International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Q.4102
(02/2022)

SERIES Q: SWITCHING AND SIGNALLING, AND ASSOCIATED MEASUREMENTS AND TESTS

Protocols and signalling for peer-to-peer communications

Hybrid peer-to-peer communications: Peer protocol

Recommendation ITU-T Q.4102

Recommendation ITU-T Q.4102

Hybrid peer-to-peer communications: Peer protocol

Summary

Recommendation ITU-T Q.4102 describes the peer protocol for communication among peers. The peer protocol enables peers to organize a tree-based overlay network in a hybrid peer-to-peer network and distribute data over the overlay network. This Recommendation also specifies the connection types among peers, resource elements types used in the message header, protocol messages exchanged among peers and information flows for describing behaviours of peer.

History

Edition	Recommendation	Approval	Study Group	Unique ID[*]
1.0	ITU-T Q.4102	2022-02-13	11	11.1002/1000/14923

Keywords

Hybrid peer-to-peer, peer protocol, tree-based overlay network.

[*] To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

Table of Contents

Recommendation ITU-T Q.4102

Hybrid peer-to-peer communications: Peer protocol

1 Scope

This Recommendation describes peer protocol for tree-based overlay network in hybrid peer-to-peer (HP2P) communications as follows:

– connection types between peers;

– resource elements of protocol messages;

– protocol messages and its parameters;

– information flows for describing behaviours of peer.

2 References

The following ITU-T Recommendations and other references contain provisions that, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T Q.4100]	Recommendation ITU-T Q.4100 (2020), *Hybrid peer-to-peer communications: Functional architecture*.
[ITU-T Q.4101]	Recommendation ITU-T Q.4101 (2021), *Hybrid peer-to-peer communications: Tree and data recovery procedures*.
[ITU-T Q.4103]	Recommendation ITU-T Q.4103, *Hybrid peer-to-peer (communications: Overlay management protocol*.
[IETF RFC 7159]	IETF RFC 7159 (2014), *The JavaScript Object Notation (JSON) Data Interchange Format*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 buffermap [b-ITU-T X.609]: A map showing the downloading status of fragments comprising a shared content.

3.1.2 corresponding peer [b-ITU-T X.609.4]: A peer that performs an operation on the request received from the requesting peer.

3.1.3 hybrid overlay network [ITU-T Q.4100]: A peer-to-peer overlay network in which participating peers exchange data using the pull and push method. The hybrid overlay network also provides a way to organize and maintain a tree-style path for pushing data to all peers without loops, as well as fetching data from other peers simultaneously.

3.1.4 overlay network [b-ITU-T X.1162]: An overlay network is a virtual network that runs on top of another network. Like any other network, the overlay network comprises a set of nodes and links between them. Because the links are logical ones, they may correspond to many physical links of the underlying network.

3.1.5 peer [b-ITU-T X.1161]: Communication node on P2P network that functions simultaneously as both "client" and "server" to the other nodes on the network.

3.1.6 peer-to-peer (P2P) [b-ITU-T Y.2206]: A system is considered to be P2P if the nodes of the system share their resources in order to provide the service the system supports. The nodes in the system both provide services to other nodes and request services from other nodes.

NOTE – Peer is the node in a P2P system.

3.1.7 requesting peer [b-ITU-T X.609.4]: A peer that sends a request to the other corresponding peer.

3.2 Terms defined in this Recommendation

None.

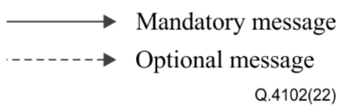
4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

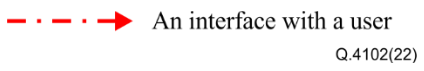
HOMS	Hybrid Overlay Management Server
HP2P	Hybrid Peer-to-Peer
NTP	Network Time Protocol
P2P	Peer-to-Peer

5 Conventions

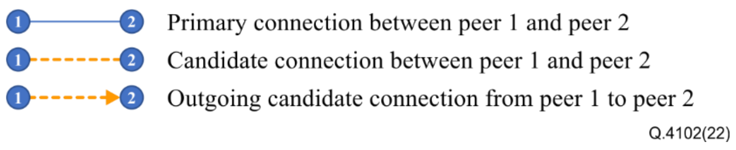
In this Recommendation, message flows among peers to describe the operation of each message and behaviours of peer protocol are drawn with solid arrow lines or dashed arrow lines. The solid arrow line indicates a mandatory message and the dashed arrow line indicates an optional message.



An interface with a user is drawn with a dot dashed line.



The connection between peers is presented by solid lines or dashed lines. A solid line between peers is a primary connection and a dashed line between peers is a candidate connection. A dashed arrow line between peers indicates an outgoing or incoming candidate connection.


1
2
Primary connection between peer 1 and peer 2
1
2
Candidate connection between peer 1 and peer 2
1
2
Outgoing candidate connection from peer 1 to peer 2
Q.4102(22)

In the response message under clause 7.2, *rsp-code* is a combination of the request type for classifying the type of request and the response type for classifying the type of response. For example, *rsp-code* will be 2200 if the request type is "2" and the response type is "200". However, figures under clause 7.2 do not show the exact value of *rsp-code* for ease of understanding about the meaning of the response.

Resource elements in clause 7.1 and fields of message header and extension under clause 7.2 are encoded in JavaScript object notation (JSON) [IETF RFC 7159], and the grammar used in representing objects defined in this Recommendation is as follows:

– "NUMBER", "STRING", "BOOLEAN" and "LIST" are types used to indicate number, string, boolean and list respectively;
– An array of collective values are enclosed in brackets "[ ]" with values separated by commas ",".

6 Connection types

This clause describes the connections between peers. The concept of a path (primary path / secondary path) described in [ITU-T Q.4100] is defined as a connection in this Recommendation. These connections between peers in a tree-based overlay network are classified as follows:

– **Primary connection**: This connection is established for data broadcasting between peers. A peer in a tree-based overlay network has one or more primary connections for sending and receiving data from other peers. However, it maintains only one primary connection between two identical peers.
– **Candidate connection**: This connection is not used for data broadcasting between peers, but it is established in advance to restore fast in loss of primary connection. When a peer has a problem to send and receive data through the primary connection, it is necessary to replace the old primary connection with the new primary connection immediately. Therefore, when a peer detects a loss of a primary connection, it switches one of the candidate connections to a new primary connection.

7 Resource element types and messages of peer protocol

7.1 Resource elements

This clause specifies the basic resource elements to be used in messages to convey information regarding the operations of peer.

7.1.1 PEER_BUFFERMAP

All peers maintain a caching buffer list for data received from other peers, which consists of a set of caching buffers in the form of a circular queue for each data source. The PEER_BUFFERMAP element includes detailed information about caching buffer list in Table 7-1.

Table 7-1 – PEER_BUFFERMAP resource element

Keyword	Type	Description
buffmaplist	LIST[BUFFERMAP]	*buffmaplist* includes the list of buffermap information such as source peer id and sequence of data packets described in [ITU-T Q.4101].

7.1.2 BUFFERMAP

The BUFFERMAP element includes detailed information on the caching buffer such as an identifier of source peer and a sequence list of data packets information received from source peer in Table 7-2.

Table 7-2 – BUFFERMAP resource element

Keyword	Type	Description
source-peer-id	STRING	*source-peer-id* indicates the identifier of the peer who has sent the BROADCAST_DATA request message.
sequence-list	LIST[NUMBER]	*Sequence-list* includes the list of sequence values in the data packet received through the BROADCAST_DATA request message.

7.2 Messages

This clause describes the messages exchanged between peers to be used in the reference point R2 that is specified in [ITU-T Q.4100] and the header of messages is represented as JavaScript object notation (JSON) [IETF RFC 7159].

Figure 7-1 shows the format and length of the message.

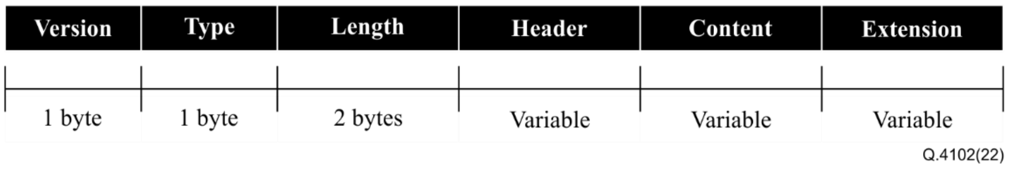


Figure 7-1 – Message format of peer protocol

- *version*: this field means the version of the protocol, and the current version is assigned as 0x01.
- *type*: this field means the encoding type of the header, and 0x01 and 0x10 are assigned as type values. The current version is assigned as 0x01. In this case, 0x01 is defined as a text type and 0x10 is defined as a binary encoding type.
- *length*: this field means the length of the header field.
- *header*: this field contains a code that distinguishes whether a message is a request or a response and the necessary parameters for the message code.
- *content*: this field contains content data. The length of the content can be checked through the length value of the *payload* field included in the header.
- *extension*: this field contains private purpose parameters.

The following subclauses describe the message flow between peers, the peer operation related to the message, and the detailed format of the header part of each message.

7.2.1 HELLO_PEER

A peer can initiate overlay network join by exchange of HELLO_PEER request and response message with other peers belonging to the overlay network. Figure 7-2 shows the message flow of HELLO_PEER.

