Modeling MCPTT and User Behavior in ns-3

Wesley Garey[1][a], Thomas R. Henderson[2][b], Yishen Sun[1][c], Richard Rouil[1][d]
and Samantha Gamboa[3,4][e]

[1]*Wireless Networks Division, National Institute of Standards and Technology, Gaithersburg, Maryland, U.S.A.*
[2]*Department of Electrical & Computer Engineering, University of Washington, Seattle, Washington, U.S.A.*
[3]*National Institute of Standards and Technology, Gaithersburg, Maryland, U.S.A.*
[4]*Prometheus Computing LLC, Sylva, North Carolina, U.S.A.*

Keywords: D2D, LTE, MCPTT, Modeling, ns-3, ProSe, Public Safety, Simulation.

Abstract: To support the advancement of public safety communications technology, the Third Generation Partnership Project (3GPP) has created several standards to define Mission Critical Push-To-Talk (MCPTT) over Long Term Evolution (LTE) networks. As this is a new service that can be used in dire situations, it is imperative that the behavior and performance meet the expectations of first responders. This paper introduces an extension to the network simulator, ns-3, that models MCPTT and user Push-To-Talk (PTT) activity, so that researchers can gain insights and evaluate the performance of this service. In this paper we will describe MCPTT based on 3GPP definitions, the implementation of the MCPTT model in ns-3, and some results, including Key Performance Indicators (KPIs), that can be extracted from this model.

Disclaimer: Certain commercial products are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor is it intended to imply that the commercial products identified are necessarily the best available for the purpose.

1 INTRODUCTION

Land Mobile Radio (LMR) systems for two-way voice communications have been extensively used by public safety, first responder systems for decades. Many LMR systems (analog and digital) are presently deployed worldwide. However, in an effort to improve first responder communications, public safety agencies worldwide are adopting broadband cellular systems, i.e. 4G Long Term Evolution (LTE) and eventually 5G Systems (5GS). Besides offering new broadband data services and prioritized service to first responders, a key new service called Mission Critical Push-To-Talk (MCPTT) is being deployed, with the long term objective of replacing traditional LMR systems. MCPTT is based on a series of specifications by the Third Generation Partnership Project (3GPP) (3GPP, 2019), (3GPP, 2017a), (3GPP, 2017b).

From the user's perspective, LMR and MCPTT should offer a similar service experience, but the underlying technology supporting both (dedicated analog or digital radio channels, versus a shared, general-purpose, fixed-infrastructure cellular network) is significantly different. Successful voice communication despite possibly challenging network conditions is paramount for safety and mission effectiveness among first responders (Sun et al., 2019). Because of this, it is imperative that thorough studies are conducted to evaluate the operation and performance of MCPTT. Field trials, lab testing, and interoperability exercises will be essential to fully vet MCPTT and instill confidence that it can be used to complement and possibly even replace LMR systems. High fidelity network simulations offer another framework to evaluate future MCPTT performance in ways that small-scale lab and field testing cannot easily accomplish. In particular, network simulators offer the ability to scale to large network sizes and public safety

[a] https://orcid.org/0000-0003-1572-8208
[b] https://orcid.org/0000-0001-8091-2787
[c] https://orcid.org/0000-0002-6334-253X
[d] https://orcid.org/0000-0003-0387-0880
[e] https://orcid.org/0000-0002-6476-7003

Garey, W., Henderson, T., Sun, Y., Rouil, R. and Gamboa, S.
Modeling MCPTT and User Behavior in ns-3.
DOI: 10.5220/0010513200300041
In *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2021)*, pages 30-41
ISBN: 978-989-758-528-9

incident scenarios involving hundreds or even thousands of notional MCPTT users, and they allow for completely repeatable experiments.

This paper reports on what we believe to be the first publicly available simulation model for MCPTT, based on the network simulator, ns-3[1], whose 4G LTE models have been extended by several of our previous efforts (Rouil et al., 2017), (Gamboa et al., 2019), (Garey et al., 2020) to support models for 3GPP Proximity Services (ProSe), Device-to-Device (D2D) communications, and the User Equipment (UE) relay service, UE-to-Network Relay. Our MCPTT model provides standards-aligned implementations of the call control and floor control protocols defined by 3GPP, for both on-network and off-network operations, as well as traces to capture standardized Key Performance Indicator (KPI) measurements such as mouth-to-ear latency and network access time.

Section 2 gives some general information about MCPTT and its core components. Section 3 presents works related to this paper. Section 4 describes the ns-3 model, including its features, outputs, verification and limitations. Section 5 presents the reader with a case study to show how one can simulate MCPTT using our model. Finally, Section 6 provides concluding remarks.

2 BACKGROUND

A Push-To-Talk (PTT) service is used to provide an arbitrated method for two or more users to communicate. In terms of a simple voice communication service, at least from the user's perspective, this service could provide a means of communication that is very similar to that of what can be achieved with walkie-talkies. In the ideal case, when a user wants to talk, the user will request permission to do so, traditionally, by pushing the PTT button on their communication device (3GPP, 2019), and then the service is expected to handle this request by allocating resources and granting permissions accordingly to enable communication between the users. MCPTT is an advanced PTT service based on LTE that can be used by first responders during mission critical situations (3GPP, 2019). Thus, it is expected that MCPTT services will be available regardless of the user's connectivity with an LTE network (3GPP, 2019). This is why MCPTT supports two modes of operation: on-network and off-network.

On-network mode follows the traditional client-server architecture, where the PTT service is provided by a centralized MCPTT server via the internet that is accessible through the LTE core network. The off-network case follows the peer-to-peer architecture, with the PTT service being provided by the client devices in a distributed manner. This is made possible with the use of ProSe, which enables D2D communication for devices in close proximity. In this case the MCPTT client is responsible for allocating resources and performing arbitration. In either mode, it is expected that this service will be available, allow users to request permission to talk, and provide a deterministic mechanism to arbitrate between those requests (3GPP, 2019).

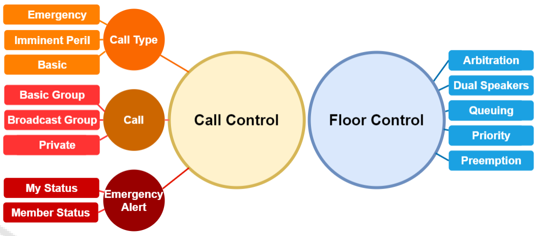


Figure 1: Main MCPTT components.

