

BFCPbis Working Group
Internet-Draft
Obsoletes: 4582 (if approved)
Intended status: Standards Track
Expires: August 19, 2014

G. Camarillo
Ericsson
K. Drage
Alcatel-Lucent
T. Kristensen
Cisco
J. Ott
Aalto University
C. Eckel
Cisco
February 15, 2014

The Binary Floor Control Protocol (BFCP)
draft-ietf-bfcpbis-rfc4582bis-11

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. Changes from RFC 4582 are summarized in Section 16.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 19, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	6
2. Terminology	6
3. Scope	7
3.1. Floor Creation	9
3.2. Obtaining Information to Contact a Floor Control Server	9
3.3. Obtaining Floor-Resource Associations	9
3.4. Privileges of Floor Control	10
4. Overview of Operation	10
4.1. Floor Participant to Floor Control Server Interface	11
4.2. Floor Chair to Floor Control Server Interface	15
5. Packet Format	16
5.1. COMMON-HEADER Format	16
5.2. Attribute Format	19
5.2.1. BENEFICIARY-ID	21
5.2.2. FLOOR-ID	21
5.2.3. FLOOR-REQUEST-ID	22
5.2.4. PRIORITY	22
5.2.5. REQUEST-STATUS	23
5.2.6. ERROR-CODE	24
5.2.6.1. Error-Specific Details for Error Code 4	25
5.2.7. ERROR-INFO	25
5.2.8. PARTICIPANT-PROVIDED-INFO	26
5.2.9. STATUS-INFO	27
5.2.10. SUPPORTED-ATTRIBUTES	27
5.2.11. SUPPORTED-PRIMITIVES	28
5.2.12. USER-DISPLAY-NAME	29
5.2.13. USER-URI	29
5.2.14. BENEFICIARY-INFORMATION	30
5.2.15. FLOOR-REQUEST-INFORMATION	31
5.2.16. REQUESTED-BY-INFORMATION	32

5.2.17. FLOOR-REQUEST-STATUS	32
5.2.18. OVERALL-REQUEST-STATUS	33
5.3. Message Format	34
5.3.1. FloorRequest	34
5.3.2. FloorRelease	34
5.3.3. FloorRequestQuery	34
5.3.4. FloorRequestStatus	35
5.3.5. UserQuery	35
5.3.6. UserStatus	35
5.3.7. FloorQuery	36
5.3.8. FloorStatus	36
5.3.9. ChairAction	36
5.3.10. ChairActionAck	36
5.3.11. Hello	37
5.3.12. HelloAck	37
5.3.13. Error	37
5.3.14. FloorRequestStatusAck	38
5.3.15. FloorStatusAck	38
5.3.16. Goodbye	38
5.3.17. GoodbyeAck	38
6. Transport	39
6.1. Reliable Transport	39
6.2. Unreliable Transport	40
6.2.1. Congestion Control	42
6.2.2. ICMP Error Handling	42
6.2.3. Fragmentation Handling	43
6.2.4. NAT Traversal	44
7. Lower-Layer Security	44
8. Protocol Transactions	45
8.1. Client Behavior	46
8.2. Server Behavior	46
8.3. Timers	47
8.3.1. Request Retransmission Timer, T1	47
8.3.2. Response Retransmission Timer, T2	47
8.3.3. Timer Values	47
9. Authentication and Authorization	48
9.1. TLS/DTLS Based Mutual Authentication	48
10. Floor Participant Operations	49
10.1. Requesting a Floor	49
10.1.1. Sending a FloorRequest Message	49
10.1.2. Receiving a Response	50
10.1.3. Reception of a Subsequent FloorRequestStatus Message	51
10.2. Cancelling a Floor Request and Releasing a Floor	52
10.2.1. Sending a FloorRelease Message	52
10.2.2. Receiving a Response	52
11. Chair Operations	53
11.1. Sending a ChairAction Message	53

11.2. Receiving a Response	54
12. General Client Operations	55
12.1. Requesting Information about Floors	55
12.1.1. Sending a FloorQuery Message	55
12.1.2. Receiving a Response	55
12.1.3. Reception of a Subsequent FloorStatus Message	56
12.2. Requesting Information about Floor Requests	56
12.2.1. Sending a FloorRequestQuery Message	57
12.2.2. Receiving a Response	57
12.3. Requesting Information about a User	57
12.3.1. Sending a UserQuery Message	58
12.3.2. Receiving a Response	58
12.4. Obtaining the Capabilities of a Floor Control Server	59
12.4.1. Sending a Hello Message	59
12.4.2. Receiving Responses	59
13. Floor Control Server Operations	59
13.1. Reception of a FloorRequest Message	60
13.1.1. Generating the First FloorRequestStatus Message	60
13.1.2. Generation of Subsequent FloorRequestStatus Messages	62
13.2. Reception of a FloorRequestQuery Message	63
13.3. Reception of a UserQuery Message	64
13.4. Reception of a FloorRelease Message	66
13.5. Reception of a FloorQuery Message	67
13.5.1. Generation of the First FloorStatus Message	67
13.5.2. Generation of Subsequent FloorStatus Messages	69
13.6. Reception of a ChairAction Message	69
13.7. Reception of a Hello Message	70
13.8. Error Message Generation	71
14. Security Considerations	71
15. IANA Considerations	72
15.1. Attribute Subregistry	72
15.2. Primitive Subregistry	73
15.3. Request Status Subregistry	74
15.4. Error Code Subregistry	75
16. Changes from RFC 4582	76
16.1. Extensions for an unreliable transport	76
16.2. Other changes	77
17. Acknowledgements	78
18. References	79
18.1. Normative References	79
18.2. Informational References	79
Appendix A. Example Call Flows for BFCP over an Unreliable Transport	81
Appendix B. Motivation for Supporting an Unreliable Transport	84
B.1. Motivation	85
B.1.1. Alternatives Considered	86
B.1.1.1. ICE TCP	86

B.1.1.2.	Teredo	87
B.1.1.3.	GUT	87
B.1.1.4.	UPnP IGD	87
B.1.1.5.	NAT PMP	88
B.1.1.6.	SCTP	88
B.1.1.7.	BFCP over UDP transport	88
Authors' Addresses	89

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 [13] 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, and subsequent sections provide the normative specification of BFCP.

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, RFC 2119 [1] and indicate requirement levels for compliant implementations.

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 colocated 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 colocated. The protocol between a floor participant and a media participant (that are not colocated) 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 exist, 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 colocated 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 colocated.

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.

3. Scope

As stated earlier, BFCP is a protocol to coordinate access to shared resources in a conference following the requirements defined in [13]. Floor control complements other functions defined in the XCON conferencing framework [14]. 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 [14].

Figure 1 shows the tasks that BFCP can perform.

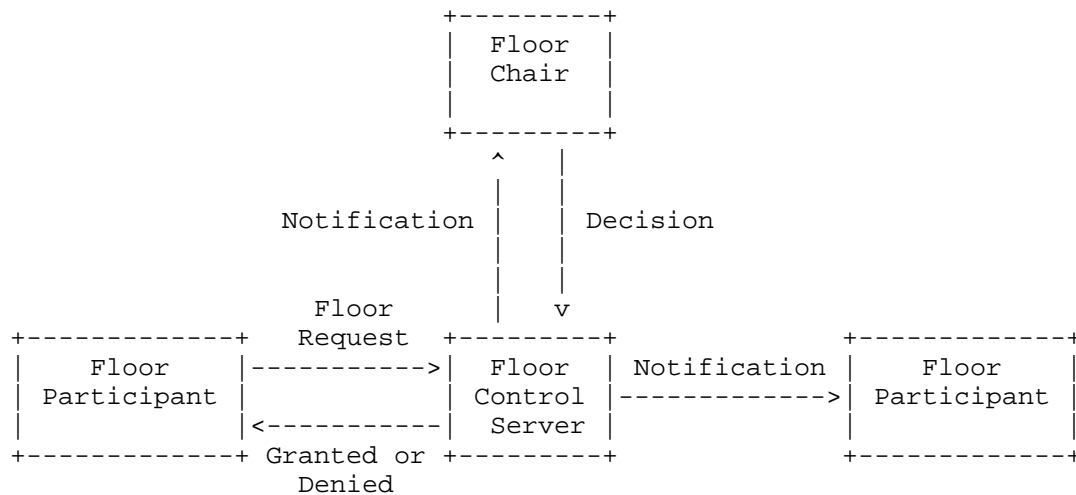


Figure 1: Functionality provided by BFCP

BFCP provides a means:

- o for floor participants to send floor requests to floor control servers.
- o for floor control servers to grant or deny requests to access a given resource from floor participants.
- o for floor chairs to send floor control servers decisions regarding floor requests.
- o 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 [14]. 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 CCMP [18] can specify such floor-related settings in the <floor-information> element [17] 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. This data includes 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 an SDP offer/answer [12] exchange, which is described in [9]. How to establish a connection to a BFCP floor control server outside the context of an offer/answer exchange is described in [3]. Other mechanisms are described in the XCON framework [14] (and other related documents).

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 [12] exchange. How to use an SDP offer/answer exchange to obtain these associations is described in [9].

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 [14] (and other related documents). According to the conferencing system policies, conference control clients using CCMP [18] 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 [19].

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 [14].

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 colocated 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.

Figures 2 and 3 below 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
        Floor ID: 543
<-----

```

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. 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
<pre> (1) FloorQuery Transaction ID: 257 User ID: 234 FLOOR-ID: 543 </pre>	<pre> -----> </pre>
<pre> (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 </pre>	

```
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.

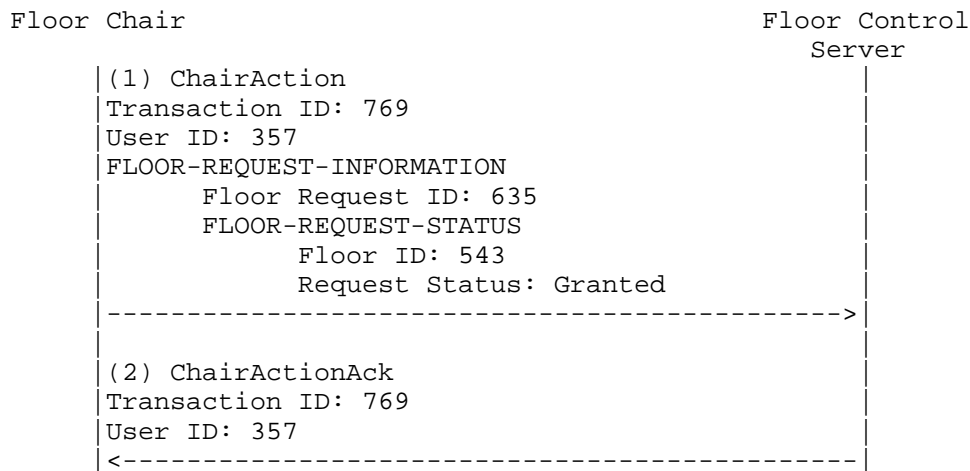


Figure 4: Chair instructing the floor control server

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.

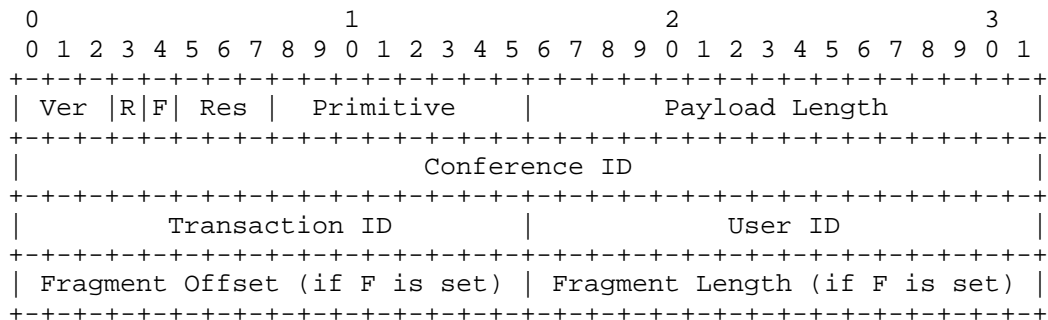


Figure 5: COMMON-HEADER format

Ver: This 3-bit field defines the version of BFCP that this message adheres to. This specification defines two versions: 1 and 2. The version field MUST be set to 1 when using BFCP over a reliable

transport, i.e. as in [2]. The version field MUST be set to 2 when using BFCP over an unreliable transport with the extensions specified in this document. If an endpoint receives a message with an unsupported version field value, the receiving server MUST send an Error message with parameter value 12 (Unsupported Version) to indicate this.

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 MUST be ignored by the receiver.

Res: At this point, 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
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.

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 the message belongs to.

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.

5.2. Attribute Format

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

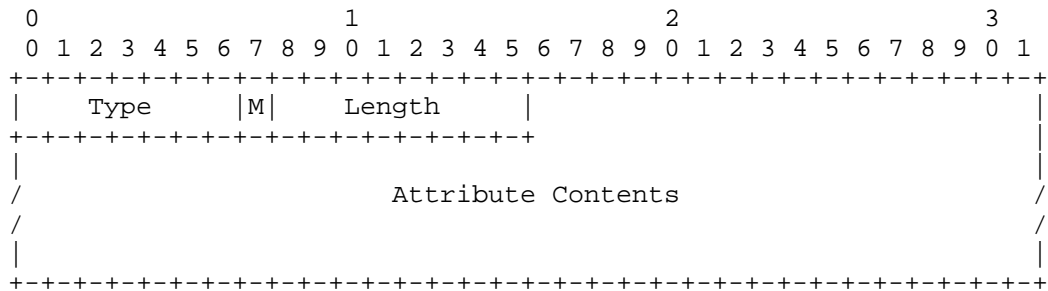


Figure 6: Attribute format

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. In all other cases, the unrecognized attribute is 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.

```

      0                               1                               2                               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|0 0 0 0 0 0 1|M|0 0 0 0 0 1 0 0|          Beneficiary ID          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

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.

```

      0                               1                               2                               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|0 0 0 0 0 1 0|M|0 0 0 0 0 1 0 0|          Floor ID          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

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.

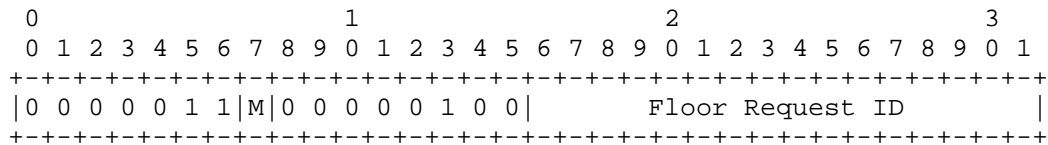


Figure 9: FLOOR-REQUEST-ID format

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.

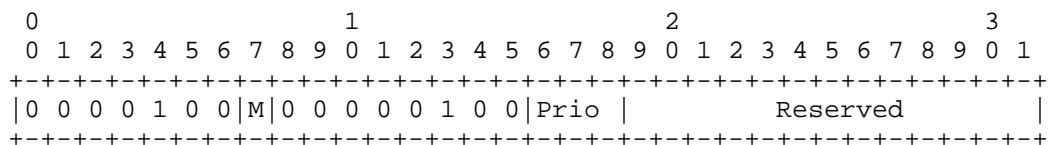


Figure 10: PRIORITY format

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
1	Low
2	Normal
3	High
4	Highest

Table 3: Priority values

Reserved: At this point, 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.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|0 0 0 0 1 0 1|M|0 0 0 0 0 1 0 0|Request Status |Queue Position |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

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.

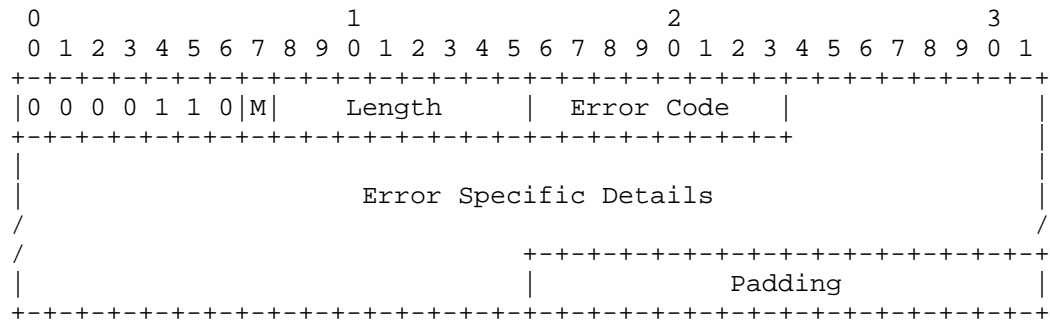


Figure 12: ERROR-CODE format

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
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 by a BFCP entity 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, 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.

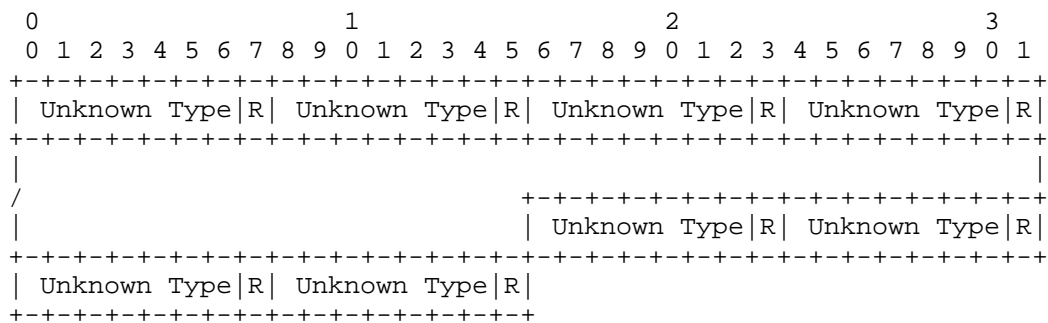


Figure 13: Unknown attributes format

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.

R: At this point, 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.

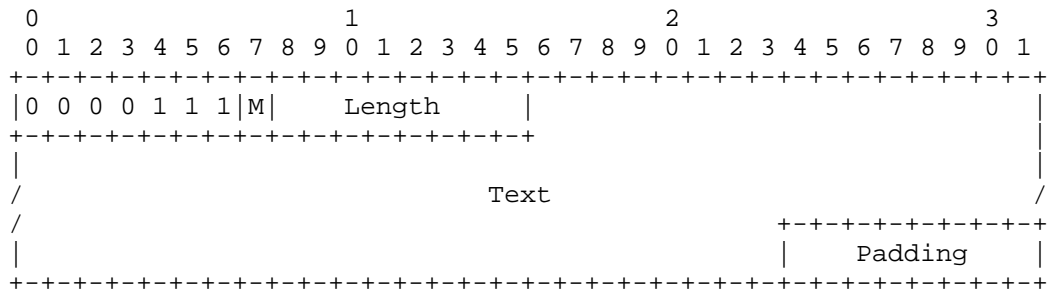


Figure 14: ERROR-INFO format

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

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.

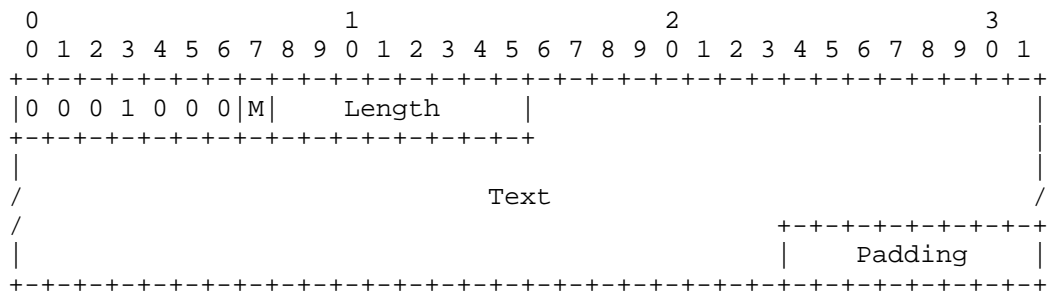


Figure 15: PARTICIPANT-PROVIDED-INFO format

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

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.

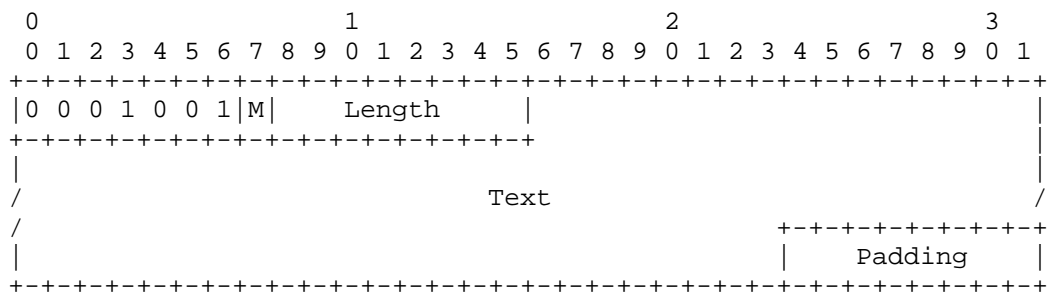


Figure 16: STATUS-INFO format

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

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.

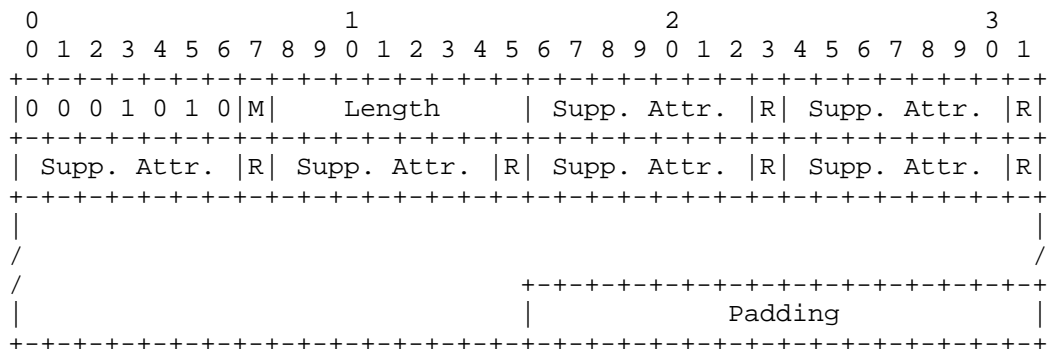


Figure 17: SUPPORTED-ATTRIBUTES format

Supp. Attr.: These fields contain the Types of the attributes that are supported by the floor control server in the following format:

R: Reserved: 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.

