Stream: Internet Engineering Task Force (IETF)
RFC: 8855
Obsoletes: 4582
Category: Standards Track
Published: January 2021
ISSN: 2070-1721
Authors: G. Camarillo, *Ericsson*; K. Drage, *Jotron*; T. Kristensen, *Jotron*; J. Ott, *Technical University Munich*; C. Eckel, *Cisco*

RFC 8855
The Binary Floor Control Protocol (BFCP)

Abstract

Floor control is a means to manage joint or exclusive access to shared resources in a (multiparty) conferencing environment. Thereby, floor control complements other functions -- such as conference and media session setup, conference policy manipulation, and media control -- that are realized by other protocols.

This document specifies the Binary Floor Control Protocol (BFCP). BFCP is used between floor participants and floor control servers, and between floor chairs (i.e., moderators) and floor control servers.

This document obsoletes RFC 4582.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8855.

Copyright Notice

Table of Contents

1. Introduction

Within a conference, some applications need to manage the access to a set of shared resources, such as the right to send media to a particular media session. Floor control enables such applications to provide users with coordinated (shared or exclusive) access to these resources.

The Requirements for Floor Control Protocol [18] list a set of requirements that need to be met by floor control protocols. The Binary Floor Control Protocol (BFCP), which is specified in this document, meets these requirements.

In addition, BFCP has been designed so that it can be used in low-bandwidth environments. The binary encoding used by BFCP achieves a small message size (when message signatures are not used) that keeps the time it takes to transmit delay-sensitive BFCP messages to a minimum.

Delay-sensitive BFCP messages include FloorRequest, FloorRelease, FloorRequestStatus, and ChairAction. It is expected that future extensions to these messages will not increase the size of these messages in a significant way.

The remainder of this document is organized as follows: Section 2 defines the terminology used throughout this document, Section 3 discusses the scope of BFCP (i.e., which tasks fall within the scope of BFCP and which ones are performed using different mechanisms), Section 4 provides a non-normative overview of BFCP operation. The subsequent sections provide the normative specification of BFCP. Section 16 summarizes changes from RFC 4582 [3].

2. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [1] [10] when, and only when, they appear in all capitals, as shown here.

Media Participant: An entity that has access to the media resources of a conference (e.g., it can receive a media stream). In floor-controlled conferences, a given media participant is typically co-located with a floor participant, but it does not need to be. Third-party floor requests consist of having a floor participant request a floor for a media participant when they are not co-located. The protocol between a floor participant and a media participant (that are not co-located) is outside the scope of this document.

Client: A floor participant or a floor chair that communicates with a floor control server using BFCP.

Floor: A temporary permission to access or manipulate a specific shared resource or set of resources.

Floor Chair: A logical entity that manages one floor (grants, denies, or revokes a floor). An entity that assumes the logical role of a floor chair for a given transaction may assume a different role (e.g., floor participant) for a different transaction. The roles of floor chair and floor participant are defined on a transaction-by-transaction basis. BFCP transactions are defined in Section 8.

Floor Control: A mechanism that enables applications or users to gain safe and mutually exclusive or non-exclusive input access to the shared object or resource.

Floor Control Server: A logical entity that maintains the state of the floor(s), including which floors exists, who the floor chairs are, who holds a floor, etc. Requests to manipulate a floor are directed at the floor control server. The floor control server of a conference may perform other logical roles (e.g., floor participant) in another conference.

Floor Participant: A logical entity that requests floors, and possibly information about them, from a floor control server. An entity that assumes the logical role of a floor participant for a given transaction may assume a different role (e.g., a floor chair) for a different transaction.

The roles of floor participant and floor chair are defined on a transaction-by-transaction basis. BFCP transactions are defined in Section 8. In floor-controlled conferences, a given floor participant is typically co-located with a media participant, but it does not need to be. Third-party floor requests consist of having a floor participant request a floor for a media participant when they are not co-located.

Participant: An entity that acts as a floor participant, as a media participant, or as both.

BFCP Connection: A transport association between BFCP entities, used to exchange BFCP messages.

Transaction Failure Window: When communicating over an unreliable transport, this is some period of time less than or equal to $T1*2^4$ (see Section 8.3). For reliable transports, this period of time is unbounded.

3. Scope

As stated earlier, BFCP is a protocol to coordinate access to shared resources in a conference following the requirements defined in [18]. Floor control complements other functions defined in the Centralized Conferencing (XCON) Framework [19]. The floor control protocol BFCP defined in this document only specifies a means to arbitrate access to floors. The rules and constraints for floor arbitration and the results of floor assignments are outside the scope of this document and are defined by other protocols [19].

Figure 1 shows the tasks that BFCP can perform.

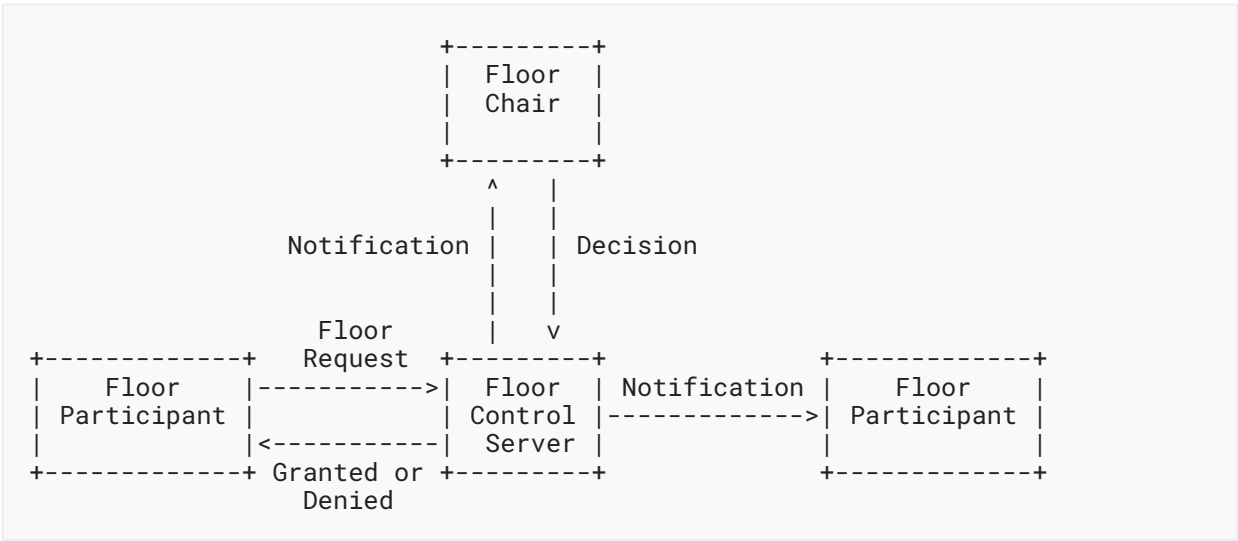


Figure 1: Functionality provided by BFCP

BFCP provides a means:

- for floor participants to send floor requests to floor control servers.

- for floor control servers to grant or deny requests to access a given resource from floor participants.
- for floor chairs to send floor control servers decisions regarding floor requests.
- for floor control servers to keep floor participants and floor chairs informed about the status of a given floor or a given floor request.

Even though tasks that do not belong to the previous list are outside the scope of BFCP, some of these out-of-scope tasks relate to floor control and are essential for creating floors and establishing BFCP connections between different entities. In the following subsections, we discuss some of these tasks and mechanisms to perform them.

3.1. Floor Creation

The association of a given floor with a resource or a set of resources (e.g., media streams) is out of the scope of BFCP as described in [19]. Floor creation and termination are also outside the scope of BFCP; these aspects are handled using the conference control protocol for manipulating the conference object. Consequently, the floor control server needs to stay up to date on changes to the conference object (e.g., when a new floor is created).

Conference control clients using Centralized Conferencing Manipulation Protocol (CCMP) [23] can specify such floor-related settings in the <floor-information> element [22] of the to-be created conference object provided in the body of a CCMP confRequest/create message issued to the conference control server.

3.2. Obtaining Information to Contact a Floor Control Server

A client needs a set of data in order to establish a BFCP connection to a floor control server. These data include the transport address of the server, the conference identifier, and a user identifier.

Clients can obtain this information in different ways. One is to use a Session Description Protocol (SDP) offer/answer [17] exchange, which is described in [12]. How to establish a connection to a BFCP floor control server is outside the context of an offer/answer exchange when using a reliable transport is described in [4]. Other mechanisms are described in the XCON Framework [19] (and other related documents). For unreliable transports, the use of an SDP offer/answer exchange is the only specified mechanism.

3.3. Obtaining Floor-Resource Associations

Floors are associated with resources. For example, a floor that controls who talks at a given time has a particular audio session as its associated resource. Associations between floors and resources are part of the conference object.

Floor participants and floor chairs need to know which resources are associated with which floors. They can obtain this information by using different mechanisms, such as an SDP offer/answer [17] exchange. How to use an SDP offer/answer exchange to obtain these associations is described in [12].

> Note that floor participants perform SDP offer/answer exchanges with the conference focus of the conference. So, the conference focus needs to obtain information about associations between floors and resources in order to be able to provide this information to a floor participant in an SDP offer/answer exchange.

Other mechanisms for obtaining this information, including discussion of how the information is made available to a (SIP) focus, are described in the XCON Framework [19] (and other related documents). According to the conferencing system policies, conference control clients using CCMP [23] can modify the floor settings of a conference by issuing CCMP confRequest/update messages providing the specific updates to the <floor-information> element of the target conference object. More information about CCMP and BFCP interaction can be found in [24].

3.4. Privileges of Floor Control

A participant whose floor request is granted has the right to use the resource or resources associated with the floor that was requested. For example, the participant may have the right to send media over a particular audio stream.

Nevertheless, holding a floor does not imply that others will not be able to use its associated resources at the same time, even if they do not have the right to do so. Determination of which media participants can actually use the resources in the conference is discussed in the XCON Framework [19].

4. Overview of Operation

This section provides a non-normative description of BFCP operations. Section 4.1 describes the interface between floor participants and floor control servers, and Section 4.2 describes the interface between floor chairs and floor control servers.

BFCP messages, which use a TLV (Type-Length-Value) binary encoding, consist of a COMMON-HEADER followed by a set of attributes. The COMMON-HEADER contains, among other information, a 32-bit conference identifier. Floor participants, media participants, and floor chairs are identified by 16-bit user identifiers.

BFCP supports nested attributes (i.e., attributes that contain attributes). These are referred to as grouped attributes.

There are two types of transactions in BFCP: client-initiated transactions and server-initiated transactions. Section 8 describes both types of transactions in detail.

4.1. Floor Participant to Floor Control Server Interface

Floor participants request a floor by sending a FloorRequest message to the floor control server. BFCP supports third-party floor requests. That is, the floor participant sending the floor request need not be co-located with the media participant that will get the floor once the floor request is

granted. FloorRequest messages carry the identity of the requester in the User ID field of the COMMON-HEADER, and the identity of the beneficiary of the floor (in third-party floor requests) in a BENEFICIARY-ID attribute.

> Third-party floor requests can be sent, for example, by floor participants that have a BFCP connection to the floor control server but that are not media participants (i.e., they do not handle any media).

FloorRequest messages identify the floor or floors being requested by carrying their 16-bit floor identifiers in FLOOR-ID attributes. If a FloorRequest message carries more than one floor identifier, the floor control server treats all the floor requests as an atomic package. That is, the floor control server either grants or denies all the floors in the FloorRequest message.

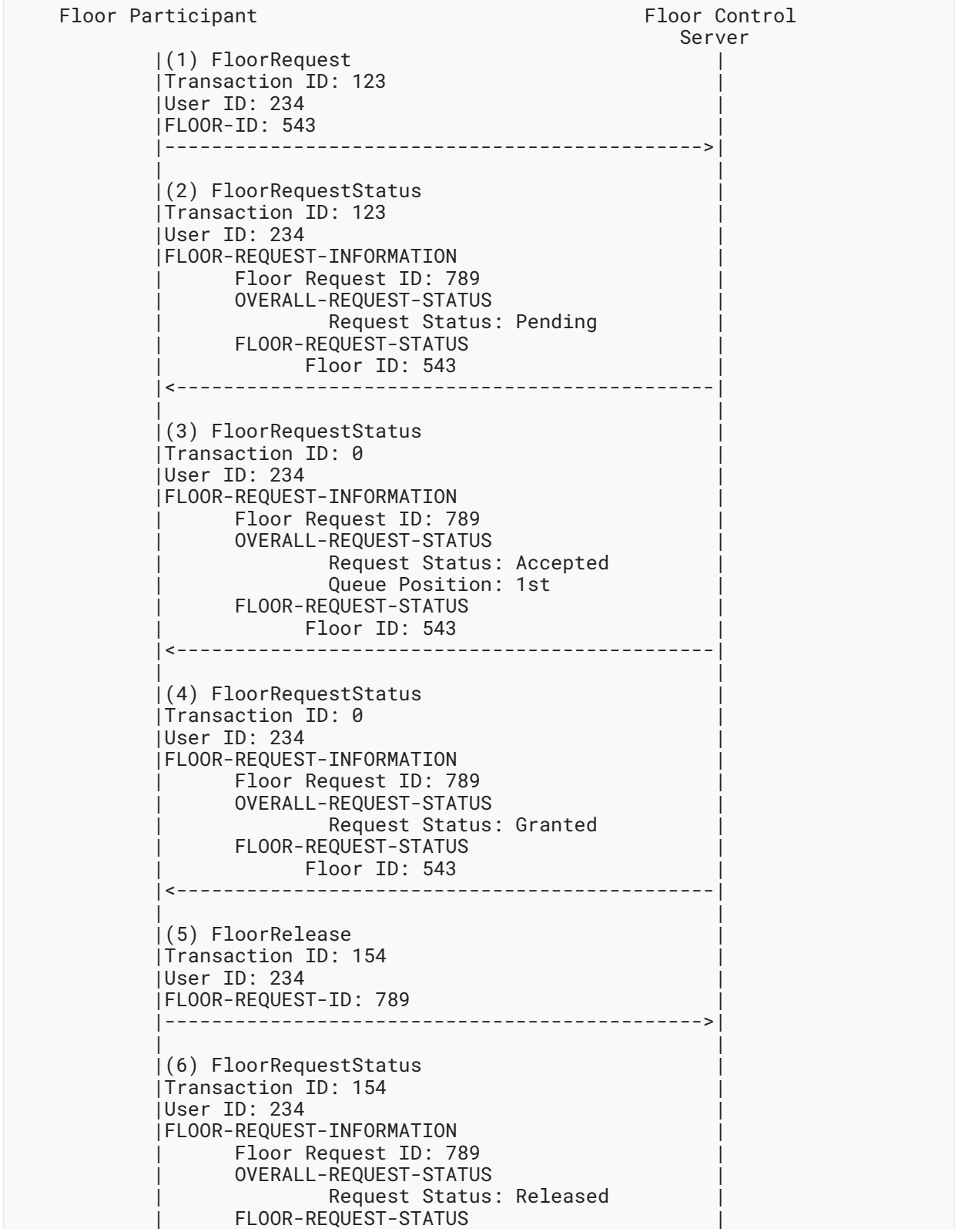
Floor control servers respond to FloorRequest messages with FloorRequestStatus messages, which provide information about the status of the floor request. The first FloorRequestStatus message is the response to the FloorRequest message from the client, and therefore has the same Transaction ID as the FloorRequest.

Additionally, the first FloorRequestStatus message carries the Floor Request ID in a FLOOR-REQUEST-INFORMATION attribute. Subsequent FloorRequestStatus messages related to the same floor request will carry the same Floor Request ID. This way, the floor participant can associate them with the appropriate floor request.

Messages from the floor participant related to a particular floor request also use the same Floor Request ID as the first FloorRequestStatus message from the floor control server.

Figure 2 and Figure 3 show examples of call flows where BFCP is used over a reliable transport. Appendix A shows the same call flow examples using an unreliable transport.

Figure 2 shows how a floor participant requests a floor, obtains it, and, at a later time, releases it. This figure illustrates the use, among other things, of the Transaction ID and the FLOOR-REQUEST-ID attribute.


Floor Participant
Floor Control Server
(1) FloorRequest
Transaction ID: 123
User ID: 234
FLOOR-ID: 543
(2) FloorRequestStatus
Transaction ID: 123
User ID: 234
FLOOR-REQUEST-INFORMATION
Floor Request ID: 789
OVERALL-REQUEST-STATUS
Request Status: Pending
FLOOR-REQUEST-STATUS
Floor ID: 543
(3) FloorRequestStatus
Transaction ID: 0
User ID: 234
FLOOR-REQUEST-INFORMATION
Floor Request ID: 789
OVERALL-REQUEST-STATUS
Request Status: Accepted
Queue Position: 1st
FLOOR-REQUEST-STATUS
Floor ID: 543
(4) FloorRequestStatus
Transaction ID: 0
User ID: 234
FLOOR-REQUEST-INFORMATION
Floor Request ID: 789
OVERALL-REQUEST-STATUS
Request Status: Granted
FLOOR-REQUEST-STATUS
Floor ID: 543
(5) FloorRelease
Transaction ID: 154
User ID: 234
FLOOR-REQUEST-ID: 789
(6) FloorRequestStatus
Transaction ID: 154
User ID: 234
FLOOR-REQUEST-INFORMATION
Floor Request ID: 789
OVERALL-REQUEST-STATUS
Request Status: Released
FLOOR-REQUEST-STATUS

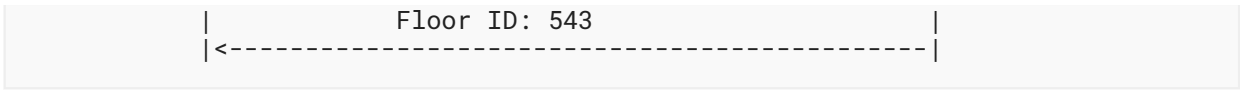
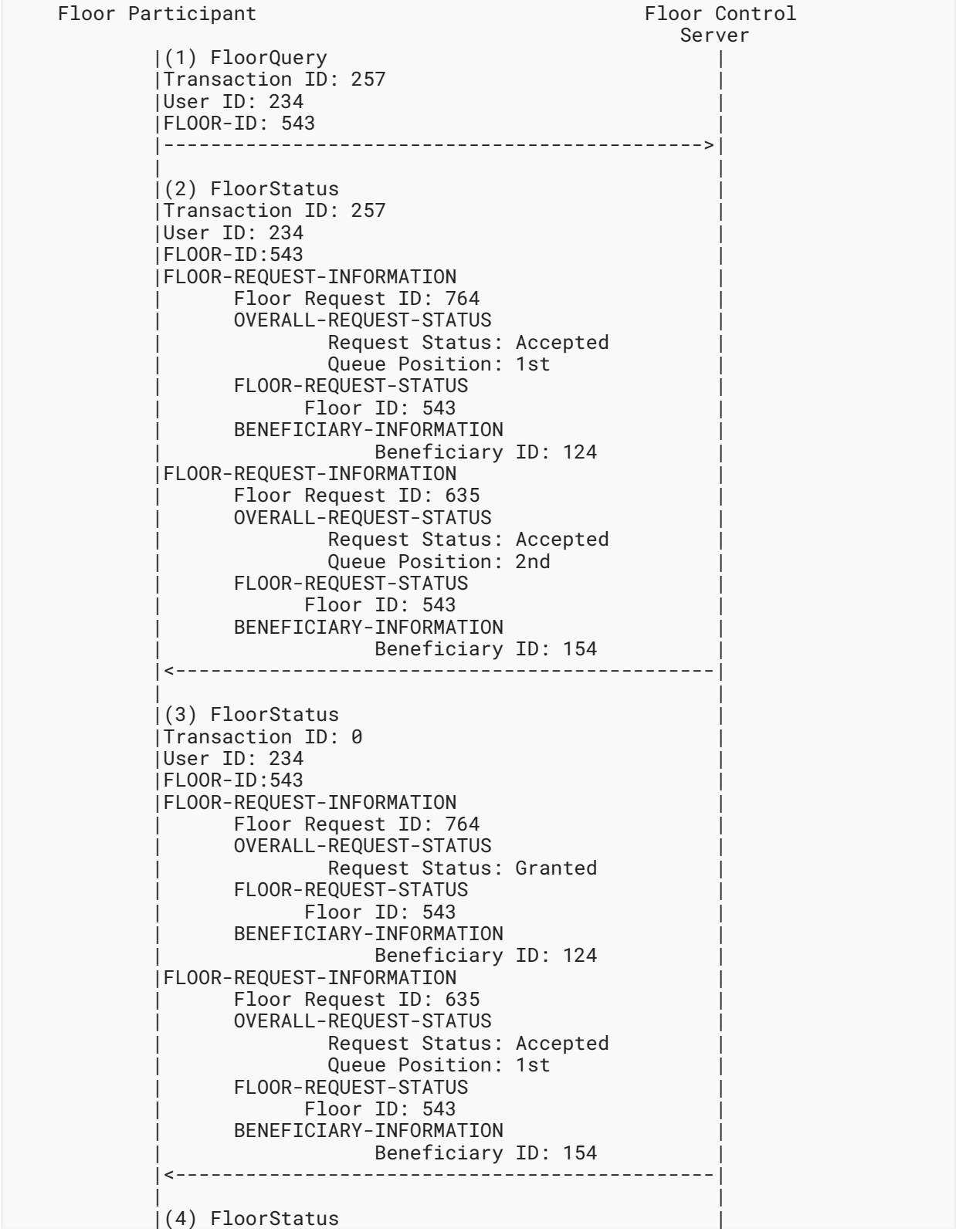


Figure 2: Requesting and releasing a floor

Figure 3 shows how a floor participant requests to be informed on the status of a floor. The first FloorStatus message from the floor control server is the response to the FloorQuery message and, as such, has the same Transaction ID as the FloorQuery message.

Subsequent FloorStatus messages consist of server-initiated transactions, and therefore their Transaction ID is 0 given this example uses a reliable transport. FloorStatus message (2) indicates that there are currently two floor requests for the floor whose Floor ID is 543. FloorStatus message (3) indicates that the floor requests with Floor Request ID 764 has been granted, and the floor request with Floor Request ID 635 is the first in the queue. FloorStatus message (4) indicates that the floor request with Floor Request ID 635 has been granted.


