Workgroup: Network Working Group
Internet-Draft:
draft-gao-flexible-session-protocol-08
Published: 18 April 2022
Intended Status: Experimental
Expires: 20 October 2022
Authors: J. Gao
Individual

Flexible Session Protocol

Abstract

FSP is a connection-oriented transport layer protocol that provides mobility and multihoming support by introducing the concept of 'upper layer thread ID', which is associated with some shared secret that is applied with some secure hash or authenticated encryption algorithm to protect authenticity of the origin of the FSP packets. It is able to provide following services to the upper layer application:

* Stream-oriented send-receive with native message boundary
* Flexibility to exploit authenticated encryption
* On-the-wire compression
* Light-weight session management

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The dominating transport layer protocols, TCP and UDP that take use of the IP address to identifying the end node, introduce the controversial role of IP address both as the identifier and as the routing locater.

Flexible Session Protocol, introduces the concept of 'upper layer thread ID' (ULTID), which was firstly suggested in [Gao2002]. The ULTID is unique in the local context of the FSP node. Its significance may be compared with the Security Parameter Index (SPI) in MOBIKE [RFC4555].

An integrity check code (ICC) field is designed in the FSP packet header to protect authenticity and optionally privacy of the FSP packet. The ICC field is designed to be associated with some shared secret indexed by the source or destination ULTID in the local context of the source or destination node, respectively. An FSP packet is assumed to originate from the same source if the ICC value associated the ULTID passes validation, regardless of the full source or destination address. In such a way FSP provides provides mobility, multi-homing and multi-path support.

ICC is either calculated by [CRC64] which protects FSP against unintended modification, or a cryptographic hash function, or cryptographically calculated with some Authenticated Encryption with Additional Data [R01] algorithm, each of which requires a shared secret key.

In the former case a weak key meant to obfuscate the CRC64 checksum is agreed by the FSP participants. In the latter two cases, the shared secret key is assumed to be installed by the upper layer application (ULA).

Either the weak key or the shared secret key is indexed by the source or destination ULTID in the local context of the sender or the receiver, respectively.

FSP facilitates secret key installation by introducing the concept of transmit transaction. Mechanism of transmit transaction also provides the session-connection synchronization service to the upper layer.

FSP is a transport layer protocol as specified in [RFC1122], provides services alike TCP [STD5] to ULA, with session layer features as suggested in [OSI_RM], most noticeably session-connection synchronization by introducing the concept of transmit transaction.

1.1. Features

1.1.1. Transport Layer Mobility Support

FSP is meant to be transport layer protocol that keeps the IP address as the routing locater but keeps it from being the key constituent of the end point identifier of FSP or any upper layer protocol built upon FSP.

1.1.2. Flexibility to exploit Authenticated Encryption

By optionally applying authenticated encryption with additional data on each FSP packet, FSP provides both authentication of message origin and privacy protection service to the upper layer application.

The shared key required by the authenticated encryption algorithm is supposed to be established and installed onto the FSP layer by the upper layer application (ULA), and by introducing the concept of transmit tranaction FSP provides mechanism for the connected ULAs to synchronize installation of the shared key. Thus FSP makes it flexible to test, try or apply new key establish algorithms.

1.1.3. Transmit Transaction

FSP introduces the concept of transmit transaction, which makes it able to mark end of message inherently at the transport layer, effectively eliminate the requirement of message delimiters at the application layer. Besides, it still provides byte-stream-oriented transfer service to its upper layer application which is familiar and friendly to application developers.

1.1.4. Light-weight session management

In addition to transmit transaction mechanism, which is arguably a feature of session-connection synchronization, the FSP upper layer application is able to fork branch connections of an established connection in zero round-trip mode without compromising security. In this way FSP provides light-weight session management service to ULA.

1.1.5. On-the-wire Compression

FSP provides on-the-wire compression service through the on-the-wire compression and fragmentation function module.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

1.3. Conventions

Conventions in describing the finite state machine (FSM) of FSP are:

ESTABLISHED	The string of alphabetic characters designates the name of the state
[API: Reset]	Upper Layer Application Programming Interface Call
{Notify}	Call back/notify the upper layer application
{new context}	Additional action before or after state transition
[Send OPCODE_OF_FSP_PACKET]	Send some FSP packet
[Retransmit OPCODE_OF_FSP_PACKET]	Retransmit some FSP packet
{On transient state Timeout}	Timed-out event
{&& additional condition}	Together with some additional condition
-->	state transition
|--	branch

2. Abbreviations and Idioms

*Connection An FSP connection is the binding of two network nodes established through some end-to-end negotiation process. It is identified by the ULTID in the local context of each network node, respectively.

*EoT End of Transaction. A transmit transaction is said to reach EoT if the EoT flag is set in a legitimate PURE_DATA, PERSIST, NULCOMMIT or MULTIPLY packet. We said that the packet terminates the transmit transaction if the EoT flag is set. An FSP end node SHALL NOT send further data if it has initiated EoT of its transmit direction unless a particular ACK_FLUSH packet is received. The particular ACK_FLUSH packet MUST acknowledge not only the packet with the EoT flag set but all of the packets sent before the packet as well. EoT, i.e. termination of transmit transaction is unilateral.

*FREWS

*The Flags and advertised REceive Window Size. It is the 32-bit combined word next to the ICC field in the normal FSP fixed header.

*ICC The Identity Check Code. It is a 64-bit value that depends on both the session key and all of the headers of the FSP packet to include the ICC, calculated with the same algorithm in the context of each FSP participant. Only a packet with correct ICC can be accepted by any FSP participant as soon as the connection has been established. Initially CRC64 is exploited to make a checksum that weekly protects the FSP packet against unintentional modification. The checksum is obfuscated with the initial session key to get the ICC. After the ULA installed the long-term session key either some cryptographic hash function or some Authenticated Encryption with Additional Data algorithm shall be applied to obtain or check the ICC.

*Session An FSP session is the transport-layer association of two network nodes. A full FSP session consists of one connection that was established from scratch and all of its branches. However for this version of FSP specification the idioms session and connection are interchangeable if not explicitly specified.

*Session Key The session key is a bit string of at least 128 bits that means to resist against masquerade attack. It is either initiated during the end-to-end negotiation phase or installed by the ULA after the FSP connection is established. The session key installed by the ULA is called the long-term session key. Here long-term means that the key could be used until the packet sequence space is exhausted. The packet sequence space is exhausted if the number of packets that use the same key reaches or exceeds 2,147,483,647(2^31-1).

*SN The Sequence Number. It is the unsigned 32-bit integer number assigned to every FSP packet except the preliminary packets. Difference of two sequence number is represented by a 32-bit signed integer. If the result of SN B subtracting SN A is greater than zero, we say that B is greater than A and the packet of the sequence number B is later than the packet of the sequence number A, although the unsigned integer representation of B may be far less that A. Consequently, as the result of A subtracting B is less than zero, we say that A is less than B and the packet of the sequence number A is earlier than the packet of the sequence number A.

*Transmit Transaction A transmit transaction in FSP is a sequence of FSP packets that were sent and marked by the ULA as one continuous stream where all packets in the stream must be acknowledged before any further packet is allowed to be sent. A ACK_CONNECT_REQ, PERSIST or MULTIPLY packet always starts a new transmit transaction. Any packet with EoT flag set, including the NULCOMMIT packet which is payload-less, terminates the transmit transaction. The NULCOMMIT both starts and terminates a payload-less transmit transaction when it is exploited to acknowledge the ACK_CONNECT_REQ or MULTIPLY packet.

*ULA The Upper Layer Application.

*ULTID The Upper Layer Thread Identifier (ULTID). It is a 32-bit word that was allocated by particular network end node of an FSP connection and is unique in the local context of the network end node. Theoretically all of the ULAs of a network end node MAY establish up to 2^31-1 FSP connections totally. Each connection MUST have a unique thread identifier (ULTID) assigned in the local context of the network end node. A session or connection in FSP does not require a global ID.

3. Key Mechanisms

3.1. Underlying Layer Services Required

FSP requires that the underlying layer provides packet delivery service. In addition FSP supposes that the underlying layer has following features:

3.1.1. Handling of High Mobility

Here high mobility refers to scenarios such as high-speed train or airplane.

FSP solves somewhat coarse-grain or low-speed mobility problem. Fine-grain or high-speed mobility is left to the underlying physical network. To make mobility support work effectively it is assumed that one end-node MUST keep its lower layer address reasonably stable while the other end-node SHOULD NOT change its lower layer address too frequently.

Here the meaning of physical network conforms to what was depicted in [RFC1122].

3.1.2. Network Address Change Notification

Network address change notification is mandatory only in the IPv6 network.

We split the IPv6 address of the IPv6 packet underlying FSP into three parts. The leftmost 64-bit long word is the network prefix, which SHOULD be the unique IPv6 prefix assigned to the host [RFC8273]. The centermost 32-bit word is called the aggregation host ID, and the rightmost 32-bit word is the ULTID. While the ULTID MUST be kept stable even during the life of an FSP session, the network prefix part MAY change when an endpoint is roaming. The aggregation host ID may change as well. The network prefix part together with the aggregation host ID part act as the traditional routing locator at the network layer.

The aggregation host ID SHOULD NOT be 0, nor hexadecimal value FDFFFFFFF, nor hexadecimal value FFFFFFFFF so that the final IPv6 address assigned can not be an IPv6 subnet anycast address[RFC2526].

It is supposed that the network layer immediately notifies the FSP layer if the network prefix or aggregation host ID changes.

An participant of an FSP connection SHALL immediately notify its peer whenever its underlying IPv6 address is changed with a KEEP_ALIVE packet. The peer shall send packet to the participant that has notified the address change with the new address.

It is supposed that the packet to inform the remote end to update the lower layer address association could reach its destination in a satisfying success rate.

3.1.3. Network Congestion Control

Network layer congestion control is mandatory only in future IPv6 network.

It is supposed that end-to-end congestion control is provided at some sub-layer of the network layer. Implementation of FSP MUST include a congestion manager [RFC3124] if such sub-layer service of the network layer is not provided.

3.2. Identifying Connection by Local ULTID

Each FSP connection prepares a pair of ULTIDs. An ULTID uniquely indexes a connection in the local context of an FSP end node. An FSP end node relies neither source IP address nor destination IP address, except the ULTID part of the near end's IPv6 address to identify an FSP connection.

Each ULTID is allocated in the local context of the two FSP participant respectively. The source ULTID and the destination ULTID of an FSP packet usually differ in their values. However, the secret keys indexed in the local contexts of the two end-points must have the same value.

3.3. Defending Against Connection Hijacking with ICC

An integrity check code (ICC) field associated with the ULTID is designed in the FSP header to protect authenticity and optionally privacy of the FSP packet. An FSP packet is assumed to originate from the same source if the ICC value associated with certain destination ULTID passes validation, regardless of the source or destination address in the underlying layer.

On initiating FSP takes use of [CRC64] to make checksum of the FSP packet to protect it against unintentional modification. The checksum is taken as the ICC.

After the ULA has installed a shared secret key, value of ICC is calculated by firstly getting the secret key associated indexed by the local ULTID, then calculating the tag value with the AES-GCM [GCM] authenticated encryption with additional data algorithm , or calculating the message authentication code with the BLAKE2 algorithm [RFC7693].

3.4. Synchronizing Key Installation with Transmit Transaction

It is proposed that it is the ULA to do key establishment and/or end- point user-agent authentication while the FSP layer provides authenticated, optionally encrypted data transfer service.

A dedicate application program interface (API) is designed for the ULA to install the secret key established by the ULA participants.

FSP facilitates shared secret key installation by introducing the concept of transmit transaction.

A transmit transaction of FSP is a sequence of FSP packets that were sent and requested by ULA as one continuous stream where all packets in the stream MUST be acknowledged before any further packet is allowed to be sent.

A flag called 'End of Transaction' (EoT) is designed in the FSP header. When it is set, it marks that the transmit transaction in the direction from the source of the FSP packet towards the destination of the FSP packet is 'committed'.

As the FSP node knows when a transmit transaction at FSP layer is terminated there is no ambiguity which packet is to be applied with the new session key when the key is installed by ULA through the dedicated API.

3.5. 0-RTT Connection Multiplication with OOB Packet

An FSP connection MAY be multiplied to get a branch or branches of the connection. Multiplication of a connection is protected by deriving new session key from the original security context of the connection.

The packet that carries the command to multiply an established FSP connection MUST be sent from a new allocated local ULTID towards the destination ULTID of the original connection. It is an out-of-band (OOB) packet in the context of the original connection and it MUST be cryptographically protected by the secret key of the original connection. The packet MAY carry payload.

The receiver of the packet MUST allocate a new local ULTID, accept the optional payload in the new context associated with the new ULTID, derive a new secret key from the secret key of the original connection, and responds from the new context. The response MAY carry payload.

