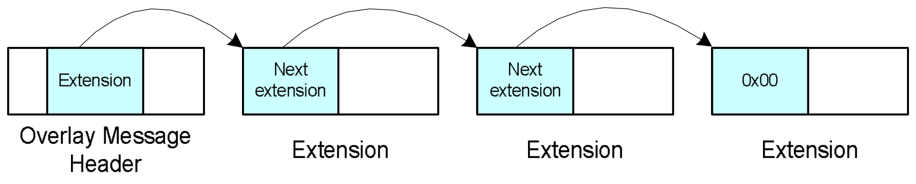

Figure 3. Organization of extensions in an overlay message.

Table 4 lists all defined values for the next header field of an overlay message. For each type, with exception of type 0x00, there is a format specification. The payload header contains application data. The finite state machine (FSM) header contains control information for providing enhanced delivery services for application data. The security extension headers are specified in the Chapter “Security Design”. Payload, Route Record, and security headers may appear only once in an overlay message.

(What happens if there is a security header but no payload? Then, there should be a digest on the message header.)

Table 2 is the list of possible values in the NextHeader field in the overlay message header:

Extension Types	**Description**
0x00	No Next Extension
0x01	Finite State Machine (FSM) Extension
0x02	Payload Extension
0x03	Route Record Extension
0x04	Jumbo Payload
0x21	Security Extension

Table 2. Extension Type Assignment.

<<Additionally needed messages: Control headers, source routing headers>>

5.3.1 FORMAT OF PAYLOAD EXTENSION HEADER

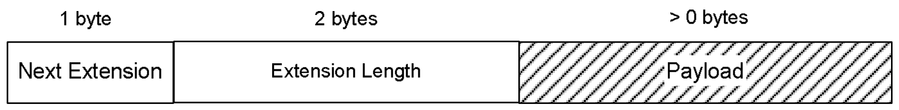

Figure 4. Payload extension.

In addition to the common fields, the payload extension header only contains application data. Some extension headers, e.g., FSM headers, can define control information for the payload. In principle, the payload header does not need to be in a particular position. In the HyperCast 3.0 implementation, the payload extension is the last extension header.

5.3.2 FINITE STATE MACHINE (FSM) EXTENSION

FSM headers carry control information for messages or message streams that are processed by a Finite State Machine (FSM) in the overlay socket, that enhances the best effort service. The extension contains a 2-byte long FSM Identifier, which specifies what type of service is implemented for the message or message stream associated with this extension. In addition to the FSM identifier, a type field defines the specific meaning for each type. Each FSM extension has the structure as shown in Figure 7. The details of the extension and the definitions of the files follow.

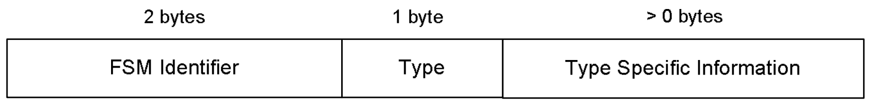

Figure 5.

The FSM Identifier (16 bits) identifies the type of finite state machine which processes a message or message stream. The finite state machines are implemented in the message store. Each finite state machine provides a certain service, which is defined by the FSM Identifier.

- 0x01: Hop-by-hop acknowledgement (H2HACK)
- 0x02: End-to-end acknowledgement (E2EACK)
- 0x03: Duplicate Elimination (DELDUPS)
- 0x04: Neighbor synchronization (SYNC)
- 0x05: Incast (INCAST)
- 0x06: Best-effort ordering (INORDER)
- 0x07: Naming (*See Documentation on Naming Service*)

A finite state machine may not support delivery of payloads with all delivery modes. If a finite state machine type does not support a delivery mode that is specified for a payload extension, the message may be dropped.

The Type field (8 bits) specifies a message type that is dependent on the FSM identifier. The following types are defined for all finite state machines:

- **Payload Data** (Type = 0x80): This message has an extension with application payload. Retransmission (Type = 0x81): This message has an extension with application payload that constitutes a retransmission.
- **Mergeable Payload** (Type = 0x82): This message has an extension with application payload and the payload is "mergeable" data.

Comment: How is this defined?

In addition to the payload, there may be numerous control messages for each finite state machine. Most of the time, an overlay socket sends such a control messages to one of its immediate neighbors. The control messages are defined in the following table.

Table 5.1. Type fields for control messages for various FSM identifiers.

