NISTIR 8236

Test Scenarios for Mission Critical Push-To-Talk (MCPTT) Off-Network Mode Protocols Implementation

Priam Varin
Yishen Sun
Wesley Garey

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8236

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 8236

Test Scenarios for Mission Critical Push-To-Talk (MCPTT) Off-Network Mode Protocols Implementation

Priam Varin
Yishen Sun
Wesley Garey
Wireless Networks Division
Communications Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8236

October 2018

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

Abstract

The document provides both an overview and technical details of test scenarios that are designed by our group. These test scenarios can be used to check the protocol implementation of the 3rd Generation Partnership Project (3GPP) defined off-network mode operations of mission critical push-to-talk (MCPTT), covering floor control, call control, and call type control. This report includes test scenarios for basic group calls, broadcast group calls, and private calls. A test scenario is presented in a table format, and sometimes is accompanied by a figure that shows the protocol messages exchanged over the air as well.

Keywords

MCPTT; off-network; basic group call; broadcast group call; private call; call control; floor control; test scenarios

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Motivation

Public safety agencies require real-time applications. Mission critical voice (MCV) real-time application remains the most critical means of communications for first responders in emergency situations and cannot be compromised. The major MCV communication device is using the mission critical push-to-talk (MCPTT) technology, which achieves public safety grade PTT voice quality. A few general public safety requirements/expectations of push-to-talk communications over Long Term Evolution (LTE) may be found in [1]. The 3rd Generation Partnership Project (3GPP) working groups have defined a set of specifications [2-7] for MCPTT operations over LTE in its recent releases: Release-13, Release-14, and Release 15, for both on-network mode and off-network mode.

When mission critical voice communication is desired while first responders are out of LTE network coverage or when the network coverage is very weak, it is a very attractive or the only feasible option for public safety users to operate in MCPTT off-network mode. However, due to the lack of time or effort spent on off-network scenarios by the 3GPP working groups, the completeness and correctness of MCPTT off-network mode are yet to be fully verified. In fact, although the relevant 3GPP Release-13 specifications were officially completed in March 2016, quite a number of change requests (CRs) were submitted to LTE work groups to correct errors. New features and enhancements of existing functionalities were discussed for Release-14 and Release 15, as well. Therefore, evaluating the performance and limitations of MCPTT off-network operations is of primary interest to the public safety community. For this reason, it is one of the main research focuses of our division. Unless noted otherwise, the MCPTT protocols, research findings, or both that will be discussed in the remainder of the document all refer to MCPTT off-network mode.

Since the spring of 2016, our group has been developing an MCPTT off-network mode model in ns-3. Major features of this model include basic group call capability, broadcast group call capability, private call capability, floor control, call control, call type control, and emergency alert. In order to check the correctness of protocol implementation defined in that ns-3 model, our group has designed 61 test scenarios for MCPTT off-network mode operations. Future revisions of this document may include additional scenarios. These test scenarios were used to check our implementation of the MCPTT off-network mode protocols using ns-3 simulation tool. As can be seen from the example shown in Section 1.3, as well as throughout the remaining document, these test scenarios are based on the 3GPP MCPTT off-network mode protocols and are not ns-3 specific. Therefore, there are potential benefits of utilizing these test scenarios to verify the MCPTT off-network mode protocol implementations in other embodiments, e.g., prototype or commercial devices. Utilizing the information provided in these scenarios, one could create conformance tests or interoperability tests, as shown through examples in the Appendix.

1.2. Overview

The remainder of the document is organized as follows. Example test scenarios are given in Section 1.3, so that the interpretation and utilization of a test scenario may be illustrated in more details. Section 2 is the collection of all test scenarios that our group has designed for MCPTT off-network mode basic group call, including 14 test scenarios on floor control (Section 2.1), 12 test scenarios on call control (Section 2.2), and 21 test scenarios on call type control (Section 2.3). Section 3 summarizes all test scenarios we designed for MCPTT off-network broadcast group call, including 4 scenarios on call control (Section 3.1). Section 4 presents test scenarios developed for MCPTT off-network private call, including 8 scenarios on call control (Section 4.1) and 8 scenarios on call type control (Section 4.2). Summary and future work is discussed in Section 5.

Test scenarios are designed to cover possible state transitions, as well as the main procedures of MCPTT off-network mode operations.

1.3. Test scenario examples

In this section, a test scenario example will be explained in detail, in order to help the interpretation and utilization of other test scenarios listed in the remainder of the document.

Each test scenario is presented in a table format. The table captures important information that needs to be checked for each test scenario, such as event triggers, sequences, message exchanges, timers, counters, and states. In addition to the table, a corresponding figure is included in some of the test scenarios. The figure is provided to highlight the sequence of MCPTT messages exchanged over the air, and thus can be used as a reference for the design of 3GPP working group, RAN5, test scenarios. Note that the information contained in a figure is a subset of information already included in the table of the same scenario. However, a figure provides a better visualization of the interaction between users and the timeline of events. For those test scenarios where there are none or very few over-the-air messages involved, a figure is not shown.

As an example, Table 1 is the corresponding table of the "Floor Release" test scenario with 2 User Equipments (UEs) in the MCPTT group call. The 1st row names the columns, which will be explained in more details later in this section. The 2nd row specifies the initial states of each UE when the test scenario starts. Starting from the 3rd row, the table lists a sequence of events organized chronologically, from the earliest (at the top of the table) to the latest (at the bottom of the table). Each row corresponds to an event/step of the test scenario, and there are 2 steps in this example, shown as 3rd and 4th row in Table 1.

Table 1: Floor release, 2 UEs (example)

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**O: has permission**	**O: has no permission**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
1	T0	**UE A** receives MCPTT indication to stop sending the Real-time Transport Protocol (RTP) packets (*No Queued Floor Requests at UE A*)	**O: silence**	**-**	**UE A** sends Floor Release, starts T230, stops T206 and T207 (*if running*), and clears Synchronization SouRCe (SSRC) of current arbitrator	**7.2.3.5.5**
2	T0 + t_AB1	**UE B** receives Floor Release	**-**	**O: silence**	**UE B** stops rendering RTP packets, stops T203 (*if running*), starts T230, may send Floor Idle notification to MCPTT user, and clears SSRC of current arbitrator	**7.2.3.4.3**

Each column of the table focuses on a specific aspect of the test scenario:

- The first unlabeled column numbers the rows in the table for easy referencing. The row labeled zero provides the initial starting states for each scenario. The row numbers provide the ordering of the events/steps in the scenario.
- “Timeline” is the timing of the event described in the row. T0 denotes the start time of every scenario, and the timing of each event is indicated in the form of a time increment in addition to the timing of the previous event wherever possible, e.g., T0 + t_AB1. The letters A, B or C in the time increment symbol indicate the main contributor of the time increment. t_ABx represents the transmission delay from UE A to UE B. t_BAx represents the transmission delay from UE B to UE A. t_Ax represents the time between T0 and the moment the stated action is performed at UE A. The “x” represents the ordered instance, e.g., t_AB1 is the first instance of the transmission delay between UE A and UE B and t_AB2 is the second instance of the transmission delay between UE A and UE B.
- “Triggering event” indicates the event that occurs at the time specified by “Timeline” of the same row. The triggering of such an event may result in a UE state transition, as well as consequent behavior, which are specified in other columns. A triggering event can be a timer expiration, a counter reaching its upper bound, the reception of a specific message (e.g., “UE B receives Floor Release”), the sending of RTP packets, etc. A triggering event may also include certain conditions that need to be met, e.g., “No Queued Floor Request at UE A” in row 3; such conditions shall be displayed between parentheses in an italic font.
- “UE A state transition” and “UE B state transition” indicate whether there are any changes in the UEs’ states during this event, and if yes, to which state. In Table 1, UE A starts in the ‘O: has permission’ state, and then transitions to the ‘O: silence’ state after the first event. UE B starts in the ‘O: has no permission’ state, and then transitions to the ‘O: silence’ state after two events.
- “Consequent behavior” lists the actions that shall be taken immediately as the consequence of the “Triggering event”. These actions focus on timer start/stop, counter maintenance,

the type of control messages to be generated, the indication to MCPTT user, the handling of RTP packets in general, and the update of certain key parameters. Further details of how to set control message fields, update other parameters or both can be found by referring to the corresponding clause(s) provided in column "Reference for details". Note that if more than one action is present per UE, they are separated by commas. If there are more than one UE, a semicolon is used to separate the list of actions for UE from the list of actions for another UE. If several UEs share the same set of action(s) (e.g., share the same "Reference for details" clause), they may be grouped together.

- "Reference for details" provides references to the corresponding clause(s) in 3GPP specifications which are used as the basis when designing the test scenario and which are associated with that particular event. For floor control, test scenarios are designed based on 3GPP TS 24.380 v14.3.0 [5]. For call control and call type control, test scenarios are designed based on 3GPP TS 24.379 v14.2.0 [4]. These references are listed in the same order as that of the UEs described in the "Consequent behavior" section.

Additionally, throughout the table, several other color and font schemes are used:

- A UE represented in a bold font (such as "**UE A**") indicates that this UE is the main protagonist at play in a given cell description: in a "Triggering event" column, that means the embolden UE is the one for which we expect an action to occur before the scenario continues to unfold; in a "Consequent behavior" column, that means the text description represents the behavior for that embolden UE.
- A blue-font text indicates an MCPTT message (such as "Floor Request")
- A green-font text indicates an MCPTT timer or counter (such as "T205" or "C205", respectively)
- Supplemental conditions and/or explanations can be encountered in various columns (such as already mentioned in the "Triggering event" column) by being displayed between parentheses using an italic font.

As another example, Fig. 1 is the figure corresponding to test scenario "Floor request – deny". The interpretation of a figure follows the following practice:

- State transitions of UEs, e.g., UE A and UE B in Fig. 1, are shown on the vertical axes.
- Events that occur during the tests scenario are organized temporally in a "top-to-bottom" fashion.
- Directional arrows between MCPTT clients represent messages exchanged between network entities, with the message name labeled. Arrows with solid lines indicate that messages are received successfully, whereas dotted lined arrows represent messages that cannot be received successfully by the MCPTT entity at the receiving UE, e.g., due to channel loss.
- Additional information may be included as well, such as timer start/stop, message preparation. Those action descriptions follow a color code: red for UE A, blue for UE B and orange for UE C.

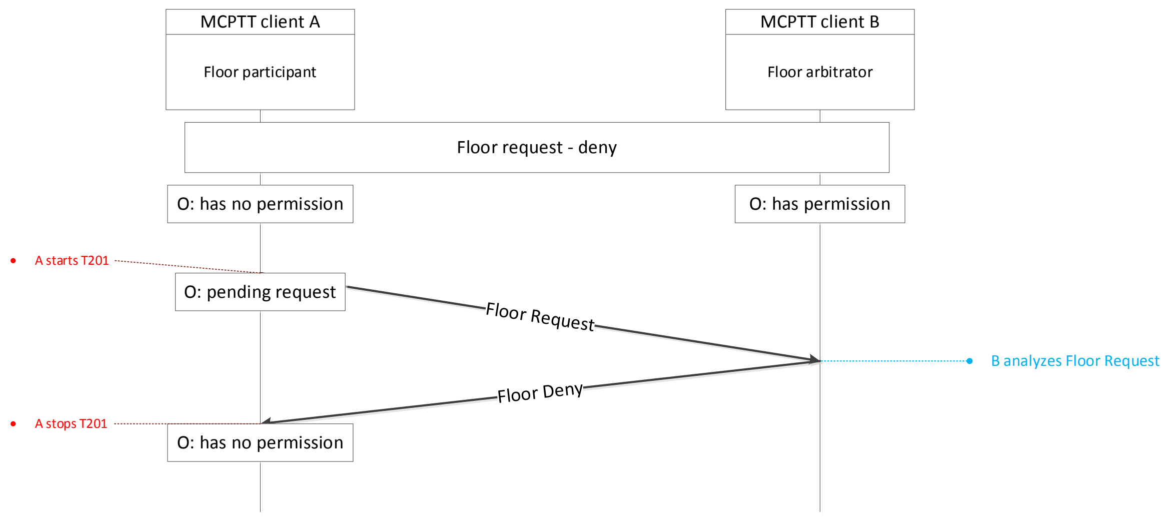

Figure 1: Floor request – deny, 2 UEs (example)

2. Test scenarios details – basic group call

A brief overview of test scenarios to be discussed in this section is summarized by Table 2:

Table 2: Test scenarios overview

Subject	Sub-area	Number of scenarios	Reference 3GPP specifications
Basic Group Call – Floor Control	Floor Request	5	TS 24.380 v14.3.0
	Floor Release	4	
	Session Initialization	4	
	Session Release	1	
Basic Group Call – Call Control	Call Setup	3	TS 24.379 v14.2.0
	Call Merge	1	
	Call Release	7	
	Call Reject	1	
Basic Group Call – Call Type Control	Call Type Initialization	10	TS 24.379 v14.2.0
	Call Type Upgrade	3	
	Call Type Downgrade	4	
	Call Release	2	
	Call Merge	2	
Broadcast Group Call – Call Control	Call Setup	2	TS 24.379 v14.2.0
	Call Release	2	
Private Call – Call Control	Call Setup	5	TS 24.379 v14.2.0
	Call Cancellation	2	
	Call Expiration	1	
Private Call – Call Type Control	Enter Private Call	1	TS 24.379 v14.2.0
	Enter Private Emergency Call	1	
	Private Call Upgrade	3	
	Private Call Downgrade	3	

2.1. Basic group call – floor control

Floor control is the arbitration system in an MCPTT Service that determines who has the authority to transmit (talk) at a point in time during an MCPTT call [2].
The state transitions that are analyzed in each test scenario of Section 2.1 are based on Fig. 7.2.3.1-1 of [5]. Floor control test scenarios can be categorized into 4 groups: floor request, floor release, session initialization, and session release.

2.1.1. Floor request

2.1.1.1. Floor request – idle

This scenario describes the process of requesting the floor, with all (e.g., 3) participants idle, i.e., beginning in the 'O: silence' state. In this scenario, no other user intends to take the floor, except the one (UE A), which sends the floor request message. After multiple rounds of sending floor request messages, UE A assumes the floor. The other UEs' actions to the received floor control messages are shown.

Scenario Purpose: To observe that one and only one UE will become floor arbitrator and begin transmitting (talking) when there is currently no active floor arbitrator and the details of the protocol that allow this to happen.

Table 3: Floor request – idle

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**O: silence**	**O: silence**	**O: silence**	-	-
1	T0	**UE A** pushes PTT button	**O: pending request**	-	-	**UE A** sends Floor Request, stops T230, starts T201 and initializes C201	**7.2.3.3.2**
2	T0 + t_AB1	**UE B** and **UE C** receive Floor Request	-	-	-	**UE B** and **UE C** discard the floor control message	**7.2.3.1** *(7.2.3.3.5 applicable to private call only)*
3	T0 + T201	**UE A** T201 expires	-	-	-	**UE A** resends Floor Request, starts T201 and increments C201	**7.2.3.6.9**
4	T0 + T201 + t_AB2	**UE B** and **UE C** receive Floor Request	-	-	-	**UE B** and **UE C** discard the floor control message	**7.2.3.1** *(7.2.3.3.5 applicable to private call only)*
5	T0 + 2*T201	**UE A** T201 expires	-	-	-	**UE A** resends Floor Request, and starts T201 and increments C201	**7.2.3.6.9**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
6	T0 + (n - 1) * T201 + t_ABn	**UE B** and **UE C** receive Floor Request	-	-	-	**UE B** and **UE C** discard the floor control message	**7.2.3.1** *(7.2.3.3.5 applicable to private call only)*
7	T0 + n*T201	**UE A** T201 expires (*C201 at upper limit*)	**O: has permission**	-	-	**UE A** sends Floor Taken	**7.2.3.6.6**
8	T0 + n*T201 + t_AB (n+1)	**UE B** and **UE C** receive Floor Taken	-	**O: has no permission**	**O: has no permission**	**UE B** and **UE C** stop T230, start T203 and set SSRC of granted floor arbitrator	**7.2.3.3.6**

And the corresponding figure is Fig. 2:

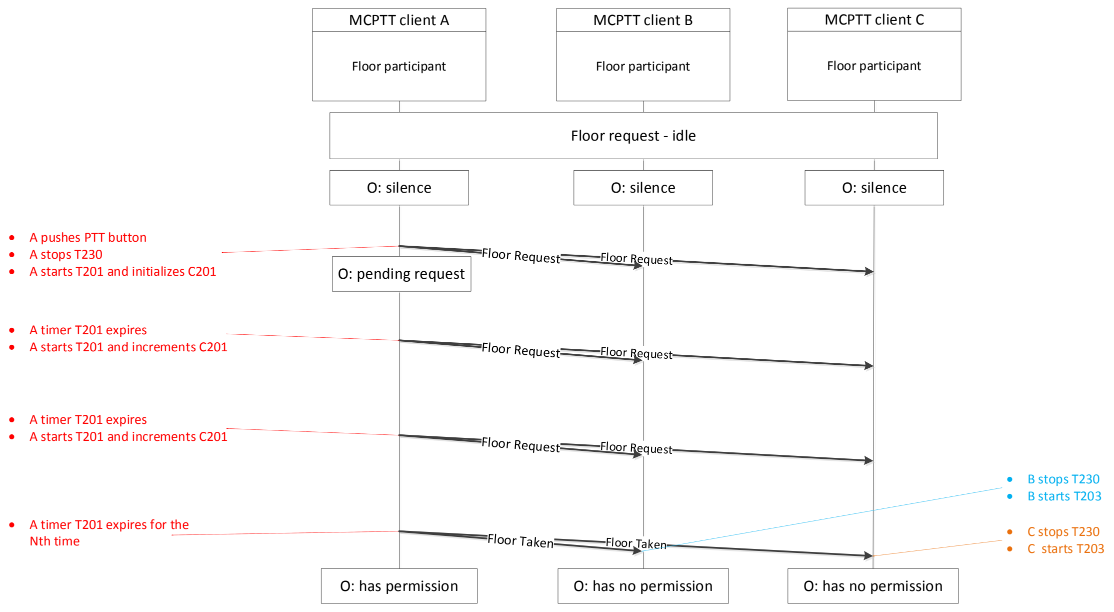

Figure 2: Floor request – idle

2.1.1.2.Floor request – idle multiple floor requests

This scenario describes the process of requesting the floor, in an alternative way compared to the previous scenario: all three participants begin in the ‘O: silence’ state and are at first not interested in taking the floor, hence transitioning to the ‘O: start-stop’ state. After a while, two users – UE A and UE C – declare their intention to take the floor.

Scenario Purpose: To observe that a UE will become floor arbitrator when there is currently no active floor arbitrator and when two UEs requested the floor at a close time; to also observe that after being granted the floor and once done talking, the floor arbitrator tries to pass on the floor to the other queued user.

Table 4: Floor request – idle multiple floor requests

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**O: silence**	**O: silence**	**O: silence**	-	-
1	T0	**UE A** T230 expires	**O: start-stop**	-	-	**UE A** terminates the floor participant state transition diagram	**7.2.3.3.7**
2	T0 + t_B1	**UE B** T230 expires	-	**O: start-stop**	-	**UE B** terminates the floor participant state transition diagram	**7.2.3.3.7**
3	T0 + t_C1 (*with t_C1 > t_B1*)	**UE C** T230 expires	-	-	**O: start-stop**	**UE C** terminates the floor participant state transition diagram	**7.2.3.3.7**
4	T0 + t_C2 (*with t_C2 > t_C1*)	**UE C** presses PTT button	-	-	**O: pending request**	**UE C** creates an instance of floor participant, sends Floor Request, starts T201 and initializes C201	**7.2.3.2.5**
5	T0 + t_C2 + t_CA1	**UE A** and **UE B** receive Floor Request	-	-	-	**UE A** and **UE B** discard the floor control message	**7.2.3.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
6	T0 + t_A1 (*with t_A1 > t_C2 + t_CA1*)	**UE A** presses PTT button	**O: pending request**	-	-	**UE A** creates an instance of floor participant, sends Floor Request, starts T201 and initializes C201	**7.2.3.2.5**
7	T0 + t_A1 + t_AB1	**UE B** and **UE C** receive Floor Request	-	-	-	**UE B** discards the floor control message; **UE C** restarts T201 and resets C201	**7.2.3.1 7.2.3.6.10**
8	T0 + t_A1 + T201	**UE A** T201 expires	-	-	-	**UE A** sends Floor Request, starts T201 and increments C201	**7.2.3.6.9**
9	T0 + t_A1 + T201 + t_AB2 +	**UE B** and **UE C** receive Floor Request	-	-	-	**UE B** discards the floor control message; **UE C** restarts T201 and resets C201	**7.2.3.1 7.2.3.6.10**
10	T0 + t_A1 + n*T201	**UE A** T201 expires (*upper limit of C201 reached*)	**O: has permission**	-	-	**UE A** sends Floor Taken	**7.2.3.6.6**
11	T0 + t_A1 + n*T201 + t_AB3	**UE B** and **UE C** receive Floor Taken	-	**O: has no permission**	-	**UE B** creates an instance of floor participant, sets SSRC of current floor arbitrator to SSRC of UE A, and starts T203; **UE C** sets SSRC of current floor arbitrator to SSRC of UE A, restarts T201 and resets C201	**7.2.3.2.6 7.2.3.6.11**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
12	T0 + t_A1 + n*T201 + t_AB3 + T201	**UE C** T201 expires	-	-	-	**UE C** sends Floor Request, starts T201 and increments C201	**7.2.3.6.9**
13	T0 + t_A1 + n*T201 + t_AB3 + T201+ t_CA2	**UE A** and **UE B** receive Floor Request	-	-	-	**UE A** stores the request and sends Floor Queue Position Info; **UE B** discards the floor control message	**7.2.3.5.4** **7.2.3.1**
14	T0 + t_A1 + n*T201 + t_AB3 + T201+ t_CA2 + t_AB4	**UE B** and **UE C** receive Floor Queue Position Info	-	-	**O: queued**	**UE B** discards the floor control message; **UE C** updates the queue status, and stops T201	**7.2.3.1** **7.2.3.6.3**
15	T0 + t_A2 (*with t_A2 > t_A1 + n*T201 + t_AB3 + T201+ t_CA2 + t_AB4*)	**UE A** sends RTP media	-	-	-	**UE A** starts T206	**7.2.3.5.2**
16	T0 + t_A2 + t_AB5	**UE B** and **UE C** receive RTP media	-	-	-	**UE B** renders RTP packets, starts T203; **UE C** renders RTP media, and starts T203	**7.2.3.4.6** **7.2.3.8.2**
17	T0 + t_C3 (*with t_C3 > t_A2 + t_AB5*)	**UE C** requests the queue position information	-	-	-	**UE C** sends Floor Queue Position Request, starts T204 and initializes C204	**7.2.3.8.11**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
18	T0 + t_C3 + t_CA3	**UE A** and **UE B** receive Floor Queue Position Request	-	-	-	**UE A** sends Floor Queue Position Info; **UE B** discards the floor control message	**7.2.3.5.8 7.2.3.1**
19	T0 + t_C3 + t_CA3 + t_AB6	**UE B** and **UE C** receive Floor Queue Position Info	-	-	-	**UE B** discards the floor control message; **UE C** updates the queue position, and stops T204	**7.2.3.1 7.2.3.8.3**
20	T0 + t_A3 (*with t_A3 > t_C3 + t_CA3 + t_AB6*)	**UE A** stops sending RTP media	**O: pending granted**	-	-	**UE A** stops T206 and T207, sends Floor Granted to UE C, sets stored SSRC of current floor arbitrator to SSRC of UE C, starts T205 and initializes C205	**7.2.3.5.6**
21	T0 + t_A3 + t_AB7	**UE B** and **UE C** receive Floor Granted	-	-	-	**UE B** stops rendering RTP packets, sets stored SSRC of candidate floor arbitrator to SSRC of UE C, and starts T203; **UE C** stops rendering RTP packets, notifies the user about the floor grant, and starts T233	**7.2.3.4.5 7.2.3.8.6**
22	T0 + t_A3 + t_AB7 + T233	**UE C** T233 expires	-	-	**O: silence**	**UE C** starts T230	**7.2.3.8.7**

And the corresponding figure is Fig. 3:

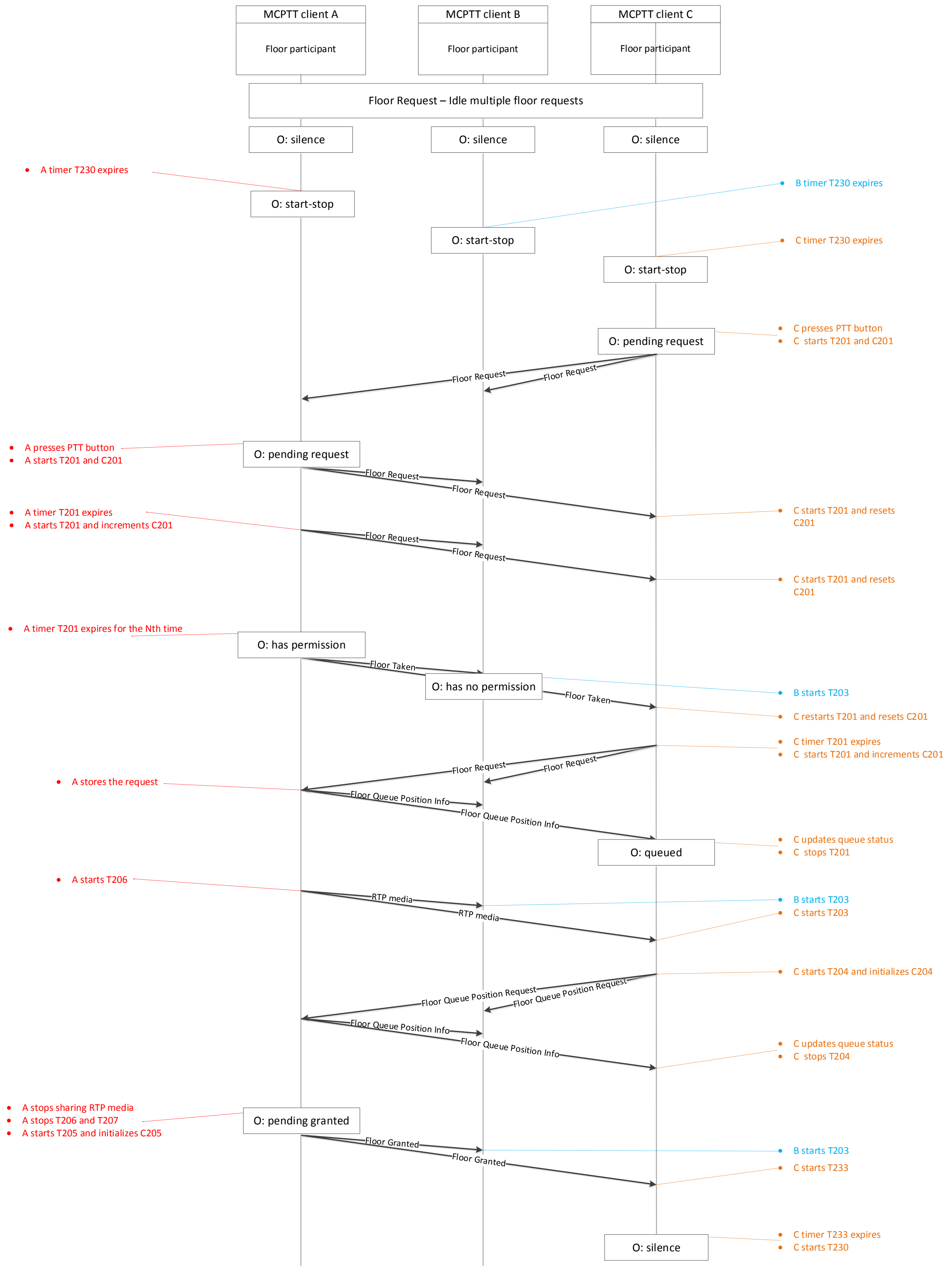

Figure 3: Floor request – idle multiple floor requests

2.1.1.3.Floor request – queued

This scenario describes another possible process of requesting the floor, with participants beginning in the 'O: has permission' and the 'O: has no permission' state. Queuing is supported and used in this scenario: a floor participant will ask for the floor, wait in queue and finally be granted the floor. For the first few steps of this scenario, it is assumed that UE B does not receive the various floor request messages until a later time (T0 + 4*T201 + t_AB5). In this scenario, in addition to the multiple transmissions and receptions of floor request messages, several timer expirations (timers T201 and T205) are observed.