There are two main components that comprise MCPTT, call control and floor control. These components are defined respectively in 3GPP (2017a) and 3GPP (2017b). Several of the primary elements associated with each are also shown in Figure 1. Call control is responsible for managing call state coordination, including aspects such as resource allocation, management of logical channels, user information, group affiliation, and call type. Call control supports various types of calls, including basic group calls, private calls, and broadcast group calls. Group calls may be established for communication between two or more users in a particular MCPTT group, while private calls can exist between two users regardless of their group affiliation. Call control also supports basic, imminent peril, and emergency call types. These call types indicate the status of a call and are used to determine which network resources will be used by that particular call. Calls can be established, upgraded, downgraded, joined, left, rejoined, and released, and these actions are all accomplished through call control.

Floor control is responsible for providing the arbitration logic to determine who can talk at any given time during an ongoing call (3GPP, 2017b). MCPTT floor control currently supports queuing of requests, priority, and preemption. On-network floor control also supports dual speakers (i.e., two users talking at the same time). Queuing is a feature that can be used to coordinate passing the floor around from one requesting user to another during a busy call. Ad-

[1] https://www.nsnam.org/

ditionally, priority and preemption make it possible for users with a higher priority to interrupt the current speaker and take the floor immediately. While the assignment of user priority is not defined in the standard, it would be possible for users with a higher level of operational authority to always have higher priority than users with less authority, and to temporarily assign higher priorities to any user depending on their status (e.g., a user that ends up in an emergency situation).

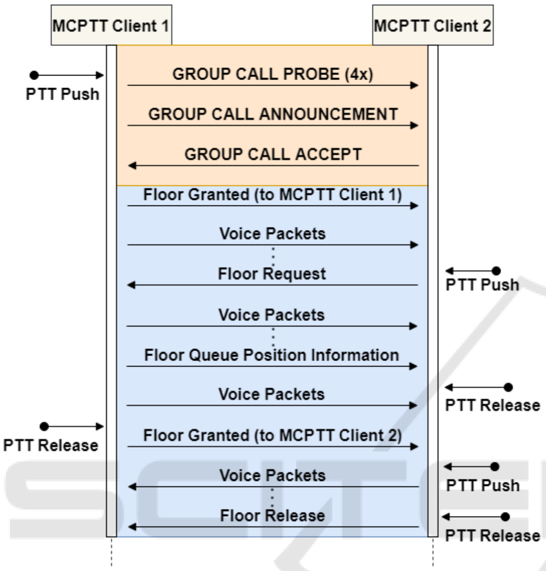


Figure 2: Off-network basic group call setup and use example.

Figure 2 illustrates some of the signaling that could take place during a basic, off-network, group call between two MCPTT users. The call control signaling is highlighted in orange while the floor control portion is highlighted in blue. In this example the call is initiated by a PTT push on Client 1 that triggers a probe to be sent four times, and if there is no response, the client concludes that there is no existing call for the group. After determining a call does not exist, Client 1 allocates floor control resources and sends out a call announcement to the other members in the group. Client 2 receives the announcement, accepts the call, and also allocates floor control resources. At this point, Client 1 grants itself the floor and notifies the user that they may begin speaking, at which point the client begins to send media packets that capture the user's voice to Client 2. During Client 1's transmission the user of Client 2 pushes the PTT button, which results in sending a floor request to Client 1. Since queuing is enabled in this example, Client 1 places the received request in the queue and responds with a queue information message to Client 2. Upon receiving the queue information message, Client 2 makes the user aware that the user must wait to speak. After the user of Client 1 is done speaking, the user releases the PTT button which results in Client 1 sending a granted message to Client 2, indicating that the user of Client 2 may begin speaking. Client 2 then makes the user aware, who then pushes the PTT button to accept the grant and begin speaking. Once the user of Client 2 is finished speaking, the user releases the PTT button and a floor release message is sent to Client 1 to indicate that the user of Client 2 is done speaking. At this point, the floor would be idle but could continue to be passed around between Client 1 and Client 2 until the call ends or is released by the users on the call. In addition, any of the features that were previously discussed, such as upgrading the call or additional users joining the call could also take place.

3 RELATED WORKS

In this section we will discuss some literature related to this model. First, this work extends our previously published work, including Sun et al. (2019) and Garey et al. (2020) that use earlier versions of the MCPTT model presented in this paper. Specifically, Sun et al. (2019) uses the ns-3 extension to create an analytical model that characterizes KPI 1 (access time) for off-network mode taking into account various factors that impact access time, including the scenario under which access is being requested, key ProSe and MCPTT configurations, and channel conditions. The work in Garey et al. (2020) is an extension of Sun et al. (2019) that also makes use of this model. In Garey et al. (2020), KPI 1, packet loss, delay, and data rate are used as performance indicators of the MCPTT application to study the impact of traffic, user density, range, and the underlying technology by considering both ProSe and Wi-Fi in the analysis. Our present work is distinguished by the inclusion of on-network MCPTT models, a novel orchestrated pusher model leveraging the translation of public safety call logs into empirical distributions driving the simulation, and a new example highlighting various modes of operation in a notional public safety scenario. Where our previous work in Gamboa et al. (2019) analyzed UE-to-Network Relay performance with an abstracted MCPTT model, we are able herein to run a highly complete simulation implementation of MCPTT

Outside of our own work, several others have published work related to MCPTT operations and performance. In Feng and Li (2019), a conflict-adaptive back-off solution is proposed and analyzed using a Markov chain to investigate the performance of an enhanced version of the floor control protocol that could increase the likelihood of a successful floor request when multiple requests are initiated simultaneously. Brady and Roy (2020) perform an analysis with data collected from LMR and LTE systems in addition to numerical computations to compare how a transition from LMR to LTE would reduce wait times for first responders that wish to speak. Choi et al. (2019) uses a testbed to verify that with LTE MCPTT KPI measurements meet their respective requirements. Kim et al. (2019) uses an MCPTT test bed to investigate how a state-based uplink-scheduler can be used to reduce the latency of MCPTT signaling, in order to improve the overall performance of an MCPTT service. Kim et al. (2018) shows how the use of dynamic schedulers can be used to prioritize physical resources in real time for MCPTT users to increase download and upload speeds when the LTE network is congested. Sanchoyerto et al. (2019) predicts that MCPTT services will perform much better with the use of 5G instead of 4G because of the estimated reduction in latency across the network. Höyhtyä et al. (2018) discusses how prioritization of physical resources and rapidly deployable networks can be utilized by mission critical services in 5G applications. Solozabal et al. (2018) proposes a Mobile Edge Computing (MEC) architecture specifically for the use of MCPTT over 5G that could reduce call setup times and observed KPI values. Atanasov et al. (2020) also discusses the use of MEC to support mission critical services but without the use of Internet Protocol Multimedia Subsystem (IMS) that can provide improvements in performance.