Floor Participant
Floor Control Server
(1) FloorQuery
Transaction ID: 257
User ID: 234
FLOOR-ID: 543
(2) FloorStatus
Transaction ID: 257
User ID: 234
FLOOR-ID:543
FLOOR-REQUEST-INFORMATION
Floor Request ID: 764
OVERALL-REQUEST-STATUS
Request Status: Accepted
Queue Position: 1st
FLOOR-REQUEST-STATUS
Floor ID: 543
BENEFICIARY-INFORMATION
Beneficiary ID: 124
FLOOR-REQUEST-INFORMATION
Floor Request ID: 635
OVERALL-REQUEST-STATUS
Request Status: Accepted
Queue Position: 2nd
FLOOR-REQUEST-STATUS
Floor ID: 543
BENEFICIARY-INFORMATION
Beneficiary ID: 154
(3) FloorStatus
Transaction ID: 0
User ID: 234
FLOOR-ID:543
FLOOR-REQUEST-INFORMATION
Floor Request ID: 764
OVERALL-REQUEST-STATUS
Request Status: Granted
FLOOR-REQUEST-STATUS
Floor ID: 543
BENEFICIARY-INFORMATION
Beneficiary ID: 124
FLOOR-REQUEST-INFORMATION
Floor Request ID: 635
OVERALL-REQUEST-STATUS
Request Status: Accepted
Queue Position: 1st
FLOOR-REQUEST-STATUS
Floor ID: 543
BENEFICIARY-INFORMATION
Beneficiary ID: 154
(4) FloorStatus

```
            |Transaction ID: 0                                |
            |User ID: 234                                     |
            |FLOOR-ID:543                                     |
            |FLOOR-REQUEST-INFORMATION                        |
            |      Floor Request ID: 635                      |
            |      OVERALL-REQUEST-STATUS                     |
            |              Request Status: Granted            |
            |      FLOOR-REQUEST-STATUS                       |
            |            Floor ID: 543                        |
            |      BENEFICIARY-INFORMATION                    |
            |                  Beneficiary ID: 154            |
            |<------------------------------------------------|
```

Figure 3: Obtaining status information about a floor

FloorStatus messages contain information about the floor requests they carry. For example, FloorStatus message (4) indicates that the floor request with Floor Request ID 635 has as the beneficiary (i.e., the participant that holds the floor when a particular floor request is granted) the participant whose User ID is 154. The floor request applies only to the floor whose Floor ID is 543. That is, this is not a multi-floor floor request.

> A multi-floor floor request applies to more than one floor (e.g., a participant wants to be able to speak and write on the whiteboard at the same time). The floor control server treats a multi-floor floor request as an atomic package. That is, the floor control server either grants the request for all floors or denies the request for all floors.

4.2. Floor Chair to Floor Control Server Interface

Figure 4 shows a floor chair instructing a floor control server to grant a floor.

> Note, however, that although the floor control server needs to take into consideration the instructions received in ChairAction messages (e.g., granting a floor), it does not necessarily need to perform them exactly as requested by the floor chair. The operation that the floor control server performs depends on the ChairAction message and on the internal state of the floor control server.

For example, a floor chair may send a ChairAction message granting a floor that was requested as part of an atomic floor request operation that involved several floors. Even if the chair responsible for one of the floors instructs the floor control server to grant the floor, the floor control server will not grant it until the chairs responsible for the other floors agree to grant them as well. In another example, a floor chair may instruct the floor control server to grant a floor to a participant. The floor control server needs to revoke the floor from its current holder before granting it to the new participant.

So, the floor control server is ultimately responsible for keeping a coherent floor state using instructions from floor chairs as input to this state.

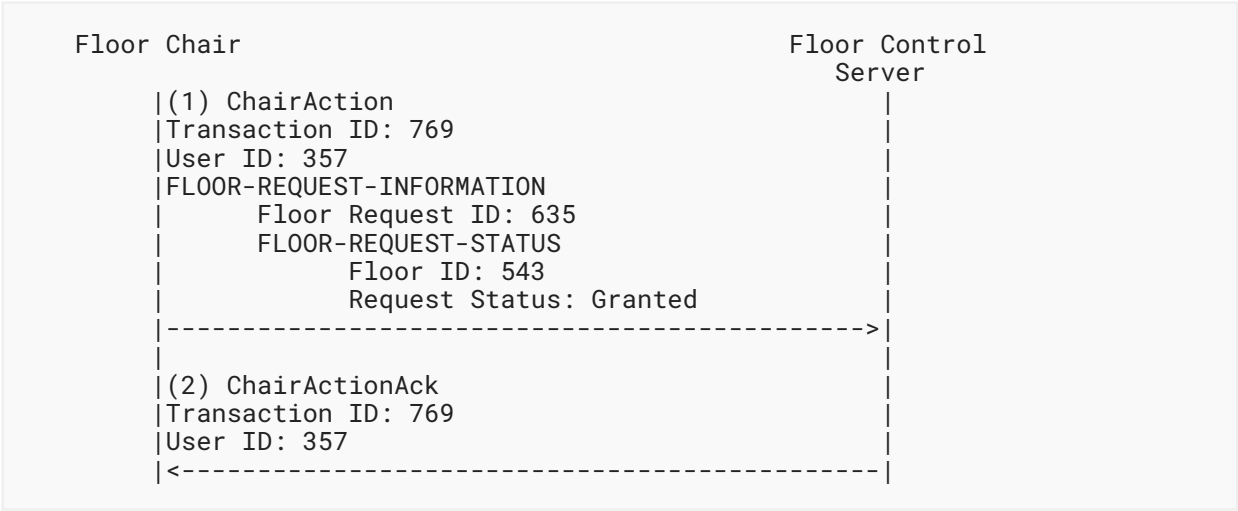


Figure 4: Chair instructing the floor control server

5. Packet Format

BFCP packets consist of a 12-octet COMMON-HEADER followed by attributes. All the protocol values **MUST** be sent in network byte order.

5.1. COMMON-HEADER Format

The following is the format of the COMMON-HEADER.

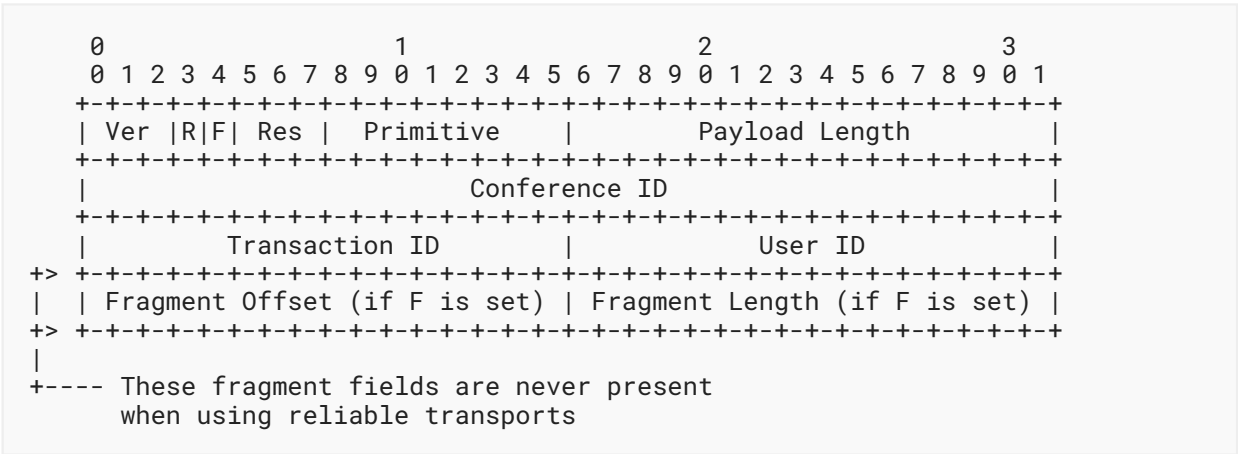


Figure 5: COMMON-HEADER format

Ver: This 3-bit field defines the version of BFCP to which this message adheres. This specification defines two versions: 1 and 2. The version field **MUST** be set to 1 when using BFCP over a reliable transport. The version field **MUST** be set to 2 when using BFCP over an unreliable transport. If a floor control server receives a message with an unsupported version

field value or a message with a version number that is not permitted with the transport over which it was received, the server **MUST** indicate it does not support the protocol version by sending an Error message with parameter value 12 (Unsupported Version). Note that BFCP entities supporting only the [3] subset will not support this parameter value.

R: The Transaction Responder (R) flag bit has relevance only for use of BFCP over an unreliable transport. When cleared, it indicates that this message is a request initiating a new transaction, and the Transaction ID that follows has been generated for this transaction. When set, it indicates that this message is a response to a previous request, and the Transaction ID that follows is the one associated with that request. When BFCP is used over a reliable transport, the flag has no significance and **MUST** be cleared by the sender and **MUST** be ignored by the receiver.

F: The Fragmentation (F) flag bit has relevance only for use of BFCP over an unreliable transport. When cleared, the message is not fragmented. When set, it indicates that the message is a fragment of a large, fragmented BFCP message. (The optional fields Fragment Offset and Fragment Length described below are present only if the F flag is set). When BFCP is used over a reliable transport, the flag has no significance and **MUST** be cleared by the sender, and the flag **MUST** be ignored by the receiver. In the latter case, the receiver should also ignore the Fragment Offset and Fragment Length fields when processing the COMMON-HEADER.

Res: The 3 bits in the reserved field **MUST** be set to zero by the sender of the message and **MUST** be ignored by the receiver.

Primitive: This 8-bit field identifies the main purpose of the message. The following primitive values are defined:

Value	Primitive	Direction
1	FloorRequest	P -> S
2	FloorRelease	P -> S
3	FloorRequestQuery	P -> S ; Ch -> S
4	FloorRequestStatus	P <- S ; Ch <- S
5	UserQuery	P -> S ; Ch -> S
6	UserStatus	P <- S ; Ch <- S
7	FloorQuery	P -> S ; Ch -> S
S: Floor Control Server P: Floor Participant Ch: Floor Chair		

Value	Primitive	Direction
8	FloorStatus	P <- S ; Ch <- S
9	ChairAction	Ch -> S
10	ChairActionAck	Ch <- S
11	Hello	P -> S ; Ch -> S
12	HelloAck	P <- S ; Ch <- S
13	Error	P <- S ; Ch <- S
14	FloorRequestStatusAck	P -> S ; Ch -> S
15	FloorStatusAck	P -> S ; Ch -> S
16	Goodbye	P -> S ; Ch -> S ; P <- S ; Ch <- S
17	GoodbyeAck	P -> S ; Ch -> S ; P <- S ; Ch <- S
S: Floor Control Server P: Floor Participant Ch: Floor Chair		

Table 1: BFCP primitives

Payload Length: This 16-bit field contains the length of the message in 4-octet units, excluding the COMMON-HEADER. If a floor control server receives a message with an incorrect Payload Length field value, the receiving server **MUST** send an Error message with parameter value 13 (Incorrect Message Length) to indicate this and then discard the message. Other entities that receive a message with an incorrect length **MUST** discard the message.

> Note: BFCP is designed to achieve small message size, as explained in Section 1, and BFCP entities are **REQUIRED** to keep the BFCP message size smaller than the size limited by the 16-bit Payload Length field. To convey information not strictly related to floor control, other protocols should be used, such as the XCON Framework (cf. Section 3).

Conference ID: This 32-bit unsigned integer field identifies the conference to which the message belongs. It is **RECOMMENDED** that the conference identifier be randomly chosen. (Note that the use of predictable conference identifiers in conjunction with a nonsecure transport protocol makes BFCP susceptible to off-path data injection attacks, where an attacker can forge a request or response message.)

Transaction ID: This field contains a 16-bit value that allows users to match a given message with its response (see Section 8).

User ID: This field contains a 16-bit unsigned integer that uniquely identifies a participant within a conference.

> The identity used by a participant in BFCP, which is carried in the User ID field, is generally mapped to the identity used by the same participant in the session establishment protocol (e.g., in SIP). The way this mapping is performed is outside the scope of this specification.

Fragment Offset: This optional field is present only if the F flag is set and contains a 16-bit value that specifies the number of 4-octet units contained in previous fragments, excluding the COMMON-HEADER.

Fragment Length: This optional field is present only if the F flag is set and contains a 16-bit value that specifies the number of 4-octet units contained in this fragment, excluding the COMMON-HEADER. BFCP entities that receive message fragments that, individually or collectively, exceed the Payload Length value **MUST** discard the message. Additionally, if the receiver is a floor control server, it **MUST** also send an Error message with parameter value 13 (Incorrect Message Length)

5.2. Attribute Format

BFCP attributes are encoded in TLV (Type-Length-Value) format. Attributes are 32-bit aligned.

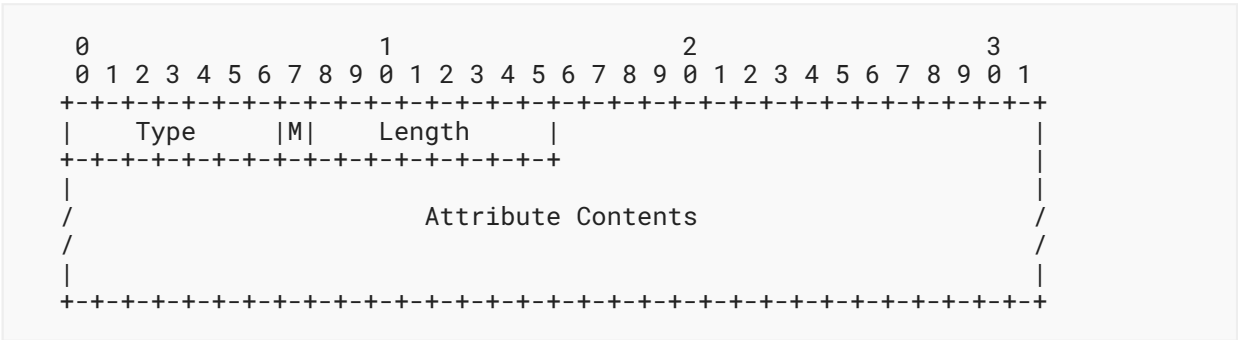


Figure 6: Attribute format

Type: This 7-bit field contains the type of the attribute. Each attribute, identified by its type, has a particular format. The attribute formats defined are:

Unsigned16: The contents of the attribute consist of a 16-bit unsigned integer.

OctetString16: The contents of the attribute consist of 16 bits of arbitrary data.

OctetString: The contents of the attribute consist of arbitrary data of variable length.

Grouped: The contents of the attribute consist of a sequence of attributes.

> Note that extension attributes defined in the future may define new attribute formats.

The following attribute types are defined:

Type	Attribute	Format
1	BENEFICIARY-ID	Unsigned16
2	FLOOR-ID	Unsigned16
3	FLOOR-REQUEST-ID	Unsigned16
4	PRIORITY	OctetString16
5	REQUEST-STATUS	OctetString16
6	ERROR-CODE	OctetString
7	ERROR-INFO	OctetString
8	PARTICIPANT-PROVIDED-INFO	OctetString
9	STATUS-INFO	OctetString
10	SUPPORTED-ATTRIBUTES	OctetString
11	SUPPORTED-PRIMITIVES	OctetString
12	USER-DISPLAY-NAME	OctetString
13	USER-URI	OctetString
14	BENEFICIARY-INFORMATION	Grouped
15	FLOOR-REQUEST-INFORMATION	Grouped
16	REQUESTED-BY-INFORMATION	Grouped
17	FLOOR-REQUEST-STATUS	Grouped
18	OVERALL-REQUEST-STATUS	Grouped

Table 2: BFCP attributes

M:

The 'M' bit, known as the Mandatory bit, indicates whether support of the attribute is **REQUIRED**. If a floor control server receives an unrecognized attribute with the 'M' bit set, the server **MUST** send an Error message with parameter value 4 (Unknown Mandatory Attribute) to indicate this. The 'M' bit is significant for extension attributes defined in other documents only. All attributes specified in this document **MUST** be understood by the receiver so that the setting of the 'M' bit is irrelevant for these. Unrecognized attributes, such as those that might be specified in future extensions, that do not have the 'M' bit set are ignored, but the message is processed.

Length: This 8-bit field contains the length of the attribute in octets, excluding any padding defined for specific attributes. The length of attributes that are not grouped includes the Type, 'M' bit, and Length fields. The Length in grouped attributes is the length of the grouped attribute itself (including Type, 'M' bit, and Length fields) plus the total length (including padding) of all the included attributes.

Attribute Contents: The contents of the different attributes are defined in the following sections.

5.2.1. BENEFICIARY-ID

The following is the format of the BENEFICIARY-ID attribute.

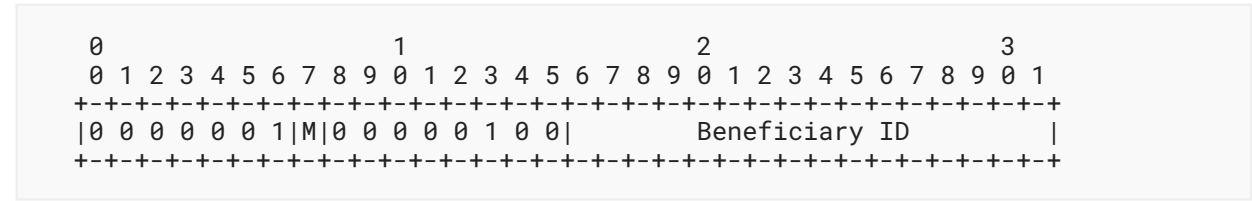


Figure 7: BENEFICIARY-ID format

Beneficiary ID: This field contains a 16-bit value that uniquely identifies a user within a conference.

> Note that although the formats of the Beneficiary ID and of the User ID field in the COMMON-HEADER are similar, their semantics are different. The Beneficiary ID is used in third-party floor requests and to request information about a particular participant.

5.2.2. FLOOR-ID

The following is the format of the FLOOR-ID attribute.

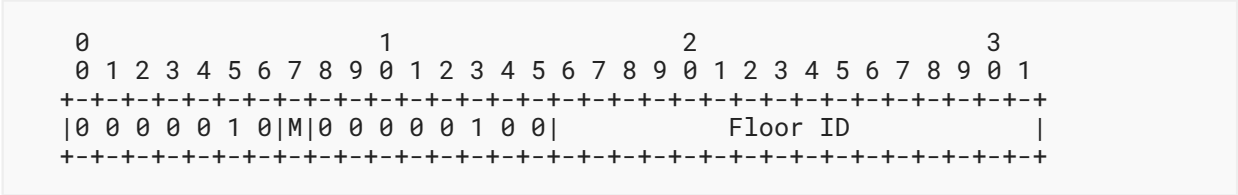


Figure 8: FLOOR-ID format

Floor ID: This field contains a 16-bit value that uniquely identifies a floor within a conference.

5.2.3. FLOOR-REQUEST-ID

The following is the format of the FLOOR-REQUEST-ID attribute.

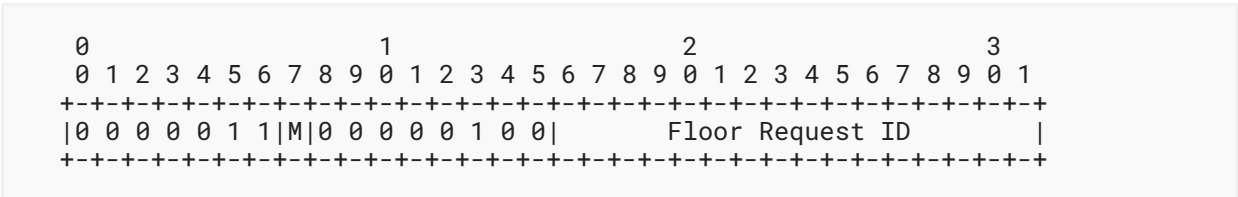


Figure 9: FLOOR-REQUEST-ID format

Floor Request ID: This field contains a 16-bit value that identifies a floor request at the floor control server.

5.2.4. PRIORITY

The following is the format of the PRIORITY attribute.

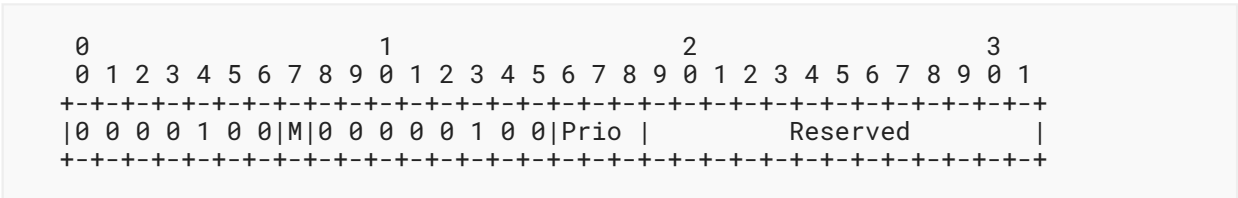


Figure 10: PRIORITY format

Prio: This field contains a 3-bit Priority value, as shown in Table 3. Senders **SHOULD NOT** use values higher than 4 in this field. Receivers **MUST** treat values higher than 4 as if the value received were 4 (Highest). The default Priority value when the PRIORITY attribute is missing is 2 (Normal).

Value	Priority
0	Lowest

Value	Priority
1	Low
2	Normal
3	High
4	Highest

Table 3: Priority values

Reserved: The 13 bits in the reserved field **MUST** be set to zero by the sender of the message and **MUST** be ignored by the receiver.

5.2.5. REQUEST-STATUS

The following is the format of the REQUEST-STATUS attribute.

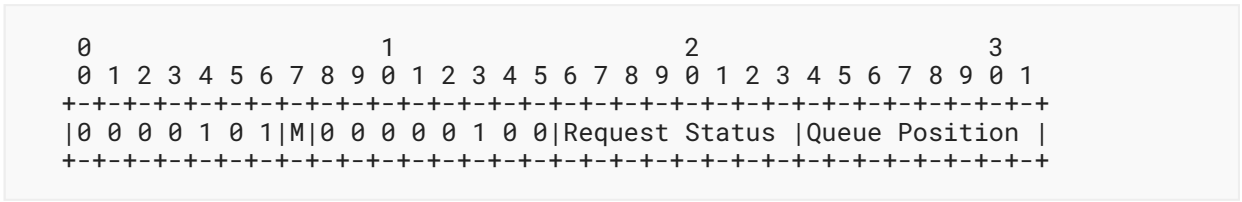


Figure 11: REQUEST-STATUS format

Request Status: This 8-bit field contains the status of the request, as described in the following table.