Scenario Purpose: To observe that a floor request is properly handled by the floor arbitrator. That is the floor request is queued and later the floor is granted.

Table 5: Floor request – queued

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**O: has no permission**	**O: has permission**	**O: has no permission**	-	-
1	T0	**UE A** pushes PTT button	**O: pending request**	-	-	**UE A** sends Floor Request, starts T201 and initializes C201	**7.2.3.4.2**
2	T0 + T201	**UE A** T201 expires	-	-	-	**UE A** resends Floor Request, starts T201 and increments C201	**7.2.3.6.9**
3	T0 + 2*T201	**UE A** T201 expires	-	-	-	**UE A** resends Floor Request, starts T201 and increments C201	**7.2.3.6.9**
4	T0 + t_B1 (*with t_B1 > 2*T201*)	**UE B** sends RTP packets	-	-	-	**UE B** starts T206	**7.2.3.5.2**
5	T0 + t_B1 + t_BA1	**UE A** and **UE C** receive RTP packets	-	-	-	**UE A** restarts T203, and resets C201; **UE C** restarts T203	**7.2.3.6.2** **7.2.3.4.6**
6	T0 + 3*T201	**UE A** T201 expires	-	-	-	**UE A** sends Floor Request, and starts T201 and increments C201	**7.2.3.6.9**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
7	T0 + 4*T201	**UE A** T201 expires	-	-	-	**UE A** sends Floor Request, starts T201 and increments C201	**7.2.3.6.9**
8	T0 + 4*T201 + t_AB1	**UE B** receives Floor Request	-	-	-	**UE B** stores the request, and sends Floor Queue Position Info	**7.2.3.5.4**
9	T0 + 4*T201 + t_AB1 + t_BA2	**UE A** and **UE C** receive Floor Queue Position Info	**O: queued**	-	-	**UE A** updates the queue status and stops T201	**7.2.3.6.3**
10	T0 + t_B2 (*with + 4*T201 + t_AB1 + t_BA2*)	**UE B** sends RTP packets	-	-	-	**UE B** starts T206	**7.2.3.5.2**
11	T0 + t_B2 + t_BA3	**UE A** and **UE C** receive RTP packets	-	-	-	**UE A** restarts T203; **UE C** restarts T203	**7.2.3.8.2 7.2.3.4.6**
12	T0 + t_B3 (*with t_B3 > t_B2 + t_BA3*)	**UE B** releases PTT button	-	**O: pending granted**	-	**UE B** stops sending RTP packets, sends Floor Granted, stops T206 and T207 (*if running*), sets SSRC of current arbitrator to SSRC of user (UE A) to whom the floor was granted, starts T205 and initializes C205 (*the upper limit value of C205 is 3*)	**7.2.3.5.6**
13	T0 + t_B3 + t_BA4	**UE A** and **UE C** receive Floor Granted	-	-	-	**UE A** starts T233, and shall notify the MCPTT user of Floor Grant; **UE C** restarts T203 and sets the stored SSRC of the current arbitrator to the SSRC of UE A	**7.2.3.8.6 7.2.3.4.5**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
14	T0 + t_B3 + T205	**UE B** T205 expires	-	-	-	**UE B** sends Floor Granted, restarts T205 and increments C205	**7.2.3.7.3**
15	T0 + t_B3 + T205 + t_BA5	**UE A** and **UE C** receive Floor Granted for the 2nd time	-	-	-	**UE A** shall notify the MCPTT user of Floor Grant; **UE C** restarts T203	**7.2.3.8.6** **7.2.3.4.5**
16	T0 + t_B3 + 2*T205	**UE B** T205 expires (*for the second time*)	-	-	-	**UE B** sends Floor Granted, restarts T205 and increments C205	**7.2.3.7.3**
17	T0 + t_B3 + 2*T205 + t_BA6	**UE A** and **UE C** receive Floor Granted	-	-	-	**UE A** shall notify the MCPTT user of Floor Grant; **UE C** restarts T203	**7.2.3.8.6** **7.2.3.4.5**
18	T0 + t_B3 + 3*T205	**UE B** T205 expires (*the upper limit value of C205 is 3*)	-	-	-	**UE B** starts T233, resets C205	**7.2.3.7.4**
19	T0 + t_A1 (*with t_A1 > t_B3 + 2*T205 + t_BA6*)	**UE A** pushes PTT button	**O: has permission**	-	-	**UE A** stops T233	**7.2.3.8.8**
20	T0 + t_A2 (*with t_A2 > t_A1*)	**UE A** sends RTP packets	-	-	-	**UE A** starts T206	**7.2.3.5.2**
21	T0 + t_A2 + t_AB2	**UE B** and **UE C** receive RTP packets	-	**O: has no permission**	-	**UE B** starts T203, and stops T233 and T205; **UE C** restarts T203	**7.2.3.7.2** **7.2.3.4.6**

And the corresponding figure is Fig. 4:

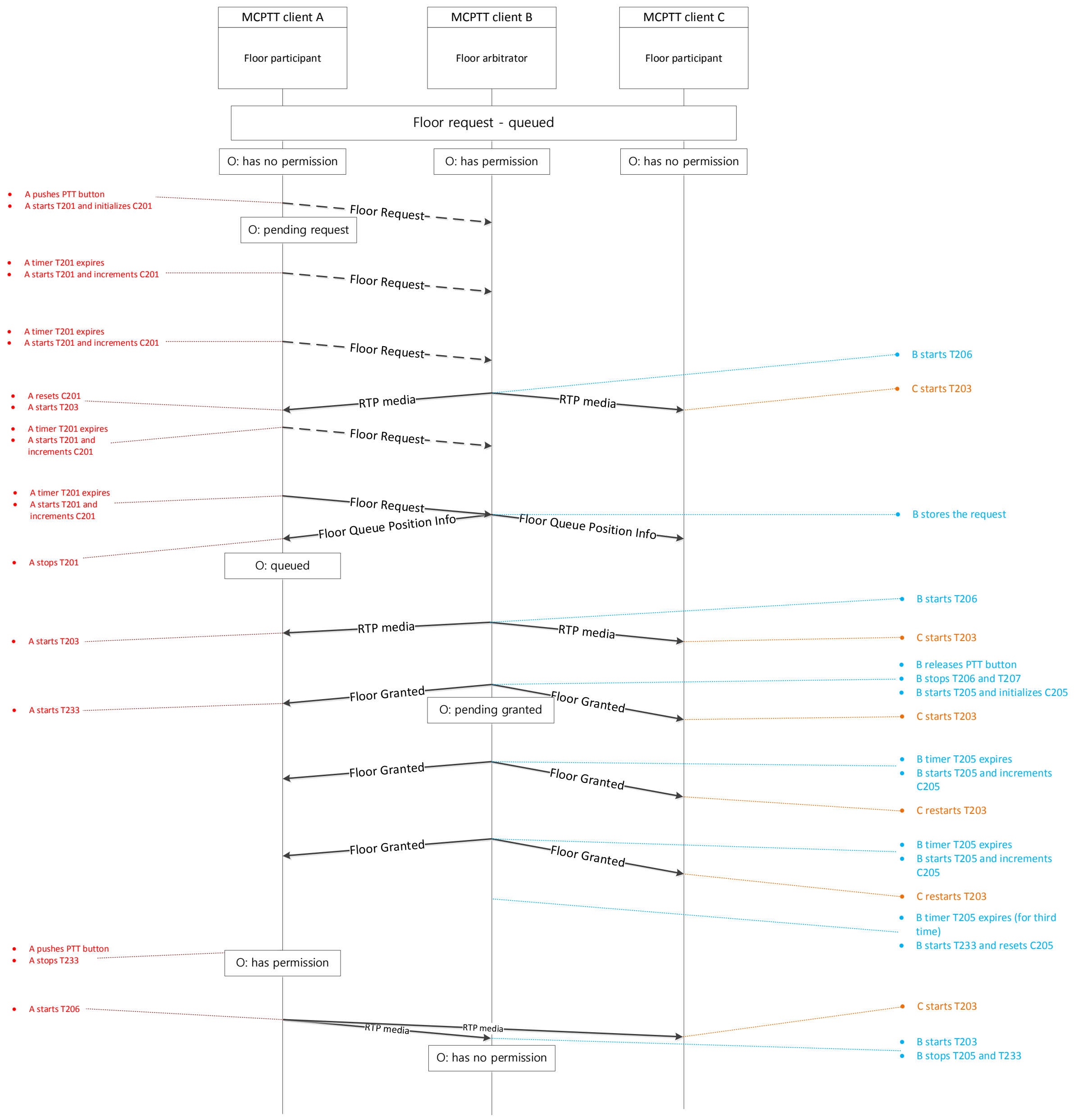

Figure 4: Floor request – queued

2.1.1.4.Floor request – denied

This scenario describes yet another possible outcome of requesting the floor, with participants beginning in the 'O: has permission' and the 'O: has no permission' state. Queuing is not supported here or the queue is already full or that the Floor Request message does not contain information that queuing is supported, so the floor participant asking for the floor is denied permission to take the floor and because the floor request is not pre-emptive.

Scenario Purpose: To observe that the floor request from UE A is denied by the floor arbitrator and UE A makes no further floor requests.

Table 6: Floor request – denied

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**O: has no permission**	**O: has permission**	**O: has no permission**	-	-
1	T0	**UE A** pushes PTT button	**O: pending request**	-	-	**UE A** sends Floor Request, starts T201 and initializes C201	**7.2.3.4.2**
2	T0 + t_AB1	**UE B** and **UE C** receive Floor Request	-	-	-	**UE B** analyzes Floor Request, generates and sends Floor Deny; **UE C** discards the floor control message	**7.2.3.5.4 7.2.3.1**
3	T0 + t_AB1 + t_BA1	**UE A** and **UE C** receive Floor Deny	**O: has no permission**	-	-	**UE A** provides Floor Deny notification to the user, stops T201, and starts T203; **UE C** discards the floor control message	**7.2.3.6.4 7.2.3.1**

And the corresponding figure is Fig. 5:

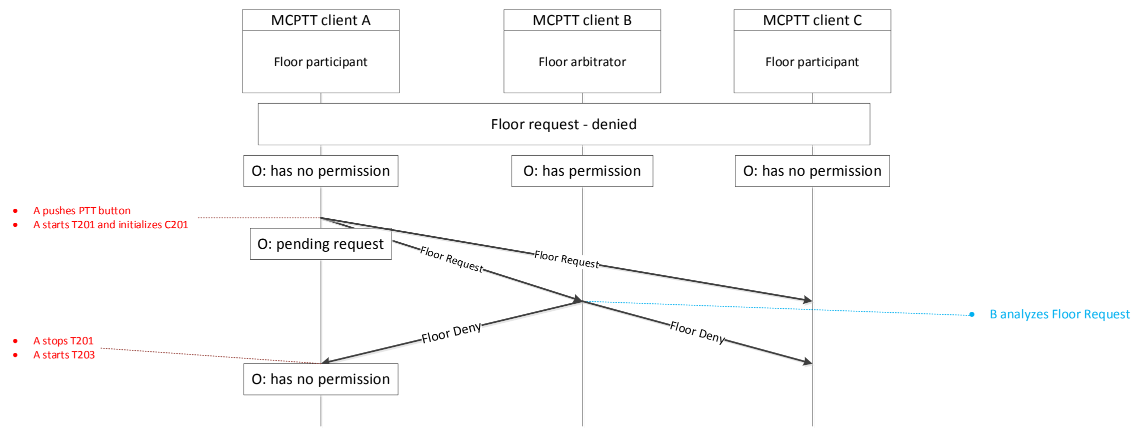

Figure 5: Floor request – denied

2.1.1.5.Floor request – pre-emptive

This scenario describes the process of requesting the floor, when the user requesting the floor has a higher floor priority than the current speaker.

Scenario Purpose: To observe that the current floor arbitrator relinquishes the floor when another user requests the floor with a higher floor priority (pre-emptive).

Table 7: Floor request – pre-emptive

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0			**O: has no permission**	**O: has permission**	**O: has no permission**		
1	T0	**UE B** sends RTP packet	-	-	-	**UE B** starts T206	**7.2.3.5.2**
2	T0 + t_BA1	**UE A** and **UE C** receive RTP packets	-	-	-	**UE A** and **UE C** start T203	**7.2.3.4.6**
3	T0 + t_B1	**UE B** sends RTP packet	-	-	-	**UE B** starts T206	**7.2.3.5.2**
4	T0 + t_B1 + t_BA2	**UE A** and **UE C** receive RTP packet	-	-	-	**UE A** and **UE C** start T203	**7.2.3.4.6**
5	T0 + t_A1 (*with t_A1 > t_B1 + t_BA2*)	**UE A** receives indication from MCPTT user that the user wants to send media	**O: pending request**	**-**	**-**	**UE A** sends Floor Request, starts T201 and initializes C201	**7.2.3.4.2**
6	T0 + t_A1 + t_AB1	**UE B** and **UE C** receive Floor Request	-	**O: pending granted**	**-**	**UE B** determines UE A's pre-emption priority is higher than its own priority, stops sending RTP packets, stops T206 and T207(*if running*), sends Floor Granted, starts T205 and initializes C205; **UE C** discards the floor control message	**7.2.3.5.7** **7.2.3.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
7	T0 + t_A1 + t_AB1 + t_BA3	**UE A** and **UE C** receive Floor Granted	**O: has permission**	-	-	**UE A** stops rendering RTP packets**,** stops T201 and T203, and may provide floor granted notification to MCPTT user; **UE C** stops rendering RTP packets, restarts T203, sets the stored SSRC of candidate arbitrator to the SSRC of UE A, and may provide floor taken notification	**7.2.3.6.7 7.2.3.4.5**
8	T0 + t_A1 + t_AB1 + T205	**UE B** T205 expires	-	-	-	**UE B** resends Floor Granted, restarts T205 and increments C205	**7.2.3.7.3**
9	T0 + t_A1 + t_AB1 + T205 + t_BA4	**UE A** and **UE C** receive Floor Granted	-	-	-	**UE A** discards the floor control message; **UE C** checks if SSRC of Floor Participant (UE B) sending the Floor Granted matches SSRC of current arbitrator (UE A)	**7.2.3.1 7.2.3.4.5**
10	T0 + t_A2 (*with t_A2 > t_A1 + t_AB1 + T205 + t_BA4*)	**UE A** sends RTP packets	-	-	-	**UE A** starts T206	**7.2.3.5.2**
11	T0 + t_A2 + t_AB2	**UE B** and **UE C** receive RTP Packets	-	**O: has no permission**	-	**UE B** stops T205 and T233, and starts T203; **UE C** starts T203	**7.2.3.7.2 7.2.3.4.6**

And the corresponding figure is Fig. 6:

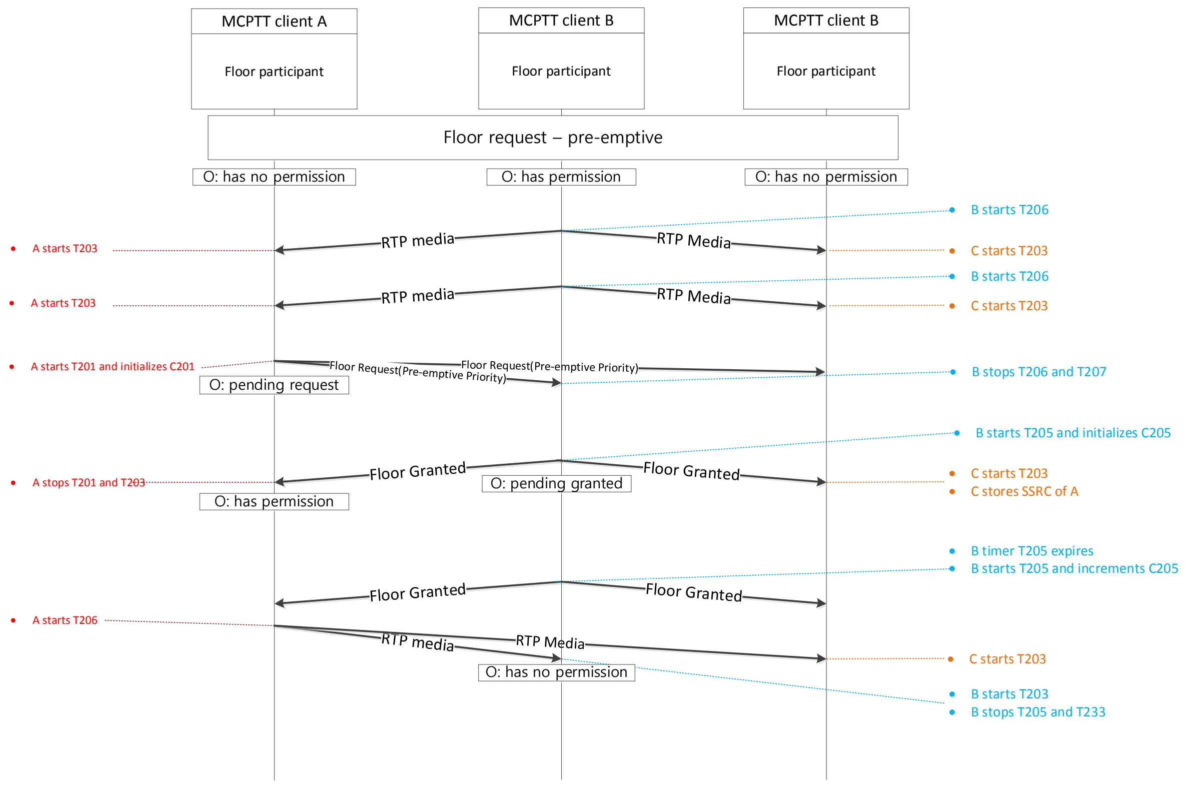

Figure 6: Floor request – pre-emptive

2.1.2. Floor release

2.1.2.1.Floor release by the floor arbitrator

In this scenario, one user has the permission to talk (floor arbitrator), while the other one does not. The floor arbitrator (UE A) stops talking and releases the floor and there is no other user waiting in the queue. It is assumed in this scenario that the floor participant (UE B) receives Floor Release message before the expiry of its own T203 timer.

Scenario Purpose: To observe that when the current arbitrator stops transmitting with an empty queue, that it sends a Floor Release message and that the participants go back to the "O: silence" state.

Table 8: Floor release by the floor arbitrator

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**O: has permission**	**O: has no permission**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
1	T0	**UE A** sends RTP packet	-	-	**UE A** starts T206	**7.2.3.5.2**
2	T0 + t_AB1	**UE B** receives RTP packet	-	-	**UE B** restarts T203	**7.2.3.4.6**
3	T0 + t_A1 (*with t_A1 > t_AB1*)	**UE A** receives MCPTT indication to stop sending RTP packets	**O: silence**	-	**UE A** sends Floor Release, starts T230, stops T206 and T207 (*if running*), and clears SSRC of current arbitrator	**7.2.3.5.5**
4	T0 + t_A1 + t_AB2 (*with (t_A1+t_AB2) < (t_AB1+T203)*)	**UE B** receives Floor Release	-	**O: silence**	**UE B** stops rendering RTP packets, stops T203 and starts T230, may send floor idle notification to MCPTT user, and clears SSRC of current arbitrator	**7.2.3.4.3**

2.1.2.2.Floor release by a queued floor participant

In this scenario, one user has permission to talk, and another is queued. The queued user removes his request at the beginning of the test scenario.

Scenario Purpose: To observe that a queued user can cancel its floor request during a call.

Table 9: Floor release by a queued floor participant

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**O: queued**	**O: has permission**	**O: has no permission**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE A** receives MCPTT indication to release the pending request	**O: has no permission**	-	-	**UE A** sends Floor Release, stops T233 (*if running*)	**7.2.3.8.5**
2	T0 + t_AB1	**UE B** and **UE C** receive Floor Release	-	-	-	**UE B** removes UE A from the queue; **UE C** discards the floor control message	**7.2.3.5.3** **7.2.3.1**

2.1.2.3.Floor release by the floor arbitrator with queued participants

In this scenario, one user has the permission to talk (floor arbitrator), while the other ones are waiting in the queue. The first floor arbitrator stops talking, releases the floor, and grants the floor to the first floor participant in the queue. The new floor arbitrator takes the floor and talks, then stops talking for quite a while without releasing floor explicitly. After a timer (T203) expires, other floor participants assume there is no floor arbitrator.

Scenario Purpose: To observe that a floor arbitrator can release the medium and that the floor is correctly given to the next queued user, who becomes the new floor arbitrator. When the new floor arbitrator stops talking but does not release the floor explicitly, observe that the other floor participants assume that there is no longer a floor arbitrator (i.e., T203 expires).

Table 10: Floor release by the floor participant with queued participants

	Timeline	Triggering event	UE A state transition	UE B State transition	UE C State transition	Consequent behavior	Reference for details
0	-	-	**O: queued**	**O: has permission**	**O: queued**	-	-

	Timeline	Triggering event	UE A state transition	UE B State transition	UE C State transition	Consequent behavior	Reference for details
1	T0	**UE B** releases PTT button	-	**O: pending granted**	-	**UE B** stops sending RTP packets, sends Floor Granted (*for UE A*), stops T206 and T207 (*if running*), sets SSRC of current arbitrator to SSRC of user to whom the floor was granted (UE A), starts T205 and initializes C205	**7.2.3.5.6**
2	T0 + t_BA1	**UE A** and **UE C** receive Floor Granted	-	-	-	**UE A** stops rendering RTP packets, starts T233 (*if not running*), shall notify the MCPTT user of Floor Grant; **UE C** stops rendering RTP packets, restarts T203 and stores SSRC of candidate arbitrator to SSRC of UE A	**7.2.3.8.6** **7.2.3.8.9**
3	T0 + t_A1 (*with t_A1 > t_BA1*)	**UE A** pushes PTT button	**O: has permission**	-	-	**UE A** stops T233	**7.2.3.8.8**
4	T0 + t_A2 (*with t_A2 > t_A1*)	**UE A** sends RTP packets	-	-	-	**UE A** starts T206	**7.2.3.5.2**
5	T0 + t_A2 + t_AB1	**UE B** and **UE C** receive RTP packets	-	**O: has no permission**	-	**UE B** stops T205 and starts T203; **UE C** restarts T203	**7.2.3.7.2** **7.2.3.8.2**

	Timeline	Triggering event	UE A state transition	UE B State transition	UE C State transition	Consequent behavior	Reference for details
6	T0 + t_A2 + t_AB1 + T203	**UE C** timer T203 expires	-	-	**O: pending request**	**UE C** stops rendering RTP packets, clears SSRC of floor arbitrator, sends Floor Request, starts T201 and initializes C201	**7.2.3.8.10**
7	T0 + t_A2 + t_AB1 + T203	**UE B** timer T203 expires	-	**O: silence**	-	**UE B** stops rendering RTP packets, clears stored SSRC of current arbitrator, starts T230	**7.2.3.4.4**

Note: Due to the independence (not synchronized) of the timers T203 in the UEs, the ordering of step 6 and step 7 may be reversed. We intentionally terminate the scenario at this point and do not follow through with the reception of UE C's Floor Request message by UE A or UE B.

2.1.2.4.Floor release by pre-empted floor arbitrator

In this scenario, two users request the floor. Queuing is not supported. The floor is pre-emptively granted to one of them, who shortly afterwards releases it (before sending any RTP media).

Scenario Purpose: To observe that a user can pre-emptively request the floor and have it granted to him, that a new user tries to request it as well, and then finally that the floor arbitrator eventually releases the floor (before sending RTP media).