All of the related works mentioned above address key issues facing MCPTT and other mission critical service operations and performance. However, many of the analyses focus solely on a single aspect of either on-network or off-network mode operation. The model that we will present gives researchers an opportunity to perform more comprehensive studies of MCPTT using the ns-3 simulator to analyze KPIs, behavior, and other performance metrics with a high degree of control and repeatability. This model could also be used to verify, study, and combine the related literature mentioned in this paper, as well as to perform large scale simulations that would otherwise require a large degree of coordination and many resources to execute using a real system or testbed equipment, if it is even possible. Furthermore, since our model is published as an open source extension of ns-3, we present the user with an opportunity to expand on current and future contributions made by the ns-3 and public safety communication research communities.

4 MODEL

In this section we discuss the MCPTT model that is implemented in ns-3. To start, we describe the application model in Section 4.1, which is an abstraction of the MCPTT service defined by 3GPP in 3GPP (2017a) and 3GPP (2017b). Next, we describe the pusher model that simulates user PTT activity using call log data in Section 4.2. Then, in Section 4.3 we discuss what outputs can be collected from our model, and how we verified our model's behavior in Section 4.4. Finally, in Section 4.5 we discuss the model's limitations.

4.1 Application Model

As mentioned in Section 2, MCPTT is a mission critical voice communication service originally designed to operate over LTE, specifically, for public safety. Moreover, MCPTT is a relatively small part of a much larger system. This means that for the most realistic and comprehensive model of MCPTT, one must also model LTE and support ProSe. This made ns-3 a prime candidate for our MCPTT model, as it is an open source network simulator that contains models for a number of communication technologies including LTE and ProSe (Rouil et al., 2017). It also supports the development of application layer models, primarily for generating traffic, which is where MCPTT fits into the network stack.

We also looked into using several other tools such as Vienna[2] and SimuLTE[3]. However, we found that Vienna is primarily a system and link layer simulator based on Monte Carlo simulations that does not model applications at the packet level. While SimuLTE is capable of modeling applications at the packet level through the use of "modules", we determined from experimentation that ns-3 offered greater control of application behavior and traffic for our purposes. SimuLTE also focuses on network-controlled D2D which requires UEs to be in-coverage (Nardini et al., 2018), whereas the D2D model in ns-3 supports in-coverage, out-of-coverage, and partial coverage situations.

To extend ns-3 with an MCPTT model, we created two new ns-3 applications that follow the

[2] https://www.nt.tuwien.ac.at/research/mobile-communications/vccs/

[3] https://simulte.com

MCPTT service defined in 3GPP (2017a) and 3GPP (2017b) very faithfully. They are the `McpttPttApp`, which represents the client application, and the `McpttServerApp`, which represents the server application. These two ns-3 applications coordinate several components that are necessary to provide many of the features described in Section 2. The client-side application, whose architecture is depicted in Figure 3, is the entity that offers a user Application Programming Interface (API) for MCPTT. This includes functions for pushing/releasing the PTT button, initiating/releasing calls, upgrading/downgrading calls, and switching between multiple calls. It also contains some information, such as the MCPTT user ID that would normally be stored in the Mission Critical Services (MCS) Management Object (MO) from 3GPP (2018a). The server-side application is intended to be installed on a server node and primarily communicates with the client application using a combination of MCPTT and Session Initiation Protocol (SIP) defined messages.

Both applications maintain a set of `McpttCall` objects that represent MCPTT calls. Only one call can be "selected" at a time by the client application, and this is the call that is affected by functions from the application's API. Each call contains an `McpttCallMachine` object, an `McpttFloorMachine` object, and channels for floor control and media messages. The channels used for call control messages reside in and are maintained by the MCPTT applications. The `McpttCallMachine` and `McpttFloorMachine` classes serve as interfaces that are specialized by concrete classes to allow the user to configure a call taking into account the variety of call types and modes of operation. For example, the two state machines needed to perform an off-network, basic group call would be the state machines from Section 10.2.1 in 3GPP (2017a) and Section 7.2.3 in 3GPP (2017b), which are implemented by the `McpttCallMachineGrpBasic` and `McpttOffNetworkFloorParticipant` classes in our model. Other parameters and configurations, such as group affiliation and resource allocation, is handled by the `McpttCallMachine`, while priority and preemption is handled by the `McpttFloorMachine`.

In addition to the main MCPTT components described above, we also created several supplementary components to facilitate MCPTT operations. This includes the `McpttCounter`, `McpttTimer`, `McpttChannel`, and `McpttFloorQueue` classes. `McpttCounter` and `McpttTimer` objects are used to model counters and timers, while the `McpttFloorQueue` class is used as needed to store floor requests when queuing is enabled. The `McpttChannel` class models the logical channels necessary for communication between the applications.

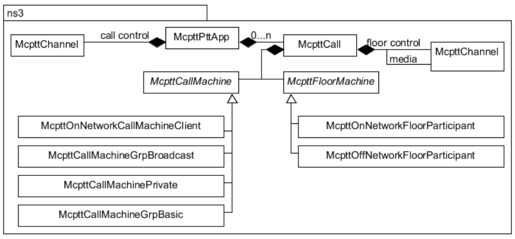


Figure 3: Client-side application architecture.

Along with those components we also model the call control and floor control messages necessary for signaling between various state machines, which also include a simplified SIP module to handle the SIP related procedures and messages when operating in on-network mode. For example, the class `McpttFloorMsgRequest` represents the "Floor Request" message from the floor control protocol, and the `McpttMediaMsg` represents the Real-time Transport Protocol (RTP) media messages used to transmit voice when a user is speaking.