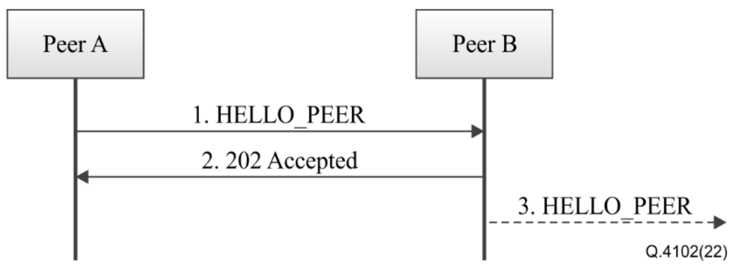


Figure 7-2 – Message flow of HELLO_PEER

① When a new peer A wants to join an overlay network, it sends a HELLO_PEER request message to one of the peers (peer B in Figure 7-2) participating in the overlay network.

The list of peers participating in the overlay network is obtained from a hybrid overlay management server (HOMS). At this time, the peer sending a HELLO_PEER request message acts as a requesting peer and the responding peer that interacts with requesting peer acts as a corresponding peer.

② Peer B receiving the HELLO_PEER request message firstly checks whether it is acceptable and then processes it if possible. Peer B can then send a 202 (accepted) response which indicates it has received the HELLO_PEER request message.

③ Peer B transfers the received HELLO_PEER request message to the acceptable peers connected to their primary connection.

This is determined by considering the maximum number of requesting connections. The maximum number of requesting connections is the same as the value of the *conn_num* field within the HELLO_PEER request message, or 1 is reduced here. The latter is when peer B handles the HELLO_PEER request message.

Peer B forwards the HELLO_PEER request message to the peers with a primary connection relationship if the maximum number of requesting connections is greater than 0. Peer B divides the maximum number of requesting connections by the number of peers in the primary connection relationship and then forwards the HELLO_PEER request message to each peer in the primary connection relationship.

Table 7-3 describes the syntax and semantics of each field in the header of a HELLO_PEER request message.

Table 7-3 – Header format of HELLO_PEER request

Syntax	Description

{
    "req-code ": NUMBER,
    "req-params": {
        "operation": {
            "overlay-id": STRING,
            "conn_num": NUMBER,
            "ttl": NUMBER,
            "recovery": BOOLEAN
        },
        "peer": {
            "peer-id": STRING,
	– *req-code* indicates the type of the request message. In case of a HELLO_PEER request message, *req-code* value is set as 1.
	– *overlay-id* indicates an identifier of the overlay network that the new peer wants to participate.
	– *conn_num* indicates the number of connections that the new peer wants to establish with other peers.
	– *ttl* indicates the number of steps of the message the peer wants to pass on to. The new participating peer specifies the initial ttl value, and the peer receiving the HELLO_PEER message reduces the ttl value by one when forwarding the message.

Table 7-3 – Header format of HELLO_PEER request

Syntax	Description
"address": STRING, "ticket-id": NUMBER } } }	– *recovery* is used when an emergency recovery is needed. If a new peer joins the overlay network for the first time, this field is not used. – *peer-id* indicates the identifier of the new participating peer. – *address* indicates network address information of new participating peer that can be accessed by other peers. – *ticket-id* indicates the ticket-id value for the new participating peer. *ticket-id*, a value that indicates the order of peer joined to the hybrid overlay network, which is taken from the HOMS defined in [ITU-T Q.4100]. This means a peer with a low ticket-id value has first joined the corresponding overlay network. When recovering from an overlay network problem, a ticket ID plays an important role in preventing a loop by causing the peer with a large ticket ID value to attempt recovery of a broken connection.

Table 7-4 describes the syntax and semantics of each field in the header of a HELLO_PEER response message.

Table 7-4 – Header format of HELLO_PEER response

Syntax	Description

{
"rsp-code": NUMBER
}
	– *rsp-code* indicates which response message is for which request. The *rsp-code* is a combination of the request type in classifying the type of request and the response type for classifying the type of response. The request type is defined in the *req-code* in the request message. In case of a HELLO_PEER response message, *rsp-code* value is set as 1202 if the result of the request is successful.
	• *rsp-code* value 1202 is a combination of the request type (1) and response type (202). Response type 202 means 'accepted'. Therefore, *rsp-code* value 1202 means an 'accepted' response to the HELLO_PEER request message.

7.2.2 ESTAB_PEER

A candidate connection can be established by exchange of an ESTAB_PEER request and response messages. An ESTAB_PEER request message is used by a peer which received a HELLO_PEER request message from a newly joining peer to directly connect to the new peer that wants to join. Figure 7-3 shows the message flows of ESTAB_PEER messages.

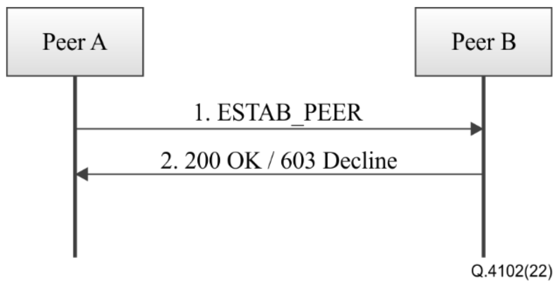


Figure 7-3 – Message flow of ESTAB_PEER

① Peer A sends an ESTAB_PEER request message to peer B.

When peer A receives the HELLO_PEER request message, peer A firstly checks if it has room in its network connection capability. If peer A is available, an ESTAB_PEER request message is used for establishing connection to the peer's address of HELLO_PEER request message. At this time, peer A acts as a requesting peer and peer B acts as a corresponding peer for the ESTAB_PEER request message.

② Upon receipt of the ESTAB_PEER request message, peer B can send a 200 (OK) or 603 (decline) response.

The peer B sends a 200 response if peer B accepts a new connection through the ESTAB_PEER request message, or 603 if it refuses. At this time, if peer A receives the 200 response on the ESTAB_PEER request message, a candidate connection between the two peers is established.

If peer B has received ESTAB_PEER request messages beyond ESTAB_PEER_TIMEOUT, peer B sends a 603 response. ESTAB_PEER_TIMEOUT is the maximum waiting time for receiving ESTAB_PEER request messages from other peers.

Table 7-5 describes the syntax and semantics of each field in the header of an ESTAB_PEER request message.

Table 7-5 – Header format of ESTAB_PEER request

Syntax	Description

{
  "req-code": NUMBER,
  "req-params": {
    "operation": {
      "overlay-id": STRING
    },
    "peer": {
      "peer-id": STRING,
      "ticket-id": NUMBER
    }
  }
}
	– *req-code* indicates the type of the request message. In case of an ESTAB_PEER request message, *req-code* value is set as 2.
	– *overlay-id* indicates the identifier of an overlay network.
	– *peer-id* indicates the identifier of the peer sending an ESTAB_PEER request.
	– *ticket-id* indicates the ticket-id value of the peer sending the ESTAB_PEER request message. *ticket-id* is assigned from the HOMS through the HybridOverlayJoin message defined in [ITU-T Q.4103].

Table 7-6 describes the syntax and semantics of each field in the header of an ESTAB_PEER response message.

Table 7-6 – Header format of ESTAB_PEER response

Syntax	Description

{
"rsp-code": NUMBER
}
	– *rsp-code* indicates which response message is for which request. In case of an ESTAB_PEER response message, it is possible to have the 2200 or 2603 as a *rsp-code* value.
	• *rsp-code* value 2200 is a combination of the request type (2) and response type (200). Response type 200 means 'OK'. Therefore, *rsp-code* value 2200 means an 'OK' response to the ESTAB_PEER request message.
	• *rsp-code* value 2603 is a combination of the request type (2) and the response type (603). Response type 603 means 'decline'. Therefore, *rsp-code* value 2603 is a 'decline' response to the ESTAB_PEER request message.

7.2.3 PROBE_PEER

Network distance between two peers in a candidate connection relationship can be calculated by the exchange of the PROBE_PEER request and response messages. When a new peer that wants to join the overlay can establish multiple candidate connections through HELLO_PEER and ESTAB_PEER request messages, a peer sends a PROBE_PEER request message that initiates checking the network distance for selecting the primary connection. Figure 7-4 shows the message flow of a PROBE_PEER.

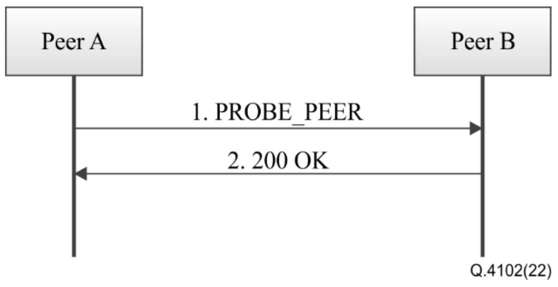


Figure 7-4 – Messages flow of PROBE_PEER

① Peer A sends a PROBE_PEER request message with a network time protocol (NTP) timestamp information to peer B, and peer B answers the 200 response message with a NTP timestamp information.

② Peer A who receives a 200 response message to the PROBE_PEER request message uses the NTP timestamp information to calculate the network cost between corresponding peers. This process is repeated for each corresponding peer associated with the requesting peer.

Table 7-7 describes the syntax and semantics of each field in the header of a PROBE_PEER request message.

Table 7-7 – Header format of PROBE_PEER request

Syntax	Description

{
    "req-code": NUMBER,
    "req-params": {
        "operation": {
            "ntp-time": STRING
        }
    }
}
	– *req-code* indicates the type of the request message. In case of a PROBE_PEER request message, *req-code* value is set as 3.
	– *ntp-time* indicates the NTP timestamp or local time of a peer.

Table 7-8 describes the syntax and semantics of each field in the header of a PROBE_PEER response message.

Table 7-8 – Header format of PROBE_PEER response

Syntax	Description

{
    "rsp-code": NUMBER,
    "rsp-params": {
        "operation": {
            "ntp-time": STRING
        }
    }
}
	– *rsp-code* indicates which response message is for which request. In case of a PROBE _PEER response message, *rsp-code* value is set as 3200.
	• *rsp-code* value 3200 is a combination of the request type (3) and response type (200). Therefore, *rsp-code* value 3200 is an 'OK' response to the PROBE_PEER request message.
	– Time value of this *ntp-time* is used as the same one as the *ntp-time* within the PROBE_PEER request message (Table 7-7). The peer receiving this response checks the network distance by comparing it with its own NTP timestamp or local timestamp value.