Table 11: Floor release by pre-empted floor arbitrator

	Timeline	Triggering Event	UE A state transition	UE B State transition	UE C State transition	Consequent behavior	Reference for details
0	-	-	**O: has permission**	**O: has no permission**	**O: has no permission**	-	-
1	T0	**UE B** pushes PTT button	-	**O: pending request**	-	**UE B** sends Floor Request, starts T201 and initializes C201	**7.2.3.4.2**

	Timeline	Triggering Event	UE A state transition	UE B State transition	UE C State transition	Consequent behavior	Reference for details
2	T0 + t_BA1	**UE A** and **UE C** receive Floor Request	**O: pending granted**	-	-	**UE A** stops sending RTP packets, sends Floor Granted, stops T206 and T207, starts T205 and initializes C205; **UE C** discards the floor control message	**7.2.3.5.7 7.2.3.1**
3	T0 + t_BA1 + t_AB1	**UE B** and **UE C** receive Floor Granted	-	**O: has permission**	-	**UE B** stops rendering RTP packets, and stops T201 and T203; **UE C** stops rendering RTP packets, sets stored SSRC of candidate floor arbitrator to SSRC of UE B, and starts T203	**7.2.3.6.7 7.2.3.4.5**
4	T0 + t_C1 (*with t_C1 > t_BA1 + t_AB1*)	**UE C** pushes PTT button	-	-	**O: pending request**	**UE C** sends Floor Request, starts T201 and initializes C201	**7.2.3.4.2**
5	T0 + t_C1 + t_CA1	**UE A** and **UE B** receive Floor Request	-	-	-	**UE A** sends Floor Deny; **UE B** sends Floor Deny	**7.2.3.7.10 7.2.3.5.4**
6	T0 + t_C1 + t_CA1 + t_AB2	**UE B** and **UE C** receive Floor Deny (*from* ***UE A***)	-	-	**O: has no permission**	**UE B** discards the floor control message; **UE C** stops T201, and starts T203	**7.2.3.1 7.2.3.6.4**
7	T0 + t_C1 + t_CA1 + t_BA2 (*with t_BA2 > t_AB2*)	**UE A** and **UE C** receive Floor Deny (*from* ***UE B***)	-	-	-	**UE A** and **UE C** discard the floor control message	**7.2.3.1**

	Timeline	Triggering Event	UE A state transition	UE B State transition	UE C State transition	Consequent behavior	Reference for details
8	T0 + t_B1 (*with t_B1 > max (t_C1 + t_CA1 + t_AB2 ; t_C1 + t_CA1 + t_BA2*))	**UE B** releases PTT button	-	**O: silence**	-	**UE B** clears stored SSRC of current arbitrator, sends Floor Release, stops T206 and T207, and starts T230	**7.2.3.5.5**
9	T0 + t_B1 + t_BA3	**UE A** and **UE C** receive Floor Release	-	-	**O: silence**	Since queuing is not supported, **UE A** remains in its current state; **UE C** clears SSRC of current floor arbitrator and candidate arbitrator, stops T203, and starts T230	**7.2.3.7.9** **7.2.3.4.3**
10	T0 + t_BA1 + T205	**UE A** T205 expires	-	-	-	**UE A** sends Floor Granted, starts T205 and increments C205	**7.2.3.7.3**
11	T0 + t_BA1 + T205 + t_AB3	**UE B** and **UE C** receive Floor Granted	-	-	**O: has no permission**	**UE B** discards the floor control message; **UE C** sets SSRC of candidate floor arbitrator to SSRC of UE B, stops T230, and starts T203	**7.2.3.1** **7.2.3.3.4**
12	T0 + t_BA1 + n*T205	**UE A** T205 expires (*C205 at its upper limit*)	**O: silence**	-	-	**UE A** clears SSRC of current floor arbitrator, starts T230, and resets C205	**7.2.3.7.5**
13	T0 + t_BA1 + T205 + t_AB3 + T203	**UE C** T203 expires	-	-	**O: silence**	**UE C** stops rendering RTP packets, clears SSRC of current floor arbitrator, and starts T230	**7.2.3.4.4**

And the corresponding figure is Fig. 7:

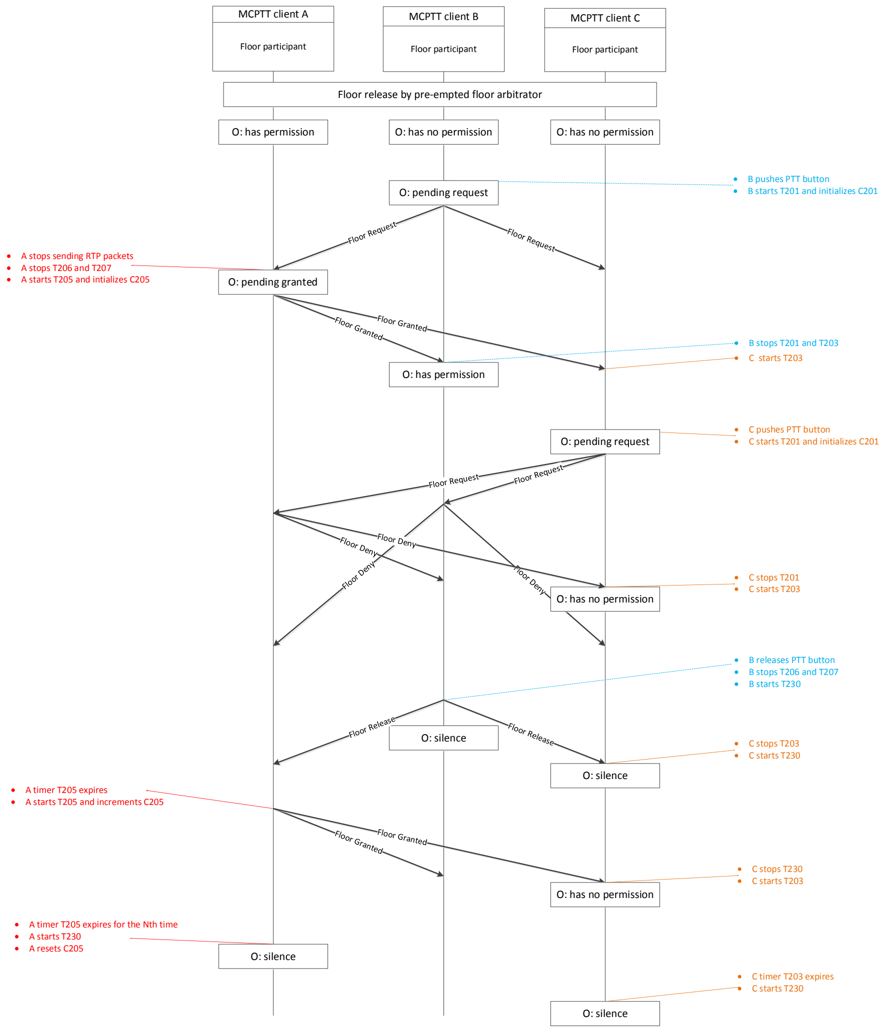

Figure 7: Floor release by pre-empted floor arbitrator

2.1.3. Session initialization

2.1.3.1. Session initialization – normal

This scenario focuses on the initialization of the MCPTT session. It covers the creation of the floor control entity and the sending of the first data packet after the call starts.

Scenario Purpose: To observe that the originating MCPTT call participant becomes arbitrator (Floor Grant) and starts transmitting (RTP packets).

Table 12: Session initialization – normal

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
0			**Start-stop**	**Start-stop**	**Start-stop**	-	-
1	T0	**UE A**, **UE B** and **UE C** receive from call control entity a request to initiate Floor Control	**O: has permission**	**O: silence**	**O: silence**	**UE A** creates an instance of Floor Control Participant, and sends Floor Granted; **UE B** and **UE C** create an instance of Floor Control Participant, and start T230	**7.2.3.2.2 7.2.3.2.3**
2	T0 + t_AB1	**UE B** and **UE C** receives Floor Granted	-	**O: has no permission**	**O: has no permission**	**UE B** and **UE C** start T203, stop T230, and set SSRC of current arbitrator to the SSRC of user (UE A) whom the floor was granted to	**7.2.3.3.4**
3	T0 + t_A1 (*with t_A1 > t_AB1*)	**UE A** sends RTP packets	-	-	-	**UE A** starts T206	**7.2.3.5.2**
4	T0 + t_A1 + t_AB2	**UE B** and **UE C** receive RTP packets	-	-	-	**UE B** and **UE C** restart T203 and set SSRC of current arbitrator to SSRC of RTP packet	**7.2.3.4.6**

And the corresponding figure is Fig. 8:

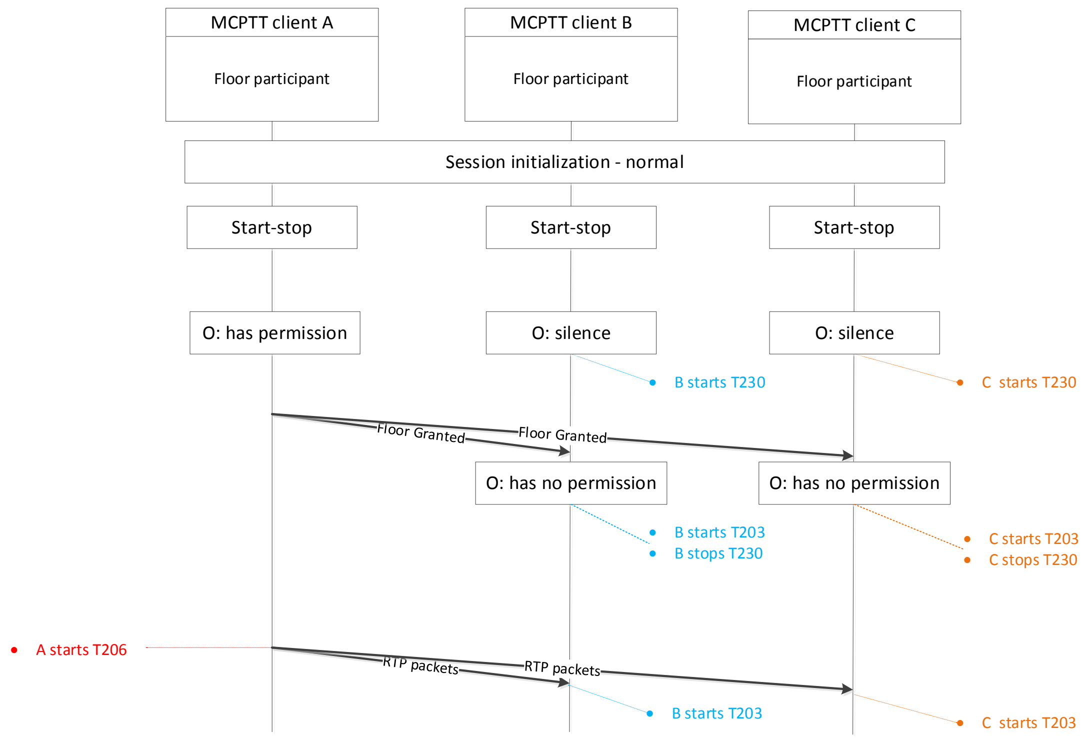

Figure 8: Session initialization – normal

2.1.3.2. Session initialization – message lost

Similar to the scenario in Section 2.1.3.1, this scenario focuses on the initialization of the MCPTT session. It covers the creation of the floor control entity and the sending of the first data packet after the call starts. However, the floor granted message sent by UE A at time T0 is lost and does not reach UE B or UE C. UE B and UE C receive the RTP packet first instead.

Scenario Purpose: To observe that a UE can start receiving and rendering RTP packets from arbitrator.

Table 13: Session initialization – message lost

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**Start-stop**	**Start-stop**	**Start-stop**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE A**, **UE B** and **UE C** receive from call control entity request to initiate Floor Control	**O: has permission**	**O: silence**	**O: silence**	**UE A** creates an instance of Floor Control Participant, and sends Floor Granted; **UE B** and **UE C** create an instance of Floor Control Participant, and start T230	**7.2.3.2.2 7.2.3.2.3**
2	T0 + t_A1	**UE A** sends RTP packet	**-**	**-**	**-**	**UE A** starts T206	**7.2.3.5.2**
3	T0 + t_A1 + t_AB1	**UE B** and **UE C** receive RTP packet	**-**	**O: has no permission**	**O: has no permission**	**UE B** and **UE C** stop T230, start T203, and set SSRC of current arbitrator to SSRC (UE A) of RTP packet	**7.2.3.3.3**

And the corresponding figure is Fig. 9:

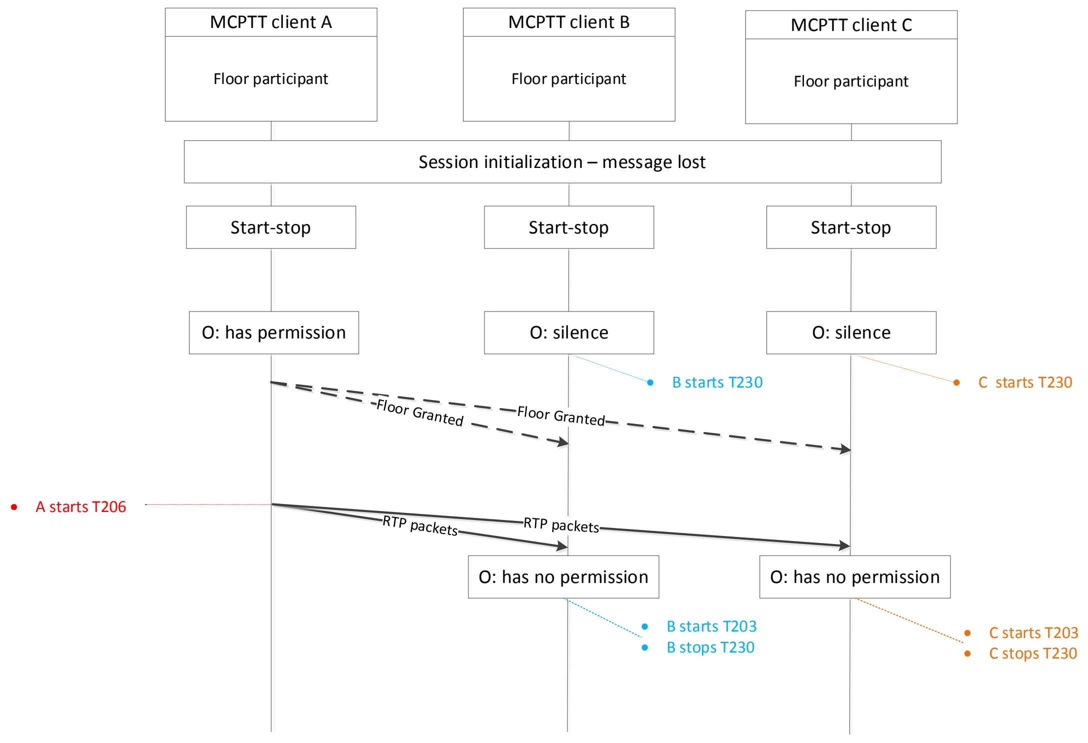

Figure 9: Session initialization – message lost

2.1.3.3.Session initialization – private call

This scenario focuses on the initialization of the MCPTT session for a private call. It covers the creation of the floor control entity and the sending of the first data packet after the call starts, as well as the floor arbitrator giving back the floor to the second user once it is done talking.

Scenario purpose: To observe that the originating MCPTT call participant becomes arbitrator (Floor Grant) of the private call and starts transmitting (RTP packets), then once done talking, hands over the floor to the other user.

Table 14: Session initialization – private call

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**Consequent behavior**	**Reference for details**
0			**Start-stop**	**Start-stop**	-	-
1	T0	**UE A** and **UE B** receive from call control entity a request to initiate Floor Control	**O: has permission**	**O: has no permission**	**UE A** creates an instance of Floor Control Participant, and sends Floor Granted; **UE B** creates an instance of Floor Control Participant, and starts T203	**7.2.3.2.2** **7.2.3.2.4**
2	T0 + t_AB1	**UE B** receives Floor Granted	-	-	**UE B** stops rendering RTP packets, sets SSRC of candidate arbitrator to SSRC of UE A, and restarts T203	**7.2.3.4.5**
3	T0 + t_A1 (*with t_A1 > + t_AB1*)	**UE A** sends RTP media packets	-	-	**UE A** starts T206	**7.2.3.5.2**
4	T0 + t_A1 + t_AB2	**UE B** receives RTP packets	-	-	**UE B** renders receives RTP packets, sets stored SSRC of current arbitrator to SSRC of UE A, clears stores SSRC of candidate arbitrator, and restarts T203	**7.2.3.4.6**
5	T0 + t_A2 (*with t_A2 > + t_A1 + t_AB2*)	**UE A** releases PTT button	**O: silence**	-	**UE A** sends Floor Release, stops T206 and T207, clears SSRC of current floor arbitrator, and starts T230	**7.2.3.5.5**

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
6	T0 + t_A2 + t_AB3	**UE B** receives Floor Release	-	**O: silence**	**UE B** stops rendering RTP packets, clears SSRC of current floor arbitrator, stops T203 and starts T230	**7.2.3.4.3**
7	T0 + t_B1 (*with t_B1 > + t_A2 + t_AB3*)	**UE B** presses PTT button	-	**O: pending request**	**UE B** sends Floor Request, starts T201 and initializes C201, and stops T230	**7.2.3.3.2**
8	T0 + t_B1 + t_BA1	**UE A** receives Floor Request	**O: pending granted**	-	**UE A** sends Floor Granted, stops T230 and starts T205	**7.2.3.3.5**
9	T0 + t_B1 + t_BA1 + t_AB4	**UE B** receives Floor Granted	-	**O: has permission**	**UE B** stops rendering RTP packets, sets SSRC of current floor arbitrator to own SSRC, and stops T201 and T203	**7.2.3.6.7**

And the corresponding figure is Fig. 10:

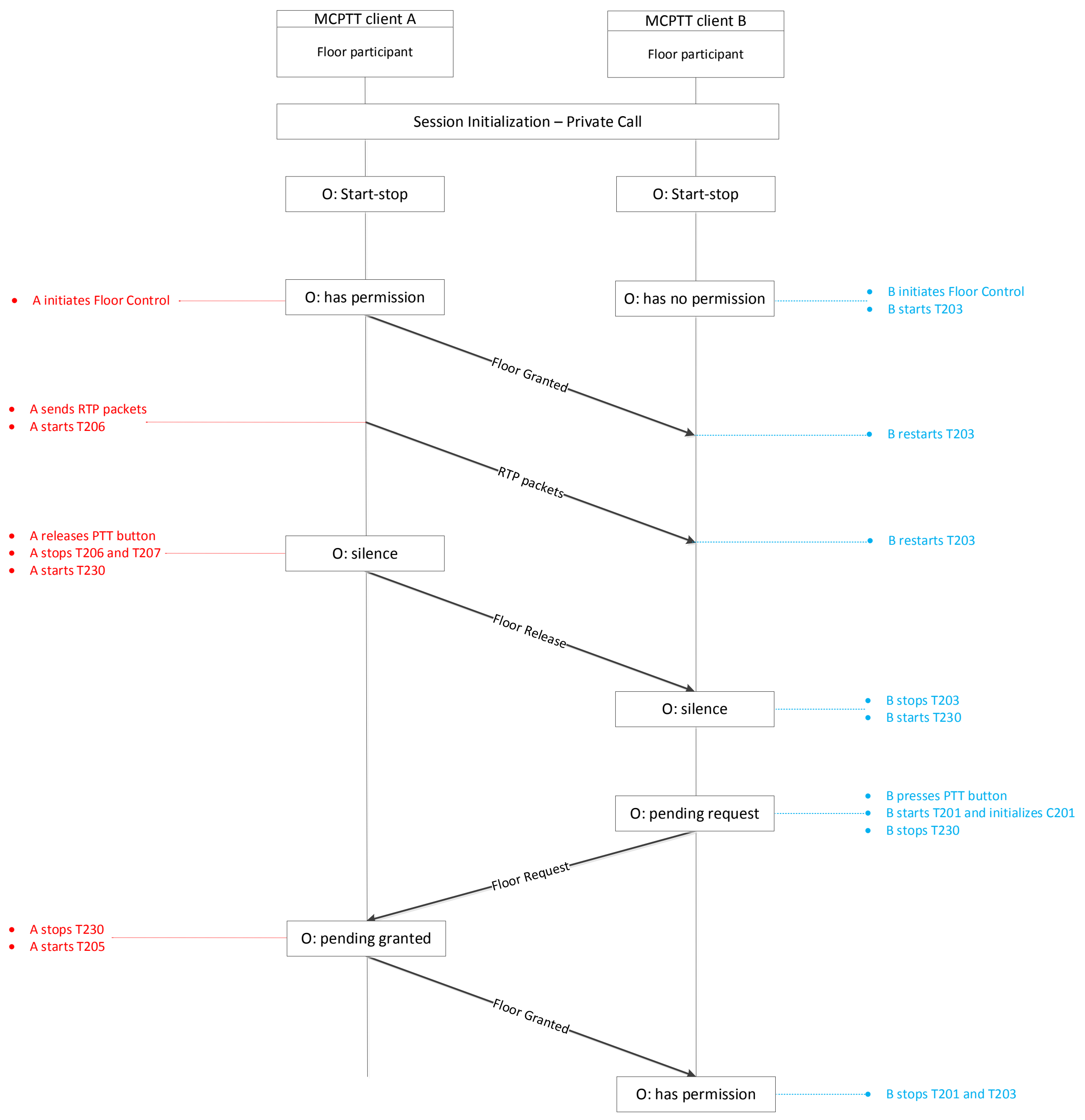

Figure 10: Session initialization – private call

2.1.3.4.Session initialization – broadcast call

This scenario focuses on the initialization of the MCPTT session for a broadcast call.

Scenario purpose: To observe that the originating MCPTT call participant becomes arbitrator (Floor Grant) of the broadcast call.

Table 15: Session initialization – broadcast call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0			**Start-stop**	**Start-stop**	**Start-stop**	-	-
1	T0	**UE A, UE B** and **UE C** receive from call control entity a request to initiate Floor Control	**O: has permission**	**O: has no permission**	**O: has no permission**	**UE A** creates an instance of Floor Control Participant, and sends Floor Granted; **UE B** and **UE C** create an instance of Floor Control Participant, and start T203	**7.2.3.2.2 7.2.3.2.9**
2	T0 + t_AB1	**UE B** and **UE C** receive Floor Granted	-	-		**UE B** and **UE C** set SSRC of candidate arbitrator to SSRC of UE A, and restart T203	**7.2.3.4.5**

2.1.4. Session release

This scenario focuses on the release of the MCPTT session, including the deletion of the floor control entity.

Scenario Purpose: To observe that the floor control state machine is terminated when the call is terminated.

Table 16: Session release

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**Any state**	**Any state**	**Any state**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE A, UE B,** and **UE C** receive an MCPTT call release request	**Start-stop**	**Start-stop**	**Start-stop**	**UE A, UE B,** and **UE C** stop sending Floor Control messages towards other Floor participants, request the MCPTT client to stop sending and receiving RTP media packets, release every resource and stop every running timer, and terminate the instance of Floor Participant state diagram	**7.2.3.9.2**

2.2. Basic group call – call control

The call control protocol is the protocol used to control the session needed to support MCPTT. It addresses issues such as how to initiate a call, how to terminate a call, and how to respond when receiving a call. An MCPTT off-network mode call may be a basic group call, a private call, or broadcast group call. Test scenarios presented in Section 2.2 and Section 2.3 are designed for basic group calls.

The state transitions that are analyzed in each test scenario of Section 2.2 are based on Figure 10.2.2.2-1 of [4]. Call Control test scenarios can be categorized into 4 groups: call setup, call merge, call release, and call reject.

2.2.1. Call setup

2.2.1.1.Call setup – join a call

In this scenario, a user decides to join an ongoing basic group call. It receives an answer to its group call probe and joins the call as a terminating participant.

Scenario purpose: To observe that a user joins an ongoing group call.

Table 17: Call setup – join a call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S1: start-stop**	**S3: part of ongoing call**	**S3: part of ongoing call**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	Indication from **UE A** to join a group call	**S2: waiting for call announce ment**	-	-	**UE A** stores MCPTT group ID as MCPTT group ID of the call, creates a Call Type Control state machine, generates and sends a GROUP CALL PROBE message, and starts TFG1 and TFG3	**10.2.2.4.2.1**
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** check if MCPTT group ID IE matches stored MCPTT Group ID of the call, stop TFG2, start TFG2, and set stored probe response of the call to "true"	**10.2.2.4.2.3**
3	T0 + t_AB1 + TFG2 (*with TFG2 in UE B < TFG2 in UE C*)	**UE B** TFG2 expires	-	-	-	**UE B** generates and sends GROUP CALL ANNOUNCEME NT, sets stored probe response value of the call to "false", and starts TFG2	**10.2.2.4.4.1**

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
4	T0 + t_AB1 + TFG2 + t_BA1	**UE A** and **UE C** receive GROUP CALL ANNOUNCEMENT	**S3: part of ongoing call**	**-**	**-**	**UE A** checks if MCPTT Group ID IE matches stored MCPTT Group ID of the call, stops TFG1 and TFG3, stores the values of Session Description Protocol (SDP) IE, Call Identifier IE, originating MCPTT user ID IE, Refresh interval IE, Call start time IE of the GROUP CALL ANNOUNCEMENT as corresponding values, establishes media session, starts Floor Control as terminating Floor Participant, and starts TFG2 and TFG6; **UE C** stops TFG2, starts TFG2, and sets the stored probe response of the call to “false”.	**10.2.2.4.3.2 10.2.2.4.4.2**

And the corresponding figure is Fig. 11:

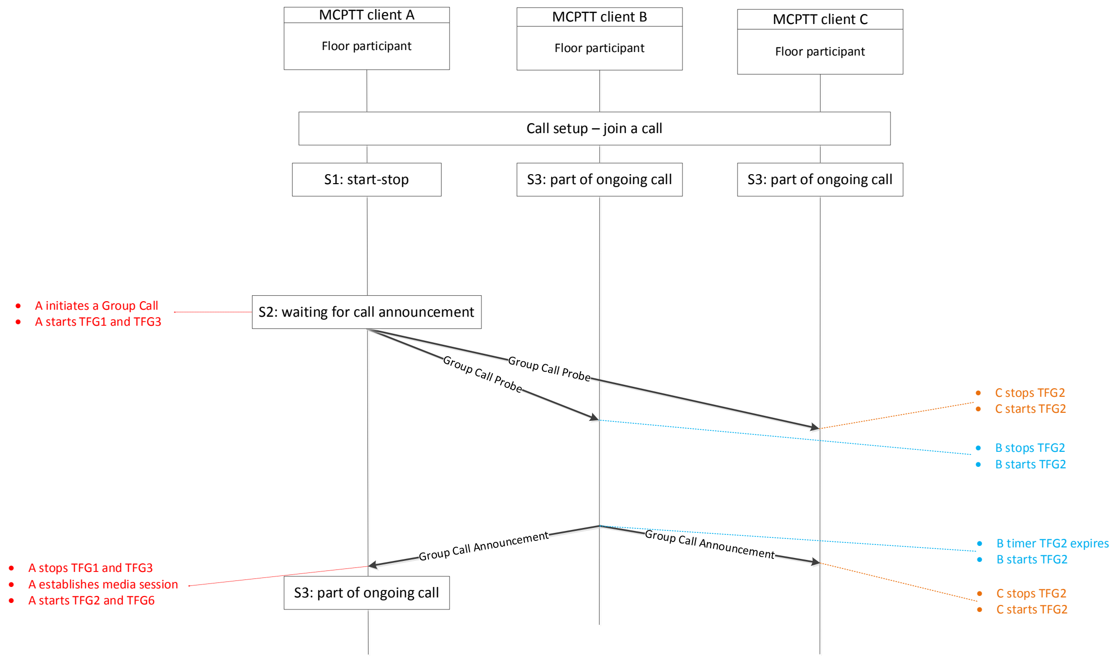

Figure 11: Call setup – join a call

2.2.1.2.Call setup – establish a new call with confirm mode indication IE in the Call Announcement message

In this scenario, a user decides to establish a new group call, and initiates the call as the originating participant. The confirm mode indication IE is included in the Call Announcement message.

Scenario purpose: To observe that a user can establish a new group call with confirm mode indication IE in the Call Announcement message and be joined by other members of the same group.

In this scenario, UE B is configured as “MCPTT User acknowledgement is required upon a terminating call request reception”. while UE C is configured as “MCPTT User acknowledgement not required upon a terminating call request reception”.