Value	Status
1	Pending
2	Accepted
3	Granted
4	Denied
5	Cancelled
6	Released
7	Revoked

Table 4: Request Status values

Queue Position: This 8-bit field contains, when applicable, the position of the floor request in the floor request queue at the server. If the Request Status value is different from Accepted, if the floor control server does not implement a floor request queue, or if the floor control server does not want to provide the client with this information, all the bits of this field **SHOULD** be set to zero.

A floor request is in Pending state if the floor control server needs to contact a floor chair in order to accept the floor request, but has not done it yet. Once the floor control chair accepts the floor request, the floor request is moved to the Accepted state.

5.2.6. ERROR-CODE

The following is the format of the ERROR-CODE attribute.

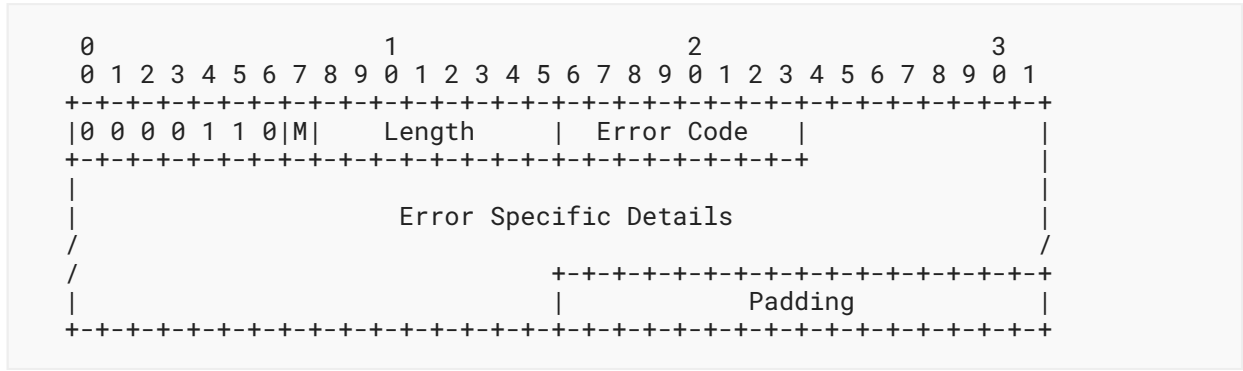


Figure 12: ERROR-CODE format

Error Code: This 8-bit field contains an error code from the following table. If an error code is not recognized by the receiver, then the receiver **MUST** assume that an error exists, and therefore that the original message that triggered the Error message to be sent is processed, but the nature of the error is unclear.

Value	Meaning
1	Conference Does Not Exist
2	User Does Not Exist
3	Unknown Primitive
4	Unknown Mandatory Attribute
5	Unauthorized Operation
6	Invalid Floor ID
7	Floor Request ID Does Not Exist

Value	Meaning
8	You have Already Reached the Maximum Number of Ongoing Floor Requests for This Floor
9	Use TLS
10	Unable to Parse Message
11	Use DTLS
12	Unsupported Version
13	Incorrect Message Length
14	Generic Error

Table 5: Error Code meaning

> Note: The Generic Error error code is intended to be used when an error occurs and the other specific error codes do not apply.

Error Specific Details: Present only for certain error codes. In this document, this field is present only for Error Code 4 (Unknown Mandatory Attribute). See Section 5.2.6.1 for its definition.

Padding: One, two, or three octets of padding added so that the contents of the ERROR-CODE attribute is 32-bit aligned. If the attribute is already 32-bit aligned, no padding is needed.

The Padding bits **MUST** be set to zero by the sender and **MUST** be ignored by the receiver.

5.2.6.1. Error Specific Details for Error Code 4

The following is the format of the Error Specific Details field for Error Code 4.

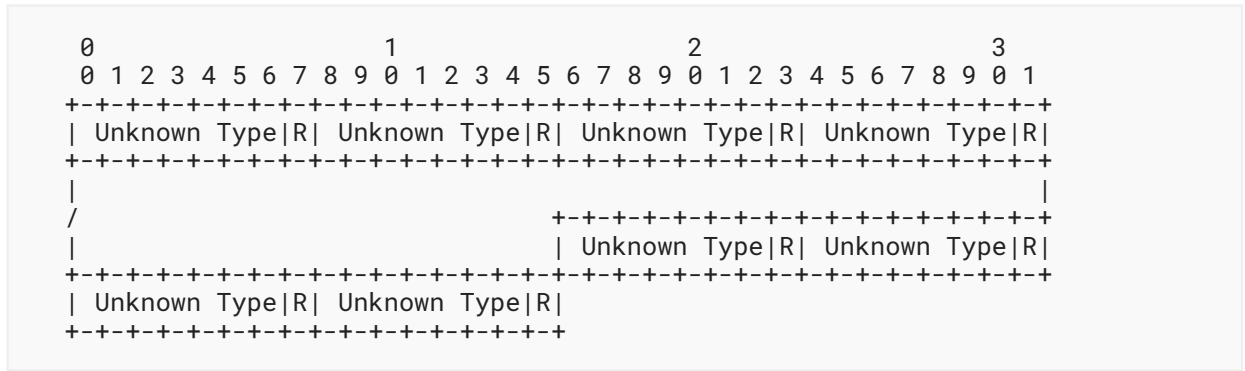


Figure 13: Unknown attributes format

Unknown Type: These 7-bit fields contain the Types of the attributes (which were present in the message that triggered the Error message) that were unknown to the receiver.

Reserved (R): This bit is reserved. It **MUST** be set to zero by the sender of the message and **MUST** be ignored by the receiver.

5.2.7. ERROR-INFO

The following is the format of the ERROR-INFO attribute.

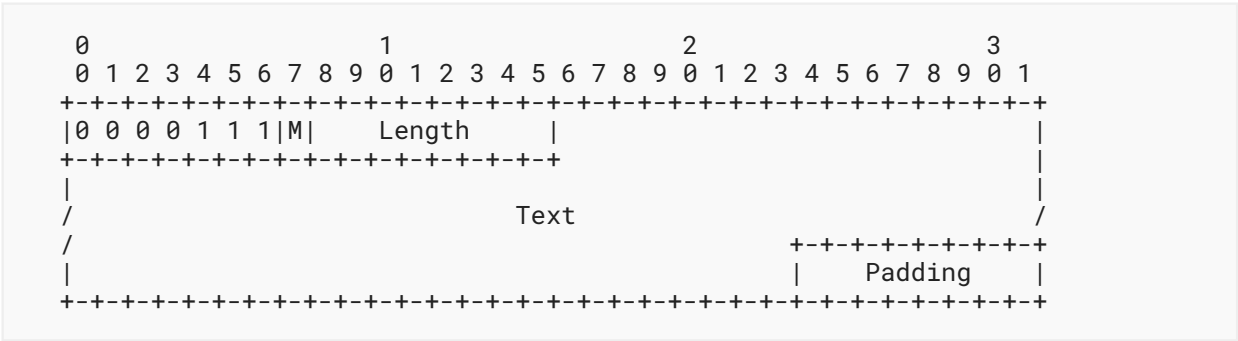


Figure 14: ERROR-INFO format

Text: This field contains UTF-8 encoded text [9].

In some situations, the contents of the Text field may be generated by an automaton. If this automaton has information about the preferred language of the receiver of a particular ERROR-INFO attribute, it **MAY** use this language to generate the Text field.

Padding: One, two, or three octets of padding added so that the contents of the ERROR-INFO attribute is 32-bit aligned. The Padding bits **MUST** be set to zero by the sender and **MUST** be ignored by the receiver. If the attribute is already 32-bit aligned, no padding is needed.

5.2.8. PARTICIPANT-PROVIDED-INFO

The following is the format of the PARTICIPANT-PROVIDED-INFO attribute.

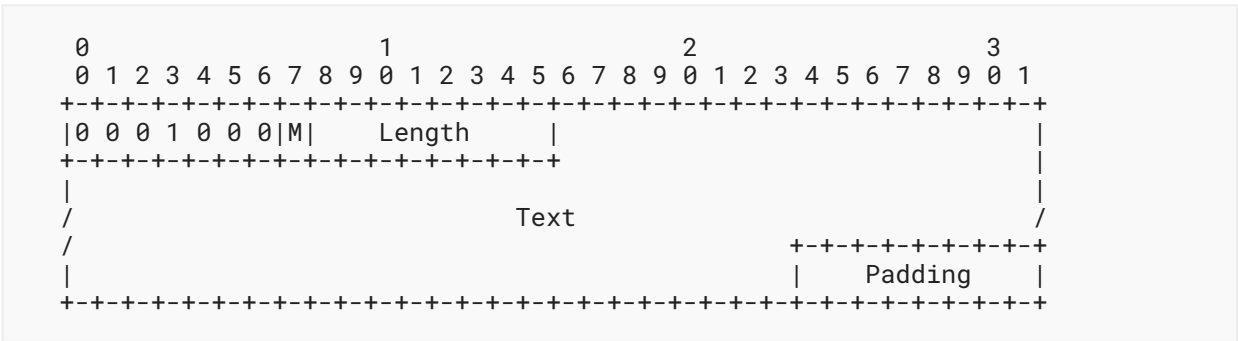


Figure 15: PARTICIPANT-PROVIDED-INFO format

Text: This field contains UTF-8 encoded text [9].

Padding: One, two, or three octets of padding added so that the contents of the PARTICIPANT-PROVIDED-INFO attribute is 32-bit aligned. The Padding bits **MUST** be set to zero by the sender and **MUST** be ignored by the receiver. If the attribute is already 32-bit aligned, no padding is needed.

5.2.9. STATUS-INFO

The following is the format of the STATUS-INFO attribute.

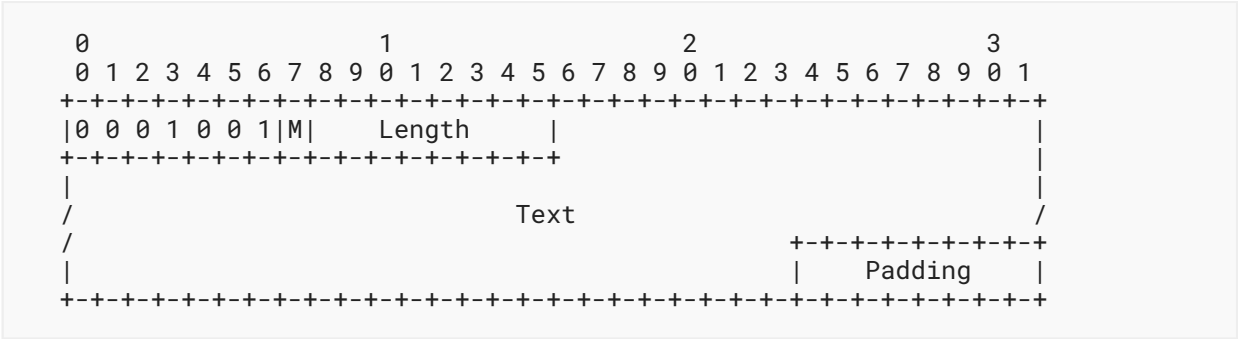


Figure 16: STATUS-INFO format

Text: This field contains UTF-8 encoded text [9].

In some situations, the contents of the Text field may be generated by an automaton. If this automaton has information about the preferred language of the receiver of a particular STATUS-INFO attribute, it **MAY** use this language to generate the Text field.

Padding: One, two, or three octets of padding added so that the contents of the STATUS-INFO attribute is 32-bit aligned. The Padding bits **MUST** be set to zero by the sender and **MUST** be ignored by the receiver. If the attribute is already 32-bit aligned, no padding is needed.

5.2.10. SUPPORTED-ATTRIBUTES

The following is the format of the SUPPORTED-ATTRIBUTES attribute.

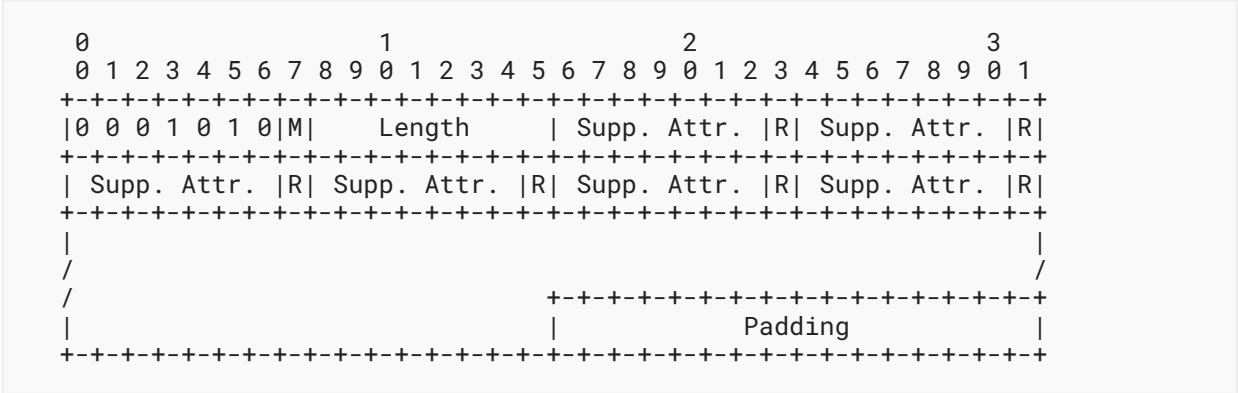


Figure 17: SUPPORTED-ATTRIBUTES format

Supp. Attr.: These fields contain the BFCP attribute types that are supported by the floor control server. See Table 2 for the list of BFCP attributes.

Reserved (R): This bit **MUST** be set to zero upon transmission and **MUST** be ignored upon reception.

Padding: One, two, or three octets of padding added so that the contents of the SUPPORTED-ATTRIBUTES attribute is 32-bit aligned. If the attribute is already 32-bit aligned, no padding is needed.

The Padding bits **MUST** be set to zero by the sender and **MUST** be ignored by the receiver.

5.2.11. SUPPORTED-PRIMITIVES

The following is the format of the SUPPORTED-PRIMITIVES attribute.

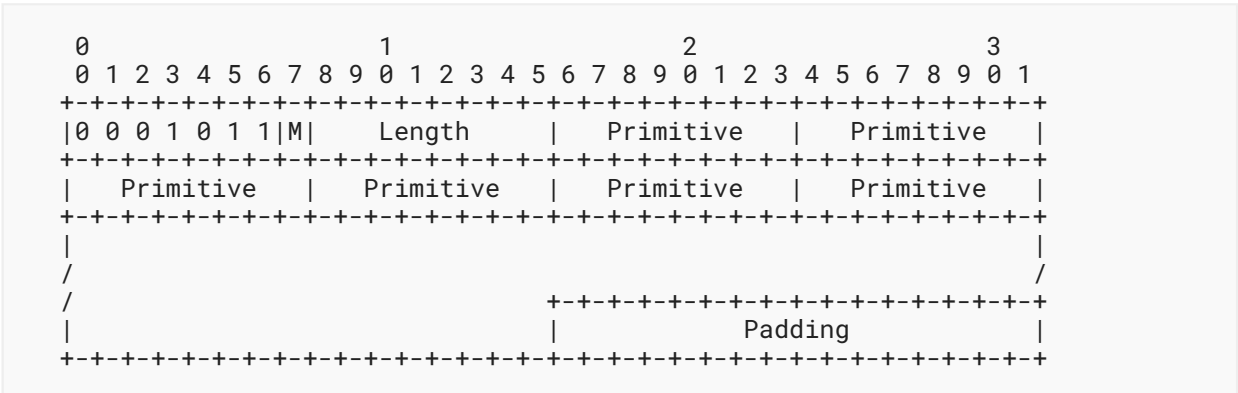


Figure 18: SUPPORTED-PRIMITIVES format

Primitive: These fields contain the types of the BFCP messages that are supported by the floor control server. See Table 1 for the list of BFCP primitives.

Padding: One, two, or three octets of padding added so that the contents of the SUPPORTED-PRIMITIVES attribute is 32-bit aligned. If the attribute is already 32-bit aligned, no padding is needed.

The Padding bits **MUST** be set to zero by the sender and **MUST** be ignored by the receiver.

5.2.12. USER-DISPLAY-NAME

The following is the format of the USER-DISPLAY-NAME attribute.

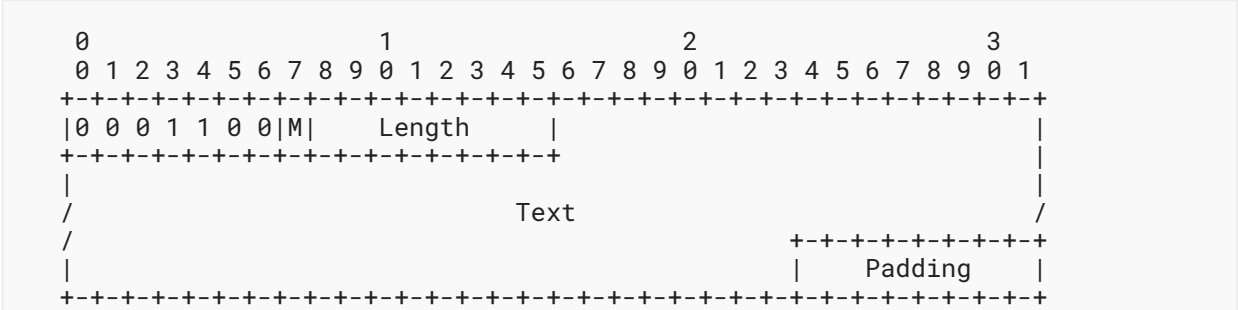


Figure 19: USER-DISPLAY-NAME format

Text: This field contains the UTF-8 encoded name of the user.

Padding: One, two, or three octets of padding added so that the contents of the USER-DISPLAY-NAME attribute is 32-bit aligned. The Padding bits **MUST** be set to zero by the sender and **MUST** be ignored by the receiver. If the attribute is already 32-bit aligned, no padding is needed.

5.2.13. USER-URI

The following is the format of the USER-URI attribute.

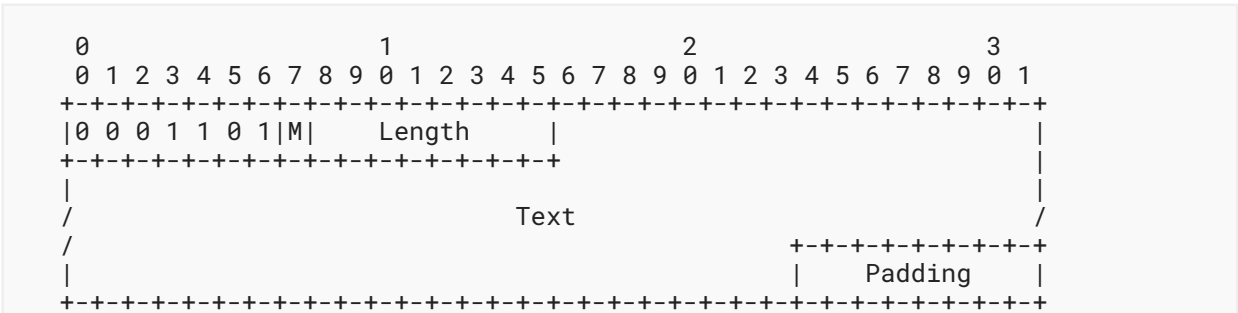


Figure 20: USER-URI format

Text:

This field contains the UTF-8 encoded user's contact URI, that is, the URI used by the user to set up the resources (e.g., media streams) that are controlled by BFCP. For example, in the context of a conference set up by SIP, the USER-URI attribute would carry the SIP URI of the user.

> Messages containing a user's URI in a USER-URI attribute also contain the user's User ID. This way, a client receiving such a message can correlate the user's URI (e.g., the SIP URI the user used to join a conference) with the user's User ID.

Padding: One, two, or three octets of padding added so that the contents of the USER-URI attribute is 32-bit aligned. The Padding bits **MUST** be set to zero by the sender and **MUST** be ignored by the receiver. If the attribute is already 32-bit aligned, no padding is needed.

5.2.14. BENEFICIARY-INFORMATION

The BENEFICIARY-INFORMATION attribute is a grouped attribute that consists of a header, which is referred to as BENEFICIARY-INFORMATION-HEADER, followed by a sequence of attributes. The following is the format of the BENEFICIARY-INFORMATION-HEADER:

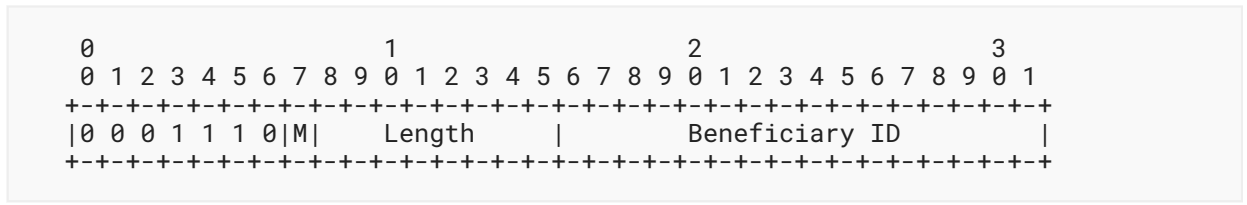


Figure 21: BENEFICIARY-INFORMATION-HEADER format

Beneficiary ID: This field contains a 16-bit value that uniquely identifies a user within a conference.

The following is the ABNF (Augmented Backus-Naur Form) [5] of the BENEFICIARY-INFORMATION grouped attribute. (EXTENSION-ATTRIBUTE refers to extension attributes that may be defined in the future.)

```
BENEFICIARY-INFORMATION =  BENEFICIARY-INFORMATION-HEADER
                           [USER-DISPLAY-NAME]
                           [USER-URI]
                          *EXTENSION-ATTRIBUTE
```

Figure 22: BENEFICIARY-INFORMATION format

5.2.15. FLOOR-REQUEST-INFORMATION

The FLOOR-REQUEST-INFORMATION attribute is a grouped attribute that consists of a header, which is referred to as FLOOR-REQUEST-INFORMATION-HEADER, followed by a sequence of attributes. The following is the format of the FLOOR-REQUEST-INFORMATION-HEADER:

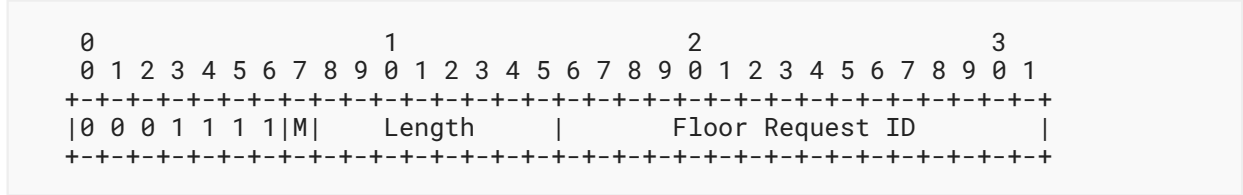


Figure 23: FLOOR-REQUEST-INFORMATION-HEADER format

Floor Request ID: This field contains a 16-bit value that identifies a floor request at the floor control server.