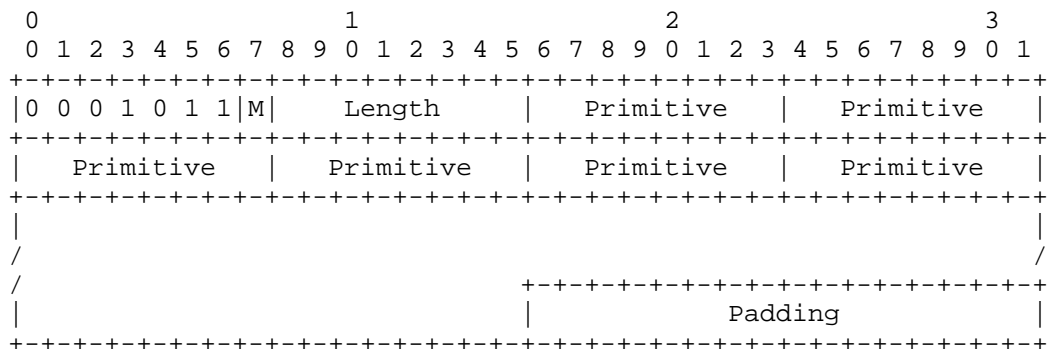


Figure 18: SUPPORTED-PRIMITIVES format

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.

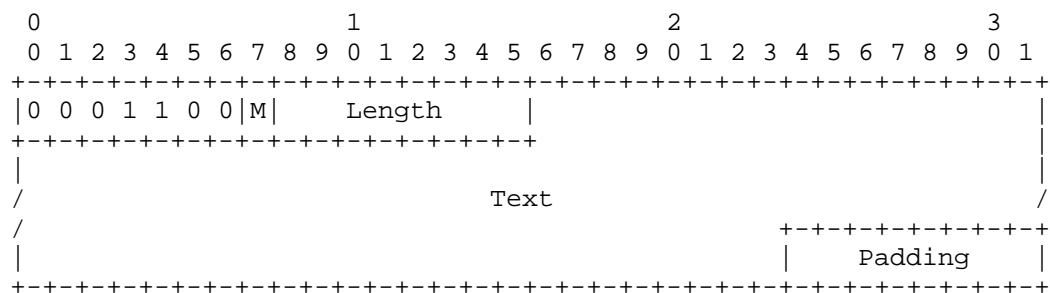


Figure 19: USER-DISPLAY-NAME format

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.

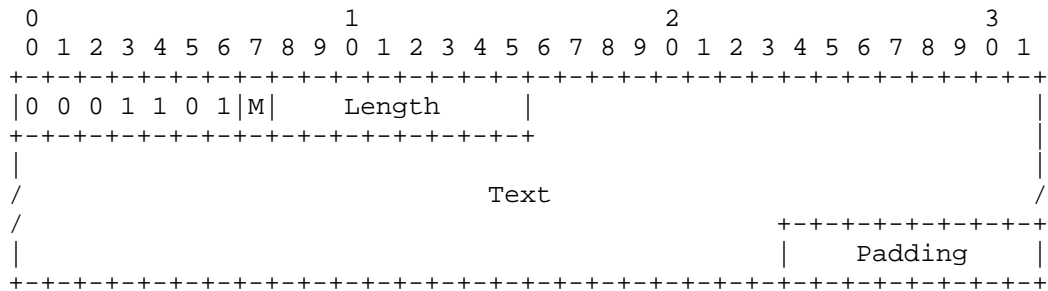


Figure 20: USER-URI format

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:

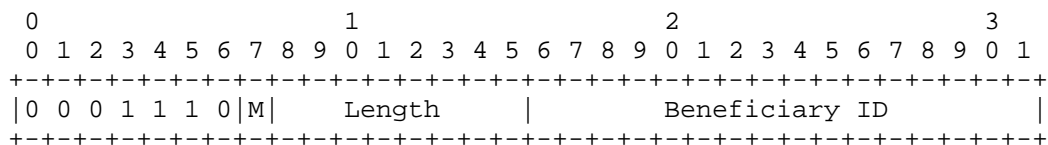


Figure 21: BENEFICIARY-INFORMATION-HEADER format

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) [4] 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:

0	1	2	3
0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9
+-----+-----+-----+-----+			
0 0 0 1 1 1 1 M Length Floor Request ID			
+-----+-----+-----+-----+			

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:

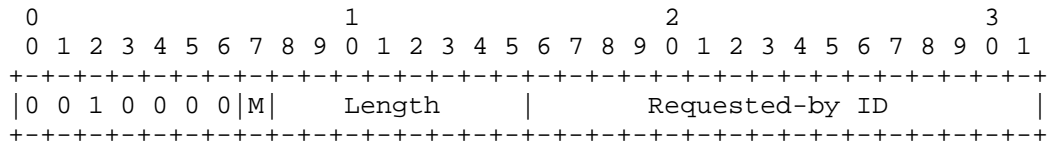


Figure 25: REQUESTED-BY-INFORMATION-HEADER format

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:

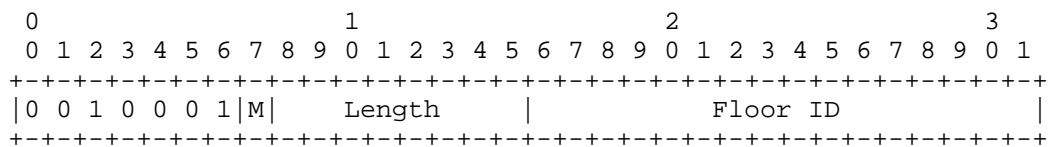


Figure 27: FLOOR-REQUEST-STATUS-HEADER format

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:

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
0 0 1 0 0 1 0 M										Length										Floor Request ID																			

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) [4] 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 FloorRequest 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 check the liveliness of floor control servers by sending a Hello message. 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 should 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 how clients obtain information to contact the floor control server (e.g., using an SDP offer/answer exchange [9] or the procedure specified in [3]). Two transports are supported: TCP, 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.

Informational note: In practice products are configured to try one transport initially and use the other one as a fallback. Whether TCP or UDP is chosen as underlying transport depends on type of product and the nature of the environment it is deployed. Here Appendix B are important to consider.

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 from the conference), the floor control server closes (i.e., a graceful close) the TCP connection towards the client.

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 service as present and available only upon receiving the HelloAck message. Situations where the floor control service is considered to have become unavailable due to ICMP messages is described in Section 6.2.2 and the behavior when timers fire 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 back from the floor control server within a retransmission 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 or 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. A server-initiated request (e.g., a FloorStatus with an update from the floor control server) received by a participant before the initial FloorRequestStatus message that closes the client-initiated transaction that was instigated by the FloorRequest MUST be treated as superseding the information conveyed in any such late arriving response. 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.

RFC 5018 [3] specifies how to establish a TCP connection to a floor control server outside the context of an offer/answer exchange. When using UDP the same set of data is needed for a BFCP connection as listed in [3], Section 3, i.e. transport address of the server, the conference identifier, and the user identifier. The procedures and considerations for resolving a host name into an IP address also applies to BFCP over an unreliable transport. In [3], Section 4 applies, but when using BFCP over an unreliable transport the floor control server that receives a BFCP message over UDP (no DTLS) SHOULD request the use of DTLS by generating an Error message with an Error code with a value of 11 (Use DTLS). A floor control server that is

configured to require DTLS MUST request the use of DTLS this way. The recommendations for authentication in [3], Section 5 and the security considerations in [3], Section 6 also apply when an unreliable transport is used, both for certificate-based server authentication and for client authentication based on a pre-shared secret.

6.2.1. Congestion Control

BFCP may be characterized to generate "low data-volume" traffic, per the classification in [25]. Nevertheless it is necessary to ensure 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. In case 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 500ms and the interval MUST be doubled after each retransmission attempt. This is identical to the specification of the timer A and its initial value T1 in SIP as described in Section 17.1.1.2 of [16].

6.2.2. ICMP Error Handling

If a BFCP entity receives an ICMP port unreachable message mid-connection, the entity MUST treat the BFCP connection as closed (e.g., an implicit Goodbye message from the peer). The entity MAY attempt to re-establish the BFCP connection afresh. The new BFCP connection will appear as originating from a wholly new floor participant, chair or floor control server with all state previously held about that participant lost.

Informational note: The recommendation to treat the connection as closed in this case, stems from the fact that the peer entities cannot rely on IP and port tuple to uniquely identify the participant, nor would extending Hello to include an attribute that advertised what identity the entity previously was assigned (i.e., a User ID) be acceptable due to session hijacking.

In deployments where NAT appliances or other such devices are present and affecting port reachability for each entity, one possibility is to utilize the peer connectivity checks, relay use and NAT pinhole maintenance mechanisms defined in ICE [15].

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 being required 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 MTU size available between the sender and the receiver and MAY run MTU discovery, such as [20][21][22], for this purpose.

When transmitting a BFCP message with size greater than the path MTU, the sender MUST fragment the message into a series of N contiguous data ranges. The value for N is defined as $\text{ceil}(\text{message size} / \text{MTU size})$, where ceil is the integer ceiling function. 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 MUST be smaller than the path MTU. The F flag in the COMMON-HEADER 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 bytes contained in the previous fragments. The Fragment Length contains the length of the fragment itself. 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 for 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 [7].

When deciding message fragment size 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 [7].

6.2.4. NAT Traversal

One of the key benefits when 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 ICE [15].

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 [11] Binding Indication for keep-alives, as described for ICE [15]. [23], Section 6.7 provides useful recommendations for middlebox interaction when DTLS is used.

Informational 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 straight forward to distinguish between STUN and BFCP packets even without checking the STUN magic cookie [11].

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/RTCP [10].

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 [6] and MUST support DTLS [7] 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 ciphersuite [6].

Which party, the client or the floor control server, acts as the TLS/DTLS server depends on how the underlying TLS/DTLS connection is established. For a TCP/TLS connection established using an SDP offer/answer exchange [9], the answerer (which may be the client or the floor control server) always acts as the TLS server. If the TCP connection is lost, the active endpoint, i.e., the current TLS client, is responsible for re-establishing the TCP connection. Unless a new TLS session is negotiated, subsequent SDP offers and answers will not impact the previously negotiated TLS roles.

For a UDP/DTLS connection established using the an SDP offer/answer exchange, either party can be the DTLS server depending on the setup attributes exchanged; examples can be found in [23].

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. The request carries a Transaction ID in its common header, which the floor control server copies into the response. Clients use Transaction ID values to match responses with previously issued requests.

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). Since they do not trigger any response, their Transaction ID is set to 0.

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. The Transaction ID MUST be non-zero and unique in the context of outstanding transactions over an unreliable transport. The request carries a Transaction ID in its common header, which the

client copies into the response. Floor control servers use Transaction ID values to match responses with previously issued requests.

When using BFCP over an unreliable transport, it is important that the initiator of a transaction 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).

When using BFCP over an unreliable transport, all requests will use retransmission timer T1 (see Section 8.3) until the transaction is completed.

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.

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 0 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 **MUST** set the Transaction ID value in the common header to a number that is different from 0 and that **MUST NOT** be reused, i.e. monotonically increasing value as defined in Section 8. 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 against 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 have occurred. The timer doubles on each retransmit, 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 failed. Implementations **SHOULD** follow the reestablishment procedure described in section 6 (e.g., initiate a new offer/answer [12] exchange).

8.3.2. Response Retransmission Timer, T2

T2 is a timer that, when fires, 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, whilst that value is retained until the timer has fired. Refer to Section 6.2.3 for the role this timer has 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	0.5s
T2	Response retransmission timer	10s

Table 6: Timers

The default value for T1 is 500 ms, this is an estimate of the RTT for completing the transaction. T1 **MAY** be chosen larger, and this is **RECOMMENDED** if it is known in advance that the RTT is larger.

Regardless of the value of T1, the exponential backoffs on retransmissions described in Section 8.3.1 MUST be used.

T2 SHALL be set such that it encompasses all legal retransmissions per T1 plus a factor to accommodate network latency between BFCP entities. The default value is based on the sum of the three retransmissions related to T1 using its default value (7.5s) and an extra 2.5s is added to take into account potential messages in transit due to latency.

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.

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 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 [9].

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. This User ID will be used by the floor control server to authenticate and authorize the request. 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 name space 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. This User ID will be used by the floor control server to authenticate and authorize the request.

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 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.

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. This User ID will be used by the floor control server to authenticate and authorize the request.

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. This User ID will be used by the floor control server to authenticate and authorize the request.

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 FloorRequest 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. This User ID will be used by the floor control server to authenticate and authorize the request.

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. This User ID will be used by the floor control server to authenticate and authorize the request.

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. This User ID will be used by the floor control server to authenticate and authorize the request.

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 0. 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 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. If the version given in the Hello message is not supported, the receiving server MUST instead send an Error message with parameter value 12 (Unsupported Version).

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 non-null encryption always be used. BFCP entities MAY use other security mechanisms as long as they provide similar security properties.

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 high-jack 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 RECOMMENDED that TLS/DTLS be used with a non-null encryption algorithm.

15. IANA Considerations

[Editorial note: This section instructs the IANA to register new entries in the BFCP Primitive subregistry in Section 15.2 and for the BFCP Error Code subregistry in Section 15.4.]

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

15.1. Attribute Subregistry

This section establishes the Attribute subregistry under the BFCP Parameters registry. As per the terminology in RFC 5226 [5], the registration policy for BFCP attributes shall be "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 XXXX]
2	FLOOR-ID	[RFC XXXX]
3	FLOOR-REQUEST-ID	[RFC XXXX]
4	PRIORITY	[RFC XXXX]
5	REQUEST-STATUS	[RFC XXXX]
6	ERROR-CODE	[RFC XXXX]
7	ERROR-INFO	[RFC XXXX]
8	PARTICIPANT-PROVIDED-INFO	[RFC XXXX]
9	STATUS-INFO	[RFC XXXX]
10	SUPPORTED-ATTRIBUTES	[RFC XXXX]
11	SUPPORTED-PRIMITIVES	[RFC XXXX]
12	USER-DISPLAY-NAME	[RFC XXXX]
13	USER-URI	[RFC XXXX]
14	BENEFICIARY-INFORMATION	[RFC XXXX]
15	FLOOR-REQUEST-INFORMATION	[RFC XXXX]
16	REQUESTED-BY-INFORMATION	[RFC XXXX]
17	FLOOR-REQUEST-STATUS	[RFC XXXX]
18	OVERALL-REQUEST-STATUS	[RFC XXXX]

Table 7: Initial values of the BFCP Attribute subregistry

15.2. Primitive Subregistry

[Editorial note: This section instructs the IANA to register the following new values for the BFCP Primitive subregistry:
FloorRequestStatusAck, FloorStatusAck, Goodbye, and GoodbyeAck.]

This section establishes the Primitive subregistry under the BFCP Parameters registry. As per the terminology in RFC 5226 [5], the registration policy for BFCP primitives shall be "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 XXXX]
2	FloorRelease	[RFC XXXX]
3	FloorRequestQuery	[RFC XXXX]
4	FloorRequestStatus	[RFC XXXX]
5	UserQuery	[RFC XXXX]
6	UserStatus	[RFC XXXX]
7	FloorQuery	[RFC XXXX]
8	FloorStatus	[RFC XXXX]
9	ChairAction	[RFC XXXX]
10	ChairActionAck	[RFC XXXX]
11	Hello	[RFC XXXX]
12	HelloAck	[RFC XXXX]
13	Error	[RFC XXXX]
14	FloorRequestStatusAck	[RFC XXXX]
15	FloorStatusAck	[RFC XXXX]
16	Goodbye	[RFC XXXX]
17	GoodbyeAck	[RFC XXXX]

Table 8: Initial values of the BFCP primitive subregistry

15.3. Request Status Subregistry

This section establishes the Request Status subregistry under the BFCP Parameters registry. As per the terminology in RFC 5226 [5], the registration policy for BFCP request status shall be "Specification Required". For the purposes of this subregistry, the BFCP request status 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 XXXX]
2	Accepted	[RFC XXXX]
3	Granted	[RFC XXXX]
4	Denied	[RFC XXXX]
5	Cancelled	[RFC XXXX]
6	Released	[RFC XXXX]
7	Revoked	[RFC XXXX]

Table 9: Initial values of the Request Status subregistry

15.4. Error Code Subregistry

[Editorial note: This section instructs the IANA to register the following new values for the BFCP Error Code subregistry: 10, 11, 12, 13 and 14.]

This section establishes the Error Code subregistry under the BFCP Parameters registry. As per the terminology in RFC 5226 [5], the registration policy for BFCP error codes shall be "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 XXXX]
2	User does not Exist	[RFC XXXX]
3	Unknown Primitive	[RFC XXXX]
4	Unknown Mandatory Attribute	[RFC XXXX]
5	Unauthorized Operation	[RFC XXXX]
6	Invalid Floor ID	[RFC XXXX]
7	Floor Request ID Does Not Exist	[RFC XXXX]
8	You have Already Reached the Maximum Number of Ongoing Floor Requests for this Floor	[RFC XXXX]
9	Use TLS	[RFC XXXX]
10	Unable to parse message	[RFC XXXX]
11	Use DTLS	[RFC XXXX]
12	Unsupported Version	[RFC XXXX]
13	Incorrect Message Length	[RFC XXXX]
14	Generic Error	[RFC XXXX]

Table 10: Initial Values of the Error Code subregistry

16. Changes from RFC 4582

Following is the list of technical changes and other non-trivial fixes from [2].

16.1. Extensions for an unreliable transport

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):
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. Where the specification for an unreliable transport amongst other issues deals with reliability, congestion control, fragmentation and ICMP.
7. Mandate DTLS (Section 7 and Section 9):
Mandate DTLS support when transport over UDP is used.
8. Transaction changes (Section 8):
Server-initiated transactions over an unreliable transport has non-zero and unique Transaction ID. Over an unreliable transport, the retransmit timers T1 and T2 described in Section 8.3 apply.
9. Requiring timely response (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):
Describing 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):
Introduction to and motivation for extending BFCP to support an unreliable transport.

16.2. Other changes

The clarification and bug fixes:

1. ABNF fixes (Figure 22, Figure 24, ="fig:reqby-information"/>, Figure 28, Figure 30, and the ABNF figures in Section 5.3): Although formally correct in [2], 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):
Change from SUPPORTED-PRIMITVIES to SUPPORTED-PRIMITIVES in the second paragraph.
3. Corrected attribute type (Section 13.1.1):
Change from PARTICIPANT-PROVIDED-INFO to PRIORITY attributed 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. Assorted clarifications (Across the document):
Non-functional language clarifications and some corrections in the normative language as a result of reviews.

17. Acknowledgements

The XCON WG chairs, Adam Roach and Alan Johnston, provided useful ideas for RFC 4582 [2]. 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. Heggstad, Trond G. Andersen, Gonzalo Camarillo, Roni Even, Lorenzo Miniero, Joerg Ott, Eoin McLeod, Mark K. Thompson, Hadriel Kaplan, Dan Wing, Cullen Jennings, David Benham, Nivedita Melinker, 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.

18. References

18.1. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [2] Camarillo, G., Ott, J., and K. Drage, "The Binary Floor Control Protocol (BFCP)", RFC 4582, November 2006.
- [3] Camarillo, G., "Connection Establishment in the Binary Floor Control Protocol (BFCP)", RFC 5018, September 2007.
- [4] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [5] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [6] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [7] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [9] Camarillo, G. and T. Kristensen, "Session Description Protocol (SDP) Format for Binary Floor Control Protocol (BFCP) Streams", draft-ietf-bfcpbis-rfc4583bis-09 (work in progress), February 2014.
- [10] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, July 2007.
- [11] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

18.2. Informational References

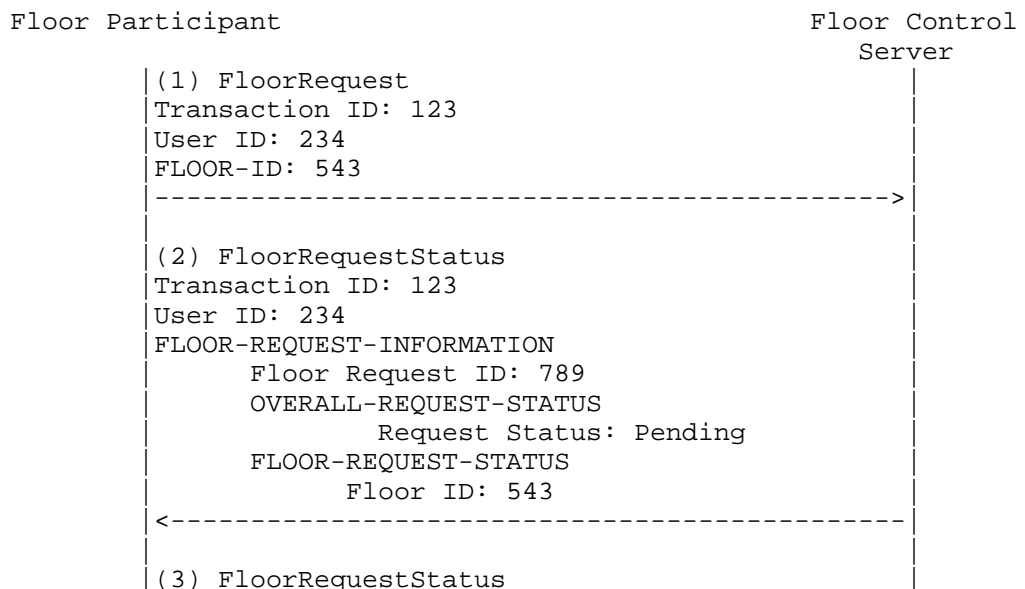
- [12] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [13] Koskelainen, P., Ott, J., Schulzrinne, H., and X. Wu, "Requirements for Floor Control Protocols", RFC 4376, February 2006.
- [14] Barnes, M., Boulton, C., and O. Levin, "A Framework for Centralized Conferencing", RFC 5239, June 2008.