7.2.4 SET_PRIMARY

A primary connection can be established by exchange of a SET_PRIMARY request and response messages. A SET_PRIMARY request message is used when the peer selects one of the candidate connections established through the ESTAB_PEER message as the primary connection. Peers wishing to join the overlay network can send SET_PRIMARY request messages to the peer with the best network distance through HELLO_PEER, ESTAB_PEER, and PROBE_PEER messages. Figure 7-5 shows the message flow of a SET_PRIMARY.

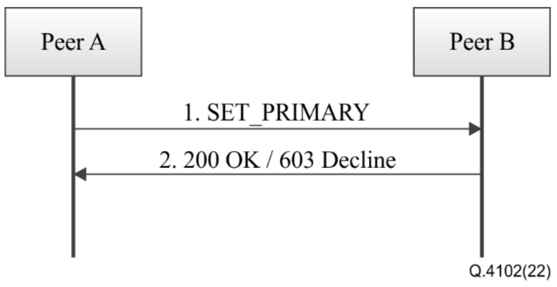


Figure 7-5 – Message flow of SET_PRIMARY

① Peer A sends a SET_PRIMARY request message with its own PEER_BUFFERMAP to peer B. PEER_BUFFERMAP expresses its own caching buffer information.

Peer A can process as follows according to its response to the SET_PRIMARY request message.

Firstly, if the requesting peer receives a 200 response to the SET_PRIMARY request message, the primary connection is established, and data can be transmitted from the overlay network.

The requesting peer compares its own buffer map with the buffer map of the corresponding peer, and if there is data that the requesting peer does not have, the requesting peer can request the data from the corresponding peer.

Secondly, if the requesting peer receives a 603 response to the SET_PRIMARY request message, the requesting peer selects one among the other peers and tries again.

② Upon receipt of the SET_PRIMARY request message, the corresponding peer B can respond in two ways.

First, if the primary connection setting is possible, the corresponding peer answers a 200 response to the SET_PRIMARY request message after internal processing with the primary connection setting. At this time, the 200 response to the SET_PRIMARY request message includes PEER_BUFFERMAP representing the state of its own caching buffer. The corresponding peer compares its own buffer map with the requesting peer (peer A in Figure 7-5)'s buffer map, and if there is data that the corresponding peer does not have, it can request data from the requesting peer.

Second, if the primary connection cannot be established due to an internal problem, the corresponding peer answers the 603 response message to the SET_PRIMARY request message.

Table 7-9 describes the syntax and semantics of each field in the header of a SET_PRIMARY request message.

Table 7-9 – Header format of SET_PRIMARY request

Syntax	Description

{
    "req-code": NUMBER,
    "req-params": {
        "buffermap": PEER_BUFFERMAP
        }
    }
}
	– *req-code* indicates the type of the request message. In case of a SET_PRIMARY request message, *req-code* value is set as 4.
	– *buffermap* indicates the caching buffer status of requesting peer.

Table 7-10 describes the syntax and semantics of each field in the header of a SET_PRIMARY response message.

Table 7-10 – Header format of SET_PRIMARY response

Syntax	Description

{
    "rsp-code": NUMBER,
    "rsp-params": {
        "buffermap": PEER_BUFFERMAP
    }
}
	– *rsp-code* indicates which response message is for which request. In case of a SET_PRIMARY response message, it is possible to have the 4200 or 4603 as a *rsp-code* value.
	– *rsp-code* value 4200 is a combination of the request type (4) and response type (200). Therefore, *rsp-code* value 4200 means an 'OK' response to the SET_PRIMARY request message.
	– *rsp-code* value 4603 is a combination of the request type (4) and response type (603). Therefore, *rsp-code* value 4603 is a 'decline' response to the SET_PRIMARY request message. Response type 603 is set when the corresponding peer receiving the SET_PRIMARY request cannot establish a primary connection with the requesting peer due to internal problems.
	– *buffermap* indicates the caching buffer status of the corresponding peer.

7.2.5 SET_CANDIDATE

A candidate connection relationship can optionally be confirmed by exchange of SET_CANDIDATE request and response messages. A SET_CANDIDATE request message is optionally used by a peer willing to express a candidate connection relationship to other peers. After a new peer generates multiple candidate connections from different peers via ESTAB_PEER messages and checks the network distance via PROBE_PEER messages, the new peer sends SET_PRIMARY request messages to the peer with the best network distance. The new peer can also send SET_CANDIDATE request messages to the other peers to confirm the relationship among them. Maintaining the candidate connection is intended for quick switchover when the primary connection has a problem. Figure 7-6 shows the message flow of SET_CANDIDATE.

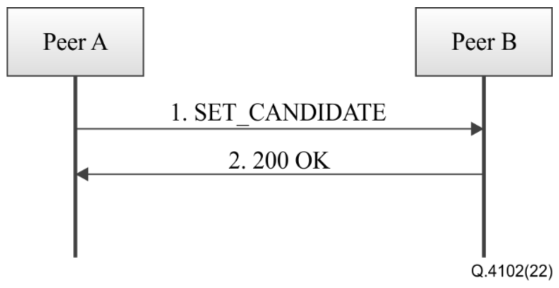


Figure 7-6 – Message flow of SET_CANDIDATE

① Peer A sends a SET_CANDIDATE request message to peer B. In this case, a requesting peer sending the SET_CANDIDATE request message considers the corresponding peer as 'outgoing candidate'. On the contrary, a corresponding peer considers the requesting peer as an 'incoming candidate'.

② Upon receipt of the SET_CANDIDATE request message, the corresponding peer sends a 200 response to the SET_CANDIDATE request message after internal processing with the candidate connection setting.

Table 7-11 describes the syntax and semantics of each field in the header of a SET_CANDIDATE request message.

Table 7-11 – Header format of SET_CANDIDATE request

Syntax	**Description**

{
    "req-code": NUMBER
}
	– *req-code* indicates the type of the request message. In case of a SET_CANDIDATE request message, *req-code* value is set as 5.

Table 7-12 describes the syntax and semantics of each field in the header of a SET_CANDIDATE response message.

Table 7-12 – Header format of SET_CANDIDATE response

Syntax	**Description**

{
    "rsp-code": NUMBER
}
	– *rsp-code* indicates the code value of the response message. In case of a SET_CANDIDATE response message, *rsp-code* value is set as 5200.
	• *rsp-code* value 5200 is a combination of the request type (5) and response type (200). Therefore, *rsp-code* value 5200 is an 'OK' response to the SET_CANDIDATE request message.

7.2.6 BROADCAST_DATA

Data can be delivered by exchange of BROADCAST_DATA request and response messages. A BROADCAST_DATA message is used by a peer to broadcast data to other peers in a primary connection relationship. Both binary and text are possible for the data to be transmitted. The peer receiving this message forwards it to other peers that have a primary connection with it. Figure 7-7 shows the message flow of BROADCAST_DATA.

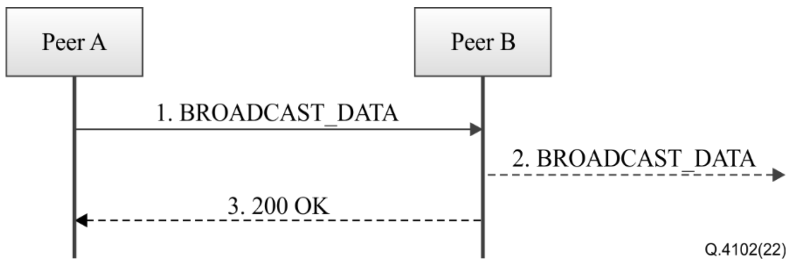


Figure 7-7 – Message flow of BROADCAST_DATA

① In order to deliver data over an overlay network, peer A sends a BROADCAST_DATA request message to peer B with the length of the content data, the type of content data in the *payload* field, and the data it wants to deliver to peer B (the peers associated with the primary connection).

② Peer B receiving the BROADCAST_DATA request message delivers the received BROADCAST_DATA request message to other peers in the primary connection except for the requesting peer of the BROADCAST_DATA request message. This BROADCAST_DATA request message is delivered until it reaches the leaf node of the tree.

③ Upon receipt of the BROADCAST_DATA request message, peer B sends a 200 response when the *ack* field of the BROADCAST_DATA request message is true.

Peer ID is included in this BROADCAST_DATA request message to identify who created and sent this data for the first time. *Payload* field within the BROADCAST_DATA request message is composed of length and content-type which contains the information about the data that will be transmitted.

Table 7-13 describes the syntax and semantics of each field in the header of a BROADCAST_DATA request message.

Table 7-13 – Header format of BROADCAST_DATA request

Syntax	Description

{
    "req-code": NUMBER,
    "req-params": {
        "operation": {
            "ack": BOOLEAN
        },
        "peer": {
            "peer-id": STRING,
            "sequence": NUMBER
        },
        "payload": {
            "length": NUMBER,
            "content-type": STRING
        }
    }
}
	– *req-code* indicates the type of the request message. In case of the BROADCAST_DATA request message, *req-code* value is set as 6.
	– *ack* has a TRUE value if requesting peer wants to receive a response message. Default value is FALSE.
	– *peer-id* indicates the ID of the peer that initially generated this BROADCAST_DATA request message, and it is possible to identify who created and sent this data for the first time.
	– Whenever self-generated data is sent through the BROADCAST_DATA message, the value of the s*equence* field is increased by 1 and allocated. When forwarding data is received from another peer, this value of the *sequence* field is not modified.
	– *length* indicates the length of the data that is carried and is described in bytes.