Table 18: Call setup – establish a new call with confirm mode indication IE in the Call Announcement message

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S1: start-stop**	**S1: start-stop**	**S1: start-stop**	-	-

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
1	T0	Indication from **UE A** to initiate a group call	**S2: waiting for call announce ment**	-	-	**UE A** stores MCPTT group ID as MCPTT group ID of the call, creates a Call type Control state machine, generates and sends a GROUP CALL PROBE message, and starts TFG1 and TFG3	**10.2.2.4.2.1**
2	T0 + TFG3	**UE A** TFG3 expires	-	-	-	**UE A** generates and sends GROUP CALL PROBE, and starts TFG3	**10.2.2.4.2.2**
3	T0 + TFG3 + t_AB1	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** discard the message	**10.2.2.4.7.1**
4	T0 + n*TFG3	**UE A** TFG3 expires	-	-	-	**UE A** generates and sends GROUP CALL PROBE, and starts TFG3	**10.2.2.4.2.2**
5	T0 + nTFG3 + t_ABn	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** discard the message	**10.2.2.4.7.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
6	T0 + TFG1	**UE A** TFG1 expires	**S3: part of ongoing call**	**-**	**-**	**UE A** stops TFG3, generates and stores SDP body, generates and stores the call identifier, selects and stores refresh interval, stores its own MCPTT ID as originating MCPTT user ID, stores current UTC time as call start time, generates and sends GROUP CALL ANNOUNCEMENT with confirm mode indication IE, establishes media session, starts Floor Control as originating floor participant, and starts TFG2 and TFG6	**10.2.2.4.3.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
7	T0 + TFG1 + t_AB(n+1)	**UE B** and **UE C** receive GROUP CALL ANNOUNCEMENT (*contains Confirm mode Indication IE*)	-	**S5: pending user action with confirm indication** *(UE B is configured as "MCPTT User acknowledgement is required upon a terminating call request reception")*	**S3: part of ongoing call** *(UE C is configured as "MCPTT User acknowledgement not required upon a terminating call request reception")*	**UE B** and **UE C** check if MCPTT group ID IE from GROUP CALL ANNOUNCEMENT does not match MCPTT group ID of the call stored for other state machines, store value of SDP IE, Call Identifier IE, Originating MCPTT user ID IE, Refresh Interval IE, MCPTT Group ID IE, Call start time IE of the GROUP CALL ANNOUNCEMENT as corresponding values, create Call Type Control state machine; **UE B** starts TFG4; **UE C** establishes media session based on stored SDP body of the call, starts Floor Control as terminating floor participant, generates and sends GROUP CALL ACCEPT message, and starts TFG2 and TFG6	**10.2.2.4.3.3**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
8	T0 + TFG1 + t_AB(n+1) + t_CA1	**UE A** and **UE B** receive GROUP CALL ACCEPT from UE C	-	-	-	**UE A** checks if MCPTT group ID IE of GROUP CALL ACCEPT message matches stored MCPTT group ID of the call, and informs MCPTT user about call acceptance; **UE B** discards the group call control message	**10.2.2.4.3.6 10.2.2.4.7.1**
9	T0 + t_B1 *(with (TFG1 + t_AB(n+1) + t_CA1) < t_B1)*	**UE B** accepts the terminating call with confirm indication	-	**S3: part of ongoing call**	-	**UE B** establishes media session based on stored SDP body of the call, starts Floor Control as terminating floor participant, generates and sends GROUP CALL ACCEPT message, and starts TFG2 and TFG6	**10.2.2.4.3.4**
10	T0 + t_B1 + t_BA1	**UE A** and **UE C** receive GROUP CALL ACCEPT from UE B	-	-	-	**UE A** and **UE C** check if MCPTT group ID IE of GROUP CALL ACCEPT message matches stored MCPTT group ID of the call, and inform MCPTT user about call acceptance	**10.2.2.4.3.6**

And the corresponding figure is Fig. 12:

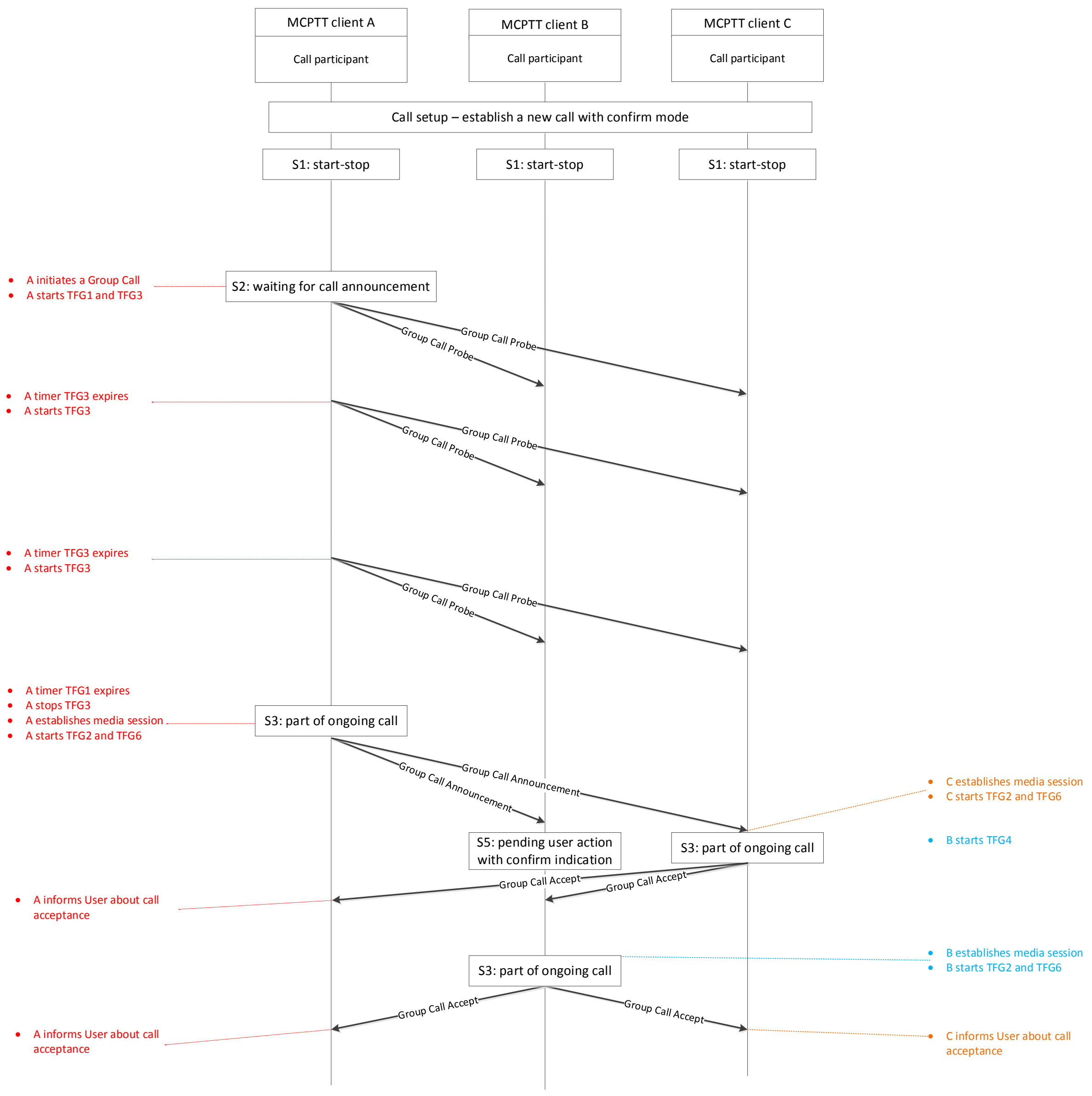

Figure 12: Call setup – establish a new call with confirm mode

2.2.1.3.Call setup – establish a new call without confirm mode indication IE in the Call Announcement message

In this scenario, a user decides to establish a new group call, and initiates the call as the originating participant. The confirm mode indication IE is not included in the Call Announcement message.

Scenario purpose: To observe that a user can establish a new group call without confirm mode indication IE in the Call Announcement message and be joined by other members of the same group.

In this scenario, UE B is configured as “MCPTT User acknowledgement is required upon a terminating call request reception”, while UE C is configured as “MCPTT User acknowledgement not required upon a terminating call request reception”.

Table 19: Call setup – establish a new call without confirm mode indication IE in the Call Announcement message

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**S1: start-stop**	**S1: start-stop**	**S1: start-stop**	-	-
1	T0	Indication from **UE A** to initiate a group call	**S2: waiting for call announcement**	-	-	**UE A** stores MCPTT group ID as MCPTT group ID of the call, creates a Call type Control state machine, generates and sends a GROUP CALL PROBE message, and starts TFG1 and TFG3	**10.2.2.4.2.1**
2	T0 + TFG3	**UE A** TFG3 expires	-	-	-	**UE A** generates and sends GROUP CALL PROBE, and starts TFG3	**10.2.2.4.2.2**
3	T0 + TFG3 + t_AB1	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** discard the message	**10.2.2.4.7.1**
4	T0 + n*TFG3	**UE A** TFG3 expires	-	-	-	**UE A** generates and sends GROUP CALL PROBE, and starts TFG3	**10.2.2.4.2.2**
5	T0 + n*TFG3 + t_ABn	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** discard the message	**10.2.2.4.7.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
6	T0 + TFG1	**UE A** TFG1 expires	**S3: part of ongoing call**	**-**	**-**	**UE A** stops TFG3, generates and stores SDP body, generates and stores the call identifier, selects and stores refresh interval, stores its own MCPTT ID as originating MCPTT user ID, stores current UTC time as call start time, generates and sends GROUP CALL ANNOUNCEMENT with no confirm mode indication IE, establishes media session, starts Floor Control as originating floor participant, and starts TFG2 and TFG6	**10.2.2.4.3.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
7	T0 + TFG1 + t_AB(n+1)	**UE B** and **UE C** receive GROUP CALL ANNOUNCEMENT (*does not contain Confirm mode Indication IE*)	-	**S4: pending User action without confirm indication** (*UE B is configured as MCPTT User acknowledgement required upon a terminating call request reception)*	**S3: part of ongoing call** (*UE C is configured as MCPTT User acknowledgement not required upon a terminating call request reception)*	**UE B** and **UE C** check if MCPTT group ID IE from GROUP CALL ANNOUNCEMENT does not match MCPTT group ID of the call stored for other state machines, store value of SDP IE, Call Identifier IE, Originating MCPTT user ID IE, Refresh Interval IE, MCPTT Group ID IE, Call start time IE of the GROUP CALL ANNOUNCEMENT as corresponding values, create Call Type Control state machine; **UE B** starts TFG4; **UE C** establishes media session based on stored SDP body of the call, starts Floor Control as terminating floor participant, and starts TFG2 and TFG6	**10.2.2.4.3.3**
8	T0 + t_C1 (*with t_C1 > TFG1 + t_AB(n+1)*)	**UE B** accepts the terminating call	-	**S3: part of ongoing call**	-	**UE B** establishes media session based on SDP body of the call, starts Floor Control as terminating floor participant, and starts TFG2 and TFG6	**10.2.2.4.3.5**

2.2.2. Call merge

In this scenario, there is already an ongoing group call with UE A and UE B. Although UE C tries to join the group call by sending a GROUP CALL PROBE, it does not receive a response from UE A or UE B (e.g., temporarily out of range), thus UE C starts a new group call for the same group. Once UE A, UE B and UE C are within transmission range of each other, UE C receives the GROUP CALL ANNOUNCEMENT message from UE A or UE B, and since the group IDs match and the starting time of UE C's group call is later than the original group call, UE C merges with the existing group call. There is no state change for UE A or UE B during the merging process.

Scenario purpose: To observe that a user, separated from other users already engaged in an ongoing group call, creates his own group call; once this user gets back in range of the other ongoing group call, the newly created group call merges with the older one.

Table 20: Call merge

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**S3: part of ongoing call** (*group ID=n*)	**S3: part of ongoing call** (*group ID=n*)	**S1: start-stop**	-	-
1	T0	Indication from **UE C** to initiate a group call with group ID=n	-	-	**S2: waiting for call announcement**	**UE C** stores MCPTT group ID as MCPTT group ID of the call, creates a Call Type Control state machine, generates and sends a GROUP CALL PROBE message with MCPTT group ID = n, and starts TFG1 and TFG3	**10.2.2.4.2.1**
2	T0 + TFG3	**UE C** TFG3 expires	-	-	-	**UE C** generates and sends GROUP CALL PROBE, and starts TFG3	**10.2.2.4.2.2**
3	T0 + n*TFG3	**UE C** TFG3 expires	-	-	-	**UE C** generates and sends GROUP CALL PROBE, and starts TFG3	**10.2.2.4.2.2**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
4	T0 + TFG1	**UE C** TFG1 expires	-	-	**S3: part of ongoing call** (*group ID=n, but not the same group call as UE A and UE B*)	**UE C** sets the originating MCPTT user ID of the call to the ID of UE C, sets the call identifier, the SDP body, the refresh interval, establishes media session, starts floor control as originating participant, generates and sends GROUP CALL ANNOUNCEMENT, stops TFG3, and starts TFG2 and TFG6	**10.2.2.4.3.1**
5	T0 + TFG1 + TFG2	**UE C** TFG2 expires	-	-	-	**UE C** generates and sends GROUP CALL ANNOUNCEMENT, and starts TFG2	**10.2.2.4.4.1**
6	T0 + TFG1 + TFG2 + t_CA1	**UE A** and **UE B** receive GROUP CALL ANNOUNCEMENT (*from UE C with MCPTT group ID matching stored MCPTT group ID of the call*)	-	-	-	**UE A** and **UE B** compare the Call start time IE of the GROUP CALL ANNOUNCEMENT message from UE C and determine it is not lower than the stored call start time of the call and take no further action	**10.2.2.4.6.1**
7	T0 + t_A1 (*with t_A1 > TFG1 + TFG2 + t_CA1*)	**UE A** TFG2 expires	-	-	-	**UE A** generates and sends GROUP CALL ANNOUNCEMENT, and starts TFG2	**10.2.2.4.4.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
8	T0 + t_A1 + t_AB1	**UE C** and **UE B** receive GROUP CALL ANNOUNCEMENT (*with MCPTT group ID matching stored MCPTT group ID of the call*)	-	-	-	**UE C** stores value of SDP IE, Call Identifier IE, Originating MCPTT user ID IE, Refresh Interval IE and Call start time IE, adjusts the media session, stops TFG6 and TFG2, and starts again TFG6 and TFG2; **UE B** stops TFG2 and starts TFG2	**10.2.2.4.6.1 10.2.2.4.4.2**

And the corresponding figure is Fig. 13:

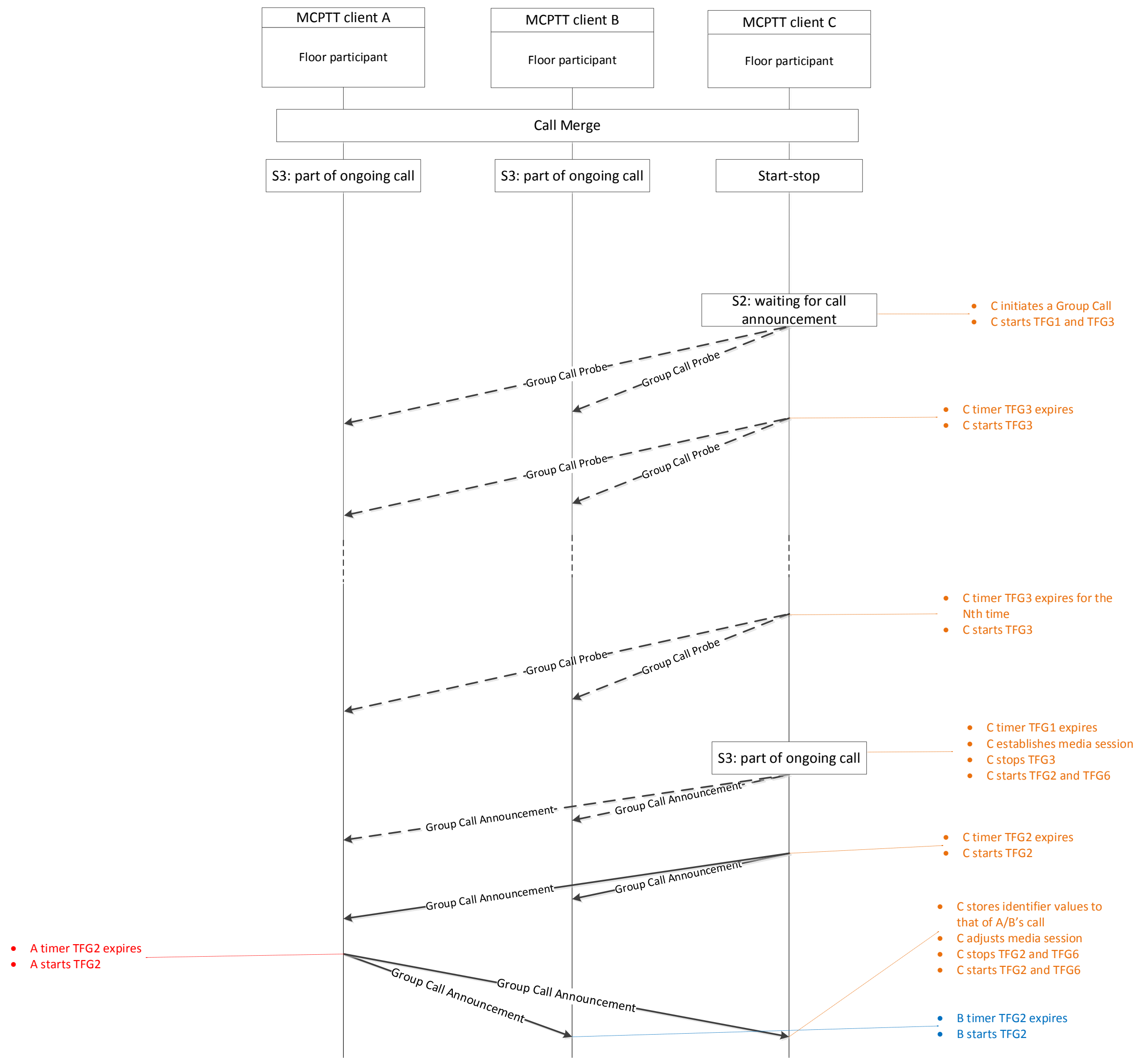

Figure 13: Call merge

2.2.3. Call release

2.2.3.1.Call release while in the call

In this scenario, UE A starts in the state, “S3: part of ongoing call”, and decides to withdraw from the call. We assume here that TFG5 < TFG2 for this particular scenario.

Scenario purpose: To observe that a user is able to release from an already ongoing call.

Table 21: Call release while in the call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S3: part of ongoing call**	**S3: part of ongoing call**	**S3: part of ongoing call**	-	-
1	T0	**UE A** indication from MCPTT user to release	**S6: ignoring incoming call announcements**	-	-	**UE A** releases media session, stops TFG2, and starts TFG5	**10.2.2.4.5.1**
2	T0 + t_B1	**UE B** TFG2 expires	-	-	-	**UE B** generates and sends GROUP CALL ANNOUNCEMENT, if stored probe response value of the call is set to "true", sets stored probe response value of the call to "false", and starts TFG2	**10.2.2.4.4.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
3	T0 + t_B1 + t_BA1	**UE A** and **UE C** receive GROUP CALL ANNOUNCEMENT	-	-	-	**UE A** stores value of SDP IE, Call identifier IE, Originating MCPTT user ID IE, refresh interval IE, Call start time IE of the GROUP CALL ANNOUNCEMENT as corresponding values, stops TFG5, and starts TFG5; **UE C** stops TFG2, starts TFG2 and sets probe response of the call to "false"	**10.2.2.4.5.2 10.2.2.4.4.2**
4	T0 + TFG5	**UE A** TFG5 expires	**S1: start-stop**	-	-	**UE A** releases stored SDP body, call identifier, originating MCPTT user ID, refresh interval, MCPTT Group ID, call start time of the call, and destroys Call Type Control state machine	**10.2.2.4.5.4**

2.2.3.2.Call release after call probe and return to "S1: start-stop"

In this scenario, a user in the "S2: waiting for call announcement" state decides to release its probe to join the call, and transitions to "S1: start-stop" state. The user does not receive any Group Call Announcement before transitioning to "S1: start-stop" state.

Scenario purpose: To observe that a user can release its probe to find an ongoing call and returns to “S1: start-stop” state.

Table 22: Call release after call probe and return to “S1: start-stop”

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S1: start-stop**	**S3: part of ongoing call**	**S3: part of ongoing call**	-	-
1	T0	Indication from **UE A** to join a group call	**S2: waiting for call announcement**	-	-	**UE A** stores MCPTT group ID as MCPTT group ID of the call, creates a Call Type Control state machine, generates and sends a GROUP CALL PROBE message, and starts TFG1 and TFG3	**10.2.2.4.2.1**
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** check if MCPTT group ID IE matches stored MCPTT Group ID of the call, stop TFG2, start TFG2, and set stored probe response of the call to "true"	**10.2.2.4.2.3**
3	T0 + t_A1	**UE A** indication from MCPTT User to release the Group Call	**S7: waiting for call announcement after call release**	-	-	**UE A** stops TFG3	**10.2.2.4.5.5**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
4	T0 + TFG1 (*with TFG1 < (T_AB1+ TFG2)*)	**UE A** TFG1 expires	**S1: start-stop**	**-**	**-**	**UE A** releases stored MCPTT Group ID of the call, and destroys Call Type Control state machine	**10.2.2.4.5.8**

2.2.3.3. Call release after call probe and enter "S6: ignoring incoming call announcements"

In this scenario, a user in the "S2: waiting for call announcement" state decides to release its probe to join the call and receives a Group Call Announcement before it can transition to "S1: start-stop" state. Thus, it enters "S6: ignoring incoming call announcements" instead of S1.

Scenario purpose: To observe that a user can release its probe to find an ongoing call and enters "S6: ignoring incoming call announcements" state.

Table 23: Call release after call probe and enter "S6: ignoring incoming call announcements"

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S1: start-stop**	**S3: part of ongoing call**	**S3: part of ongoing call**	-	-
1	T0	Indication from **UE A** to join a group call	**S2: waiting for call announcement**	-	-	**UE A** stores MCPTT group ID as MCPTT group ID of the call, creates a Call Type Control state machine, generates and sends a GROUP CALL PROBE message, and starts TFG1 and TFG3	**10.2.2.4.2.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** check if MCPTT group ID IE matches stored MCPTT Group ID of the call, stop TFG2, start TFG2, and set stored probe response of the call to "true"	**10.2.2.4.2.3**
3	T0 + t_A1	**UE A** indication from MCPTT User to release the Group Call	**S7: waiting for call announce ment after call release**	-	-	**UE A** stops TFG3	**10.2.2.4.5.5**
4	T0 + t_AB1 + TFG2 (*with (t_AB1 + TFG2) <TFG1*)	**UE C** TFG2 expires	-	-	-	**UE C** generates and sends GROUP CALL ANNOUNCEMEN T, and starts TFG2	**10.2.2.4.4.1**
5	T0 + t_AB1 + TFG2 + t_CA1	**UE A** and **UE B** receive GROUP CALL ANNOUN CEMENT	**S6: ignoring incoming call announce ments**	-	-	**UE A** stores values of SDP IE, Call Identifier IE, Originating MCPTT user ID IE, Refresh interval IE and Call Start Time IE of the GROUP CALL ANNOUNCEMEN T message as corresponding values, stops TFG1, and starts TFG5; **UE B** stops TFG2 and starts again TFG2	**10.2.2.4.5.7 10.2.2.4.4.2**

2.2.3.4.Call release while pending user action – without confirm indication

In this scenario, a user in the “S4: pending user action without confirmation indication” state decides to release the call. In this scenario, instead of going back to the “S1: start-stop” state, the user decides to initiate a group call again before TFG5 expires, although the user (UE A) released the call before.

Scenario purpose: To observe that after receiving notification (GROUP CALL ANNOUNCEMENT) of an ongoing call, a UE (without confirmation mode) can release its request to join a call, only to re-enable it later and effectively join the call.

Table 24: Call release while pending user action – without confirm indication

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S4: pending user action without confirm indication**	**S3: part of ongoing call**	**S3: part of ongoing call**	-	-
1	T0	**UE A** indication from MCPTT user to release the call	**S6: ignoring incoming call announcements**	-	-	**UE A** releases media session, stops TFG2 and TFG4, and starts TFG5	**10.2.2.4.5.1**
2	T0 + t_A1	**UE A** indication from MCPTT user to initiate a Group Call for an MCPTT group ID matching stored MCPTT Group ID of the call	**S3: part of ongoing call**	-	-	**UE A** stops TFG5, establishes media session, starts Floor Control as terminating Floor Participant, and starts TFG2 and TFG6	**10.2.2.4.5.3**

2.2.3.5.Call release while pending user action – with confirm indication

In this scenario, a user in the "S5: pending user action with confirmation indication" state decides to release its request to join.

Scenario purpose: To observe that UE A does not respond to the request to join the ongoing call and ignores future GROUP CALL ANNOUNCEMENT messages until the call is terminated or the UE moves out of range so that GROUP CALL ANNOUNCEMENT messages cease to be received by UE A and permits TFG5 to timeout, thus returning UE A to the S1: start-stop state.

Table 25: Call release while pending user action – with confirm indication

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S5: pending user action with confirm indication**	**S3: part of ongoing call**	**S3: part of ongoing call**	-	-
1	T0	**UE A** indication from MCPTT user to release the call	**S6: ignoring incoming call announcements**	-	-	**UE A** stops TFG4, and starts TFG5	**10.2.2.4.5.1**
2	T0 + t_B1	**UE B** TFG2 expires	-	-	-	**UE B** generates and sends GROUP CALL ANNOUNCEMENT, if stored probe response value of the call is set to "true", sets stored probe response value of the call to "false", and starts TFG2	**10.2.2.4.4.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
3	T0 + t_B1 + t_BA1	**UE A** and **UE C** receive GROUP CALL ANNOUNCEMENT	-	-	-	**UE A** stores value of SDP IE, Call identifier IE, Originating MCPTT user ID IE, refresh interval IE, Call start time IE of the GROUP CALL ANNOUNCEMENT, stops TFG5, and starts TFG5; **UE C** stops TFG2, starts TFG2, and sets probe response of the call to "false"	**10.2.2.4.5.2 10.2.2.4.4.2**
4	T0 + TFG5	**UE A** TFG5 expires	**S1: start-stop**	-	-	**UE A** releases stored SDP body, call identifier, originating MCPTT user ID, refresh interval, MCPTT Group ID, call start time of the call, and destroys Call Type Control state machine	**10.2.2.4.5.4**

2.2.3.6.Call release when maximum duration of the call is reached

In this scenario, one of the participants withdraws from the call because the maximum duration (TFG6) of the call is reached. UE A is then afterwards considered out of range of UE B and UE C, and is unable to receive their Group Call Announcement messages.

Scenario purpose: To observe that when the maximum duration of a call occurs for a user, that user withdraws from the call; to observe that this same user, now considered out of range from the other users, goes back to S1: start-stop state of the state machine.

Table 26: Call release when maximum duration of the call is reached

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S3: part of ongoing call**	**S3: part of ongoing call**	**S3: part of ongoing call**	-	-
1	T0	**UE A** TFG6 expires	**S6: ignoring incoming call announcements**	-	-	**UE A** releases the media session, stops TFG2, and starts TFG5	**10.2.2.4.5.9**
2	T0 + TFG5	**UE A** TFG5 expires	**S1: start-stop**	-	-	**UE A** releases stored SDP body, call identifier, originating MCPTT user ID, refresh interval, MCPTT Group ID, call start time of the call, and destroys Call Type Control state machine	**10.2.2.4.5.4**

2.2.3.7.Call release and setup

In this scenario, after sending a GROUP CALL PROBE message, a user decides to release the call, but later wants to start a new group call with the same group.

Scenario purpose: To observe that a user undecidedly tries to initiate a group call, release such initialization, then initiates it again with no other ongoing call having already been established.

Table 27: Call release and setup

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S1: start-stop**	**S1: start-stop**	**S1: start-stop**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	Indication from **UE A** to initiate a group call	**S2: waiting for call announcement**	-	-	**UE A** stores MCPTT group ID as MCPTT group ID of the call, creates a Call type Control state machine, generates and sends a GROUP CALL PROBE message, and starts TFG1 and TFG3	**10.2.2.4.2.1**
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** discard the call control message	**10.2.2.4.7.1**
3	T0 + t_A1	Indication from **UE A** to release the group call	**S7: waiting for call announcement after call release**	-	-	**UE A** stops TFG3	**10.2.2.4.5.5**
4	T0 + t_A2 (*with t_A2 > t_A1*)	Indication from **UE A** to initiate a group call for MCPTT group ID matching stored MCPTT group ID of the call	**S2: waiting for call announcement**	-	-	**UE A** stops TFG1, generates and sends GROUP CALL PROBE, and starts TFG1 and TFG3	**10.2.2.4.5.6**
5	T0 + t_A2 + t_AB2	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** discard the call control message	**10.2.2.4.7.1**

2.2.4. Call reject

In this scenario, UE A decides to establish a group call since there is no existing one. However, UE B rejects the call explicitly, and UE C rejects the call implicitly due to no timely user action.

Scenario purpose: To observe that a user establishes a new group call, with other users rejecting (in different manners) this call.