- [15] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [16] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [17] Novo, O., Camarillo, G., Morgan, D., and J. Urpalainen, "Conference Information Data Model for Centralized Conferencing (XCON)", RFC 6501, March 2012.
- [18] Barnes, M., Boulton, C., Romano, S., and H. Schulzrinne, "Centralized Conferencing Manipulation Protocol", RFC 6503, March 2012.
- [19] Barnes, M., Miniero, L., Presta, R., and SP. Romano, "Centralized Conferencing Manipulation Protocol (CCMP) Call Flow Examples", RFC 6504, March 2012.
- [20] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.
- [21] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.
- [22] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [23] 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, May 2010.
- [24] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, February 2006.
- [25] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
- [26] Thaler, D., "Teredo Extensions", RFC 6081, January 2011.
- [27] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [28] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, March 2012.

- [29] Manner, J., Varis, N., and B. Briscoe, "Generic UDP Tunnelling (GUT)", draft-manner-tsvwg-gut-02 (work in progress), July 2010.
- [30] Stucker, B., Tschofenig, H., and G. Salgueiro, "Analysis of Middlebox Interactions for Signaling Protocol Communication along the Media Path", draft-ietf-mmusic-media-path-middleboxes-05 (work in progress), July 2012.
- [31] Guha, S. and P. Francis, "Characterization and Measurement of TCP Traversal through NATs and Firewalls", 2005, <<http://saikat.guha.cc/pub/imc05-tcpnat.pdf>>.
- [32] Ford, B., Srisuresh, P., and D. Kegel, "Peer-to-Peer Communication Across Network Address Translators", April 2005, <<http://www.brynosaurus.com/pub/net/p2pnat.pdf>>.

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

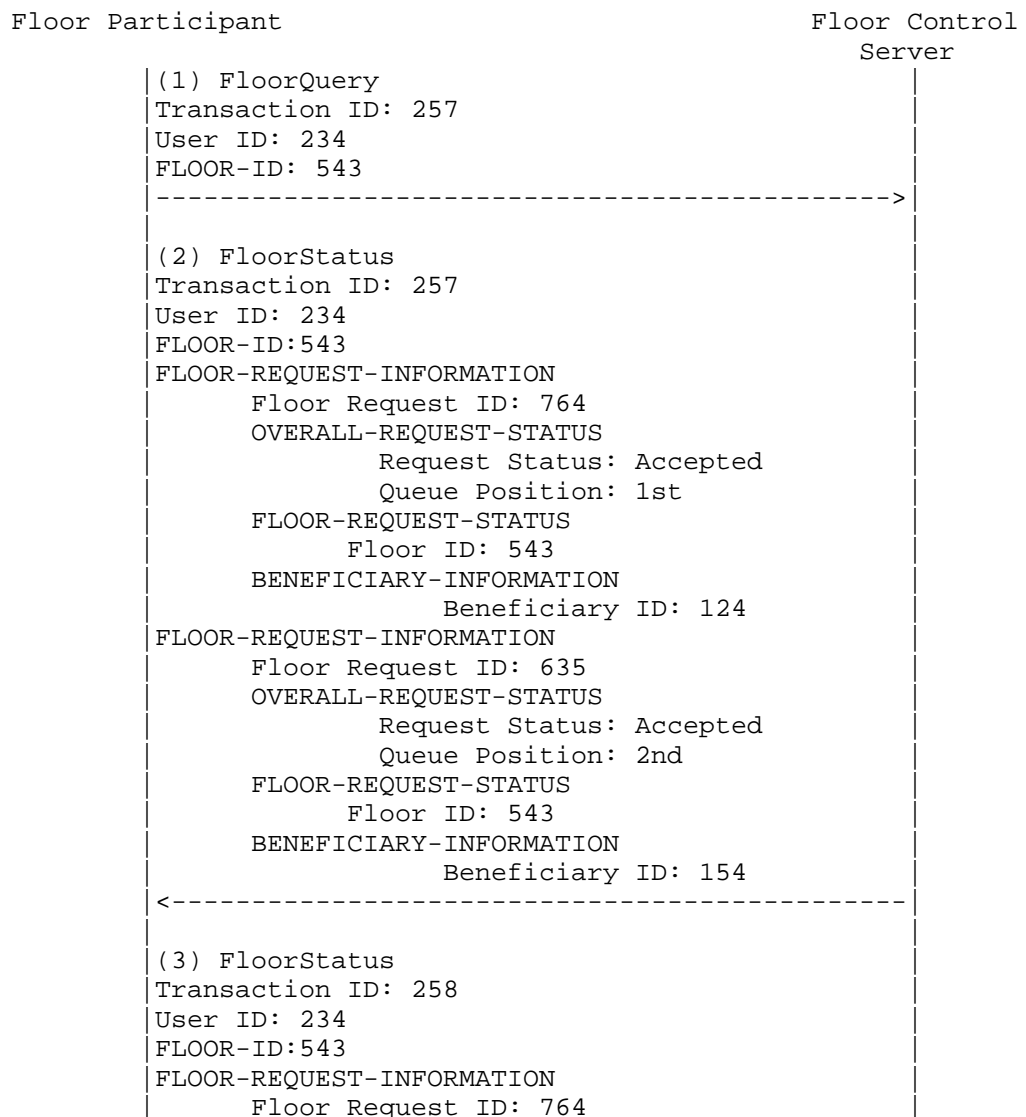
With reference to Section 4.1, the following figures show representative call-flows for requesting and releasing a floor, and obtaining status information about a floor when BFCP is deployed over an unreliable transport. The figures here show a loss-less interaction.



```
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 ID: 124
User ID: 234
----->
(5) FloorRequestStatus
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 ID: 125
User ID: 234
----->
(7) FloorRelease
Transaction ID: 126
User ID: 234
FLOOR-REQUEST-ID: 789
----->
(8) FloorRequestStatus
Transaction ID: 126
User ID: 234
FLOOR-REQUEST-INFORMATION
    Floor Request ID: 789
    OVERALL-REQUEST-STATUS
        Request Status: Released
    FLOOR-REQUEST-STATUS
        Floor ID: 543
<-----
```

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 154. It does not and SHOULD NOT be followed by a FloorRequestStatusAck message from the floor participant to the floor control server.



```

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) FloorStatusAck
Transaction ID: 258
User ID: 234
----->
(5) FloorStatus
Transaction ID: 259
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
<-----
(6) FloorStatusAck
Transaction ID: 259
User ID: 234
----->

```

Figure 49: Obtaining status information about a floor

Appendix B. Motivation for Supporting an Unreliable Transport

This appendix is contained in this document 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 exists behind a NAT. BFCP roles are negotiated in the offer/answer exchange as specified in [9], resulting in one endpoint being responsible for opening the TCP connection used for the BFCP communication.

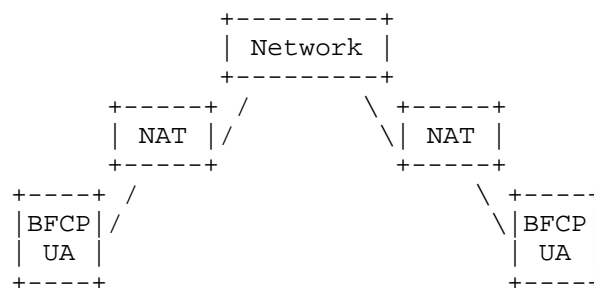


Figure 50: Use Case

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 [30].

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., 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, 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 [28]. 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 it was not

possible to establish a TCP connection in 11% of the cases [31]. 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 [32].

It is worth noticing that BFCP over UDP were already 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 would co-exist in the market. The purpose of this draft is to formalize and publish the extension from the standard specification to facilitate complete interoperability between implementations.

B.1.1.1. 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 not chosen alternatives work in a number of scenarios, they are not sufficient, in and of themselves, to address the use case targeted by this draft. The last alternative, presented in Appendix B.1.1.7, is the selected one and is specified in this draft.

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

B.1.1.1.1. ICE TCP

ICE TCP [28] 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 [28]) 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 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 to recently, or because they do not properly process the subsequent SYNACK. Implementing various techniques advocated for candidate collection in [28] 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 [24] enables nodes located behind one or more IPv4 NATs to obtain IPv6 connectivity by tunneling packets over UDP. Teredo extensions [26] 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 draft. However, running the service requires the help of "Teredo servers" and "Teredo relays" [24]. These servers and relays generally do not exist in the existing 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 transition 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 [29] 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 draft has expired and its future is still not clear 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 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 straight forward to specify a BFCP binding for SCTP [27], and then tunnel SCTP over UDP in the use case described in Appendix B.1. SCTP is gaining some momentum currently. There is ongoing discussion in the RTCWeb WG regarding this approach. However, this approach for tunneling over UDP was not mature enough when considered and 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, it is recommended to follow the guidelines provided in [25].

Minor changes to the transaction model are introduced in that all requests now have an appropriate response to complete the transaction. The requests are sent with a retransmit 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 versions of this draft, have demonstrated this approach to be

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 is specified in this document.

Authors' Addresses

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: gonzalo.camarillo@ericsson.com

Keith Drage
Alcatel-Lucent
Quadrant, StoneHill Green, Westlea
Swindon, Wilts
UK

Email: drage@alcatel-lucent.com

Tom Kristensen
Cisco
Philip Pedersens vei 22
N-1366 Lysaker
Norway

Email: tomkrist@cisco.com, tomkri@ifi.uio.no

Joerg Ott
Aalto University
Otakaari 5 A
Espoo, FIN 02150
Finland

Email: jo@comnet.tkk.fi

Charles Eckel
Cisco
707 Tasman Drive
California, CA 95035
United States

Email: eckelcu@cisco.com

BFCPbis Working Group
Internet-Draft
Obsoletes: 4582 (if approved)
Intended status: Standards Track
Expires: May 16, 2016

G. Camarillo
Ericsson
K. Drage
Alcatel-Lucent
T. Kristensen
Cisco
J. Ott
Aalto University
C. Eckel
Cisco
November 13, 2015

The Binary Floor Control Protocol (BFCP)
draft-ietf-bfcpbis-rfc4582bis-16

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. Changes from RFC 4582 are summarized in Section 16.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 16, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	5
2. Terminology	5
3. Scope	7
3.1. Floor Creation	8
3.2. Obtaining Information to Contact a Floor Control Server	8
3.3. Obtaining Floor-Resource Associations	8
3.4. Privileges of Floor Control	9
4. Overview of Operation	9
4.1. Floor Participant to Floor Control Server Interface	10
4.2. Floor Chair to Floor Control Server Interface	14
5. Packet Format	15
5.1. COMMON-HEADER Format	15
5.2. Attribute Format	18
5.2.1. BENEFICIARY-ID	21
5.2.2. FLOOR-ID	21
5.2.3. FLOOR-REQUEST-ID	21
5.2.4. PRIORITY	22
5.2.5. REQUEST-STATUS	22
5.2.6. ERROR-CODE	23
5.2.6.1. Error-Specific Details for Error Code 4	24
5.2.7. ERROR-INFO	25
5.2.8. PARTICIPANT-PROVIDED-INFO	26
5.2.9. STATUS-INFO	26
5.2.10. SUPPORTED-ATTRIBUTES	27
5.2.11. SUPPORTED-PRIMITIVES	27
5.2.12. USER-DISPLAY-NAME	28
5.2.13. USER-URI	29
5.2.14. BENEFICIARY-INFORMATION	29
5.2.15. FLOOR-REQUEST-INFORMATION	30
5.2.16. REQUESTED-BY-INFORMATION	31
5.2.17. FLOOR-REQUEST-STATUS	31

5.2.18. OVERALL-REQUEST-STATUS	32
5.3. Message Format	33
5.3.1. FloorRequest	33
5.3.2. FloorRelease	33
5.3.3. FloorRequestQuery	33
5.3.4. FloorRequestStatus	34
5.3.5. UserQuery	34
5.3.6. UserStatus	34
5.3.7. FloorQuery	35
5.3.8. FloorStatus	35
5.3.9. ChairAction	35
5.3.10. ChairActionAck	35
5.3.11. Hello	36
5.3.12. HelloAck	36
5.3.13. Error	36
5.3.14. FloorRequestStatusAck	36
5.3.15. FloorStatusAck	37
5.3.16. Goodbye	37
5.3.17. GoodbyeAck	37
6. Transport	37
6.1. Reliable Transport	38
6.2. Unreliable Transport	39
6.2.1. Congestion Control	40
6.2.2. ICMP Error Handling	41
6.2.3. Fragmentation Handling	41
6.2.4. NAT Traversal	42
7. Lower-Layer Security	43
8. Protocol Transactions	43
8.1. Client Behavior	44
8.2. Server Behavior	44
8.3. Timers	45
8.3.1. Request Retransmission Timer, T1	45
8.3.2. Response Retransmission Timer, T2	45
8.3.3. Timer Values	46
9. Authentication and Authorization	46
9.1. TLS/DTLS Based Mutual Authentication	47
10. Floor Participant Operations	48
10.1. Requesting a Floor	48
10.1.1. Sending a FloorRequest Message	48
10.1.2. Receiving a Response	49
10.1.3. Reception of a Subsequent FloorRequestStatus Message	50
10.2. Cancelling a Floor Request and Releasing a Floor	50
10.2.1. Sending a FloorRelease Message	50
10.2.2. Receiving a Response	51
11. Chair Operations	51
11.1. Sending a ChairAction Message	51
11.2. Receiving a Response	53
12. General Client Operations	53

12.1.	Requesting Information about Floors	53
12.1.1.	Sending a FloorQuery Message	53
12.1.2.	Receiving a Response	54
12.1.3.	Reception of a Subsequent FloorStatus Message	55
12.2.	Requesting Information about Floor Requests	55
12.2.1.	Sending a FloorRequestQuery Message	55
12.2.2.	Receiving a Response	55
12.3.	Requesting Information about a User	56
12.3.1.	Sending a UserQuery Message	56
12.3.2.	Receiving a Response	57
12.4.	Obtaining the Capabilities of a Floor Control Server	57
12.4.1.	Sending a Hello Message	57
12.4.2.	Receiving Responses	57
13.	Floor Control Server Operations	58
13.1.	Reception of a FloorRequest Message	58
13.1.1.	Generating the First FloorRequestStatus Message	59
13.1.2.	Generation of Subsequent FloorRequestStatus Messages	60
13.2.	Reception of a FloorRequestQuery Message	61
13.3.	Reception of a UserQuery Message	63
13.4.	Reception of a FloorRelease Message	64
13.5.	Reception of a FloorQuery Message	65
13.5.1.	Generation of the First FloorStatus Message	66
13.5.2.	Generation of Subsequent FloorStatus Messages	67
13.6.	Reception of a ChairAction Message	68
13.7.	Reception of a Hello Message	69
13.8.	Error Message Generation	69
14.	Security Considerations	70
15.	IANA Considerations	71
15.1.	Attribute Subregistry	71
15.2.	Primitive Subregistry	72
15.3.	Request Status Subregistry	73
15.4.	Error Code Subregistry	74
16.	Changes from RFC 4582	75
16.1.	Extensions for an unreliable transport	75
16.2.	Other changes	76
17.	Acknowledgements	77
18.	References	77
18.1.	Normative References	78
18.2.	Informational References	79
Appendix A.	Example Call Flows for BFCP over an Unreliable Transport	81
Appendix B.	Motivation for Supporting an Unreliable Transport	85
B.1.	Motivation	85
B.1.1.	Alternatives Considered	87
B.1.1.1.	ICE TCP	87
B.1.1.2.	Teredo	87
B.1.1.3.	GUT	88
B.1.1.4.	UPnP IGD	88

B.1.1.5. NAT PMP	88
B.1.1.6. SCTP	89
B.1.1.7. BFCP over UDP transport	89
Authors' Addresses	89

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 [15] 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, and subsequent sections provide the normative specification of BFCP.

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, RFC 2119 [1] and indicate requirement levels for compliant implementations.

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 colocated 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 colocated.

The protocol between a floor participant and a media participant (that are not colocated) 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 exist, 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 colocated 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 colocated.

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 $T1 \cdot 2^4$

(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 [15]. Floor control complements other functions defined in the XCON conferencing framework [16]. 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 [16].

Figure 1 shows the tasks that BFCP can perform.

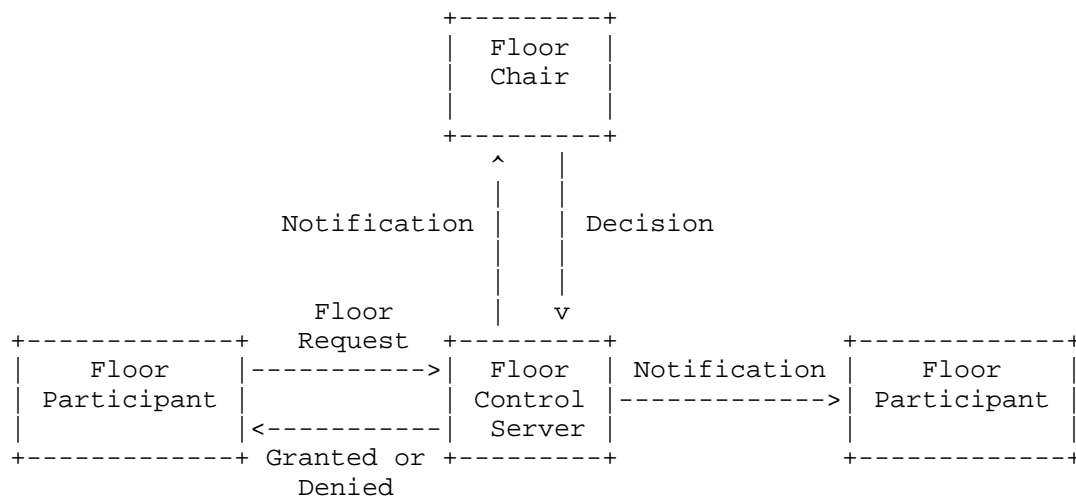


Figure 1: Functionality provided by BFCP

BFCP provides a means:

- o for floor participants to send floor requests to floor control servers.
- o for floor control servers to grant or deny requests to access a given resource from floor participants.
- o for floor chairs to send floor control servers decisions regarding floor requests.

- o 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 [16]. 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 CCMP [21] can specify such floor-related settings in the <floor-information> element [20] 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. This data includes 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 an SDP offer/answer [14] exchange, which is described in [10]. How to establish a connection to a BFCP floor control server 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 [16] (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 [14] exchange. How to use an SDP offer/answer exchange to obtain these associations is described in [10].

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 [16] (and other related documents). According to the conferencing system policies, conference control clients using CCMP [21] 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 [22].

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 [16].

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 colocated 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.

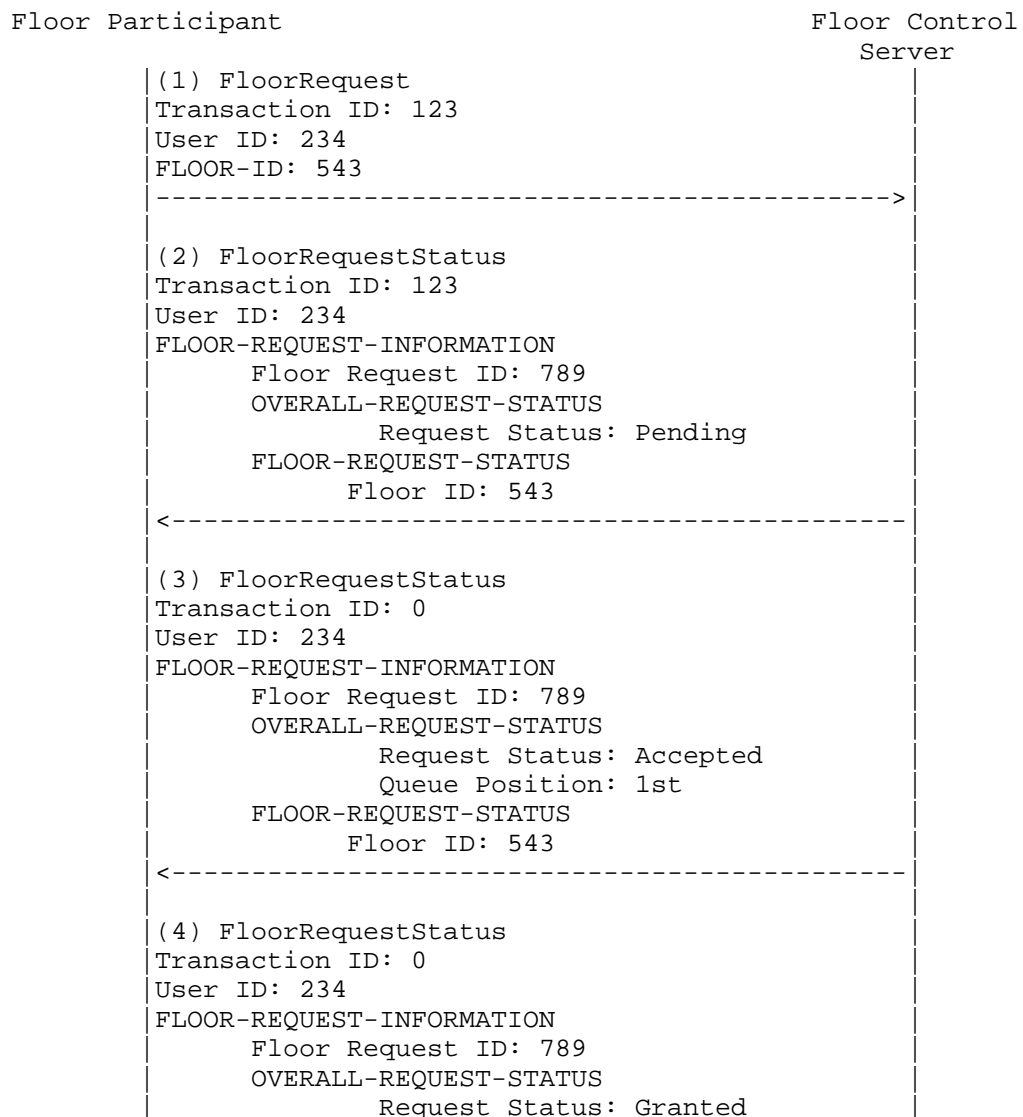
Floor control servers respond to FloorRequest messages with FloorRequestStatus messages, which provide information about the status of the floor request. The first FloorRequestStatus message is the response to the FloorRequest message from the client, and therefore has the same Transaction ID as the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

Figures 2 and 3 below 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.



