A Robust Push-to-Talk Service for Wireless Mesh Networks

Yair Amir, Raluca Musăloiu-Elefteri, Nilo Rivera
Department of Computer Science, Johns Hopkins University

Abstract—**Push-to-Talk (PTT) is a useful capability for rapidly deployable wireless mesh networks used by first responders. PTT allows several users to speak with each other while using a single, half-duplex, communication channel, such that only one user speaks at a time while all other users listen.**

This paper presents the architecture and protocol of a robust distributed PTT service for wireless mesh networks. The architecture supports any 802.11 client with SIP-based (Session Initiation Protocol) VoIP software and enables the participation of regular phones. Collectively, the mesh nodes provide the illusion of a single third party call controller, enabling clients to participate via any reachable mesh node. Each PTT group instantiates its own logical floor control manager that is highly available and resilient to mesh connectivity changes such as node crashes and recoveries and network partitions and merges. Experimental results on a fully deployed mesh network consisting of 14 mesh nodes and tens of emulated clients demonstrate the scalability and robustness of the system.

I. Introduction

Push-to-Talk (PTT) is a well known service in the law enforcement and public safety communities, where coordination and spectral efficiency are key for efficient communication. Some cell phone companies offer a similar service in the commercial world. However, core differences in motivation drive these two sectors. Cellular phone systems are designed for the busiest hour, as outages impact revenue, while public safety systems are designed for worst case scenarios, as outages impact lives.

Unfortunately, first responders cannot always rely on pre-existing ground communication infrastructure. For example, the White House report on hurricane Katrina [1] states that 1,477 cell towers were incapacitated, leaving millions unable to communicate. Wireless mesh networks have emerged as a viable technology that allows for rapid deployment of an instant infrastructure [2]. Mobile clients can roam throughout the area covered by the mesh and seamlessly handoff between access points while utilizing real-time applications such as VoIP [3], [4]. These attributes make wireless mesh networks an appealing technology for first responders. While centralized solutions for providing PTT service exist (e.g., POC [5](Push-To-Talk Over Cellular)), there are currently no solutions for a robust and efficient PTT service that can be applied in more dynamic environments.

A PTT system requires an arbitration mechanism (also known as *floor control*) which determines the order in which participants speak. All participants that wish to communicate with each other form a PTT group. As the name suggests, they request to talk by pressing a button. In contrast to peer-to-peer VoIP systems, data must be disseminated from the speaker to *all* the participants in a given PTT group.

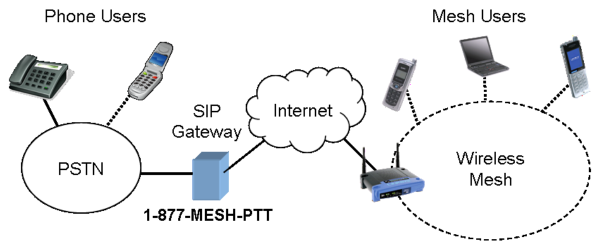


Fig. 1. Push-to-Talk system overview.

Building a robust and practical Push-to-Talk system for the wireless mesh environment is challenging for several reasons. First, it requires the ability to coordinate communication between users even when part of the infrastructure is unavailable (mesh node crashes) or when there is intermittent connectivity between nodes (network partitions and merges). This rules out traditional approaches such as POC, where arbitration is assured by a centralized point. Second, it must operate correctly when users join and leave the network, when they are partitioned away, lose their connectivity, or move from one access point to another. Third, it must use the wireless medium efficiently and should provide low transfer times between users' requests. Last but not least, an important property for first responders is the ability to integrate regular PSTN (Public Switched Telephone Network) and cellular phone users, allowing them to seamlessly participate in the PTT sessions conducted by the wireless mesh PTT service at a disaster site.

This paper presents the architecture and protocol of a robust distributed PTT service for wireless mesh networks. Collectively, the mesh nodes provide the illusion of a single third party call controller (3pcc), enabling clients to participate via any reachable mesh node. Each PTT group (also referred to as a PTT session) instantiates its own logical floor control manager that is responsible for keeping track of the floor requests of the participants and for issuing Permission-to-Speak when a participant releases the floor. Any of the mesh nodes in the network can play the controlling role for a session. To maintain high availability, each controller node is continuously monitored by every mesh node with a participating PTT client and is quickly replaced if it becomes unavailable due to a crash or network partition. The controller relinquishes its role to another mesh node upon determining that this node is better situated (network-wise) to control the PTT session, based on the current locations of the clients participating in the session. In addition to improved performance, this migration increases the availability of the service in the face of network partitions because it keeps the controller in the "center of gravity" of the clients in the PTT session.