Service FSM ID	**Definition and Interpretation of Type field**
0x00 No Service	None defined
0x01: **H2HACK** Hop-by-hop acknowledgement	0x01: H2HNACK Request for retransmission
	0x02: H2HACK Acknowledgement
	0x03: Request H2HACK Request for a H2HACK
	0x04: Local_Reset Resets state machine for this message
0x02: **E2EACK** End-to-end acknowledgement	0x05: H2HNACK Requests for Retransmission
	0x06: FULL_E2E_ACK Sends a full acknowledgement
	0x07: FINAL_PARTIAL_E2E_ACK Sends a partial acknowledgement.
	0x08: E2E_REQUEST Requests an E2EACK.
	0x09: E2E_RESET Resets a message.
0x03: **DELDUPS** Duplicate Elimination	Uses types with payload only.
0x04: **SYNC** Synchronization	0x0a: QUERY_SYNC_ALL Requests information about messages stored from a neighbor.

Comment: ????? Is this used?

	0x0b: QUERY_MSG_SYNC Requests State Information for a specific list of messages.
	0x0c: HAVEIT Sent in response to a QUERY_SYNC_ALL, QUERY_MSG_SYNC message or sent without any request.
	0x0d: DONT_HAVE_IT Sent in response to a QUERY_SYNC_ALL, QUERY_MSG_SYNC, or sent without any request.
	0x0e: NACK_SYNC Requests the transmission/retransmission of a single message.
0x05: **INCAST** Incast	0x0f: INCAST_REQUEST Request for an Incast transmission.
0x06: **INORDER** Best-effort ordering	Uses types with payload only.

Comment: This should no longer exist.

The finite state machines with identifiers FSM ID H2HACK, E2EACK, DELDUPS, SYNC, or INCAST implement services for a single individual application payload. These finite state machines are message-oriented. These extension headers contain a message identifier that is created at the source of the message when the application payload is sent for the first time in a payload extension of an overlay message.

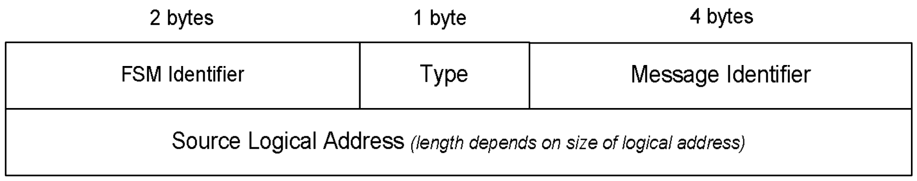
2 bytes
1 byte
4 bytes
FSM Identifier
Type
Message Identifier
Source Logical Address (length depends on size of logical address)

Figure 6. Format of an message-oriented FSM extension.

JL (7/30/04): The description of the message format for the FSM state machines is incomplete. It seems that the format needs to be documented.

```
                  1                   2                   3
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
```
Only if Type ==0x0B
```
+---------------------------------------------------------------+
|Digest Size|////////////////////Digest//////////////////////|
+---------------------------------------------------------------+
```

Figure 5.7. Message Header of ADF Message.

Comment: The format is not clear, even with the following information fro.
"The length of the message digest used in QUERY_SYNC_ALL message is 16
bytes long and the algorithm used can be refercenced to:
A. Rousskov and D. Wessels, "Cache Digest", In 3rd International WWW Caching Workshop, June 1998."
I have no information, if our code is actually from the paper, or if the reference is mentioned for citation purposes. It seems possible that Weisheng has provided the code. We may have to re-think and re-implement this part.

The message identifier (8 bytes) should uniquely identify a message among all messages currently processed by all overlay sockets of the overlay network. Uniqueness is not enforced and a finite state machine processes different payloads with identical message identifier as the same message. In our implementation, the message identifier is composed of a 4-byte hash over the physical address of the overlay socket and a 4-byte random number.

The source logical address is the logical address of the overlay socket which originally generated this message. This field is used for control data (such as acknowledgments) which are sent to the source of a payload message. The source logical address is needed to calculate the next hop (parent) in the tree with the source as root. The size of the address is known from the LAS field in the overlay message header. The source logical address field is not present in message types that contain payload (Type= 0x80, 0x81, 0x82), since these messages already carry the source logical address in the overlay message header.