Table 7-13 – Header format of BROADCAST_DATA request

Syntax	Description
	– *content-type* indicates the kind of data that is carried. It is described as the MIME type such as application/JSON, application/XML, etc. The processing of content according to content-type depends on the application using the peer protocol.

The BROADCAST_DATA request message shall use the extension field to specify an application specific parameter.

Table 7-14 describes the syntax and semantics of each field in the extension of the BROADCAST_DATA request message.

Table 7-14 – Extension format of BROADCAST_DATA request

Syntax	Description
{ "app-id": STRING }	– *app-id* indicates the identifier of the application using the peer protocol

Table 7-15 describes the syntax and semantics of each field in the header of the BROADCAST_DATA response message.

Table 7-15 – Header format of BROADCAST_DATA response

Syntax	Description
{ "rsp-code": NUMBER }	– *rsp-code* indicates which response message is for which request. In case of a BROADCAST_DATA response message, *rsp-code* value is set as 6200. • *rsp-code* value 6200 is a combination of the request type (6) and the response type (200). Therefore, *rsp-code* value 6200 is an 'OK' response to the BROADCAST_DATA request message.

7.2.7 HEARTBEAT

The status of a connection between two peers can be checked by exchange of HEARTBEAT request and response messages. The peer can check the connection status of each other between peers. HEARTBEAT request message is periodically used to inform whether two peers are connected to each other or not. Figure 7-8 shows the message flow of HEARTBEAT.

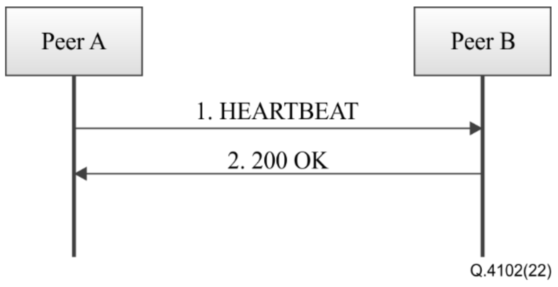


Figure 7-8 – Message flow of HEARTBEAT

① Peer A generates a HEARTBEAT request message to inform that the connection with a specific peer is maintained, and sends it to peer B, which wants to notify that the connection is maintained.

② Upon receipt of the HEARTBEAT request message, peer B sends a 200 response to the HEARTBEAT request message after internal processing.

Table 7-16 describes the syntax and semantics of each field in the header of the HEARTBEAT request message.

Table 7-16 – Header format of HEARTBEAT request

Syntax	Description

{
"req-code": NUMBER
}
	– *req-code* indicates the type of the request message. In case of a HEARTBEAT request message, *req-code* value is set as 7.

Table 7-17 describes the syntax and semantics of each field in the header of the HEARTBEAT response message.

Table 7-17 – Header format of HEARTBEAT response

Syntax	Description

{
"rsp-code": NUMBER
}
	– *rsp-code* indicates which response message is for which request. In case of the HEARTBEAT response message, *rsp-code* value is set as 7200.
	• *rsp-code* value 7200 is a combination of the request type (7) and the response type (200). Therefore, *rsp-code* value 7200 is an 'OK' response to the HEARTBEAT request message.

7.2.8 SCAN_TREE

The structure of an overlay network can be identified by the exchange of SCAN_TREE request and response messages. The peer can scan connection features among peers with the primary connection relationship. The SCAN_TREE request message is used to initiate the scan of the connection shape of peers connected to the primary connection. Figure 7-9 shows the message flow of SCAN_TREE.

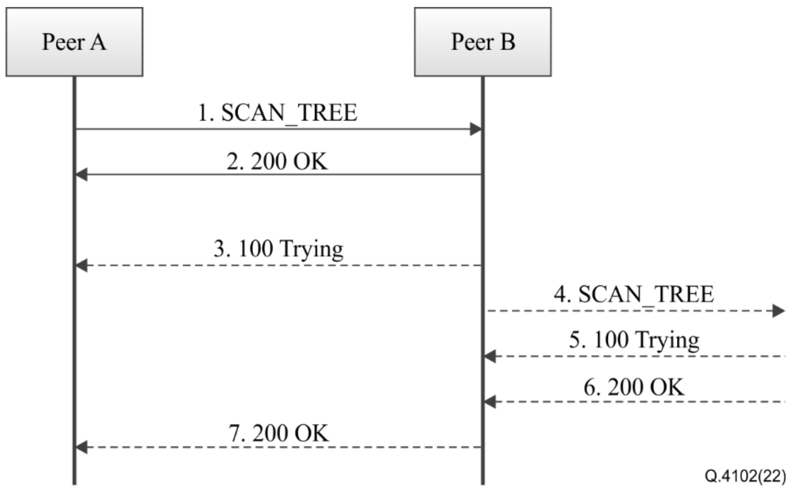


Figure 7-9 – Message flow of SCAN_TREE

① Requesting peer A generates and sends a SCAN_TREE request message to peer B.

At this time, a random digit number is added in the *cseq* field within the SCAN_TREE message. Also, peer information (peer-id, address) on the requesting peer is added in the *via* field within the SCAN_TREE message. *Via* field is a listed set of peer information that is composed of a peer-id and an address. New peer information is added in the first location of the listed set (i.e., start point of the list).

② If peer B is a leaf node of the tree, corresponding peer B sends a 200 response message to the requesting peer A.

The corresponding peer that has received the SCAN_TREE request message can be an intermediate node or a leaf node of a tree. If the corresponding peer that receives the SCAN_TREE request message has a peer relationship with another peer (except the requesting peer), that peer is an intermediate node of a tree. Otherwise, the corresponding peer is a leaf node of a tree.

At this time, peer information (peer-id, address) on the corresponding peer is added in the *path* field within the 200 response message. *Path* field is a listed set of peer information that is composed of a peer-id and an address. The information on the corresponding peer that has received the SCAN_TREE request message is added in the last location of the listed set (i.e., the end point of the list).

The destination of the 200 response message is the first address of the *via* field within SCAN_TREE (i.e., start point of the list).

③ If peer B is an intermediate node of a tree, corresponding peer B sends a 100 (trying) response message to the requesting peer A.

④ If peer B is an intermediate node of a tree, it sends the SCAN_TREE message to the other peers with a primary connection relationship (except the requesting peer).

At this time, the transferred SCAN_TREE request message includes its peer information in the *via* field. The forwarded SCAN_TREE request message contains the addition of its own peer information in the first location *via* field within the received original SCAN_TREE request message. A new generated SCAN_TREE request message is sent to other peers with a primary connection relationship (except the requesting peer). The new generated SCAN_TREE request message does not route to the requesting peer to prevent a loop.

⑤ If peer B has received a 100 response message from another peer with a primary connection relationship, the requesting peer B waits for the 200 response message after internal processing.

⑥ If peer B has received a 200 response message from another peer with a primary connection relationship, peer B sends the 200 response message to the requesting peer A.

At this time, peer B checks if the first (begin) address information *via* field matches its address information. If it matches, peer B is the original requesting peer, otherwise, peer B is the intermediate requesting peer

If the requesting peer is the intermediate requesting peer (i.e., matching), the requesting peer moves the top address information *via* field to the top address of the *path* field within the 200 response, and then transfers the changed 200 response to the top address of the *via* field. Each peer should operate stateless so that they can be transferred up and down without managing the previous history or state of request and response message.

⑦ Peer A, who has received the 200 response message, performs the internal processing to express the relationship of peers constituting the tree.

If the requesting peer is the original requesting peer, the requesting peer can identify the composition and the order of the peers in the branch that make up the tree. This can be determined from a listed set of peer information contained in a *path* field of the 200 response.

An original requesting peer that has created a SCAN_TREE request message can receive multiple 200 responses.

Table 7-18 describes the syntax and semantics of each field in the header of the SCAN_TREE request message.

Table 7-18 – Header format of SCAN_TREE request

Syntax	Description

{
    "req-code": NUMBER,
    "req-params": {
        "cseq": NUMBER,
        "overlay": {
            "overlay-id": STRING,
            "via": LIST[STRING]
        },
        "peer": {
            "peer-id": STRING,
            "address": STRING,
            "ticket-id": NUMBER
        }
    }
}
	– *req-code* indicates the type of the request message. In case of a SCAN_TREE request message, the *req-code* value is set as 8.
	– *cseq is* a command sequence number. This value is created at random by the peer that sends the SCAN_TREE message for the first time.
	– *overlay-id* indicates the identifier of the overlay network.
	– *via* indicates a listed set of peer information on the path from which the message has been delivered.
	– *peer-id* indicates the identifier for the requesting peer of the SCAN_TREE message.
	– *address* indicates the IP address information for the requesting peer of the SCAN_TREE message.
	– *ticket-id* indicates the ticket-id value for the requesting peer of the SCAN_TREE message.

Table 7-19 describes the syntax and semantics of each field in the header of the SCAN_TREE response message.