The very first response packet MUST be protected by the new secret key. The sender of the multiply command packet MUST automatically inaugurate the same secret key, derived from the secret key of the same original connection. And it MUST treat the response packet as though a transmit transaction had been committed by the responder, i.e. authenticity of the response packet is verified with the new secret key.

Thus the branch connection of a new pair of ULTIDs is established with zero round-trip overhead. This mechanism may be exploited to provide expedited data transfer or parallel data transfer service.

4. Packet Structure

4.1. FSP over UDP/IPv4

In this version of FSP, when FSP is implemented in the IPv4 network, every FSP packet MUST be encapsulated in a UDP [STD6] datagram. The UDP datagram encapsulated the FSP packet SHALL have the checksum disabled (i.e. set its value to 0) [RFC8085]. The Source and the destination ULTIDs are put at the leading position of the UDP payload. FSP fixed header, optional extension headers and FSP payload follow the ULTIDs:

```
 0                             15 16                            31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Source Port           |        Destination Port       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Length             |              0                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Upper Layer Thread ID                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Upper Layer Thread ID               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                          FSP Fix Header                       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                Optional FSP Extension Headers                 ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
~                       Optional  FSP payload                   ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.2. FSP over IPv6

When FSP is implemented over IPv6 [RFC8200], the ULTID part is embedded in the IPv6 address. FSP fixed header follows the IPv6 headers:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~                          IPv6 Header:                          ~
 0                             15 16                            31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                |
+                                        Source Network Prefix  +
|                                                                |
+                 Source Address    -----------------------------+
|                                   Source Aggregation Host ID  |
+                                   -----------------------------+
|                                 Source Upper Layer Thread ID  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                |
+                                   Destination Network Prefix  +
|                                                                |
+        Destination Address   ----------------------------------+
|                              Destination Aggregation Host ID  |
+                              ----------------------------------+
|                            Destination Upper Layer Thread ID  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                                ~
~                    Optional IPv6 Headers                       ~
~                                                                ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                |
~                          FSP Fix Header                        ~
|                                                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                Optional FSP Extension Headers                  ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                                ~
~                       Optional  FSP payload                    ~
~                                                                ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

*Network Prefix The leftmost 64-bit of the IPv6 address which MAY and usually do have different value at the difference interface of an IPv6 end- node.

*Aggregation Host ID The left 32-bit part of the rightmost 64-bit long word of the IPv6 address. All of the aggregation host ID parts of an IPv6 end- node's IPv6 addresses MUST have the same value for this version of FSP.

Network prefix and aggregation host ID make the IPv6 prefix part of the IPv6 address under the context of FSP. It is assumed that there is a unique IPv6 prefix per host [RFC8273].

RFCs related to IPv6 address configuration such as ND for IPv6 [RFC4861], SLAAC [RFC4862], 'IPv6 Subnet Model' [RFC5942], 'IPv6 Address Assignment to End Sites' [RFC6177], 'Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis' [RFC7050], DHCP [RFC8415] and 'IPv6 Node Requirements' [RFC8504] would be reviewed in separate documents.

4.3. Generic FSP Header

FSP headers include the fixed header and the extension headers. A general fixed header consists of a 32-bit FSP Header Signature and 20-byte operation-code specific fields. An extension header consists of a 32-bit Code-Mark-Length Prefix and the operation-code specific content. The length of the extension header content may be variable, provided that the tail of the full extension header align on 64-bit boundary.

Integers in the FSP fix header are transmitted in network byte order. Otherwise, integers in the FSP headers are of little-endian.

```
  0                                                             31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Header Signature                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
~              Operation Code Specific Fields                   ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.4. FSP Header Signature and Code-Mark-Length Prefix

```
 0             7 8            15 16                           31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Operation Code|     Major     |             Offset            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 0             7 8            15 16                           31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Operation Code|     Mark      |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.4.1. Operation Code

The operation code is an octet that stores the code of the command which indicates the function of the FSP packet, or specifies the type of some extension header.

Synonym	Code	Meaning
INIT_CONNECT	1	Initialize Connection
ACK_INIT_CONNECT	2	Acknowledge Initialization of Connection
CONNECT_REQUEST	3	Formally Request to Connect
ACK_CONNECT_REQ	4	Acknowledge the Connection Request
RESET	5	Reset a connection. Abort the connection, or refuse to establish the connection at the very beginning.
NULCOMMIT	6	Commit without payload. The NULCOMMIT packet MUST be payload- less, and its EoT flag MUST be set. It MAY carry optional extension headers. It always consumes a slot of the send sequence space. It is to commit current transmit transaction, otherwise the NULCOMMIT packet itself SHALL just be skipped when delivering data to ULA.
KEEP_ALIVE	7	Keep the peer alive. It is an out-of-band control packet acting as the heart-beating signal. An out-of-band control packet does not consume send sequence space itself. FSP takes use of the KEEP_ALIVE packet to inform the peer about the change of the source IP addresses. Besides, when the MIND flag is set, the KEEP_ALIVE packet is meant to tell the peer which packets should be retransmitted.
PERSIST	8	Make a connection persistent. It is meant to start a new transmit transaction after a connection migrated to CLOSABLE state. It can also acknowledge ACK_CONNECT_REQ or MULTIPLY. It MUST carry payload. It always consumes a slot of the send sequence space.
PURE_DATA	9	Pure Data. It does not carry any optional header.
ACK_FLUSH	10	ACKnowledge to remote end's commitment (FLUSHing) of transmit transaction. It is an out-of-band control packet like KEEP_ALIVE. It is sent instantly on having every packet of the last transmit

Synonym	Code	Meaning
		transaction received, meant to make acknowledgment to the remote end and let the remote end stop sending heart-beat signals.
RELEASE	11	Release the connection. RELEASE packet does not carry payload, but it always consumes a slot of the send sequence space.
MULTIPLY	12	Multiply the connection. It is sent in the context of the original connection and may carry payload and/or optional headers as an out-of-band packet.
PEER_SUBNETS	17	Tell the remote end how to address the sender of the packet in the reverse direction. It is the code of the Sink Parameter extension header.
SELECTIVE_NACK	18	Tell the remote end to retransmit the packets that were negatively acknowledged. It is the code of the Selective Negative Acknowledgment extension header.

Table 1

4.4.2. Major

It is an octet that states current FSP major version. For this FSP version it MUST be 0.

It is not mandatory for different major versions of FSP to be compatible.

4.4.3. Mark

It is an octet that marks current minor version of the extension header, under the major FSP version specified in the fixed header, and/or serves as the flag, depending on the type of the extension header.

It is mandatory for implementations to be compatible with different minor versions of FSP extension header under the same major FSP version.

It is not mandatory for minor versions of FSP extension header under the different major FSP version to be compatible, even if the minor versions happen to be the same.

4.4.4. Offset

The offset field is a 16-bit unsigned integer that specifies the offset of the first octet of the payload, starting from the begin of the FSP fixed header. If its value equals the size of the fixed header the packet has no extension.

4.4.5. Length

The length field is a 16-bit unsigned integer that specifies the length, including the Type-Version-Length Header and the size of the the operation code specific content of the extension header.

4.5. Preliminary FSP Packets

Preliminary FSP packets are the packets exchanged during the end-to-end negotiation phase of FSP connection establishment when it is impossible to calculate ICC normally.

4.5.1. Connect Initialization

```
 0                                                             31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Header Signature                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              Salt                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
 +                          Timestamp                           +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                       Init-Check-Code                         +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
~           Host Name of the Responder (optional)               ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Operation Code of this type of packet is INIT_CONNECT.

*Salt It a 32-bit random bit string that may be exploited to make secret key agreement.

*Timestamp It is a 64-bit unsigned integer that represents number of microseconds elapsed since 00:00, Jan.1, 1970, Coordinated Universal Time. It may be exploited to synchronize the clocks of the participants and/or estimate delay during data transmission in the network.

*Init-Check-Code It is a 64-bit random [RFC4086] bit string that
 means to uniquely associated with the connection initiated.

*Host Name of the Responder The optional payload of the Connect
 Initialization packet is the host name of the responder. It MUST
 be encoded in UTF-8 [RFC3629] and SHOULD be resolvable in DNS,

4.5.2. Acknowledgment to Connect Initialization

```
 0                                                              31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Header Signature                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Time Delta                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                            Cookie                             +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                       Init-Check-Code                         +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
~                         Sink Parameter                        ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Operation Code of this type of packet is ACK_INIT_CONNECT.

*Time Delta It is a 32-bit signed integer which is the difference
 between the near-end's time and the timestamp value sent in the
 Connection Initialization packet. The units and the epoch of the
 near-end's time value and the timestamp value MUST be the same.
 However, the precision or resolution of the time delta value is
 chosen arbitrarily by the responder.

*Cookie It is a 64-bit bit string cryptographically generated by
 the responder in a represent-transfer state manner. More
 specifically when the same timestamp, time delta, Init-Check-
 Code, salt, source and destination ULTIDs are sent to the
 responder, the responder MUST be able to generate the identical
 cookie value.

*Init-Check-Code It MUST be identical to the corresponding field
 in the Connect Initialization packet acknowledged.

o Sink Parameter
It is an extension header specified in 4.7.

4.5.3. Connect Request

```
 0                                                              31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Header Signature                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              Salt                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                         Time Stamp                            +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                       Init-Check-Code                         +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Initial Sequence Number                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Time Delta                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                            Cookie                             +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
~                         Sink Parameter                        ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
~           Host Name of the Initiator (optional)               ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Operation Code of this type of packet is CONNECT_REQUEST.

The value of each field that has the identical name with the one in the associated Connect Initialization and Acknowledgment to Connect Initialization packet MUST be assigned the same value as in these two packets, except header signature in the packet.

Note that the initiator SHALL fill in the time delta field as it is and SHALL NOT assume endian-ness of the time delta field.

*Sink Parameter It is an extension header specified in 4.7.

*Host Name of the Initiator The optional payload of the Connect Request packet is the host name, which is encoded in UTF-8 and SHOULD be resolvable in DNS, of the initiator. It could be

      exploited by the responder to look up the address of the
      initiator that may receive packets in the reverse direction.

4.6. Normal Fixed Header

```
 0                             15 16                            31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Header Signature                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Flags     |         Advertised Receive Window Size       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Sequence Number                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Expected Sequence Number/Out-of-Band Serial Number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                     Integrity Check Code                      +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Operation Code of a normal fixed header may be ACK_CONNECT_REQ, PURE_DATA, PERSIST, KEEP_ALIVE, ACK_FLUSH, RELEASE or MULTIPLY.

*Flags It is bit-field of width 8. From left to right:

-End of Transaction (EoT): If the EoT flag of a packet is set, it is the last packet of a transmit transaction. A packet with the EoT flag set MAY be the start and the single packet of the transmit transaction as well.

-Minimal-Delay (MIND): If the MIND flag of the Connect Request or Acknowledgment to Connect Request packet is set, the ULA prefers minimal delay and is willing to tolerate packet loss. FSP SHALL drop the packet received earliest when there is no enough receive buffer so that the latest packet received can be saved and the delay to deliver data to ULA is minimized. If the MIND flag has been set the EoT flag of any following packet is simply ignored. The MIND flag of an FSP packet may be set only if the packet does not belong to a compressed transmit transaction. Payload of each FSP packet is delivered to the ULA as an independent message if the MIND flag has been set.

-Compressed (CPR) If the CPR flag of the first packet of a transmit transaction is set, compression is applied on the octet stream of the payloads of the continuous packets that constitute the transmit transaction, or to put it simply, the payload octet stream of the transaction transaction. Such transmit transaction is called a compressed transmit transaction. When the CPR flag of a packet is set, CPR flag of each following packet is ignored until reaching termination of the transmit transaction. If the payload octet stream of the transmit transaction is compressed the Minimal-Delay flag of any packet in the transmit transaction MUST be cleared.

*ECN-Echo (ECE): The Explicit Congestion Notification Echo flag of a packet is set if the sender of the packet has received an underlying IPv6 packet with Congestion Experienced code point set for its peer as stated in "The Addition of Explicit Congestion Notification (ECN) to IP" [RFC3168].

*Send Rate Reduced (SRR): The SRR flag of each packet SHALL be set as soon as the sender has received a legitimate packet with ECE flag set and has informed the congestion manager to reduce the send rate, until a sent packet with SRR flag set is acknowledged.

*The remaining 3 bits are reserved.

Advertised Receive Window Size It is a 20-bit unsigned integer that stores number of the free blocks in the receive buffer of the sender of the packet that contains the receive window size field. It is count from the slot meant to accept the packet with the expected sequence number. The sender must ensure that the difference between the latest sequence number sent out and the largest expected sequence number received does not exceed the value of the latest advertised receive window size received.

Sequence Number Each in-band FSP packet is assigned a 32-bit unsigned integer as the sequence number. The sequence number assigned for in-band FSP packets MUST be in strict order. An out-of-band packet that has the operation code of KEEP_ALIVE, ACK_FLUSH or MULTIPLY MUST be assigned a sequence number that falls in the receive window.

Expected Sequence Number The 'expected sequence number/out-of-band serial number' field stores the earliest sequence number of the packets that were not yet received in the receive window of the sender for in-band packet. It is an accumulative acknowledgment. Any packet with the sequence number before the received Expected Sequence Number is supposed to have been received by the remote end.

Out-of-band Serial Number The 'expected sequence number/out-of-band serial number' field stores a 32-bit unsigned integer that numbers the out-of-band FSP packet separately from the sequence space of the in-band packets. Each time an out-of-band packet is sent the out-of-band serial number SHALL increase by one. It is assumed that in the life of the session no two packets have the same sequence number and the same out-of-band serial number simultaneously.

Integrity Check Code The ICC.

4.7. Sink Parameter

```
 0                                                             31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Code-Mark-Length Prefix                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Listener ID/Host ID                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
~               Addressable Network Prefixes                    ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Operation Code in the Header Signature of this extension header is PEER_SUBNETS.