The message format for messages with FSM ID INORDER are as follows:

FSM Identifier *(2 bytes)*	Type *(1 byte)*	Stream Identifier *(4 bytes)*
Sequence number *(8 bytes)*		

Figure 8. Figure 9. Format of a stream-oriented FSM extension.

Stream identifier (8 bytes): A number that identifies the stream.

Type (1 byte) The only type that is provided is 0x80 (Payload). There are no control messages for this FSM identifier.

Sequence number: A number that indicates the offset of the payload within a stream. The sequence number is set by the source of the message. The initial sequence number is 0. The sequence number for this FSM is incremented by one for each a message.

Comment: Checked with code. This should no longer be zero per discussion. The initial value should be a random number and the sequence number should be the modulo. We need to check if the sequence number is calculated correctly in overflow cases.

5.3.3 ROUTE RECORD EXTENSION

The route record extensions contains the list of logical addresses of those overlay sockets that this message has passed through. The route record message can be used to avoid forwarding loops in delivery of overlay message (Some overlay protocols may be susceptible to loops). An attributed in the configuration file (RouteRecordSize) determines if and how many logical addresses are recorded. For example, when the value of attribute RouteRecordSize is set to two each overlay message has a route record extension containing the logical addresses of the last two overlay sockets that this message has passed through. When the value is not set or set to zero or if the value is not set, then the route record extension is not used. When routes are recorded, then, whenever an overlay socket forwards a message, it appends its own logical address to the list. When the route record of an overlay message has reached the limit and a new logical address needs to be added, the address at the beginning of list is removed to make a room.

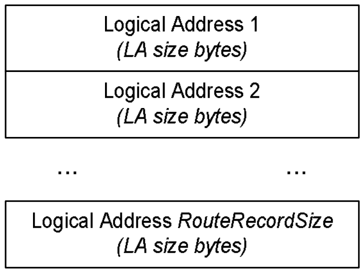

Figure 10. Format of Route Record Extension.

The Route Record is composed of a list of between one and *RouteRecordSize* logical addresses. The size of the logical address is determined by the LA Size field in the overlay message header.

5.3.4 SECURITY EXTENSION[i]

All overlay messages in an overlay socket with the security level set to protocol integrity, integrity, or confidentiality contain a security extension, also called *security header*. Overlay messages with a security extension header are also called secure overlay messages. The security extension is specified in the preceding header by a next header field with value 0x21. The security extension contains, among others, the MACs for the header and the payload and, if the neighborhood key method is enable, an encrypted message key. The format of a security header is shown in the Figure 12. The contents of the fields is as follows:

[i] This is duplicated in the Chapter on Security Architecture.

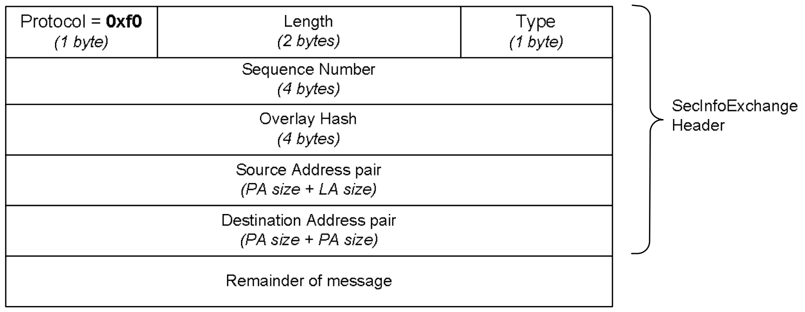

Figure 11. Format of the security extension.

Next Header (1 byte):
Specifies the type of extension following this header.

Length (4 bytes):
The length of the security header in bytes following the Length field, i.e., not include the Next Header and Length fields.

Sequence Number (4 bytes):
Specifies the sequence number of the message. The field is used in the same was as described for the *SecInfoExchange* header. The sender of a message increments the sequence number before each message transmission. With the neighborhood method, when the number of messages, both protocol and overlay messages, exceeds the maximum allowed number, then a new neighborhood key is generated and transmitted in a *KeyUpdate* message, and the sequence number is set to zero.

Encrypted Message Key (0-32 bytes):
Contains the encrypted message key. This field is not used when the group key method is executed, i.e., the length of the field is zero. The length of the encryption key is determined from the configuration attributes.

SPI (4 bytes):
The field SPI (Security Parameter Index) can contain a security association identifier, which is a random value identifying the security association for this message. The field is currently not used.