Table 7-19 – Header format of SCAN_TREE response

Syntax	Description

{
    "rsp-code": NUMBER,
    "rsp-params": {
	– *rsp-code* indicates which response message is for which request. In case of a SCAN_TREE response message, it is possible to have the 8200 or 8100 as an *rsp-code* value.

```
        "cseq": NUMBER,
        "overlay": {
            "overlay-id": STRING,
            "via": LIST[STRING],
            "path": LIST[STRING]
        },
        "peer": {
            "peer-id": STRING,
            "address": STRING,
            "ticket-id": NUMBER
        }
    }
    }
}
```
	• When the corresponding peer receiving the SCAN_TREE request message is a leaf node, 8200 is used. *rsp-code* value 8200 is a combination of the request type (8) and the response type (200). Therefore, *rsp-code* value 8200 is an 'OK' response to the SCAN_TREE request message.
	• When the corresponding peer receiving the SCAN_TREE request message is not a leaf node, 8100 is used. *rsp-code* value 8100 is a combination of the request type (8) and the response type (100). Code value 100 means 'trying'. Therefore, *rsp-code* value 8100 is a 'trying' response to the SCAN_TREE request message.
	– *cseq* is a command sequence number. The *cseq* value in the SCAN_TREE request message is copied as it is.
	– *ovelay-id* indicates the identifier of the overlay network. The *overlay-id* value in the SCAN_TREE request message is copied as it is.
	– *via* indicates a listed set of peer information on the path from which the message has been delivered.
	path indicates a listed set of peer information on the path taken by the response message so far.
	– *peer-id* indicates the identifier for the requesting peer of SCAN_TREE request message.
	– *address* indicates the IP address information for the requesting peer of the SCAN_TREE request message.
	– *ticket-id* indicates the ticket-id value for the requesting peer of a SCAN_TREE request message.

7.2.9 RELEASE_PEER

A peer can gracefully exit from the overlay network currently participating by exchange of RELEASE_PEER request and response messages. A RELEASE_PEER request message is used when a peer no longer wants to participate in the overlay network currently participating. Requesting peer explicitly sends this message to the peer that wants to terminate the peer-to-peer connection. Figure 7-10 shows the message flows of RELEASE_PEER.

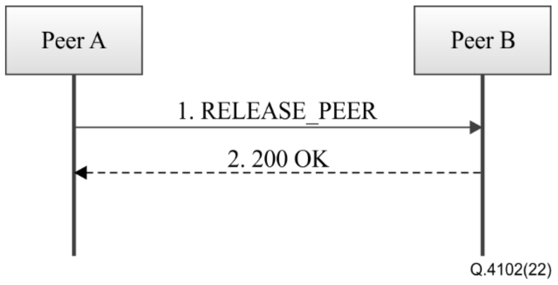


Figure 7-10 – Message flow of RELEASE_PEER

① Peer A transmits a RELEASE_PEER request message to peer B to terminate the connection.

In order to prevent any peer from terminating the connection, peer A signs the entire RELEASE_PEER request message with a private key and sends it including its own signature.

② Upon receipt of the RELEASE_PEER request message, peer B sends a 200 response after internal processing when the *ack* field of the RELEASE_PEER request message is set to TRUE.

Table 7-20 describes the syntax and semantics of each field in the header of the RELEASE_PEER request message.

Table 7-20 – Header format of RELEASE_PEER request

Syntax	Description

{
"req-code": NUMBER,
"req-params": {
"operation": {
"ack": BOOLEAN
}
}
}
	– *req-code* indicates the type of the request message. In case of a RELEASE_PEER request message, *req-code* value is set as 9.
	– *ack* has a TRUE value if a receive a response message is needed. Default value is FALSE.

Table 7-21 describes the syntax and semantics of each field in the header of the RELEASE_PEER response message.

Table 7-21 – Header format of RELEASE_PEER response

Syntax	Description

{
"rsp-code": NUMBER
}
	– *rsp-code* indicates which response message is for which request. In case of the RELEASE_PEER response message, *rsp-code* value is set as 9200.
	• *rsp-code* value 9200 is a combination of the request type (9) and the response type (200). Therefore, *rsp-code* value 9200 is an 'OK' response to the RELEASE_PEER request message.

7.2.10 BUFFERMAP

Peers can share their buffermap information with each other by exchanging BUFFERMAP request and response messages. A BUFFERMAP request message is used to initiate information sharing. Figure 7-11 shows the message flow of BUFFERMAP.

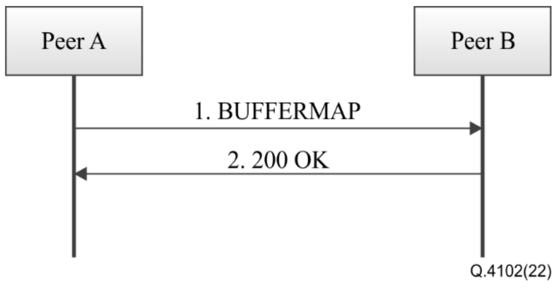


Figure 7-11 – Message flow of BUFFERMAP

① Peer A generates and sends a BUFFERMAP request message to peer B.

② Peer B sends a 200 response message including its own buffer map to the requesting peer A.

At this time, the 200 response includes the buffer map information representing its own caching state in the form of a PEER_BUFFERMAP defined in clause 7.1.1.

Table 7-22 describes the syntax and semantics of each field in the header of the BUFFERMAP request message.

Table 7-22 – Header format of BUFFERMAP request

Syntax	Description

{
    "req-code": NUMBER,
    "req-params": {
        "overlay-id": STRING
        }
    }
}
	– *req-code* indicates the type of the request message. In case of a BUFFERMAP request message, *req-code* value is set as 10.
	– *overlay-id* indicates the identifier of the overlay network.

Table 7-23 describes the syntax and semantics of each field in the header of a BUFFERMAP response message.

Table 7-23 – Header format of BUFFERMAP response

Syntax	Description

{
    "rsp-code": NUMBER,
    "rsp-params": {
        "overlay-id": STRING,
        "buffermap": PEER_BUFFERMAP
    }
}
	– *rsp-code* indicates which response message is for which request. In case of a BUFFERMAP response message, the *rsp-code* value is set as 10200.
	• *rsp-code* value 10200 is a combination of the request type (10) and the response type (200). Therefore, *rsp-code* value 10200 is an 'OK' response to the BUFFERMAP request message.
	– *overlay-id* indicates the identifier of the overlay network.
	– *buffermap* expresses its own caching buffer state in the form of PEER_BUFFERMAP.

7.2.11 GET_DATA

A peer can get specific data from other peers by exchange of GET_DATA request and response messages. A GET_DATA message is used to request the data with the specific sequence number held by another peer. Figure 7-12 shows the message flow of GET_DATA.

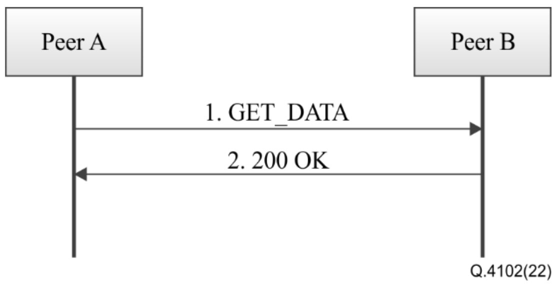


Figure 7-12 – Message flow of GET_DATA

① Requesting peer A generates and sends a GET_DATA request message to peer B.

At this time, the GET_DATA request message includes the sequence number that a requesting peer A wants to obtain.

② Peer B sends a 200 response message containing data of the requested sequence number to requesting peer A.

At this time, the 200 response includes the content data by the requesting peer's request in the *content* field, the length and the type of content data in the *payload* field in the *header* field, and the identifier of the application in the *extension* field.

Table 7-24 describes the syntax and semantics of each field in the header in the GET_DATA request message.

Table 7-24 – Header format of GET_DATA request

Syntax	Description

{
"req-code": NUMBER,
"req-params": {
"overlay-id": STRING,
"source-id": STRING,
"sequence": NUMBER
}
}
	– *req-code* indicates the type of the request message. In case of GET_DATA request message, *req-code* value is set as 11.
	– *overlay-id* indicates the identifier of the overlay network.
	– *source_id* indicates the source ID of the desired data.
	– *sequence* indicates the sequence number of the desired data.

Table 7-25 describes the syntax and semantics of each field in the header in the GET_DATA response message.

Table 7-25 – Header format of GET_DATA response

Syntax	Description

{
"rsp-code": NUMBER,
"rsp-params": {
"peer": {
"peer-id": STRING
},
	– *rsp-code* indicates which response message is for which request. In case of a GET_DATA response message, the *rsp-code* value is set as 11200.
	• *rsp-code* value 11200 is a combination of the request type (11) and the response type (200). Therefore, *rsp-code*

"sequence": NUMBER,
"payload": {
"length": NUMBER,
"content-type": STRING
}
}
}
	value 11200 is an 'OK' response to the GET_DATA request message.
	– *peer-id* indicates the peer ID of the data packet included in the content data, which matches the *source-id* of the GET_DATA request message.
	– *sequence* indicates the sequence number of the data packet included in the content data, which matches the sequence of the GET_DATA request message.
	– *length* indicates the length of the data that is carried and is described in bytes.
	– *content-type* indicates the kind of data that is carried. It is described as the MIME type such as application/JSON, application/XML, etc. The processing of content according to content-type depends on the application using the peer protocol.

The GET_DATA response message shall use an extension field to specify application specific parameter.

Table 7-26 describes the syntax and semantics of each field in the extension in the GET_DATA response message.

Table 7-26 – Extension format of GET_DATA response

Syntax	Description

{
"app-id": STRING
}
	– app-id indicates the identifier of an application using the peer protocol

8 Protocol behaviours

This clause describes the information flows of protocol operations such as joining and leaving an overlay network, broadcast of data, pruning of dead peer, tree and data recovery and tree scanning.

8.1 Joining phase

This clause describes the case of joining a new existing overlay network. The new peer chooses peers from a list of peers and sends a HELLO_PEER request message with the maximum number of connections to the peers to which it wishes to connect. The receiving peer that can accept the new connection sends an ESTAB_PEER request message so that the new peer makes the desired connection.

Figure 8-1 shows the procedure for peer A to join an existing overlay network. In the existing overlay network, it is assumed that peer 1 has a primary connection with peer 2, and peer 2 with peer 3.