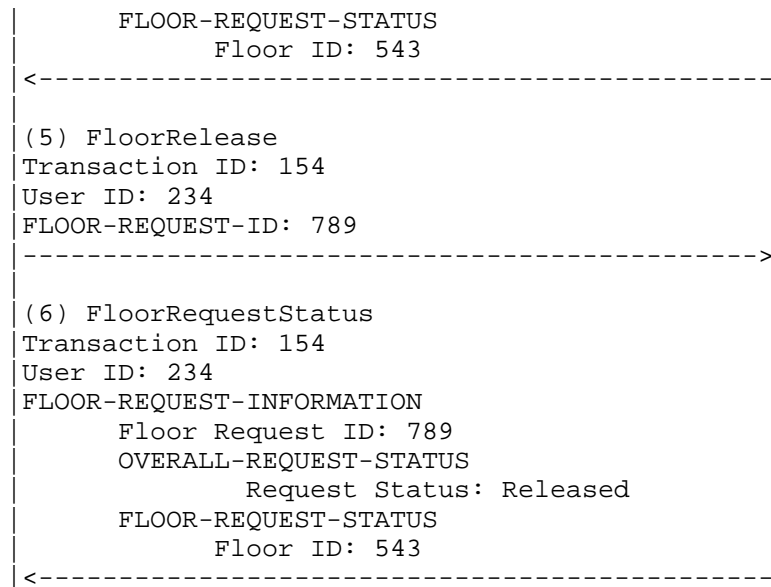
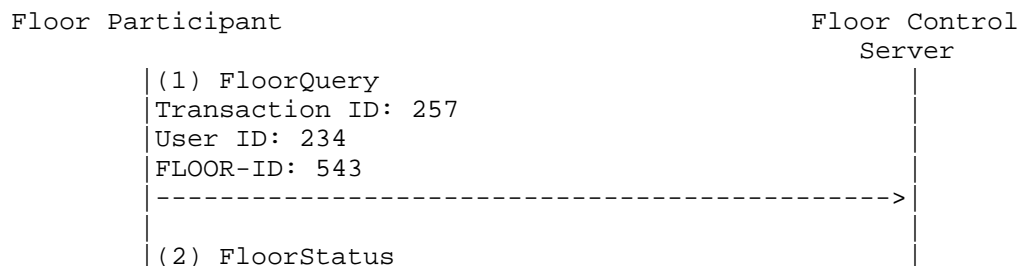


Figure 2: Requesting and releasing a floor

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.



```
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.

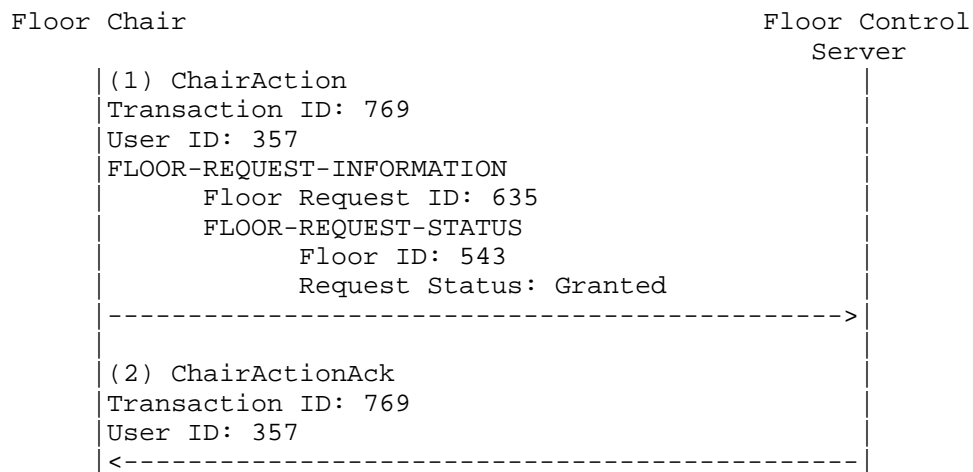


Figure 4: Chair instructing the floor control server

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.

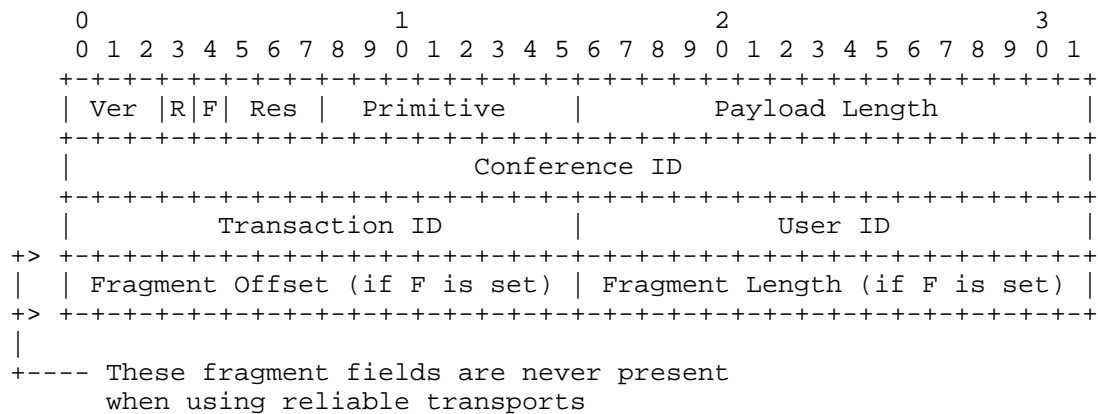


Figure 5: COMMON-HEADER format

Ver: This 3-bit field defines the version of BFCP that this message adheres to. 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 process the COMMON-HEADER as not having the Fragment Offset and Fragment Length fields present.

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
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 non-secure 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.

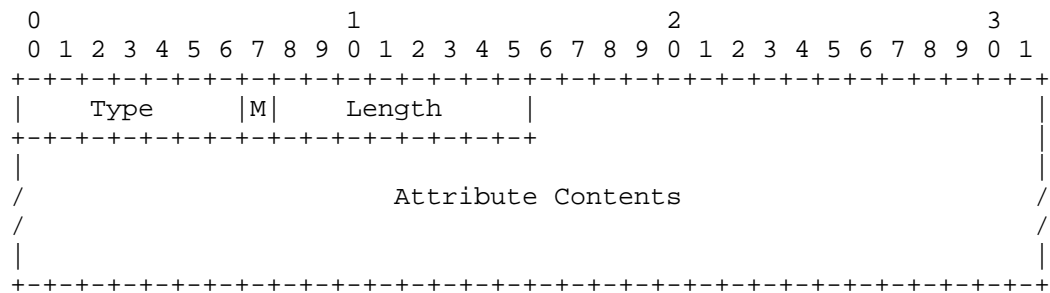


Figure 6: Attribute format

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.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|0 0 0 0 0 0 1|M|0 0 0 0 0 1 0 0|          Beneficiary ID          |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

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.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|0 0 0 0 0 1 0|M|0 0 0 0 0 1 0 0|          Floor ID          |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

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.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|0 0 0 0 0 1 1|M|0 0 0 0 0 1 0 0|    Floor Request ID          |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

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.

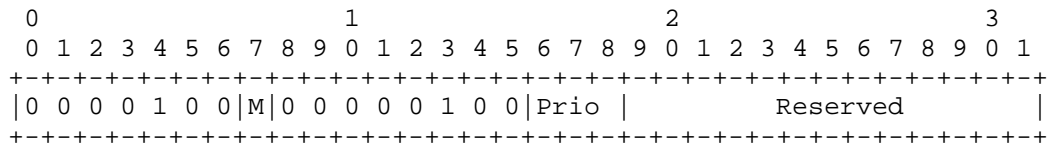


Figure 10: PRIORITY format

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
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.

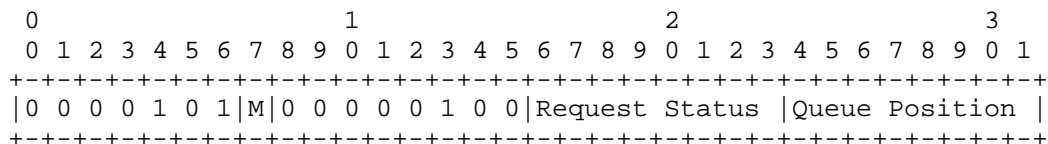


Figure 11: REQUEST-STATUS format

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.

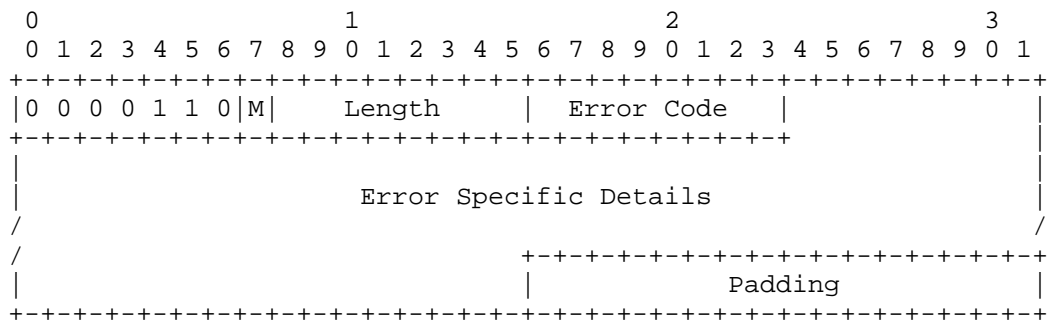


Figure 12: ERROR-CODE format

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
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, 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.

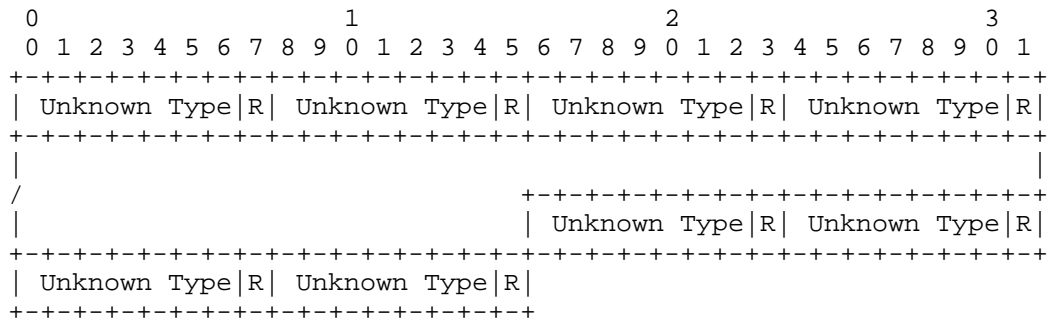


Figure 13: Unknown attributes format

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.

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.

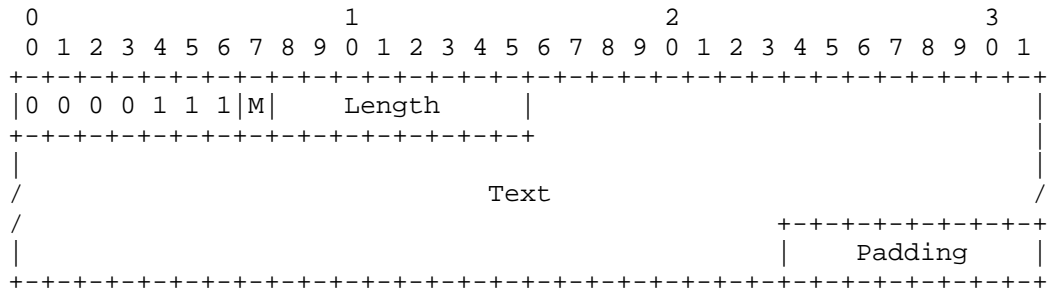


Figure 14: ERROR-INFO format

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

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.

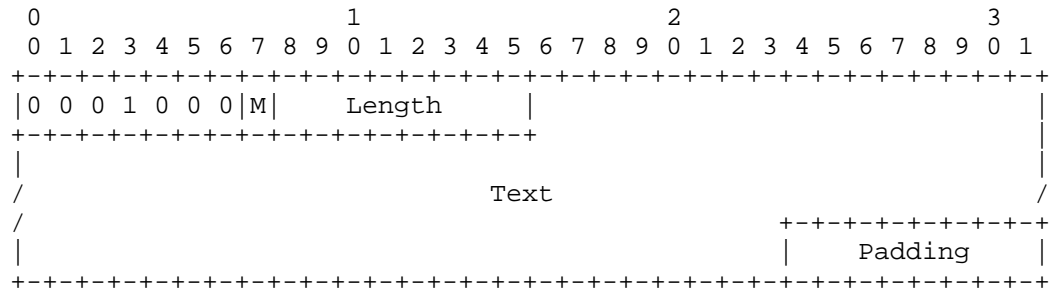


Figure 15: PARTICIPANT-PROVIDED-INFO format

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

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.

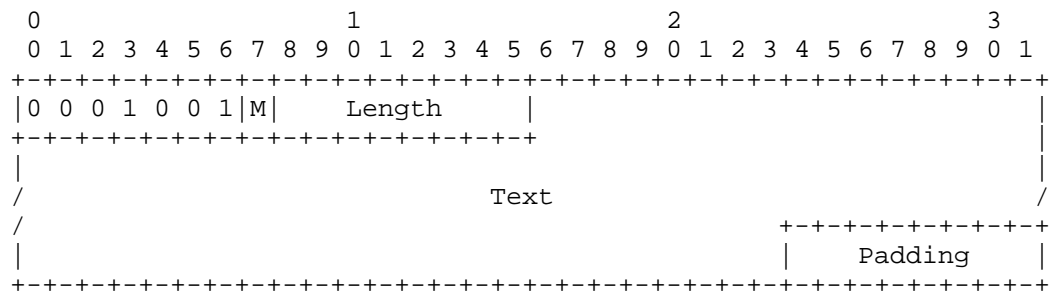


Figure 16: STATUS-INFO format

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

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.

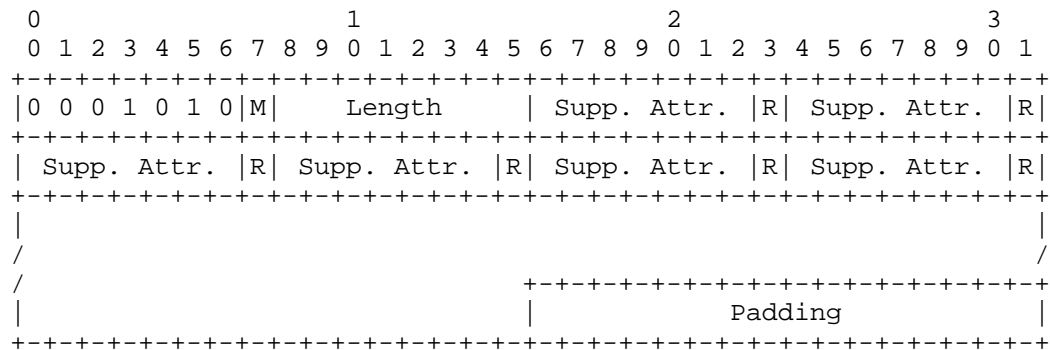


Figure 17: SUPPORTED-ATTRIBUTES format

Supp. Attr.: These fields contain the Types of the attributes that are supported by the floor control server in the following format:

R: Reserved: 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.

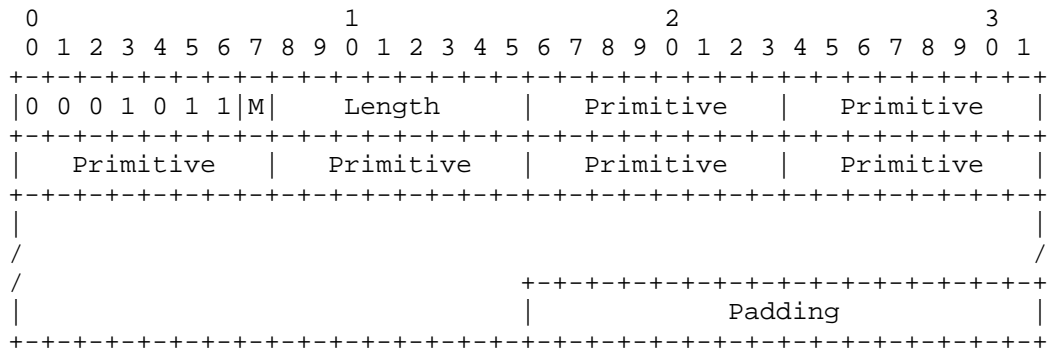


Figure 18: SUPPORTED-PRIMITIVES format

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.

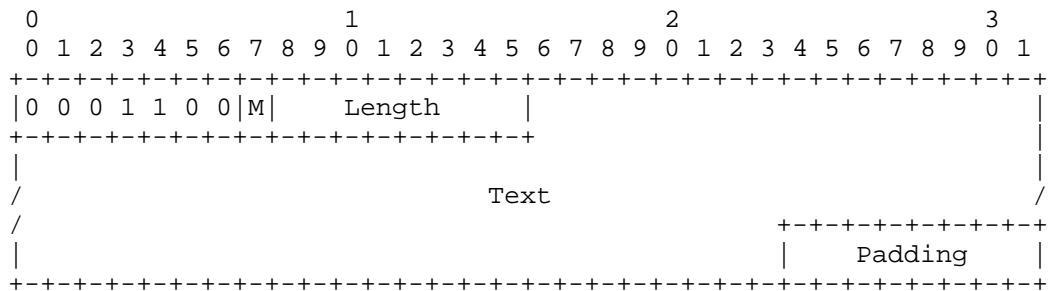


Figure 19: USER-DISPLAY-NAME format

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.

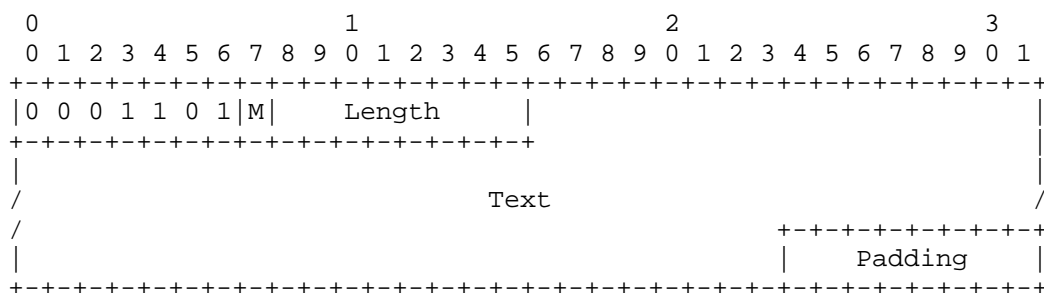


Figure 20: USER-URI format

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:

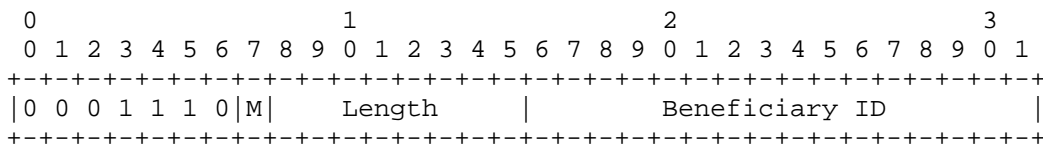


Figure 21: BENEFICIARY-INFORMATION-HEADER format

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:

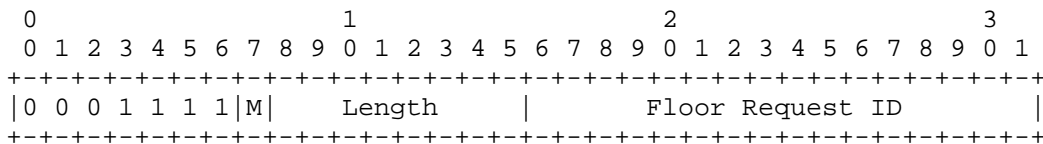


Figure 23: FLOOR-REQUEST-INFORMATION-HEADER format

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:

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0 0 1 0 0 0 0 |M|      Length      |      Requested-by ID      |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

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:

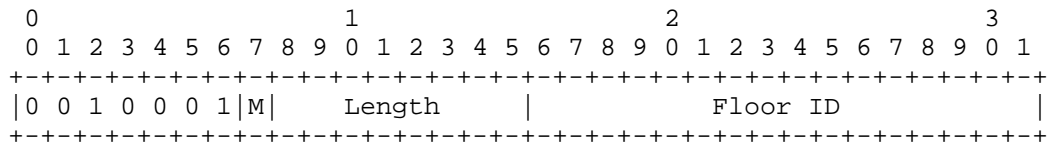


Figure 27: FLOOR-REQUEST-STATUS-HEADER format

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:

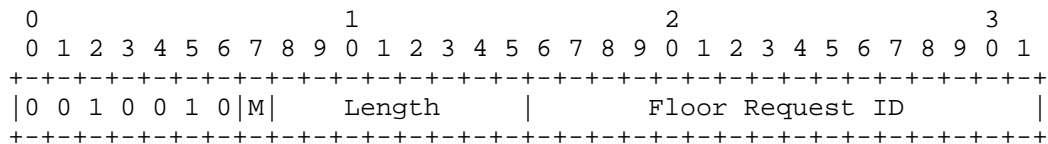


Figure 29: OVERALL-REQUEST-STATUS-HEADER format

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 liveliness 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 how to contact the floor control server, as described in Section 3.2. Two transports are supported: TCP, 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.

Informational note: In practice, products are configured to try one transport first and 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 from 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 service 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 back 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 or 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. A server-initiated request (e.g., a FloorStatus with an update from the floor control server) received by a participant before the initial FloorRequestStatus message that closes the client-initiated transaction that was instigated by the FloorRequest MUST be treated as superseding the information conveyed in any such late arriving response. 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 to generate "low data-volume" traffic, per the classification in [13]. Nevertheless it is necessary to ensure 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. In case 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 500ms, 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 [18], 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 being required 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 [23][24][25], for this purpose.

When transmitting a BFCP message with 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 $\text{ceil}((\text{message size} - \text{COMMON-HEADER size}) / (\text{path MTU size} - \text{COMMON-HEADER size}))$, 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 for 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 message fragment size 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 when 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 ICE [17].

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 [12] Binding Indication for keep-alives, as described for ICE [17]. Section 6.7 of [26] provides useful recommendations for middlebox interaction when DTLS is used.

Informational 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 straight forward to distinguish between STUN and BFCP packets even without checking the STUN magic cookie [12].

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/RTCP [11].

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 [7] 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 [28], BFCP entities SHOULD support the following cipher suites:

- o TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- o TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- o TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- o 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 0 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 against 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 have occurred. The timer is computed using the smoothed round-trip time algorithm defined in [2] with an initial retransmission timeout (RTO) value of 500ms and clock granularity (G) of 100ms. In contrast to step 2.4 of Section 2 of [2], if the computed value of RTO is less than 500ms, then RTO shall be set to 500ms. 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 point 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, whilst that value is retained until the timer has fired. Refer to Section 6.2.3 for the role this timer has 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	0.5s (initial)
T2	Response retransmission timer	$(T1 * 2^4) * 1.25$

Table 6: Timers

The initial value for T1 is 500ms, 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 [10].

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 name space 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.

10.1.1.2. 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.

13.1.1.1. 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 0. 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 BENEFCIARY-ID attribute or, in the absence of a BENEFCIARY-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 BENEFCIARY-ID attribute from the UserQuery message into a BENEFCIARY-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 BENEFCIARY-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 is IPsec [19] to effectively secure a non-secure BFCP connection.

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

[Note to IANA: Much of this text exists from the previous version of this document. While the old and new additions to the registries are presented here, the items for which IANA needs to take action with respect to this draft are highlighted with "Note to IANA", as with this note and the one immediately following. Throughout this document, though, RFC XXXX needs to be replaced with this RFC and the IANA registries for BFCP should to refer only to this RFC.]

[Note to IANA: This section instructs the IANA to register new entries in the BFCP Primitive subregistry in Section 15.2 and for the BFCP Error Code subregistry in Section 15.4.]

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

15.1. Attribute Subregistry

This section establishes the Attribute subregistry under the BFCP Parameters registry. As per the terminology in RFC 5226 [6], the registration policy for BFCP attributes shall be "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 XXXX]
2	FLOOR-ID	[RFC XXXX]
3	FLOOR-REQUEST-ID	[RFC XXXX]
4	PRIORITY	[RFC XXXX]
5	REQUEST-STATUS	[RFC XXXX]
6	ERROR-CODE	[RFC XXXX]
7	ERROR-INFO	[RFC XXXX]
8	PARTICIPANT-PROVIDED-INFO	[RFC XXXX]
9	STATUS-INFO	[RFC XXXX]
10	SUPPORTED-ATTRIBUTES	[RFC XXXX]
11	SUPPORTED-PRIMITIVES	[RFC XXXX]
12	USER-DISPLAY-NAME	[RFC XXXX]
13	USER-URI	[RFC XXXX]
14	BENEFICIARY-INFORMATION	[RFC XXXX]
15	FLOOR-REQUEST-INFORMATION	[RFC XXXX]
16	REQUESTED-BY-INFORMATION	[RFC XXXX]
17	FLOOR-REQUEST-STATUS	[RFC XXXX]
18	OVERALL-REQUEST-STATUS	[RFC XXXX]

Table 7: Initial values of the BFCP Attribute subregistry

15.2. Primitive Subregistry

[Note to IANA: This section instructs the IANA to register the following new values for the BFCP Primitive subregistry: FloorRequestStatusAck, FloorStatusAck, Goodbye, and GoodbyeAck.]

This section establishes the Primitive subregistry under the BFCP Parameters registry. As per the terminology in RFC 5226 [6], the registration policy for BFCP primitives shall be "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 XXXX]
2	FloorRelease	[RFC XXXX]
3	FloorRequestQuery	[RFC XXXX]
4	FloorRequestStatus	[RFC XXXX]
5	UserQuery	[RFC XXXX]
6	UserStatus	[RFC XXXX]
7	FloorQuery	[RFC XXXX]
8	FloorStatus	[RFC XXXX]
9	ChairAction	[RFC XXXX]
10	ChairActionAck	[RFC XXXX]
11	Hello	[RFC XXXX]
12	HelloAck	[RFC XXXX]
13	Error	[RFC XXXX]
14	FloorRequestStatusAck	[RFC XXXX]
15	FloorStatusAck	[RFC XXXX]
16	Goodbye	[RFC XXXX]
17	GoodbyeAck	[RFC XXXX]

Table 8: Initial values of the BFCP primitive subregistry

15.3. Request Status Subregistry

This section establishes the Request Status subregistry under the BFCP Parameters registry. As per the terminology in RFC 5226 [6], the registration policy for BFCP request status shall be "Specification Required". For the purposes of this subregistry, the BFCP request status 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 XXXX]
2	Accepted	[RFC XXXX]
3	Granted	[RFC XXXX]
4	Denied	[RFC XXXX]
5	Cancelled	[RFC XXXX]
6	Released	[RFC XXXX]
7	Revoked	[RFC XXXX]

Table 9: Initial values of the Request Status subregistry

15.4. Error Code Subregistry

[Note to IANA: This section instructs the IANA to register the following new values for the BFCP Error Code subregistry: 10, 11, 12, 13 and 14.]

This section establishes the Error Code subregistry under the BFCP Parameters registry. As per the terminology in RFC 5226 [6], the registration policy for BFCP error codes shall be "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 XXXX]
2	User does not Exist	[RFC XXXX]
3	Unknown Primitive	[RFC XXXX]
4	Unknown Mandatory Attribute	[RFC XXXX]
5	Unauthorized Operation	[RFC XXXX]
6	Invalid Floor ID	[RFC XXXX]
7	Floor Request ID Does Not Exist	[RFC XXXX]
8	You have Already Reached the Maximum Number of Ongoing Floor Requests for this Floor	[RFC XXXX]
9	Use TLS	[RFC XXXX]
10	Unable to parse message	[RFC XXXX]
11	Use DTLS	[RFC XXXX]
12	Unsupported Version	[RFC XXXX]
13	Incorrect Message Length	[RFC XXXX]
14	Generic Error	[RFC XXXX]

Table 10: Initial Values of the Error Code subregistry

16. Changes from RFC 4582

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

16.1. Extensions for an unreliable transport

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):
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. Where the specification for an unreliable transport amongst other issues deals with reliability, congestion control, fragmentation and ICMP.
7. Mandate DTLS (Section 7 and Section 9):
Mandate DTLS support when transport over UDP is used.
8. Transaction changes (Section 8):
Server-initiated transactions over an unreliable transport has non-zero and unique Transaction ID. Over an unreliable transport, the retransmit timers T1 and T2 described in Section 8.3 apply.
9. Requiring timely response (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):
Describing 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):
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, ="fig:reqby-information"/>, 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):
Change from SUPPORTED-PRIMITVIES to SUPPORTED-PRIMITIVES in the second paragraph.
 3. Corrected attribute type (Section 13.1.1):
Change from PARTICIPANT-PROVIDED-INFO to PRIORITY attributed 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. Assorted clarifications (Across the document):
Language clarifications as a result of reviews. Also, the normative language where tightened where appropriate, i.e. changed from SHOULD strength to MUST in a number of places.
17. Acknowledgements

The XCON WG 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. Heggstad, Trond G. Andersen, Gonzalo Camarillo, Roni Even, Lorenzo Miniero, Joerg 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.

18. References

18.1. 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, <<http://www.rfc-editor.org/info/rfc2119>>.
- [2] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, DOI 10.17487/RFC2988, November 2000, <<http://www.rfc-editor.org/info/rfc2988>>.
- [3] Camarillo, G., Ott, J., and K. Drage, "The Binary Floor Control Protocol (BFCP)", RFC 4582, DOI 10.17487/RFC4582, November 2006, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc5234>>.
- [6] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [7] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc3629>>.
- [10] Camarillo, G., Kristensen, T., and P. Jones, "Session Description Protocol (SDP) Format for Binary Floor Control Protocol (BFCP) Streams", draft-ietf-bfcpbis-rfc4583bis-12 (work in progress), September 2015.

- [11] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, DOI 10.17487/RFC4961, July 2007, <<http://www.rfc-editor.org/info/rfc4961>>.
- [12] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <<http://www.rfc-editor.org/info/rfc5389>>.
- [13] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, DOI 10.17487/RFC5405, November 2008, <<http://www.rfc-editor.org/info/rfc5405>>.

18.2. Informational References

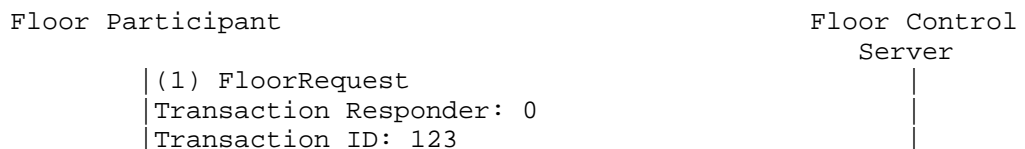
- [14] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <<http://www.rfc-editor.org/info/rfc3264>>.
- [15] Koskelainen, P., Ott, J., Schulzrinne, H., and X. Wu, "Requirements for Floor Control Protocols", RFC 4376, DOI 10.17487/RFC4376, February 2006, <<http://www.rfc-editor.org/info/rfc4376>>.
- [16] Barnes, M., Boulton, C., and O. Levin, "A Framework for Centralized Conferencing", RFC 5239, DOI 10.17487/RFC5239, June 2008, <<http://www.rfc-editor.org/info/rfc5239>>.
- [17] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<http://www.rfc-editor.org/info/rfc5245>>.
- [18] 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, <<http://www.rfc-editor.org/info/rfc3261>>.
- [19] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<http://www.rfc-editor.org/info/rfc4301>>.

- [20] 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, <<http://www.rfc-editor.org/info/rfc6501>>.
- [21] Barnes, M., Boulton, C., Romano, S., and H. Schulzrinne, "Centralized Conferencing Manipulation Protocol", RFC 6503, DOI 10.17487/RFC6503, March 2012, <<http://www.rfc-editor.org/info/rfc6503>>.
- [22] Barnes, M., Miniero, L., Presta, R., and S. Romano, "Centralized Conferencing Manipulation Protocol (CCMP) Call Flow Examples", RFC 6504, DOI 10.17487/RFC6504, March 2012, <<http://www.rfc-editor.org/info/rfc6504>>.
- [23] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <<http://www.rfc-editor.org/info/rfc1191>>.
- [24] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <<http://www.rfc-editor.org/info/rfc1981>>.
- [25] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <<http://www.rfc-editor.org/info/rfc4821>>.
- [26] 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, <<http://www.rfc-editor.org/info/rfc5763>>.
- [27] 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, <<http://www.rfc-editor.org/info/rfc6951>>.
- [28] 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, <<http://www.rfc-editor.org/info/rfc7525>>.

- [29] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, DOI 10.17487/RFC4380, February 2006, <<http://www.rfc-editor.org/info/rfc4380>>.
- [30] Thaler, D., "Teredo Extensions", RFC 6081, DOI 10.17487/RFC6081, January 2011, <<http://www.rfc-editor.org/info/rfc6081>>.
- [31] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<http://www.rfc-editor.org/info/rfc4960>>.
- [32] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, DOI 10.17487/RFC6544, March 2012, <<http://www.rfc-editor.org/info/rfc6544>>.
- [33] Manner, J., Varis, N., and B. Briscoe, "Generic UDP Tunnelling (GUT)", draft-manner-tsvwg-gut-02 (work in progress), July 2010.
- [34] Stucker, B., Tschofenig, H., and G. Salgueiro, "Analysis of Middlebox Interactions for Signaling Protocol Communication along the Media Path", draft-ietf-mmusic-media-path-middleboxes-07 (work in progress), May 2013.
- [35] Guha, S. and P. Francis, "Characterization and Measurement of TCP Traversal through NATs and Firewalls", 2005, <<http://saikat.guha.cc/pub/imc05-tcpnat.pdf/>>.
- [36] Ford, B., Srisuresh, P., and D. Kegel, "Peer-to-Peer Communication Across Network Address Translators", April 2005, <<http://www.brynosaurus.com/pub/net/p2pnat.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 loss-less interaction.