Table 28: Call reject

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**S1: start-stop**	**S1: start-stop**	**S1: start-stop**	-	-
1	T0	Indication from **UE A** to initiate a group call	**S2: waiting for call announce ment**	-	-	**UE A** stores MCPTT group ID as MCPTT group ID of the call, creates a Call Type Control state machine, generates and sends a GROUP CALL PROBE message, and starts TFG1 and TFG3	**10.2.2.4.2.1**
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** discard the call control message	**10.2.2.4.7.1**
3	T0 + TFG3	**UE A** TFG3 expires	-	-	-	**UE A** generates and sends GROUP CALL PROBE, and starts TFG3	**10.2.2.4.2.2**
4	T0 + TFG3 + t_AB2	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** discard the call control message	**10.2.2.4.7.1**
5	T0 + n*TFG3	**UE A** TFG3 expires	-	-	-	**UE A** generates and sends GROUP CALL PROBE, and starts TFG3	**10.2.2.4.2.2**

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
6	T0 + n*TFG3 + t_ABn	**UE B** and **UE C** receive GROUP CALL PROBE	-	-	-	**UE B** and **UE C** discard the call control message	**10.2.2.4.7.1**
7	T0 + TFG1	**UE A** TFG1 expires	**S3: part of ongoing call**	-	-	**UE A** stops TFG3, generates and stores SDP body, generates and stores the call identifier, selects and stores refresh interval, stores its own MCPTT ID as originating MCPTT user ID, stores current UTC time as call start time, generates and sends GROUP CALL ANNOUNCEMENT, establishes media session, starts Floor Control as originating floor participant, and starts TFG2 and TFG6	**10.2.2.4.3.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
8	T0 + TFG1 + t_AB(n+1)	**UE B** and **UE C** receive GROUP CALL ANNOUNCEMENT	-	**S4: pending user action without confirm indication** (*MCPTT User acknowledgement required upon a terminating call request reception and GROUP CALL ANNOUNCEMENT does not contain Confirm mode indication IE*)	**S4: pending user action without confirm indication** (*MCPTT User acknowledgement required upon a terminating call request reception and GROUP CALL ANNOUNCEMENT does not contain Confirm mode indication IE*)	**UE B** and **UE C** check if MCPTT group ID IE from GROUP CALL ANNOUNCEMENT does not match MCPTT group ID of the call stored for other state machines, store value of SDP IE, Call Identifier IE, Originating MCPTT user ID IE, Refresh Interval IE, MCPTT Group ID IE, Call start time IE of the GROUP CALL ANNOUNCEMENT as corresponding values, create Call Type Control state machine, and start TFG4	**10.2.2.4.3.3**
9	T0 + t_B1 (*with t_B1 > TFG1 + t_AB(n+1)*)	**UE B** rejects terminating call	-	**S6: ignoring incoming call announcements**	-	**UE B** stops TFG4, and starts TFG5	**10.2.2.4.3.7**
10	T0 + TFG1 + t_AB(n+1) + TFG4	**UE C** TFG4 expires	-	-	**S6: ignoring incoming call announcements**	**UE C** starts TFG5	**10.2.2.4.3.8**

2.3. Basic group call – call type control

The call type control state machine exists when a UE is part of an ongoing group call. The basic call control state machine has a related call type control state machine. The call type control state machine provides additional detail, i.e., call type, about the status of the call taking place: emergency group call, imminent peril group call, and basic group call.
The state transitions that are analyzed in each test scenario of Section 2.3 are based on "Figure 10.2.3.2-1: Call type control state machine" of TS 24.379 v14.2.0 [4]. Call type control test scenarios can be categorized into 5 groups: call type initialization, call type upgrade, call type downgrade, call release, and call merge.

Note: Interaction dependence between basic call control and basic call type control.

2.3.1. Call type initialization

2.3.1.1. Call type initialization – establish a new basic call

At the beginning of this scenario, there is no ongoing call established. UE A starts a new basic group call and initializes the call type state machine. It is important to note that this test scenario only depicts the changes happening at the call type state machine. There are other call control messages exchanged in the meanwhile, such as the GROUP CALL ANNOUNCEMENT messages, GROUP CALL PROBE messages, or both.

Scenario purpose: To observe that the "basic group call" type state is entered after creation of a new basic group call.

Table 29: Call type initialization – establish a new basic call

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**T0: waiting for call to establish**	**T0: Waiting for call to establish**	**T0: Waiting for call to establish**	-	-
1	T0	**UE A** initiates GROUP CALL PROBE	-	-	-	Follow 10.2.3.4.2 for lengthy detailed behavior (*the stored current call type is set to "BASIC GROUP CALL"*)	**10.2.3.4.2**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
2	T0 + TFG1	**UE A** TFG1 expires (*hence a* GROUP CALL ANNOUNCEMENT *message shall be sent from the basic call control state machine*)	**T2: in-progress basic group call**	**-**	**-**	**UE A** no action, except for state change	**10.2.3.4.6**

2.3.1.2.Call type initialization – establish a new emergency call

At the beginning of this scenario, there is no ongoing call established. UE A starts a new emergency group call and initializes the call type state machine. It is important to note that this test scenario only depicts the changes happening at the call type state machine. There are other call control messages exchanged in the meanwhile, such as the GROUP CALL ANNOUNCEMENT messages, GROUP CALL PROBE messages, or both.

Scenario purpose: To observe that the “emergency group call” type state is entered after creation of a new emergency group call.

Table 30: Call type initialization – establish a new emergency call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T0: waiting for call to establish**	**T0: Waiting for call to establish**	**T0: Waiting for call to establish**	-	-
1	T0	**UE A** initiates GROUP CALL PROBE	-	-	-	Follow 10.2.3.4.2 for lengthy detailed behavior (*the stored current call type is set to “EMERGENCY GROUP CALL”*)	**10.2.3.4.2**

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
2	T0 + TFG1	**UE A** TFG1 expires (*hence a* GROUP CALL ANNOUNCEMENT *message shall be sent from the basic call control state machine*)	**T1: in-progress emergency group call**	**-**	**-**	**UE A** starts TFG13	**10.2.3.4.6**

2.3.1.3.Call type initialization – establish a new imminent peril call

At the beginning of this scenario, there is no ongoing call established. UE A starts a new imminent peril group call and initializes the call type state machine. It is important to note that this test scenario only depicts the changes happening at the call type state machine. There are other call control messages exchanged in the meanwhile, such as the GROUP CALL ANNOUNCEMENT messages, GROUP CALL PROBE messages, or both.

Scenario purpose: To observe that the “imminent peril group call” type state is entered after creation of a new imminent peril group call.

Table 31: Call type initialization – establish a new imminent peril call

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**T0: waiting for call to establish**	**T0: Waiting for call to establish**	**T0: Waiting for call to establish**	-	-
1	T0	**UE A** initiates GROUP CALL PROBE	-	-	-	Follow 10.2.3.4.2 for lengthy detailed behavior (*the stored current call type is set to “IMMINENT PERIL GROUP CALL”*)	**10.2.3.4.2**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
2	T0 + TFG1	**UE A** TFG1 expires (*hence a* GROUP CALL ANNOUNCEMENT *message shall be sent from the basic call control state machine*)	**T3: in-progress imminent peril group call**	**-**	**-**	**UE A** starts TFG14	**10.2.3.4.6**

2.3.1.4.Call type initialization – join an emergency call after call probe

In this scenario, there is an ongoing emergency group call. UE A sends a call probe and joins this emergency group call.

Scenario purpose: To observe that the emergency call is joined after receiving an emergency-based GROUP CALL ANNOUNCEMENT message.

Table 32: Call type initialization – join an emergency call after call probe

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T0: waiting for call to establish**	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	-	-
1	T0	**UE A** initiates GROUP CALL PROBE	-	-	-	Follow 10.2.3.4.2 for lengthy detailed behavior	**10.2.3.4.2**
2	T0 + t_B1	**UE B** **TFG2** expires				**UE B** sends a GROUP CALL ANNOUNCEMENT	**NOTE: happens outside call type control**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
3	T0 + t_B1 + t_BA1	**UE A** receives GROUP CALL ANNOUNCEMENT from UE B	**T1: in-progress emergency group call**	-	-	**UE A** sets current call type to "EMERGENCY GROUP CALL", sets Proximity Based Services (ProSe) per-packet priority to value of MCPTT off-network emergency group call, sets stored last call type change time and last user to change call type to corresponding values of GROUP CALL ANNOUNCEMENT, and starts TFG13	**10.2.3.4.3**

2.3.1.5.Call type initialization – join an imminent peril call after call probe

In this scenario, there is an ongoing imminent peril group call. UE A sends a GROUP CALL PROBE and joins this imminent peril group call.

Scenario purpose: To observe that the imminent peril call is joined after receiving an imminent peril-based GROUP CALL ANNOUNCEMENT message.

Table 33: Call type initialization – join an imminent peril call after GROUP CALL PROBE

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	--	-	**T0: waiting for call to establish**	**T3: in-progress imminent peril group call**	**T3: in-progress imminent peril group call**	-	-
1	T0	**UE A** initiates GROUP CALL PROBE	-	-	-	Follow 10.2.3.4.2 for lengthy detailed behavior	**10.2.3.4.2**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
2	T0 + t_B1	**UE B** **TFG2** expires				**UE B** sends a GROUP CALL ANNOUNCEMENT	**NOTE: happens outside call type control**
3	T0 + t_B1 + t_BA1	**UE A** receives GROUP CALL ANNOUNCEMENT from UE B	**T3: in-progress imminent peril group call**	-	-	**UE A** checks if current call type is not "EMERGENCY GROUP CALL", sets current call type to "IMMINENT PERIL GROUP CALL", sets ProSe per-packet priority to value of MCPTT off-network imminent peril group call, sets stored last call type change time and last user to change call type to corresponding values of GROUP CALL ANNOUNCEMENT, and starts TFG14	**10.2.3.4.3**

2.3.1.6.Call type initialization – join a basic group call after group call probe

In this scenario, there is an ongoing basic group call. UE A sends a GROUP CALL PROBE message and joins this basic group call.

Scenario purpose: To observe that the basic group call is joined after receiving a basic group call-based GROUP CALL ANNOUNCEMENT message.

Table 34: Call type initialization – join a basic group call after group call probe

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T0: waiting for call to establish**	**T2: in-progress basic group call**	**T2: in-progress basic group call**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE A** initiates GROUP CALL PROBE	-	-	-	Follow 10.2.3.4.2 for lengthy detailed behavior	**10.2.3.4.2**
2	T0 + t_B1	Either **UE B** or **UE C** **TFG2** expires				**UE B** or **UE C** sends a GROUP CALL ANNOUNCEMENT	**NOTE: happens outside call type control**
3	T0 + t_B1 + t_BA1	**UE A** receives GROUP CALL ANNOUNCEMENT	**T2: in-progress basic group call**	-	-	**UE A** checks if current call type is not "EMERGENCY GROUP CALL", sets current call type to "BASIC GROUP CALL", and sets stored last call type change time and last user to change call type to corresponding values of GROUP CALL ANNOUNCEMENT	**10.2.3.4.3**

2.3.1.7. Call type initialization – join a basic group call with user acknowledgement required

In this scenario, a basic group call is ongoing. UE A receives a group call announcement message of that call which requires user acknowledgment. The MCPTT user of UE A accepts the call.

Scenario purpose: To observe that the basic group call state is entered after joining an already established basic group call (with acknowledgement required).

Table 35: Call type initialization – join a basic group call with user acknowledgement required

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T0: waiting for call to establish**	**T2: in-progress basic group call**	**T2: in-progress basic group call**	-	-
1	T0	**UE B** TFG2 expires	-	-	-	**UE B** sends a GROUP CALL ANNOUNCEMENT (*with Call type IE set to "BASIC GROUP CALL"*)	**NOTE: happens outside call type control**
2	T0 + t_BA1	**UE A** receives GROUP CALL ANNOUNCEMENT (*with MCPTT user acknowledgement required*)	-	-	-	Follow 10.2.3.4.4. for lengthy detailed behavior **UE A** stores current call type as "BASIC GROUP CALL"	**10.2.3.4.4**
3	T0 + t_A1 (*with t_A1 > t_BA1*)	**UE A** accepts the call	**T2: in-progress basic group call**	-	-	**UE A** no action, except for state change	**10.2.3.4.6**

2.3.1.8.Call type initialization – join an imminent peril group call with user acknowledgement required

In this scenario, an imminent peril group call is ongoing. UE A receives a group call announcement message of that call which requires user acknowledgment. The MCPTT user of UE A accepts the call.

Scenario purpose: To observe that the imminent peril call state is entered after joining an already established imminent peril group call (with acknowledgement required).

Table 36: Call type initialization – join an imminent peril group call with user acknowledgement required

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T0: waiting for call to establish**	**T3: in-progress imminent peril group call**	**T3: in-progress imminent peril group call**	-	-
1	T0	**UE B** TFG2 expires	-	-	-	**UE B** sends a GROUP CALL ANNOUNCEMENT (*with Call type IE set to "IMMINENT PERIL GROUP CALL"*)	**NOTE: happens outside call type control**
2	T0 + t_BA1	**UE A** receives GROUP CALL ANNOUNCEMENT (*with MCPTT user acknowledgement required*)	-	-	-	Follow 10.2.3.4.4. for lengthy detailed behavior **UE A** stores current call type as "IMMINENT PERIL GROUP CALL"	**10.2.3.4.4**
3	T0 + t_A1 (*with t_A1 > t_BA1*)	**UE A** accepts the call	**T3: in-progress imminent peril group call**	-	-	**UE A** starts TFG14	**10.2.3.4.6**

2.3.1.9. Call type initialization – join an emergency group call with user acknowledgement required

In this scenario, an emergency group call is ongoing. UE A receives a group call announcement message of that call which requires user acknowledgment. The MCPTT user of UE A accepts the call.

Scenario purpose: To observe that the emergency group call state is entered after joining an already established emergency group call.

Table 37: Call type initialization – join an emergency group call with user acknowledgement required

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**T0: waiting for call to establish**	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	-	-
1	T0	**UE B** TFG2 expires	-	-	-	**UE B** sends a GROUP CALL ANNOUNCEMENT (*with Call type IE set to "EMERGENCY GROUP CALL"*)	**NOTE: happens outside call type control**
2	T0 + t_BA1	**UE A** receives GROUP CALL ANNOUNCEMENT (*with MCPTT user acknowledgement required*)	-	-	-	Follow 10.2.3.4.4. for lengthy detailed behavior **UE A** stores current call type as "EMERGENCY GROUP CALL"	**10.2.3.4.4**
3	T0 + t_A1 (*with t_A1 > t_BA1*)	**UE A** accepts the call	**T1: in-progress emergency group call**	-	-	**UE A** starts TFG13	**10.2.3.4.6**

2.3.1.10. Call type initialization – join an emergency group call without user acknowledgement required

In this scenario, an emergency group call is ongoing. UE A receives a group call announcement that does not require user acknowledgment and joins the call. This scenario can be generalized to other scenarios in which the ongoing group call is basic or imminent peril.

Scenario purpose: To observe that the emergency call state is entered after joining an already established emergency group call (without acknowledgement required), based on the call type setting in the GROUP CALL ANNOUNCEMENT message.

Table 38: Call type initialization – join an emergency group call without user acknowledgement required

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T0: waiting for call to establish**	**T1: in-progress emergency group call**	**T1: in-progress emergency group**	-	-
1	T0	**UE B** TFG2 expires	-	-	-	**UE B** sends a GROUP CALL ANNOUNCEMENT (*with Call type IE set to "EMERGENCY GROUP CALL"*)	**NOTE: happens outside call type control**
2	T0 + t_BA1	**UE A** receives GROUP CALL ANNOUNCEMENT (*without MCPTT user acknowledgement required*)	**T1: in-progress emergency group call**	-	-	Follow 10.2.3.4.5 for lengthy detailed behavior **UE A** stores current call type as set to "EMERGENCY GROUP CALL"	**10.2.3.4.5**

2.3.2. Call type upgrade

2.3.2.1. Call upgrade from basic to imminent peril group call

In this scenario, a basic group call is ongoing. A user decides to upgrade the call to an imminent peril group call, so other users' call types are upgraded accordingly.

Scenario purpose: To observe that a basic group call can be upgraded to an imminent peril group call.

Table 39: Call upgrade from basic to imminent peril group call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T2: in-progress basic group call**	**T2: in-progress basic group call**	**T2: in-progress basic group call**	-	-

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
1	T0	**UE A** requests to upgrade the call to "IMMINENT PERIL GROUP CALL"	**T3: in-progress imminent peril group call**	-	-	**UE A** sets current call type to "IMMINENT PERIL GROUP CALL" and accordingly sets ProSe per-packet priority, starts TFG14, stores UTC time as last call type change time of the call as well as own MCPTT user ID as last user to change call type of the call, generates and sends GROUP CALL ANNOUNCEMENT	**10.2.3.4.7.1**
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL ANNOUNCEMENT	-	**T3: imminent peril group call**	**T3: imminent peril group call**	Follow 10.2.3.4.7.2 for lengthy detailed behavior	**10.2.3.4.7.2**

2.3.2.2.Call upgrade from basic to emergency group call

In this scenario, a basic group call is ongoing. A user decides to upgrade the call to an emergency group call, so other users' call types are upgraded accordingly.

Scenario purpose: To observe that a basic group call can be upgraded to an emergency group call.

Table 40: Call upgrade from basic to emergency group call

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**T2: in-progress basic group call**	**T2: in-progress basic group call**	**T2: in-progress basic group call**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE A** requests to upgrade the call to "EMERGENCY GROUP CALL"	**T1: in-progress emergency group call**	-	-	**UE A** sets current call type accordingly as well as current ProSe per-packet priority, starts TFG13, stores UTC time as last call type change time of the call as well as own MCPTT user ID as last user to change call type of the call, and generates and sends GROUP CALL ANNOUNCEMENT	**10.2.3.4.7.1**
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL ANNOUNCEMENT	-	**T1: in-progress emergency group call**	**T: in-progress emergency group call**	Follow 10.2.3.4.7.2 for lengthy detailed behavior	**10.2.3.4.7.2**

2.3.2.3.Call upgrade from imminent peril group call

In this scenario, an imminent peril group call is ongoing. A user decides to upgrade the call to an emergency group call, and the other users' call types are upgraded accordingly.

Scenario purpose: To observe that an imminent peril group call can be upgraded to an emergency group call.

Table 41: Call upgrade from imminent peril group call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T3: in-progress imminent peril group call**	**T3: in-progress imminent peril group call**	**T3: in-progress imminent peril group call**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE A** requests to upgrade the call to "EMERGENCY GROUP CALL"	**T1: in-progress emergency group call**	-	-	**UE A** sets current call type accordingly as well as current ProSe per-packet priority, starts TFG13 and stops TFG14, stores UTC time as last call type change time of the call as well as own MCPTT user ID as last user to change call type of the call, generates and sends GROUP CALL ANNOUNCEMENT	**10.2.3.4.7.1**
2	T0 + t_AB1	**UE B and UE C** receive GROUP CALL ANNOUNCEMENT	-	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	Follow 10.2.3.4.7.2 for lengthy detailed behavior	**10.2.3.4.7.2**

2.3.3. Call type downgrade

2.3.3.1. Explicit downgrade from emergency call

In this scenario, an emergency group call is in progress. An authorized user (UE A) decides to downgrade the call to a basic group call. In addition, multiple TFG11 expirations are observed in this test scenario. It is assumed that if there are n “GROUP CALL EMERGENCY END” messages, then the first n - 1 are lost, and UE B and UE C only receive the nth message.

Scenario purpose: To observe that an emergency group call can explicitly be downgraded to a basic group call.

Table 42: Explicit downgrade from emergency call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	-	-
1	T0	**UE A** requests to downgrade the call	**T2: in-progress basic group call**	-	-	**UE A** sets stored current call type to "BASIC GROUP CALL" as well as per-packet priority, call type change time and last user to change call type accordingly, generates and sends GROUP CALL EMERGENCY END, stops TFG13, starts TFG11 and initializes CFG11	**10.2.3.4.8.1**
2	T0 + TFG11	**UE A** TFG11 expires	-	-	-	**UE A** generates and sends GROUP CALL EMERGENCY END, increments value of CFG11, and starts TFG11 (*if value of CFG11 is less than its upper limit*)	**10.2.3.4.8.2**
3	T0 + n*TFG11	**UE A** TFG11 expires (*CFG11 at its upper limit*)	-	-	-	**UE A** generates and sends GROUP CALL EMERGENCY END, and increments value of CFG11 (*now equal to its upper limit*)	**10.2.3.4.8.2**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
4	T0 + n*TFG11 + t_AB1	**UE B** and **UE C** receive GROUP CALL EMERGENCY END	-	**T2: in-progress basic group call**	**T2: in-progress basic group call**	**UE B** and **UE C** set stored last call type change time and last user to change call type to corresponding values of GROUP CALL EMERGENCY END message, set stored current call type to "BASIC GROUP CALL" as well as current ProSe per-packet priority to corresponding value, and stop TFG13	**10.2.3.4.8.3**

2.3.3.2.Explicit downgrade from imminent peril call

In this scenario, an imminent peril group call is in progress. An authorized user (UE A) decides to downgrade the call. In addition, multiple TFG12 expirations are observed in this test scenario. It is assumed that if there are n "GROUP CALL IMMINENT PERIL END" messages, then the first n - 1 are lost, thus UE B and UE C only receive the nth message.

Scenario purpose: To observe that an imminent peril group call can explicitly be downgraded to a basic group call.

Table 43: Explicit downgrade from imminent peril call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T3: in-progress imminent peril group call**	**T3: in-progress imminent peril group call**	**T3: in-progress imminent peril group call**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE A** requests to downgrade the call	**T2: in-progress basic group call**	-	-	**UE A** sets stored current call type to “BASIC GROUP CALL” as well as per-packet priority, call type change time and last user to change call type accordingly, generates and sends GROUP CALL IMMINENT PERIL END, stops TFG14, starts TFG12 and initializes CFG12	**10.2.3.4.8.4**
2	T0 + TFG12	**UE A** TFG12 expires	-	-	-	**UE A** generates and sends GROUP CALL IMMINENT PERIL END, increments value of CFG12, and starts TFG12 *(because value of CFG12 is less than its upper limit)*	**10.2.3.4.8.5**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
3	T0 + n*TFG12	**UE A** TFG12 (*with CFG12 at upper limit - 1*)	-	-	-	**UE A** generates and sends GROUP CALL IMMINENT PERIL END, and increments value of CFG12	**10.2.3.4.8.5**
4	T0 + n*TFG12 + t_AB1	**UE B** and **UE C** receive GROUP CALL IMMINENT PERIL END	-	**T2: in-progress basic group call**	**T2: in-progress basic group call**	**UE B** and **UE C** set stored last call type change time and last user to change call type to corresponding values of GROUP CALL IMMINENT PERIL END message, set stored current call type to "BASIC GROUP CALL" as well as current ProSe per-packet priority to corresponding value, and stop TFG14	**10.2.3.4.8.6**

2.3.3.3.Implicit downgrade from emergency call

In this scenario, an emergency group call is in progress. The expiration of a user's timer, TFG13, invokes the implicit downgrade of the call type.

Scenario purpose: To observe that an emergency group call can implicitly (timer expiration) initiate the downgrade to a basic group call.

Table 44: Implicit downgrade from emergency call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	-	-
1	T0	**UE A** TFG13 expires	**T2: in-progress basic group call**	-	-	**UE A** sets stored current call type to "BASIC GROUP CALL" as well as current ProSe per-packet priority to corresponding value, and sets current UTC time as last call type change time of the call and own MCPTT user ID as last user to change call type of the call	**10.2.3.4.8.8**

2.3.3.4.Implicit downgrade from imminent peril call

In this scenario, an imminent peril group call is in progress. The expiration of a user’s timer, TFG14, invokes the downgrade of the call type.

Scenario purpose: To observe that an imminent peril group call can implicitly (timer expiration) initiate the downgrade to a basic group call.

Table 45: Implicit downgrade from imminent peril call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T3: in-progress imminent peril group call**	**T3: in-progress imminent peril group call**	**T3: in-progress imminent peril group call**	-	-

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
1	T0	**UE A** TFG14 expires	**T2: in-progress basic group call**	-	-	**UE A** sets stored current call type to "BASIC GROUP CALL" as well as current ProSe per-packet priority to corresponding value, and sets current UTC time as last call type change time of the call and own MCPTT user ID as last user to change call type of the call	**10.2.3.4.8.9**

2.3.4. Call release

2.3.4.1.Call release after call establishment

In this scenario, a group call is ongoing (any call type state). One of the users releases the call.

Scenario purpose: To observe that a user can release the call, regardless of the ongoing call type state.

Table 46: Call release after call establishment

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**UE C state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**T2: in-progress basic**, **T3: imminent peril** or **T1: emergency group call** (same call type state as UE B and UE C)	**T2: in-progress basic**, **T3: imminent peril** or **T1: emergency group call** (same call type state as UE A and UE C)	**T2: in-progress basic**, **T3: imminent peril** or **T1: emergency group call** (same call type state as UE A and UE B)	-	-
1	T0	**UE A** requests to release the call	**T0: waiting for call to establish**	-	-	**UE A** releases stored current call type, ProSe per-packet priority, Last call type change time and Last user to change call type	**10.2.3.4.10**

2.3.4.2.Call release before call establishment

In this scenario, there is no ongoing group call yet. One of the users attempts to establish a call but releases it before completion.

Scenario purpose: To observe that a user attempting to establish a call can release from the call prior to completion.

Table 47: Call release before call establishment

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T0: waiting for call to establish**	**T0: waiting for call to establish**	**T0: waiting for call to establish**	**-**	-
1	T0	**UE A** requests to release or reject the call	**-**	**-**	**-**	**UE A** releases stored current call type, ProSe per-packet priority, Last call type change time and Last user to change call type	**10.2.3.4.11**

2.3.5. Call merge

2.3.5.1.Call merge from different call type states

In this scenario, there are two ongoing group calls, (UE A and UE B) and (UE C), both group calls with the same group ID, but different call type states. UE A and UE B receive a group call announcement from UE C, and adjust their call type status accordingly.

Scenario purpose: To observe that the call type states are properly updated after the merge.

Table 48: Call merge from different call type states

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T3: in-progress imminent peril group call**	**T3: in-progress imminent peril group call**	**T1: in-progress emergency group call**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE C TFG2** expires	-	-	-	**UE C** sends GROUP CALL ANNOUNCEMENT	**NOTE: happens outside call type control**
2	T0 + t_CA1	**UE A** and **UE B** receive GROUP CALL ANNOUNCEMENT message	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	-	Follow 10.2.3.4.9 for lengthy detailed behavior	**10.2.3.4.9**

2.3.5.2.Call merge from same call type state

In this scenario, there are two ongoing group calls with the same group ID and same call type state, but one group call (UE C) has a lower call start time IE than the other group (UE A and UE B). A call merge occurs for UE A and UE B when each receive a GROUP CALL ANNOUNCEMENT from UE C. There is no state change, just updates to the internal variables.

Scenario purpose: To observe that the call type internal variables are properly updated after the merge.

Table 49: Call merge from same call type state

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	**T1: in-progress emergency group call**	-	-
1	T0	**UE C TFG2** expires	-	-	-	**UE C** sends GROUP CALL ANNOUNCEMENT	**NOTE: happens outside call type control**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
2	T0 + t_CA1	**UE A** and **UE B** receive GROUP CALL ANNOUNCEMENT message	-	-	-	**UE A** and **UE B** store the value of the Last call type change time IE, store the value of the Last user to change call type IE, and set PPPP to value for emergency group call	**10.2.3.4.9**

The scenario can be generalized similarly to the situation when all three UEs have the same group ID, the same Call Type and the same call start time, but the call identifier IE of UE C's group call is lower than that of the group call that UE A and UE B are in.

3. Test scenarios details – broadcast group call

According to 3GPP TS 24.379 v14.2.0 [4], a broadcast group call is "a group call where the initiating MCPTT user expects no response from the other MCPTT users, so that when the user's transmission is complete, so is the call." No floor control procedure is specified here, since it is assumed that there shall be no change in floor arbitration throughout the entire duration of the broadcast group call.

3.1. Broadcast group call – call control

3.1.1. Call setup

3.1.1.1. Call setup – establish a new call

In this scenario a user, UE A, decides to establish a broadcast group call (no existing one yet). UE B requires user acknowledgement, while UE C does not.