LA length (1 byte):
The length of the logical address of the overlay socket.

LA of Sender (variable):
The logical address of the neighbor that forwarded this message. The field is used to look up certificates and keys for the neighbor.

Header MAC Length (1 byte):
Length of the MAC for the header of the overlay message in bytes.

Header MAC (≥ 16 bytes):
Contains the MAC for the overlay headers. Precisely, the MAC is computed over the byte array of the entire message, with the Header MAC field and the payload field removed.

Payload MAC Length (1 byte):
Indicates the length of Payload MAC field in bytes.

Payload MAC (≥ 16 bytes):
Contains the MAC for the payload. If the payload is encrypted, this is the MAC of the encrypted payload. **(What happens to the field if there is no payload?)**

An overlay socket must compute the extension header in the following order: (1) Encrypt the payload field of the payload header; (2) Compute the payload MAC; and (3) Compute the Header MAC. When encryption is required, the payload MAC is computed over the encrypted payload field. The header MAC is computed over the entire message, with exception of the MAC header field and the payload field in the payload extension header.

With the group key method the group key is used for the payload encryption and the computation of both MACs. Here, the message key field is not used. With neighborhood keys, the message key is used for payload encryption and the payload MAC. The neighborhood key is used to encrypt the message key and to compute the header MAC.

For an incoming secure overlay message, an overlay socket verifies the sequence number and the header MAC. If either of these checks fails, the message is dropped without further processing. Otherwise, with the neighborhood key scheme, the message key is decrypted. When an overlay socket forwards a secure overlay message, it recomputes the header MAC and the sequence number, re-encrypts the message key in the security header (if present), and updates the LA Sender field with its own logical address. The header MAC is recomputed either with the group key or the neighborhood key of the local overlay socket. The encrypted message payload, the payload MAC, and the message key are not modified when the message is forwarded. The forwarding of messages with security headers can be viewed as an operation that replaces a security header.

When a secure message arrives at an overlay socket that is a destination of the message, a message is processed like any incoming message. Then the payload MAC is verified either with the group key or the message key. If not successful, the message is dropped. Otherwise, the payload is decrypted with the group key or the message, and delivered to the application. Incoming multicast and flood messages are delivered to the application, but may also be forwarded to other overlay sockets. Here, it is advisable that the message is forwarded before the payload is decrypted, otherwise the processing time for decrypting the payload at intermediate hops increases the latency of a packet.

When messages are transmitted with an enhanced delivery semantics (see Chapter *MessageStore*) they may be stored in the message store of the overlay socket, and may be forwarded at a later time, e.g., to retransmit a message when no acknowledgment has been received. When a message is stored in the message key, it is important that the decrypted message key is stored together with the message.

5.3.5 JUMBO PAYLOAD EXTENSION

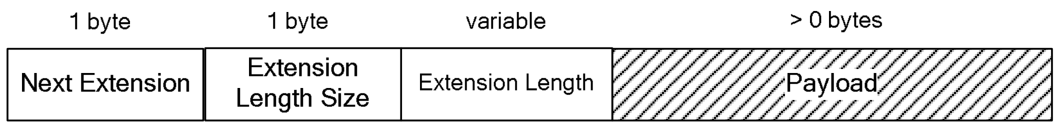

Figure 12. Jumbo Payload extension.

This extension is used for messages where the payload exceeds 2^16 -1.The operations are following the design of the IPv6 Jumbogram [RFC2675]. The Jumbo payload extension, if present, must be the first extension and must follow the overlay message header. If this extension is present, then the field of the overlay message length in the header must be zero.

Extension Length Size (1 byte) Number of bytes of the Extension Length field

Extension Length (variable): Length of the payload field in bytes.

5.4 APPENDIX A: IMPLEMENTATION INFORMATION

a. Security Processor

Extension is an abstract class. It defines the following methods that must be implemented by a concrete class, such as PayloadExtension, which extends class Extension.

byte getExtensionType() //return the type of the extension

byte[] toByteArray() //return the byte array converted from the extension

int getSize() //return the length of byte array converted from the extension

PayloadExtension also defines the following method for getting the payload.

byte[] getPayload() //return the payload byte array in this extension

All extensions are recognized as Extension instances in the OL_Message class. The OL_Message class is not aware of EncryptedPayloadExtension, and the encryption and decryption of the payload is hidden to the OL_Message and application. The process of encryption and decryption is discussed in Section 7.15.