* Listener ID It is the ULTID of the responder that is in LISTENING state.

* Host ID It is the aggregation host id of the sender. It SHALL be 0 if it is in the IPv4 network.

* Addressable Network Prefixes These are up to 4 64-bit words that specify the network prefixes of the lower layer interfaces that are addressable by the receiver in the reverse direction. In this version of the FSP 'Addressable Network Prefixes' field is of fixed length. The last network prefix which is non-zero is the last resort one. There MUST be at least one non-zero network prefix. If there are more than one non-zero network prefixes those other than the last resort are load-balanced preferred. In an IPv6 network, the addressable network prefix is the leftmost 64 bits of the IPv6 address. The receiver of the Addressable Network Prefixes SHALL send packet in the reverse direction, i.e. to the sender of the field with the destination IPv6 address generated by combining a preferred network prefix with the aggregation host id and the ULTID part of the source address in the IPv6 header of the received packet that eventually carries the Addressable Network Prefixes. Such feature MAY be exploited to handle links with unidirectional connectivity, but it is NOT RECOMMENTED. In an IPv4 network for compatibility with the IPv6 addressed ULA the 64-bit word of the addressable network prefix specified is composed as following Figure:

```
 0                            15 16                            31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  0x2002 (IPv6 6to4 prefix)    |IPv4 address (leftmost 16 bits)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|IPv4 address(rightmost 16 bits)|   UDP port number (16 bits)   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Sender of the Sink Parameter packet SHOULD be NAT-aware in the IPv4 network. If it is able to obtain the from the NAT box (as defined in "Traditional IP Network Address Translator (Traditional NAT)" [RFC3022]) through Port Control Protocol (PCP) [RFC6887] SHOULD fill in the IPv4 address and UDP port number fields with the public IP value that were obtained. If it does not have such capability, it SHALL fill in the addressable network prefix with all binary zeroes.

4.8. Selective Negative Acknowledgment

```
 0                                                             31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Code-Mark-Length Prefix                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Expected Sequence Number                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Delay Sample Sequence Number                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Acknowledgement Delay                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Gap Width                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Data Length                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
~          Further pairs of  (Gap Width, Data Length)           ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The operation code of this type of extension header is SNACK.block.

*Expected Sequence Number It stores the earliest sequence number of the packets that were not yet received in the receive window of the sender. It is an accumulative acknowledgment. Any packet with the sequence number before the received Expected Sequence Number is supposed to have been received by the remote end. The expected sequence number is represented in little endian.

*Delay Sample Sequence Number It stores the sequence number of the packet that the acknowledgement delay was calculated on. The delay sample sequence number is represented in little endian.

* Acknowledgement Delay A 32-bit unsigned integer specifies the delay in microseconds between sending the packet containing the SNACK extension header and accepting the last packet that is accumulatively acknowledged by the SNACK extension header. The acknowledgement delay is represented in little endian.

* Gap Width and Data Length The SNACK header contains the descriptor of the receive window gaps. The descriptor itself is a list of entries. The length of the list can be zero which means that there is no gap in the receive window. If the list is not empty, each entry contains the width of one gap (Gap Width) in the receive window and the length of the continuously received data following the gap (Data Length), respectively. The unit of aforementioned length of gaps or number of packets is buffer block which size is agreed upon connection establishment. Gap width and data length MUST be transmitted in little endian.

4.9. RESET

The 'RESET' packet is a special command packet meant to interrupt connection setup process or disconnect abruptly. Operation Code of the packet is RESET.

Structure of a RESET packet in C code snippet with unnamed union applied:

```
struct FSP$HeaderSignature
{
    octet          opCode;
    octet    major;
    uint16_t offset;
};

struct FSP_RejectConnect
{
     FSP$HeaderSignature hs;
     /* bit field to describe reasons for reset */
     uint32_t reasons;
     /* sequence numbers */
     union
     {
             timestamp_t timeStamp;
             struct
             {
                     uint32_t initial;
                     uint32_t expected;
             } sn;
     };
     /* uniqueness proof */
     union
     {
             uint64_t integrityCode;
             uint64_t cookie;
             uint64_t initCheckCode;
     };
};
```

When the RESET packet is the response to a Connect Initialization packet both the timeStamp and the initCheckCode fields of the RESET packet MUST be set to the same values of Time Stamp and Init-Check-Code in the Connect Initialization packet, respectively.

When the RESET packet is the response to a Connect Request packet both the timeStamp and the cookie fields of the RESET packet MUST be set to the same value of Time Stamp and Cookie in the Connect Request packet, respectively.

When the RESET packet is the response to a packet with a normal fixed header, the sn.initial, the sn.expected and the integrityCode of the RESET packet MUST be set as to specification of normal fixed header field Sequence Number, Expected Sequence Number and Integrity Check Code, respectively.

5. The Finite State Machine

5.1. NON_EXISTENT

```
---[API: Listen]-->LISTENING
|--[API: Connect]-->CONNECT_BOOTSTRAP-->[Send INIT_CONNECT]
|--[API: Multiply]-->CLONING-->[Send MULTIPLY]
```

NON_EXISTENT is a pseudo-state before a connection is created by the ULA calling API Listen, Connect or Multiply (or after a connection is reset).

5.2. LISTENING

```
---[API: Reset]-->NON_EXISTENT
|<-->[Rcv.INIT_CONNECT]{&& accepted}[Send ACK_INIT_CONNECT]
|<-->[Rcv.CONNECT_REQUEST]
    |--{&& duplication detected}--[Retransmit ACK_CONNECT_REQ]
    |--{&& no duplication}-->{Notify}
      |-->[API: Accept]
          -->{new context}CHALLENGING-->[Send ACK_CONNECT_REQ]
      |-->[API: Reject]-->[Send RESET]{abort new context, if any}
```

LISTENING is a state entered by the ULA calling API Listen.

5.3. CONNECT_BOOTSTRAP

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|-->[Rcv.ACK_INIT_CONNECT]-->CONNECT_AFFIRMING
    -->[Send CONNECT_REQUEST]
|--{On transient state Timeout}-->NON_EXISTENT-->[Notify]
```

CONNECT_BOOTSTRAP is a state entered by the ULA calling API Connect, before receiving the acknowledgement of the remote end to the connection initialization packet.

5.4. CONNECT_AFFIRMING

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|--[Rcv.ACK_CONNECT_REQ]-->[Notify]
   |-->{Callback return to accept}
      |-->{EOT}
         |-->{No Payload to Send}-->COMMITTING2-->[Send NULCOMMIT]
         |-->{Has Payload to Send}
            |-->{ULA-flushing}-->COMMITTING2
                -->[Send PERSIST with EoT]
            |-->{Not ULA-flushing}-->PEER_COMMIT-->[Send PERSIST]
      |-->{Not EOT}
         |-->{No Payload to Send}-->COMMITTING-->[Send NULCOMMIT]
         |-->{Has Payload to Send}
            |-->{ULA-flushing}-->COMMITTING
                -->[Send PERSIST with EoT]
            |-->{Not ULA-flushing}-->ESTABLISHED-->[Send PERSIST]
   |-->{Callback return to reject]-->NON_EXISTENT-->[Send RESET]
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
|--{On transient state Timeout}-->NON_EXISTENT-->[Notify]
```

CONNECT_AFFIRMING is a state entered by the ULA affirming to send connect request after receiving the acknowledgement to the connection initialization packet.

5.5. CHALLENGING

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|<-->[API: Send{new data}]{just pre-buffer}
|--[Rcv.NULCOMMIT]
   |-->{ULA-flushing}-->CLOSABLE-->[Notify]
       -->[Send ACK_FLUSH]
   |-->{Not ULA-flushing}-->PEER_COMMIT-->[Notify]
       -->[Send ACK_FLUSH]
|--[Rcv.PERSIST]
   |-->{EOT}
      |-->{ULA-flushing}-->CLOSABLE-->[Notify]
          -->[Send ACK_FLUSH]
      |-->{Not ULA-flushing}-->PEER_COMMIT-->[Notify]
          -->[Send ACK_FLUSH]
   |-->{Not EoT}
      |-->{ULA-flushing}-->COMMITTED-->[Notify]
          -->[Send delay SNACK]
      |-->{Not ULA-flushing}-->ESTABLISHED-->[Notify]
          -->[Send delay SNACK]
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
|--{On transient state Timeout}-->NON_EXISTENT-->[Notify]
```

CHALLENGING is a state entered by the ULA accepting the connection request after a new connection context has been incarnated. The new connection is incarnated by the FSP context of the near end in the LISTENING state as a legitimate CONNECT_REQUEST packet is received.

5.6. ACTIVE

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|--[API: Send{flush}]-->COMMITTING{Urge to commit}
|<-->[API: Send{more data}][Send PURE_DATA]
|--[Rcv.NULCOMMIT]
    |-->PEER_COMMIT-->[Send ACK_FLUSH]-->[Notify]
|--[Rcv.PURE_DATA]
    |--{EOT}-->PEER_COMMIT
       -->[Send ACK_FLUSH]-->[Notify]
    |--{Not EOT}-->[Send SNACK]-->[Notify]
|--[Rcv.PERSIST]
    |--{EOT}-->PEER_COMMIT
       -->[Send ACK_FLUSH]-->[Notify]
    |--{Not EOT}-->[Send SNACK]
|--[Rcv.MULTIPLY]{passive multiplication}
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
|--{On Idle Timeout}-->NON_EXISTENT-->[Notify]
```

ACTIVE, also known as ESTABLISHED, is a state that the FSP participant has finished end-to-end negotiation but has not committed current transmit transaction nor fully received the latest transmit transaction of the remote end.

5.7. COMMITTING

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|--[Rcv.ACK_FLUSH]-->COMMITTED-->[Notify]
|--[Rcv.NULCOMMIT]
    |-->COMMITTING2-->[Send ACK_FLUSH]-->[Notify]
|--[Rcv.PURE_DATA]
    |--{EOT}-->COMMITTING2-->[Send ACK_FLUSH]-->[Notify]
    |--{Not EOT}-->[Send SNACK]-->[Notify]
|--[Rcv.MULTIPLY]{passive multiplication}
|--[Rcv.RELEASE]
    |--{EOT}-->SHUT_REQUESTED-->[Send ACK_FLUSH]-->[Notify]
    |--{Not EOT}-->[Send SNACK lazily]
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
|--{On Idle Timeout}-->NON_EXISTENT-->[Notify]
```

COMMITTING is a state that the FSP participant has committed the transmit transaction but has not fully received the latest transmit transaction of the remote end, nor the acknowledgement to the transmit transaction commitment has been received.

The participant in COMMITTING state SHALL NOT transmit further data until current transmit transaction commitment is acknowledged.

5.8. COMMITTED

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|--[API: Send{more data}]-->ACTIVE-->[Send PERSIST]
|--[API: Send{flush}]-->COMMITTING-->{Flush the send queue}
|--[Rcv.NULCOMMIT]-->CLOSABLE
   -->[Send ACK_FLUSH]-->[Notify]
|--[Rcv.PURE_DATA]
    |-->{EOT}-->CLOSABLE
        -->[Send ACK_FLUSH]-->[Notify]
    |-->{Not EOT}-->[Send SNACK]-->[Notify]
|--[Rcv.PERSIST]
    |-->{EOT}-->CLOSABLE
        -->[Send ACK_FLUSH]-->[Notify]
    |-->{Not EOT}-->[Send SNACK]
|--[Rcv.MULTIPLY]{passive multiplication}
|--[Rcv.RELEASE]
    |--{EOT}-->SHUT_REQUESTED
       -->[Send ACK_FLUSH]-->[Notify]
    |--{Not EOT}-->[Send SNACK lazily]
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
|--{On Idle Timeout}-->NON_EXISTENT-->[Notify]
```

COMMITTED is a state that the FSP participant has committed current transmit transaction and has received the acknowledgement to the transmit transaction commitment, but has not fully received the latest transmit transaction of the remote end.