The following is the ABNF of the FLOOR-REQUEST-INFORMATION grouped attribute. (EXTENSION-ATTRIBUTE refers to extension attributes that may be defined in the future.)

```
FLOOR-REQUEST-INFORMATION =   FLOOR-REQUEST-INFORMATION-HEADER
                              [OVERALL-REQUEST-STATUS]
                             1*FLOOR-REQUEST-STATUS
                              [BENEFICIARY-INFORMATION]
                              [REQUESTED-BY-INFORMATION]
                              [PRIORITY]
                              [PARTICIPANT-PROVIDED-INFO]
                             *EXTENSION-ATTRIBUTE
```

Figure 24: FLOOR-REQUEST-INFORMATION format

5.2.16. REQUESTED-BY-INFORMATION

The REQUESTED-BY-INFORMATION attribute is a grouped attribute that consists of a header, which is referred to as REQUESTED-BY-INFORMATION-HEADER, followed by a sequence of attributes. The following is the format of the REQUESTED-BY-INFORMATION-HEADER:

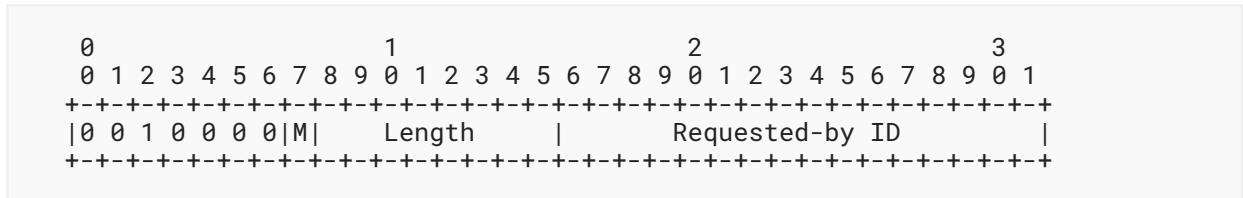


Figure 25: REQUESTED-BY-INFORMATION-HEADER format

Requested-by ID: This field contains a 16-bit value that uniquely identifies a user within a conference.

The following is the ABNF of the REQUESTED-BY-INFORMATION grouped attribute. (EXTENSION-ATTRIBUTE refers to extension attributes that may be defined in the future.)

```
REQUESTED-BY-INFORMATION =  REQUESTED-BY-INFORMATION-HEADER
                            [USER-DISPLAY-NAME]
                            [USER-URI]
                           *EXTENSION-ATTRIBUTE
```

Figure 26: REQUESTED-BY-INFORMATION format

5.2.17. FLOOR-REQUEST-STATUS

The FLOOR-REQUEST-STATUS attribute is a grouped attribute that consists of a header, which is referred to as FLOOR-REQUEST-STATUS-HEADER, followed by a sequence of attributes. The following is the format of the FLOOR-REQUEST-STATUS-HEADER:

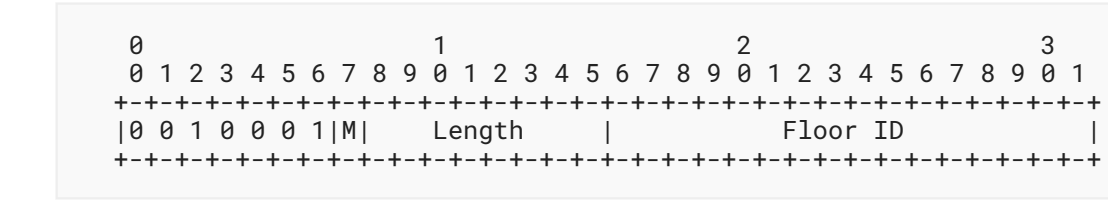


Figure 27: FLOOR-REQUEST-STATUS-HEADER format

Floor ID: this field contains a 16-bit value that uniquely identifies a floor within a conference.

The following is the ABNF of the FLOOR-REQUEST-STATUS grouped attribute. (EXTENSION-ATTRIBUTE refers to extension attributes that may be defined in the future.)

```
FLOOR-REQUEST-STATUS =  FLOOR-REQUEST-STATUS-HEADER
                        [REQUEST-STATUS]
                        [STATUS-INFO]
                       *EXTENSION-ATTRIBUTE
```

Figure 28: FLOOR-REQUEST-STATUS format

5.2.18. OVERALL-REQUEST-STATUS

The OVERALL-REQUEST-STATUS attribute is a grouped attribute that consists of a header, which is referred to as OVERALL-REQUEST-STATUS-HEADER, followed by a sequence of attributes. The following is the format of the OVERALL-REQUEST-STATUS-HEADER:

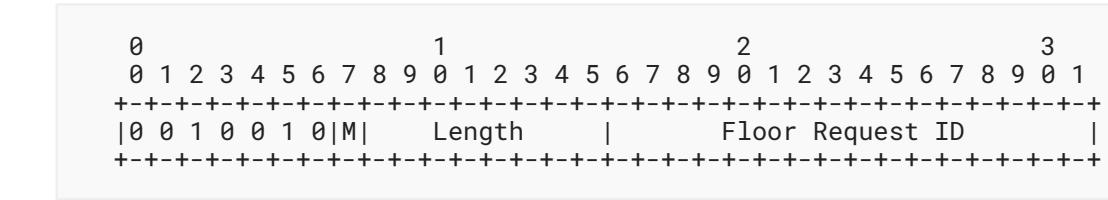


Figure 29: OVERALL-REQUEST-STATUS-HEADER format

Floor Request ID: This field contains a 16-bit value that identifies a floor request at the floor control server.

The following is the ABNF of the OVERALL-REQUEST-STATUS grouped attribute. (EXTENSION-ATTRIBUTE refers to extension attributes that may be defined in the future.)

```
OVERALL-REQUEST-STATUS =  OVERALL-REQUEST-STATUS-HEADER
                          [REQUEST-STATUS]
                          [STATUS-INFO]
                         *EXTENSION-ATTRIBUTE
```

Figure 30: OVERALL-REQUEST-STATUS format

5.3. Message Format

This section contains the normative ABNF (Augmented Backus-Naur Form) [5] of the BFCP messages. Extension attributes that may be defined in the future are referred to as EXTENSION-ATTRIBUTE in the ABNF.

5.3.1. FloorRequest

Floor participants request a floor by sending a FloorRequest message to the floor control server. The following is the format of the FloorRequest message:

```
FloorRequest =   COMMON-HEADER
                1*FLOOR-ID
                  [BENEFICIARY-ID]
                  [PARTICIPANT-PROVIDED-INFO]
                  [PRIORITY]
                  *EXTENSION-ATTRIBUTE
```

Figure 31: FloorRequest format

5.3.2. FloorRelease

Floor participants release a floor by sending a FloorRelease message to the floor control server. Floor participants also use the FloorRelease message to cancel pending floor requests. The following is the format of the FloorRelease message:

```
FloorRelease =  COMMON-HEADER
                FLOOR-REQUEST-ID
               *EXTENSION-ATTRIBUTE
```

Figure 32: FloorRelease format

5.3.3. FloorRequestQuery

Floor participants and floor chairs request information about a floor request by sending a FloorRequestQuery message to the floor control server. The following is the format of the FloorRequestQuery message:

```
FloorRequestQuery =  COMMON-HEADER
                     FLOOR-REQUEST-ID
                    *EXTENSION-ATTRIBUTE
```

Figure 33: FloorRequestQuery format

5.3.4. FloorRequestStatus

The floor control server informs floor participants and floor chairs about the status of their floor requests by sending them FloorRequestStatus messages. The following is the format of the FloorRequestStatus message:

```
FloorRequestStatus =  COMMON-HEADER
                      FLOOR-REQUEST-INFORMATION
                     *EXTENSION-ATTRIBUTE
```

Figure 34: FloorRequestStatus format

5.3.5. UserQuery

Floor participants and floor chairs request information about a participant and the floor requests related to this participant by sending a UserQuery message to the floor control server. The following is the format of the UserQuery message:

```
UserQuery =  COMMON-HEADER
             [BENEFICIARY-ID]
            *EXTENSION-ATTRIBUTE
```

Figure 35: UserQuery format

5.3.6. UserStatus

The floor control server provides information about participants and their related floor requests to floor participants and floor chairs by sending them UserStatus messages. The following is the format of the UserStatus message:

```
UserStatus =  COMMON-HEADER
              [BENEFICIARY-INFORMATION]
             *FLOOR-REQUEST-INFORMATION
             *EXTENSION-ATTRIBUTE
```

Figure 36: UserStatus format

5.3.7. FloorQuery

Floor participants and floor chairs request information about a floor or floors by sending a FloorQuery message to the floor control server. The following is the format of the FloorQuery message:

```
FloorQuery =  COMMON-HEADER
             *FLOOR-ID
             *EXTENSION-ATTRIBUTE
```

Figure 37: FloorQuery format

5.3.8. FloorStatus

The floor control server informs floor participants and floor chairs about the status (e.g., the current holder) of a floor by sending them FloorStatus messages. The following is the format of the FloorStatus message:

```
FloorStatus =  COMMON-HEADER
              *FLOOR-ID
              *FLOOR-REQUEST-INFORMATION
              *EXTENSION-ATTRIBUTE
```

Figure 38: FloorStatus format

5.3.9. ChairAction

Floor chairs send instructions to floor control servers by sending them ChairAction messages. The following is the format of the ChairAction message:

```
ChairAction =  COMMON-HEADER
               FLOOR-REQUEST-INFORMATION
              *EXTENSION-ATTRIBUTE
```

Figure 39: ChairAction format

5.3.10. ChairActionAck

Floor control servers confirm that they have accepted a ChairAction message by sending a ChairActionAck message. The following is the format of the ChairActionAck message:

```
ChairActionAck =  COMMON-HEADER
                 *EXTENSION-ATTRIBUTE
```

Figure 40: ChairActionAck format

5.3.11. Hello

Floor participants and floor chairs **MAY** check the liveness of floor control servers by sending a Hello message. Additionally, clients communicating with a floor control server over an unreliable transport use the Hello message to initiate communication with the server. The following is the format of the Hello message:

```
Hello =  COMMON-HEADER
        *EXTENSION-ATTRIBUTE
```

Figure 41: Hello format

5.3.12. HelloAck

Floor control servers confirm that they are alive on reception of a Hello message by sending a HelloAck message. The following is the format of the HelloAck message:

```
HelloAck =  COMMON-HEADER
            SUPPORTED-PRIMITIVES
            SUPPORTED-ATTRIBUTES
           *EXTENSION-ATTRIBUTE
```

Figure 42: HelloAck format

5.3.13. Error

Floor control servers inform floor participants and floor chairs about errors processing requests by sending them Error messages. The following is the format of the Error message:

```
Error =  COMMON-HEADER
         ERROR-CODE
         [ERROR-INFO]
        *EXTENSION-ATTRIBUTE
```

Figure 43: Error format

5.3.14. FloorRequestStatusAck

When communicating over an unreliable transport, floor participants and chairs acknowledge the receipt of a subsequent FloorRequestStatus message from the floor control server (cf. Section 13.1.2) by sending a FloorRequestStatusAck message. The following is the format of the FloorRequestStatusAck message:

```
FloorRequestStatusAck =  (COMMON-HEADER)
                        *EXTENSION-ATTRIBUTE
```

Figure 44: FloorRequestStatusAck format

5.3.15. FloorStatusAck

When communicating over an unreliable transport, floor participants and chairs acknowledge the receipt of a subsequent FloorStatus message from the floor control server (cf. Section 13.5.2) by sending a FloorStatusAck message. The following is the format of the FloorStatusAck message:

```
FloorStatusAck =  (COMMON-HEADER)
                 *EXTENSION-ATTRIBUTE
```

Figure 45: FloorStatusAck format

5.3.16. Goodbye

BFCP entities communicating over an unreliable transport that wish to dissociate themselves from their remote participant do so through the transmission of a Goodbye. The following is the format of the Goodbye message:

```
Goodbye =  (COMMON-HEADER)
          *EXTENSION-ATTRIBUTE
```

Figure 46: Goodbye format

5.3.17. GoodbyeAck

BFCP entities communicating over an unreliable transport acknowledge the receipt of a Goodbye message from a peer. The following is the format of the GoodbyeAck message:

```
GoodbyeAck =  (COMMON-HEADER)
             *EXTENSION-ATTRIBUTE
```

Figure 47: GoodbyeAck format

6. Transport

The transport over which BFCP entities exchange messages depends on the information the clients obtain for contacting the floor control server, as described in Section 3.2. Two transports are supported: TCP, which is appropriate where connectivity is not impeded by network elements such as NAT devices or media relays; and UDP for those deployments where TCP may not be applicable or appropriate.

> Note: In practice, products are configured to try one transport first and then use the other transport as a fallback. Whether TCP or UDP is chosen as underlying transport depends on the type of product and the deployment environment. See Appendix B for additional considerations.

6.1. Reliable Transport

BFCP entities may elect to exchange BFCP messages using TCP connections. TCP provides an in-order reliable delivery of a stream of bytes. Consequently, message framing needs to be implemented in the application layer. BFCP implements application-layer framing using TLV-encoded attributes.

A client **MUST NOT** use more than one TCP connection to communicate with a given floor control server within a conference. Nevertheless, if the same physical box handles different clients (e.g., a floor chair and a floor participant), which are identified by different User IDs, a separate connection per client is allowed.

If a BFCP entity (a client or a floor control server) receives data that cannot be parsed, the entity **MUST** close the TCP connection, and the connection **SHOULD** be reestablished. Similarly, if a TCP connection cannot deliver a BFCP message and times out or receives an ICMP port unreachable message mid-connection, the TCP connection **SHOULD** be reestablished.

The way connection reestablishment is handled depends on how the client obtains information to contact the floor control server. Once the TCP connection is reestablished, the client **MAY** resend those messages for which it did not get a response from the floor control server.

If a floor control server detects that the TCP connection towards one of the floor participants is lost, it is up to the local policy of the floor control server what to do with the pending floor requests of the floor participant. In any case, it is **RECOMMENDED** that the floor control server keep the floor requests (i.e., that it does not cancel them) while the TCP connection is reestablished.

If a client wishes to end its BFCP connection with a floor control server, the client closes (i.e., a graceful close) the TCP connection towards the floor control server. If a floor control server wishes to end its BFCP connection with a client (e.g., the focus of the conference informs the floor control server that the client has been kicked out of the conference), the floor control server closes (i.e., a graceful close) the TCP connection towards the client.

In cases where a BFCP entity reestablishes a connection due to protocol errors as described above, the entity **SHOULD NOT** repeatedly reestablish the connection. Rather, if the same protocol errors persist, the entity **MUST** cease attempts and **SHOULD** report the error to the human user and/or log the event. This does not preclude the entity from reestablishing a connection when facing a different set of errors. That said, entities **MUST** avoid overloading the server with reestablishment requests. A connection **MUST NOT** be reestablished too frequently. The frequency is a matter of implementation, but **SHOULD NOT** be attempted more than once in a 30 second period of time.

6.2. Unreliable Transport

BFCP entities may elect to exchange BFCP messages using UDP datagrams. UDP is an unreliable transport where neither delivery nor ordering is assured. Each BFCP UDP datagram **MUST** contain exactly one BFCP message or message fragment. To keep large BFCP messages from being fragmented at the IP layer, the fragmentation of BFCP messages that exceed the path MTU size is performed at the BFCP level. Considerations related to fragmentation are covered in Section 6.2.3. The message format for BFCP messages is the same regardless of whether the messages are sent in UDP datagrams or over a TCP stream.

Clients **MUST** announce their presence to the floor control server by sending a Hello message. The floor control server responds to the Hello message with a HelloAck message. The client considers the floor control server as present and available only upon receiving the HelloAck message. The behavior when timers fire, including the determination that a connection is broken, is described in Section 8.3.

As described in Section 8, each request sent by a floor participant or chair forms a client transaction that expects an acknowledgement message from the floor control server within a transaction failure window. Concordantly, messages sent by the floor control server that initiate new transactions (e.g., FloorStatus announcements as part of a FloorQuery subscription) require acknowledgement messages from the floor participant and chair entities to which they were sent.

If a floor control server receives data that cannot be parsed, the receiving server **MUST** send an Error message with parameter value 10 (Unable to Parse Message) indicating receipt of a malformed message, given that it is possible to parse the received message to such an extent that an Error message may be built.

Entities **MUST** have at most one outstanding request transaction per peer at any one time. Implicit subscriptions occur for a client-initiated request transaction whose acknowledgement is implied by the first server-initiated response for that transaction, followed by zero of more subsequent server-initiated messages corresponding to the same transaction. An example is a FloorRequest message for which there are potentially multiple responses from the floor control server as it processes intermediate states until a terminal state (e.g., Granted or Denied) is attained. The subsequent changes in state for the request are new transactions whose Transaction ID is determined by the floor control server and whose receipt by the client participant is acknowledged with a FloorRequestStatusAck message.

By restricting entities to having at most one pending transaction open in a BFCP connection, both the out-of-order receipt of messages as well as the possibility for congestion are mitigated. Additional details regarding congestion control are provided in Section 6.2.1. If a participant receives a server-initiated request (e.g., a FloorStatus from the floor control server) while waiting for a response to a client-initiated transaction (e.g., the participant sent a FloorRequest and is waiting for a FloorRequestStatus response), then the participant **MUST** treat the server-initiated request as superseding any response to its client-initiated transaction. As the floor control server cannot send a second update to the implicit floor status subscription until the first is acknowledged, ordinality is maintained.

If a client wishes to end its BFCP connection with a floor control server, it is **REQUIRED** that the client send a Goodbye message to dissociate itself from any allocated resources. If a floor control server wishes to end its BFCP connection with a client (e.g., the focus of the conference informs the floor control server that the client has been kicked out from the conference), it is **REQUIRED** that the floor control server send a Goodbye message towards the client.

6.2.1. Congestion Control

BFCP may be characterized as generating "low data-volume" traffic, per the classification in [15]. Nevertheless, it is necessary to ensure that suitable and necessary congestion control mechanisms are used for BFCP over UDP. As described in Section 6.2, within the same BFCP connection, every entity -- client or server -- is only allowed to send one request at a time, and await the acknowledging response. This way, at most one datagram is sent per RTT given the message is not lost during transmission. If the message is lost, the request retransmission timer T1 specified in Section 8.3.1 will fire, and the message is retransmitted up to three times, in addition to the original transmission of the message. The default initial interval **MUST** be set to 500 ms, but is adjusted dynamically as described in Section 8.3.1. The interval **MUST** be doubled after each retransmission attempt. This is similar to the specification of the timer A and its initial value T1 in SIP as described in Section 17.1.1.2 of [20], except that the value of T1 in this protocol is not fixed from one transaction to another.

6.2.2. ICMP Error Handling

ICMP is not usable when BFCP is running over an unreliable transport due to risks associated with off-path attacks. Any ICMP messages associated with BFCP running over an unreliable transport **MUST** be ignored.

6.2.3. Fragmentation Handling

When using UDP, a single BFCP message could be fragmented at the IP layer if its overall size exceeds the path MTU of the network. To avoid this happening at the IP layer, a fragmentation scheme for BFCP is defined below.

BFCP is designed for achieving small message size, due to the binary encoding as described in Section 1. The fragmentation scheme is therefore deliberately kept simple and straightforward, since the probability of fragmentation of BFCP messages is small. By design, the fragmentation scheme does not acknowledge individual BFCP message fragments. The whole BFCP message is acknowledged if received completely.

BFCP entities **SHOULD** consider the path MTU size available between the sender and the receiver and **MAY** run MTU discovery, such as described in [25], [26], and [27], for this purpose.

When transmitting a BFCP message with a size greater than the path MTU, the sender **MUST** fragment the message into a series of N contiguous data ranges. The size of each of these N messages **MUST** be smaller than the path MTU to help prevent fragmentation overlap attacks. The value for N is defined as ceil((message size -- COMMON-HEADER size) / (path MTU size -- COMMON-HEADER size)), where ceil is the integer ceiling function, and the COMMON-HEADER size includes the Fragment Offset and Fragment Length fields. The sender then creates N BFCP fragment messages (one for each data range) with the same Transaction ID. The size of each of

these N messages, with the COMMON-HEADER included, **MUST** be smaller than the path MTU. The F flag in the COMMON-HEADER in all the fragments is set to indicate fragmentation of the BFCP message.

For each of these fragments, the Fragment Offset and Fragment Length fields are included in the COMMON-HEADER. The Fragment Offset field denotes the number of 4-octet units contained in the previous fragments, excluding the COMMON-HEADER. The Fragment Length contains the length of the fragment itself, also excluding the COMMON-HEADER. Note that the Payload Length field contains the length of the entire, unfragmented message.

When a BFCP implementation receives a BFCP message fragment, it **MUST** buffer the fragment until either it has received the entire BFCP message, or until the Response Retransmission Timer expires. The state machine should handle the BFCP message only after all the fragments of the message have been received.

If a fragment of a BFCP message is lost, the sender will not receive an acknowledgement for the message. Therefore the sender will retransmit the message with same transaction ID as specified in Section 8.3. If the acknowledgement message sent by the receiver is lost, then the entire message will be resent by the sender. The receiver **MUST** then retransmit the acknowledgement. The receiver **MAY** discard an incomplete buffer utilizing the Response Retransmission Timer, starting the timer after the receipt of the first fragment.

> A Denial of Service (DoS) attack utilizing the fragmentation scheme described above is mitigated by the fact that the Response Retransmission Timer is started after receipt of the first BFCP message fragment. In addition, the Payload Length field can be compared with the Fragment Offset and Fragment Length fields to verify the message fragments as they arrive. To make DoS attacks with spoofed IP addresses difficult, BFCP entities **SHOULD** use the cookie exchange mechanism in DTLS [8].

When deciding the size of the message fragment based on path MTU, the BFCP fragmentation handling should take into account how the DTLS record framing expands the datagram size as described in Section 4.1.1.1 of [8].

6.2.4. NAT Traversal

One of the key benefits of using UDP for BFCP communication is the ability to leverage the existing NAT traversal infrastructure and strategies deployed to facilitate transport of the media associated with the video conferencing sessions. Depending on the given deployment, this infrastructure typically includes some subset of Interactive Connectivity Establishment (ICE) [16].