```
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
```

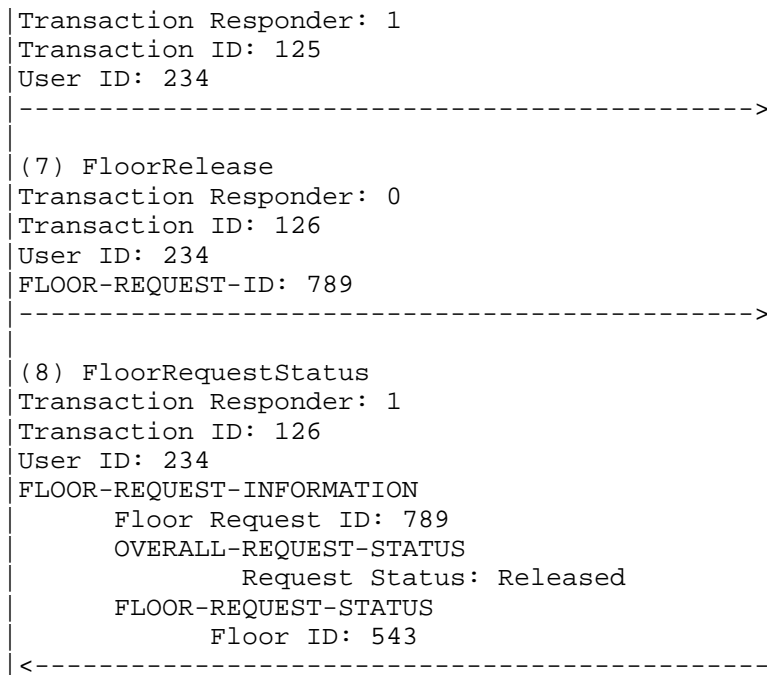
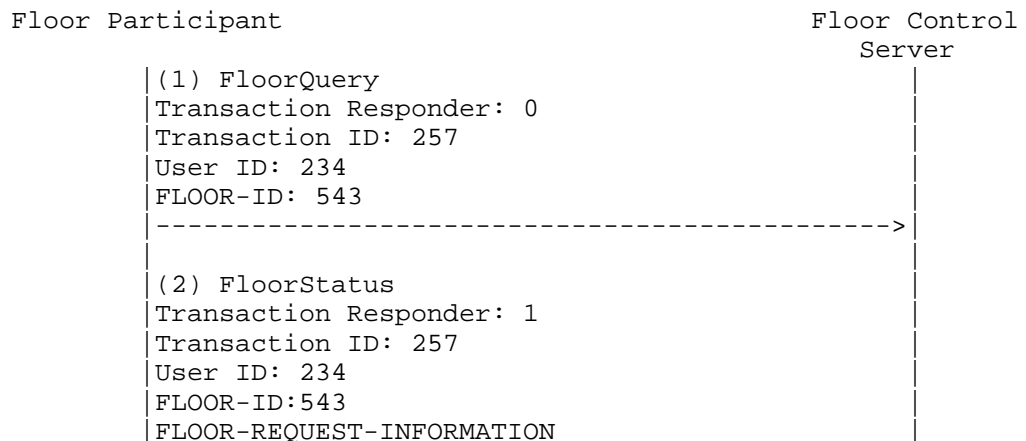


Figure 48: Requesting and releasing a floor

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 154. As such, it is not followed by a FloorRequestStatusAck message from the floor participant to the floor control server.



```
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
```

```
<-----
(4) FloorStatusAck
Transaction Responder: 1
Transaction ID: 258
User ID: 234
----->
```

```

(5) FloorStatus
Transaction Responder: 0
Transaction ID: 259
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
<-----
(6) FloorStatusAck
Transaction Responder: 1
Transaction ID: 259
User ID: 234
----->

```

Figure 49: Obtaining status information about a floor

Appendix B. Motivation for Supporting an Unreliable Transport

This appendix is contained in this document 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 exists behind a NAT. BFCP roles are negotiated in the offer/answer exchange as specified in [10], resulting in one endpoint being responsible for opening the TCP connection used for the BFCP communication.

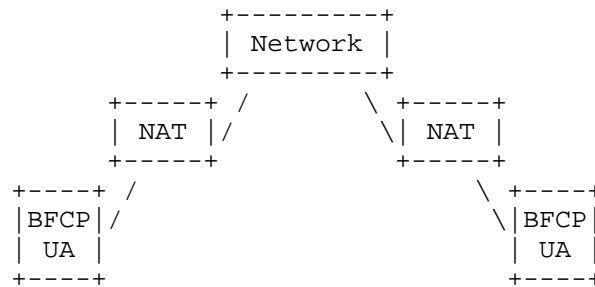


Figure 50: Use Case

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 [34].

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., 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, 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 [32]. 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 it was not possible to establish a TCP connection in 11% of the cases [35]. 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 [36].

It is worth noticing 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 co-exist in the market. The purpose of this draft is to formalize and publish the extension from the standard specification to facilitate complete interoperability between implementations.

B.1.1.1. 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 not chosen alternatives work in a number of scenarios, they are not sufficient, in and of themselves, to address the use case targeted by this draft. The last alternative, presented in Appendix B.1.1.7, is the selected one and is specified in this draft.

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

B.1.1.1.1. ICE TCP

ICE TCP [32] 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 [32]) than enabling UDP connectivity in the same scenarios. This happens because many of the currently deployed NATs in video conferencing networks do not support the flow of TCP handshake packets seen in 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 to recently, or because they do not properly process the subsequent SYNACK. Implementing various techniques advocated for candidate collection in [32] 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.1.1.1.2. Teredo

Teredo [29] enables nodes located behind one or more IPv4 NATs to obtain IPv6 connectivity by tunneling packets over UDP. Teredo extensions [30] 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 draft. However, running the service requires the help of "Teredo servers" and "Teredo relays" [29]. These servers and relays generally do not exist in the existing 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 transition 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 [33] 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 draft has expired and its future is still not clear 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 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 straight forward to specify a BFCP binding for Sctp [31], 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 WG regarding this approach, which resulted in [27]. However, this approach for tunneling over UDP was not mature enough when considered and 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, it is recommended to follow the guidelines provided in [13].

Minor changes to the transaction model are introduced in that all requests now have an appropriate response to complete the transaction. The requests are sent with a retransmit 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 versions of this draft, have demonstrated this approach to be 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.

Authors' Addresses

Gonzalo Camarillo
Ericsson
Hirsalantie 11
FI-02420 Jorvas
Finland

Email: gonzalo.camarillo@ericsson.com

Keith Drage
Alcatel-Lucent
Quadrant, StoneHill Green, Westlea
Swindon, Wilts
UK

Email: drage@alcatel-lucent.com

Tom Kristensen
Cisco
Philip Pedersens vei 1
NO-1366 Lysaker
Norway

Email: tomkrist@cisco.com, tomkri@ifi.uio.no

Joerg Ott
Aalto University
Otakaari 5 A
FI-02150 Espoo
Finland

Email: jo@comnet.tkk.fi

Charles Eckel
Cisco
707 Tasman Drive
California, CA 95035
United States

Email: eckelcu@cisco.com

BFCPbis Working Group
Internet-Draft
Obsoletes: 4583 (if approved)
Intended status: Standards Track
Expires: August 19, 2014

G. Camarillo
Ericsson
T. Kristensen
Cisco
February 15, 2014

Session Description Protocol (SDP) Format for Binary Floor Control
Protocol (BFCP) Streams
draft-ietf-bfcpbis-rfc4583bis-09

Abstract

This document specifies how to describe Binary Floor Control Protocol (BFCP) streams in Session Description Protocol (SDP) descriptions. User agents using the offer/answer model to establish BFCP streams use this format in their offers and answers.

This document obsoletes RFC 4583. Changes from RFC 4583 are summarized in Section 12.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 19, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Fields in the 'm' Line	3
4. Floor Control Server Determination	4
5. The 'confid' and 'userid' SDP Attributes	6
6. Association between Streams and Floors	6
7. BFCP Version Negotiation	7
8. BFCP Connection Management	8
8.1. TCP Connection Management	8
9. Authentication	8
10. Examples	9
11. Security Considerations	11
12. IANA Considerations	12
12.1. Registration of SDP 'proto' Values	12
12.2. Registration of the SDP 'floorctrl' Attribute	12
12.3. Registration of the SDP 'confid' Attribute	13
12.4. Registration of the SDP 'userid' Attribute	13
12.5. Registration of the SDP 'floorid' Attribute	13
12.6. Registration of the SDP 'bfcpsver' Attribute	14
13. Changes from RFC 4583	14
14. Acknowledgements	15
15. Normative References	15
Authors' Addresses	17

1. Introduction

As discussed in the BFCP (Binary Floor Control Protocol) specification [8], a given BFCP client needs a set of data in order to establish a BFCP connection to a floor control server. This data include the transport address of the server, the conference identifier, and the user identifier.

One way for clients to obtain this information is to use an offer/answer [4] exchange. This document specifies how to encode this information in the SDP session descriptions that are part of such an offer/answer exchange.

User agents typically use the offer/answer model to establish a number of media streams of different types. Following this model, a BFCP connection is described as any other media stream by using an SDP 'm' line, possibly followed by a number of attributes encoded in 'a' lines.

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, RFC 2119 [1] and indicate requirement levels for compliant implementations.

3. Fields in the 'm' Line

This section describes how to generate an 'm' line for a BFCP stream.

According to the SDP specification [11], the 'm' line format is the following:

```
m=<media> <port> <proto> <fmt> ...
```

The media field MUST have a value of "application".

The port field is set depending on the value of the proto field, as explained below. A port field value of zero has the standard SDP meaning (i.e., rejection of the media stream) regardless of the proto field.

When TCP is used as the transport, the port field is set following the rules in [7]. Depending on the value of the 'setup' attribute (discussed in Section 8.1), the port field contains the port to which the remote endpoint will direct BFCP messages or is irrelevant (i.e., the endpoint will initiate the connection towards the remote endpoint) and should be set to a value of 9, which is the discard port.

When UDP is used as the transport, the port field contains the port to which the remote endpoint will direct BFCP messages regardless of the value of the 'setup' attribute.

This document defines four values for the proto field: TCP/BFCP, TCP/TLS/BFCP, UDP/BFCP, and UDP/TLS/BFCP. TCP/BFCP is used when BFCP runs directly on top of TCP, TCP/TLS/BFCP is used when BFCP runs on top of TLS, which in turn runs on top of TCP. Similarly, UDP/BFCP is used when BFCP runs directly on top of UDP, and UDP/TLS/BFCP is used when BFCP runs on top of DTLS [12], which in turn runs on top of UDP.

The fmt (format) list is not applicable to BFCP. The fmt list of 'm' lines in the case of any proto field value related to BFCP SHOULD contain a single "*" character. If the the fmt list contains any other value it is ignored.

The following is an example of an 'm' line for a BFCP connection:

```
m=application 50000 TCP/TLS/BFCP *
```

4. Floor Control Server Determination

When two endpoints establish a BFCP stream, they need to determine which of them acts as a floor control server. In the most common scenario, a client establishes a BFCP stream with a conference server that acts as the floor control server. Floor control server determination is straight forward because one endpoint can only act as a client and the other can only act as a floor control server.

However, there are scenarios where both endpoints could act as a floor control server. For example, in a two-party session that involves an audio stream and a shared whiteboard, the endpoints need to decide which party will be acting as the floor control server.

Furthermore, there are situations where both the offerer and the answerer act as both clients and floor control servers in the same session. For example, in a two-party session that involves an audio stream and a shared whiteboard, one party acts as the floor control server for the audio stream and the other acts as the floor control

server for the shared whiteboard.

This document defines the 'floorctrl' SDP media-level attribute to perform floor control determination. Its Augmented BNF syntax [2] is:

```
floor-control-attribute = "a=floorctrl:" role *(SP role)
role                    = "c-only" / "s-only" / "c-s"
```

The offerer includes this attribute to state all the roles it would be willing to perform:

c-only: The offerer would be willing to act as a floor control client only.

s-only: The offerer would be willing to act as a floor control server only.

c-s: The offerer would be willing to act both as a floor control client and as a floor control server.

If an SDP media description in an offer contains a 'floorctrl' attribute, the answerer accepting that media MUST include a 'floorctrl' attribute in the corresponding media description of the answer. The answerer includes this attribute to state which role the answerer will perform. That is, the answerer chooses one of the roles the offerer is willing to perform and generates an answer with the corresponding role for the answerer. Table 1 shows the corresponding roles for an answerer, depending on the offerer's role.

Offerer	Answerer
c-only	s-only
s-only	c-only
c-s	c-s

Table 1: Roles

The following are the descriptions of the roles when they are chosen by an answerer:

c-only: The answerer will act as a floor control client.
Consequently, the offerer will act as a floor control server.

s-only: The answerer will act as a floor control server.
Consequently, the offerer will act as a floor control client.

c-s: The answerer will act both as a floor control client and as a floor control server. Consequently, the offerer will also act both as a floor control client and as a floor control server.

Endpoints that use the offer/answer model to establish BFCP connections MUST support the 'floorctrl' attribute. A floor control server acting as an offerer or as an answerer SHOULD include this attribute in its session descriptions.

If the 'floorctrl' attribute is not used in an offer/answer exchange, by default the offerer and the answerer will act as a floor control client and as a floor control server, respectively.

The following is an example of a 'floorctrl' attribute in an offer. When this attribute appears in an answer, it only carries one role:

```
a=floorctrl:c-only s-only c-s
```

5. The 'confid' and 'userid' SDP Attributes

This document defines the 'confid' and the 'userid' SDP media-level attributes. These attributes are used by a floor control server to provide a client with a conference ID and a user ID, respectively. Their Augmented BNF syntax [2] is:

```
confid-attribute      = "a=confid:" conference-id
conference-id         = token
userid-attribute      = "a=userid:" user-id
user-id              = token
```

The 'confid' and the 'userid' attributes carry the decimal integer representation of a conference ID and a user ID, respectively.

Endpoints that use the offer/answer model to establish BFCP connections MUST support the 'confid' and the 'userid' attributes. A floor control server acting as an offerer or as an answerer MUST include these attributes in its session descriptions.

6. Association between Streams and Floors

This document defines the 'floorid' SDP media-level attribute. Its Augmented BNF syntax [2] is:

```
floor-id-attribute = "a=floorid:" token [" mstrm:" token *(SP token)]
```

The 'floorid' attribute is used in the SDP media description for BFCP media. It defines a floor identifier and, possibly, associates it with one or more media streams. The token representing the floor ID is the integer representation of the Floor ID to be used in BFCP. The token representing the media stream is a pointer to the media stream, which is identified by an SDP label attribute [9].

Endpoints that use the offer/answer model to establish BFCP connections **MUST** support the 'floorid' and the 'label' attributes. A floor control server acting as an offerer or as an answerer **MUST** include these attributes in its session descriptions.

Note: In [15] 'm-stream' was erroneously used in Section 10. Although the example was non-normative, it is implemented by some vendors and occurs in cases where the endpoint is willing to act as an server. Therefore, it is **RECOMMENDED** to support parsing and interpreting 'm-stream' the same way as 'mstrm' when receiving.

7. BFCP Version Negotiation

This document defines the 'bfcvver' SDP media-level attribute. Its Augmented BNF syntax [2] is:

```
bfcvver-attribute = "a=bfcvver:" bfcvver *(SP bfcvver)
bfcvver           = token
```

The 'bfcvver' attribute defines the list of the versions of BFCP supported by the endpoint. Tokens representing versions **MUST** be integers matching the "Version" field that would be presented in the BFCP COMMON-HEADER [8]. The version of BFCP to be used will then be confirmed with a BFCP-level Hello/HelloAck.

Endpoints that use the offer/answer model to establish BFCP connections **SHOULD** support the 'bfcvver' attribute. A floor control server acting as an offerer or as an answerer **SHOULD** include this attribute in its session descriptions. However, endpoints that support RFCXXXX, and not only the RFC 4583 subset, are **REQUIRED** to support and, when acting as a floor control server to use the 'bfcvver' attribute.

If a 'bfcvver' attribute is not present, default values are inferred from the transport specified in the m-line (Section 3). In accordance with definition of the Version field in [8], when used over a reliable transport the default value is "1", and when used over an unreliable transport the default value is "2".

8. BFCP Connection Management

BFCP connections can use TCP or UDP as the underlying transport. BFCP entities exchanging BFCP messages over UDP will direct the BFCP messages to the peer side connection address and port provided in the SDP 'm' line. TCP connection management is more complicated and is described below.

8.1. TCP Connection Management

The management of the TCP connection used to transport BFCP is performed using the 'setup' and 'connection' attributes, as defined in [7].

The 'setup' attribute indicates which of the endpoints (client or floor control server) initiates the TCP connection. The 'connection' attribute handles TCP connection reestablishment.

The BFCP specification [8] describes a number of situations when the TCP connection between a client and the floor control server needs to be reestablished. However, that specification does not describe the reestablishment process because this process depends on how the connection was established in the first place. BFCP entities using the offer/answer model follow the following rules.

When the existing TCP connection is reset following the rules in [8], the client MUST generate an offer towards the floor control server in order to reestablish the connection. If a TCP connection cannot deliver a BFCP message and times out, the entity that attempted to send the message (i.e., the one that detected the TCP timeout) MUST generate an offer in order to reestablish the TCP connection.

Endpoints that use the offer/answer model to establish TCP connections MUST support the 'setup' and 'connection' attributes.

9. Authentication

When a BFCP connection is established using the offer/answer model, it is assumed that the offerer and the answerer authenticate each other using some mechanism. TLS/DTLS is the preferred mechanism, but other mechanisms are possible and outside the scope of this document. Once this mutual authentication takes place, all the offerer and the answerer need to ensure is that the entity they are receiving BFCP messages from is the same as the one that generated the previous offer or answer.

When SIP is used to perform an offer/answer exchange, the initial

mutual authentication takes place at the SIP level. Additionally, SIP uses S/MIME [6] to provide an integrity-protected channel with optional confidentiality for the offer/answer exchange. BFCP takes advantage of this integrity-protected offer/answer exchange to perform authentication. Within the offer/answer exchange, the offerer and answerer exchange the fingerprints of their self-signed certificates. These self-signed certificates are then used to establish the TLS/DTLS connection that will carry BFCP traffic between the offerer and the answerer.

BFCP clients and floor control servers follow the rules in [10] regarding certificate choice and presentation. This implies that unless a 'fingerprint' attribute is included in the session description, the certificate provided at the TLS-/DTLS-level MUST either be directly signed by one of the other party's trust anchors or be validated using a certification path that terminates at one of the other party's trust anchors [5]. Endpoints that use the offer/answer model to establish BFCP connections MUST support the 'fingerprint' attribute and MUST include it in their session descriptions.

When TLS is used with TCP, once the underlying connection is established, the answerer acts as the TLS server regardless of its role (passive or active) in the TCP establishment procedure.

Endpoints that use the offer/answer model to establish a DTLS association MUST support the 'setup' attribute, as defined in [7]. When DTLS is used with UDP, the 'setup' attribute indicates which of the endpoints (client or floor control server) initiates the DTLS association setup. The requirements for the offer/answer exchange specified in [13], Section 5 MUST be followed when using DTLS.

Informational note: How to determine which endpoint to initiate the TLS/DTLS association depends on the selected underlying transport. It was decided to keep the original semantics in [15] for TCP to retain backwards compatibility. When using UDP, the procedure above was preferred since it adheres to [13] as used for DTLS-SRTP, it does not overload offer/answer semantics, and it works for offerless INVITE in scenarios with B2BUAs.

10. Examples

For the purpose of brevity, the main portion of the session description is omitted in the examples, which only show 'm' lines and their attributes.

The following is an example of an offer sent by a conference server

to a client.

```
m=application 50000 TCP/TLS/BFCP *
a=setup:passive
a=connection:new
a=fingerprint:SHA-1 \
    4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB
a=floorctrl:s-only
a=confid:4321
a=userid:1234
a=floorid:1 mstrm:10
a=floorid:2 mstrm:11
a=bfcper:1
m=audio 50002 RTP/AVP 0
a=label:10
m=video 50004 RTP/AVP 31
a=label:11
```

Note that due to RFC formatting conventions, this document splits SDP across lines whose content would exceed 72 characters. A backslash character marks where this line folding has taken place. This backslash and its trailing CRLF and whitespace would not appear in actual SDP content.

The following is the answer returned by the client.

```
m=application 9 TCP/TLS/BFCP *
a=setup:active
a=connection:new
a=fingerprint:SHA-1 \
    3D:B4:7B:E3:CC:FC:0D:1B:5D:31:33:9E:48:9B:67:FE:68:40:E8:21
a=floorctrl:c-only
a=bfcper:1
m=audio 55000 RTP/AVP 0
m=video 55002 RTP/AVP 31
```

A similar example using unreliable transport and DTLS is shown below, where the offer is sent from a client.

```
m=application 50000 UDP/TLS/BFCP *
a=setup:actpass
a=fingerprint:SHA-1 \
    4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB
a=floorctrl:c-only s-only
a=confid:4321
a=userid:1234
a=floorid:1 mstrm:10
a=floorid:2 mstrm:11
a=bfcvver:2
m=audio 50002 RTP/AVP 0
a=label:10
m=video 50004 RTP/AVP 31
a=label:11
```

The following is the answer returned by the server.

```
m=application 55000 UDP/TLS/BFCP *
a=setup:active
a=fingerprint:SHA-1 \
    3D:B4:7B:E3:CC:FC:0D:1B:5D:31:33:9E:48:9B:67:FE:68:40:E8:21
a=floorctrl:s-only
a=confid:4321
a=userid:1234
a=floorid:1 mstrm:10
a=floorid:2 mstrm:11
a=bfcvver:2
m=audio 55002 RTP/AVP 0
m=video 55004 RTP/AVP 31
```

11. Security Considerations

The BFCP [8], SDP [11], and offer/answer [4] specifications discuss security issues related to BFCP, SDP, and offer/answer, respectively. In addition, [7] and [10] discuss security issues related to the establishment of TCP and TLS connections using an offer/answer model. Furthermore, when using DTLS over UDP, considerations for its use with RTP and RTCP are presented in [13]. The requirements for the offer/answer exchange, as listed in Section 5 of that document, MUST be followed.

An initial integrity-protected channel is REQUIRED for BFCP to exchange self-signed certificates between a client and the floor control server. For session descriptions carried in SIP [3], S/MIME [6] is the natural choice to provide such a channel.

12. IANA Considerations

[Editorial note: The changes in Section 12.1 instruct the IANA to register the two new values UDP/BFCP and UDP/TLS/BFCP for the SDP 'proto' field. The new section Section 12.6 registers a new SDP "bfcper" attribute. The rest is unchanged from [14].]

12.1. Registration of SDP 'proto' Values

The IANA has registered the following values for the SDP 'proto' field under the Session Description Protocol (SDP) Parameters registry:

Value	Reference
TCP/BFCP	[RFC XXXX]
TCP/TLS/BFCP	[RFC XXXX]
UDP/BFCP	[RFC XXXX]
UDP/TLS/BFCP	[RFC XXXX]

Table 2: Values for the SDP 'proto' field

12.2. Registration of the SDP 'floorctrl' Attribute

The IANA has registered the following SDP att-field under the Session Description Protocol (SDP) Parameters registry:

Contact name: Gonzalo.Camarillo@ericsson.com

Attribute name: floorctrl

Long-form attribute name: Floor Control

Type of attribute: Media level

Subject to charset: No

Purpose of attribute: The 'floorctrl' attribute is used to perform floor control server determination.

Allowed attribute values: 1*("c-only" / "s-only" / "c-s")

12.3. Registration of the SDP 'confid' Attribute

The IANA has registered the following SDP att-field under the Session Description Protocol (SDP) Parameters registry:

Contact name: Gonzalo.Camarillo@ericsson.com

Attribute name: confid

Long-form attribute name: Conference Identifier

Type of attribute: Media level

Subject to charset: No

Purpose of attribute: The 'confid' attribute carries the integer representation of a Conference ID.

Allowed attribute values: A token

12.4. Registration of the SDP 'userid' Attribute

The IANA has registered the following SDP att-field under the Session Description Protocol (SDP) Parameters registry:

Contact name: Gonzalo.Camarillo@ericsson.com

Attribute name: userid

Long-form attribute name: User Identifier

Type of attribute: Media level

Subject to charset: No

Purpose of attribute: The 'userid' attribute carries the integer representation of a User ID.

Allowed attribute values: A token

12.5. Registration of the SDP 'floorid' Attribute

The IANA has registered the following SDP att-field under the Session Description Protocol (SDP) Parameters registry:

Contact name: Gonzalo.Camarillo@ericsson.com

Attribute name: floorid

Long-form attribute name: Floor Identifier

Type of attribute: Media level

Subject to charset: No

Purpose of attribute: The 'floorid' attribute associates a floor with one or more media streams.

Allowed attribute values: Tokens

12.6. Registration of the SDP 'bfcvver' Attribute

The IANA has registered the following SDP att-field under the Session Description Protocol (SDP) Parameters registry:

Contact name: Gonzalo.Camarillo@ericsson.com

Attribute name: bfcvver

Long-form attribute name: BFCP Version

Type of attribute: Media level

Subject to charset: No

Purpose of attribute: The 'bfcvver' attribute lists supported BFCP versions.

Allowed attribute values: Tokens

13. Changes from RFC 4583

Following is the list of technical changes and other fixes from [15].

Main purpose of this work was to add signaling support necessary to support BFCP over unreliable transport, as described in [8], resulting in the following changes:

1. Fields in the 'm' Line (Section 3):

The section is re-written to remove reference to the exclusivity of TCP as a transport for BFCP streams. The proto field values UDP/BFCP and UDP/TLS/BFCP added.

2. Authentication (Section 9):
In last paragraph, made clear that a TCP connection was described.
3. Security Considerations (Section 11):
For the DTLS over UDP case, mention existing considerations and requirements for the offer/answer exchange in [13].
4. Registration of SDP 'proto' Values (Section 12.1):
Register the two new values UDP/BFCP and UDP/TLS/BFCP in the SDP parameters registry.
5. BFCP Version Negotiation (Section 7):
A new 'bfcv' SDP media-level attribute is added in order to signal supported version number.

The clarification and bug fixes:

1. Errata ID: 712 (Section 4 and Section 6):
Language clarification. Don't use terms like an SDP attribute is "used in an 'm' line", instead make clear that the attribute is a media-level attribute.
2. Fix typo in example (Section 10):
Do not use 'm-stream' in the SDP example, use the correct 'mstrm' as specified in Section 10. Recommend interpreting 'm-stream' if it is received, since it is present in some implementations.
3. Assorted clarifications (Across the document):
Non-functional language clarifications and some corrections in the normative language as a result of reviews.

14. Acknowledgements

Joerg Ott, Keith Drage, Alan Johnston, Eric Rescorla, Roni Even, and Oscar Novo provided useful ideas for the original [15]. The authors also acknowledge contributions to the revision of BFCP for use over an unreliable transport from Geir Arne Sandbakken, Charles Eckel, Alan Ford, Eoin McLeod and Mark Thompson. Useful and important final reviews were done by Ali C. Begen, Mary Barnes and Charles Eckel.

15. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [2] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [3] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [4] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [5] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [6] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling", RFC 5750, January 2010.
- [7] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, September 2005.
- [8] Camarillo, G., Drage, K., Kristensen, T., Ott, J., and C. Eckel, "The Binary Floor Control Protocol (BFCP)", draft-ietf-bfcpbis-rfc4582bis-11 (work in progress), February 2014.
- [9] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", RFC 4574, August 2006.
- [10] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 4572, July 2006.
- [11] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.
- [12] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [13] 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, May 2010.
- [14] Camarillo, G., Ott, J., and K. Drage, "The Binary Floor Control Protocol (BFCP)", RFC 4582, November 2006.

- [15] Camarillo, G., "Session Description Protocol (SDP) Format for Binary Floor Control Protocol (BFCP) Streams", RFC 4583, November 2006.

Authors' Addresses

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Gonzalo.Camarillo@ericsson.com

Tom Kristensen
Cisco
Philip Pedersens vei 22
N-1366 Lysaker
Norway

Email: tomkrist@cisco.com, tomkri@ifi.uio.no

BFCPbis Working Group
Internet-Draft
Obsoletes: 4583 (if approved)
Intended status: Standards Track
Expires: June 11, 2019

G. Camarillo
Ericsson
T. Kristensen
Cisco
C. Holmberg
Ericsson
December 8, 2018

Session Description Protocol (SDP) Format for Binary Floor Control
Protocol (BFCP) Streams
draft-ietf-bfcpbis-rfc4583bis-27

Abstract

This document defines the Session Description Protocol (SDP) offer/answer procedures for negotiating and establishing Binary Floor Control Protocol (BFCP) streams.

This document obsoletes RFC 4583. Changes from RFC 4583 are summarized in Section 14.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 11, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Conventions	3
3. Floor Control Roles	4
4. Fields in the 'm' Line	4
5. SDP Attributes	5
5.1. SDP 'floorctrl' Attribute	5
5.2. SDP 'confid' Attribute	7
5.3. SDP 'userid' Attribute	8
5.4. SDP 'floorid' Attribute	9
5.5. SDP 'bfcprver' Attribute	10
6. Multiplexing Considerations	11
7. BFCP Connection Management	12
7.1. TCP Connection Management	12
8. TLS/DTLS Considerations	13
9. ICE Considerations	13
10. SDP Offer/Answer Procedures	14
10.1. Generating the Initial SDP Offer	15
10.2. Generating the SDP Answer	15
10.3. Offerer Processing of the SDP Answer	16
10.4. Modifying the Session	17
11. Examples	17
12. Security Considerations	19
13. IANA Considerations	20
13.1. Registration of SDP 'proto' Values	20
13.2. Registration of the SDP 'floorctrl' Attribute	20
13.3. Registration of the SDP 'confid' Attribute	20
13.4. Registration of the SDP 'userid' Attribute	20
13.5. Registration of the SDP 'floorid' Attribute	21
13.6. Registration of the SDP 'bfcprver' Attribute	21
14. Changes from RFC 4583	21
15. Acknowledgements	22
16. References	22
16.1. Normative References	22
16.2. Informational References	24
Authors' Addresses	24

1. Introduction

As discussed in the BFCP (Binary Floor Control Protocol) specification [I-D.ietf-bfcpbis-rfc4582bis], a given BFCP client needs a set of data in order to establish a BFCP connection to a floor control server. This data includes the transport address of the server, the conference identifier, and the user identifier.

One way for clients to obtain this information is to use an SDP offer/answer [RFC3264] exchange. This document specifies how to encode this information in the SDP session descriptions that are part of such an offer/answer exchange.

User agents typically use the offer/answer model to establish a number of media streams of different types. Following this model, a BFCP connection is described as any other media stream by using an SDP 'm' line, possibly followed by a number of SDP lines that also apply to the BFCP connection.

Section 4 defines how the field values in 'm' line representing a BFCP connection are set.

Section 5 defines SDP attributes that are used when negotiating a BFCP connection.

Section 6 defines multiplexing considerations for a BFCP connection.

Section 7 defines procedures for managing a BFCP connection.

Section 8 defines TLS and DTLS considerations when negotiating a BFCP connection.

Section 9 defines the Interactive Connectivity Establishment (ICE) [RFC8445] considerations when negotiating a BFCP connection.

Section 10 defines the SDP offer/answer procedures for negotiating a BFCP connection.

2. Conventions

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Floor Control Roles

When two endpoints establish a BFCP stream, they need to determine which of them acts as a floor control client and which acts as a floor control server.

Once the roles have been determined, the roles will apply to all BFCP-controlled streams associated with the BFCP stream.

4. Fields in the 'm' Line

According to the SDP specification [RFC4566], the 'm' line format is the following:

```
m=<media> <port> <proto> <fmt> ...
```

This section describes how to generate an 'm' line of an SDP Media Description ('m' section) describing a BFCP stream.

The media field MUST have a value of "application".

The port field is set depending on the value of the proto field, as explained below. A port field value of zero has the standard SDP meaning (i.e., rejection of the media stream) regardless of the proto field.

When TCP is used as the transport, the port field is set following the rules in [RFC4145]. Depending on the value of the 'setup' attribute (discussed in Section 7.1), the port field contains the port to which the remote endpoint will direct BFCP messages, or in the case where the endpoint will initiate the connection towards the remote endpoint, should be set to a value of 9.

When UDP is used as the transport, the port field contains the port to which the remote endpoint will direct BFCP messages regardless of the value of the 'setup' attribute.

This document defines five values for the proto field: TCP/BFCP, TCP/DTLS/BFCP, TCP/TLS/BFCP, UDP/BFCP, and UDP/TLS/BFCP.

The proto value are used as described below:

'TCP/BFCP' is used for TCP transport of BFCP without TLS encryption, and is backward compatible with RFC 4583 compliant endpoints.

'TCP/TLS/BFCP' is used for TCP transport of BFCP with TLS encryption, and is backward compatible with RFC 4583 compliant endpoints that support TLS.

'UDP/BFCP' is used for UDP transport of BFCP without DTLS encryption [RFC6347].

'UDP/TLS/BFCP' is used for UDP transport of BFCP with DTLS encryption. This is one of the options when ICE is used (Section 9). It can also be used without ICE when backward compatibility with RFC 4583 compliant endpoints is not required.

'TCP/DTLS/BFCP' is used for TCP transport of BFCP with DTLS encryption, running on top of TCP using the framing method defined in [RFC4571], with DTLS packets being sent and received instead of RTP/RTCP packets using the shim defined in RFC 4571 such that the length field defined in RFC 4571 precedes each DTLS message. This is one of the options when ICE is used (Section 9). It can also be used without ICE when backward compatibility with RFC 4583 compliant endpoints is not required.

The fmt (format) list is not applicable to BFCP. The fmt list of 'm' lines in the case of any proto field value related to BFCP MUST contain a single "*" character. If the the fmt list contains any other value it MUST be ignored.

The following is an example of an 'm' line for a BFCP connection:

```
m=application 50000 TCP/TLS/BFCP *
```

5. SDP Attributes

5.1. SDP 'floorctrl' Attribute

This section defines the SDP 'floorctrl' media-level attribute. The attribute is used to determine the floor control roles (client and server) for the endpoints associated with the BFCP stream.

Attribute Name: floorctrl

Attribute Value: floor-control

Usage Level: media

Charset Dependent: No

Mux Category: TBD

The Augmented BNF syntax [RFC5234] for the attribute is:

```
floor-control = role *(SP role)
role = "c-only" / "s-only" / "c-s"
```

An endpoint includes the attribute to indicate the role(s) it would be willing to perform for the BFCP-controlled media streams:

c-only: The endpoint is willing to act as floor control client.

s-only: The endpoint is willing to act as floor control server only.

When inserted in an offer, the offerer MAY indicate multiple attribute values ("c-only" and "s-only"). When inserted in an answer, the answerer MUST indicate only one attribute value: "c-only" or "s-only". The answerer indicates the role taken by the answerer. The offerer will then take the opposite role.

In [RFC4583], there was a third attribute specified, "c-s", which meant that an endpoint was willing to act as both floor control client and floor control server at the same time for the BFCP stream, taking different roles for different BFCP-controlled media streams. The feature was underspecified and implemented in different ways, in particular many implementations interpreted "c-s" to mean that the endpoint is willing to act as either client or server (equivalent to "c-only s-only"). An implementation compliant to this specification MUST NOT include the "c-s" floorctrl attribute value in an offer or in an answer, but MUST accept the attribute value in an offer and process it as equivalent to "c-only s-only" (or "s-only c-only"). Also, as an implementation compliant to this specification is only allowed to include one role, either 'c-only' or 's-only', in an answer, each endpoint will only take one role, and as a result the endpoint will take the same role for each BFCP-controlled media stream associated with the BFCP stream.

Table 1 shows the roles that the answerer is allowed to take, based on what roles the offerer has indicated that it is willing to take.

Offerer	Answerer
c-only	s-only
s-only	c-only
c-s	c-only
c-s	s-only

Table 1: Roles

Endpoints compliant with [RFC4583] might not include the 'floorctrl' attribute in offers and answerer. If the 'floorctrl' attribute is not present, in order to be interoperable with such endpoints, the offerer will act as floor control client and the answerer will act as floor control server.

The SDP Offer/Answer procedures for the 'floorctrl' attribute are defined in Section 10.

The following is an example of a 'floorctrl' attribute in an offer:

```
a=floorctrl:c-only s-only
```

5.2. SDP 'confid' Attribute

This section defines the SDP 'confid' media-level attribute. The attribute is used by a floor control server to convey the conference ID value to the floor control client, using decimal integer representation.

Attribute Name: confid

Attribute Value: conference-id

Usage Level: media

Charset Dependent: No

Mux Category: TBD

The Augmented BNF syntax [RFC5234] for the attribute is:

conference-id = 1*DIGIT

DIGIT = <DIGIT defined in [RFC5234]>

The maximum value of the attribute is determined by the
COMMON-HEADER format [I-D.ietf-bfcpbis-rfc4582bis].

The SDP Offer/Answer procedures for the 'confid' attribute are
defined in Section 10.

5.3. SDP 'userid' Attribute

This section defines the SDP 'userid' media-level attribute. The
attribute is used by a floor control server to convey the user ID
value to the floor control client, using decimal integer
representation.

Attribute Name: userid

Attribute Value: user-id

Usage Level: media

Charset Dependent: No

Mux Category: TBD

The Augmented BNF syntax [RFC5234] for the attribute is:

user-id = 1*DIGIT

DIGIT = <DIGIT defined in [RFC5234]>

The maximum value of the attribute is determined by the COMMON-HEADER format [I-D.ietf-bfcpbis-rfc4582bis].

The SDP Offer/Answer procedures for the 'userid' attribute are defined in Section 10.

5.4. SDP 'floorid' Attribute

This section defines the SDP 'floorid' media-level attribute. The attribute conveys a floor identifier, using decimal integer representation, and optionally pointers to one or more BFCP-controlled media streams.

Attribute Name: floorid

Attribute Value: floor-id

Usage Level: media

Charset Dependent: No

Mux Category: TBD

The Augmented BNF syntax [RFC5234] for the attribute is:

floor-id = 1*DIGIT SP "mstrm:" token *(SP token)

DIGIT = <DIGIT defined in [RFC5234]>

token = <token defined in [RFC4566]>

The maximum value of the attribute is determined by the FLOOR-ID format [I-D.ietf-bfcpbis-rfc4582bis].

The floor identifier value is the integer representation of the Floor ID to be used in BFCP. Each media stream pointer value is associated with an SDP 'label' attribute [RFC4574] of a media stream.

The SDP Offer/Answer procedures for the 'floorid' attribute are defined in Section 10.

Note: In [RFC4583] 'm-stream' was erroneously used in Section 11. Although the example was non-normative, it is implemented by some vendors and occurs in cases where the endpoint is willing to act as a server. Therefore, it is RECOMMENDED to support parsing and interpreting 'm-stream' the same way as 'mstrm' when receiving.

5.5. SDP 'bfcpver' Attribute

This section defines the SDP 'bfcpver' media-level attribute. The attribute is used to negotiate the BFCP version, using decimal integer representation.

The Augmented BNF syntax [RFC5234] for the attributes is:

Attribute Name: bfcper

Attribute Value: bfcper-version

Usage Level: media

Charset Dependent: No

Mux Category: TBD

The Augmented BNF syntax [RFC5234] for the attribute is:

```
bfcper-version = version *(SP version)
version        = 1*DIGIT
```

DIGIT = <DIGIT defined in [RFC5234]>

The maximum value of the attribute is determined by the COMMON-HEADER format [I-D.ietf-bfcperbis-rfc4582bis].

An endpoint uses the 'bfcper' attribute to convey the version(s) of BFCP supported by the endpoint, using integer values. For a given version, the attribute value representing the version MUST match the "Version" field that would be presented in the BFCP COMMON-HEADER [I-D.ietf-bfcperbis-rfc4582bis]. The BFCP version that will eventually be used will be conveyed with a BFCP-level Hello/HelloAck.

Endpoints compliant with [RFC4583] might not always include the 'bfcper' attribute in offers and answers. The attribute value, if present, MUST be in accordance with the definition of the Version field in [I-D.ietf-bfcperbis-rfc4582bis]. If the attribute is not present, endpoints MUST assume a default value in accordance with [I-D.ietf-bfcperbis-rfc4582bis]: when used over a reliable transport the default attribute value is "1", and when used over an unreliable transport the default attribute value is "2". The value is inferred from the transport specified in the 'm' line (Section 4) of the 'm' section associated with the stream.

The SDP Offer/Answer procedures for the 'bfcper' attribute are defined in Section 10.

6. Multiplexing Considerations

[I-D.ietf-mmusic-sdp-bundle-negotiation] defines how multiplexing of multiple media streams can be negotiated. This specification does not define how BFCP streams can be multiplexed with other media streams. Therefore, a BFCP stream MUST NOT be associated with a

BUNDLE group [I-D.ietf-mmusic-sdp-bundle-negotiation]. Note that BFCP-controlled media streams might be multiplexed with other media streams.

[I-D.ietf-mmusic-sdp-mux-attributes] defines the mux categories for the SDP attributes defined in this specification, except for the 'bfcpxver' attribute. Table 2 defines the mux category for the 'bfcpxver' attribute:

Name	Notes	Level	Mux Category
bfcpxver	Needs further analysis in a separate specification	M	TBD

Table 2: Multiplexing Attribute Analysis

7. BFCP Connection Management

BFCP streams can use TCP or UDP as the underlying transport. Endpoints exchanging BFCP messages over UDP send the BFCP messages towards the peer using the connection address and port provided in the SDP 'c' and 'm' lines. TCP connection management is more complicated and is described in the following Section.

Note: When using Interactive Connectivity Establishment (ICE) [RFC8445], TCP/DTLS/BFCP, or UDP/TLS/BFCP, the straight-forward procedures for connection management as UDP/BFCP described above apply. TCP/TLS/BFCP follows the same procedures as TCP/BFCP and is described below.

7.1. TCP Connection Management

The management of the TCP connection used to transport BFCP messages is performed using the SDP 'setup' and 'connection' attributes [RFC4145]. The 'setup' attribute indicates which of the endpoints initiates the TCP connection. The 'connection' attribute handles TCP connection re-establishment.

The BFCP specification [I-D.ietf-bfcpxbis-rfc4582bis] describes a number of situations when the TCP connection between a floor control client and the floor control server needs to be re-established. However, that specification does not describe the re-establishment process because this process depends on how the connection was established in the first place. Endpoints using the offer/answer mechanism follow the following rules.

When the existing TCP connection is closed and re-established following the rules in [I-D.ietf-bfcpbis-rfc4582bis], the floor control client MUST send an offer towards the floor control server in order to re-establish the connection. If a TCP connection cannot deliver a BFCP message and times out, the endpoint that attempted to send the message (i.e., the one that detected the TCP timeout) MUST send an offer in order to re-establish the TCP connection.

Endpoints that use the offer/answer mechanism to negotiate TCP connections MUST support the 'setup' and 'connection' attributes.

8. TLS/DTLS Considerations

When DTLS is used with UDP, the generic procedures defined in Section 5 of [I-D.ietf-mmusic-dtls-sdp] MUST be followed.

When TLS is used with TCP, once the underlying connection is established, the answerer always acts as the TLS server. If the TCP connection is lost, the active endpoint [RFC4583] is responsible for re-establishing the TCP connection. Unless a new TLS connection is negotiated, subsequent SDP offers and answers will not impact the previously negotiated TLS roles.

Note: For TLS, it was decided to keep the original procedures in [RFC4583] to determine which endpoint acts as the TLS server in order to retain backwards compatibility.

9. ICE Considerations

Generic SDP offer/answer procedures for Interactive Connectivity Establishment (ICE) are defined in [I-D.ietf-mmusic-ice-sip-sdp].

When BFCP is used with UDP based ICE candidates [RFC8445] then the procedures for UDP/TLS/BFCP are used.

When BFCP is used with TCP based ICE candidates [RFC6544] then the procedures for TCP/DTLS/BFCP are used.

Based on the procedures defined in [I-D.ietf-mmusic-dtls-sdp], endpoints treat all ICE candidate pairs associated with a BFCP stream on top of a DTLS association as part of the same DTLS association. Thus, there will only be one BFCP handshake and one DTLS handshake even if there are multiple valid candidate pairs, and if BFCP media is shifted between candidate pairs (including switching between UDP to TCP candidate pairs) prior to nomination. If new candidates are added, they will also be part of the same DTLS association.

In order to maximize the likelihood of interoperability between the endpoints, all ICE enabled BFCP-over-DTLS endpoints SHOULD implement support for UDP/TLS/BFCP.

When an SDP offer or answer conveys multiple ICE candidates for a BFCP stream, UDP based candidates SHOULD be included and the default candidate SHOULD be chosen from one of those UDP candidates. If UDP transport is used for the default candidate, then the 'm' line proto value MUST be 'UDP/TLS/BFCP'. If TCP transport is used for the default candidate, the 'm' line proto value MUST be 'TCP/DTLS/BFCP'.

Note: Usage of ICE with protocols other than UDP/TLS/BFCP and TCP/DTLS/BFCP is outside of scope for this specification.

10. SDP Offer/Answer Procedures

This section defines the SDP offer/answer [RFC3264] procedures for negotiating and establishing a BFCP stream. Generic procedures for DTLS are defined in [I-D.ietf-mmusic-dtls-sdp]. Generic procedures for TLS are defined in [RFC8122].

This section only defines the BFCP-specific procedures. Unless explicitly stated otherwise, the procedures apply to an 'm' section describing a BFCP stream. If an offer or answer contains multiple 'm' sections describing BFCP streams, the procedures are applied independently to each stream.

Within this document, 'initial offer' refers to the first offer, within an SDP session (e.g. a SIP dialog when the Session Initiation Protocol (SIP) [RFC3261] is used to carry SDP) in which the offerer indicates that it wants to negotiate the establishment of a BFCP stream.

If the 'm' line 'proto' value is 'TCP/TLS/BFCP', 'TCP/DTLS/BFCP' or 'UDP/TLS/BFCP', the offerer and answerer follow the generic procedures defined in [RFC8122].

If the 'm' line proto value is 'TCP/BFCP', 'TCP/TLS/BFCP', 'TCP/DTLS/TCP' or 'UDP/TLS/BFCP', the offerer and answerer use the SDP 'setup' attribute according to the procedures in [RFC4145].

If the 'm' line proto value is 'TCP/BFCP', 'TCP/TLS/BFCP' or 'TCP/DTLS/BFCP', the offerer and answerer use the SDP 'connection' attribute according to the procedures in [RFC4145].

Note: The use of source-specific SDP parameters [RFC5576] is not defined for BFCP streams.

10.1. Generating the Initial SDP Offer

When the offerer creates an initial offer, the offerer MUST include an SDP 'floorctrl' attribute (Section 5.1) and an SDP 'bfcvver' attribute (Section 5.5) in the 'm' section.

In addition, if the offerer includes an SDP 'floorctrl' attribute with 's-only' or 'c-s' attribute values in the offer, the offerer:

- o MUST include an SDP 'confid' attribute (Section 5.2) in the 'm' section; and
- o MUST include an SDP 'userid' attribute (Section 5.3) in the 'm' section; and
- o MUST include an SDP 'floorid' attribute (Section 5.4) in the 'm' section; and
- o MUST include an SDP 'label' attribute ([RFC4574]) with the 'm' section of each BFCP-controlled media stream.

Note: If the offerer includes an SDP 'floorctrl' attribute with a 'c-s' attribute value, or both a 'c-only' and a 's-only' attribute value in the offer, the attribute values above will only be used if it is determined (Section 5.1) that the offerer will act as floor control server.

10.2. Generating the SDP Answer

When the answerer receives an offer that contains an 'm' section describing a BFCP stream, the answerer MUST check whether it supports one or more of the BFCP versions supported by the offerer (Section 5.5). If the answerer does not support any of the BFCP versions, it MUST NOT accept the 'm' section. Otherwise, if the answerer accepts the 'm' section, it:

- o MUST insert a corresponding 'm' section in the answer, with an identical 'm' line proto value [RFC3264]; and
- o MUST include a 'bfcvver' attribute in the 'm' section. The versions indicated by the answer MUST be the same or a subset of the versions indicated by the offerer in the corresponding offer; and
- o MUST, if the offer contained an SDP 'floorctrl' attribute, include a 'floorctrl' attribute in the 'm' section.

In addition, if the answerer includes an SDP 'floorctrl' attribute with an 's-only' attribute value in the answer, the answerer:

- o MUST include an SDP 'confid' attribute in the 'm' section; and
- o MUST include an SDP 'userid' attribute in the 'm' section; and
- o MUST include an SDP 'floorid' attribute in the 'm' section; and
- o MUST include an SDP 'label' attribute in the 'm' section of each BFCP-controlled media stream.

Note: An offerer compliant with [RFC4583] might not include 'floorctrl' and 'bfcpsver' attributes in offers, in which cases the default values apply.

Once the answerer has sent the answer, the answerer:

- o MUST, if the answerer is the active endpoint, and if a TCP connection associated with the 'm' section is to be established (or re-established), initiate the establishing of the TCP connection; and
- o MUST, if the answerer is the active endpoint, and if an TLS/DTLS connection associated with the 'm' section is to be established (or re-established), initiate the establishing of the TLS/DTLS connection (by sending a ClientHello message).

If the answerer does not accept the 'm' section in the offer, it MUST assign a zero port value to the 'm' line of the corresponding 'm' section in the answer. In addition, the answerer MUST NOT establish a TCP connection or a TLS/DTLS connection associated with the 'm' section.

10.3. Offerer Processing of the SDP Answer

When the offerer receives an answer that contains an 'm' section with a non-zero port value, describing a BFCP stream, the offerer:

- o MUST, if the offerer is the active endpoint, and if a TCP connection associated with the 'm' section is to be established (or re-established), initiate the establishing of the TCP connection; and
- o MUST, if the offerer is the active endpoint, and if an TLS/DTLS connection associated with the 'm' section is to be established (or re-established), initiate the establishing of the TLS/DTLS connection (by sending a ClientHello message).

Note: An answerer compliant with [RFC4583] might not include 'floorctrl' and 'bfcprver' attributes in answers, in which cases the default values apply.

If the 'm' line in the answer contains a zero port value, or if the offerer for some other reason does not accept the answer (e.g., if the answerer only indicates support of BFCP versions not supported by the offerer), the offerer MUST NOT establish a TCP connection or a TLS/DTLS connection associated with the 'm' section.

10.4. Modifying the Session

When an offerer sends an updated offer, in order to modify a previously established BFCP stream, it follows the procedures in Section 10.1, with the following exceptions:

- o If the BFCP stream is carried on top of TCP, and if the offerer does not want to re-establish an existing TCP connection, the offerer MUST include an SDP 'connection' attribute with a value of "existing", in the 'm' section; and
- o If the offerer wants to disable a previously established BFCP stream, it MUST assign a zero port value to the 'm' line associated with the BFCP connection, following the procedures in [RFC3264].

11. Examples

For the purpose of brevity, the main portion of the session description is omitted in the examples, which only show 'm' sections and their 'm' lines and attributes.

The following is an example of an offer sent by a conference server to a client.


```
m=application 50000 TCP/TLS/BFCP *
a=setup:actpass
a=connection:new
a=fingerprint:sha-256 \
    19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04: \
    BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=floorctrl:c-only s-only
a=confid:4321
a=userid:1234
a=floorid:1 mstrm:10
a=floorid:2 mstrm:11
a=bfcper:1 2
m=audio 50002 RTP/AVP 0
a=label:10
m=video 50004 RTP/AVP 31
a=label:11
```

Note that due to RFC formatting conventions, this document splits SDP across lines whose content would exceed 72 characters. A backslash character marks where this line folding has taken place. This backslash and its trailing CRLF and whitespace would not appear in actual SDP content.

The following is the answer returned by the client.

```
m=application 9 TCP/TLS/BFCP *
a=setup:active
a=connection:new
a=fingerprint:sha-256 \
    6B:8B:F0:65:5F:78:E2:51:3B:AC:6F:F3:3F:46:1B:35: \
    DC:B8:5F:64:1A:24:C2:43:F0:A1:58:D0:A1:2C:19:08
a=floorctrl:c-only
a=bfcper:1
m=audio 55000 RTP/AVP 0
m=video 55002 RTP/AVP 31
```

A similar example using unreliable transport and DTLS is shown below, where the offer is sent from a client.

```
m=application 50000 UDP/TLS/BFCP *
a=setup:actpass
a=dtls-id:abc3dl
a=fingerprint:sha-256 \
    19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04: \
    BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=floorctrl:c-only s-only
a=confid:4321
a=userid:1234
a=floorid:1 mstrm:10
a=floorid:2 mstrm:11
a=bfcper:1 2
m=audio 50002 RTP/AVP 0
a=label:10
m=video 50004 RTP/AVP 31
a=label:11
```

The following is the answer returned by the server.