5.9. PEER_COMMIT

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|--[API: Send{flush}]-->{Mark EoT or append NULCOMMIT}
    -->COMMITTING2-->{Do Send}
|--[API: Shutdown]-->PRE_CLOSED-->{Append RELEASE}
    -->{Do Send}
|<-->[API: Send{more data}][Send PURE_DATA]
|<-->[Rcv.PURE_DATA]{just prebuffer}
|<-->[Rcv.NULCOMMIT]-->[Send ACK_FLUSH]
|--[Rcv.PERSIST]
    |<-->{EOT}-->[Send ACK_FLUSH]
        --{&& is new transaction}-->[Notify]
    |-->{Not EOT}-->ACTIVE-->[Send SNACK]
|--[Rcv.MULTIPLY]{passive multiplication}
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
|--{On Idle Timeout}-->NON_EXISTENT-->[Notify]
```

PEER_COMMIT is a state that the FSP participant has not committed current transmit transaction but has fully received the latest transmit transaction of the remote end, and the acknowledgement to the transmit transaction commitment has not been received yet.

5.10. COMMITTING2

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|--[API: Shutdown]-->PRE_CLOSED-->{Append RELEASE}-->{Do Send}
|<-->[Rcv.PURE_DATA]{just prebuffer}
|<-->[Rcv.NULCOMMIT]-->[Send ACK_FLUSH]
|--[Rcv.ACK_FLUSH]-->CLOSABLE-->[Notify]
|--[Rcv.PERSIST]
    |--{EOT}--{but a new transaction}
        -->[Send ACK_FLUSH]-->[Notify]
    |--{Not EOT}-->COMMITTING-->[Send SNACK]
|--[Rcv.MULTIPLY]{passive multiplication}
|--[Rcv.RELEASE]-->SHUT_REQUESTED
    -->[Send ACK_FLUSH]-->[Notify]
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
|--{On Idle Timeout}-->NON_EXISTENT-->[Notify]
```

COMMITTING2 is a state that the FSP participant has committed current transmit transaction and has fully received the latest transmit transaction of the remote end, but the acknowledgement to the transmit transaction commitment has not been received yet.

The participant in COMMITTING2 state SHALL NOT transmit further data until current transmit transaction commitment is acknowledged.

5.11. CLOSABLE

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|--[API: Send{more data}]-->PEER_COMMIT-->[Send PERSIST]
|--[API: Send{flush}]-->COMMITTING2-->{Flush the send queue}
|--[API: Shutdown]-->PRE_CLOSED-->{Append RELEASE}-->{Do Send}
|<-->[Rcv.PURE_DATA]{just prebuffer}
|<-->[Rcv.NULCOMMIT]-->[Send ACK_FLUSH]
|--[Rcv.PERSIST]
    |--{EOT}--{but a new transaction}
       -->[Send ACK_FLUSH]-->[Notify]
    |-->{Not EOT}-->COMMITTED-->[Send SNACK]
|--[Rcv.MULTIPLY]{passive multiplication}
|--[Rcv.RELEASE]-->SHUT_REQUESTED
   -->[Send ACK_FLUSH]-->[Notify]
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
```

CLOSABLE is a state that the FSP participant has committed current transmit transaction and has received the acknowledgement to the transmit transaction commitment, and has fully received the latest transmit transaction of the remote end.

5.12. SHUT_REQUESTED

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|--[API: Shutdown]-->CLOSED-->[Notify]
|<-->[Rcv.RELEASE]-->[Send ACK_FLUSH]
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
```

SHUT_REQUESTED is a state entered when a legitimate RELEASE packet was received.

A connection context MAY persist in SHUT_REQUESTED state until the session key runs out of life, or the host system needs to recycle the resource allocated. A connection in SHUT_REQUESTED state MAY be resurrected.

5.13. PRE_CLOSED

```
---[API: Reset]-->NON_EXISTENT-->[Send RESET]
|--[Rcv.ACK_FLUSH]-->CLOSED-->[Notify]
|--[Rcv.RELEASE]-->[Notify]-->CLOSED
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
|--{On transient state Timeout}-->CLOSED-->[Notify]
```

PRE_CLOSED is a state entered on the ULA calling the API Shutdown in COMMITTING2 or CLOSABLE state.

5.14. CLOSED

```
   |--{On Recycling Needed}-->NON_EXISTENT
```

CLOSED is a state migrated from PRE_CLOSED state on receiving a legitimate ACK_FLUSH packet from the remote end, or from SHUT_REQUESTED state on the ULA calling the API Shutdown.

Unlike TCP [STD7], CLOSED state in FSP is not fictional. Instead a connection context MAY persist in CLOSED state until the session key runs out of life, or the host system needs to recycle the resource allocated to the CLOSED session. A connection in CLOSED state MAY be resurrected.

5.15. CLONING

```
---[API: Reset]-->NON_EXISTENT
|<-->[API: Send{new data}]{just prebuffer}
|<-->[Rcv.PURE_DATA]{just prebuffer}
|--[Rcv.NULCOMMIT]
    |--{&& Not ULA-flushing}-->PEER_COMMIT
       -->[Send ACK_FLUSH]-->[Notify]
    |--{&& ULA-flushing}-->CLOSABLE
        --->[Send ACK_FLUSH]-->[Notify]
|--[Rcv.PERSIST]
    |-->{Not EOT}
        |--{&& Not ULA-flushing}-->ACTIVE
           -->[Send SNACK]-->[Notify]
        |--{&& ULA-flushing}-->COMMITTED
           -->[Send SNACK]-->[Notify]
    |-->{EOT}
        |--{&& Not ULA-flushing}-->PEER_COMMIT
           -->[Send ACK_FLUSH]-->[Notify]
        |--{&& ULA-flushing}-->CLOSABLE
           -->[Send ACK_FLUSH]-->[Notify]
|--[Rcv.RESET]-->NON_EXISTENT-->[Notify]
|--{On transient state Timeout}-->NON_EXISTENT-->[Notify]
```

CLONING is a state entered by ULA calling the API Multiply from any state that may accepting an out-of-band packet.

5.16. Passive Multiplication

```
{ACTIVE, COMMITTING, COMMITTED, PEER_COMMIT, COMMITTING2, CLOSABLE}
  |-->/MULTIPLY/-->[API{Callback}]-->{new context}CONNECT_AFFIRMING
     |-->[{Return Accept}]
        |-->{has payload prebuffered}
            -->{Send packet(s) starting with PERSIST}
        |-->{without payload}-->[Send NULCOMMIT]
     |-->[{Return Reject}]-->{abort creating new context}
         -->[Send RESET]
```

In the ACTIVE, COMMITTING, COMMITTED, PEER_COMMIT, COMMITTING2 or CLOSABLE state an FSP end node MAY accept its peer's connection multiplication request and transit to the unnamed, temporary passive multiplication state.

5.17. Typical State Transitions

This section is informative.

5.17.1. Typical Main Connection

```
***                      Bootstrapping                 ***
CONNECT_BOOTSTRAP  ------ INIT_CONNECT ------->  LISTENING
    |              <---- ACK_CONNECT_INIT -----
CONNECT_AFFIRMING  ------ CONNECT_REQUEST ---->      |

*** Connection affirmation, carrying welcome messages ***
                                                     |
                                                 {Accept}
   {and send a single packet welcome message immediately}
                                                     |
    |            <- ACK_CONNECT_REQ c/w EoT -  CHALLENGING
PEER_COMMITTED
    |
{Callback, to send ticket immediately}
{(a fictional identification token of a single packet)}
    |
COMMITTING2        ----- PERSIST c/w EoT ----->      |
                                                 CLOSABLE
    |              <-------- ACK_FLUSH --------      |
CLOSABLE
                                                     |
                                  {Send Server's Challenge}
                                                     |
    |           <----- PERSIST w/o EoT -----  PEER_COMMITED
COMMITTED          ---- KEEP_ALIVE(SNACK) ---->
                             .
                             .

                             .                       |
                                                 {Flush}
                                                     |
    |            <---- PURE_DATA c/w EoT ----  COMMITTING2
CLOSABLE
                   --------- ACK_FLUSH ------->      |
                                                 CLOSABLE
    |
{Send Client's Response}
    |
PEER_COMMITTED     ----- PERSIST w/o EoT ----->      |
                                                 COMMITTED
    |              <--- KEEP_ALIVE(SNACK) -----      |
                             .
    |              ---- PURE_DATA w/o EoT ---->  COMMITTED
PEER_COMMITTED     <--- KEEP_ALIVE(SNACK) ----       |
                             .
                             .

    |                        .
{Flush}
```

```
      |
    COMMITTING2        ---- PURE_DATA c/w EoT ---->      |
                                                    CLOSABLE
      |                <-------- ACK_FLUSH --------
    CLOSABLE

  ** Typical C/S request-response exchange on application layer **
      |
    {Send Request}
      |
    PEER_COMMITTED     ----- PERSIST w/o EoT ----->      |
                                                    COMMITTED
      |                <--- KEEP_ALIVE(SNACK) -----      |
                                 .
      |                ---- PURE_DATA w/o EoT ---->  COMMITTED
    PEER_COMMITTED     <--- KEEP_ALIVE(SNACK) -----      |
                                 .
                                 .
      |                          .
    {Flush}
      |
    COMMITTING2        ---- PURE_DATA c/w EoT ---->      |
                                                    CLOSABLE
      |                <-------- ACK_FLUSH --------
    CLOSALBE
                                                  {Send Response}
                                                         |
      |                <----- PERSIST w/o EoT -----  PEER_COMMITED
    COMMITTED
                       ---- KEEP_ALIVE(SNACK) ---->      |
                                 .
      |                <---- PURE_DATA w/o EoT ----  PEER_COMMITED
    COMMITTED          ---- KEEP_ALIVE(SNACK) ---->      |
                                 .
                                 .
                                 .
                                                    {Flush}
                                                         |
      |                <---- PURE_DATA c/w EoT ----  COMMITTING2
    CLOSABLE
                       --------- ACK_FLUSH ------->      |
                                                    CLOSALBE
                                 .
                                 .
                                 .
     *** Following request-responses, e.g. HTTP pipelining ***
                                 .
                                 .
                                 .
```

```
CLOSABLE                                         CLOSALBE
                             .
                             .
 *** End of connection, in a typical C/S application ***
                                                     |
                                                 {Shutdown}
                                                     |
   |               <---------- RELEASE --------  PRE_CLOSED
SHUT_REQUESTED
                   --------- ACK_FLUSH ------->      |
   |                                               CLOSED
{Shutdown}
   |
 CLOSED
```

5.17.2. Typical Clone Connection for Get Resource

```
    Client                                                 Server
    *** Suppose the browser fork a new connection to request ***
    *** some resource and the URI is short enough to be      ***
    *** encapsulated in a single packet                      ***
   {MultiplyAndWrite}
        |                                    {In Clonable State}
   CLONING            ---- MULTIPLY c/w EoT ---->       *
                                         {Make Clone Connection}
                           {and return resource data immediately}
                                                        |
        |             <---- PERSIST w/o EoT ------  PEER_COMMITTED
   COMMITTED          ---- KEEP_ALIVE(SNACK) ---->

                      <---- PUER_DATA w/o EoT ----  PEER_COMMITTED
   COMMITTED          ---- KEEP_ALIVE(SNACK) ---->
                                .
                                .
                                .                       |
                                                    {Flush}
                                                        |
        |             <--- PURE_DATA c/w EoT -----  COMMITTING2
   CLOSABLE
                      --------- ACK_FLUSH ------->      |
                                                    CLOSABLE
                                .
                                .
                   *** End of Connection ***
```

5.17.3. Typical Clone Connection for Push Message

```
Initiator of Multiplication              Responder of Multiplication
  *** Suppose the server fork a new connection to push message ***
  (Used to be server side)                  (Used to be client side)
      |
  {MultiplyAndWrite}
      |                                         {In Clonable State}
  CLONING            ---- MULTIPLY w/o EoT ---->       *
                                         {Make Clone Connection}
           *** suppose no immediately available responding data ***
                                                       |
                                                   COMMITTING
      |              <--- NULCOMMIT(c/w EoT) ----      |
  PEER_COMMITTED
      |              --------- ACK_FLUSH ------->  COMMITTED
                               .
      |              ---- PURE_DATA w/o EoT ---->  COMMITTED
  PEER_COMMITTED     <--- KEEP_ALIVE(SNACK) ----       |
                               .
                               .
      |                        .
  {Flush}
      |

  COMMITTING2        ---- PURE_DATA c/w EoT ---->      |
                                                   CLOSABLE
      |              <-------- ACK_FLUSH --------
  CLOSABLE

                               .
                               .
                  *** End of Connection ***
```

5.17.4. Simultaneous Shutdown

```
CLOSABLE                                         CLOSABLE
   |                                                 |
{Shutdown}                                       {Shutdown}
   |                                                 |
PRE_CLOSED                                       PRE_CLOSED
   |  \                                          /   |
   |   \                                        /    |
   |    \                                      /     |
   |     \------------- RELEASE --------------/----->+
   +<------------------- RELEASE ------------/       |
   |                                                 CLOSED
CLOSED
```

6. End-to-End Negotiation

End-to-end negotiation of FSP session occurs in the connection establishment phase. Connection establishment process of FSP consists of two and a half pairs of packet exchanges for connection initialization, connection incarnation and the last confirmation. During the process various optional header or payload MAY be carried in the FSP preliminary packets to negotiate end-to-end session parameters.