The application also includes a configurable media source to model the transmission of RTP packets when a user has the floor. This is accomplished by the `McpttMediaSrc` that can be configured to generate a payload of size S at a data rate of D. With this component the user of this model can mimic the traffic generated by a Constant Bit Rate (CBR) encoder. For example, the media source could be configured with $S = 60$ B and $D = 24$ kbit/s to send packets at a size and interval that resembles the traffic generated by the Adaptive Multi-Rate Wideband (AMR-WB) codec at 23.85 kbit/s (ITU-T, 2003).

4.2 Pusher Model

In addition to the protocol elements described in Section 4.1, we also model user activity to automate the use of the MCPTT service. This is accomplished by the `McpttPusher` class that is attached to each instance of an `McpttPttApp` object. We refer to this class as a "pusher" since it generates "push" and "release" events over the course of a simulation to mimic a user pushing and then later releasing the PTT button on their device. The default and simplest form of our pusher model is when an instance of the `McpttPusher` class acts independently, using random variables. In this mode of operation, which we refer to as "automatic mode," the pusher uses two random variables: one for generating a time span before a push indication will occur, and another for generating a time span before a subsequent release indication will occur.

Even though automatic mode allows the user to configure PTT activity for an individual pusher, it can be cumbersome to quantify, configure, and characterize the overall channel activity for a group of MCPTT users. This is due to the fact that PTT events are generated independently of one another in automatic mode. This also means that unlike the users of most public safety organizations, independent pushers do not take into account politeness or situational behavior based on what the other pushers' actions are or the current state of the call. As a result, we defined another approach, called "orchestrated mode," that provides a more coordinated pusher model across the users in a call. The benefits of orchestrated mode include the automated selection of pushers to produce PTT events and the ability to configure parameters at the system level according to the desired rate of activity. In contrast, automatic mode would require the ns-3 user to configure pushers by scaling the values used for each random variable based on the number of pushers that will be participating in each call to achieve an overall activity rate.

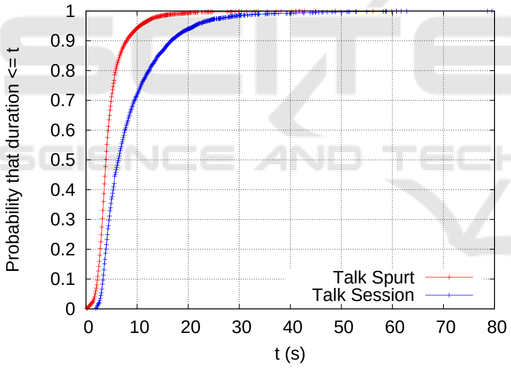


Figure 4: Talk spurt and talk session duration CDFs.

Orchestrated mode is achieved by using an instance of the `McpttPusherOrchestrator` class. This class is a centralized entity that controls a set of pushers by scheduling its own push and release events just like an individual pusher, but then randomly selects a pusher from the set it is orchestrating to carry out the actions. We also analyzed public safety call logs that were taken from an existing LMR system to further enhance the pusher model. During our analysis we determined that our model needed to capture two key elements to simulate the PTT behavior of such users: "talk spurts" and "talk sessions." We define the length of a talk spurt to be the time from when a PTT event begins until it ends, while the length of a talk session is simply the total length of consecutive talk spurts from one or more UEs in the same group. This resulted in the data sets plotted in Figure 4, for which a Cumulative Distribution Function (CDF) is used to capture the distribution of 8381 talk spurt lengths and 4606 talk session lengths. These data sets enable us to use orchestrated mode in combination with empirical random variables to resemble the overall PTT activity of a real public safety group call. The first enhancement comes from the `McpttPusherOrchestratorSpurtCdf` class, which creates and configures an underlying `McpttPusherOrchestrator` instance specifically to schedule talk spurt lengths based the talk spurt length CDF from Figure 4. The second enhancement comes from the `McpttPusherOrchestratorSessionCdf` class that can be attached to an `McpttPusherOrchestratorSpurtCdf` instance to enforce talk sessions based on the talk session length CDF in Figure 4.

While we were able to gather similar information about the interarrival times of talk spurts and talk sessions, we do not know if these statistics are highly dependent on the number of devices in a group call because while the call logs provide valuable information about active users, we do not know the system configuration, the total number of groups, or the size of those groups. Thus, we could not generalize interarrival times with respect to different group sizes. Therefore, we leave it up to the user to control interarrival times using an "activity factor." With this approach, we assume that the interarrival times of talk sessions and talk spurts follow an exponential distribution, which is scaled by the activity factor set by the user. This means that the user can specify individual values between (0, 1] for both parameters to determine the overall PTT activity during a simulation, with 0 representing no activity and 1 representing non-stop activity. The equation used to determine the mean interarrival time of talk spurts ($\overline{x_{IAT}}$) is:

$$\overline{x_{IAT}} = \overline{x_T} * (\frac{1}{VAF} - 1) \quad (1)$$

where $\overline{x_T}$ is the average duration of a talk spurt, which we determined to be 4.69 s in our logs, and VAF is the activity factor set by the user. The equation used to determine the mean interarrival time of talk sessions ($\overline{y_{IAT}}$) is:

$$\overline{y_{IAT}} = \overline{y_T} * (\frac{1}{SAF} - 1) \quad (2)$$

where $\overline{y_T}$ is the average duration of a talk session, which we determined to be 8.58 s in our logs, and SAF is the activity factor set by the user.