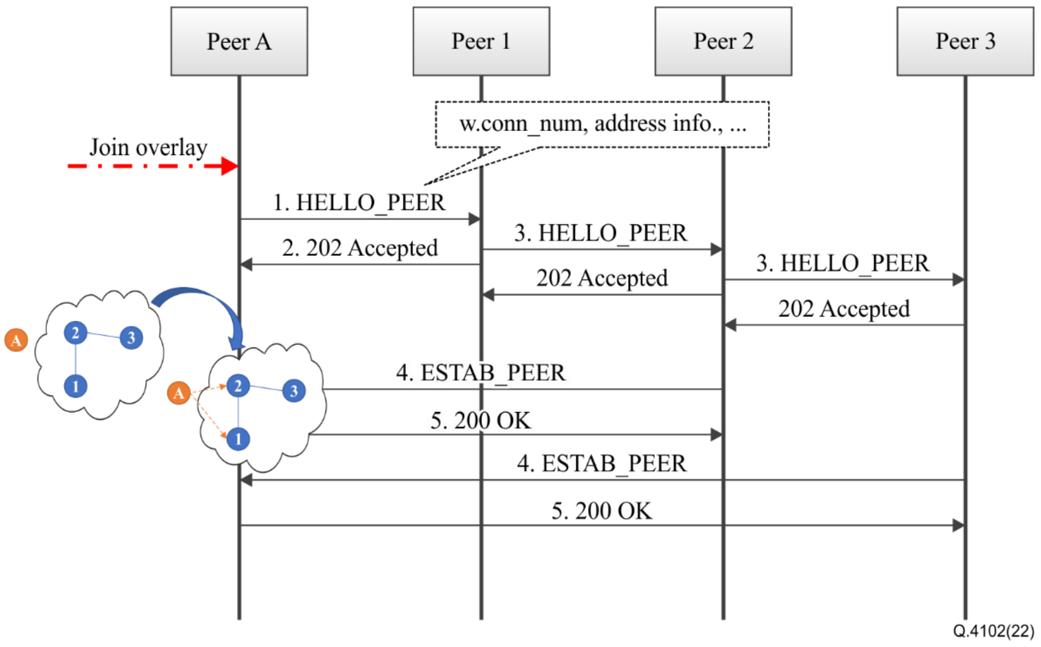


Figure 8-1 – Message flow for joining an overlay network

① When peer A wants to join an overlay network, it sends a HELLO_PEER request message to one of the peers participating in the overlay network. Peer A sends a HELLO_PEER request message to peer 1.

At this time, the list of peers participating in the overlay network is obtained from the HOMS through the HybridOverlayJoin message defined in [ITU-T Q.4103].

The HELLO_PEER request message must contain *conn_num* and *address* information of peer A as specified in clause 7.2.1 of this Recommendation. *conn_num* information is the number of requesting connections and address information is one that can be accessed by other peers.

② Peer 1 firstly checks whether the received HELLO_PEER request message is acceptable, and if possible, sends a 202 response message to peer A.

③ Peer 1 checks whether the number of requesting connections in the HELLO_PEER request message is satisfied according to the maximum number of requests specified in clause 7.2.1 of this Recommendation, and if not, delivers the received HELLO_PEER request message to the peer connected through the primary connection.

At this time, it is assumed that peer A sets the number of requested connections to 1 or more. Therefore, peer 1 forwards the HELLO_PEER request message to peer 2, and peer 2 forwards it to peer 3.

④ Peers receiving the HELLO_PEER request message checks whether the resource is available and then sends an ESTAB_PEER request message to peer A. In Figure 8-1, peer 2 and peer 3 send an ESTAB_PEER request message to peer A.

⑤ Peer A receiving the ESTAB_PEER message sends a 200 response if it wishes to accept the request. If peer A receives the ESTAB_PEER message after ESTAB_PEER_TIMEOUT, it sends a 603 response as specified in clause 7.2.2 of this Recommendation.

Thereafter, peer A calculates the network distance using PROBE_PEER messages for the created connections as specified in clause 7.2.3 of this Recommendation. Peer A shall send a SET_PRIMARY request message to one peer with the optimum network distance as specified in clause 7.2.4 of this Recommendation, and may send a SET_CANDIDATE request message to the others as specified in clause 7.2.5 of this Recommendation.

8.2 Service phase

8.2.1 Broadcast of data

This clause describes when a peer wants to send data to other peers participating in the same overlay network. Peer A sends a BROADCAST_DATA request message to the peers in the primary connection relationship, and the peer receiving this message relays it to the other peers, and the BROADCAST_DATA message is delivered to the leaf node through the intermediate nodes of the tree-based overlay network.

Figure 8-2 shows the procedure for peer A to deliver data to peers in a primary connection. It is assumed that peer A has a primary connection with peer 2, peer 2 with peers 1 and 3, and peer 1 with peer 4.

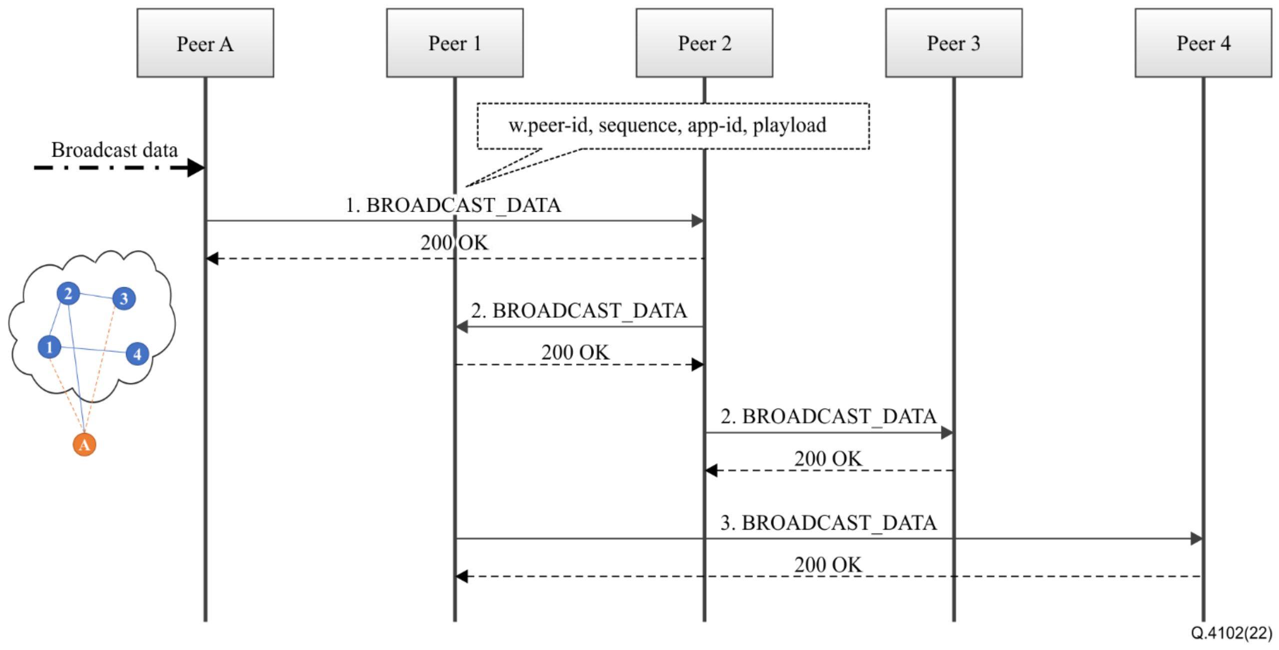


Figure 8-2 – Message flow for broadcast of data

① Peer A generates and sends a BROADCAST_DATA request message to peer 2 with the primary connection relationship.

Since this BROADCAST_DATA request message was originated by peer A, peer A assigns values to the *peer-id* and *sequence* fields when it generates the BROADCAST_DATA request message. The *Payload* field within the BROADCAST_DATA request message is composed of *length* and *content-type* fields which contain information about the content data that will be transmitted. Peer A can also assign the identifier of an application using the peer protocol to the *app-id* field within an extension.

Upon receipt of the BROADCAST_DATA request message, peer 2 sends a 200 response when the *ack* field of the BROADCAST_DATA request message is set to TRUE.

② Peer 2, receiving the BROADCAST_DATA message, delivers the received BROADCAST_DATA request message to peer 1 and peer 3 in the primary connection.

Peer ID is included in this BROADCAST_DATA message to identify who created and sent this data for the first time.

③ The BROADCAST_DATA request message is delivered until it reaches the leaf node of the tree. Therefore, peer 1 forwards the BROADCAST_DATA request message to peer 4.

Upon receipt of the BROADCAST_DATA request message, peer 1, peer 3, and peer 4 sends a 200 response when the *ack* field of the BROADCAST_DATA request message is set to TRUE.

8.2.2 Pruning of a dead peer

If a peer leaves gracefully by explicitly sending a RELEASE_PEER request message to its peers with a primary connection relationship. The corresponding peers can detect and initiate a tree recovery procedure if the leaving peer is peer in a primary connection relationship with a lower *ticket-id*. In the case of an ungraceful leaving, the peer with a higher *ticket-id* is responsible for detecting a dead peer by periodically sending a HEARTBEAT request message. The *heartbeat-interval* and *heartbeat-timeout* of the HEARTBEAT request message is provided from HOMS when the peer joins the overlay network.

Figure 8-3 shows the procedures for detecting a dead peer leaving ungracefully.