6.1. Connect Initialization

The initiator sends the INIT_CONNECT packet to the responder: (INIT_CONNECT, Salt, Timestamp, Init-Check-Code [, Responder's Host Name])

Connection initialization MAY be syndicated with optional address resolution at the gateway in the IPv6 network by carrying the responder's host name in the INIT_CONNECT packet.

If it does carry the responder's host name it MUST take the link-local interface address [RFC4291] as the source IPv6 address and the default link-local gateway address, FE80::1, as the destination IPv6 address no matter whether the global unicast IP address of the default gateway is configured.

If the gateway that relays the INIT_CONNECT packet finds that the responder is on the same link-local network with the initiator it SHALL change the source and the destination IP addresses of the INIT_CONNECT packet to the link-local IP addresses of the initiator and the responder respectively, and relay the packet onto the same link-local network.

On receiving the INIT_CONNECT packet that carries the responder's host name the link-local gateway MUST resolute the responder's global unicast IPv6 address and map the initiator's global unicast IPv6 address, and replace the destination and source address of the INIT_CONNECT packet respectively, unless it finds that the initiator and the responder are on the same link-local network, where the gateway SHALL process the packet as stated in the previous statement.

The gateway SHALL silently ignore the INIT_CONNECT packet if it is unable to resolve the IP address of the responder.

If the destination address is the default link-local gateway address while the INIT_CONNECT does not carry the responder's host name payload, it is supposed that the gateway is the intent destination of the connection to initialize.

6.2. Response to Connect Initialization

The responder sends acknowledgment to the initiator:

(ACK_INIT_CONNECT, Time-delta, Cookie, Init-Check-Code Reflected, Responder's Sink Parameter)

If the responder is ready to accept the connection, it SHALL generate a cookie which is meant to be reflected by the initiator. The responder MUST send the ACK_INIT_CONNECT packet with the new allocated local ULTID instead of the original listening ULTID. The initiator should be able to find out the original listening ULTID by searching its own connection context.

In the Responder's Sink Parameter the original listener ULTID MUST be set to the right value.

The destination address of the packet sent back MUST be set to the source address of the corresponding Connect Initialization packet whose source and destination address MAY be updated by some intermediary such as the link-local gateway of the initiator.

The responder SHALL NOT make state transition on receiving INIT_CONNECT packet.

If the responder refuses to accept the connection, it SHALL silently discard the INIT_CONNECT packet.

6.3. Connection Incarnation Request

(CONNECT_REQUEST, Salt, Timestamp, Init-Check-Code, Initial SN, Time- delta Reflected, Cookie Reflected, Initiator's Sink Parameter [, Initiator's Host Name])

The initiator accepts the Response to Connect Initialization packet if and only if both the destination ULTID of the response packet matches the source ULTID of the connect initialization packet and the Init-Check-Code reflected in the response packet matches the Init- Check-Code in the connect initialization packet.

If the response packet is accepted the initiator formally requests to establish the connection by sending the CONECT_REQUEST packet.

In the CONNECT_REQUEST packet the value of the Timestamp, the Init-Check-Code and the Salt field MUST be the same as in the INIT_CONNECT packet while the value of the Cookie Reflected field and the Time- delta Reflected field MUST be the same as in the ACK_INIT_ CONNECT packet, respectively.

The initiator MUST send the packet towards the remote ULTID that the responder has preserved and sent with the ACK_INIT_CONNECT packet. It MUST fill the original listener ID field in the Initiator's Sink Parameter with the right value.

The source address of the CONNECT_REQUEST packet MUST be set to the destination address of the received ACK_INIT_CONNECT packet, while the network prefix and host-id part of the destination address MUST be set to the source address of the received ACK_INIT_CONNECT packet in the IPv6 network.

The initiator SHALL save the cookie value that the responder has given to make up the weak session key.

The initiator MUST fill the Initial SN field with the sequence number of the packet that will follow CONNECT_REQUEST. The CONNECT_REQUEST packet is payload free and does not consume the sequence space.

The optional fields Initiator's Host Name is put as the payload of the CONNECT_REQUEST packet. If presented it MAY be exploited by the responder as the last resort to resolute the most recent IP address of the initiator in some extraordinary scenarios such as the initiator has hibernated for a considerably long time.

6.4. Connection Incarnation Response

Case 1: (ACK_CONNECT_REQ, FREWS, Initial SN, Expected SN, ICC[, Payload])

Case 2: (RESET, Reason of Failure, Timestamp Reflected, Copy of Cookie Reflected)

The responder responds as in case 1 if the reflected cookie was validated, resources were successfully allocated and the initial context of the connection was setup. Otherwise it SHOULD respond as in case 2.

However, if the cookie is invalid, the responder SHALL silently discard the CONNECT_REQUEST packet.

The Initial SN in case 1 is the initial sequence number of the responder. The responder should fill in the field with a random 32-bit unsigned integer. As the ACK_CONNECT_REQ packet may carry payload the sequence number of the responder starts from the ACK_CONNECT_REQ packet.

The Expected SN MUST equal to the Initial SN specified in the corresponding CONNECT_ REQUEST packet.

6.5. The Last Confirmation

Case 1: (NULCOMMIT, FREWS, Initial SN, Expected SN, ICC)

Case 2: (PERSIST, FREWS, Initial SN, Expected SN, ICC, payload)

Case 3: (RESET, Reason of Failure, Initial SN, Expected SN, ICC)

The initiator of the connection MUST eventually confirm to the responder that the connection is established by sending a payload-less NULCOMMIT packet (case 1) or a PERSIST packet with payload (case 2).

Of course the initiator MAY quit to establish the connection by sending a legitimate RESET packet (case 3).

6.6. Retransmission

The initiator SHALL retransmit the INIT_CONNECT packet if the corresponding ACK_INIT_CONNECT packet is not received in some limit time (by default 15 seconds).

The initiator SHALL retransmit the CONNECT_CONNECT packet if the corresponding ACK_CONNECT_REQ packet is not received in some limit time (by default 15 seconds).

The responder SHALL NOT retransmit ACK_INIT_CONNECT or ACK_CONNECT_REQ packet.

The initiator SHOULD retransmit the right INIT_CONNECT packet or CONNECT_CONNECT packet until the legitimate ACK_CONNECT_REQ packet is eventually received.

It SHALL give up if the time starting from the very first INIT_CONNECT packet was sent has exceed a longer timed-out value (by default 60 seconds) before the legitimate ACK_CONNECT_REQ packet is received.

7. Quad-party Session Key Installation

It is assumed that in the scenarios applying FSP it is the ULA to do key establishment and/or end-point authentication while the FSP layer provides authenticated, optionally encrypted data transfer service. The ULA installs the established shared secret key as the new session key of the FSP layer. Together they establish a secure channel between two application end-points.

In a typical scenario the ULA endpoints first setup the FSP connection where resistance against connection redirection is weakly enforced by CRC64. After the pair of ULA endpoints have established a shared secret key, they install the secret key. Authenticity of the FSP packets sent later is cryptographically protected by the new secret key and resistance against various attacks is secured.

Although transmit transaction is actually uni-directional the secret key is shared bi-directionally in this version of FSP.

Protocol for installation of the shared secret key is quad-party in the sense that both the upper layer application and the FSP layer of both the participant nodes MUST agree on the moment of certain state to install the shared secret key.

It is arguably much more flexible for the application layer protocols to adopt new key establishment algorithm while offloading routine authentication and optionally encryption of the data to the underlying layers where it may be much easier to exploit hardware-acceleration.

7.1. API for Session Key Installation

A dedicate application program interface (API) is designed for the ULA to install the secret key established by the ULA participants. The API SHOULD take four parameters:

* A 'handle' to state the connection context for installing the session key

* A octet string of initial key materials (IKM)

* An integer to state the length of IKM. The unit is octet.

* An integer to state the desired length of the effective session key if AEAD is applied. The unit is bit. For this version of FSP desired length of the effective session key is either 128 or 256.

The peer MUST have commit a transmit transaction and it SHALL install the same secret key on receiving the FSP packet with the EoT flag set.

The ULA SHOULD have installed the new shared secret key, or install it instantly after accepting the packet with the EoT flag set. If the new secret key has ever been installed the packet received after the one with the EoT flag set MUST adopt the new secret key.

7.2. Time to Call API for Session Key Installation

A participant MAY install new session key if and only if the packet with the latest sequence number it has received has EoT flag marked.

7.3. Time to Take New Session Key into Effect

By committing a transmit transaction a ULA participant clearly tells the underlying FSP layer that the next packet sent MAY adopt a new secret key. On receiving a packet with the EoT flag set the ULA is informed that the next packet received MAY adopt a new shared secret key.

After the ULA of a network node installed a new session key, every packet to send with sequence number later than the one with the EoT flag set just before the API to install session key was called MUST adopt the new session key in the FSP layer of the network node.

Every packet received with the sequence number later than the one with EoT flag set when the ULA called the API to install session key MUST be validated with the new session key. If the new secret key has ever been installed the packet received after the one with the EoT flag set MUST adopt the new secret key.

7.4. Generating the Initial Session Key

When the ULA install the secret key, it is required to provide the initial key material which might have unbalanced bit randomness, not the session key itself. HMAC-based Extract-and-Expand Key Derivation Function (HKDF) [RFC5869] is applied to generate the initial session key.

Given raw key material ikm, length of the ikm nB in octets, intended master key length lenb in bits, || is octet string concatenation,

If HMAC only is designated, the nB octets of ikm is hashed into 64 octets which is taken as the master key.

If AEAD is designated, the initial session key, or the first secret key for packet authentication and payload encryption is obtained as specified in [RFC5869]:

Key Extract phase,
Let Km = BLAKE2(zeros || ikm) , where zeros is
32 octets of integer 0 BLAKE2b algorithm without key is applied.
The result Km is the master key.

Key Expand phase,

Let Ks = BLAKE2(Km, info) , where

Km is the master key generated in previous phase,

info is concatenation of the arbitrary ASCII string "Establishes an FSP session", which is 26-octet long, 3 octets of integer 0, and 1 octet of integer 1.

BLAKE2b algorithm with key is applied. The key applied MUST be the master key Km.

The result Ks is the initial session key, or the first secret key for packet authentication and payload encryption. For this version FSP the generated Ks is a fixed-length AES key together with a 4-octet salt. The salt is meant to be passed to AES-GCM as the initialization vector together with the sequence number and expected sequence number fields in the normal FSP fixed header:

```
 0                                                             31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Salt                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Expected Sequence Number/Out-of-band Serial Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The salt is called 'the original salt', separated from 'the second salt' in the out-of-band packet.

7.5. Internal Rekeying

In this version of FSP it is forced to re-key on sending or receiving every 536870912? (2^29) packets.

Let Ks' = BLAKE2(Km, H || info') , where: Km is the master key generated as in section 7.4.

H is the 16-octet internal hash sub-key of AES-GCM of previous session key,
info' is concatenation of the arbitrary ASCII string "Sustains an FSP connection", which is 26-octet long and the 4 octets in network order of the 32-bit unsigned integer that specifies the batch index of the session key.

BLAKE2b algorithm with key is applied. The key applied MUST be the master key Km.

The result Ks' is the new session key, together with the new salt meant to be form the IV.

The batch index of the initial session key is 1, and it is increased by 1 every time before it is to re-key.

7.6. Sample Sequence of Installing Session Key

This section is informative.