For example, Figure 5 shows the resulting timeline of events if $VAF = 1$, $SAF = 0.25$, we have two

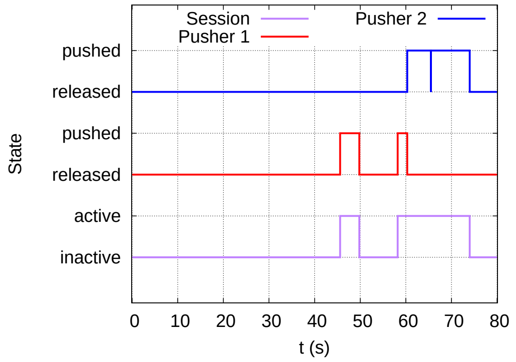


Figure 5: Orchestrated pusher model example.

pushers, and the total simulation time is 80 *s*. In this graph, the simulation time is on the x-axis while the states of sessions and pushers are on the y-axis. This indicates that there is an active talk session for 25 % of the total simulation time, and that within each active talk session a PTT button is pushed 100 % of the time. Note that, even though both *VAF* and *SAF* are configurable, setting $VAF = 1$ best matches the activity from the call logs that we analyzed. This is due to the fact that PTT events recorded in these logs and, consequently, captured in all of the statistics that we collected, include "hang time". Hang time is an LMR system parameter that, as captured in the call logs, is additional time added to the end of a user's transmission so that if a consecutive transmission occurs for that same user before a given amount of time has passed, it is recorded as one event. This means that it is likely the case that there were more occurrences of PTT events than what is recorded in the logs, and the actual lengths of those PTT events were shorter. For example, if the system hang time is 3 s, a user pushes the PTT button for 2 s, releases the PTT button, and then 1 s later pushes the PTT button again for another 2 s before releasing it, this is recorded as one PTT event with a duration of $2+1+2+3 = 8$ s.

Our third enhancement to orchestrator mode comes from the `McpttPusherOrchestratorContention` class that accounts for overlapping PTT events, which are necessary for queuing and preemption to occur. An instance of this class can be attached to the `McpttPusherOrchestrator` to simulate colliding PTT events. This is accomplished by generating a value between [0, 1) with a uniform random variable and comparing it to a threshold set by the user whenever a PTT event scheduled by the underlying orchestrator occurs. We refer to this threshold as the Contention Probability (CP), and if the random value generated is less than CP then an additional PTT event will occur at anytime during the original event. For example, if there were originally 100 PTT events in the previous example, we then included an `McpttPusherOrchestratorContention` instance, and set $CP = 0.1$ (10 %), then approximately 10 out of the 100 PTT events generated by the original orchestrator would trigger an additional overlapping PTT event for a total of 110 PTT events. This means that, in total, approximately 9 % of all PTT events would be generated by the newly included orchestrator. In addition to the overall probability, to calculate the probability that any individual pusher's PTT event will overlap with another pusher's PTT event (PC_I), Equation 3 can be used.

$$PC_I = \frac{2*CP}{1+CP} \tag{3}$$

4.3 Outputs

The current implementation of our model contains several traces that can be used to derive information about various aspects of the MCPTT application. The message and state machine traces can be parsed post-simulation to study behavior and performance.

Table 1: Message trace snippet.

time (s)	nodeid	rx/tx	bytes	message
30.54	1	TX	58	McpttFloorMsgGranted
30.59	2	RX	58	McpttFloorMsgGranted

The message trace captures all MCPTT application message exchanges between state machines during a simulation. This includes all call control, floor control, and media messages, each of which can be filtered at the user's discretion. Table 1 shows the column names and two sample rows of data that can be included in this trace. Both traces include general information such as the ID of the node that created the entry in the trace and the time at which that record was created. Columns that are included specific to the message trace are those that indicate whether or not a message was sent or received, the size of the message, and the name of the message. For example, the first row of data in Table 1 indicates that 30.54 s into the simulation, node 1 sent a "Floor Granted" message that was 58 B in size. With this trace, information such as packet loss, delay, jitter, and data rate can be determined with varying granularity (e.g., per call, per application, etc.).

The data included in the state machine trace captures internal state information for various MCPTT state machines. Therefore, on top of the general fields, it also includes fields such as the name of the state machine that the record is for, the previous state of the state machine, and the new state of the state machine. With this trace, information such as the status of a call and the state of an application can be extracted. The user could also use this trace to detect operations and behavior that does not result in or is not the result of any message exchanges, such as inactivity or termination events.

In addition to the basic traces mentioned above, we also have traces to capture KPIs that measure the performance of MCPTT as defined by 3GPP (2019). The KPIs that our model is currently capable of reporting include access time (KPI 1) and mouth-to-ear latency (KPI 3).

To capture KPI 1, which measures the delay between a request to speak and the corresponding grant, our model uses the state transitions of the floor control state machine. The initial request to speak is indicated by a change to the 'O: Pending Request' state, and the grant is indicated by a subsequent transition to the 'O: Has Permission' state. The trace includes a result field to capture the outcome of a PTT request as there are several that need to be considered. The delay and outcomes of KPI 1 can be traced to an output file as depicted in Table 2.

Table 2: KPI 1 trace snippet.

time (s)	userid	callid	result	latency (s)
7.246	1	0	I	0.024
25.907	3	1	Q	2.653
28.952	4	1	I	0.024
34.017	3	1	D	0.651

The five possible outcomes that may result from a PTT request can be filtered based on the fourth column in Table 2, with "I" denoting "immediate", "Q" denoting "queued", "D" denoting "denied", "F" denoting "failed", and "A" denoting "abandoned". The meaning of each outcome value can be found in Table 3. Table 2 illustrates that 7.246 s into the simulation, the user with MCPTT User ID 1, was granted the floor immediately upon request and it took 24 ms. When measuring access time using this trace only "I" and "Q" outcomes align with the definition of KPI 1, and it is important to note that "Q" outcomes will be heavily dependent on pusher behavior. It is also worth mentioning that 3GPP (2019) specifies that KPI 1 should be less than 300 ms for 99 % of all on-network MCPTT requests when there is negligible backhaul delay and less than 70 % load per node. There is no similar requirement defined for off-network mode as of this writing.

To capture KPI 3 in our model, the RTP packets of a talk spurt include a timestamp to mark when they were generated. When those RTP packets are received by a receiving MCPTT application it will check whether the packet contains a newer 'start-of-talkspurt' timestamp, which indicates that a new talk spurt exists. The latency of each new talk spurt is then traced by each receiving application as depicted in Table 4. From the row in Table 4 we can see that 3.727 s into the simulation, node 8 detected a new talk spurt from Synchronization Source (SSRC) 3 that had a latency (i.e., KPI 3) of 24 ms.

Table 3: KPI 1 trace outcome definitions.

Result	Definition
I	Granted immediately.
Q	Queued and subsequently granted.
D	Denied immediately.
F	Timeout or unexpected sequence.
A	PTT button was released.