Scenario purpose: To observe that a new broadcast group call can be established and that users can join in.

Table 50: Call setup – establish a new call

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**B1: start-stop**	**B1: start-stop**	**B1: start-stop**	-	-
1	T0	**UE A** initiates a broadcast group call	**B2: in progress broadcast group call**	-	-	**UE A** generates and stores SDP, call identifier and MCPTT user ID of the call, generates and sends GROUP CALL BROADCAST, starts floor control as originating floor participant, establishes media session, and starts TFB2	**10.3.2.4.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL BROADCAST	-	**B3: pending user action**	**B2: in-progress broadcast group call**	**UE B** stores the call identifier, current call type, SDP, originating MCPTT user ID IE and MCPTT group ID IE of the call, and starts TFB3; **UE C** stores the call identifier, current call type, SDP, originating MCPTT user ID IE and MCPTT group ID IE of the call, establishes media session, starts floor control as terminating participant and starts TFB1	**10.3.2.4.2**
3	T0 + t_AB1 + t_B1	**UE B** accepts the incoming broadcast group call	-	**B2: in-progress broadcast group call**	-	**UE B** establishes media session, starts floor control as terminating participant, stops TFB3 and starts TFB1	**10.3.2.4.3**

3.1.1.2.Call setup – establish a new call with refusal to join

In this scenario, a user, UE A, decides to establish a broadcast group call (no existing one yet). UE B requires user acknowledgement, while UE C does not. One of the users, UE B, decides to reject the newly initiated call.

Scenario purpose: To observe that a new broadcast group call can be established and that a user can refuse to join the call.

Table 51: Call setup – establish a new call with refusal to join

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**B1: start-stop**	**B1: start-stop**	**B1: start-stop**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE A** initiates a broadcast group call	**B2: in progress broadcast group call**	-	-	**UE A** generates and stores SDP, call identifier and MCPTT user ID of the call, generates and sends GROUP CALL BROADCAST, starts floor control as originating floor participant, establishes media session, and starts TFB2	**10.3.2.4.1**
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL BROADCAST	-	**B3: pending user action**	**B2: in-progress broadcast group call**	**UE B** stores the call identifier, current call type, SDP, originating MCPTT user ID IE and MCPTT group ID IE of the call, and starts TFB3; **UE C** stores the call identifier, current call type, SDP, originating MCPTT user ID IE and MCPTT group ID IE of the call, establishes media session, starts floor control as terminating participant and starts TFB1	**10.3.2.4.2**
3	T0 + t_AB1 + t_B1	**UE B** rejects the incoming broadcast group call	-	**B4: ignoring same call ID**	-	**UE B** stops TFB3	**10.3.2.4.4**

And the corresponding figure is Fig. 14:

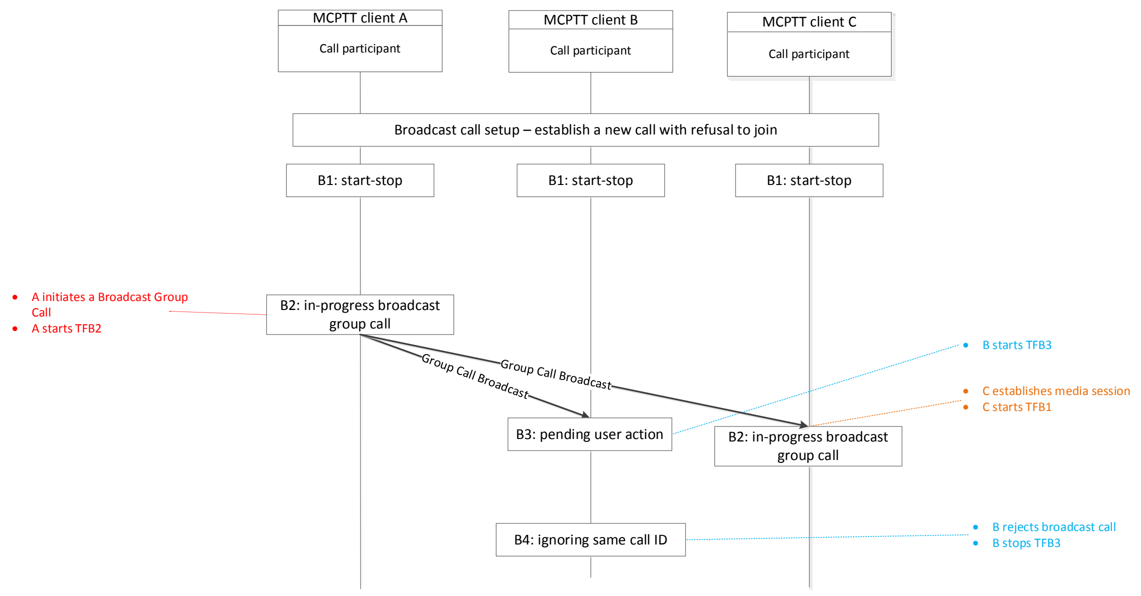

Figure 14: Broadcast call setup – establish a new call with refusal to join

3.1.2. Call release

3.1.2.1.Call release by the originating user

In this scenario, a broadcast group call is released by the originating user of the broadcast group call. UE A is considered the originating user. The originating user releases the call.

Scenario purpose: To observe that an ongoing broadcast group call can be released by the originating user.

Table 52: Call release by the originating user

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**B2: in progress broadcast group call**	**B2: in progress broadcast group call**	**B2: in progress broadcast group call**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
1	T0	**UE A** releases broadcast group call	**B1: start-stop**	-	-	**UE A** releases media session, generates and sends GROUP CALL BROADCAST END, stops floor control, and stops TFB2	**10.3.2.4.7**
2	T0 + t_AB1	**UE B** and **UE C** receive GROUP CALL BROADCAST END	-	**B1: start-stop**	**B1: start-stop**	**UE B** and **UE C** release media session and stop floor control	**10.3.2.4.8**

3.1.2.2.Call release by a participant

In this scenario, a terminating user (i.e., not the originating user of the broadcast group call) releases from the broadcast group call. UE A is considered the originating user. A terminating user, UE B, releases the call. We consider this scenario starts after UE A has already talked for a while (nearing TFB1 expiry), and that there is only enough time left to retransmit the GROUP CALL BROADCAST message once.

Scenario purpose: To observe that an ongoing broadcast group call can be released by a participating user.

Table 53: Call release by a participant

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
0	-	-	**B2: in progress broadcast group call**	**B2: in progress broadcast group call**	**B2: in progress broadcast group call**	-	-
1	T0	**UE B** releases in-progress broadcast group call	-	**B4: ignoring same call ID**	-	**UE B** releases the media session, and stops floor control	**10.3.2.4.6**
2	T0 + t_A1	**UE A** TFB2 expires	-	-	-	**UE A** generates and sends GROUP CALL BROADCAST, and starts TFB2	**10.3.2.4.9**

	Timeline	Triggering event	UE A state transition	UE B state transition	UE C state transition	Consequent behavior	Reference for details
3	T0 + t_A1 + t_AB1	**UE B** and **UE C** receive GROUP CALL BROADCAST	-	-	-	**UE B** checks that identifier in GROUP CALL BROADCAST matches with the stored call identifier, and starts TFB1; **UE C:** undefined action / behavior	**10.3.2.4.10 N/A**
4	T0 + t_C1 (*with t_C1 > t_A1 + t_AB1*)	**UE C** TFB1 expires	-	-	**B1: start-stop**	**UE C** clears stored call identifier	**10.3.2.4.11**
5	T0 + t_B1 (*with t_B1 > t_A1 + t_AB1*)	**UE B** TFB1 expires	-	**B1: start-stop**	-	**UE B** clears stored call identifier	**10.3.2.4.11**

And the corresponding figure is Fig. 15:

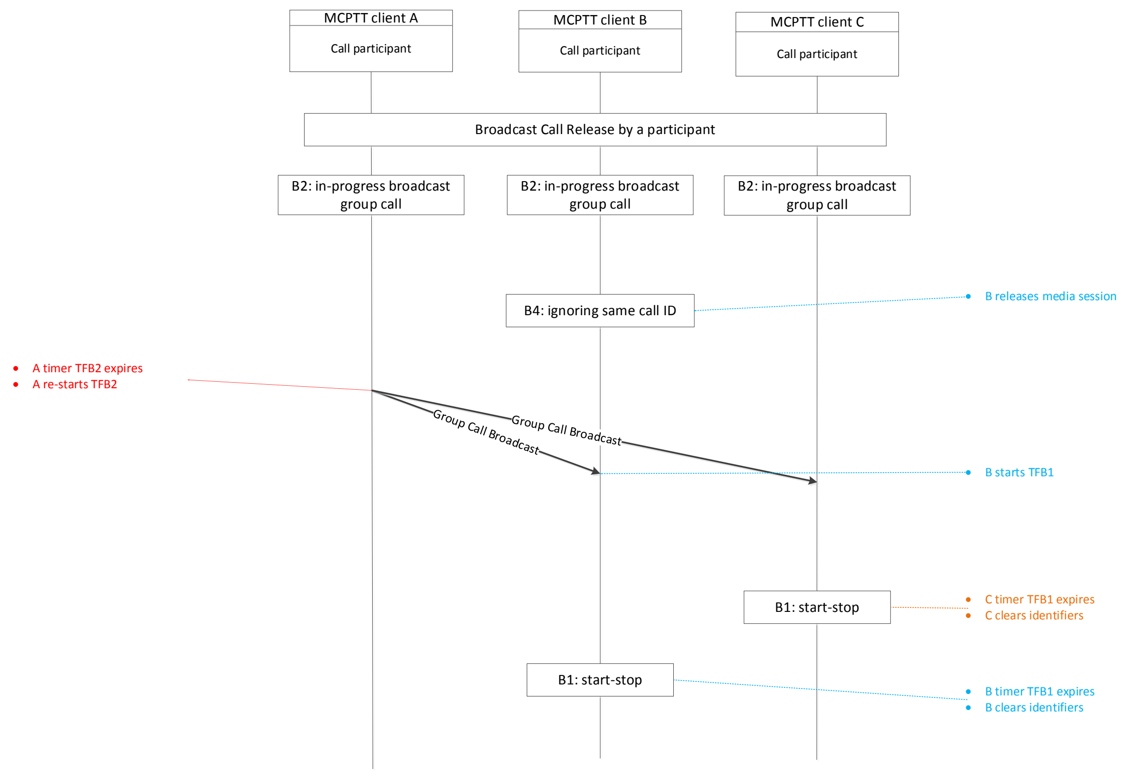

Figure 15: Broadcast call release by a participant

4. Test scenarios details – private call

A private call is a call occurring between two MCPTT clients. A private call can operate under automatic commencement mode or the manual commencement mode, modifying the way the call shall be established. 3GPP TS 24.379 v14.2.0 [4], clause 11 describes the logic and functioning of private calls, specifically clause 11.2 for off-network private call.

4.1. Private call – call control

4.1.1. Private call setup

4.1.1.1. Private call setup – establish new call in automatic mode

In this scenario a user, UE A, decides to establish a private call (no existing one yet). The call is made using automatic commencement mode.

Scenario purpose: To observe that a private call can be initiated (in automatic commencement mode).

Table 54: Private call setup – establish new call in automatic mode

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**P0: start-stop**	**P0: start-stop**	-	-
1	T0	Indication from **UE A** to initiate a private call to UE B	**P2: waiting for call response**	-	**UE A** generates and stores Call Identifier, MCPTT user ID, user ID of the callee, user location, offer SDP, creates call type control state machine and establishes end-to-end security (*if needed*), stores commencement mode as "AUTOMATIC COMMENCEMENT MODE", generates and sends PRIVATE CALL SETUP REQUEST, starts TFP1 and initializes CFP1	**11.2.2.4.2.1**
2	T0 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST	-	**P5: pending**	**UE B** stores Call Identifier IE as call identifier, MCPTT user ID of the caller IE as caller ID and own MCPTT user ID as callee IE, creates call type control state machine, generates and stores answer SDP, generates and sends PRIVATE CALL ACCEPT, establishes a media session, starts TFP4 and initializes CFP4	**11.2.2.4.3.2**

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
3	T0 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL ACCEPT	**P4: part of ongoing call**	-	**UE A** stores SDP answer, generates and sends PRIVATE CALL ACCEPT ACK, establishes a media session, starts floor control as terminating floor participant, stops TFP1, and starts TFP5	**11.2.2.4.2.8**
4	T0 + t_AB1 + t_BA1 + t_AB2	**UE B** receives PRIVATE CALL ACCEPT ACK	-	**P4: part of ongoing call**	**UE B** starts floor control as terminating MCPTT client, stops TFP4 and starts TFP5	**11.2.2.4.3.4**

And the corresponding figure is Fig. 16:

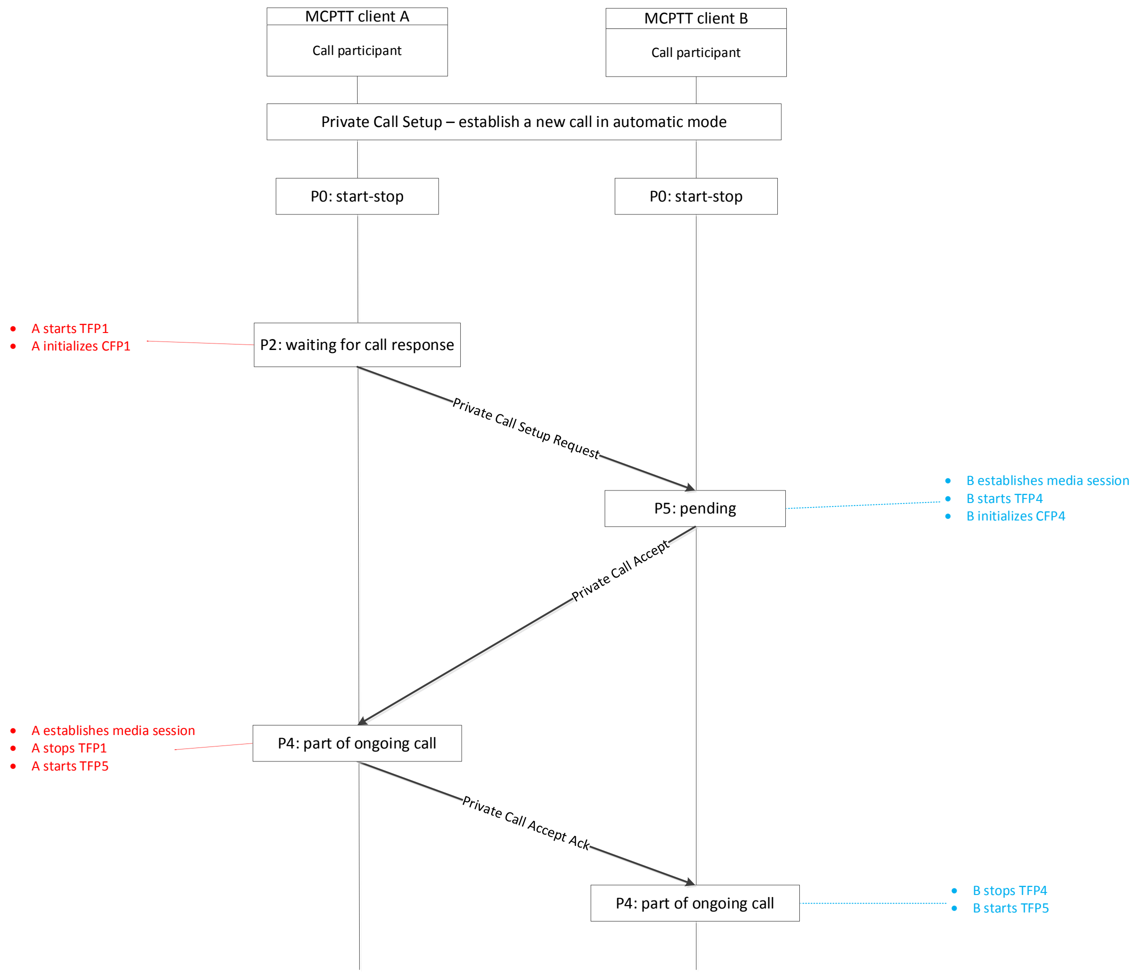

Figure 16: Private call setup – establish new call in automatic mode

4.1.1.2.Private call setup – establish and cancel new call in automatic mode

In this scenario a user, UE A, decides to establish a private call (no existing one yet). The call is made using automatic commencement mode. The UE initiating the call decides to cancel the call almost immediately afterwards and the SDP offer does not contain a dedicated key attribute.

Scenario purpose: To observe that a private call can be initiated (in automatic commencement mode) and canceled before it is established.

Table 55: Private call setup – establish and cancel new call in automatic mode

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**P0: start-stop**	**P0: start-stop**	--	-
1	T0	Indication from **UE A** to initiate a private call to UE B	**P2: waiting for call response**	-	**UE A** generates and stores Call Identifier, MCPTT user ID, user ID of the callee, user location, offer SDP, creates call type control state machine and establishes end-to-end security (*if needed*), stores commencement mode as "AUTOMATIC COMMENCEMENT MODE", generates and sends PRIVATE CALL SETUP REQUEST, starts TFP1 and initializes CFP1	**11.2.2.4.2.1**
2	T0 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST	-	**P5: pending**	**UE B** stores Call Identifier IE as call identifier, MCPTT user ID of the caller IE as caller ID and own MCPTT user ID as callee IE, creates call type control state machine, generates and stores answer SDP, generates and sends PRIVATE CALL ACCEPT, establishes a media session, starts TFP4 and initializes CFP4	**11.2.2.4.3.2**

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
3	T0 + t_A1	**UE A** cancels the private call request	**P3: waiting for release response**	-	**UE A** generates and sends PRIVATE CALL RELEASE, and starts TFP3	**11.2.2.4.2.9**
4	T0 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL ACCEPT	-	-	**UE A** discards the message	**11.2.2.4.6.1**
5	T0 + t_A1 + t_AB2	**UE B** receives PRIVATE CALL RELEASE	-	**-**	**UE B** discards the message	**11.2.2.4.6.1**
6	T0 + t_A1 + n*TFP3 (*with CFP3 at its upper limit*)	**UE A** TFP3 expires	**P1: ignoring same call ID**		**UE A** terminates the media session, releases the call type control state machine, and starts TFP7	**11.2.2.4.5.3**
7	T0 + t_A1 + n*TFP3 + TFP7	**UE A** TFP7 expires	**P0: start-stop**		**UE A** clears the stored call identifier	**11.2.2.4.5.7**

And the corresponding figure is Fig. 17:

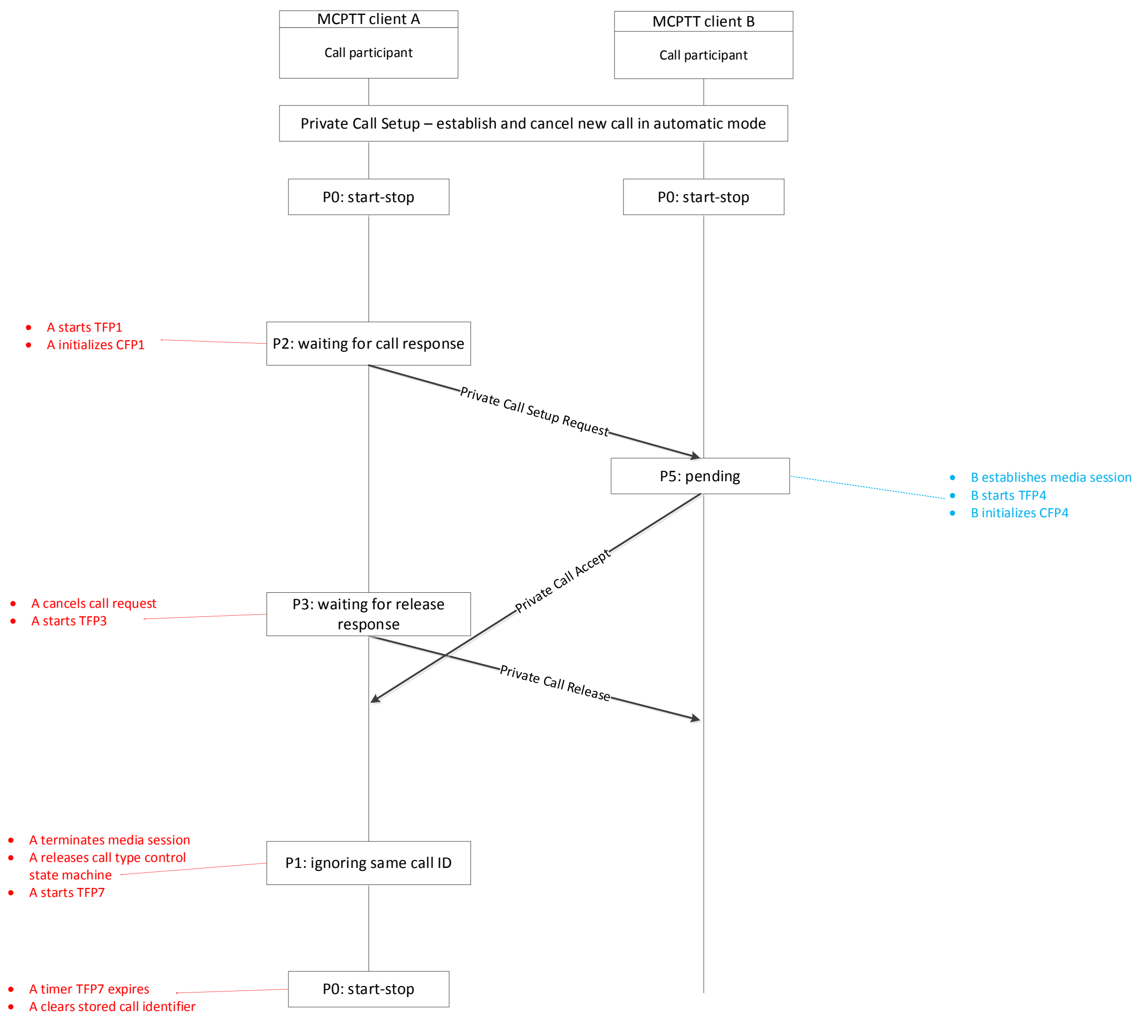
MCPTT client A
Call participant
MCPTT client B
Call participant
Private Call Setup – establish and cancel new call in automatic mode
P0: start-stop
P0: start-stop
A starts TFP1
A initializes CFP1
P2: waiting for call response
Private Call Setup Request
P5: pending
B establishes media session
B starts TFP4
B initializes CFP4
Private Call Accept
A cancels call request
A starts TFP3
P3: waiting for release response
Private Call Release
A terminates media session
A releases call type control state machine
A starts TFP7
P1: ignoring same call ID
A timer TFP7 expires
A clears stored call identifier
P0: start-stop

Figure 17: Private call setup – establish and cancel new call in automatic mode

4.1.1.3.Private call setup – establish new call in manual mode

In this scenario a user, UE A, decides to establish a private call (no existing one yet). The call is made using manual commencement mode, and the SDP offer does not contain a dedicated key attribute.

Scenario purpose: To observe that a private call can be initiated (in manual commencement mode), without any key attribute in the SDP offer.

Table 56: Private call setup – establish new call in manual mode

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**P0: start-stop**	**P0: start-stop**	-	-
1	T0	Indication from **UE A** to initiate a private call to UE B	**P2: waiting for call response**	-	**UE A** generates and stores Call Identifier, MCPTT user ID, user ID of the callee, user location, offer SDP, creates call type control state machine, stores commencement mode as "MANUAL COMMENCEMENT MODE", generates and sends PRIVATE CALL SETUP REQUEST, starts TFP1 and initializes CFP1	**11.2.2.4.2.1**
2	T0 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST	-	**P5: pending**	**UE B** stores Call Identifier IE as call identifier, MCPTT user ID of the caller IE as caller ID and own MCPTT user ID as callee ID, creates call type control state machine, generates and stores answer SDP, generates and sends PRIVATE CALL RINGING, and starts TFP2	**11.2.2.4.4.1**
3	T0 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL RINGING	-	-	**UE A** remains in its state	**11.2.2.4.2.3**
4	T0 + t_B1 (*with t_B1* > *t_AB1*)	**UE B** accepts the incoming private call	-	-	**UE B** generates and stores SDP, generates and sends PRIVATE CALL ACCEPT, establishes a media session, stops TFP2, starts TFP4 and initializes CFP4	**11.2.2.4.4.3**
5	T0 + t_B1 + t_BA2	**UE A** receives PRIVATE CALL ACCEPT	**P4: part of ongoing call**	-	**UE A** stores SDP answer, generates and sends PRIVATE CALL ACCEPT ACK, establishes a media session, starts floor control as terminating floor participant, stops TFP1 and TFP2, and starts TFP5	**11.2.2.4.2.8**

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
6	T0 + t_B1 + t_BA2 + t_AB2	**UE B** receives PRIVATE CALL ACCEPT ACK	-	**P4: part of ongoing call**	**UE B** starts floor control as terminating MCPTT client, stops TFP4 and starts TFP5	**11.2.2.4.4.5**

And the corresponding figure is Fig. 18:

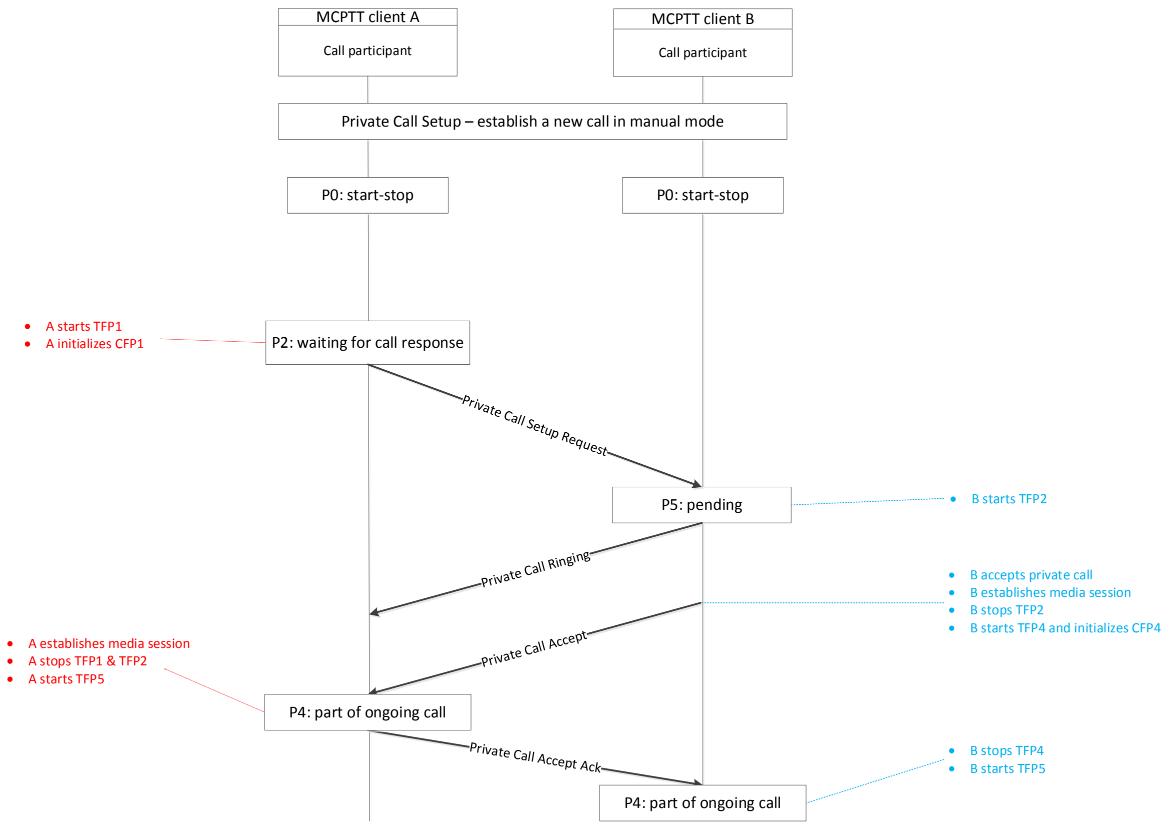

Figure 18: Private call setup – establish new call in manual mode

4.1.1.4.Private call setup – establish and cancel new call in manual mode

In this scenario a user, UE A, decides to establish a private call (no existing one yet). The call is made using manual commencement mode. The UE initiating the call decides to cancel it almost immediately after.