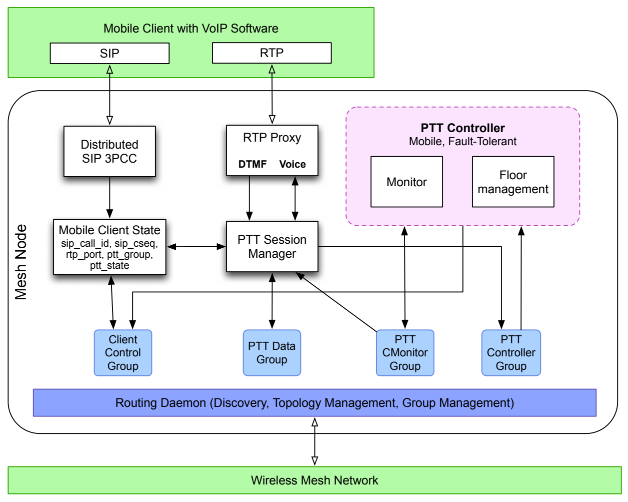


Fig. 2. System architecture.

The main contributions of this paper are:

- The first robust Push-to-Talk service for wireless mesh networks that can withstand connectivity changes such as node crashes, network partitions, and network merges.
- Novel use of multicast for localized access points coordination to share PTT client state, such that the entire network appear to the client as a single call controller.
- Novel decentralized floor control protocol that maintains a different logical controller for each PTT session and adaptively migrates it to the most suitable node in the network.
- An architecture that uses standard signaling for session control that allows regular PSTN phones users (e.g., cell phone users) and unmodified VoIP SIP phones to seamlessly participate in PTT sessions.

We implemented the proposed Push-to-Talk architecture and protocol within the SMesh open source wireless mesh system [6] and evaluated it in a 14-node testbed deployed across 3 buildings. In our tests, users experienced less than 150 ms interruption while the system switches between speakers. We show how the system scales to tens of clients, with an overhead of under 1 Kbps per client with 42 clients in the mesh. Then, we show that in our testbed, the system scales to 18 simultaneous PTT groups when dual-radio and packet aggregation are used. Lastly, an elaborate scenario with 40 clients divided among 10 different PTT sessions demonstrates that the system remains highly available during mesh network connectivity changes.

II. Background and Related Work

PTT allows half-duplex communication between multiple participants which request to speak by pressing a button. On a PTT group only one user is granted Permission-to-Speak at a time, while all the other users listen. DaSilva et al. [7] provide a good survey about PTT technologies. Floor control, an integral part of PTT, has been studied extensively over the years [8]–[10]. Some approaches to decentralized floor control are presented in [11]. A basic level of fault tolerance is built into some of these protocols to enable crash recovery.

PTT is commonly used by law enforcement and public safety communities to efficiently communicate between multiple users. Public safety agencies usually rely on trunked networks, known as Land Mobile Radio (LMR) systems, for voice and data communication [12]. The two major LMR systems are Project-25 [13], which is deployed over North America, and Terrestrial Trunked Radio (TETRA), which is deployed over Europe. Stringent guidelines for PTT, such as 500 ms one-way delay for voice packets to all listeners of a group, ensure that the system operates with acceptable performance.

Cell phone users also benefit from PTT type services that are now offered by telecommunication companies. A common standard, known as Push-to-Talk over Cellular (PoC) [5], allows PTT from different cellular network carriers to interoperate with one another. PoC uses VoIP protocols (SIP, RTP, etc) between clients and the PoC server. A floor control mechanism, referred to as Talk Burst Control Protocol, arbitrates communication in each group. The performance requirements of PoC are less demanding than those in LMR systems. For example, the standard specifies that end-to-end delay should typically be no more than 1.6 seconds and that the turnaround time from the time a user releases the floor until it hears another user speak should be no longer than 4 seconds. An initial evaluation on a GPRS cellular network is shown in [14].

Balachandran et al. show a unifying system for bridging LMR and commercial wireless access technologies [15]. Both LMR and commercial PTT solutions (PoC) rely on a central point of arbitration and send a separate unicast voice stream to each member of the PTT group. On these networks, the inefficiency inherent in using multiple unicast streams is not that costly over the wired backbone medium. Such a design would yield a multi-hop wireless mesh network useless with just a few users, and therefore is not a good fit in our case.