In order to facilitate the initial establishment of NAT bindings, and to maintain those bindings once established, BFCP entities using an unreliable transport are **RECOMMENDED** to use STUN [14] Binding Indication for keepalives, as described for ICE [16]. Section 6.7 of [28] provides useful recommendations for middlebox interaction when DTLS is used.

> Note: Since the version number is set to 2 when BFCP is used over an unreliable transport, cf. the Ver field in Section 5.1, it is straightforward to distinguish between STUN and BFCP packets even without checking the STUN magic cookie [14].

In order to facilitate traversal of BFCP packets through NATs, BFCP entities using an unreliable transport are **RECOMMENDED** to use symmetric ports for sending and receiving BFCP packets, as recommended for RTP/RTP Control Protocol (RTCP) [13].

7. Lower-Layer Security

BFCP relies on lower-layer security mechanisms to provide replay and integrity protection and confidentiality. BFCP floor control servers and clients (which include both floor participants and floor chairs) **MUST** support TLS for transport over TCP [11] and **MUST** support DTLS [8] for transport over UDP. Any BFCP entity **MAY** support other security mechanisms.

BFCP entities **MUST** support, at a minimum, the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite [7] for backwards compatibility with existing implementations of RFC 4582. In accordance with the recommendations and guidelines in [30], BFCP entities **SHOULD** support the following cipher suites:

- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

8. Protocol Transactions

In BFCP, there are two types of transactions: client-initiated transactions and server-initiated transactions.

Client-initiated transactions consist of a request from a client to a floor control server and a response from the floor control server to the client.

Server-initiated transactions have different requirements and behavior depending on underlying transport:

> When using a reliable transport, server-initiated transactions consist of a single message from a floor control server to a client (notifications). They do not trigger any response.
>
> When using an unreliable transport, server-initiated transactions consist of a request from a floor control server to a client and a response from the client to the floor control server.

When using BFCP over an unreliable transport, retransmission timer T1 (see Section 8.3) **MUST** be used for all requests until the transaction is completed. Note that while T1 varies over time, it remains constant for the duration of a given transaction and is only updated at the completion of a transaction.

8.1. Client Behavior

A client starting a client-initiated transaction **MUST** set the Conference ID in the COMMON-HEADER of the message to the Conference ID for the conference that the client obtained previously.

The client **MUST** set the Transaction ID value in the COMMON-HEADER to a number that is different from 0 and that **MUST NOT** be reused in another message from the client until a response from the server is received for the transaction. The client uses the Transaction ID value to match this message with the response from the floor control server. When using BFCP over an unreliable transport, it is important to choose a Transaction ID value that lets the receiver distinguish the reception of the next message in a sequence of BFCP messages from a retransmission of a previous message. Therefore, BFCP entities using an unreliable transport **MUST** use monotonically increasing Transaction ID values (except for wrap-around).

A client receiving a server-initiated transaction over an unreliable transport **MUST** copy the Transaction ID from the request received from the server into the response.

8.2. Server Behavior

A floor control server sending a response within a client-initiated transaction **MUST** copy the Conference ID, the Transaction ID, and the User ID from the request received from the client into the response.

Server-initiated transactions **MUST** contain a Transaction ID equal to zero when BFCP is used over a reliable transport. Over an unreliable transport, the Transaction ID shall have the same properties as for client-initiated transactions. The server uses the Transaction ID value to match this message with the response from the floor participant or floor chair.

8.3. Timers

When BFCP entities are communicating over an unreliable transport, two retransmission timers are employed to help mitigate the loss of datagrams. Retransmission and response caching are not required when BFCP entities communicate over a reliable transport.

8.3.1. Request Retransmission Timer, T1

T1 is a timer that schedules retransmission of a request until an appropriate response is received or until the maximum number of retransmissions has occurred. The timer is computed using the smoothed round-trip time algorithm defined in [2] with an initial retransmission timeout (RTO) value of 500 ms and clock granularity (G) of 100 ms. In contrast to step 2.4 of Section 2 of [2], if the computed value of RTO is less than 500 ms, then RTO shall be set to 500 ms. Timer T1 **MUST** be adjusted with the reception of a response to each request transmitted in order to compute an accurate RTO value, which is the effective T1 value. The RTT value R is the time in milliseconds from the time when a request is transmitted to the time the initial response to that request is received. Responses to retransmitted packets **MUST NOT** be used to recompute the RTO value, as one cannot determine if a response is to an initial or retransmitted request. If T1 always expires

on the initial transmission of a new request, this would suggest the recommended initial T1 (and RTO) value is too low and **SHOULD** be increased by doubling the initial values of T1 (and RTO) until T1 does not expire when sending a new request.

When retransmitting a request, timer T1 is doubled with each retransmission, failing after three unacknowledged retransmission attempts.

If a valid response is not received for a client- or server-initiated transaction, the implementation **MUST** consider the BFCP connection as broken. Implementations **SHOULD** follow the reestablishment procedure described in Section 6.

8.3.2. Response Retransmission Timer, T2

T2 is a timer that, when fired, signals that the BFCP entity can release knowledge of the transaction against which it is running. It is started upon the first transmission of the response to a request and is the only mechanism by which that response is released by the BFCP entity. Any subsequent retransmissions of the same request can be responded to by replaying the cached response, while that value is retained until the timer has fired. Refer to Section 6.2.3 for this timer's role in the fragmentation handling scheme.

8.3.3. Timer Values

The table below defines the different timers required when BFCP entities communicate over an unreliable transport.

Timer	Description	Value/s
T1	Initial request retransmission timer	0.5 s (initial)
T2	Response retransmission timer	$(T1*2^4)*1.25$

Table 6: Timers

The initial value for T1 is 500 ms, which is an estimate of the RTT for completing the transaction. Computation of this value follows the procedures described in Section 8.3.1, which includes exponential backoffs on retransmissions.

T2 **MUST** be set such that it encompasses all legal retransmissions per T1 plus a factor to accommodate network latency between BFCP entities, processing delays, etc.

9. Authentication and Authorization

BFCP clients **SHOULD** authenticate the floor control server before sending any BFCP message to it or accepting any BFCP message from it. Similarly, floor control servers **SHOULD** authenticate a client before accepting any BFCP message from it or sending any BFCP message to it.

If the signaling or control protocol traffic used to set up the conference is authenticated and confidentiality and integrity protected, and the extensions in this document are supported, the BFCP clients **MUST** authenticate the floor control server, and the floor control servers **MUST** authenticate the client before communicating as described above. Note that BFCP entities supporting only the [3] subset may not comply with this mandatory authentication requirement.

BFCP supports TLS/DTLS mutual authentication between clients and floor control servers, as specified in Section 9.1. This is the **RECOMMENDED** authentication mechanism in BFCP.

Note that future extensions may define additional authentication mechanisms.

In addition to authenticating BFCP messages, floor control servers need to authorize them. On receiving an authenticated BFCP message, the floor control server checks whether the client sending the message is authorized. If the client is not authorized to perform the operation being requested, the floor control server generates an Error message, as described in Section 13.8, with an error code with a value of 5 (Unauthorized Operation). Messages from a client that cannot be authorized **MUST NOT** be processed further.

9.1. TLS/DTLS Based Mutual Authentication

BFCP supports TLS/DTLS based mutual authentication between clients and floor control servers. If TLS/DTLS is used, an initial integrity-protected channel is **REQUIRED** between the client and the floor control server that can be used to exchange their certificates (which **MAY** be self-signed certificates) or, more commonly, the fingerprints of these certificates. These certificates are used at TLS/DTLS establishment time.

> The implementation of such an integrity-protected channel using SIP and the SDP offer/answer model is described in [12].

BFCP messages received over an authenticated TLS/DTLS connection are considered authenticated. A floor control server that receives a BFCP message over TCP/UDP (no TLS/DTLS) **MAY** request the use of TLS/DTLS by generating an Error message, as described in Section 13.8, with an error code with a value of 9 (Use TLS) or a value of 11 (Use DTLS) respectively. Clients configured to require the use of TLS/DTLS **MUST** ignore unauthenticated messages.

Note that future extensions may define additional authentication mechanisms that may not require an initial integrity-protected channel (e.g., authentication based on certificates signed by a certificate authority).

As described in Section 9, floor control servers need to perform authorization before processing any message. In particular, the floor control server **MUST** check that messages arriving over a given authenticated TLS/DTLS connection use an authorized User ID (i.e., a User ID that the user that established the authenticated TLS/DTLS connection is allowed to use).

10. Floor Participant Operations

This section specifies how floor participants can perform different operations, such as requesting a floor, using the protocol elements described in earlier sections. Section 11 specifies operations that are specific to floor chairs, such as instructing the floor control server to grant or revoke a floor, and Section 12 specifies operations that can be performed by any client (i.e., both floor participants and floor chairs).

10.1. Requesting a Floor

A floor participant that wishes to request one or more floors does so by sending a FloorRequest message to the floor control server.

10.1.1. Sending a FloorRequest Message

The ABNF in Section 5.3.1 describes the attributes that a FloorRequest message can contain. In addition, the ABNF specifies normatively which of these attributes are mandatory, and which ones are optional.

The floor participant sets the Conference ID and the Transaction ID in the COMMON-HEADER following the rules given in Section 8.1.

The floor participant sets the User ID in the COMMON-HEADER to the floor participant's identifier. If the sender of the FloorRequest message (identified by the User ID) is not the participant that would eventually get the floor (i.e., a third-party floor request), the sender **SHOULD** add a BENEFICIARY-ID attribute to the message identifying the beneficiary of the floor.

> Note that the namespace for both the User ID and the Beneficiary ID is the same. That is, a given participant is identified by a single 16-bit value that can be used in the User ID in the COMMON-HEADER and in several attributes: BENEFICIARY-ID, BENEFICIARY-INFORMATION, and REQUESTED-BY-INFORMATION.

The floor participant **MUST** insert at least one FLOOR-ID attribute in the FloorRequest message. If the client inserts more than one FLOOR-ID attribute, the floor control server will treat all the floor requests as an atomic package. That is, the floor control server will either grant or deny all the floors in the FloorRequest message.

The floor participant may use a PARTICIPANT-PROVIDED-INFO attribute to state the reason why the floor or floors are being requested. The Text field in the PARTICIPANT-PROVIDED-INFO attribute is intended for human consumption.

The floor participant may request that the server handle the floor request with a certain priority using a PRIORITY attribute.

10.1.2. Receiving a Response

A message from the floor control server is considered a response to the FloorRequest message if the message from the floor control server has the same Conference ID, Transaction ID, and User ID as the FloorRequest message, as described in Section 8.1. On receiving such a response, the floor participant follows the rules in Section 9 that relate to floor control server authentication.

The successful processing of a FloorRequest message at the floor control server involves generating one or several FloorRequestStatus messages. The floor participant obtains a Floor Request ID in the Floor Request ID field of a FLOOR-REQUEST-INFORMATION attribute in the first FloorRequestStatus message from the floor control server. Subsequent FloorRequestStatus messages from the floor control server regarding the same floor request will carry the same Floor Request ID in a FLOOR-REQUEST-INFORMATION attribute as the initial FloorRequestStatus message. This way, the floor participant can associate subsequent incoming FloorRequestStatus messages with the ongoing floor request.

The floor participant obtains information about the status of the floor request in the FLOOR-REQUEST-INFORMATION attribute of each of the FloorRequestStatus messages received from the floor control server. This attribute is a grouped attribute, and as such it includes a number of attributes that provide information about the floor request.

The OVERALL-REQUEST-STATUS attribute provides information about the overall status of the floor request. If the Request Status value is Granted, all the floors that were requested in the FloorRequest message have been granted. If the Request Status value is Denied, all the floors that were requested in the FloorRequest message have been denied. A floor request is considered to be ongoing while it is in the Pending, Accepted, or Granted states. If the floor request value is unknown, then the response is still processed. However, no meaningful value can be reported to the user.

The STATUS-INFO attribute, if present, provides extra information that the floor participant can display to the user.

The FLOOR-REQUEST-STATUS attributes provide information about the status of the floor request as it relates to a particular floor. The STATUS-INFO attribute, if present, provides extra information that the floor participant can display to the user.

The BENEFICIARY-INFORMATION attribute identifies the beneficiary of the floor request in third-party floor requests. The REQUESTED-BY-INFORMATION attribute need not be present in FloorRequestStatus messages received by the floor participant that requested the floor, as this floor participant is already identified by the User ID in the COMMON-HEADER.

The PRIORITY attribute, when present, contains the priority that was requested by the generator of the FloorRequest message.

If the response is an Error message, the floor control server could not process the FloorRequest message for some reason, which is described in the Error message.

10.1.3. Reception of a Subsequent FloorRequestStatus Message

When communicating over an unreliable transport and upon receiving a FloorRequestStatus message from a floor control server, the participant **MUST** respond with a FloorRequestStatusAck message within the transaction failure window to complete the transaction.

10.2. Cancelling a Floor Request and Releasing a Floor

A floor participant that wishes to cancel an ongoing floor request does so by sending a FloorRelease message to the floor control server. The FloorRelease message is also used by floor participants that hold a floor and would like to release it.

10.2.1. Sending a FloorRelease Message

The ABNF in Section 5.3.2 describes the attributes that a FloorRelease message can contain. In addition, the ABNF specifies normatively which of these attributes are mandatory, and which ones are optional.

The floor participant sets the Conference ID and the Transaction ID in the COMMON-HEADER following the rules given in Section 8.1. The floor participant sets the User ID in the COMMON-HEADER to the floor participant's identifier.

> Note that the FloorRelease message is used to release a floor or floors that were granted and to cancel ongoing floor requests (from the protocol perspective, both are ongoing floor requests). Using the same message in both situations helps resolve the race condition that occurs when the FloorRelease message and the FloorGrant message cross each other on the wire.

The floor participant uses the FLOOR-REQUEST-ID that was received in the response to the FloorRequest message that the FloorRelease message is cancelling.

> Note that if the floor participant requested several floors as an atomic operation (i.e., in a single FloorRequest message), all the floors are released as an atomic operation as well (i.e., all are released at the same time).

10.2.2. Receiving a Response

A message from the floor control server is considered a response to the FloorRelease message if the message from the floor control server has the same Conference ID, Transaction ID, and User ID as the FloorRelease message, as described in Section 8.1. On receiving such a response, the floor participant follows the rules in Section 9 that relate to floor control server authentication.

If the response is a FloorRequestStatus message, the Request Status value in the OVERALL-REQUEST-STATUS attribute (within the FLOOR-REQUEST-INFORMATION grouped attribute) will be Cancelled or Released.

If the response is an Error message, the floor control server could not process the FloorRequest message for some reason, which is described in the Error message.

It is possible that the FloorRelease message crosses on the wire with a FloorRequestStatus message from the server with a Request Status different from Cancelled or Released. In any case, such a FloorRequestStatus message will not be a response to the FloorRelease message, as its Transaction ID will not match that of the FloorRelease.

11. Chair Operations

This section specifies how floor chairs can instruct the floor control server to grant or revoke a floor using the protocol elements described in earlier sections.

Floor chairs that wish to send instructions to a floor control server do so by sending a ChairAction message.

11.1. Sending a ChairAction Message

The ABNF in Section 5.3.9 describes the attributes that a ChairAction message can contain. In addition, the ABNF specifies normatively which of these attributes are mandatory, and which ones are optional.

The floor chair sets the Conference ID and the Transaction ID in the COMMON-HEADER following the rules given in Section 8.1. The floor chair sets the User ID in the COMMON-HEADER to the floor chair's identifier.

The ChairAction message contains instructions that apply to one or more floors within a particular floor request. The floor or floors are identified by the FLOOR-REQUEST-STATUS attributes and the floor request is identified by the FLOOR-REQUEST-INFORMATION-HEADER, which are carried in the ChairAction message.

For example, if a floor request consists of two floors that depend on different floor chairs, each floor chair will grant its floor within the floor request. Once both chairs have granted their floor, the floor control server will grant the floor request as a whole. On the other hand, if one of the floor chairs denies its floor, the floor control server will deny the floor request as a whole, regardless of the other floor chair's decision.

The floor chair provides the new status of the floor request as it relates to a particular floor using a FLOOR-REQUEST-STATUS attribute. If the new status of the floor request is Accepted, the floor chair **MAY** use the Queue Position field to provide a queue position for the floor request. If the floor chair does not wish to provide a queue position, all the bits of the Queue Position field **MUST** be set to zero. The floor chair **MUST** use the Status Revoked to revoke a floor that was granted (i.e., Granted status) and **MUST** use the Status Denied to reject floor requests in any other status (e.g., Pending and Accepted).

The floor chair **MAY** add an OVERALL-REQUEST-STATUS attribute to the ChairAction message to provide a new overall status for the floor request. If the new overall status of the floor request is Accepted, the floor chair can use the Queue Position field to provide a queue position for the floor request.

> Note that a particular floor control server can implement a different queue for each floor containing all the floor requests that relate to that particular floor, a general queue for all floor requests, or both. Also note that a floor request can involve several floors and that a ChairAction message can only deal with a subset of these floors (e.g., if a single floor chair is not authorized to manage all the floors). In this case, the floor control server will combine the instructions received from the different floor chairs in FLOOR-REQUEST-STATUS attributes to come up with the overall status of the floor request.
>
> Note that, while the action of a floor chair may communicate information in the OVERALL-REQUEST-STATUS attribute, the floor control server may override, modify, or ignore this field's content.

The floor chair **MAY** include STATUS-INFO attributes to state the reason why the floor or floors are being accepted, granted, or revoked. The Text in the STATUS-INFO attribute is intended for human consumption.

11.2. Receiving a Response

A message from the floor control server is considered a response to the ChairAction message if the message from the server has the same Conference ID, Transaction ID, and User ID as the ChairAction message, as described in Section 8.1. On receiving such a response, the floor chair follows the rules in Section 9 that relate to floor control server authentication.

A ChairActionAck message from the floor control server confirms that the floor control server has accepted the ChairAction message. An Error message indicates that the floor control server could not process the ChairAction message for some reason, which is described in the Error message.

12. General Client Operations

This section specifies operations that can be performed by any client. That is, they are not specific to floor participants or floor chairs. They can be performed by both.

12.1. Requesting Information about Floors

A client can obtain information about the status of a floor or floors in different ways, which include using BFCP and using out-of-band mechanisms. Clients using BFCP to obtain such information use the procedures described in this section.

Clients request information about the status of one or several floors by sending a FloorQuery message to the floor control server.

12.1.1. Sending a FloorQuery Message

The ABNF in Section 5.3.7 describes the attributes that a FloorQuery message can contain. In addition, the ABNF specifies normatively which of these attributes are mandatory, and which ones are optional.

The client sets the Conference ID and the Transaction ID in the COMMON-HEADER following the rules given in Section 8.1. The client sets the User ID in the COMMON-HEADER to the client's identifier.

The client inserts in the message all the Floor IDs it wants to receive information about. The floor control server will send periodic information about all of these floors. If the client does not want to receive information about a particular floor any longer, it sends a new FloorQuery message removing the FLOOR-ID of this floor. If the client does not want to receive information about any floor any longer, it sends a FloorQuery message with no FLOOR-ID attribute.

12.1.2. Receiving a Response

A message from the floor control server is considered a response to the FloorQuery message if the message from the floor control server has the same Conference ID, Transaction ID, and User ID as the FloorQuery message, as described in Section 8.1. On receiving such a response, the client follows the rules in Section 9 that relate to floor control server authentication.

On reception of the FloorQuery message, the floor control server **MUST** respond with a FloorStatus message or with an Error message. If the response is a FloorStatus message, it will contain information about one of the floors the client requested information about. If the client did not include any FLOOR-ID attribute in its FloorQuery message (i.e., the client does not want to receive information about any floor any longer), the FloorStatus message from the floor control server will not include any FLOOR-ID attribute either.

FloorStatus messages that carry information about a floor contain a FLOOR-ID attribute that identifies the floor. After this attribute, FloorStatus messages contain information about existing (one or more) floor requests that relate to that floor. The information about each particular floor request is encoded in a FLOOR-REQUEST-INFORMATION attribute. This grouped attribute carries a Floor Request ID that identifies the floor request, followed by a set of attributes that provide information about the floor request.

After the first FloorStatus, the floor control server will continue sending FloorStatus messages, periodically informing the client about changes on the floors the client requested information about.

12.1.3. Reception of a Subsequent FloorStatus Message

When communicating over an unreliable transport and upon receiving a FloorStatus message from a floor control server, the participant **MUST** respond with a FloorStatusAck message within the transaction failure window to complete the transaction.

12.2. Requesting Information about Floor Requests

A client can obtain information about the status of one or several floor requests in different ways, which include using BFCP and using out-of-band mechanisms. Clients using BFCP to obtain such information use the procedures described in this section.

Clients request information about the current status of a floor request by sending a FloorRequestQuery message to the floor control server.

Requesting information about a particular floor request is useful in a number of situations. For example, on reception of a FloorRequest message, a floor control server may choose to return FloorRequestStatus messages only when the floor request changes its state (e.g., from Accepted to Granted), but not when the floor request advances in its queue. In this situation, if the user requests it, the floor participant can use a FloorRequestQuery message to poll the floor control server for the status of the floor request.

12.2.1. Sending a FloorRequestQuery Message

The ABNF in Section 5.3.3 describes the attributes that a FloorRequestQuery message can contain. In addition, the ABNF specifies normatively which of these attributes are mandatory, and which ones are optional.

The client sets the Conference ID and the Transaction ID in the COMMON-HEADER following the rules given in Section 8.1. The client sets the User ID in the COMMON-HEADER to the client's identifier.

The client **MUST** insert a FLOOR-REQUEST-ID attribute that identifies the floor request at the floor control server.

12.2.2. Receiving a Response

A message from the floor control server is considered a response to the FloorRequestQuery message if the message from the floor control server has the same Conference ID, Transaction ID, and User ID as the FloorRequestQuery message, as described in Section 8.1. On receiving such a response, the client follows the rules in Section 9 that relate to floor control server authentication.

If the response is a FloorRequestStatus message, the client obtains information about the status of the FloorRequest the client requested information about in a FLOOR-REQUEST-INFORMATION attribute.

If the response is an Error message, the floor control server could not process the FloorRequestQuery message for some reason, which is described in the Error message.

12.3. Requesting Information about a User

A client can obtain information about a participant and the floor requests related to this participant in different ways, which include using BFCP and using out-of-band mechanisms. Clients using BFCP to obtain such information use the procedures described in this section.