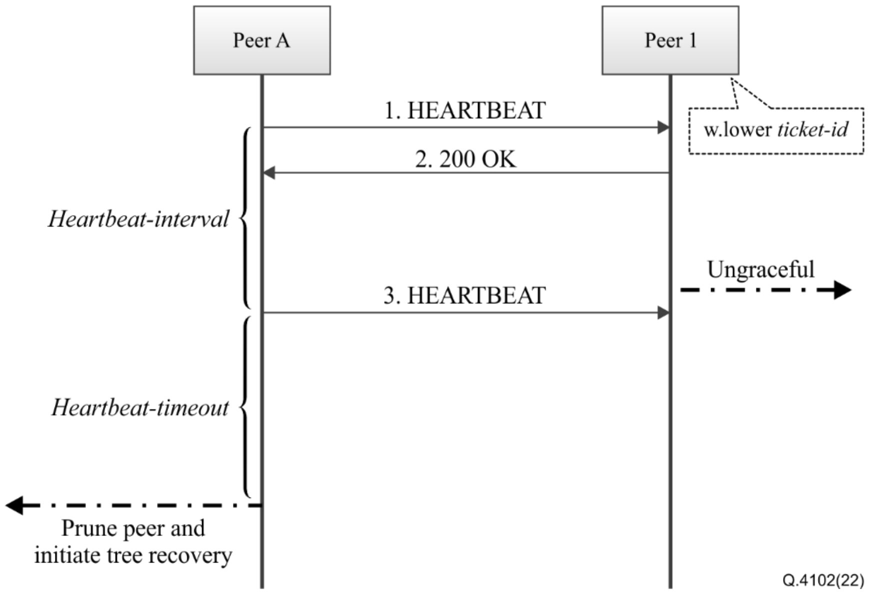


Figure 8-3 – Message flows for pruning of dead peer

① Peer A with a higher *ticket-id* sends a HEARTBEAT request message to peer 1 in the primary connection relationship.

② Peer 1 with a lower *ticket-id* responds with a 200 response on receiving the HEARTBEAT request message.

③ After a specific time is set as *heartbeat-interval*, peer A sends a HEARTBEAT request message. If the corresponding peer does not respond to this request, the requesting peer waits until *heartbeat-timeout* passes. If there is no response after the *heartbeat-timeout* time, the requesting peer treats the nonresponsive peer as a dead peer and initiates a tree recovery procedure.

8.2.3 Tree recovery

When a peer detects an event regarding the leaving of a peer in a primary connection relationship with a higher *ticket-id*, it initiates a tree recovery procedure. It can detect the leave by using a periodic HEARTBEAT message or from an event fired from the underlying transport layer. The procedures for recovering the hybrid overlay network are specified in clause 7.2 of [ITU-T Q.4101] in detail.

Figure 8-4 shows the procedures for tree recovery. It is assumed that peer A established a primary connection with peer 1, and it has a candidate connection with peer 2 and peer 3. It is assumed that the overlay network does not require data recovery. This is useful in the case where recovering timely old data is useless, such as live media streaming.

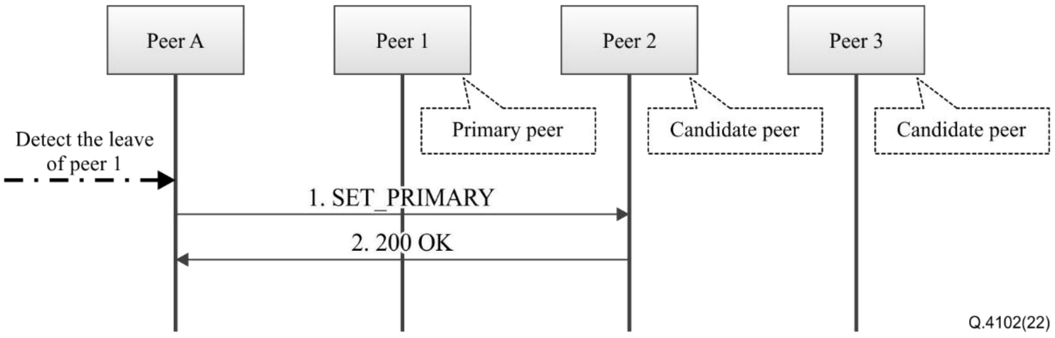


Figure 8-4 – Message flow for tree recovery

① Peer A sends a SET_PRIMARY request message to its outgoing candidate connection that is connected to peer 2.

② If peer 2 has enough resource, it responds with success (200). On sending and receiving the response, both peers regard a primary connection between them as established, and they can use the connection for data delivery immediately.

If a peer receiving a SET_PRIMARY request message does not have enough resources available, it responds with an error response. On receiving the error response, the requesting peer immediately sends a SET_PRIMARY request message to another peer in the candidate connection relationship. If there no more exists a candidate connection, the requesting peer requests entry into the peer list to HOMS as soon as possible. On acquiring the entry peer list from HOMS, the requesting peer initiates an overlay joining procedures specified in clause 8.1 of this Recommendation with its own ticket-id that is issued at the initial joining phase. If there is no entry peer list from HOMS, it indicates that the requesting peer is the one that has the lowest *ticket-id*. In this case, it stops the initiation procedures to wait for the tree recovery procedure initiated by other peers with higher *ticket-id*.

8.2.4 Data recovery

When a hybrid overlay network requires data recovery that has been lost during the recovery of the tree, all peers keep their own cache for a pre-specified amount of time and quantity of messages. The amount and the number of messages to be cached are specified on creating the overlay network, and the policies are delivered to each peer on its joining by interacting with HOMS. The procedures for data recovery are specified in clause 8.2.2 of [ITU-T Q.4101] in detail.

Figure 8-5 shows the information flow for recovering data lost during the tree recovery procedures. It is assumed that the peers are configured to recover the data using push mode.

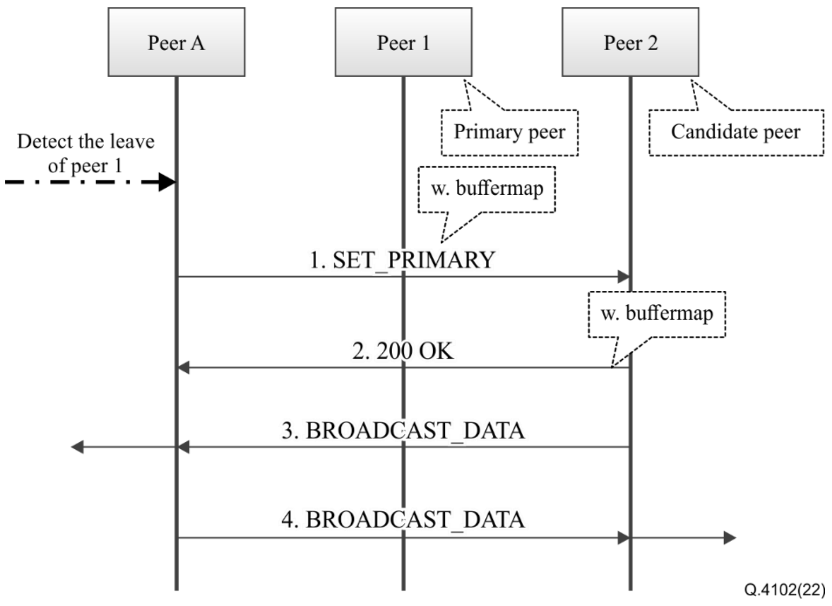


Figure 8-5 – Message flow for data recovery

① Peer A sends a SET_PRIMARY request message to its outgoing candidate connection that is connected to peer 2. In this case, the peer embeds its buffermap in this message.

② If the corresponding peer 2 has enough resources, it responds with success (200). On sending the response message, peer 2 also embeds its buffermap.

③ Through step 2, peer 2 figures out what data packet needs to be pushed to peer A. Peer 2 sends a BROADCAST_DATA request message to peer A for pushing data packets that peer A does not have. On receiving the BROADCAST_DATA request message, peer A immediately sends it to its other primary connections.

④ On receiving a 200 response to the SET-PRIMARY request message from peer 2, peer 1 also sends data packets to peer 2 using a BROADCAST_DATA request message. On receiving the BROADCAST_DATA request message, peer 2 immediately sends it to its other primary connections.

8.2.5 Scanning of tree

This clause describes a case where a peer participating in an overlay network wants to obtain information about peers in a primary connection relationship with itself. Peer A sends a SCAN_TREE request message to the peer in the primary connection relationship. The peer receiving the SCAN_TREE request message sends a modified version of the SCAN_TREE request message to other peers. Information on peers connected through the primary connection can be obtained through the *path* field of the 200 response message transmitted from the leaf node.

Figure 8-6 shows the procedure for peer A to obtain information about peers in a primary connection relationship. It is assumed that peer A has a primary connection with peer 2, peer 2 with peers 1 and 3, and peer 1 with peer 4.