Scenario purpose: To observe that a private call can be initiated (in manual commencement mode) and canceled before it is established.

Table 57: Private call setup – establish and cancel new call in manual mode

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**P0: start-stop**	**P0: start-stop**	-	-
1	T0	Indication from **UE A** to initiate a private call to UE B	**P2: waiting for call response**	-	**UE A** generates and stores Call Identifier, MCPTT user ID, user ID of the callee, user location, offer SDP, creates call type control state machine, stores commencement mode as "MANUAL COMMENCEMENT MODE", generates and sends PRIVATE CALL SETUP REQUEST, starts TFP1 and initializes CFP1	**11.2.2.4.2.1**
2	T0 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST	-	**P5: pending**	**UE B** stores Call Identifier IE as call identifier, MCPTT user ID of the caller IE as caller ID and own MCPTT user ID as callee ID, creates call type control state machine, generates and stores answer SDP, generates and sends PRIVATE CALL RINGING, and starts TFP2	**11.2.2.4.4.1**
3	T0 + t_A1	**UE A** cancels the private call request	**P3: waiting for release response**	-	**UE A** generates and sends PRIVATE CALL RELEASE, and starts TFP3	**11.2.2.4.2.9**
4	T0 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL RINGING	-	-	**UE A** discards the call control message	**11.2.2.4.6.1**
5	T0 + t_A1 + t_AB2	**UE B** receives PRIVATE CALL RELEASE	-	**P1: ignoring same call ID**	**UE B** generates and sends PRIVATE CALL RELEASE ACK, stops TFP4, starts TFP7 and releases the call type control state machine	**11.2.2.4.4.8**
6	T0 + t_A1 + t_AB2 + t_BA2	**UE A** receives PRIVATE CALL RELEASE ACK	**P1: ignoring same call ID**	-	**UE A** terminates the media session, stops TFP3 and starts TFP7	**11.2.2.4.5.5**

And the corresponding figure is Fig. 19:

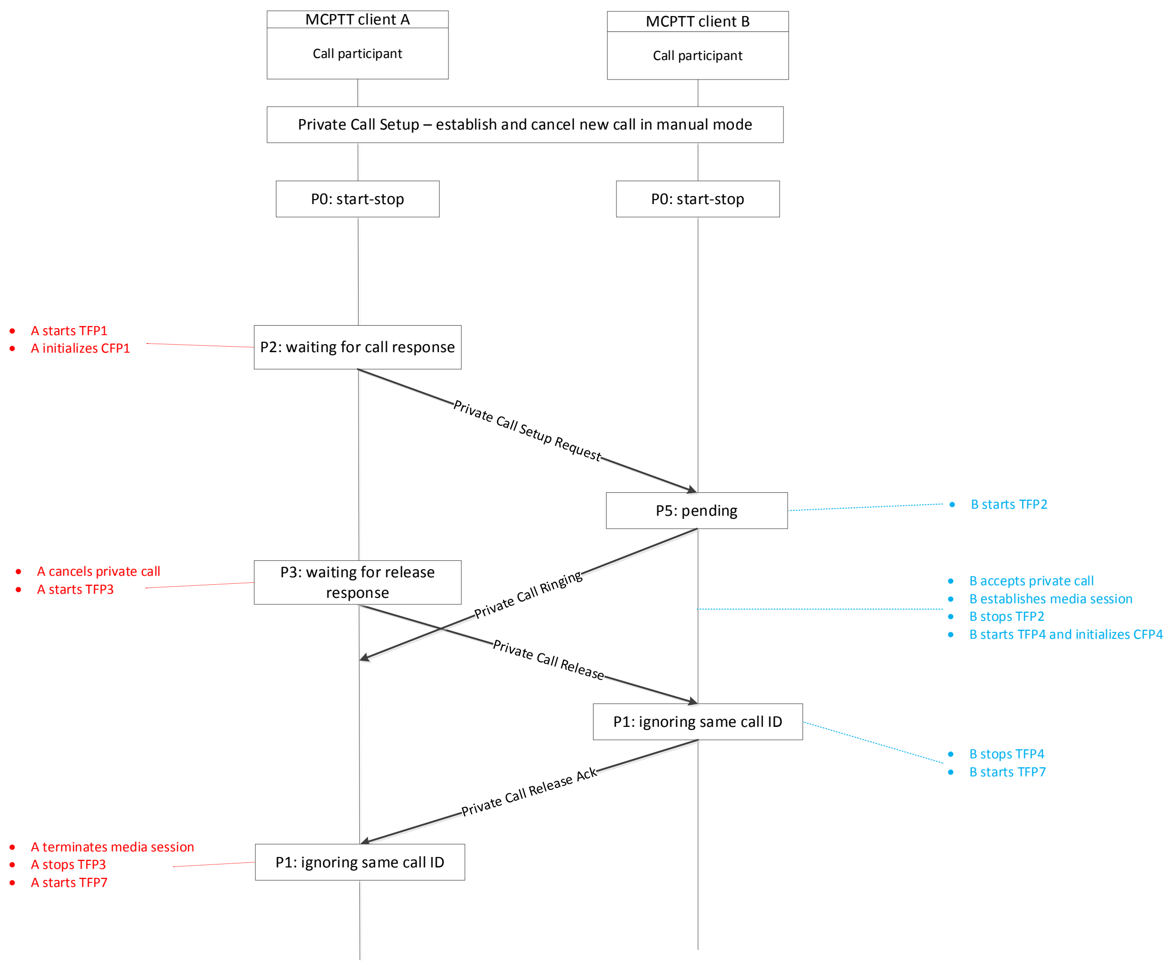

Figure 19: Private call setup – establish and cancel new call in manual mode

4.1.1.5. Private call setup – failure to establish new call

In this scenario a user, UE A, decides to establish a private call. The call is made using automatic commencement mode. Both users have different call identifiers, hence the call fails to establish. Two attempts are made by UE A.

Scenario purpose: To observe that a private call cannot be established between two users with different call identifiers.

Table 58: Private call setup – failure to establish a new call

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**P0: start-stop**	**P0: start-stop**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
1	T0	Indication from **UE A** to initiate a private call to UE B	**P2: waiting for call response**	-	**UE A** generates and stores Call Identifier, MCPTT user ID, user ID of the callee, user location, offer SDP, creates call type control state machine and establishes end-to-end security (*if needed*), stores commencement mode as "AUTOMATIC COMMENCEMENT MODE", generates and sends PRIVATE CALL SETUP REQUEST, starts TFP1 and initializes CFP1	**11.2.2.4.2.1**
2	T0 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST	-	**P1: ignoring same call ID**	**UE B** compares and acknowledges difference between Call Identifiers, stores Call Identifier IE of received message as call identifier, MCPTT user ID of caller IE as caller ID, own MCPTT user ID as callee ID, generates and sends PRIVATE CALL REJECT, and starts TFP7	**11.2.2.4.3.1**
3	T0 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL REJECT	**P1: ignoring same call ID**	-	**UE A** releases call control state machine, stops TFP1 and TFP2, and starts TFP7	**11.2.2.4.2.7**
4	T0 + t_AB1 + t_BA1 + t_A1	Indication from **UE A** to initiate a private call	**P2: waiting for call response**	-	**UE A** generates and stores Call Identifier, MCPTT user ID, user ID of the callee, user location, offer SDP, creates call type control state machine and establishes end-to-end security (*if needed*), stores commencement mode as "AUTOMATIC COMMENCEMENT MODE", generates and sends PRIVATE CALL SETUP REQUEST, starts TFP1 and initializes CFP1	**11.2.2.4.2.1**

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
5	T0 + t_AB1 + t_BA1 + t_A1 + t_AB2	**UE B** receives PRIVATE CALL SETUP REQUEST	-	-	**UE B** compares and acknowledges difference between Call Identifiers, stores Call Identifier IE of received message as call identifier, MCPTT user ID of caller IE as caller ID, own MCPTT user ID as callee ID, generates and sends PRIVATE CALL REJECT, and starts TFP7	**11.2.2.4.3.1**
6	T0 + t_AB1 + t_BA1 + t_A1 + t_AB2 + t_BA2	**UE A** receives PRIVATE CALL REJECT	**P1: ignoring same call ID**	-	**UE A** releases call control state machine, stops TFP1 and TFP2, and starts TFP7	**11.2.2.4.2.7**

And the corresponding figure is Fig. 20:

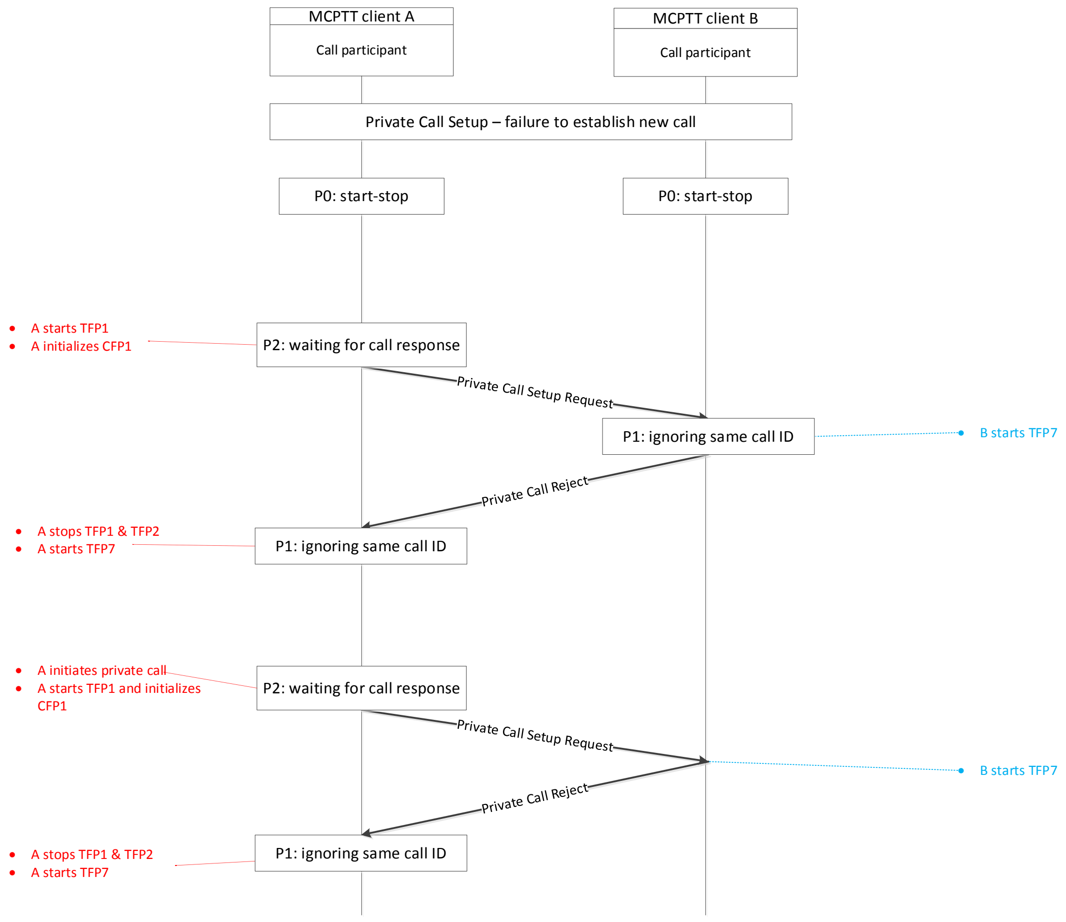

Figure 20: Private call setup – failure to establish a new call

4.1.2. Private call cancellation

4.1.2.1.Private call cancellation – cancel an ongoing call (normal)

In this scenario a user, UE A, decides to cancel an ongoing private call as part of the usual normal call release (i.e., the user wants to end the private call).

Scenario purpose: To observe that a user can cancel an ongoing private call.

Table 59: Private call cancellation – cancel an ongoing call (normal)

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**P4: part of ongoing call**	**P4: part of ongoing call**	-	-
1	T0	**UE A** releases the private call	**P3: waiting for release response**	-	**UE A** generates and sends PRIVATE CALL RELEASE, starts TFP3 and initializes CFP3	**11.2.2.4.5.1**
2	T0 + t_AB1	**UE B** receives PRIVATE CALL RELEASE	-	**P1: ignoring same call ID**	**UE B** generates and sends PRIVATE CALL RELEASE ACK, terminates media session, releases the call type control state machine and starts TFP7	**11.2.2.4.5.4**
3	T0 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL RELEASE ACK	**P1: ignoring same call ID**	-	**UE A** terminates the media session, stops TFP3 and starts TFP7	**11.2.2.4.5.5**

4.1.2.2.Private call cancellation – cancel an ongoing call (timer expiry)

In this scenario a user, UE A, decides to cancel an ongoing private call. However, those Private Call Release messages are lost and not received by UE B (out of range). Hence no response (i.e., Private Call Release Ack) message is received at UE A.

Scenario purpose: To observe that a user can cancel an ongoing private call using timer, due to a non-responding user (e.g., out of range).

Table 60: Private call cancellation – cancel an ongoing call (timer expiry)

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**P4: part of ongoing call**	**P4: part of ongoing call**	-	-
1	T0	**UE A** releases the private call	**P3: waiting for release response**	-	**UE A** generates and sends PRIVATE CALL RELEASE, starts TFP3 and initializes CFP3	**11.2.2.4.5.1**
2	T0 + TFP3	**UE A** TFP3 expires	-	-	**UE A** generates and sends PRIVATE CALL RELEASE, starts TFP3 and increments CFP3	**11.2.2.4.5.2**
3	T0 + n*TFP3	**UE A** TFP3 expires (*CFP3's upper limit*)	**P1: ignoring same call ID--**	-	**UE A** terminates the media session, releases the call type control state machine and starts TFP7	**11.2.2.4.5.3**
4	T0 + TFP5 *(with TFP5 > n*TFP3)*	**UE B** TFP5 expires	-	**P1: ignoring same call ID**	**UE B** terminates the media session, releases the call type control state machine and starts TFP7	**11.2.2.4.5.6**

And the corresponding figure is Fig. 21:

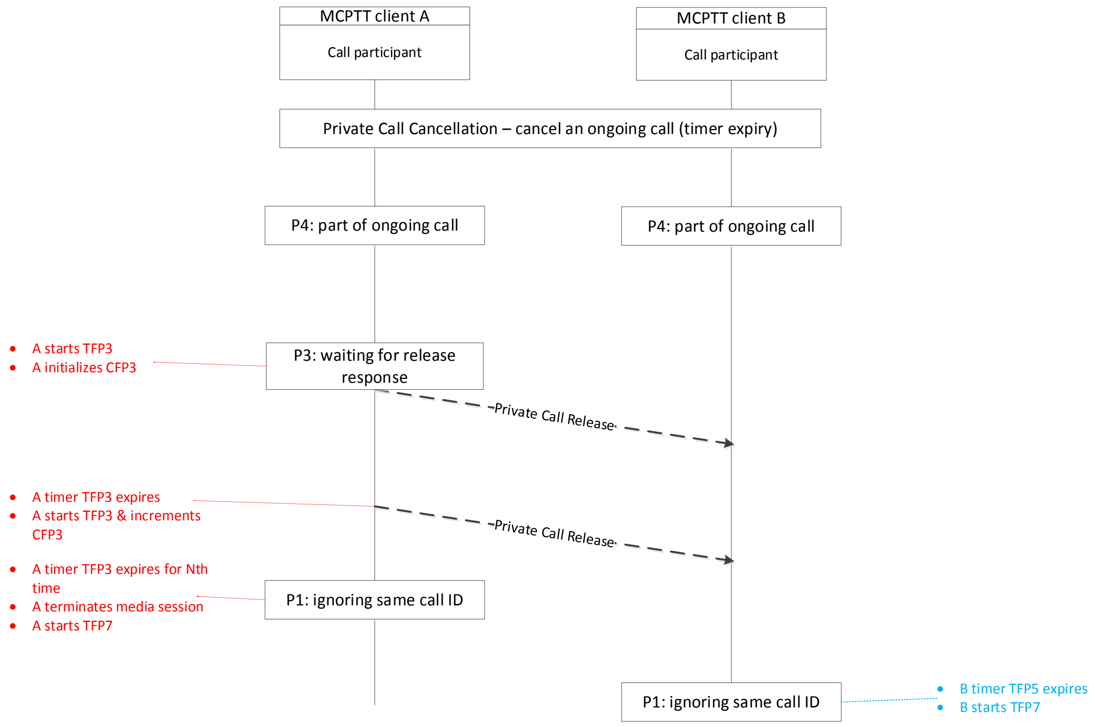

Figure 21: Private call cancellation – cancel an ongoing call (timer expiry)

4.1.3. Private call expiration

In this scenario a private call is ongoing. Timer TFP5 expires (lasted the maximum time for a private call), hence the call is terminated. The call is renewed afterwards.

Scenario purpose: To observe that a private call is terminated when the maximum duration for the call is reached (TFP5).

Table 61: Private call maximum duration expiration

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**P4: part of ongoing call**	**P4: part of ongoing call**	-	-
1	T0	**UE A** TFP5 expires	**P1: ignoring same call ID**	-	**UE A** releases the call type control state machine and terminates the media session, and starts TFP7	**11.2.2.4.5.6**

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
2	T0 + t_B1	**UE B** TFP5 expires	-	**P1: ignoring same call ID**	**UE B** releases the call type control state machine and terminates the media session, and starts TFP7	**11.2.2.4.5.6**
3	T0 + t_A1 (*with t_A1* > + *t_B1*)	Indication from **UE A** to initiate a private call	**P2: waiting for call response**	-	**UE A** generates and stores Call Identifier, MCPTT user ID, user ID of the callee, user location, offer SDP, creates call type control state machine and establishes end-to-end security (*if needed*), stores commencement mode as "MANUAL COMMENCEMENT MODE", generates and sends PRIVATE CALL SETUP REQUEST, starts TFP1 and initializes CFP1	**11.2.2.4.2.1**
4	T0 + t_A1 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST	-	**P5: pending**	**UE B** stores Call Identifier IE as call identifier, MCPTT user ID of the caller IE as caller ID and own MCPTT user ID as callee ID, creates call type control state machine, generates and stores answer SDP, generates and sends PRIVATE CALL RINGING, and starts TFP2	**11.2.2.4.4.1**
5	T0 + t_A1 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL RINGING	-	-	-	**11.2.2.4.2.3**

4.2. Private call – call type control

Note: Interaction dependence between private call control and private call type control.

4.2.1. Enter private call

In this scenario no ongoing private call is established yet. UE A eventually establishes a call. The stored emergency state is set to “False”. The various messages sent are originating from the private call control state machine, not the private call type state machine.

Scenario purpose: To observe that the private call state type is entered upon establishment of a private call.

Table 62: Enter private call

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**Q0: waiting for the call to be established**	**Q0: waiting for the call to be established**	-	-
1	T0	**UE A** initiates a Private Call	-	-	**UE A** sets current Call Type to "PRIVATE CALL" and stores current ProSe per-packet priority to MCPTT off-network private call corresponding value (*a PRIVATE CALL SETUP REQUEST is sent by UE A's basic call control state machine*)	**11.2.3.4.2**
2	T0 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST (*message sent from the basic call control state machine*)	-	-	**UE B** sets the current call type to "PRIVATE CALL" and sets current ProSe per-packet priority to MCPTT off-network private call corresponding value	**11.2.3.4.3**
3	T0 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL ACCEPT (*message sent from the basic call control state machine*)	**Q1: in-progress private call**	-	-	**11.2.3.4.4**

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
4	T0 + t_AB1 + t_BA1 + t_AB2	**UE B** receives PRIVATE CALL ACCEPT ACK (*message sent from the basic call control state machine*)	-	**Q1: in-progress private call**	-	**11.2.3.4.4**

4.2.2. Enter private emergency call

In this scenario no ongoing private call is established yet. UE A eventually establishes a call. The stored emergency state is set to "True".

Scenario purpose: To observe that the emergency private call state type is entered upon establishment of a private emergency call**.**

Table 63: Enter private emergency call

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**Q0: waiting for the call to be established**	**Q0: waiting for the call to be established**	-	-
1	T0	**UE A** initiates a Private Call	-	-	**UE A** sets current Call Type to "EMERGENCY PRIVATE CALL" and stores current ProSe per-packet priority to MCPTT off-network private call corresponding value (*a PRIVATE CALL SETUP REQUEST is sent by UE A's basic call control state machine*)	**11.2.3.4.2**
2	T0 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST (*message sent from the basic call control state machine*)	-	-	**UE B** sets the current call type to "EMERGENCY PRIVATE CALL" and sets current ProSe per-packet priority to MCPTT off-network private call corresponding value	**11.2.3.4.3**

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
3	T0 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL ACCEPT (*message sent from the basic call control state machine*)	**Q2: in-progress emergency private call**	-	**UE A** starts TFP8	**11.2.3.4.4**
4	T0 + t_AB1 + t_BA1 + t_AB2	**UE B** receives PRIVATE CALL ACCEPT ACK (*message sent from the basic call control state machine*)	-	**Q2: in-progress emergency private call**	**UE B** starts TFP8	**11.2.3.4.4**

4.2.3. Private call upgrade

4.2.3.1.Private call upgrade – upgrade call

In this scenario an ongoing private call is already established. UE A decides to upgrade the call to an emergency private call.

Scenario purpose: To observe that a private call can be upgraded to an emergency private call, with corresponding state types.

Table 64: Private call upgrade – upgrade call

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**Q1: in-progress private call**	**Q1: in-progress private call**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
1	T0	**UE A** upgrades call to Emergency Call	**Q2: in-progress emergency private call**	-	**UE A** checks if emergency leaf node from user profile is set to "True", generates and stores emergency offer SDP, updates caller ID as own MCPTT user ID and callee ID as MCPTT user ID of the other user, stores current user location, sets current call type to "EMERGENCY PRIVATE CALL", generates and sends PRIVATE CALL SETUP REQUEST, sets ProSe per-packet priority to corresponding value, starts TFP1 and initializes CFP1	**11.2.3.4.5.1**
2	T0 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST	-	**Q2: in-progress emergency private call**	**UE B** generates and stores emergency answer SDP, updates caller ID with MCPTT user ID of the caller IE and callee ID with own MCPTT user ID, generates and sends PRIVATE CALL ACCEPT, sets ProSe per-packet priority to corresponding value, sets current call type to "EMERGENCY PRIVATE CALL", and starts TFP8	**11.2.3.4.5.6**
3	T0 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL ACCEPT	-	-	**UE A** stores SDP answer IE as emergency SDP answer, generates and sends PRIVATE CALL ACCEPT ACK, establishes media session, stops TFP1 and starts TFP8	**11.2.3.4.5.3**

4.2.3.2.Private call upgrade – rejected upgrade call

In this scenario an ongoing private call is already established. UE A decides to upgrade the call to an emergency private call. The media session cannot be established on UE B’s side hence the upgrade is rejected. A TFP1 timer expiration occurs due to the first private call setup request message being lost.

Scenario purpose: To observe that a private call upgrade can be rejected by the called user.

Table 65: Private call upgrade – rejected upgrade call

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**Q1: in-progress private call**	**Q1: in-progress private call**	-	-
1	T0	**UE A** upgrades call to Emergency Call	**Q2: in-progress emergency private call**	-	**UE A** checks if emergency leaf node from user profile is set to "True", generates and stores emergency offer SDP, updates caller ID as own MCPTT user ID and callee ID as MCPTT user ID of the other user, stores current user location, sets current call type to "EMERGENCY PRIVATE CALL", generates and sends PRIVATE CALL SETUP REQUEST, sets ProSe per-packet priority to corresponding value, starts TFP1 and initializes CFP1	**11.2.3.4.5.1**
2	T0 + TFP1	**UE A** TFP1 expires	-	-	**UE A** may update user location, generates and sends PRIVATE CALL SETUP REQUEST, starts TFP1 and increments CFP1	**11.2.3.4.5.2**
3	T0 + TFP1 + t_AB1	**UE B** receives PRIVATE CALL SETUP REQUEST	-	-	**UE B** sets call identifier IE with the call identifier in the received message, sets MCPTT user ID of the caller IE and of the callee IE, sets Reason IE as either "FAILED" or "MEDIA FAILURE", generates and sends PRIVATE CALL REJECT	**11.2.3.4.5.6**
4	T0 + TFP1 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL REJECT	**Q1: in-progress private call**	-	**UE A** sets ProSe per-packet priority to corresponding value, sets current call type to "PRIVATE CALL" and stops TFP1	**11.2.3.4.5.4**

And the corresponding figure is Fig. 22:

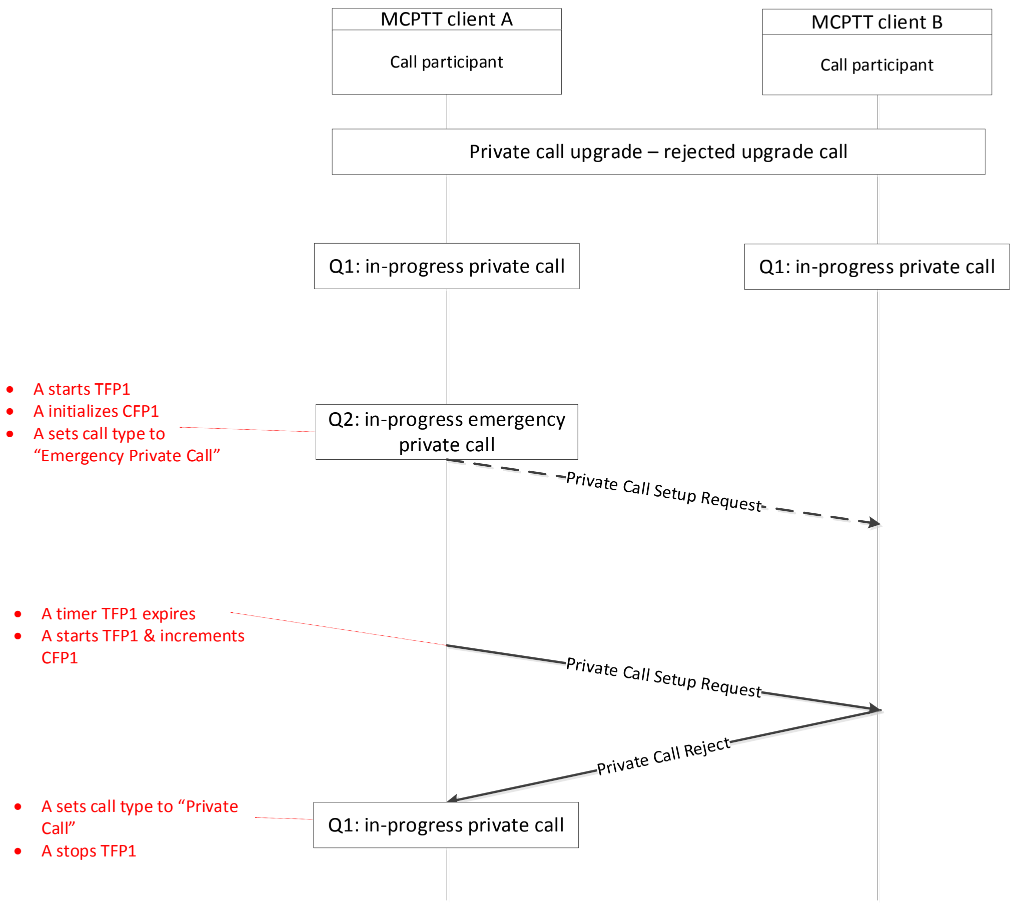
MCPTT client A
Call participant
MCPTT client B
Call participant
Private call upgrade – rejected upgrade call
Q1: in-progress private call
Q1: in-progress private call
A starts TFP1
A initializes CFP1
A sets call type to "Emergency Private Call"
Q2: in-progress emergency private call
Private Call Setup Request
A timer TFP1 expires
A starts TFP1 & increments CFP1
Private Call Setup Request
Private Call Reject
A sets call type to "Private Call"
A stops TFP1
Q1: in-progress private call

Figure 22: Private call upgrade – rejected upgrade call

4.2.3.3.Private call upgrade – failed upgrade call

In this scenario an ongoing private call is already established. UE A decides to upgrade the call to an emergency private call. However, in the meantime UE A and UE B get out of range of each other. UE B does not respond to the upgrade, hence UE A releases the call.