Clients request information about a participant and the floor requests related to this participant by sending a UserQuery message to the floor control server.

This functionality may be useful for floor chairs or floor participants interested in the display name and the URI of a particular floor participant. In addition, a floor participant may find it useful to request information about itself. For example, a floor participant, after experiencing connectivity problems (e.g., its TCP connection with the floor control server was down for a while and eventually was re-established), may need to request information about all the floor requests associated to itself that still exist.

12.3.1. Sending a UserQuery Message

The ABNF in Section 5.3.5 describes the attributes that a UserQuery message can contain. In addition, the ABNF specifies normatively which of these attributes are mandatory, and which ones are optional.

The client sets the Conference ID and the Transaction ID in the COMMON-HEADER following the rules given in Section 8.1. The client sets the User ID in the COMMON-HEADER to the client's identifier.

If the floor participant the client is requesting information about is not the client issuing the UserQuery message (which is identified by the User ID in the COMMON-HEADER of the message), the client **MUST** insert a BENEFICIARY-ID attribute.

12.3.2. Receiving a Response

A message from the floor control server is considered a response to the UserQuery message if the message from the floor control server has the same Conference ID, Transaction ID, and User ID as the UserQuery message, as described in Section 8.1. On receiving such a response, the client follows the rules in Section 9 that relate to floor control server authentication.

If the response is a UserStatus message, the client obtains information about the floor participant in a BENEFICIARY-INFORMATION grouped attribute and about the status of the floor requests associated with the floor participant in FLOOR-REQUEST-INFORMATION attributes.

If the response is an Error message, the floor control server could not process the UserQuery message for some reason, which is described in the Error message.

12.4. Obtaining the Capabilities of a Floor Control Server

A client that wishes to obtain the capabilities of a floor control server does so by sending a Hello message to the floor control server.

12.4.1. Sending a Hello Message

The ABNF in Section 5.3.11 describes the attributes that a Hello message can contain. In addition, the ABNF specifies normatively which of these attributes are mandatory, and which ones are optional.

The client sets the Conference ID and the Transaction ID in the COMMON-HEADER following the rules given in Section 8.1. The client sets the User ID in the COMMON-HEADER to the client's identifier.

12.4.2. Receiving Responses

A message from the floor control server is considered a response to the Hello message by the client if the message from the floor control server has the same Conference ID, Transaction ID, and User ID as the Hello message, as described in Section 8.1. On receiving such a response, the client follows the rules in Section 9 that relate to floor control server authentication.

If the response is a HelloAck message, the floor control server could process the Hello message successfully. The SUPPORTED-PRIMITIVES and SUPPORTED-ATTRIBUTES attributes indicate which primitives and attributes, respectively, are supported by the server.

If the response is an Error message, the floor control server could not process the Hello message for some reason, which is described in the Error message.

13. Floor Control Server Operations

This section specifies how floor control servers can perform different operations, such as granting a floor, using the protocol elements described in earlier sections.

On reception of a message from a client, the floor control server **MUST** check whether the value of the primitive is supported. If it is not, the floor control server **MUST** send an Error message, as described in Section 13.8, with Error Code 3 (Unknown Primitive).

On reception of a message from a client, the floor control server **MUST** check whether the value of the Conference ID matched an existing conference. If it does not, the floor control server **MUST** send an Error message, as described in Section 13.8, with Error Code 1 (Conference Does Not Exist).

On reception of a message from a client, the floor control server follows the rules in Section 9 that relate to the authentication of the message.

On reception of a message from a client, the floor control server **MUST** check whether it understands all the mandatory ('M' bit set) attributes in the message. If the floor control server does not understand all of them, the floor control server **MUST** send an Error message, as described in Section 13.8, with Error Code 4 (Unknown Mandatory Attribute). The Error message **SHOULD** list the attributes that were not understood.

13.1. Reception of a FloorRequest Message

On reception of a FloorRequest message, the floor control server follows the rules in Section 9 that relate to client authentication and authorization. If while processing the FloorRequest message, the floor control server encounters an error, it **MUST** generate an Error response following the procedures described in Section 13.8.

> BFCP allows floor participants to have several ongoing floor requests for the same floor (e.g., the same floor participant can occupy more than one position in a queue at the same time). A floor control server that only supports a certain number of ongoing floor requests per floor participant (e.g., one) can use Error Code 8 (You have Already Reached the Maximum Number of Ongoing Floor Requests for This Floor) to inform the floor participant.

When communicating over an unreliable transport and upon receiving a FloorRequest from a participant, the floor control server **MUST** respond with a FloorRequestStatus message within the transaction failure window to complete the transaction.

13.1.1. Generating the First FloorRequestStatus Message

The successful processing of a FloorRequest message by a floor control server involves generating one or several FloorRequestStatus messages, the first of which **SHOULD** be generated as soon as possible. If the floor control server cannot accept, grant, or deny the floor request right away (e.g., a decision from a chair is needed), it **SHOULD** use a Request Status value of Pending in the OVERALL-REQUEST-STATUS attribute (within the FLOOR-REQUEST-INFORMATION grouped attribute) of the first FloorRequestStatus message it generates.

> The policy that a floor control server follows to grant or deny floors is outside the scope of this document. A given floor control server may perform these decisions automatically while another may contact a human acting as a chair every time a decision needs to be made.

The floor control server **MUST** copy the Conference ID, the Transaction ID, and the User ID from the FloorRequest into the FloorRequestStatus, as described in Section 8.2. Additionally, the floor control server **MUST** add a FLOOR-REQUEST-INFORMATION grouped attribute to the FloorRequestStatus. The attributes contained in this grouped attribute carry information about the floor request.

The floor control server **MUST** assign an identifier that is unique within the conference to this floor request, and **MUST** insert it in the Floor Request ID field of the FLOOR-REQUEST-INFORMATION attribute. This identifier will be used by the floor participant (or by a chair or chairs) to refer to this specific floor request in the future.

The floor control server **MUST** copy the Floor IDs in the FLOOR-ID attributes of the FloorRequest into the FLOOR-REQUEST-STATUS attributes in the FLOOR-REQUEST-INFORMATION grouped attribute. These Floor IDs identify the floors being requested (i.e., the floors associated with this particular floor request).

The floor control server **SHOULD** copy (if present) the contents of the BENEFICIARY-ID attribute from the FloorRequest into a BENEFICIARY-INFORMATION attribute inside the FLOOR-REQUEST-INFORMATION grouped attribute. Additionally, the floor control server **MAY** provide the display name and the URI of the beneficiary in this BENEFICIARY-INFORMATION attribute.

The floor control server **MAY** provide information about the requester of the floor in a REQUESTED-BY-INFORMATION attribute inside the FLOOR-REQUEST-INFORMATION grouped attribute.

The floor control server **MAY** copy (if present) the PRIORITY attribute from the FloorRequest into the FLOOR-REQUEST-INFORMATION grouped attribute.

> Note that this attribute carries the priority requested by the participant. The priority that the floor control server assigns to the floor request depends on the priority requested by the participant and the rights the participant has according to the policy of the conference. For example, a participant that is only allowed to use the Normal priority may request Highest priority for a floor request. In that case, the floor control server would ignore the priority requested by the participant.

The floor control server **MAY** copy (if present) the PARTICIPANT-PROVIDED-INFO attribute from the FloorRequest into the FLOOR-REQUEST-INFORMATION grouped attribute.

13.1.2. Generation of Subsequent FloorRequestStatus Messages

A floor request is considered to be ongoing as long as it is not in the Cancelled, Released, or Revoked states. If the OVERALL-REQUEST-STATUS attribute (inside the FLOOR-REQUEST-INFORMATION grouped attribute) of the first FloorRequestStatus message generated by the floor control server did not indicate any of these states, the floor control server will need to send subsequent FloorRequestStatus messages.

When the status of the floor request changes, the floor control server **SHOULD** send new FloorRequestStatus messages with the appropriate Request Status. The floor control server **MUST** add a FLOOR-REQUEST-INFORMATION attribute with a Floor Request ID equal to the one sent in the first FloorRequestStatus message to any new FloorRequestStatus related to the same floor request. (The Floor Request ID identifies the floor request to which the FloorRequestStatus applies.)

When using BFCP over a reliable transport, the floor control server **MUST** set the Transaction ID of subsequent FloorRequestStatus messages to zero. When using BFCP over an unreliable transport, the Transaction ID **MUST** be non-zero and unique in the context of outstanding transactions over an unreliable transport as described in Section 8.

> The rate at which the floor control server sends FloorRequestStatus messages is a matter of local policy. A floor control server may choose to send a new FloorRequestStatus message every time the floor request moves in the floor request queue, while another may choose only to send a new FloorRequestStatus message when the floor request is Granted or Denied.

The floor control server may add a STATUS-INFO attribute to any of the FloorRequestStatus messages it generates to provide extra information about its decisions regarding the floor request (e.g., why it was denied).

> Floor participants and floor chairs may request to be informed about the status of a floor following the procedures in Section 12.1. If the processing of a floor request changes the status of a floor (e.g., the floor request is granted and consequently the floor has a new holder), the floor control server needs to follow the procedures in Section 13.5 to inform the clients that have requested that information.

The COMMON-HEADER and the rest of the attributes are the same as in the first FloorRequestStatus message.

The floor control server can discard the state information about a particular floor request when this reaches a status of Cancelled, Released, or Revoked.

When communicating over an unreliable transport and a FloorRequestStatusAck message is not received within the transaction failure window, the floor control server **MUST** retransmit the FloorRequestStatus message according to Section 6.2.

13.2. Reception of a FloorRequestQuery Message

On reception of a FloorRequestQuery message, the floor control server follows the rules in Section 9 that relate to client authentication and authorization. If while processing the FloorRequestQuery message, the floor control server encounters an error, it **MUST** generate an Error response following the procedures described in Section 13.8.

The successful processing of a FloorRequestQuery message by a floor control server involves generating a FloorRequestStatus message, which **SHOULD** be generated as soon as possible.

When communicating over an unreliable transport and upon receiving a FloorRequestQuery from a participant, the floor control server **MUST** respond with a FloorRequestStatus message within the transaction failure window to complete the transaction.

The floor control server **MUST** copy the Conference ID, the Transaction ID, and the User ID from the FloorRequestQuery message into the FloorRequestStatus message, as described in Section 8.2. Additionally, the floor control server **MUST** include information about the floor request in the FLOOR-REQUEST-INFORMATION grouped attribute to the FloorRequestStatus.

The floor control server **MUST** copy the contents of the FLOOR-REQUEST-ID attribute from the FloorRequestQuery message into the Floor Request ID field of the FLOOR-REQUEST-INFORMATION attribute.

The floor control server **MUST** add FLOOR-REQUEST-STATUS attributes to the FLOOR-REQUEST-INFORMATION grouped attribute identifying the floors being requested (i.e., the floors associated with the floor request identified by the FLOOR-REQUEST-ID attribute).

The floor control server **SHOULD** add a BENEFICIARY-ID attribute to the FLOOR-REQUEST-INFORMATION grouped attribute identifying the beneficiary of the floor request. Additionally, the floor control server **MAY** provide the display name and the URI of the beneficiary in this BENEFICIARY-INFORMATION attribute.

The floor control server **MAY** provide information about the requester of the floor in a REQUESTED-BY-INFORMATION attribute inside the FLOOR-REQUEST-INFORMATION grouped attribute.

The floor control server **MAY** provide the reason why the floor participant requested the floor in a PARTICIPANT-PROVIDED-INFO.

The floor control server **MAY** also add to the FLOOR-REQUEST-INFORMATION grouped attribute a PRIORITY attribute with the Priority value requested for the floor request and a STATUS-INFO attribute with extra information about the floor request.

The floor control server **MUST** add an OVERALL-REQUEST-STATUS attribute to the FLOOR-REQUEST-INFORMATION grouped attribute with the current status of the floor request. The floor control server **MAY** provide information about the status of the floor request as it relates to each of the floors being requested in the FLOOR-REQUEST-STATUS attributes.

13.3. Reception of a UserQuery Message

On reception of a UserQuery message, the floor control server follows the rules in Section 9 that relate to client authentication and authorization. If while processing the UserQuery message, the floor control server encounters an error, it **MUST** generate an Error response following the procedures described in Section 13.8.

The successful processing of a UserQuery message by a floor control server involves generating a UserStatus message, which **SHOULD** be generated as soon as possible.

When communicating over an unreliable transport and upon receiving a UserQuery from a participant, the floor control server **MUST** respond with a UserStatus message within the transaction failure window to complete the transaction.

The floor control server **MUST** copy the Conference ID, the Transaction ID, and the User ID from the UserQuery message into the UserStatus message, as described in Section 8.2.

The sender of the UserQuery message is requesting information about all the floor requests associated with a given participant (i.e., the floor requests where the participant is either the beneficiary or the requester). This participant is identified by a BENEFICIARY-ID attribute or, in the absence of a BENEFICIARY-ID attribute, by a the User ID in the COMMON-HEADER of the UserQuery message.

The floor control server **MUST** copy, if present, the contents of the BENEFICIARY-ID attribute from the UserQuery message into a BENEFICIARY-INFORMATION attribute in the UserStatus message. Additionally, the floor control server **MAY** provide the display name and the URI of the participant about which the UserStatus message provides information in this BENEFICIARY-INFORMATION attribute.

The floor control server **SHOULD** add to the UserStatus message a FLOOR-REQUEST-INFORMATION grouped attribute for each floor request related to the participant about which the message provides information (i.e., the floor requests where the participant is either the beneficiary or the requester). For each FLOOR-REQUEST-INFORMATION attribute, the floor control server follows the following steps.

The floor control server **MUST** identify the floor request the FLOOR-REQUEST-INFORMATION attribute applies to by filling the Floor Request ID field of the FLOOR-REQUEST-INFORMATION attribute.

The floor control server **MUST** add FLOOR-REQUEST-STATUS attributes to the FLOOR-REQUEST-INFORMATION grouped attribute identifying the floors being requested (i.e., the floors associated with the floor request identified by the FLOOR-REQUEST-ID attribute).

The floor control server **SHOULD** add a BENEFICIARY-ID attribute to the FLOOR-REQUEST-INFORMATION grouped attribute identifying the beneficiary of the floor request. Additionally, the floor control server **MAY** provide the display name and the URI of the beneficiary in this BENEFICIARY-INFORMATION attribute.

The floor control server **MAY** provide information about the requester of the floor in a REQUESTED-BY-INFORMATION attribute inside the FLOOR-REQUEST-INFORMATION grouped attribute.

The floor control server **MAY** provide the reason why the floor participant requested the floor in a PARTICIPANT-PROVIDED-INFO.

The floor control server **MAY** also add to the FLOOR-REQUEST-INFORMATION grouped attribute a PRIORITY attribute with the Priority value requested for the floor request.

The floor control server **MUST** include the current status of the floor request in an OVERALL-REQUEST-STATUS attribute to the FLOOR-REQUEST-INFORMATION grouped attribute. The floor control server **MAY** add a STATUS-INFO attribute with extra information about the floor request.

The floor control server **MAY** provide information about the status of the floor request as it relates to each of the floors being requested in the FLOOR-REQUEST-STATUS attributes.

13.4. Reception of a FloorRelease Message

On reception of a FloorRelease message, the floor control server follows the rules in Section 9 that relate to client authentication and authorization. If while processing the FloorRelease message, the floor control server encounters an error, it **MUST** generate an Error response following the procedures described in Section 13.8.

The successful processing of a FloorRelease message by a floor control server involves generating a FloorRequestStatus message, which **SHOULD** be generated as soon as possible.

When communicating over an unreliable transport and upon receiving a FloorRelease from a participant, the floor control server **MUST** respond with a FloorRequestStatus message within the transaction failure window to complete the transaction.

The floor control server **MUST** copy the Conference ID, the Transaction ID, and the User ID from the FloorRelease message into the FloorRequestStatus message, as described in Section 8.2.

The floor control server **MUST** add a FLOOR-REQUEST-INFORMATION grouped attribute to the FloorRequestStatus. The attributes contained in this grouped attribute carry information about the floor request.

The FloorRelease message identifies the floor request it applies to using a FLOOR-REQUEST-ID. The floor control server **MUST** copy the contents of the FLOOR-REQUEST-ID attribute from the FloorRelease message into the Floor Request ID field of the FLOOR-REQUEST-INFORMATION attribute.

The floor control server **MUST** identify the floors being released (i.e., the floors associated with the floor request identified by the FLOOR-REQUEST-ID attribute) in FLOOR-REQUEST-STATUS attributes to the FLOOR-REQUEST-INFORMATION grouped attribute.

The floor control server **MUST** add an OVERALL-REQUEST-STATUS attribute to the FLOOR-REQUEST-INFORMATION grouped attribute. The Request Status value **SHOULD** be Released, if the floor (or floors) had been previously granted, or Cancelled, if the floor (or floors) had not been previously granted. The floor control server **MAY** add a STATUS-INFO attribute with extra information about the floor request.

13.5. Reception of a FloorQuery Message

On reception of a FloorQuery message, the floor control server follows the rules in Section 9 that relate to client authentication. If while processing the FloorQuery message, the floor control server encounters an error, it **MUST** generate an Error response following the procedures described in Section 13.8.

When communicating over an unreliable transport and upon receiving a FloorQuery from a participant, the floor control server **MUST** respond with a FloorStatus message within the transaction failure window to complete the transaction.

A floor control server receiving a FloorQuery message from a client **SHOULD** keep this client informed about the status of the floors identified by FLOOR-ID attributes in the FloorQuery message. Floor control servers keep clients informed by using FloorStatus messages.

An individual FloorStatus message carries information about a single floor. So, when a FloorQuery message requests information about more than one floor, the floor control server needs to send separate FloorStatus messages for different floors.

The information FloorQuery messages carry may depend on the user requesting the information. For example, a chair may be able to receive information about pending requests, while a regular user may not be authorized to do so.

13.5.1. Generation of the First FloorStatus Message

The successful processing of a FloorQuery message by a floor control server involves generating one or several FloorStatus messages, the first of which **SHOULD** be generated as soon as possible.

The floor control server **MUST** copy the Conference ID, the Transaction ID, and the User ID from the FloorQuery message into the FloorStatus message, as described in Section 8.2.

If the FloorQuery message did not contain any FLOOR-ID attribute, the floor control server sends the FloorStatus message without adding any additional attribute and does not send any subsequent FloorStatus message to the floor participant.

If the FloorQuery message contained one or more FLOOR-ID attributes, the floor control server chooses one from among them and adds this FLOOR-ID attribute to the FloorStatus message. The floor control server **SHOULD** add a FLOOR-REQUEST-INFORMATION grouped attribute for each floor request associated to the floor. Each FLOOR-REQUEST-INFORMATION grouped attribute contains a number of attributes that provide information about the floor request. For each FLOOR-REQUEST-INFORMATION attribute, the floor control server follows the following steps.

The floor control server **MUST** identify the floor request the FLOOR-REQUEST-INFORMATION attribute applies to by filling the Floor Request ID field of the FLOOR-REQUEST-INFORMATION attribute.

The floor control server **MUST** add FLOOR-REQUEST-STATUS attributes to the FLOOR-REQUEST-INFORMATION grouped attribute identifying the floors being requested (i.e., the floors associated with the floor request identified by the FLOOR-REQUEST-ID attribute).

The floor control server **SHOULD** add a BENEFICIARY-ID attribute to the FLOOR-REQUEST-INFORMATION grouped attribute identifying the beneficiary of the floor request. Additionally, the floor control server **MAY** provide the display name and the URI of the beneficiary in this BENEFICIARY-INFORMATION attribute.

The floor control server **MAY** provide information about the requester of the floor in a REQUESTED-BY-INFORMATION attribute inside the FLOOR-REQUEST-INFORMATION grouped attribute.

The floor control server **MAY** provide the reason why the floor participant requested the floor in a PARTICIPANT-PROVIDED-INFO.

The floor control server **MAY** also add to the FLOOR-REQUEST-INFORMATION grouped attribute a PRIORITY attribute with the Priority value requested for the floor request.

The floor control server **MUST** add an OVERALL-REQUEST-STATUS attribute to the FLOOR-REQUEST-INFORMATION grouped attribute with the current status of the floor request. The floor control server **MAY** add a STATUS-INFO attribute with extra information about the floor request.

The floor control server **MAY** provide information about the status of the floor request as it relates to each of the floors being requested in the FLOOR-REQUEST-STATUS attributes.

13.5.2. Generation of Subsequent FloorStatus Messages

If the FloorQuery message carried more than one FLOOR-ID attribute, the floor control server **SHOULD** generate a FloorStatus message for each of them (except for the FLOOR-ID attribute chosen for the first FloorStatus message) as soon as possible. These FloorStatus messages are generated following the same rules as those for the first FloorStatus message (see Section 13.5.1), but their Transaction ID is 0 when using a reliable transport and non-zero and unique in the context of outstanding transactions when using an unreliable transport (cf. Section 8).

After generating these messages, the floor control server sends FloorStatus messages, periodically keeping the client informed about all the floors for which the client requested information. The Transaction ID of these messages **MUST** be 0 when using a reliable transport and non-zero and unique in the context of outstanding transactions when using an unreliable transport (cf. Section 8).

> The rate at which the floor control server sends FloorStatus messages is a matter of local policy. A floor control server may choose to send a new FloorStatus message every time a new floor request arrives, while another may choose to only send a new FloorStatus message when a new floor request is Granted.

When communicating over an unreliable transport and a FloorStatusAck message is not received within the transaction failure window, the floor control server **MUST** retransmit the FloorStatus message according to Section 6.2.

13.6. Reception of a ChairAction Message

On reception of a ChairAction message, the floor control server follows the rules in Section 9 that relate to client authentication and authorization. If while processing the ChairAction message, the floor control server encounters an error, it **MUST** generate an Error response following the procedures described in Section 13.8.

The successful processing of a ChairAction message by a floor control server involves generating a ChairActionAck message, which **SHOULD** be generated as soon as possible.

When communicating over an unreliable transport and upon receiving a ChairAction from a chair, the floor control server **MUST** respond with a ChairActionAck message within the transaction failure window to complete the transaction.

The floor control server **MUST** copy the Conference ID, the Transaction ID, and the User ID from the ChairAction message into the ChairActionAck message, as described in Section 8.2.

The floor control server needs to take into consideration the operation requested in the ChairAction message (e.g., granting a floor) but does not necessarily need to perform it as requested by the floor chair. The operation that the floor control server performs depends on the ChairAction message and on the internal state of the floor control server.