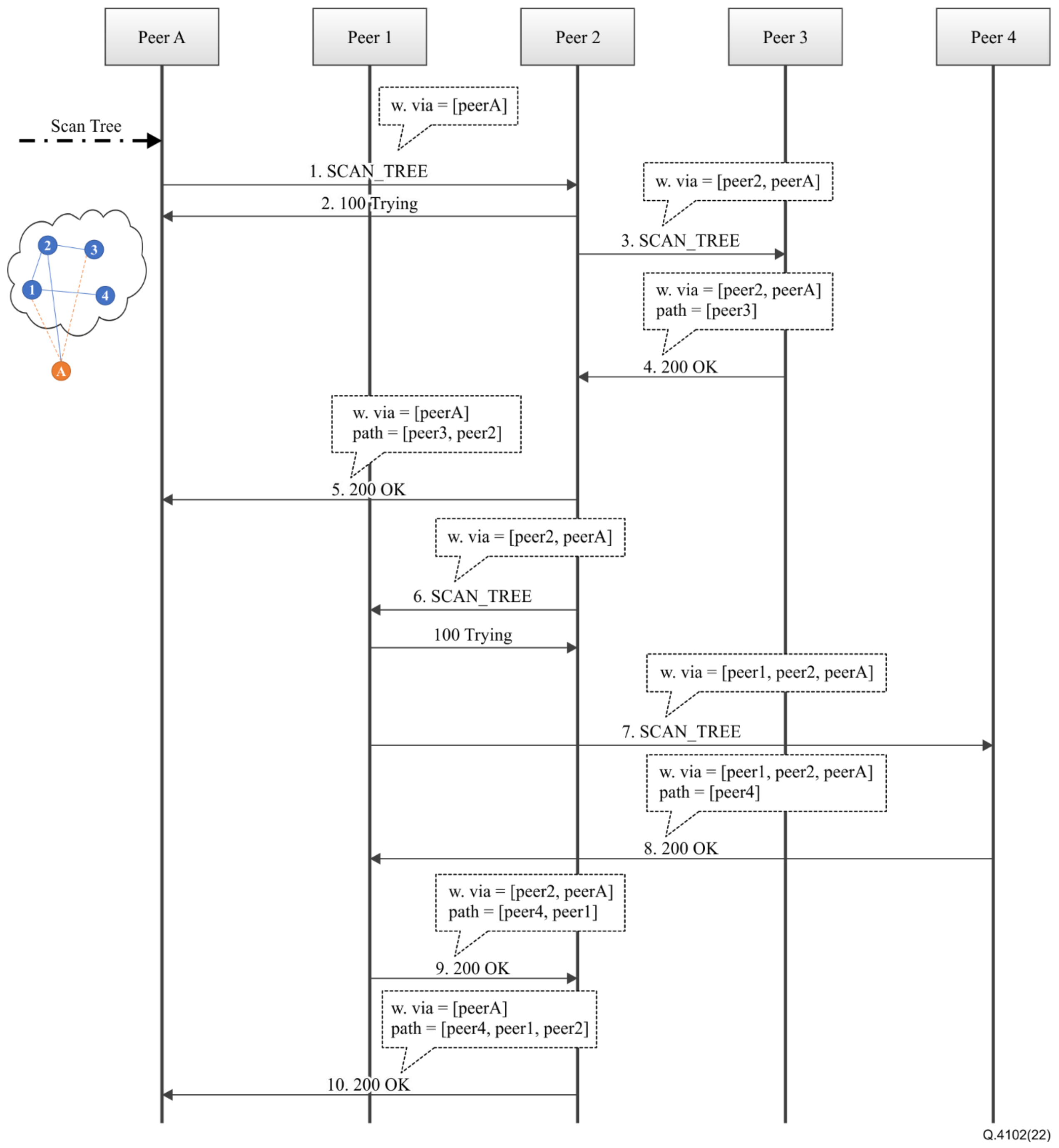


Figure 8-6 – Message flow for network scanning

① Requesting peer A generates and sends a SCAN_TREE request message to peer 2.

At this time, a random digit number is added in the *CSEQ* field and the peer information (peer-id, address) on the requesting peer is added in the *via* field within the SCAN_TREE request message.

② Since peer 2 is an intermediate node of tree, it sends a 100 response message to the requesting peer A.

③ Since peer 2 is an intermediate node of tree, it also sends a revised version of the SCAN_TREE request message to peer 3 and peer 1 with a primary connection relationship (except the requesting peer for loop prevention).

At this time, the new SCAN_TREE request message includes peer 2's information in the *via* field.

④ Since peer 3 is a leaf node of tree, peer 3 sends a 200 response message to peer 2.

At this time, peer information (peer-id and address) on peer 3 is added in the *path* field within the 200 response message. *Path* field is a list of peer information. Information on peer 3 is added at the end of the *Path* field.

The destination of the 200 response message is the origin of the SCAN_TREE request message which is listed at the beginning of the *via* field within SCAN_TREE.

⑤ Peer 2, receives a 200 response message from peer 3, sends the 200 response message to peer A.

Whenever a peer receives a response message, it checks whether its information is at the end of *via* field in the received response message. If the information is at the end of the *via* field, it indicates that the peer received as the response message is the origin of the SCAN_TREE request message. Otherwise, the peer is an intermediate node in the overlay network.

Since peer 2 is the intermediate requesting peer (i.e., matching), peer 2 moves its information at the beginning of *via* field within the received 200 response to the end of *path* field within the newly generated 200 response, and then transfers the newly generated 200 response to the peer at the beginning of the *via* field, which is peer A in Figure 8-6.

⑥ Since peer 1, which is the received SCAN_TREE request message from peer 2, is an intermediate node of tree, peer 1 sends a 100 response message to the requesting peer 2.

⑦ Since peer 1 is an intermediate node of tree, it transfers the SCAN_TREE request message to peer 4.

⑧ Since peer 4 is a leaf node of tree, peer 4 sends a 200 response message to peer 1.

⑨ Peer 1, after receiving a 200 response message from peer 4, forwards the 200 response message to peer 2.

⑩ Peer 2, also sends the 200 response message to peer A. Peer A, who received the 200 response message performs the internal processing to express the relationship of peers constituting the tree.

Peer A can identify the composition and order of the peers in the branch that make up the tree. The order can be determined from a listed set of peer information contained in a *path* field of a 200 response.

8.3 Leaving phase

This clause describes the case when a peer needs to leave a participating overlay network. Peer A, when leaving, sends a RELEASE_PEER request message to peers in the primary connection relationship. Peers receiving the RELEASE_PEER request message changes one of the candidate connections to a primary connection. In the case of a peer whose connection relationship has changed, it notifies HOMS of the changed status.

Figure 8-7 shows the procedure for a peer A to leave a participating overlay network. It is assumed that peer A has a primary connection to peer 1 and peer 2.

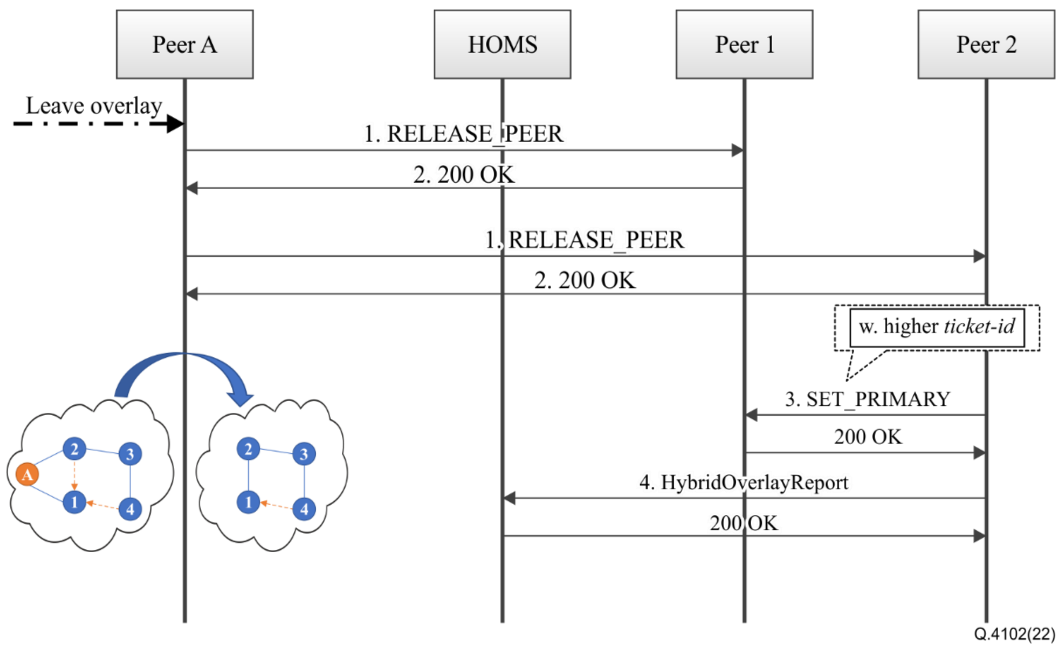


Figure 8-7 – Message flow for leaving an overlay network

① Peer A, when leaving a participating overlay network, sends a RELEASE_PEER request message to peer 1 and peer 2 in the primary connection relationship. At this time, to receive a response to the RELEASE_PEER request message, the *ack* field of the RELEASE_PEER request message is set to TRUE.

② Peer 1 and peer 2 receiving the RELEASE_PEER request message checks whether the *ack* field of the RELEASE_PEER request message is set to TRUE. After confirming that the *ack* field value is TRUE, peer1 and peer2 send a 200 response to peer A.

③ Peer 1 and peer 2 receiving the RELEASE_PEER request message changes one of the candidate connections to the primary connection. At this time, peer 2 with a higher *ticket-id* sends a SET_PRIMARY request message to peer 1 with a lower *ticket-id*.

④ Peer 2 is disconnected from peer A and has a new primary connection with peer 1. Thus peer 2 notifies HOMS of the changed status through the HybridOverlayReport message specified in clause 6.3 of [ITU-T Q.4103].

Bibliography

[b-ITU-T X.609]	Recommendation ITU-T X.609 (2015), *Managed peer-to-peer (P2P) communications: Functional architecture*.
[b-ITU-T X.609.4]	Recommendation ITU-T X.609.4 (2018), *Managed P2P communications: Multimedia streaming peer protocol*.
[b-ITU-T X.1161]	Recommendation ITU-T X.1161 (2008), *Framework for secure peer-to-peer communications*.
[b-ITU-T X.1162]	Recommendation ITU-T X.1162 (2008), *Security architecture and operations for peer-to-peer networks*.
[b-ITU-T Y.2206]	Recommendation ITU-T Y.2206 (2010), *Requirements for distributed service networking capabilities*.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	**Switching and signalling, and associated measurements and tests**
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2022