Scenario purpose: To observe that a private call upgrade fails if the user does not respond (e.g., out of range), and results in the call being released by the callee and called party receives an internal indication to release the call.

Table 66: Private call upgrade – failed upgrade call

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**Q1: in-progress private call**	**Q1: in-progress private call**	-	-

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
1	T0	**UE A** upgrades call to Emergency Call	**Q2: in-progress emergency private call**	-	**UE A** checks if emergency leaf node from user profile is set to "True", generates and stores emergency offer SDP, updates caller ID as own MCPTT user ID and callee ID as MCPTT user ID of the other user, stores current user location, sets current call type to "EMERGENCY PRIVATE CALL"", generates and sends PRIVATE CALL SETUP REQUEST, sets ProSe per-packet priority to corresponding value, starts TFP1 and initializes CFP1	**11.2.3.4.5.1**
2	T0 + TFP1	**UE A** TFP1 expires	-	-	**UE A** may update user location, generates and sends PRIVATE CALL SETUP REQUEST, starts TFP1 and increments CFP1	**11.2.3.4.5.2**
3	T0 + n*TFP1	**UE A** TFP1 expires (*CFP1 at its upper limit*)	**Q0: waiting for the call to be established**	-	**UE A** releases stored current call type, releases ProSe per-packet priority	**11.2.3.4.5.5**
4	T0 + t_B1 *(with t_B1 > n*TFP1)*	**UE B** releases the call	-	**Q0: waiting for the call to be established**	**UE B** releases stored current call type, releases ProSe per-packet priority	**11.2.3.4.7**

4.2.4. Private emergency call downgrade

4.2.4.1. Private emergency call downgrade – downgrade call TFP6 expiry

In this scenario an ongoing emergency private call is already established. UE A decides to downgrade the call to a private call. Timer TFP6 expires since the first PRIVATE CALL EMERGENCY CANCEL message is lost.

Scenario purpose: To observe that an ongoing emergency private call can be downgraded to a private call, even with the loss of a single message (PRIVATE CALL EMERGENCY CANCEL).

Table 67: Private emergency call downgrade – downgrade call TFP6 expiry

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**Q2: in-progress emergency private call**	**Q2: in-progress emergency private call**	-	-
1	T0	**UE A** cancels the emergency private call	**Q1: in-progress private call**	-	**UE A** generates and sends PRIVATE CALL EMERGENCY CANCEL, sets stored current call type to "PRIVATE CANCEL", stops TFP8, initializes CFP6 and starts TFP6	**11.2.3.4.6.1**
2	T0 + TFP6	**UE A** TFP6 expires	-	-	**UE A** generates and sends PRIVATE CALL EMERGENCY CANCEL, starts TFP6 and increments CFP6	**11.2.3.4.6.2**
3	T0 + TFP6 + t_AB1	**UE B** receives PRIVATE CALL EMERGENCY CANCEL	-	**Q1: in-progress private call**	**UE B** generates and sends PRIVATE CALL EMERGENCY CANCEL ACK, establishes a media session based on the SDP body of stored answer SDP, sets current call type to "PRIVATE CALL", sets ProSe per-packet priority to corresponding value, and stops TFP8	**11.2.3.4.6.5**
4	T0 + TFP6 + t_AB1 + t_BA1	**UE A** receives PRIVATE CALL EMERGENCY CANCEL ACK	-	-	**UE A** sets ProSe per-packet priority to corresponding value, establishes a media session based on the SDP body of stored answer SDP, and stops TFP6	**11.2.3.4.6.3**

And the corresponding figure is Fig. 23:

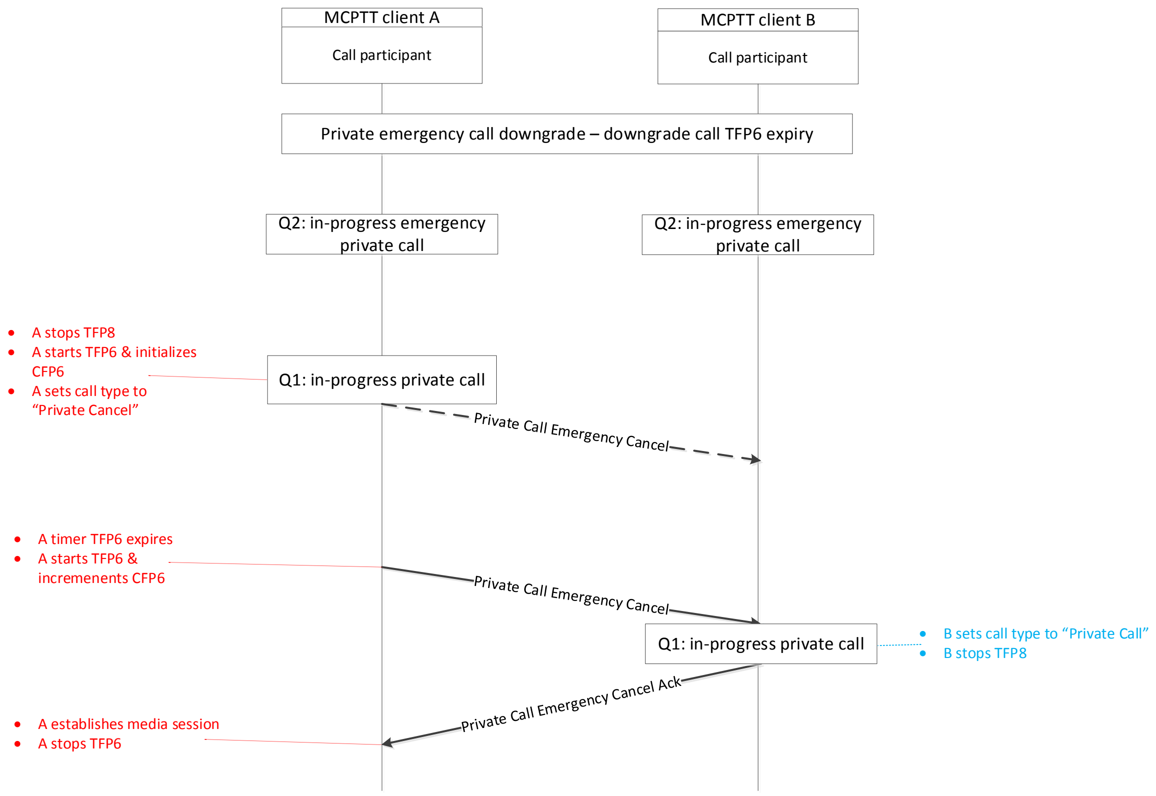
MCPTT client A
Call participant
MCPTT client B
Call participant
Private emergency call downgrade – downgrade call TFP6 expiry
Q2: in-progress emergency private call
Q2: in-progress emergency private call
• A stops TFP8
• A starts TFP6 & initializes CFP6
• A sets call type to "Private Cancel"
Q1: in-progress private call
Private Call Emergency Cancel
• A timer TFP6 expires
• A starts TFP6 & increments CFP6
Private Call Emergency Cancel
Q1: in-progress private call
• B sets call type to "Private Call"
• B stops TFP8
Private Call Emergency Cancel Ack
• A establishes media session
• A stops TFP6

Figure 23: Private emergency call downgrade – downgrade call TFP6 expiry

4.2.4.2.Private emergency call downgrade – downgrade call out of range

In this scenario an ongoing emergency private call is already established. UE A decides to downgrade the call to a private call. However, the "PRIVATE CALL EMERGENCY CANCEL" messages sent by UE A are lost (UE B is out of range) during this scenario. Thus, UE B does not respond to the downgrade, so UE A releases the call.

Scenario purpose: To observe that, when downgrading an ongoing emergency private call to a basic private call, if both users are out of range then the private call eventually gets released by UE A.

Table 68: Private emergency call downgrade – downgrade call out of range

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**Consequent behavior**	**Reference for details**
0	-	-	**Q2: in-progress emergency private call**	**Q2: in-progress emergency private call**	-	-
1	T0	**UE A** cancels the emergency private call	**Q1: in-progress private call**	-	**UE A** generates and sends PRIVATE CALL EMERGENCY CANCEL, sets stored current call type to "PRIVATE CANCEL", stops TFP8, initializes CFP6 and starts TFP6	**11.2.3.4.6.1**
2	T0 + TFP6	**UE A** TFP6 expires	-	-	**UE A** generates and sends PRIVATE CALL EMERGENCY CANCEL, starts TFP6 and increments CFP6	**11.2.3.4.6.2**
3	T0 + n*TFP6	**UE A** TFP6 expires (*with value of CFP6 at its upper limit*)	**Q0: waiting for the call to be established**	-	**UE A** releases stored current call type, releases stored ProSe per-packet priority	**11.2.3.4.6.4**

And the corresponding figure is Fig. 24:

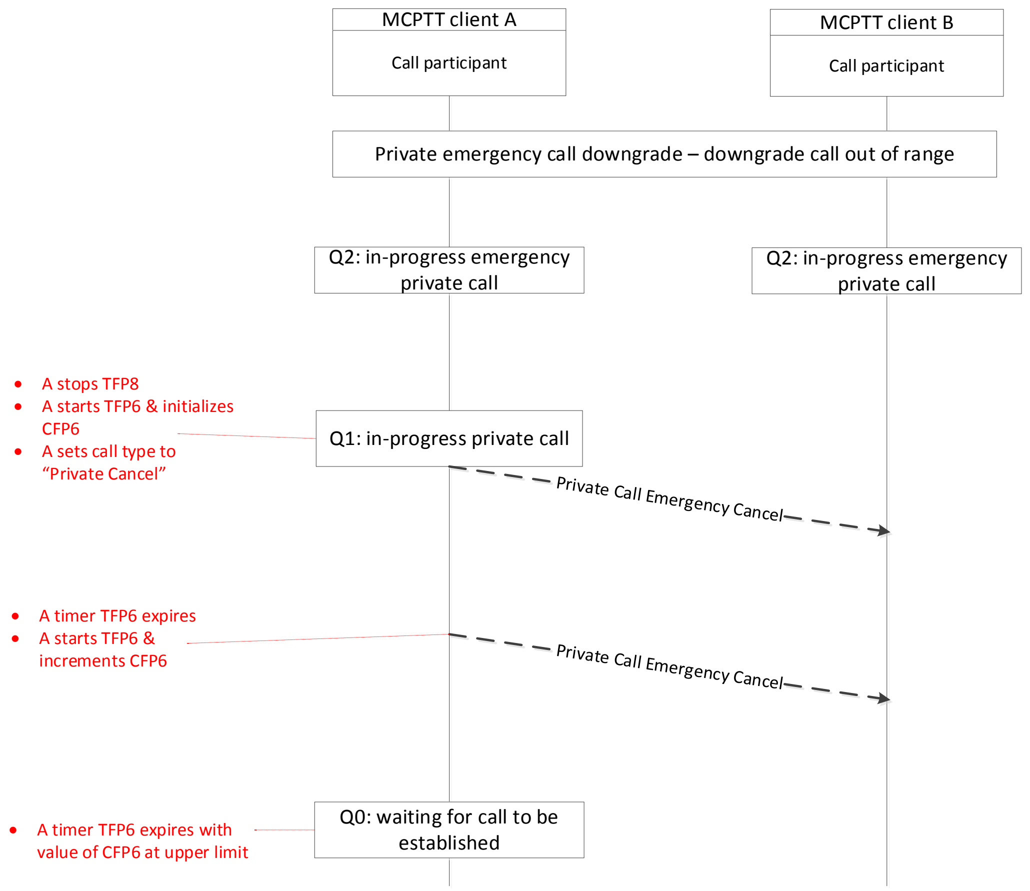
MCPTT client A
Call participant
MCPTT client B
Call participant
Private emergency call downgrade – downgrade call out of range
Q2: in-progress emergency private call
Q2: in-progress emergency private call
• A stops TFP8
• A starts TFP6 & initializes CFP6
• A sets call type to "Private Cancel"
Q1: in-progress private call
Private Call Emergency Cancel
• A timer TFP6 expires
• A starts TFP6 & increments CFP6
Private Call Emergency Cancel
• A timer TFP6 expires with value of CFP6 at upper limit
Q0: waiting for call to be established

Figure 24: Private emergency call downgrade – downgrade call out of range

4.2.4.3.Private emergency call downgrade – downgrade call TFP8 expiry

In this scenario an ongoing emergency private call is already established. Timer TFP8 expires at UE A and at UE B, so both UE A and UE B downgrade the call to a private call.

Scenario purpose: To observe that an ongoing emergency private call gets downgraded once the appropriate implicit timer (TFP8) expires.

Table 69: Private emergency call downgrade – downgrade call TFP8 expiry

	Timeline	Triggering event	UE A state transition	UE B state transition	Consequent behavior	Reference for details
0	-	-	**Q2: in-progress emergency private call**	**Q2: in-progress emergency private call**	-	-

	Timeline	**Triggering event**	**UE A state transition**	**UE B state transition**	**Consequent behavior**	**Reference for details**
1	T0	**UE A** TFP8 expires	**Q1: in-progress private call**	-	**UE A** establishes a media session based on SDP body of stored answer SDP, sets ProSe per-packet priority to corresponding value, and sets stored current call type to "PRIVATE CALL"	**11.2.3.4.6A**
2	T0 + t_B1	**UE B** TFP8 expires	-	**Q1: in-progress private call**	**UE B** establishes a media session based on SDP body of stored answer SDP, sets ProSe per-packet priority to corresponding value, and sets stored current call type to "PRIVATE CALL"	**11.2.3.4.6A**

5. Summary and future work

Test scenarios presented in this document are designed based on 3GPP's MCPTT off-network mode protocols strictly, with the objective to help the verification of protocol implementations. Benefits of these test scenarios are at least three folds:

- ✓ First, and the motivation of the effort, is that test scenarios are used to verify the ns-3 simulation model that our group developed. The simulator passed all tests.
- ✓ Second, during the process of test scenario design and simulation model verification, the completeness and correctness of 3GPP's MCPTT off-network mode operations were examined in great details. In the past one and a half years, our group has identified many errors and discrepancies in the MCPTT floor control and call control protocols, and, as a result, have submitted several CRs [8-19] addressing off-network mode operations to the 3GPP CT1 task group international meetings. All documents were well received, as all corrections were accepted.
- ✓ Third, but not least, it is possible to utilize these test scenarios to verify the MCPTT protocol implementations in other embodiments, e.g., prototype and commercial devices. It is also possible to develop more specific test scenarios that may be adopted by 3GPP, especially the sequence of message exchanges over the air, which are more observable externally based on the figures that accompany several test scenarios.

The above benefits may accelerate the deployment of MCV over LTE, especially for the out-of-coverage communication situation for public safety users. In the future, we plan to update existing test scenarios and simulation models to conform with 3GPP standards as progress is made.

Acknowledgments

The authors would like to thank Richard Rouil and David Cypher of Wireless Networks Division in Communications Technology Laboratory at National Institute of Standards and Technology for their technical discussions and suggestions, whose knowledge and insights have accelerated the achievement of the task and motivated the completion of this document.

References

[1] NPSTC Public Safety Communications Report, "Public Safety Broadband Push-to-Talk over Long Term Evolution Requirements," July 2013.
[2] 3GPP TS 22.179 v13.3.0, "Mission Critical Push To Talk (MCPTT) over LTE; Stage 1 (Release 13)", December 2015.
[3] 3GPP TS 23.179 v13.5.0, **"**Functional architecture and information flows to support mission critical communication services; Stage 2 (Release 13)", March 2017.
[4] 3GPP TS 24.379 v14.2.0, **"**Mission Critical Push To Talk (MCPTT) call control; Protocol specification (Release 14)", June 2017.
[5] 3GPP TS 24.380 v14.3.0, **"**Mission Critical Push To Talk (MCPTT) media plane control; Protocol specification (Release 14)", June 2017.
[6] 3GPP TS 24.381 v13.4.0, **"**Mission Critical Push To Talk (MCPTT) group management; Protocol specification (Release 13)", March 2017.
[7] 3GPP TS 24.384 v13.4.0, **"**Mission Critical Push To Talk (MCPTT) configuration management; Protocol specification (Release 13)", March 2017.
[8] US Department of Commerce, CT1 #99, "Corrections needed for alignment within TS", C1-163707 (revision of C1-163265), Tenerife, Spain, July 2016
[9] US Department of Commerce, FirstNet, CT1 #105, "Corrections to off-network floor control procedures (Rel-13)", C1-173438 (revision of C1-172949), Krakow, Poland, August 2017
[10] US Department of Commerce, FirstNet, CT1 #105, "Corrections to off-network floor control procedures (Rel-14)", C1-173439 (revision of C1-172949), Krakow, Poland, August 2017
[11] US Department of Commerce, FirstNet, CT1 #105, "Explicit corrections to off-network floor control procedures and steps (Rel-13)", C1-173440 (revision of C1-172950), Krakow, Poland, August 2017
[12] US Department of Commerce, FirstNet, CT1 #105, "Explicit corrections to off-network floor control procedures and steps (Rel-14)", C1-173441 (revision of C1-172950), Krakow, Poland, August 2017
[13] US Department of Commerce, FirstNet, CT1 #105, "Corrections to off-network call control procedures (Rel-13)", C1-173449 (revision of C1-173150), Krakow, Poland, August 2017
[14] US Department of Commerce, FirstNet, CT1 #105, "Corrections to off-network call control procedures (Rel-14)", C1-173450 (revision of C1-173150), Krakow, Poland, August 2017
[15] US Department of Commerce, CT1 #107, "Off-network call type control procedures: merge of two calls (Rel-13)", C1-175223 (revision of C1-174780), Reno, USA, November 2017
[16] US Department of Commerce, CT1 #107, "Off-network call type control procedures: merge of two calls (Rel-14)", C1-175224 (revision of C1-174780), Reno, USA, November 2017
[17] US Department of Commerce, CT1 #107, "Off-network Broadcast group call procedures (Rel-13)", C1-175225 (revision of C1-174781), Reno, USA, November 2017
[18] US Department of Commerce, CT1 #107, "Off-network Broadcast group call procedures (Rel-14)", C1-175226 (revision of C1-174781), Reno, USA, November 2017

[19] US Department of Commerce, CT1 #107, “Companion document for Off-network Broadcast Group Call procedures updates”, C1-174782, Reno, USA, November 2017

Appendix: Scenario usages

This appendix gives a few examples for using the scenarios described in this document in other areas of expertise, such as software testing, conformance testing, and interoperability testing.

- Software testing:

When using the scenarios for software tests, they can be used to derive white box tests, because some scenarios lack the necessary control and events to perform black box testing. For black box testing, it is assumed that input, whether it be a user indication or a message from a peer system, can be observed at the time it is consumed by the state machine being tested, and that any outputs, such as a message, can also be observed after it is produced by the state machine being tested. Thus, only inputs and outputs are used to derive a black box test. It is also assumed that the inputs and outputs can be recorded over a period of time, such that one can organize them into a sequence of events to compare with the order of events described in the scenario timeline. Therefore, to pass a test, not only should all inputs and outputs occur, but those inputs and outputs should happen in the same chronological order of the scenario timeline. Any other events or order of events should be considered a failure. In addition to the assumptions and requirements for black box testing, white box testing requires observation of state transitions, the starting, stopping, and expiration of timers, as well as changes in counter values to determine the result of a test. Thus, white box tests require additional checks to verify that after each event (e.g., user indication, reception of a message or timer expiration) all state machine variables contain the proper value based on the changes described in the scenario. These checks are conditional and should evaluate to true or false.

Black Box Testing Example

**Note: Only state machines A and B are considered for black box testing in the following scenario because the necessary control to ensure that UE B's TFG2 timer expires before UE C's TFG2 timer, which occurs at step 3 of the scenario, may be lacking.*

For the scenario, "Call setup – join call" defined in section 2.2.1.1, the following needs to be done:

1. Instantiate two "Basic Group Call" call control state machines, which we will call A and B
2. Interact with the state machines so that state machine B is in a basic group call while A is not
3. Now the following should be observed, in order, after providing an indication to A to "Initiate a basic group call":
    a. Observe that A sends a GROUP CALL PROBE message
    b. Observe that B receives the GROUP CALL PROBE message
    c. Observe that B sends a GROUP CALL ANNOUNCEMENT message with fields coded properly within a certain timer period based on TFG2 + delta.
    d. Observe that A receives the GROUP CALL ANNOUNCEMENT message sent by B

If all events are observed and observed in the order in which they are described, then model has passed the test that was derived from the scenario. If not, then the model has failed the test.

White Box Testing

For the scenario, "Call setup – join call" defined in section 2.2.1.1, the following needs to be done:

1. Instantiate three "Basic Group Call" call control state machines, which we will call A, B, and C
2. Interact with the state machines so that state machines B and C are in the same basic group call while A is not
3. Now the following should be observed and checked, in order, after providing an indication to A to "Initiate a basic group call":
    a. Observe that A sends a GROUP CALL PROBE message
    b. Checks for A:
        i. Stored MCPTT group ID matches the group ID
        ii. The corresponding Call Type Control state machine exist
        iii. The timer TFG1 is running
        iv. The timer TFG3 is running
        v. Current state is the "S2: waiting for call announcement" state
    c. Observe that:
        i. B receives the GROUP CALL PROBE message
            - Checks for B:
                a. The timer TFG2 is running
                b. The timer TFG6 is running
                c. Stored probe response for the call is set to "true"
                d. Current state is the "S3: part of ongoing call" state

        -AND-

        ii. C receives the GROUP CALL PROBE message
            - Checks for C:
                a. The timer TFG2 is running
                b. The timer TFG6 is running
                c. Stored probe response for the call is set to "true"
                d. Current state is the "S3: part of ongoing call" state
    d. Observe that B's TFG2 timer expires
    e. Observe that B sends a GROUP CALL ANNOUNCEMENT message with fields coded properly within a certain timer period based on TFG2 + delta
        i. Checks for B:
            - The timer TFG2 is running
            - The timer TFG6 is running
            - Stored probe response for the call is set to "false"
            - Current state is the "S3: part of ongoing call" state

f. Observe that A receives the GROUP CALL ANNOUNCEMENT message sent by B
    i. Checks for A:
        - The timer TFG6 is running
        - The timer TFG2 is running
        - The timer TFG1 is NOT running
        - The timer TFG3 is NOT running
        - The stored SDP matches SDP received in the message
        - The stored call ID matches call ID received in the message
        - The stored originating user ID matches originating user ID received in the message
        - The stored refresh interval matches the refresh interval received in the message
        - The stored call start time matches the call start time received in the message
        - The floor control state machine has been initialized
        - Current state is the “S3: part of ongoing call” state

g. Observe that C receives the GROUP CALL ANNOUNCEMENT message sent by B
    i. Checks for C:
        - The timer TFG2 is running
        - Stored probe response for the call is set to “false”

If all events are observed, all events are observed in the order in which they are described, and all checks evaluated to true, i.e., all expected values and statuses were found, then the model has passed the test that was derived from the scenario. If not, then the model has failed the test.

- Conformance testing:

To use this scenario for conformance testing, one must examine one UE column at a time.

Test 1

Conformance test for UE A (device under test)
First one would need to create a test purpose based on the scenario.
For example: Test purpose: To determine that the Device Under Test (DUT) can join a group call.
Second one would need to create a preamble to place the device under test into the start state stated in the cell in the row labeled with a zero and under the UE A state transition column.
For example: Preamble: Turn on device.
Next the first action of conformance test would be to have UE A initiate the joining of a group call and to observe that UE A sends a GROUP CALL PROBE message. This information is contained in the row labeled 1. *NOTE: If this was being used as an internal software test, where one has access to the internal information, then one could also confirm*

that there was a state transition from "S1: Start-stop" to "S2: waiting for call announcement" as stated in row 1 column UE A state transition and that timers TFG1 and TFG3 were started as stated in the Consequent behavior column.
If any other message is sent by UE A, then the test would be considered a failure or inconclusive.
Continuing the tester (acting as UE B) would then send a GROUP CALL ANNOUNCEMENT message properly coded to UE A.
Since there is no observable action by UE A upon the receipt of the GROUP CALL ANNOUNCEMENT message, one would need to provide an observable ending, such as observe that the UE A hears the transmission on the group that was just joined.
If one hears the transmission of the group call via UE A, then the test verdict is pass.
If this does not occur, then the test verdict is fail.

Test 2

Conformance test for UE B (device under test)
First one would need to create a test purpose based on the scenario.
For example: Test purpose: To determine that the DUT responds to a request to join a group call with the properly coded message.
One would read the cell in the row labeled with a zero and the UE B state transition column to determine the start state which is "S3: part of ongoing-call". Thus a preamble would be needed to place UE B (device under test) in this state.
For example: Preamble: Have the tester establish a group call with DUT.
The first action of the conformance test would be to have the tester (acting as UE A) to send a GROUP CALL PROBE message (*coded with a different user identifier than the one used in the preamble*) to UE B. This information is contained in the row labeled 1.
The tester (acting as UE A) would wait to see if UE B (device under test) sends a GROUP CALL ANNOUNCEMENT message with the properly coded fields back to the tester using the information contained in both row 2 and row 3 Consequent behavior column within a timer value that is TFG2 + delta.
If a GROUP CALL ANNOUNCEMENT message is received with fields properly coded, then the test verdict is pass.
Otherwise fail.

- Interoperability testing:

**Note: Only state machines A and B are considered for interoperability testing in the following scenario because the necessary control to ensure that UE B's TFG2 timer expires before UE C's TFG2 timer, which occurs at step 3 of the scenario, may be lacking.*

To use this scenario for interoperability testing, one will consider the observable interfaces between UE A and UE B.
In this case UE A and UE B are required and thus the pair of interfaces is observable.
Create a test purpose:
For example: Observe that a UE (UE A) can join an ongoing group call (started by UE B).
Preamble:

- Turn on one UE (UE B)
- Cause UE B to establish a group call
- Begin speaking on the group call

Test step 1: Turn on UE A

Test step 2: Cause UE A to want to join the existing group call.

1: Observe that UE A sends a GROUP CALL PROBE message

2: Observe that UE B sends a GROUP CALL ANNOUNCEMENT message with fields coded properly within a certain timer period based on TFG2 + delta

3: Observe (listen) that UE A hears the conversation on the group call.

If all events are observed, then verdict is pass.

If not, then verdict is fail.