A decentralized approach with a full-mesh conferencing model is presented by Lennox and Schulzrinne in [16]. Florian Maurer [17] shows a decentralized scheme for PTT. Both approaches rely on all-to-all communication of control and voice packets between users. While adequate for small conferences or PTT sessions, this approach does not scale well and does not provide the robustness necessary to support node crashes and network partitions and merges, as presented in this paper.

Complementary to our work, some research has looked at optimizing routes for PTT data traffic in wireless mesh networks. Kado et al. [18] propose a centralized tree-based routing protocol that enables a root node to compute and arbitrate routes in the network. While we also optimize routes by instead using multicast dissemination trees from each mesh node to each PTT group in the system, our focus is on the fault tolerance and availability aspects for providing a highly robust PTT system.

III. Architecture

We consider a two-tier wireless mesh network with two classes of participants: mesh nodes and mesh clients. Mesh nodes communicate with each other, possibly using multiple hops, to effectively extend the coverage area of a single access point. Mesh clients, on the other hand, connect directly to mesh nodes, each of which serves as an access point.

The mesh topology changes when wireless connectivity between the mesh nodes changes, when mesh nodes crash or recover, or when additional mesh nodes are added to expand the wireless coverage. These changes may create network partitions and merges in the wireless mesh.

Mesh clients are unmodified 802.11 devices. We do not assume any specific drivers or hardware capabilities present on the clients. Clients connect to the mesh by associating with the wireless-mesh 802.11 SSID. A client participates with any compliant VoIP application.

Regular phones from the Public Switched Telephone Network (PSTN) such as home phones, and cell phones, connect to the mesh by dialing a regular phone number, in our case 1-877-MESH-PTT. The call is routed by the PSTN to a SIP gateway that is connected to the Internet (Figure 1). Normally, a regular VoIP client registers with the SIP gateway in order to receive incoming calls. In our architecture, the mesh Internet gateway registers as an end-client with the SIP gateway and routes messages between the mesh and the phones in the PSTN. We do not make any changes to SIP, therefore our protocol integrates with already deployed SIP gateways without any changes.[1]

Figure 2 illustrates the software architecture of our PTT system. It includes the interface with the mobile client, the mesh PTT session manager for the mobile client, and the mesh PTT controller for each PTT session in the wireless mesh network. Various multicast groups, over which communication takes place, are shown. An underlying routing daemon, Spines [19], manages the routes in the mesh and provides us with overlay group management to effectively communicate on a group-based abstraction. Multicast trees are calculated in a way similar to MOSPF [20]. Each of these components is described in detail in the next sections.

IV. Interface with Mobile Clients

Our architecture interacts with clients by using well established VoIP protocols. VoIP applications use the Session Initiation Protocol (SIP [21]), to establish, modify, and terminate a VoIP session. During the SIP session establishment, the Session Description Protocol (SDP [22]) is used to describe the content of the session (i.e., voice), the underlying transport protocol (i.e., RTP), the media format, and how to send the data to the client (address, port, etc). Data is then sent using the designated transport protocol between the parties.

A third party call control (3pcc) server is normally used to inter-connect multiple parties together through a rendezvous point. Conference call managers are one type of 3pcc. Good practices for SIP-based VoIP 3pcc servers are specified in RFC 3725 [23]. In essence, from an end-client point of view, the 3pcc server looks exactly the same as another end-client.

[1]For the SIP gateway we used a service provided by Vitelity (http://vitelity.com), which redirects the packets from the telephone network to our mesh gateway.

In our architecture, all mesh nodes act as a single, virtual 3pcc server and share the state of the SIP connection with every other mesh node in the vicinity of the client (between mesh nodes that can hear the client). This is key for the system to scale as it efficiently shares information only between nodes that potentially need the state of the SIP connection as the client moves throughout the mesh, or in case the client's mesh node crashes.

To participate in the mesh PTT session, the user specifies in his VoIP application the IP address of our virtual SIP server (i.e., "sip:ptt@192.168.1.10"). This IP is the same throughout the mesh. Every mesh node intercepts packets sent to this address and follows the SIP protocol to connect the client to the mesh. Therefore, the mesh network provides the illusion of a single 3pcc to the client.