Table 4: KPI 3 trace snippet.

time (s)	ssrc	nodeid	callid	latency (s)
3.727	3	8	1	0.024

4.4 Verification

MCPTT off-network models were subject to an extensive verification, previously published in Varin et al. (2018), against 3GPP (2017a) and 3GPP (2017b) to verify the model's behavior. We have verified and validated our on-network models in several ways. First, 3GPP defined conformance tests for selected on-network mode configurations in 3GPP (2020), and we defined ns-3 unit tests to align with the testing steps in that document. Second, Nemergent Solutions, a vendor based in Spain that develops Mission Critical solutions, has published detailed packet traces for MCPTT version 13.3 on-demand, on-network, pre-arranged group call with automatic commencement[4], and we aligned our basic tests with this trace. Finally, the Public Safety Communication Research (PSCR) division at the National Institute of Standards & Technology (NIST) conducted small-scale measurements of MCPTT on-network operations using a testbed and shared packet traces with us, with which we confirmed close alignment.

4.5 Limitations

While we intend for our model to capture the behavior of an actual MCPTT system, it is not an actual system with real users. This comes with limitations that have the potential to affect studies performed with our model and may lead to unrealistic observations.

[4] https://nemergent.com/traces.html

The limitations that we will discuss in the remainder of this section include any components, features, and functionalities that we know to be lacking or absent in our current ns-3 implementation.

We mentioned in Section 4.1 that MCPTT is a part of a much larger system, and we must therefore also consider the limitations of the models that surround our MCPTT model. This includes limitations in ns-3's LTE model since it is the network technology that MCPTT was initially designed to operate over. The ProSe model does not support ProSe Per Packet Priority (PPPP), affecting the priority of physical resources used by an off-network MCPTT application when the call is in an elevated status. A similar limitation exists for on-network operation, since the LTE schedulers in ns-3 are not designed to prioritize on a bearer basis or within a bearer based on call status as specified by 3GPP (2017a). This means that any studies pertaining to call priority and resource allocation based on call status would require further development of the underlying ns-3 LTE model. Also, the UE-to-Network Relay model lacks coordination between the uplink (UL) and sidelink (SL) resource scheduling, which impacts the performance of any application running over relay (Gamboa et al., 2019). The ns-3 Evolved Packet Core (EPC) model in LTE is also simplistic, with idealized representations of the MME and no modeling of delays that may come from authorization, access control, roaming, server processing, etc. These gaps in modeling are not fundamental limitations but arise from the current state of the models. MCPTT-based simulation studies would benefit from future improvements to the ns-3 cellular modeling fidelity because prioritization and non-ideal control channels have the potential to affect measured KPIs and general statistics such as packet delay.

In Section 4.2 we already mentioned several factors from the call logs that impact our pusher model, but we did not mention additional studies that could be performed to enhance this model. For example, our model does not take into account user impatience and the effect of network congestion that leads to abandoned transactions and more user impatience, such as is done in Baynat et al. (2015). Such user behavior could also be directly studied with regard to public safety users in cases where PTT requests do not go through, and possibly influenced by different operational contexts, such as a natural disaster or a routine traffic stop. With that said, our model could be expanded to take into account environmental and/or situational details to realize more realistic PTT activities that could ultimately lead to more realistic traffic patterns.

5 CASE STUDY

At the annual Public Safety Broadband Stakeholder meeting, we demonstrated the use of this model in a large-scale, public safety, scenario with a notional duration of four hours and involving over one hundred nodes in various operational roles[5]. However, we focus herein on a smaller scale scenario because it more clearly highlights the key modes of operation (on-network via basic LTE or UE-to-Network Relay, and off-network via ProSe) that are currently supported. This scenario will be included in the next release of our public safety extensions for ns-3, with the basic goal of comparing performance metrics for three public safety network configurations.

This scenario consists of three teams of MCPTT users, with each team participating in its own MCPTT group call. All team members of team one are located within coverage of an eNodeB so they are all connected to the network (on-network). The second team is only connected with each other via ProSe since they are inside a building whose walls prevent connection to an eNodeB on the outside that would allow this team to connect to the network (off-network). A third team operates in a hybrid mode; all team members are within the building, but one team member is situated at the door with connectivity to both the team inside and the eNodeB (relay). This team member runs a UE-to-Network Relay service (defined in 3GPP (2018b)), connecting oneself and team members on the inside with the network. Figure 6 illustrates the scenario topology. The simulation can be scaled to different team sizes (with a default of 4 users per team) and run with a configurable simulation duration.

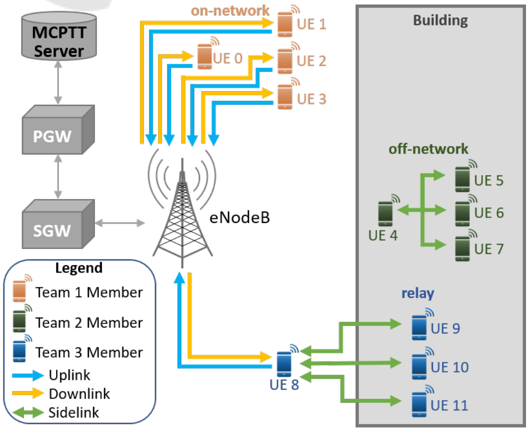
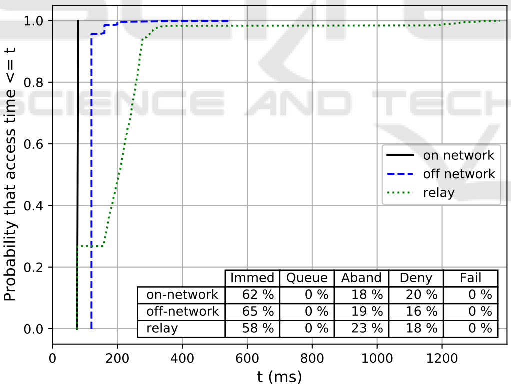


Figure 6: Example MCPTT scenario.

[5] https://www.nist.gov/ctl/pscr/simulation-and-visualization-public-safety-incidents

The ns-3 simulator has abstractions for the physical layer models, allowing insertion of path loss models that affect the packet error rates that are experienced in a simulation. This includes models that take building wall penetrations into account, one of which, the `HybridBuildingsPropagationLossModel`, we use in this example. The network technology is LTE, with a notional EPC network and a public safety server that houses the MCPTT server. In this example, the EPC round-trip delay for on-network and relay operation (from eNodeB to the MCPTT server, and back) is configured as a fixed 60 ms. The off-network SL period is configured to its lowest possible value of 40 ms.