For example, a floor chair may send a ChairAction message granting a floor that was requested as part of an atomic floor request operation that involved several floors. Even if the chair responsible for one of the floors instructs the floor control server to grant the floor, the floor control server will not grant it until the chairs responsible for the other floors agree to grant them as well.

So, the floor control server is ultimately responsible for keeping a coherent floor state using instructions from floor chairs as input to this state.

If the new Status in the ChairAction message is Accepted and all the bits of the Queue Position field are zero, the floor chair is requesting that the floor control server assign a queue position (e.g., the last in the queue) to the floor request based on the local policy of the floor control server. (Of course, such a request only applies if the floor control server implements a queue.)

13.7. Reception of a Hello Message

On reception of a Hello message, the floor control server follows the rules in Section 9 that relate to client authentication. If while processing the Hello message, the floor control server encounters an error, it **MUST** generate an Error response following the procedures described in Section 13.8.

If the version of BFCP specified in the version field of the COMMON-HEADER is supported by the floor control server, it **MUST** respond with the same version number in the HelloAck; this defines the version for all subsequent BFCP messages within this BFCP Connection.

When communicating over an unreliable transport and upon receiving a Hello from a participant, the floor control server **MUST** respond with a HelloAck message within the transaction failure window to complete the transaction.

The successful processing of a Hello message by a floor control server involves generating a HelloAck message, which **SHOULD** be generated as soon as possible. The floor control server **MUST** copy the Conference ID, the Transaction ID, and the User ID from the Hello into the HelloAck, as described in Section 8.2.

The floor control server **MUST** add a SUPPORTED-PRIMITIVES attribute to the HelloAck message listing all the primitives (i.e., BFCP messages) supported by the floor control server.

The floor control server **MUST** add a SUPPORTED-ATTRIBUTES attribute to the HelloAck message listing all the attributes supported by the floor control server.

13.8. Error Message Generation

Error messages are always sent in response to a previous message from the client as part of a client-initiated transaction. The ABNF in Section 5.3.13 describes the attributes that an Error message can contain. In addition, the ABNF specifies normatively which of these attributes are mandatory and which ones are optional.

The floor control server **MUST** copy the Conference ID, the Transaction ID, and the User ID from the message from the client into the Error message, as described in Section 8.2.

The floor control server **MUST** add an ERROR-CODE attribute to the Error message. The ERROR-CODE attribute contains an error code from Table 5. Additionally, the floor control server may add an ERROR-INFO attribute with extra information about the error.

14. Security Considerations

BFCP uses TLS/DTLS to provide mutual authentication between clients and servers. TLS/DTLS also provides replay and integrity protection and confidentiality. It is **RECOMMENDED** that TLS/DTLS with an encryption algorithm according to Section 7 always be used. In cases where signaling/control traffic is properly protected, as described in Section 9, it is **REQUIRED** to use a mandated encryption algorithm. BFCP entities **MAY** use other security mechanisms to interwork with legacy implementation that do not use TLS/DTLS as long as these mechanisms provide similar security properties. An example of other mechanisms to effectively secure a nonsecure BFCP connection is IPsec [21].

The remainder of this section analyzes some of the threats against BFCP and how they are addressed.

An attacker may attempt to impersonate a client (a floor participant or a floor chair) in order to generate forged floor requests or to grant or deny existing floor requests. Client impersonation is avoided by having servers only accept BFCP messages over authenticated TLS/DTLS connections. The floor control server assumes that attackers cannot hijack the TLS/DTLS connection and, therefore, that messages over the TLS/DTLS connection come from the client that was initially authenticated.

An attacker may attempt to impersonate a floor control server. A successful attacker would be able to make clients think that they hold a particular floor so that they would try to access a resource (e.g., sending media) without having legitimate rights to access it. Floor control server impersonation is avoided by having servers only accept BFCP messages over authenticated TLS/DTLS connections, as well as ensuring clients only send and accept messages over authenticated TLS/DTLS connections.

Attackers may attempt to modify messages exchanged by a client and a floor control server. The integrity protection provided by TLS/DTLS connections prevents this attack.

An attacker may attempt to fetch a valid message sent by a client to a floor control server and replay it over a connection between the attacker and the floor control server. This attack is prevented by having floor control servers check that messages arriving over a given authenticated TLS/DTLS connection use an authorized user ID (i.e., a user ID that the user that established the authenticated TLS/DTLS connection is allowed to use).

Attackers may attempt to pick messages from the network to get access to confidential information between the floor control server and a client (e.g., why a floor request was denied). TLS/DTLS confidentiality prevents this attack. Therefore, it is **REQUIRED** that TLS/DTLS be used with an encryption algorithm according to Section 7.

15. IANA Considerations

The IANA has created a registry for BFCP parameters called "The Binary Floor Control Protocol (BFCP) Parameters". This registry has a number of subregistries, which are described in the following sections.

15.1. Attributes Subregistry

This section establishes the "Attributes" subregistry under the BFCP Parameters registry. As per the terminology in RFC 8126 [6], the registration policy for BFCP attributes is "Specification Required". For the purposes of this subregistry, the BFCP attributes for which IANA registration is requested **MUST** be defined by a Standards Track RFC. Such an RFC **MUST** specify the attribute's type, name, format, and semantics.

For each BFCP attribute, the IANA registers its type, its name, and the reference to the RFC where the attribute is defined. The following table contains the initial values of this subregistry.

Type	Attribute	Reference
1	BENEFICIARY-ID	RFC 8855
2	FLOOR-ID	RFC 8855
3	FLOOR-REQUEST-ID	RFC 8855
4	PRIORITY	RFC 8855
5	REQUEST-STATUS	RFC 8855
6	ERROR-CODE	RFC 8855
7	ERROR-INFO	RFC 8855
8	PARTICIPANT-PROVIDED-INFO	RFC 8855
9	STATUS-INFO	RFC 8855
10	SUPPORTED-ATTRIBUTES	RFC 8855
11	SUPPORTED-PRIMITIVES	RFC 8855
12	USER-DISPLAY-NAME	RFC 8855

Type	Attribute	Reference
13	USER-URI	RFC 8855
14	BENEFICIARY-INFORMATION	RFC 8855
15	FLOOR-REQUEST-INFORMATION	RFC 8855
16	REQUESTED-BY-INFORMATION	RFC 8855
17	FLOOR-REQUEST-STATUS	RFC 8855
18	OVERALL-REQUEST-STATUS	RFC 8855

Table 7: Initial values of the BFCP Attributes subregistry

15.2. Primitives Subregistry

This section establishes the "Primitives" subregistry under the BFCP Parameters registry. As per the terminology in RFC 8126 [6], the registration policy for BFCP primitives is "Specification Required". For the purposes of this subregistry, the BFCP primitives for which IANA registration is requested **MUST** be defined by a Standards Track RFC. Such an RFC **MUST** specify the primitive's value, name, format, and semantics.

For each BFCP primitive, the IANA registers its value, its name, and the reference to the RFC where the primitive is defined. The following table contains the initial values of this subregistry.

Value	Primitive	Reference
1	FloorRequest	RFC 8855
2	FloorRelease	RFC 8855
3	FloorRequestQuery	RFC 8855
4	FloorRequestStatus	RFC 8855
5	UserQuery	RFC 8855
6	UserStatus	RFC 8855
7	FloorQuery	RFC 8855
8	FloorStatus	RFC 8855
9	ChairAction	RFC 8855
10	ChairActionAck	RFC 8855

Value	Primitive	Reference
11	Hello	RFC 8855
12	HelloAck	RFC 8855
13	Error	RFC 8855
14	FloorRequestStatusAck	RFC 8855
15	FloorStatusAck	RFC 8855
16	Goodbye	RFC 8855
17	GoodbyeAck	RFC 8855

Table 8: Initial values of the BFCP Primitives subregistry

15.3. Request Statuses Subregistry

This section establishes the "Request Statuses" subregistry under the BFCP Parameters registry. As per the terminology in RFC 8126 [6], the registration policy for BFCP request statuses is "Specification Required". For the purposes of this subregistry, the BFCP request statuses for which IANA registration is requested **MUST** be defined by a Standards Track RFC. Such an RFC **MUST** specify the value and the semantics of the request status.

For each BFCP request status, the IANA registers its value, its meaning, and the reference to the RFC where the request status is defined. The following table contains the initial values of this subregistry.

Value	Status	Reference
1	Pending	RFC 8855
2	Accepted	RFC 8855
3	Granted	RFC 8855
4	Denied	RFC 8855
5	Cancelled	RFC 8855
6	Released	RFC 8855
7	Revoked	RFC 8855

Table 9: Initial values of the Request Statuses subregistry

15.4. Error Codes Subregistry

This section establishes the "Error Codes" subregistry under the BFCP Parameters registry. As per the terminology in RFC 8126 [6], the registration policy for BFCP error codes is "Specification Required". For the purposes of this subregistry, the BFCP error codes for which IANA registration is requested **MUST** be defined by a Standards Track RFC. Such an RFC **MUST** specify the value and the semantics of the error code, and any Error Specific Details that apply to it.

For each BFCP primitive, the IANA registers its value, its meaning, and the reference to the RFC where the primitive is defined. The following table contains the initial values of this subregistry.

Value	Meaning	Reference
1	Conference Does Not Exist	RFC 8855
2	User Does Not Exist	RFC 8855
3	Unknown Primitive	RFC 8855
4	Unknown Mandatory Attribute	RFC 8855
5	Unauthorized Operation	RFC 8855
6	Invalid Floor ID	RFC 8855
7	Floor Request ID Does Not Exist	RFC 8855
8	You have Already Reached the Maximum Number of Ongoing Floor Requests for This Floor	RFC 8855
9	Use TLS	RFC 8855
10	Unable to Parse Message	RFC 8855
11	Use DTLS	RFC 8855
12	Unsupported Version	RFC 8855
13	Incorrect Message Length	RFC 8855
14	Generic Error	RFC 8855

Table 10: Initial values of the Error Codes subregistry

16. Changes from RFC 4582

The following is the list of technical changes and other non-trivial fixes from [3].

16.1. Extensions for an Unreliable Transport

The main purpose of this work was to revise the specification to support BFCP over an unreliable transport, resulting in the following changes:

1. Overview of Operation (Section 4):

   Changed the description of client-initiated and server-initiated transactions, referring to Section 8.

2. COMMON-HEADER Format (Section 5.1):

   Ver(sion) field, where the value 2 is used for the extensions for an unreliable transport. Added new R and F flag bits for an unreliable transport. Res(erved) field is now 3 bit. New optional Fragment Offset and Fragment Length fields.

3. New primitives (Section 5.1):

   Added four new primitives: FloorRequestStatusAck, FloorStatusAck, Goodbye, and GoodbyeAck.

4. New error codes (Section 5.2.6):

   Added three new error codes: "Unable to Parse Message", "Use DTLS" and "Unsupported Version". Note that two additional error codes were added, see Section 16.2.

5. ABNF for new primitives (Section 5.3):

   Added new subsections with normative ABNF for the new primitives.

6. Transport split in two (Section 6):

   Section 6 specifying the transport was split in two subsections; Section 6.1 for a reliable transport and Section 6.2 for an unreliable transport. The specification for an unreliable transport, among other issues, deals with reliability, congestion control, fragmentation and ICMP.

7. Mandated DTLS (Section 7 and Section 9):

   Mandated DTLS support when transport over UDP is used.

8. Transaction changes (Section 8):

   Server-initiated transactions over an unreliable transport have non-zero and unique Transaction IDs. Over an unreliable transport, the retransmit timers T1 and T2 described in Section 8.3 apply.

9. Timely response required (Section 8.3, Section 10.1.2, Section 10.2.2, Section 11.2, Section 12.1.2, Section 12.2.2, Section 12.3.2, Section 12.4.2, Section 10.1.3 and Section 12.1.3):

   Described that a given response must be sent within the transaction failure window to complete the transaction.

10. Updated IANA Considerations (Section 15):

    Added the new primitives and error codes to Section 15.2 and Section 15.4 respectively.

11. Examples over an unreliable transport (Appendix A):

    Added sample interactions over an unreliable transport for the scenarios in Figure 2 and Figure 3

12. Motivation for an unreliable transport (Appendix B):

    Added introduction to and motivation for extending BFCP to support an unreliable transport.

16.2. Other Changes

Clarifications and bug fixes:

1. ABNF fixes (Figure 22, Figure 24, Figure 26, Figure 28, Figure 30, and the ABNF figures in Section 5.3):

   Although formally correct in [3], the notation has changed in a number of figures to an equivalent form for clarity, e.g., `s/*1(FLOOR-ID)/[FLOOR-ID]/` in Figure 38 and `s/*[XXX]/*(XXX)/` in the other figures.
2. Typo (Section 12.4.2):

   Changed from SUPPORTED-PRIMITVIES to SUPPORTED-PRIMITIVES in the second paragraph.
3. Corrected attribute type (Section 13.1.1):

   Changed from PARTICIPANT-PROVIDED-INFO to PRIORITY attribute in the eighth paragraph, since the note below describes priority and that the last paragraph deals with PARTICIPANT-PROVIDED-INFO.
4. New error codes (Section 5.2.6):

   Added two additional error codes: "Incorrect Message Length" and "Generic Error".
5. New cipher suites (Section 7)

   Additional cipher suites are now specified which should be supported.
6. Assorted clarifications (Across the document):

   Language clarifications as a result of reviews. Also, the normative language was tightened where appropriate, i.e. changed from **SHOULD** strength to **MUST** in a number of places.

17. References

17.1. Normative References

[1] 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>.

[2] 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>.

[3] Camarillo, G., Ott, J., and K. Drage, "The Binary Floor Control Protocol (BFCP)", RFC 4582, DOI 10.17487/RFC4582, November 2006, <https://www.rfc-editor.org/info/rfc4582>.

[4] Camarillo, G., "Connection Establishment in the Binary Floor Control Protocol (BFCP)", RFC 5018, DOI 10.17487/RFC5018, September 2007, <https://www.rfc-editor.org/info/rfc5018>.

[5] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[6] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[7] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[8] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[9] 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>.

[10] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[11] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[12] Camarillo, G., Kristensen, T., and C. Holmberg, "Session Description Protocol (SDP) Format for Binary Floor Control Protocol (BFCP) Streams", RFC 8856, DOI 10.17487/RFC8856, January 2021, <https://www.rfc-editor.org/info/rfc8856>.

[13] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, DOI 10.17487/RFC4961, July 2007, <https://www.rfc-editor.org/info/rfc4961>.

[14] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <https://www.rfc-editor.org/info/rfc5389>.

[15] 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>.

[16] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/info/rfc8445>.

17.2. Informative References

[17] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <https://www.rfc-editor.org/info/rfc3264>.

[18] Koskelainen, P., Ott, J., Schulzrinne, H., and X. Wu, "Requirements for Floor Control Protocols", RFC 4376, DOI 10.17487/RFC4376, February 2006, <https://www.rfc-editor.org/info/rfc4376>.

[19] Barnes, M., Boulton, C., and O. Levin, "A Framework for Centralized Conferencing", RFC 5239, DOI 10.17487/RFC5239, June 2008, <https://www.rfc-editor.org/info/rfc5239>.

[20] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[21] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[22] Novo, O., Camarillo, G., Morgan, D., and J. Urpalainen, "Conference Information Data Model for Centralized Conferencing (XCON)", RFC 6501, DOI 10.17487/RFC6501, March 2012, <https://www.rfc-editor.org/info/rfc6501>.

[23] Barnes, M., Boulton, C., Romano, S., and H. Schulzrinne, "Centralized Conferencing Manipulation Protocol", RFC 6503, DOI 10.17487/RFC6503, March 2012, <https://www.rfc-editor.org/info/rfc6503>.

[24] Barnes, M., Miniero, L., Presta, R., and S P. Romano, "Centralized Conferencing Manipulation Protocol (CCMP) Call Flow Examples", RFC 6504, DOI 10.17487/RFC6504, March 2012, <https://www.rfc-editor.org/info/rfc6504>.

[25] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

[26] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <https://www.rfc-editor.org/info/rfc8201>.

[27] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <https://www.rfc-editor.org/info/rfc4821>.

[28] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <https://www.rfc-editor.org/info/rfc5763>.

[29] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <https://www.rfc-editor.org/info/rfc6951>.

[30] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[31] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, DOI 10.17487/RFC4380, February 2006, <https://www.rfc-editor.org/info/rfc4380>.

[32] Thaler, D., "Teredo Extensions", RFC 6081, DOI 10.17487/RFC6081, January 2011, <https://www.rfc-editor.org/info/rfc6081>.

[33] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[34] Rosenberg, J., Keranen, A., Lowekamp, B. B., and A. B. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, DOI 10.17487/RFC6544, March 2012, <https://www.rfc-editor.org/info/rfc6544>.

[35] Manner, J., Varis, N., and B. Briscoe, "Generic UDP Tunnelling (GUT)", Work in Progress, Internet-Draft, draft-manner-tsvwg-gut-02, 12 July 2010, <https://datatracker.ietf.org/doc/html/draft-manner-tsvwg-gut-02>.

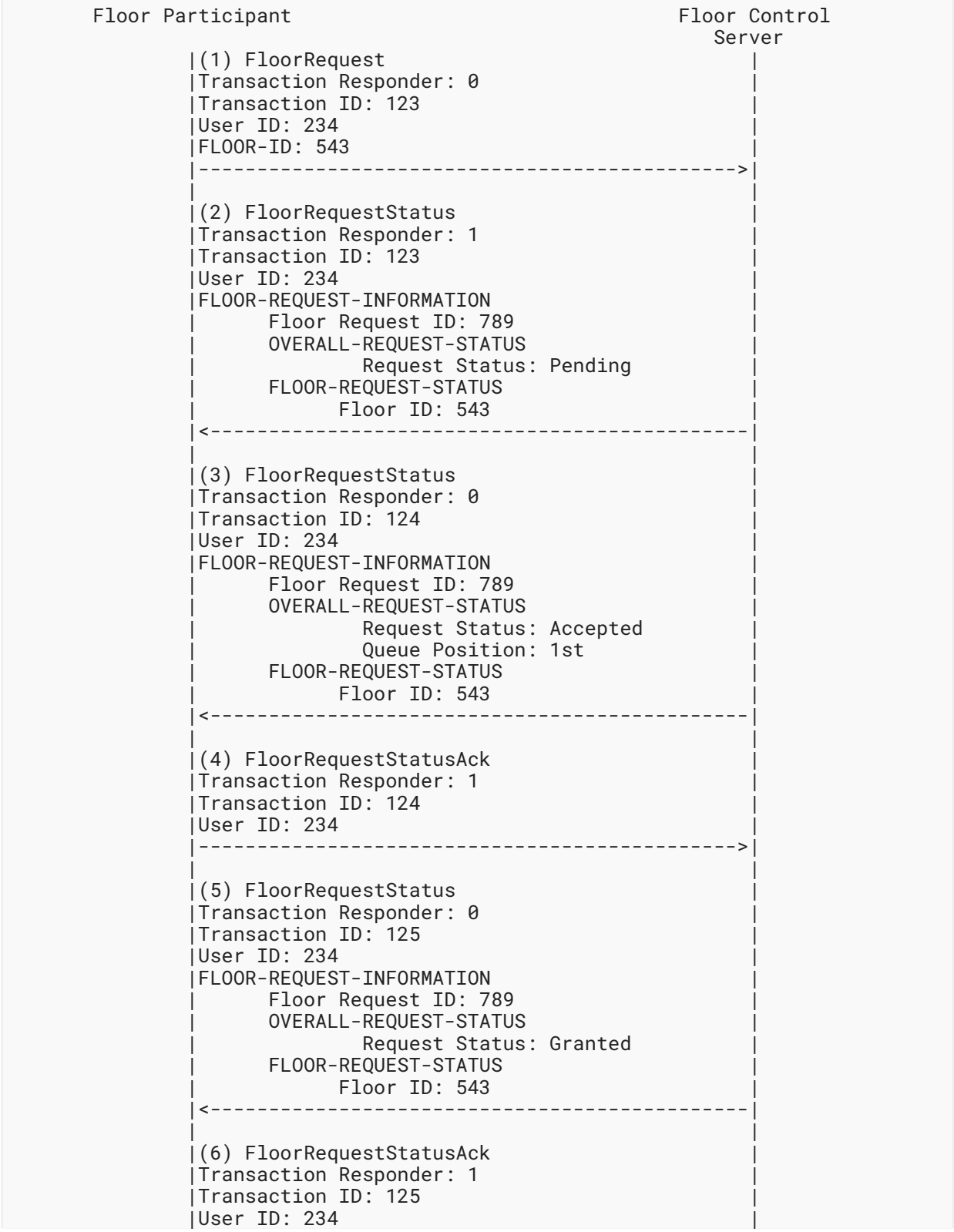
[36] Stucker, B., Tschofenig, H., and G. Salgueiro, "Analysis of Middlebox Interactions for Signaling Protocol Communication along the Media Path", Work in Progress, Internet-Draft, draft-ietf-mmusic-media-path-middleboxes-07, 30 May 2013, <https://datatracker.ietf.org/doc/html/draft-ietf-mmusic-media-path-middleboxes-07>.

[37] Guha, S. and P. Francis, "Characterization and Measurement of TCP Traversal through NATs and Firewalls", 2005, <https://www.usenix.org/legacy/event/imc05/tech/full_papers/guha/guha.pdf>.

[38] Ford, B., Srisuresh, P., and D. Kegel, "Peer-to-Peer Communication Across Network Address Translators", April 2005, <https://www.usenix.org/legacy/events/usenix05/tech/general/full_papers/ford/ford.pdf>.

Appendix A. Example Call Flows for BFCP over an Unreliable Transport

With reference to Section 4.1, the following figures show representative call flows for requesting and releasing a floor, and obtaining status information about a floor when BFCP is deployed over an unreliable transport. The figures here show a lossless interaction.