```
      Node A                                          Node B
ULA-A        FSP-A                             FSP-B        ULA-B
{Send Km-A}
               ->[seq_a2b_0]                  ->
                                                     {Send Km-B}
               <-                  [seq_b2a_0]<-
                             .
                             .
                             .
{Commit}
               ->[seq_a2b_m c/w EoT]          ->
                                                   {Install Key}
                             .
{Wait}                       .
                             .
                                                        {Commit}
               <-          [seq_b2a_n c/w EoT]<-
{Install Key}
                             .
                             .                    {Send Further}
               <-                [seq_b2a_n+1]<-
                             .
{Send Further}               .
               ->[seq_a2b_m+1]                ->
```

o Send Km-A, Send Km-B

ULA of node A and node B send there key material for key establishment, respectively.

-Commit

-ULA of node A or node B informs the FSP layer to set the End of Transaction flag of the last packet to send and flush the send buffer.

-Install Key

-ULA of node A or node B informs the FSP layer to install new session key, giving key materials for deriving the session key. A node may call Install Key if and only if its peer has just committed a transmit transaction.

Wait

The ULA MUST wait until it has received some packet with EoT set from its peer before it may install new session key. There is no mandatory calling order of Commit and Install Key. However, if a node Commit before Install Key and it wants to apply new session key for the transmit transaction next to the one it has just committed, it SHALL NOT send further data until Install Key has returned successfully. In the above example, for node A packet with sequence number [seq_a2b_m+1] will be sent by applying the new session key, for node B packet with sequence number [seq_b2a_n+1] will be sent by applying the new session key.

Send Further

ULA of node A or node B sends further data in the new transmit transaction, respectively. There is no mandatory order on which node should start new transmit transaction firstly.

8. Send and Receive

8.1. Packet Integrity Protection

8.1.1. Application of CRC64

Starting from ACK_CONNECT_REQUEST, until the ULAs have installed the shared secret CRC64 is applied to calculate the value of the ICC field. The algorithm:

1. Take pair of the ULDs as the initial value of accumulative CRC64

2. Accumulate the value of the Init-Check-Code field

3. Accumulate the value of the Cookie field successively

4. Accumulate the combined value of the salt and the timeDelta field where the former is the leftmost 32 bits and the latter is the rightmost 32 bits

5. Accumulate the value of the Time Stamp field

6. Save the accumulated CRC64 value as the pre-computed CRC64 value

The pair of the ULDs is composed of the near end's ULTID and the remote end's ULTID, where the former is the leftmost 32 bits and the latter is the rightmost 32 bits of initial value for the send direction, and the order is reversed for the receive direction.

When calculate the value ICC of a particular FSP packet, firstly set ICC to the pre-computed CRC64 value, then calculate the CRC64 checksum of the whole FSP packet, while ULTIDs are NOT included if the FSP packet is encapsulated in UDP. The result is set as the final value of the ICC field.

8.1.2. Packet Authentication Only

The ULA designates the FSP layer to either applying HMAC only or applying AEAD.

If the HMAC flag of a packet is set the pre-designated cryptographic hash function SHALL be applied to get the message authentication code (MAC) of the whole packet. Each FSP version MUST designate one and only one particular cryptographic hash function.

For this FSP version, BLAKE2 [RFC7693] is designated as the cryptographic hash function. The input key is the secret key that has been derived from the master key installed by the ULA. The input data is the full FSP packet, where the ICC field is pre-filled the pair of the ULDs. As in making CRC64 checksum, the pair of the ULDs is composed of the near end's ULTID and the remote end's ULTID, where the former is the leftmost 32 bits and the latter is the rightmost 32 bits of initial value for the send direction, and the order is reversed for the receive direction.

The hash result is truncated to 64 bits to get the final ICC.

8.1.3. Authenticated Encryption with Additional Data

FSP provides per-packet authenticated encryption service. Only one authenticated encryption algorithm is allowed for a determined version of FSP. For this FSP version, the authenticated encryption algorithm selected is GCM-AES [GCM][AES], it is applied to protect integrity of the full FSP packets, and privacy of the payload together with the extension headers, if any. The four inputs to GCM-AES authenticated encryption are:

K: the key derived by the master key installed by ULA. The length of the session key is determined by the ULA.

IV: the initial vector, 96-bit string made by concatenating a 32-bit salt, the 32-bit sequence number of the packet and the 32-bit expected sequence number field of the packet. The salt is derived by the master key installed by ULA.

P: the plaintext are the bytes following the fixed header up to the end of the original payload.

AAD: additional authenticated data, for this version of FSP it is the fixed header of the FSP packet. The source ULTID MUST be stored in the leftmost 32-bit of the ICC field while the destination ULTID MUST be stored in the rightmost 32-bit of the ICC field before the ICC value is calculated.

The length of the authentication tag MUST be 64 bits for FSP version 0 and 1. The authentication tag is stored in the ICC finally.

The inputs to GCM-AES decryption are:

K: the key derived by the master key installed by ULA. The length of the session key is determined by the ULA.

IV: the initial vector, 96-bit string made by concatenating consisted of the 32-bit salt, the 32-bit sequence number of the packet and the 32-bit expected sequence number field of the packet.

C: the cipher-text are the bytes following the fixed header up to the end of the received payload.

AAD: additional authenticated data, for this version of FSP it is the fixed header of the FSP packet. The source ULTID MUST be stored in the leftmost 32-bit of the ICC field while the destination ULTID MUST be stored in the rightmost 32-bit of the ICC field before the ICC value is calculated.

T: The authentication tag, which is fetched from the ICC field received.

Only when the outputs of GCM-AES decryption tell that the authentication tag passed verification may the receiver deliver the decrypted payload to the ULA.

8.1.4. ICC of the Out-of-Band Packet

When calculating the ICC of an out-band packet (KEEP_ALIVE, ACK_FLUSH or MULTIPLY), the ExpectedSN field SHALL be filled with the out-of- band serial number. The first 32-bit word of the fixed header is taken as the second salt.

To get or check the ICC of the out-of-band packet the original salt value that is set on deriving the session key and stored in the internal security context MUST be XORed with the second salt value before applying GCM-AES. The original salt value MUST be recovered instantly after GCM-AES is applied.

8.2. Start a New Transmit Transaction

The responder starts a transmit transaction by send the ACK_CONNECT_REQ packet which MAY terminate the transmit transaction at the same time.

When the initiator further acknowledges ACK_CONNECT_REQ with NULCOMMIT it both starts AND terminates a transmit transaction without sending any data:

```
(NULCOMMIT, FREWS, SN, ExpectedSN, ICC [, Sink Parameter])
```

Any party MAY start a new transmit transaction by sending a PERSIST packet:

```
(PERSIST, FREWS, SN, ExpectedSN, ICC, Payload)
```

PERSIST packet sent by the initiator firstly acknowledges the ACK_CONNECT_REQ packet as well.

8.3. Send a Pure Data Packet

```
(PURE_DATA, FREWS, SN, ExpectedSN, ICC, Payload)
```

After a new transmit transaction has been started further PURE_DATA packet MAY be sent until a packet with EoT flag set is sent.

8.4. Commit a Transmit Transaction

8.4.1. Initiate Transmit Transaction Commitment

A participant of an FSP connection MAY notify its peer that a transmit transaction shall be committed by setting the EoT flag of the last packet of the transmit transaction, be it NULCOMMIT, PERSIST, PURE_DATA or MULTIPLY.

8.4.2. Respond to Transmit Transaction Commitment

(ACK_FLUSH, FREWS, SN, ExpectedSN, ICC, Sink Parameter, SNACK)

If and only if all of the packets in a transmit transaction has been received MAY ACK_FLUSH packet be sent.

Whenever a legitimate packet falls in the receive window of the receiver, and the packet fills in the last gap of the sequence of current transmit transaction on receiving direction, or the packet with same sequence number has been accepted already, a responding ACK_FLUSH SHALL be sent back immediately, and the FSP layer MUST immediately notify the ULA that a transmit transaction has been committed.

The sequence number (SN) of the ACK_FLUSH packet MUST equal the latest sequence number of the legitimate packets that have been sent.

The out-of-band serial number SHALL increase by one whenever a new ACK_FLUSH packet is sent.

The ACK_FLUSH packet contains a Sink Parameter extension header, alike KEEP_ALIVE.

The ACK_FLUSH packet SHALL contain a SNACK extension header, although number of gap descriptors in the SNACK header MUST be 0.

8.4.3. Finalize Transmit Transaction Commitment

After receiving the ACK_FLUSH packet the sender of the EoT flag migrates to the COMMITTED or CLOSABLE state from the COMMITTING or COMMITTING2 state, respectively.

8.4.4. Time-out for Committing Transmit Transaction

The ULA SHALL be timed-out if there is no packet was acknowledged in some hard-coded time-out. For this version of FSP the time-out is set to 30 seconds.

8.5. Retransmission

8.5.1. Calculation of RTT

We borrows specifications for calculating RTT (and RTO) considerably from Computing TCP's Retransmission Timer [RFC6298] to calculate Retransmission Time Out (RTO).

The sender maintains two state variables, SRTT (smoothed round-trip time) and RTTVAR (round-trip time variation). In addition, we assume a clock granularity of G seconds.

Initial round trip time (RTT) for the Connection Initiator: Equals to the mean of the time elapsed when ACK_ INIT_CONNECT was received since INIT_CONNECT was sent, and the time elapsed till ACK_CONNECT_REQ was received since CONNECT_REQUEST was sent.

Initial RTT for the Connection Responder: Equals to the time elapsed when the CONNECT_REQUEST packet was received since the ACK_INIT_CONNECT packet had been received.

Initial RTT for the Initiator of Connection Multiplication: Equals to the most recent RTT of the multiplied connection.

Initial RTT for the Responder of Connection Multiplication: Equals to the most recent RTT of the multiplied connection.

  *When the Initial RTT measurement R is made, the host MUST set

```
SRTT <- R
RTTVAR <- R/2
```

  *When a subsequent RTT measurement R' is made, a host MUST set