```
m=application 55000 UDP/TLS/BFCP *
a=setup:active
a=dtls-id:abc3dl
a=fingerprint:sha-256 \
    6B:8B:F0:65:5F:78:E2:51:3B:AC:6F:F3:3F:46:1B:35: \
    DC:B8:5F:64:1A:24:C2:43:F0:A1:58:D0:A1:2C:19:08
a=floorctrl:s-only
a=confid:4321
a=userid:1234
a=floorid:1 mstrm:10
a=floorid:2 mstrm:11
a=bfcper:2
m=audio 55002 RTP/AVP 0
m=video 55004 RTP/AVP 31
```

12. Security Considerations

The BFCP [I-D.ietf-bfcperbis-rfc4582bis], SDP [RFC4566], and offer/answer [RFC3264] specifications discuss security issues related to BFCP, SDP, and offer/answer, respectively. In addition, [RFC4145] and [RFC8122] discuss security issues related to the establishment of TCP and TLS connections using an offer/answer model. Furthermore, when using DTLS over UDP, the generic offer/answer considerations defined in [I-D.ietf-mmusic-dtls-sdp] MUST be followed.

The usage of certain proto values in the SDP offer/answer negotiation will result in a BFCP stream that is not protected by TLS or DTLS. Operators will need to provide integrity protection and confidentiality protection of the BFCP stream using other means.

The generic security considerations associated with SDP attributes are defined in [RFC3264]. While the attributes defined in this specification do not reveal information about the content of individual BFCP controlled media streams, they do reveal which media streams will be BFCP controlled.

13. IANA Considerations

[Editorial note: The changes in Section 13.1 instruct the IANA to register the three new values TCP/DTLS/BFCP, UDP/BFCP and UDP/TLS/BFCP for the SDP 'proto' field. The new section Section 5.5 registers a new SDP "bfcvver" attribute. The rest is unchanged from [RFC4582].]

13.1. Registration of SDP 'proto' Values

The IANA is requested to register the following values for the SDP 'proto' field under the Session Description Protocol (SDP) Parameters registry:

Value	Reference
TCP/BFCP	[RFC XXXX]
TCP/DTLS/BFCP	[RFC XXXX]
TCP/TLS/BFCP	[RFC XXXX]
UDP/BFCP	[RFC XXXX]
UDP/TLS/BFCP	[RFC XXXX]

Table 3: Values for the SDP 'proto' field

13.2. Registration of the SDP 'floorctrl' Attribute

This document defines the SDP attribute, 'floorctrl'. The details of the attribute are defined in Section 5.1.

13.3. Registration of the SDP 'confid' Attribute

This document defines the SDP attribute, 'confid'. The details of the attribute are defined in Section 5.2.

13.4. Registration of the SDP 'userid' Attribute

This document defines the SDP attribute, 'userid'. The details of the attribute are defined in Section 5.3.

13.5. Registration of the SDP 'floorid' Attribute

This document defines the SDP attribute, 'floorid'. The details of the attribute are defined in Section 5.4.

13.6. Registration of the SDP 'bfcvver' Attribute

This document defines the SDP attribute, 'bfcvver'. The details of the attribute are defined in Section 5.5.

14. Changes from RFC 4583

Following is the list of technical changes and other fixes from [RFC4583].

Main purpose of this work was to add signaling support necessary to support BFCP over unreliable transport, as described in [I-D.ietf-bfcvver-rfc4582bis], resulting in the following changes:

1. Fields in the 'm' line (Section 4):
The section is re-written to remove reference to the exclusivity of TCP as a transport for BFCP streams. The proto field values TCP/DTLS/BFCP, UDP/BFCP and UDP/TLS/BFCP added.
2. Security Considerations (Section 12):
For the DTLS over UDP case, mention existing considerations and requirements for the offer/answer exchange in [I-D.ietf-mmusic-dtls-sdp].
3. Registration of SDP 'proto' Values (Section 13.1):
Register the three new values TCP/DTLS/BFCP, UDP/BFCP and UDP/TLS/BFCP in the SDP parameters registry.
4. BFCP Version Negotiation (Section 5.5):
A new 'bfcvver' SDP media-level attribute is added in order to signal supported version number.

In addition to the changes associated with support of BFCP over unreliable transport, the possibility for an endpoint to act as both floor control client and floor control server at the same time has been removed. An endpoint will now take the same role for all BFCP-controlled streams associated with the BFCP stream.

Clarification and bug fixes:

1. Errata ID: 712 (Section 3 and Section 10):

Language clarification. Don't use terms like an SDP attribute is "used in an 'm' line", instead make clear that the attribute is a media-level attribute.

2. Fix typo in example (Section 11):
Do not use 'm-stream' in the SDP example, use the correct 'mstrm' as specified in Section 11. Recommend interpreting 'm-stream' if it is received, since it is present in some implementations.
3. Assorted clarifications (Across the document):
Language clarifications as a result of reviews. Also, the normative language where tightened where appropriate, i.e. changed from SHOULD strength to MUST in a number of places.

15. Acknowledgements

Joerg Ott, Keith Drage, Alan Johnston, Eric Rescorla, Roni Even, and Oscar Novo provided useful ideas for the original [RFC4583]. The authors also acknowledge contributions to the revision of BFCP for use over an unreliable transport from Geir Arne Sandbakken, Charles Eckel, Alan Ford, Eoin McLeod and Mark Thompson. Useful and important final reviews were done by Ali C. Begen, Mary Barnes and Charles Eckel. In the final stages, Roman Shpount made a considerable effort in adding proper ICE support and considerations.

16. References

16.1. Normative References

[I-D.ietf-bfcpbis-rfc4582bis]

Camarillo, G., Drage, K., Kristensen, T., Ott, J., and C. Eckel, "The Binary Floor Control Protocol (BFCP)", draft-ietf-bfcpbis-rfc4582bis-16 (work in progress), November 2015.

[I-D.ietf-mmusic-dtls-sdp]

Holmberg, C. and R. Shpount, "Session Description Protocol (SDP) Offer/Answer Considerations for Datagram Transport Layer Security (DTLS) and Transport Layer Security (TLS)", draft-ietf-mmusic-dtls-sdp-32 (work in progress), October 2017.

[I-D.ietf-mmusic-ice-sip-sdp]

Petit-Huguenin, M., Nandakumar, S., and A. Keranen, "Session Description Protocol (SDP) Offer/Answer procedures for Interactive Connectivity Establishment (ICE)", draft-ietf-mmusic-ice-sip-sdp-24 (work in progress), November 2018.

- [I-D.ietf-mmusic-sdp-mux-attributes]
Nandakumar, S., "A Framework for SDP Attributes when Multiplexing", draft-ietf-mmusic-sdp-mux-attributes-17 (work in progress), February 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3261] 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>>.
- [RFC3264] 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>>.
- [RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, DOI 10.17487/RFC4145, September 2005, <<https://www.rfc-editor.org/info/rfc4145>>.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<https://www.rfc-editor.org/info/rfc4566>>.
- [RFC4571] Lazzaro, J., "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport", RFC 4571, DOI 10.17487/RFC4571, July 2006, <<https://www.rfc-editor.org/info/rfc4571>>.
- [RFC4574] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", RFC 4574, DOI 10.17487/RFC4574, August 2006, <<https://www.rfc-editor.org/info/rfc4574>>.
- [RFC4582] 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>>.
- [RFC4583] Camarillo, G., "Session Description Protocol (SDP) Format for Binary Floor Control Protocol (BFCP) Streams", RFC 4583, DOI 10.17487/RFC4583, November 2006, <<https://www.rfc-editor.org/info/rfc4583>>.

- [RFC5234] 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>>.
- [RFC6347] 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>>.
- [RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, DOI 10.17487/RFC6544, March 2012, <<https://www.rfc-editor.org/info/rfc6544>>.
- [RFC8122] Lennox, J. and C. Holmberg, "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 8122, DOI 10.17487/RFC8122, March 2017, <<https://www.rfc-editor.org/info/rfc8122>>.
- [RFC8174] 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>>.
- [RFC8445] 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>>.

16.2. Informational References

- [I-D.ietf-mmusic-sdp-bundle-negotiation] Holmberg, C., Alvestrand, H., and C. Jennings, "Negotiating Media Multiplexing Using the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-bundle-negotiation-53 (work in progress), September 2018.
- [RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, DOI 10.17487/RFC5576, June 2009, <<https://www.rfc-editor.org/info/rfc5576>>.

Authors' Addresses

Gonzalo Camarillo
Ericsson
Hirsalantie 11
FI-02420 Jorvas
Finland

Email: Gonzalo.Camarillo@ericsson.com

Tom Kristensen
Cisco
Philip Pedersens vei 1
NO-1366 Lysaker
Norway

Email: tomkrist@cisco.com, tomkri@ifi.uio.no

Christer Holmberg
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: christer.holmberg@ericsson.com

BFCPBIS Working Group
Internet-Draft
Updates: rfc4582bis, rfc4583bis (if approved)
Intended status: Standards Track
Expires: August 13, 2014

V. Pascual
A. Roman
Quobis
S. Cazeaux
France Telecom Orange
G. Salgueiro
Cisco
S. Garcia Murillo
Medooze
February 9, 2014

The WebSocket Protocol as a Transport for the Binary Floor Control
Protocol (BFCP)
draft-pascual-bfcpbis-bfcp-websocket-00

Abstract

The WebSocket protocol enables two-way realtime communication between clients and servers. This document specifies a new WebSocket sub-protocol as a reliable transport mechanism between Binary Floor Control Protocol (BFCP) entities to enable usage of BFCP in new scenarios. This document normatively updates [I-D.draft-ietf-bfcpbis-rfc4582bis] and [I-D.draft-ietf-bfcpbis-rfc4583bis]

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 13, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
2.1. Definitions	3
3. The WebSocket Protocol	4
4. The WebSocket BFCP Sub-Protocol	4
4.1. Handshake	5
4.2. BFCP encoding	5
5. BFCP WebSocket Transport	6
6. Fields in the 'm' Line	6
7. Authentication	7
8. Security Considerations	8
9. IANA Considerations	9
9.1. Registration of the WebSocket BFCP Sub-Protocol	9
9.2. Registration of the 'TCP/WS/BFCP' and 'TCP/WSS/BFCP' SDP 'proto' Values	9
9.3. Registration of the 'ws-uri' SDP media attribute	9
9.4. Acknowledgements	10
10. References	10
10.1. Normative References	10
10.2. Informative References	10
Authors' Addresses	11

1. Introduction

The WebSocket [RFC6455] protocol enables two-way message exchange between clients and servers on top of a persistent TCP connection (optionally secured with TLS [RFC5246]). The initial protocol handshake makes use of HTTP [RFC2616] semantics, allowing the WebSocket protocol to reuse existing HTTP infrastructure.

The Binary Floor Control Protocol (BFCP) is a protocol to coordinate access to shared resources in a conference. It is defined in [I-D.ietf-bfcpbis-rfc4582bis] and is used between floor participants and floor control servers, and between floor chairs (i.e., moderators) and floor control servers.

Modern web browsers include a WebSocket client stack complying with the WebSocket API [WS-API] as specified by the W3C. It is expected that other client applications (those running in personal computers and devices such as smartphones) will also make a WebSocket client stack available. This document updates [I-D.ietf-bfcpbis-rfc4582bis] and [I-D.ietf-bfcpbis-rfc4583bis] in order to enable the usage of BFCP in these scenarios.

The transport over which BFCP entities exchange messages depends on how the clients obtain information to contact the floor control server (e.g. using an SDP offer/answer exchange per [I-D.ietf-bfcpbis-rfc4583bis] or the procedure described in RFC5018 [RFC5018]). [I-D.ietf-bfcpbis-rfc4582bis] defines two transports for BFCP: TCP and UDP. This specification defines a new WebSocket sub-protocol (as defined in section 1.9 in [RFC6455]) for transporting BFCP messages between a WebSocket client and server, a new reliable and message boundary transport for BFCP. In order to enable this, this document also defines two new SDP 'proto' values.

This document does not restrict the selection nor prevent the usage of other transport mechanisms for the BFCP protocol. Transport selection is entirely at the discretion of the application. As an example, an RTCWeb applications may choose to use either DataChannel or WebSocket transport for BFCP, while non-RTCWeb applications could still benefit from the ubiquity of the WebSocket protocol and make use of the transport for BFCP defined in this document.

2. Terminology

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

2.1. Definitions

BFCP WebSocket Client: A BFCP entity capable of opening outbound connections to WebSocket servers and communicating using the WebSocket BFCP sub-protocol as defined by this document.

BFCP WebSocket Server: A BFCP entity capable of listening for inbound connections from WebSocket clients and communicating using the WebSocket BFCP sub-protocol as defined by this document.

3. The WebSocket Protocol

The WebSocket protocol [RFC6455] is a transport layer on top of TCP (optionally secured with TLS [RFC5246]) in which both client and server exchange message units in both directions. The protocol defines a connection handshake, WebSocket sub-protocol and extensions negotiation, a frame format for sending application and control data, a masking mechanism, and status codes for indicating disconnection causes.

The WebSocket connection handshake is based on HTTP [RFC2616] and utilizes the HTTP GET method with an "Upgrade" request. This is sent by the client and then answered by the server (if the negotiation succeeded) with an HTTP 101 status code. Once the handshake is completed the connection upgrades from HTTP to the WebSocket protocol. This handshake procedure is designed to reuse the existing HTTP infrastructure. During the connection handshake, client and server agree on the application protocol to use on top of the WebSocket transport. Such an application protocol (also known as a "WebSocket sub-protocol") defines the format and semantics of the messages exchanged by the endpoints. This could be a custom protocol or a standardized one (as the WebSocket BFCP sub-protocol defined in this document). Once the HTTP 101 response is processed both client and server reuse the underlying TCP connection for sending WebSocket messages and control frames to each other. Unlike plain HTTP, this connection is persistent and can be used for multiple message exchanges.

The WebSocket protocol defines message units to be used by applications for the exchange of data, so it provides a message boundary-preserving transport layer. These message units can contain either UTF-8 text or binary data, and can be split into multiple WebSocket text/binary transport frames as needed by the WebSocket stack.

The WebSocket API [WS-API] for web browsers only defines callbacks to be invoked upon receipt of an entire message unit, regardless of whether it was received in a single WebSocket frame or split across multiple frames.

4. The WebSocket BFCP Sub-Protocol

The term WebSocket sub-protocol refers to an application-level protocol layered on top of a WebSocket connection. This document specifies the WebSocket BFCP sub-protocol for carrying BFCP messages through a WebSocket connection.

4.1. Handshake

The BFCP WebSocket Client and BFCP WebSocket Server negotiate usage of the WebSocket BFCP sub-protocol during the WebSocket handshake procedure as defined in section 1.3 of [RFC6455]. The Client **MUST** include the value "bfc" in the Sec-WebSocket-Protocol header in its handshake request. The 101 reply from the Server **MUST** contain "bfc" in its corresponding Sec-WebSocket-Protocol header.

Below is an example of a WebSocket handshake in which the Client requests the WebSocket BFCP sub-protocol support from the Server:

```
GET / HTTP/1.1
Host: bfc-ws.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://www.example.com
Sec-WebSocket-Protocol: bfc
Sec-WebSocket-Version: 13
```

The handshake response from the Server accepting the WebSocket BFCP sub-protocol would look as follows:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: bfc
```

Once the negotiation has been completed, the WebSocket connection is established and can be used for the transport of BFCP messages. The WebSocket messages transmitted over this connection **MUST** conform to the negotiated WebSocket sub-protocol.

4.2. BFCP encoding

BFCP messages use a TLV (Type-Length-Value) binary encoding, therefore BFCP WebSocket Clients and BFCP WebSocket Servers **MUST** be transported in unfragmented binary WebSocket frames (FIN:1,opcode:%x2) to exchange BFCP messages. The WebSocket frame data **MUST** be a valid BFCP message, so the length of the payload of the WebSocket frame **MUST** be lower than the maximum size allowed ($2^{16} + 12$ bytes) for a BFCP message as described in [I-D.ietf-bfcpbis-rfc4582bis]. In addition, the encoding rules for reliable protocols defined in [I-D.ietf-bfcpbis-rfc4582bis] **MUST** be followed.

5. BFCP WebSocket Transport

WebSocket [RFC6455] is a reliable protocol and therefore the BFCP WebSocket sub-protocol defined by this document is a reliable BFCP transport. Thus, client and server transactions using WebSocket for transport MUST follow the procedures for reliable transports as defined in [I-D.ietf-bfcpbis-rfc4582bis] and [I-D.ietf-bfcpbis-rfc4583bis]

BFCP WebSocket clients cannot receive incoming WebSocket connections initiated by any other peer. This means that a BFCP Websocket client MUST actively initiate a connection towards a BFCP Websocket server

Each BFCP message MUST be carried within a single WebSocket message, and a WebSocket message MUST NOT contain more than one BFCP message.

6. Fields in the 'm' Line

Rules to generate an 'm' line for a BFCP stream are described in [I-D.ietf-bfcpbis-rfc4583bis], Section 3

New values are defined for the transport field: TCP/WS/BFCP and TCP/WSS/BFCP.

TCP/WS/BFCP is used when BFCP runs on top of WS, which in turn runs on top of TCP.

TCP/WSS/BFCP is used when BFCP runs on top of WSS, which in turn runs on top of TLS and TCP.

When TCP is used as the transport, the port field is set following the rules in Section 7 of [I-D.ietf-bfcpbis-rfc4582bis]. Depending on the value of the 'setup' attribute, the port field contains the port to which the remote endpoint will direct BFCP messages or is irrelevant (i.e., the endpoint will initiate the connection towards the remote endpoint) and should be set to a value of 9, which is the discard port. Connection attribute and port MUST follow the rules of [RFC4145]

Some web browsers do not allow non-secure Websocket connections to be made. So, while the recommendation to use Secure WebSockets (i.e. TCP/WSS) is for security reasons, it is also to achieve maximum compatibility among clients.

When using Secure Websockets the CNAME of the SSL certificate must match the WebSocket connection URI host, and while it is possible to generate self signed certificates with IPs as CNAME, it will not be viable in most cases for certificates signed by well known

authorities. So, a new attribute 'ws-uri' is defined in this specification to indicate the connection uri for the WebSocket Client. The Augmented BNF syntax as described in [RFC4234] is:

```
ws-uri      = "a=ws-uri:" ws-URI
```

Where ws-URI is defined in [RFC6455]

When the 'ws-uri' attribute is present in the BFCP media section of the SDP, the IP and port provided in the 'c' lines SHALL be ignored and the full uri SHALL be used instead to open the WebSocket connection

The following are examples of 'm' lines for BFCP connections:

Offer (browser):

```
m=application 9 TCP/WSS/BFCP *  
a=setup:active  
a=connection:new  
a=floorctrl:c-only  
m=audio 55000 RTP/AVP 0  
m=video 55002 RTP/AVP 31
```

Answer (server):

```
m=application 50000 TCP/WSS/BFCP *  
a=setup:passive  
a=connection:new  
a=ws-uri:wss://bfcps-ws.example.com?token=3170449312  
a=floorctrl:s-only  
a=confid:4321  
a=userid:1234  
a=floorid:1 m-stream:10  
a=floorid:2 m-stream:11  
m=audio 50002 RTP/AVP 0  
a=label:10  
m=video 50004 RTP/AVP 31  
a=label:11
```

7. Authentication

Section 9 of [I-D.ietf-bfcpsbis-rfc4582bis] states that BFCP clients and floor control servers SHOULD authenticate each other prior to accepting messages, and RECOMMENDS that mutual TLS/DTLS authentication be used. However, browser-based WebSocket clients have no control over the use of TLS in the WebSocket API [WS-API], so it is RECOMMENDED that standard Web-based methods for client and server authentication are used, as follows.

When a BFCP WebSocket client connects to a BFCP WebSocket server, it SHOULD use TCP/WSS as its transport. The WebSocket client SHOULD inspect the TLS certificate offered by the server and verify that it is valid.

Since the WebSocket API does not distinguish between certificate errors and other kinds of failure to establish a connection, it is expected that browser vendors will warn end users directly of any kind of problem with the server certificate.

A floor control server that receives a message over TCP/WS can request the use of TCP/WSS by generating an Error message, as described in Section 13.8 of [I-D.ietf-bfcpbis-rfc4582bis], with an Error code with a value of 9 (use TLS).

Prior to sending BFCP requests, a BFCP WebSocket client connects to a BFCP WebSocket server and performs the connection handshake. As described in Section 3 the handshake procedure involves a HTTP GET method request from the client and a response from the server including an HTTP 101 status code.

In order to authorize the WebSocket connection, the BFCP WebSocket server MAY inspect any cookie [RFC6265] headers present in the HTTP GET request. For many web applications the value of such a cookie is provided by the web server once the user has authenticated themselves to the web server, which could be done by many existing mechanisms. As an alternative method, the BFCP WebSocket Server could request HTTP authentication by replying to the Client's GET method request with a HTTP 401 status code. The WebSocket protocol [RFC6455] covers this usage in section 4.1:

If the status code received from the server is not 101, the WebSocket client stack handles the response per HTTP [RFC2616] procedures, in particular the client might perform authentication if it receives 401 status code.

8. Security Considerations

Considerations from [I-D.ietf-bfcpbis-rfc4582bis], [I-D.ietf-bfcpbis-rfc4583bis] and RFC5018 [RFC5018] apply.

BFCP relies on lower-layer security mechanisms to provide replay and integrity protection and confidentiality. It is RECOMMENDED that the BFCP traffic transported over a WebSocket communication be protected by using a secure WebSocket connection (using TLS [RFC5246] over TCP).

9. IANA Considerations

9.1. Registration of the WebSocket BFCP Sub-Protocol

This specification requests IANA to register the WebSocket BFCP sub-protocol under the "WebSocket Subprotocol Name" Registry with the following data:

Subprotocol Identifier: `bfcf`

Subprotocol Common Name: WebSocket Transport for BFCP (Binary Floor Control Protocol)

Subprotocol Definition: TBD: this document

9.2. Registration of the 'TCP/WS/BFCP' and 'TCP/WSS/BFCP' SDP 'proto' Values

This document defines two new values for the SDP 'proto' field under the Session Description Protocol (SDP) Parameters registry. The resulting entries are shown in Figure 1 below:

Value	Reference
-----	-----
TCP/WS/BFCP	[TBD: this document]
TCP/WSS/BFCP	[TBD: this document]

Figure 1: Values for the SDP 'proto' field

9.3. Registration of the 'ws-uri' SDP media attribute

This section instructs the IANA to register the following SDP attribute under the Session Description Protocol (SDP) Parameters registry:

Contact name TBD

Attribute name `ws-uri`

Long-form attribute name WebSocket Connection URI

Type of attribute Media level

Subject to charset No

Purpose of attribute The 'ws-uri' attribute is intended to be used as connection URI for opening the WebSocket.

Allowed attribute values A ws-URI as defined in [RFC6455]

9.4. Acknowledgements

The authors want to thank Robert Welboun, from Acme Packet, who made significant contributions to the first version of this document.

10. References

10.1. Normative References

- [I-D.ietf-bfcpbis-rfc4582bis]
Camarillo, G., Drage, K., Kristensen, T., Ott, J., and C. Eckel, "The Binary Floor Control Protocol (BFCP)", draft-ietf-bfcpbis-rfc4582bis-10 (work in progress), November 2013.
- [I-D.ietf-bfcpbis-rfc4583bis]
Camarillo, G. and T. Kristensen, "Session Description Protocol (SDP) Format for Binary Floor Control Protocol (BFCP) Streams", draft-ietf-bfcpbis-rfc4583bis-08 (work in progress), November 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, September 2005.
- [RFC4234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.
- [RFC5018] Camarillo, G., "Connection Establishment in the Binary Floor Control Protocol (BFCP)", RFC 5018, September 2007.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

10.2. Informative References

- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[WS-API] W3C and I. Hickson, Ed., "The WebSocket API", May 2012.

Authors' Addresses

Victor Pascual
Quobis

Email: victor.pascual@quobis.com

Anton Roman
Quobis

Email: anton.roman@quobis.com

Stephane Cazeaux
France Telecom Orange

Email: stephane.cazeaux@orange.com

Gonzalo Salgueiro
Cisco Systems, Inc.
7200-12 Kit Creek Road
Research Triangle Park, NC 27709
US

Email: gsalguei@cisco.com

Sergio Garcia Murillo
Medooze

Email: sergio.garcia.murillo@gmail.com