We highlight performance here by post-processing the raw KPIs described in Section 4.3 to generate empirical CDFs on a per-team basis. This includes the access time performance (KPI 1) and the mouth-to-ear latency performance (KPI 3) that are observable once an MCPTT user has successfully obtained access to the floor. Each team is configured with automatic pushers governed by normal random variables with a mean IAT of 10 s, and a mean duration of 1 s (negative values are truncated to zero). Therefore, the overall pusher busy time is roughly $\frac{4}{11}$ or 36 %.



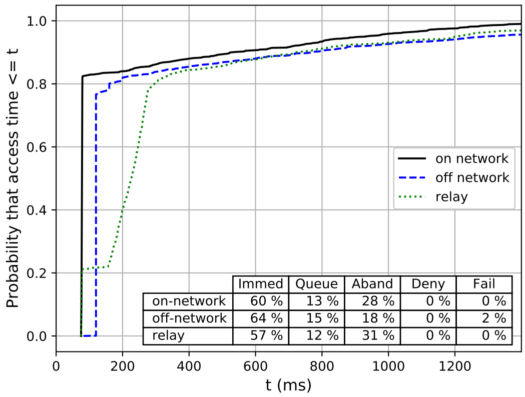
	Immed	Queue	Aband	Deny	Fail
on-network	62 %	0 %	18 %	20 %	0 %
off-network	65 %	0 %	19 %	16 %	0 %
relay	58 %	0 %	23 %	18 %	0 %

Figure 7: MCPTT Access Time for immediately granted requests with queuing disabled.

The first set of results presented is from a configuration that disables queuing of floor requests, so any request for the floor will be either granted immediately, denied, or fail or be abandoned if the floor request is not served in time. Figure 7 plots the empirical CDF of the access times observed for floor requests, for a simulation that runs until at least 1000 successful network accesses are observed for each team. Only values for immediately served or successfully queued requests contribute to the CDF of access times. Also tabulated, for each team, are the percentages observed for each type of outcome.

On-network access time is expected to provide the lowest minimum latency, because the server is centralized and can determine access immediately. In this first case without queuing enabled, on-network access time represents the sum of the core network delay (60 ms, as stated above), and the Radio Access Network (RAN) delay, which ranges from 15 ms to 19 ms in this configuration, leading to a nearly vertical line in the plot at around 75 ms. For the off-network team, access time should be around 120 ms most of the time, when the floor is idle, because by default the distributed protocol uses 3 SL periods to be confident that the floor is available before taking it. However, it could take longer because the floor control protocol also takes into account when multiple users are requesting the floor at the same time, at which point, additional SL periods may be required. In the illustrated case, it takes more time in off-network mode for floor control coordination than the round-trip time it takes for the request/response exchange with the on-network server. The UE-to-Network Relay team's performance in this case exceeds that of the on-network case, due to the extra relay hop introducing additional delay and loss in about 75 % of the requests. The lowest 25 % of values in the relay case correspond to floor requests initiated by the relay team member itself, which do not require any SL transmissions. As explained in more detail in Gamboa et al. (2019), message loss can happen due to scheduling decisions at the relay (prioritizing UL over SL transmissions, serving the attached nodes using round robin, and SL period transmission cutoff) and due to the half-duplex operation of SL transmission, so message loss may be significantly higher as compared to the on-network case. The CDF points that appear as extreme outliers of latency (greater than 1 s) are due to floor request retransmissions, the timer for which has a timeout value of 1 second.

Figure 8 shows the access time CDF from a similarly configured simulation, with the only difference being that queuing of floor requests is enabled for both on-network and off-network. In general, queuing increases the probability that a request will be served, at the expense of additional access time delay. If the floor request is queued, then the time to wait for the current talker, or those earlier in the queue, will be added to the latency. The plot shows that roughly 80 % of the values are immediately served, and the remaining 20 % of the values are queued (abandoned, denied, and failed requests are not counted in the CDF). The CDF slope above 80 % is gradual and reaches maximum values of between 2 s and 4 s for the three teams; these latencies are dependent on waiting for one or more talk spurts to end.



	Immed	Queue	Aband	Deny	Fail
on-network	60 %	13 %	28 %	0 %	0 %
off-network	64 %	15 %	18 %	0 %	2 %
relay	57 %	12 %	31 %	0 %	0 %

Figure 8: MCPTT Access Time for queued or immediately granted requests with queuing enabled.

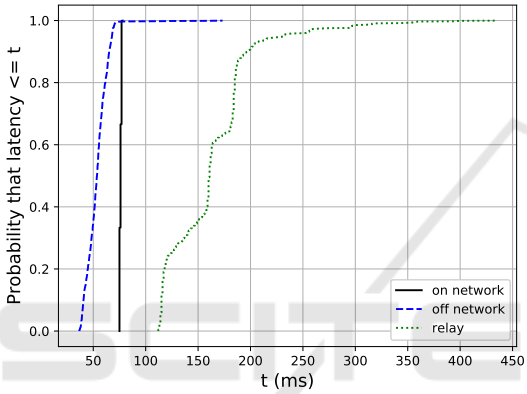


Figure 9: MCPTT Mouth-to-Ear Latency.

For this scenario, the plot shown in Figure 9 illustrates an empirical CDF of the mouth-to-ear latency of the beginning of each talk spurt in the same simulation scenario with queuing disabled. Unlike the access time statistics, the mouth-to-ear latency statistics are unaffected by the queuing configuration because the latency samples are taken after the floor has been granted. This CDF illustrates that with off-network operation, the RTP packets can flow directly between devices and do not have to go to the eNodeB, core network, server, and then back. On-network takes longer, as expected, because of the round trip that RTP packets incur (to the MCPTT server and back), while the relay experiences even more latency, as expected, due to the additional SL hop that introduces higher delays and loss. When queuing is disabled, the on-network mouth-to-ear latency and the access time latency are roughly the same (one RTT), as shown. One difference between Figures 7 and 9 for relay communications is that we no longer observe latency greater than 1 s for the data traffic because there are no lost floor request messages that need to be re-transmitted.