```
RTTVAR <- (1 - beta) * RTTVAR + beta * |SRTT - R'|
SRTT <- (1 - alpha) * SRTT + alpha * R'
```

The value of SRTT used in the update to RTTVAR is its value before updating SRTT itself using the second assignment. That is, updating RTTVAR and SRTT MUST be computed in the above order.

The above SHOULD be computed using alpha=1/8 and beta=1/4.

R' SHOULD be measured whenever a packet with the SNACK extension header is received.

Suppose the packet with the latest SN that is accumulatively acknowledged is P-latest, R' equals the time when the SNACK header is received, minus the time when P-latest was sent, minus the delay that the acknowledgment was made.

The delay that the acknowledgment was made is stored in the "Acknowledgement Delay" field of the SNACK header. It equals the time difference between the time when the acknowledgement was sent and the time when P-latest was received.

Note that the no packet with SN later than any gap described in the SNACK header is considered as the packet with the latest SN that is accumulatively acknowledged.

8.5.2. Generation and transmission of SNACK

Whenever the receiver receives a packet it SHALL shift the time to send next heartbeat signal earlier to the time of RTT since current time, if the time to send next heartbeat signal used to be later. If the time is already earlier than the time of RTT since current time, it needs not be shifted.

On the time to send the heartbeat signal the FSP node generates the SNACK header, then generate and send a new KEEP_ALIVE or ACK_FLUSH packet to carry the SNACK header. It SHALL send an ACK_FLUSH if it is in PEER_COMMIT, COMMITTING2 or CLOSABLE state, otherwise it SHALL send a KEEP_ALIVE packet.

8.5.3. Negative acknowledgment of Packets Sent

Both the ACK_FLUSH and the KEEP_ALIVE packet in FSP carry the SNACK extension header, although number of gap descriptors in the SNACK extension header in the ACK_FLUSH packet MUST be 0. We call them SNACK packets. A SNACK packet P1 is said to be later than P0, if and only if SN of P1 is later than SN of P0, or SN of P1 equals SN of P0 while the out-of-band sequence number of P1 is later than that of P0. If the latest SNACK packet is ACK_FLUSH, all the packets with the sequence number later that the expected field of the packet are assumed to be negatively acknowledged.

By convention when we specify the range, the left square bracket meant to be inclusive, while the right parenthesis meant to be exclusive, the packets with SN in the ranges:

[expectedSN, expectedSN + 1st Gap Width),

[expectedSN + 1st Gap Width + 1st Data Length, expectedSN + 1st Gap Width + 1st DataLength + 2nd Gap Width), ...

[expectedSN + 1st Gap Width + 1st Data Length... + (n-1)th Gap Width + (n-1)th Data Length, expectedSN + 1st Gap Width + 1st DataLength... + n-th Gap Width)

together with the packets with SN later than (expectedSN + 1st Gap Width + 1st DataLength + ... + n-th Gap Width), these packets are assumed to be negatively acknowledged.

8.5.4. Retransmission Interval

Until RTT measurement has been made for a packet sent between the sender and receiver, the sender SHOULD set RTO <- 1 second.

After computing new SRTT, a host MUST updated RTO <- SRTT + max (G, K*RTTVAR) where K = 4.

Clock granularity SHOULD be finer than 100msec, that is, it SHOULD be that G <= 0.1 second.

Whenever RTO is computed, if it is less than 1 second, then the RTO SHOULD be rounded up to 1 second.

An implementation MUST manage the retransmission timer(s) in such a way that A packet is never retransmitted less than one RTO after the previous transmission of that packet.

Every time an in-band packet is sent (including a retransmission), if the timer is not running, start it running so that it will expire after RTO seconds (for the current value of RTO).

When all outstanding data has been acknowledged, turn off the retransmission timer.

When the retransmission timer expires, retransmit the packets that have not been acknowledged by the receiver, but limit by the rate throttling mechanism.

Rate of retransmission MUST be throttled in a way that No more that M/2 packets may be retransmitted in a clock interval, suppose in each clock interval M packets were sent averagely.

Packet retransmission SHALL be subjected to congestion control as well.

However, at least one packet MAY be retransmitted in one clock interval, provide that the retransmission timer expires for the first packet that has not been acknowledged yet.

8.6. Flow Control

The participants of an FSP connection negotiate the initial receive window size with the FREWS field in the ACK_CONNECT_REQ packet, and the first PERSIST or NULCOMMIT packet that acknowledges the ACK_CONNECT_REQ packet, respectively. The receive window size SHALL NOT be less than 4 and SHALL be less than 2^24.

An FSP participant advertises current receive window size in the FREWS field.

An FSP participant SHALL NOT send a packet whose sequence number is later than the value of the ExpectedSN field plus the advertised receive window size, where both value come from the very packet received with the latest sequence number.

8.7. Congestion Control

FSP supposes that end-to-end congestion control is provided by some shim layer, such as the congestion manager [RFC3124] between the "traditional" IP layer and the FSP transport layer. The shim layer is considered as a sub-layer of the network layer. Implementation of FSP MUST provide such shim layer if the network layer of the end node does not provide end-to-end congestion management service.

FSP layer SHALL provide following information to the congestion manager as soon as the first packet on the fly was acknowledged by any mean, or a legitimate packet falling in the receive window with the ECE flag set is received:

* The local interface number that the packet carrying the ECE signal is accepted.

* The remote network prefix that the congestion information is meant to associate. Note that the aggregated host ID part is NOT included in the prefix.

* The traffic class. For FSP it is bisected: MIND flag set or not.

* Number of outstanding octets, including all of those in the payload AND the FSP headers.

* The effective round trip time calculated in the most recent period. Note that retransmitted packets MUST be excluded on calculating the effective RTT.

* Whether an ECN-Echo signal was received. The ECE flag of a legitimate packet falling in the receive window is the ECN-Echo signal.

* Whether a sent packet with SRR flag set is acknowledged.

The congestion manager SHOULD reduce the send rate if the FSP sender informed it that an ECN-Echo signal was received.

The sender SHALL NOT inform the congestion manager to reduce the send rate again even if further packet with ECE flag set is received, until at least one sent packet with SRR flag set is acknowledged.

A packet with ECE flag set received after the packet with SRR flag set is acknowledged SHOULD make the congestion manager reduce the send rate again.

Retransmitted packet SHALL be subjected to send rate control at the underlying congestion management service sub-layer as well.

Quota or other means to enforce fairness among various FSP connections SHOULD be provided directly to the ULA by the congestion management service.

Requirement of an FSP congestion manager would be detailed in a separate document.

8.8. On-the-Wire Compression

FSP exploits the lossless compression algorithm as per [LZ4].

If the CPR flag of the first packet of a transmit transaction is set, compression is applied on the payload octet stream of the transaction transaction.

When applying compression FSP divides source stream into multiple blocks. For this version of FSP length of each block is 128KiB (131072 octets), except the final block whose length may be less than or equal to 128KiB. The final block is the one that terminate the transmit transaction, i.e. which contains the last FSP packet of the transmit transaction. The last FSP packet of the transmit transaction has the EoT flag set. The "LZ4_compress_fast_continue" method SHALL be applied on each block. That is, data from previous compressed blocks are taken use for better compression ratio.

When transferring the result data of compressing each block, the result data is prefixed with its length. The length is expressed by a 4-octets little-endian integer.

On-the-wire compression of each transmit transactions is independent. It is the upper layer application that SHALL make agreement on which transmit transaction utilizes on-the-wire compression.

8.9. Milk Like Payload and Minimal Delay Service

An ordinary data flow is wine-like in the sense that the older data are more valuable. If it has to, data packet sent latest are dropped first. In the contrary, milk-like payload is that the newer data are more precious and outdated data packet can be discarded.

When ULA is willing to accept incomplete message the peer of the underling FSP node SHALL set the MIND flag of the first PERSIST packet that starts the first transmit transaction, and set the MIND flag of every following PURE_DATA packet, while set the Traffic Class of the underlying IPv6 packet to some registered value.

In the transmission path, any relaying middle box, be it router or switch, should reserve a reasonably short queue for the packet flow of such flow to minimize delay.

When the receive buffer overflows the receiver discards the undelivered packet received first to free buffer space for the latest packet received. However it keeps order on delivering the packets to he ULA. ULA may choose to discard packets received earlier than some threshold.

The receiver SHOULD NOT make any acknowledgement to the packet received with the MIND flag set.

Minimal delay service is asymmetric in the sense that one transmission direction the data flow may be milk-like while in the reverse direction the data flow may be wine-like.

A minimal delay service data flow is terminated by ULA via some out-of-band control mechanism.

9. Graceful Shutdown

One participant of an FSP connection MAY initiate graceful shutdown of the connection if and only if its peer has committed the most recent transmit transaction.

By initiating graceful shutdown the participant tells its peer that current transmit transaction is to be committed as well.

9.1. Initiation of Graceful Shutdown

```
(RELEASE, FREWS, SN, ExpectedSN, ICC)
```

An FSP end node MAY initiate graceful shutdown if it is in the PEER_COMMIT, COMMITTING2 or CLOSABLE state. It SHALL NOT initiate graceful shutdown if its peer has not committed current transmit transaction.

Graceful shutdown is signaled to the remote end by sending a RELEASE command packet. The FSP end node SHALL migrate to the PRE_CLOSED state just before sending the RELEASE packet.

9.2. Acknowledgment of Graceful Shutdown

The RELEASE packet may be accepted in the COMMITTING, COMMITTED, COMMITTING2, CLOSABLE or PRE_CLOSED state.

If the legitimate RELEASE packet is received in the COMMITTING or COMMITTED state, the FSP end node SHALL buffer the RELEASE packet, wait each packet of the last transmit transaction of its peer has been received, deliver all the buffered payload and then migrate to the SHUT_REQUESTED state.

If the legitimate RELEASE packet is received in the COMMITTING2 or CLOSABLE state, the FSP end node SHALL migrate to the SHUT_REQUESTED state immediately.

In either of the two cases the receiver of the RELEASE packet SHALL acknowledge the sender of the RELEASE packet with a legitimate out-of-band ACK_FLUSH packet.

If the RELEASE packet is received in the PRE_CLOSED state, it is to finalize the graceful shutdown procedure.

9.3. Finalization of Graceful Shutdown

If either the legitimate RELEASE packet or the legitimate ACK_FLUSH packet is received in the PRE_CLOSED state the grace shutdown request is supposed to be acknowledged and the shutdown procedure SHALL be finalized by that the FSP end node migrates to the CLOSED state immediately.

In SHUT_REQUESTED state the FSP node SHALL migrate to CLOSED state immediately on the Shutdown API called by the ULA.

9.4. Retransmission of RELEASE Packet

The FSP end node in the PRE_CLOSED state SHALL retransmit the RELEASE packet until it migrates to CLOSED state or it is timed out.

As RELEASE is the in-band packet retransmission of the RELEASE packet is subjected to the normal retransmission rule.

10. Mobility and Multi-home Support

10.1. Heartbeat Signals

FSP requires that the participants periodically send the heartbeat signals.

The participant in the ACTIVE, COMMITTING, COMMITTED, PEER_COMMIT, COMMITING2 or CLOSABLE state MUST send the KEEP_ ALIVE packet as the heart-beat signal periodically to retain the connection in case that underlying IP address has changed.

```
(KEEP_ALIVE, FREWS, SN, ExpectedSN, ICC, Sink Parameter, SNACK)

(ACK_FLUSH, FREWS, SN, ExpectedSN, ICC, Sink Parameter, SNACK)
```

Heartbeat signal packet is an out-of-band control packet. It does not carry payload. The packet is either KEEP_ALIVE or ACK_FLUSH, depending on whether the peer's transmission transactin has been committed. The sequence number of the packet SHALL be set to the latest sequence number of all of the packets that have been sent.

Only the FSP node in the ACTIVE, COMMITTING, COMMITTED, PEER_COMMIT, COMMITING2 or CLOSABLE state MAY process the heartbeat signal.

In this version of FSP the heartbeat period is arbitrarily set to 600 seconds.

The sequence number (SN) of the heartbeat signal packet MUST equal the latest sequence number of the legitimate packets that have been sent.

The out-of-band serial number SHALL increase by one whenever a new hearbeat signal packet is sent.

10.2. Active Address Change Signaling

During communication process the FSP participant whose underlying IP address is changed SHOULD inform its peer such change by transmit a heartbeat signal packet so that the peer can retransmit the packets that were negatively acknowledged, if any.

Such informing hearbeat signal packet SHALL be sent in the ACTIVE, COMMITTING, COMMITTED, PEER_COMMIT, COMMITING2 or CLOSABLE state.

Informing heartbeat signal packet SHOULD be sent more frequently than a normal heartbeat signaling packet.

For this version of FSP informing heartbeat signal packet SHALL be retransmitted every 4 RTT interval until the heuristic acknowledgement is received.

10.3. Heuristic Remote Address Change Adaptation

A participant of the FSP connection SHALL set the source address of the packet to transmit or retransmit to new IP address as soon as the near-end IPv4 address or IPv6 network prefix has changed. The ULTID field MUST remain the same.

When a new packet with a later sequence number is received and the source IP address of the packet is found to be different with the preserved IP address of the remote end, the receiver SHOULD automatically update the preserved IP address of the remote end to the source IP address of the new packet, unless there is a Sink Parameter header in the packet.

If the sequence number of the packet received is not the latest in the receive window the preserved IP address of the remote end SHALL

NOT be updated even if the source address of the received packet has changed.

10.4. Heuristic Address Change Acknowledgement

The address change signaling heartbeat signal packet is supposed to be acknowledged if a packet targeted at the new IP address that the heartbeat signal packet has informed is received.

10.5. NAT-traversal and Multihoming

When FSP is implemented over UDP in the IPv4 network, each endpoint of the FSP connection is bound one and only one IPv4 address as soon as the connection is established. Each endpoint SHALL choose the source IPv4 address of the last packet received as the destination IPv4 address of the packet that it is to send later. By this mean FSP over UDP is NAT-friendly.

When FSP is implemented over IPv6 as soon as the connection is established the IPv6 address may be changed dynamically, and one more alternate IP address may be added or removed dynamically for individual endpoint as well, provided that ULTID part keeps unchanged while the host IDs part of all IPv6 address of the endpoint are of the same value at any given moment.

The sender may choose as the source IP address by selecting any network prefix that it has most-recently sent to its peer in the allowed address list field of the Sink Parameter header, joining with the host ID in the Sink Parameter header and the stable ULTID of the sender, and choose as the destination IP address by selecting any network prefix in the allowed address list field of the Sink Parameter header most-recently received from its peer, joining with the peer's host ID and the peer's ULTID. Thus multiple multi-homed paths MAY co-exist between the two FSP endpoints.

10.6. Explicit Multi-home Informing

If an FSP end node is configured with multiple IPv4 address other than the loop-back address, or with multiply global unicast IPv6 address, it MAY advertise multiple underlying addresses to the remote end by put them in the addressable network prefix list of the Sink Parameter extension header. The Sink Parameter extension header may be carried in the CONNECT_REQUEST, ACK_CONNECT_REQ, NULCOMMIT, MULTIPLY or KEEP_ALIVE packet.

Any participant of the communication SHALL NOT make discrimination of the source or destination IP address of any packet provided that both the source ULTID and the destination ULTID keep unchanged and the ICC field passes verification.

11. Connection Multiplication

Connection multiplication is the process of incarnating a new connection context by re-using security context of an established connection.

11.1. Request to Multiply Connection

```
(MULTIPLY, FREWS, SN, Salt, ICC [, Sink Parameter] [, payload])
```

The initiator's initial sequence number of the new connection is the sequence number of the packet that piggybacks the connection multiplication header. The ExpectedSN field of the normal packet store a Salt value instead.

The FREWS field MUST be processed in the new connection context while the ICC MUST be calculated with the session key of the original connection.

The new connection inherits the remaining key life. ULA SHOULD negotiate new session key and/or install new session key as soon as possible.

The optional payload of the MULTIPLY packet MUST be processed in the new connection context.

The MULTIPLY packet is an out-of-band command packet in the original connection context.

11.2. Response to Connection Multiplication Request

```
Case 1: (NULCOMMIT, FREWS, SN, ExpectedSN, ICC [, Sink
Parameter])

Case 2: (PERSIST, FREWS, SN, ExpectedSN, ICC, Payload)