Floor Participant
Floor Control Server
(1) FloorRequest
Transaction Responder: 0
Transaction ID: 123
User ID: 234
FLOOR-ID: 543
(2) FloorRequestStatus
Transaction Responder: 1
Transaction ID: 123
User ID: 234
FLOOR-REQUEST-INFORMATION
Floor Request ID: 789
OVERALL-REQUEST-STATUS
Request Status: Pending
FLOOR-REQUEST-STATUS
Floor ID: 543
(3) FloorRequestStatus
Transaction Responder: 0
Transaction ID: 124
User ID: 234
FLOOR-REQUEST-INFORMATION
Floor Request ID: 789
OVERALL-REQUEST-STATUS
Request Status: Accepted
Queue Position: 1st
FLOOR-REQUEST-STATUS
Floor ID: 543
(4) FloorRequestStatusAck
Transaction Responder: 1
Transaction ID: 124
User ID: 234
(5) FloorRequestStatus
Transaction Responder: 0
Transaction ID: 125
User ID: 234
FLOOR-REQUEST-INFORMATION
Floor Request ID: 789
OVERALL-REQUEST-STATUS
Request Status: Granted
FLOOR-REQUEST-STATUS
Floor ID: 543
(6) FloorRequestStatusAck
Transaction Responder: 1
Transaction ID: 125
User ID: 234

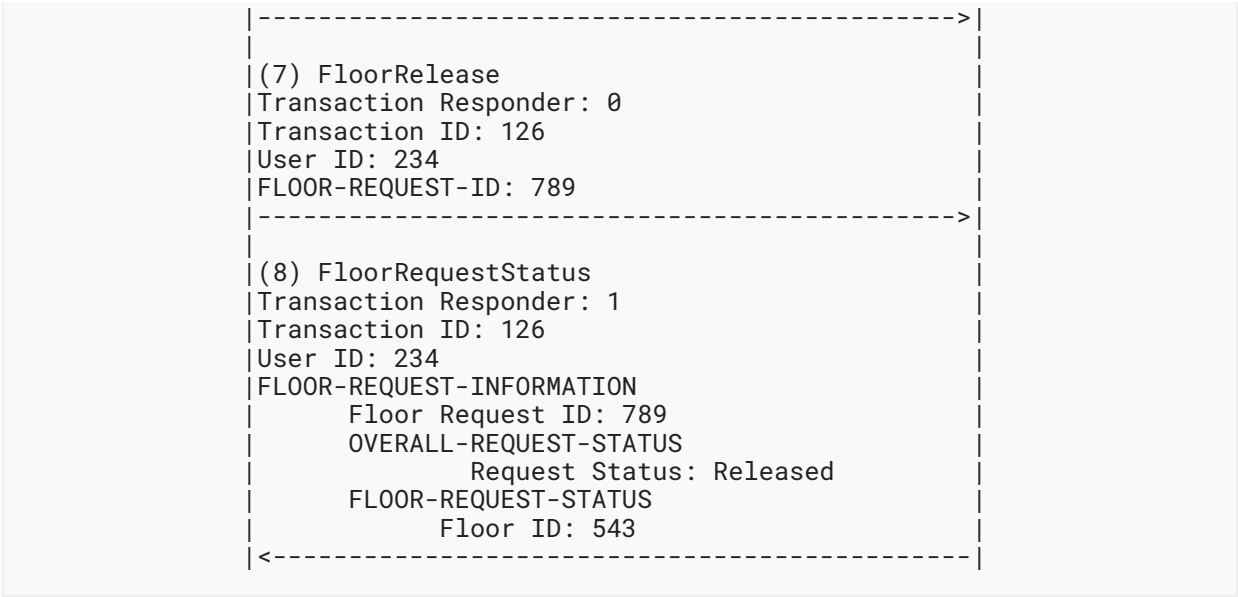
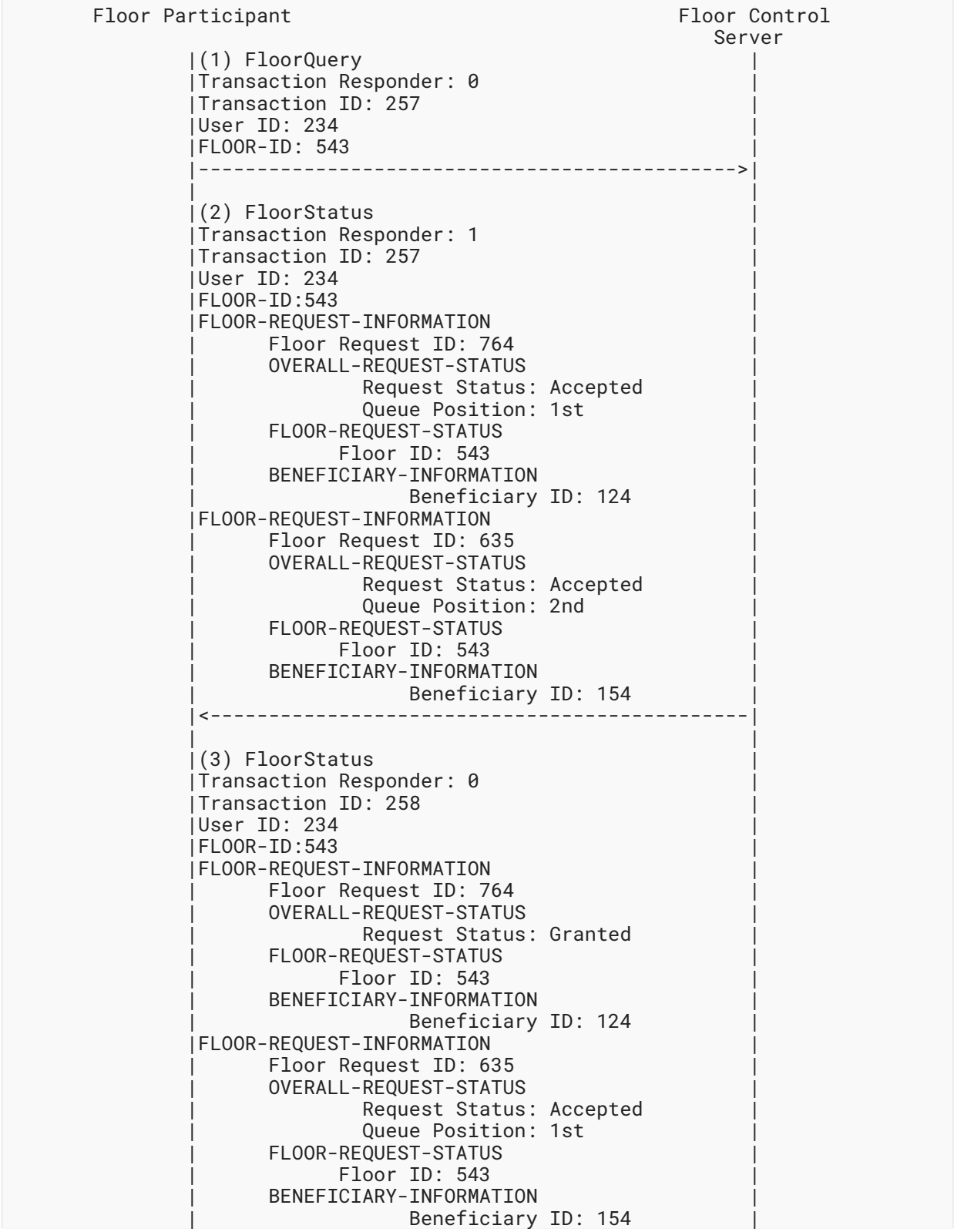


Figure 48: Requesting and releasing a floor

Note that in Figure 48, the FloorRequestStatus message from the floor control server to the floor participant is a transaction-closing message as a response to the client-initiated transaction with Transaction ID 126. As such, it is not followed by a FloorRequestStatusAck message from the floor participant to the floor control server.


Floor Participant
Floor Control Server
(1) FloorQuery
Transaction Responder: 0
Transaction ID: 257
User ID: 234
FLOOR-ID: 543
(2) FloorStatus
Transaction Responder: 1
Transaction ID: 257
User ID: 234
FLOOR-ID:543
FLOOR-REQUEST-INFORMATION
Floor Request ID: 764
OVERALL-REQUEST-STATUS
Request Status: Accepted
Queue Position: 1st
FLOOR-REQUEST-STATUS
Floor ID: 543
BENEFICIARY-INFORMATION
Beneficiary ID: 124
FLOOR-REQUEST-INFORMATION
Floor Request ID: 635
OVERALL-REQUEST-STATUS
Request Status: Accepted
Queue Position: 2nd
FLOOR-REQUEST-STATUS
Floor ID: 543
BENEFICIARY-INFORMATION
Beneficiary ID: 154
(3) FloorStatus
Transaction Responder: 0
Transaction ID: 258
User ID: 234
FLOOR-ID:543
FLOOR-REQUEST-INFORMATION
Floor Request ID: 764
OVERALL-REQUEST-STATUS
Request Status: Granted
FLOOR-REQUEST-STATUS
Floor ID: 543
BENEFICIARY-INFORMATION
Beneficiary ID: 124
FLOOR-REQUEST-INFORMATION
Floor Request ID: 635
OVERALL-REQUEST-STATUS
Request Status: Accepted
Queue Position: 1st
FLOOR-REQUEST-STATUS
Floor ID: 543
BENEFICIARY-INFORMATION
Beneficiary ID: 154

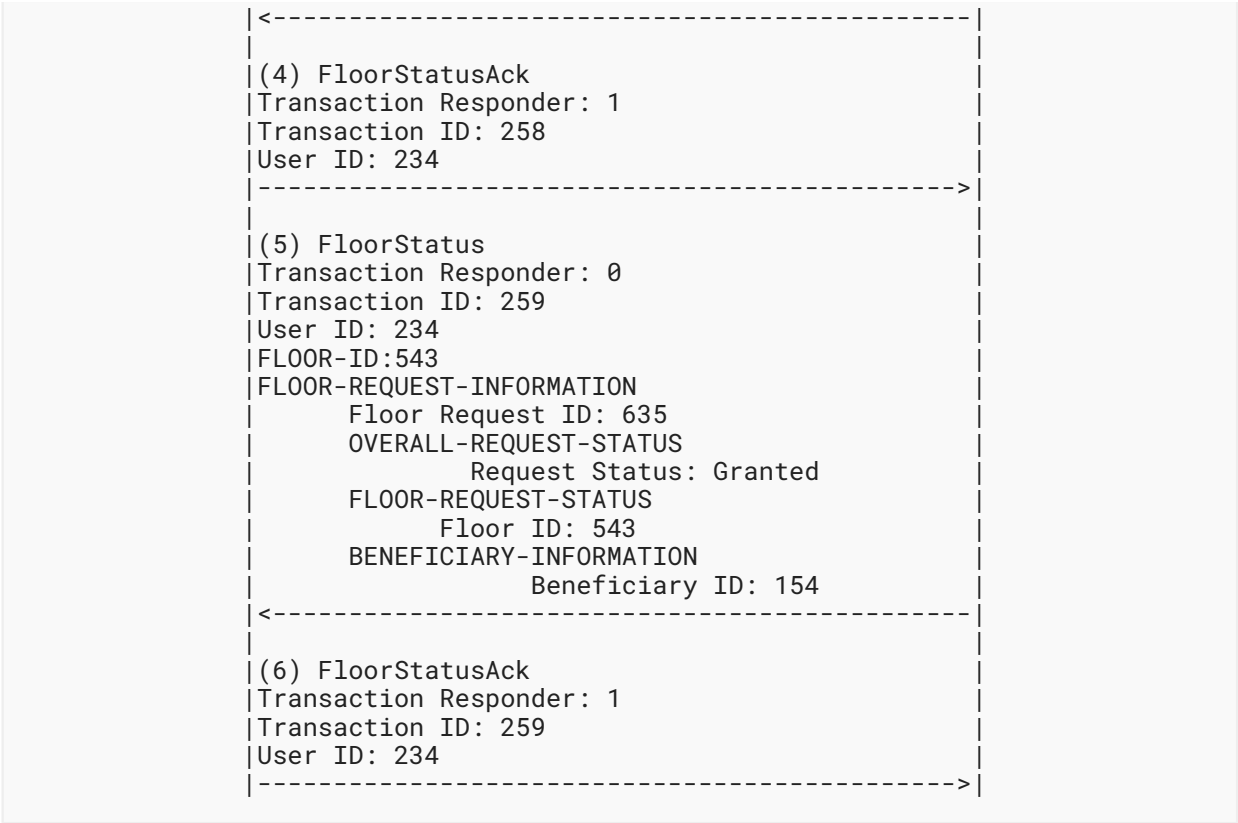


Figure 49: Obtaining status information about a floor

Appendix B. Motivation for Supporting an Unreliable Transport

This appendix is provided as an aid to understand the background and rationale for adding support for unreliable transport.

B.1. Motivation

In existing video conferencing deployments, BFCP is used to manage the floor for the content sharing associated with the conference. For peer-to-peer scenarios, including business-to-business conferences and point-to-point conferences in general, it is frequently the case that one or both endpoints exist behind a NAT. BFCP roles are negotiated in the offer/answer exchange as specified in [12], resulting in one endpoint being responsible for opening the TCP connection used for the BFCP communication.

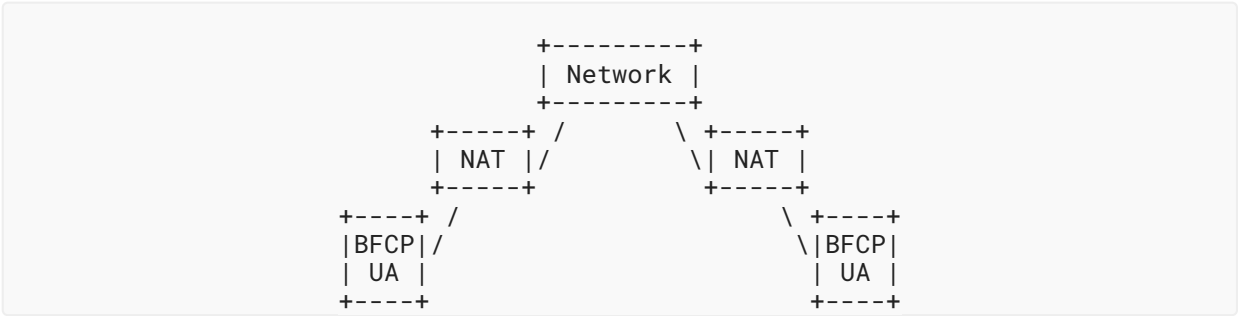


Figure 50: Use case

The communication session between the video conferencing endpoints typically consists of a number of RTP over UDP media streams for audio and video and a BFCP connection for floor control. Existing deployments are most common in, but not limited to, enterprise networks. In existing deployments, NAT traversal for the RTP streams works using ICE and/or other methods, including those described in [36].

When enhancing an existing SIP-based video conferencing deployment with support for content sharing, the BFCP connection often poses a problem. The reasons for this fall into two general classes. First, there may be a strong preference for UDP-based signaling in general. On high-capacity endpoints (e.g., Public Switched Telephone Network (PSTN) gateways or SIP/H.323 inter-working gateways), TCP can suffer from head-of-line blocking, and it uses many kernel buffers. Network operators view UDP as a way to avoid both of these. Second, the establishment and traversal of the TCP connection involving ephemeral ports, as is typically the case with BFCP over TCP, can be problematic, as described in Appendix A of [34]. A broad study of NAT behavior and peer-to-peer TCP establishment for a comprehensive set of TCP NAT traversal techniques over a wide range of commercial NAT products concluded that it was not possible to establish a TCP connection in 11% of the cases [37]. The results are worse when focusing on enterprise NATs. A study of hole-punching as a NAT traversal technique across a wide variety of deployed NATs reported consistently higher success rates when using UDP than when using TCP [38].

It is worth noting that BFCP over UDP is already being used in real deployments, underlining the necessity to specify a common way to exchange BFCP messages where TCP is not appropriate, to avoid a situation where multiple different and non-interoperable implementations would coexist in the market. The purpose of this document is to extend the standard specification to support unreliable transport in order to facilitate complete interoperability between implementations.

B.1.1. Alternatives Considered

In selecting the approach of defining UDP as an alternate transport for BFCP, several alternatives were considered and explored to some degree. Each of these is discussed briefly in the following subsections. In summary, while the alternatives that were not chosen work in a number of scenarios, they are not sufficient, in and of themselves, to address the use case targeted by this document. The last alternative, presented in Appendix B.1.1.7, was selected and is specified in this document.

It is also worth noting that the IETF Transport Area was asked for a way to tunnel TCP over UDP, but at that point there was no consensus on how to achieve that.

B.1.1.1. ICE TCP

ICE TCP [34] extends ICE to TCP-based media, including the ability to offer a mix of TCP- and UDP-based candidates for a single stream. ICE TCP has, in general, a lower success probability for enabling TCP connectivity without a relay if both of the hosts are behind a NAT (see Appendix A of [34]) than enabling UDP connectivity in the same scenarios. The happens because many of the currently deployed NATs in video conferencing networks do not support the flow of TCP handshake packets seen in the case of TCP simultaneous-open, either because they do not allow incoming TCP SYN packets from an address to which a SYN packet has been sent recently, or because they do not properly process the subsequent SYNACK. Implementing various techniques advocated for candidate collection in [34] should increase the success probability, but many of these techniques require support from some network elements (e.g., from the NATs). Such support is not common in enterprise NATs.

B.1.1.2. Teredo

Teredo [31] enables nodes located behind one or more IPv4 NATs to obtain IPv6 connectivity by tunneling packets over UDP. Teredo extensions [32] provide additional capabilities to Teredo, including support for more types of NATs and support for more efficient communication.

As defined, Teredo could be used to make BFCP work for the video conferencing use cases addressed in this document. However, running the service requires the help of "Teredo servers" and "Teredo relays" [31]. These servers and relays generally do not exist in current video conferencing deployments. It also requires IPv6 awareness on the endpoints. It should also be noted that ICMP6, as used with Teredo to complete an initial protocol exchange and confirm that the appropriate NAT bindings have been set up, is not a conventional feature of IPv4 or even IPv6, and some currently deployed IPv6 firewalls discard ICMP messages. As these networks continue to evolve and tackle the transaction to IPv6, Teredo servers and relays may be deployed, making Teredo available as a suitable alternative to BFCP over UDP.

B.1.1.3. GUT

GUT [35] attempts to facilitate tunneling over UDP by encapsulating the native transport protocol and its payload (in general the whole IP payload) within a UDP packet destined to the well-known port GUT_P. Unfortunately, it requires user-space TCP, for which there is not a readily available implementation, and creating one is a large project in itself. This document has expired, and its future is still unclear as it has not yet been adopted by a working group.

B.1.1.4. UPnP IGD

Universal Plug and Play Internet Gateway Devices (UPnP IGD) sit on the edge of the network, providing connectivity to the Internet for computers internal to the LAN, but do not allow Internet devices to connect to computers on the internal LAN. IGDs enable a computer on an internal LAN to create port mappings on their NAT, through which hosts on the Internet can send data that will be forwarded to the computer on the internal LAN. IGDs may be self-contained hardware devices or may be software components provided within an operating system.

In considering UPnP IGD, several issues exist. Not all NATs support UPnP, and many that do support it are configured with it turned off by default. NATs are often multilayered, and UPnP does not work well with such NATs. For example, a typical DSL modem acts as a NAT, and the user plugs in a wireless access point behind that, which adds another layer of NAT. The client can discover the first layer of NAT using multicast, but it is harder to figure out how to discover and control NATs in the next layer up.

B.1.1.5. NAT PMP

The NAT Port Mapping Protocol (NAT PMP) allows a computer in a private network (behind a NAT router) to automatically configure the router to allow parties outside the private network to contact it. NAT PMP runs over UDP. It essentially automates the process of port forwarding. Included in the protocol is a method for retrieving the public IP address of a NAT gateway, thus allowing a client to make this public IP address and port number known to peers that may wish to communicate with it.

Many NATs do not support PMP. In those that do support it, it has similar issues with negotiation of multilayer NATs as UPnP. Video conferencing is used extensively in enterprise networks, and NAT PMP is not generally available in enterprise-class routers.

B.1.1.6. SCTP

It would be quite straightforward to specify a BFCP binding for Stream Control Transmission Protocol (SCTP) [33], and then tunnel SCTP over UDP in the use case described in Appendix B.1. SCTP is gaining some momentum currently. There was ongoing discussion in the RTCWeb Working Group regarding this approach, which resulted in [29]. However, this approach to tunneling over UDP was not mature enough when considered and was not even fully specified.

B.1.1.7. BFCP over UDP Transport

To overcome the problems with establishing TCP flows between BFCP entities, an alternative is to define UDP as an alternate transport for BFCP, leveraging the same mechanisms in place for the RTP over UDP media streams for the BFCP communication. When using UDP as the transport, following the guidelines provided in [15] is recommended.

Minor changes to the transaction model have been introduced in that all requests now have an appropriate response to complete the transaction. The requests are sent with a retransmission timer associated with the response to achieve reliability. This alternative does not change the semantics of BFCP. It permits UDP as an alternate transport.

Existing implementations, in the spirit of the approach detailed in earlier draft versions of this document, have demonstrated that this approach is feasible. Initial compatibility among implementations has been achieved at previous interoperability events. The authors view this extension as a pragmatic solution to an existing deployment challenge. This is the chosen approach, and the extensions are specified in this document.

Acknowledgements

The XCON Working Group chairs, Adam Roach and Alan Johnston, provided useful ideas for RFC 4582 [3]. Additionally, Xiaotao Wu, Paul Kyzivat, Jonathan Rosenberg, Miguel A. Garcia-Martin, Mary Barnes, Ben Campbell, Dave Morgan, and Oscar Novo provided useful comments during the work with RFC 4582. The authors also acknowledge contributions to the revision of BFCP for use over an unreliable transport from Geir Arne Sandbakken who had the initial idea, Alfred E. Heggestad, Trond G. Andersen, Gonzalo Camarillo, Roni Even, Lorenzo Miniero, Jörg Ott, Eoin McLeod, Mark K. Thompson, Hadriel Kaplan, Dan Wing, Cullen Jennings, David Benham, Nivedita Melinkeri, Woo Johnman, Vijaya Mandava, and Alan Ford. In the final phase, Ernst Horvath did a thorough review, revealing issues that needed clarification and changes. Useful and important final reviews were done by Mary Barnes. Paul Jones helped tremendously as editor for changes addressing IESG review comments.

Authors' Addresses

Gonzalo Camarillo
Ericsson
Hirsalantie 11
FI-02420 Jorvas
Finland
Email: gonzalo.camarillo@ericsson.com

Keith Drage
Email: drageke@ntlworld.com

Tom Kristensen
Jotron AS
Ringdalskogen 8
3270 Larvik
Norway
Email: tom.kristensen@jotron.com, tomkri@ifi.uio.no

Jörg Ott
Technical University Munich
Boltzmannstrasse 3
85748 Garching
Germany
Email: ott@in.tum.de

Charles Eckel
Cisco
707 Tasman Drive
Milpitas, California 95035
United States of America
Email: eckelcu@cisco.com