6 CONCLUSION

In this paper we presented an MCPTT simulation model that extends the network simulator, ns-3. Based on our earlier work to develop standards-aligned call control and floor control simulation models for MCPTT off-network operation, we describe herein the extension to on-network operation, the enhancement of our PTT traffic generation model (based on distilling PTT call traces to parameterize stochastic talk events in ns-3), and a public safety example scenario. We also summarize the basic operation of MCPTT in practice, the architecture of the ns-3 MCPTT model and limitations thereof, the output data available for analysis, and how we verified the model using test cases and published traces from an on-network MCPTT testbed. Future work will include the development of new and existing components, features, and functionalities discussed in Section 4.5 that will enhance the accuracy and behavior of our current MCPTT model and the models that surround MCPTT for a more realistic representation of the service. Future work will also include the use of this model to further analyze and develop public safety scenarios to study the impact of system configurations, traffic loads, etc. on MCPTT performance.

REFERENCES

3GPP (2017a). Mission Critical Push To Talk (MCPTT) call control; Protocol specification. Technical Specification 24.379, 3rd Generation Partnership Project (3GPP). Version 14.4.0.

3GPP (2017b). Mission Critical Push To Talk (MCPTT) media plane control; Protocol specification. Technical Specification 24.380, 3rd Generation Partnership Project (3GPP). Version 14.4.0.

3GPP (2018a). Mission Critical Services (MCS) Management Object (MO). Technical Specification 24.483, 3rd Generation Partnership Project (3GPP). Version 14.4.0.

3GPP (2018b). Proximity-based services (ProSe); Stage 2. Technical Specification 23.303, 3rd Generation Partnership Project (3GPP). Version 15.1.0.

3GPP (2019). Mission Critical Push To Talk (MCPTT); Stage 1. Technical Specification 22.179, 3rd Generation Partnership Project (3GPP). Version 16.5.0.

3GPP (2020). Mission Critical (MC) services over LTE; Part 2: Mission Critical Push To Talk (MCPTT) User Equipment (UE) Protocol conformance specification. Technical Specification 36.579-2, 3rd Generation Partnership Project (3GPP). Version 14.6.0.

Atanasov, I., Pencheva, E., and Nametkov, A. (2020). Handling Mission Critical Calls at the Network Edge. In *2020 International Conference on Mathematics and*

Computers in Science and Engineering (MACISE), pages 6–9.

Baynat, B., Vasseur, M., and Abreu, T. (2015). Revisiting the Characterization and the Modeling of User Impatience in Ubiquitous Networks. *PE-WASUN '15: Proceedings of the 12th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor*, pages 85–91.

Brady, C. and Roy, S. (2020). Analysis of Mission Critical Push-to-Talk (MCPTT) Services Over Public Safety Networks. *IEEE Wireless Communications Letters*, 9(9):1462–1466.

Choi, S. W., Song, Y., Shin, W., and Kim, J. (2019). A Feasibility Study on Mission-Critical Push-to-Talk: Standards and Implementation Perspectives. *IEEE Communications Magazine*, 57(2):81–87.

Feng, S. and Li, H. (2019). Floor Control Conflict Resolution in Off-network Mode of MCPTT. In *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 509–513.

Gamboa, S., Thanigaivel, R., and Rouil, R. (2019). System Level Evaluation of UE-to-Network Relays in D2D-Enabled LTE Networks. In *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–7.

Garey, W., Sun, Y., and Rouil, R. (2020). Performance Evaluation of Proximity Services and Wi-Fi for Public Safety Mission Critical Voice Application. *Wireless Communications and Mobile Computing*, 2020(8198767).

Höyhtyä, M., Lähetkangas, K., Suomalainen, J., Hoppari, M., Kujanpää, K., Trung Ngo, K., Kippola, T., Heikkilä, M., Posti, H., Mäki, J., Savunen, T., Hulkkonen, A., and Kokkinen, H. (2018). Critical Communications Over Mobile Operators' Networks: 5G Use Cases Enabled by Licensed Spectrum Sharing, Network Slicing and QoS Control. *IEEE Access*, 6:73572–73582.

ITU-T (2003). Wideband coding of speech at around 16 kbit/s using Adaptive Multi-Rate Wideband (AMR-WB). Technical Specification G.722.2, TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU (ITU-T). Version 16.5.0.

Kim, H., Jo, J., Park, C., Ahn, S., Chin, H., Park, P., and Kim, Y. (2018). Dynamic Resource Scheduling Algorithm for Public Safety Network. In *2018 UKSim-AMSS 20th International Conference on Computer Modelling and Simulation (UKSim)*, pages 127–132.

Kim, J., Jo, O., and Choi, S. W. (2019). State-Based Uplink-Scheduling Scheme for Reducing Control Plane Latency of MCPTT Services. *IEEE Systems Journal*, 13(3):2547–2550.

Nardini, G., Virdis, A., and Stea, G. (2018). Modeling Network-Controlled Device-to-Device Communications in SimuLTE. *Sensors (Basel, Switzerland)*, 18(10):3551.

Rouil, R., Cintrón, F., Ben Mosbah, A., and Gamboa, S. (2017). Implementation and Validation of an LTE D2D Model for Ns-3. In *Proceedings of the Workshop on Ns-3*, WNS3 '17, page 55–62, New York, NY, USA. Association for Computing Machinery.

Sanchoyerto, A., Solozabal, R., Blanco, B., and Liberal, F. (2019). Analysis of the Impact of the Evolution Toward 5G Architectures on Mission Critical Push-to-Talk Services. *IEEE Access*, 7:115052–115061.

Solozabal, R., Sanchoyerto, A., Atxutegi, E., Blanco, B., Fajardo, J. O., and Liberal, F. (2018). Exploitation of Mobile Edge Computing in 5G Distributed Mission-Critical Push-to-Talk Service Deployment. *IEEE Access*, 6:37665–37675.

Sun, Y., Garey, W., Rouil, R., and Varin, P. (2019). Access Time Analysis of MCPTT Off-Network Mode over LTE. *Wireless Communications and Mobile Computing*, 2019(2729370).

Varin, P., Sun, Y., and Garey, W. (2018). Test Scenarios for Mission Critical Push-To-Talk (MCPTT) Off-Network Mode Protocols Implementation. Technical Report 8236, NIST.