Case 3: (RESET, Reason of Failure, SN, ExpectedSN, ICC)
```

In all of these cases the ULTID of the remote-end MUST be the value of the initiator's ULTID in the connection multiplication header.

It is REQUIRED that only a connection in the ESTABLISHED, COMMITTING, COMMITTED, PEER_COMMIT, COMMITTING2 or CLOSABLE state may accept a connection multiplication request.

In case 1 the responder admits the multiplication request AND commit the transmit transaction, the new connection enters into the PEER_COMMIT or CLOSABLE state immediately, on request of ULA.

In case 2 the responder admits the multiplication request and the new connection enters into the ESTABLISHED, PEER_COMMIT, COMMITTING or CLOSABLE state immediately, depending whether the ULA of the multiplication initiator has requested to commit the transmit transaction immediately and whether the ULA of the multiplication responder has requested to commit the transmit transaction in the reverse direction immediately.

In case 3 the responder rejects the multiplication request. To defend against spoofing attack ICC MUST be valid. The value of the SN field MUST equal the value of the 'Expected SN' field of the requesting MULTIPLY packet while the value of ExpectedSN field MUST equal the value of the 'Sequence No' field.

The new connection MUST derive new session key from the session key of the original connection where the out-of-band requesting MULTIPLY packet is received immediately.

11.3. Duplicate Detection of Connection Multiplication Request

Every time the responder of connection multiplication receives a MULTIPLY packet it MUST check the suggested responder's ULTID and the initiator's ULTID.

The responder MUST reject the multiplication request if the suggested responder's ULTID equals the near-end ULTID of some connection and the remote-end ULTID of that connection does not equal the initiator's ULTID.

The responder MUST recognize the MULTIPLY packet as a duplicate connection request if some connection matches the request and SHOULD response by retransmitting the head packet of the send queue of the matching connection, be it a PERSIST or a NULCOMMIT packet. A connection matches the MULTIPLY request if and only if the suggested responder's ULTID in the MULTIPLY packet equals the near-end ULTID of the connection and the initiator's ULTID equals the remote-end ULTID of the connection.

11.4. Retransmission

The initiating side SHALL retransmit the MULTIPLY packet if the corresponding PERSIST packet is not received in some limit time (by default 15 seconds).

11.5. Key Derivation for Branch Connection

Let K_out = BLAKE2(Km, [d] || Label || 0x00 || Context || L), where:

Km is the master key,

*[d] is one octet of integer Depth. It is alike the KDF counter mode as the NIST SP800-108. For this version of FSP it is the fixed number 1,

*Label is the fixed ASCII string "Multiply an FSP connection" which is 26-octet long for this version of FSP,

*Context is concatenation of two 32-bit words idB and idR idB is the ULTID allocated for the branch connection in the context of the multiplication initiator. idB is byte-order neutral. idR is the receiver side ULTID of the original connection that is to accept the connection multiplication request. idI or idR is byte-order neutral.

*L is a 32-bit network byte-order integer specifying the length in bits of the derived key K-out

12. Timeouts and Abrupt Close

12.1. Timeouts in End-to-End Negotiation

Initially the initiator is in the CONNECT_BOOTSTRAP state. It migrates to the CONNECT_ AFFIRMING state after it received the legitimate ACK_INIT_CONNECT packet. Then it migrates to the PEER_COMMIT or CLOSABLE state after it received the legitimate ACK_CONNECT _REQ packet, depending on the hint of ULA.

The responder incarnates a new connection context which is initially in the CHALLENGING state after accepting a legitimate Connect Request packet. Then it migrates to the COMMITTING or CLOSABLE state, depending on the packet received from its peer.

If the initiator or the responder is unable to migrate to a new state in some limit time (by default 60 seconds, except in LISTENING state) it aborts the connection by recycling the connection context.

12.2. Timeouts in Multiply

Initially the initiating side of Connection Multiplication is in the CLONING state. It migrates to the ACTIVE, COMMITTED, PEER_COMMIT or CLOSABLE state after it received the legitimate PERSIST packet. Which state to migrated depends on the EoT flag of the initiating MULTIPLY packet and the responding PERSIST packet.

If the initiating side is unable to migrate to a new state in some limit time (by default 60 seconds) it aborts multiplication by recycling the new connection context.

12.3. Timeout of Transmit Transaction Commitment

The FSP node MUST abort the connection if the time of no packet having arrived has exceed certain limit in the COMMITTING or COMMITTING2 state.

In this FSP version, timeout of transmit transaction commitment is set to 5 minutes.

12.4. Timeout of Graceful Shutdown

It simply migrates to the NON_EXISTENT pseudo-state if timeout in the PRE_CLOSED state.

In this FSP version, timeout of Graceful Shutdown is set to 1 minute.

12.5. Idle Timeout

If one participant has not received any packet nor has it sent any packet in some limit time, it MUST be abruptly closed.

In this FSP version the time limit, or the idle timeout, is set to 4 hours.

12.6. Session Key Timeout

For this FSP version if a secret key is applied for more than 2^30 times the FSP node MUST abruptly closed instantly.

12.7. Abrupt Close

An FSP node abruptly shutdown a session by sending a RESET packet and release all of the resource occupied by the the session immediately.

```
(RESET, Reason of Failure, SN, ExpectedSN, ICC)
```

13. Issues for Further Study

13.1. Resolution of ULTID in DNS

There are two patterns of IP address resolution in FSP: the DNS-compatible pattern and the proxy pattern. The former pattern relies on some name service to resolve the IP address of the responder for the initiator before they exchange end-to-end negotiation packets.

In the DNS-compatible pattern, the responder side of the FSP participants registered its address identifier, such as 'domain name' in some name service such as DNS [RFC1034][RFC1035], according to some pre-agreement at first. The initiator resolves the current IP address of the responder by consulting the name service, such as looking after the A or AAAA record [RFC3596] of the domain name in DNS.

If UDP over IPv4 is exploited as the under layer data packet delivery service the port number of the responder is firstly resolved just alike normal network application such as HTTP. Then it is extended to 32-bit ULTID, and ULTIDs of FSP can be considered as the superset of TCP port numbers.

If the string representation of IPv4/IPv6 address is applied directly as the peer's address identifier instead of the domain name there is no need for some real address resolution. But from the API caller's point of view it is a DNS-compatible mode address resolution.

13.2. Proxy Pattern for Syndicated Name Resolution

The proxy pattern of IP address resolution in FSP is to embed the address resolution information in the connection initialization packets and is designed to work in FSP over IPv6 mode only.

In IPv6 network the rightmost 32 bits of the IPv6 address directly maps to the ULTID so FSP does not need additional multiplexing mechanism such as port number. And it needs not consult SRV record or look for some entry in some 'services' file.

If the INIT_CONNECT packet carries the responder's host name it MUST take the link-local interface address as the source IPv6 address and the default link-local gateway address, FE80::1, as the destination IPv6 address no matter whether the global unicast IP address of the default gateway is configured. In such scenario the link-local gateway MUST be able to resolute the responder's host name to its global unicast IPv6 address, and the gateway MUST be able to map the initiator's link local address to its global unicast IPv6 address.

If the gateway that relays the INIT_CONNECT packet finds that the responder is on the same link-local network with the initiator it SHALL change the source and the destination IP addresses of the INIT_CONNECT packet to the link-local IP addresses of the initiator and the responder respectively, and relay the packet onto the same link-local network.

On receiving the INIT_CONNECT packet that carries the responder's host name the link-local gateway MUST resolute the responder's global unicast IPv6 address and map the initiator's global unicast IPv6 address, and replace the destination and source address of the INIT_CONNECT packet respectively, unless it finds that the initiator and the responder are on the same link-local network, where the gateway SHALL process the packet as stated in the previous statement.

13.3. Asymmetric Transmission

If there is one participant whose receive interface is not the same as the send interface the participant is called an asymmetric-transmission node.

Asymmetric transmission itself is asymmetric in the sense that one participant may be asymmetric-transmission node while its peer is a normal node that the send interface is the same receive interface.

An end node is asymmetric-transmission if it received an ACK_CONNECT_REQ packet, NULCOMMIT or PERSIST packet whose source IP address that the network interface accepting the packets reported is not in the allowed IP address list in the Sink Parameter header of the packet.

For an asymmetric-transmission remote end, the near end cannot rely on automatic IP address change detection. Instead IP address change notification mechanism shall be utilized.

However for this version of FSP asymmetric transmission support is optional.

14. Security Considerations

14.1. Deny of Service Attack

FSP is designed to mitigate effect of DoS attack by exploiting Cookie.

However, resistance against distributed DoS attack relies on external mechanism.

14.2. Replay Attack

In-band sequence number and out-of-band sequence number are exploited to resist against replay attack.

14.3. Passive Attacks

AEAD MAY be exploited by the ULA to protect it against passive attacks such as eavesdropping, gaining advantage by analyzing the data sent.

MAC only service MAY also be utilized. Together with application layer stream-mode encryption it protects the ULA against passive attacks as well.

14.4. Masquerade Attack

Both AEAD and MAC only service may be exploited to protect the endpoints against masquerade attack.

If proxy pattern for syndicated name resolution is exploited for FSP over IPv6, secure neighbor discovery [RFC3971] SHOULD be applied instead of common neighbor discovery whenever it is feasible.

14.5. Active Man-In-The-Middle Attack

The ULA SHALL take account to protect itself against MITM attack when making client authentication and key establishment.

14.6. Privacy Concerns

It is beneficial for privacy protection that the ULTID of each endpoints of an FSP connection is generated randomly [RFC7721].

15. IANA Considerations

It should be requested that the port number registered for UDP packets encapsulating FSP in the IPv4 network. The port number 18003 is exploited in the concept prototype implementation. The number is the decimal presentation of ASCII codes of the character 'F' ('x46') and 'S' ('x53') concatenated in network byte order.

It should be requested that the 'Next Header'/protocol number is assigned for FSP over IPv6. Decimal number 144 is exploited in the concept prototype implementation.

16. References

16.1. Normative References

[AES] NIST, "Advanced Encryption Standard (AES)", November 2001.

[CRC64] ECMA, "Data Interchange on 12.7 mm 48-Track Magnetic Tape Cartridges - DLT1 Format Standard, Annex B", December 1992.

[GCM] NIST, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", November 2007.

[LZ4] "LZ4: Extremely Fast Compression algorithm", <https://lz4.github.io/lz4/>.

[OSI_RM] ISO and IEC, "Information technology-Open Systems Interconnection - Basic Reference Model: The Basic Model", November 1994.

[R01] Rogaway, P., "Authenticated encryption with Associated Data", 2002.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2526] Johnson, D. and S. Deering, "Reserved IPv6 Subnet Anycast Addresses", RFC 2526, DOI 10.17487/RFC2526, March 1999, <https://www.rfc-editor.org/info/rfc2526>.

[RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, DOI 10.17487/RFC3124, June 2001, <https://www.rfc-editor.org/info/rfc3124>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC

3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, DOI 10.17487/RFC6887, April 2013, <https://www.rfc-editor.org/info/rfc6887>.

[RFC7693] Saarinen, M-J., Ed. and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI 10.17487/RFC7693, November 2015, <https://www.rfc-editor.org/info/rfc7693>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[RFC8273] Brzozowski, J. and G. Van de Velde, "Unique IPv6 Prefix per Host", RFC 8273, DOI 10.17487/RFC8273, December 2017, <https://www.rfc-editor.org/info/rfc8273>.

[STD5] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981, <https://www.rfc-editor.org/rfc/rfc791>.

[STD6] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

[STD7] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, <https://www.rfc-editor.org/rfc/rfc793>.

16.2. Informative References

[Gao2002]
Gao, J., "Fuzzy-layering and its suggestion", IETF Mail Archive, September 2002, <https://mailarchive.ietf.org/arch/msg/ietf/u-6i-6f-Etuvh80-SUuRbSCDTwg>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <https://www.rfc-editor.org/info/rfc3022>.

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", STD 88, RFC 3596, DOI 10.17487/RFC3596, October 2003, <https://www.rfc-editor.org/info/rfc3596>.

[RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <https://www.rfc-editor.org/info/rfc3971>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, DOI 10.17487/RFC4555, June 2006, <https://www.rfc-editor.org/info/rfc4555>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <https://www.rfc-editor.org/info/rfc4861>.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <https://www.rfc-editor.org/info/rfc4862>.

[RFC5942] Singh, H., Beebee, W., and E. Nordmark, "IPv6 Subnet Model: The Relationship between Links and Subnet

Prefixes", RFC 5942, DOI 10.17487/RFC5942, July 2010, <https://www.rfc-editor.org/info/rfc5942>.

[RFC6177] Narten, T., Huston, G., and L. Roberts, "IPv6 Address Assignment to End Sites", BCP 157, RFC 6177, DOI 10.17487/RFC6177, March 2011, <https://www.rfc-editor.org/info/rfc6177>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[RFC7050] Savolainen, T., Korhonen, J., and D. Wing, "Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis", RFC 7050, DOI 10.17487/RFC7050, November 2013, <https://www.rfc-editor.org/info/rfc7050>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

[RFC8415]
Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <https://www.rfc-editor.org/info/rfc8415>.

[RFC8504] Chown, T., Loughney, J., and T. Winters, "IPv6 Node Requirements", BCP 220, RFC 8504, DOI 10.17487/RFC8504, January 2019, <https://www.rfc-editor.org/info/rfc8504>.

Appendix A. Acknowledgements

Author's Address

Jun-an Gao
Beijing Capital Highway Development Group Co.,Ltd.
Shoufa Plaza-A, Liuliqiao South, Fengtai
Beijing
People's Republic of China

Email: jagao@outlook.com