Once a SIP connection is established, the user can start using the mesh PTT service by simply dialing the PTT group that it wishes to join. Each dialed key generates a Dial-Tone Multi-Frequency (DTMF [24]) signal that is sent over the RTP channel (by default, this signal is repeatedly sent over multiple RTP packets to ensure that the end-node receives it). In our approach, we intercept DTMF signals for control purposes between the end-client and the mesh. For example, a client dials "#12#" to join PTT group 12. In the same way, every time a user wishes to speak, pressing "5" or any pre-defined key combination will be interpreted as a "Request-To-Speak" control message. Once the system determines that it is the user's turn, it sends an audio signal ("beep-beep") to let the user know that he can start to speak. While other means for signaling control information are possible, DTMF is supported by most communication networks such as PSTN, allowing us to seamlessly support users from these networks.

RTP data is then sent from the client to the 3pcc virtual IP address through the client's access point (mesh node), which forwards the packets to every mesh node that has a PTT client on that group using a source-based multicast tree. Finally, each receiving mesh node forwards the packets to its corresponding end-clients.

V. Push-to-Talk Protocol

Providing a robust and scalable way to coordinate client communication is the essence of the Push-to-Talk protocol. There are several ways to approach it. One possibility is to have a unique point of management in the network that every mesh node needs to contact in order to register a request and get permission to speak. Such a protocol is easy to design and implement and is appropriate for deployment in some environments. However, this approach is not a good choice for networks that require high availability. For example, if a partition occurs in the mesh, all the clients connected to nodes that cannot reach the arbitration point will be left out of service. At the opposite extreme is the approach of total decentralization in which there is no unique entity that arbitrates the communication. Instead, the nodes in the mesh must coordinate and collectively decide on the order of serving the clients. While more complex, such a protocol

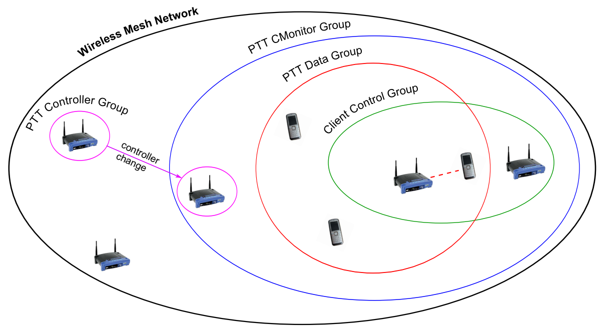


Fig. 3. Multicast groups for managing the client (*Client Control Group*), and for managing each PTT session (*PTT Controller Group*, *PTT Controller Monitoring Group*, *PTT Data Group*).

is very resilient to infrastructure failures, at the expense of a continuous communication overhead in order to maintain a consistent view between the mesh nodes in the network.

We chose a hybrid protocol that shares characteristics with both approaches. As in the centralized approach, each PTT session is managed by a controller node which is responsible for keeping track of floor requests and for issuing Permission-to-Speak after a participant releases the floor. However, each PTT session has is own controller node and any of the mesh nodes in the network can play the controlling role for any session. The controller node is continuously monitored by other nodes and rotated when a more suitable node (i.e., a node with a better geographical position in the network) becomes available. In addition, we separate floor control from data dissemination. While the arbitration is left to the best node to be the controller, the data is routed optimally to all participants through source-based multicast trees. This allows the system to be efficient and scalable.

A. Client management

For a PTT client, the entire mesh network behaves as a *single* 3pcc server. This is achieved by maintaining the state of the client on the mesh nodes in the vicinity of that client, such that any node that becomes the client's access point (the client is mobile) has the appropriate SIP and PTT information. A virtual IP is assigned to the 3pcc server, and is used by the client VoIP application to connect to.

Specifically, in order to service a client, the system requires information such as the SIP call identifier, SIP sequence number, RTP port, PTT group, PTT state (e.g., the client requests permission to speak, or has permission to speak). We maintain client's state in the vicinity of the client for several reasons. First, there is no single node responsible for the state. Instead, any node that can hear the client maintains a state for it. Thus, the state is preserved even when the client's access point crashes. Second, as the state is maintained in the vicinity of the client, the overhead is localized in the part of the network where the client is located. Finally, the client state is decoupled from the controller node, allowing the clients' requests to be recovered when the controller node crashes (or is partitioned away), as we discuss below.

Client Control Group. To share the client state between mesh nodes that can reach a client, we associate with each client an overlay multicast group. Specifically, any node that can hear the client (that is, not only its current access point) joins and periodically advertises the client state on the Client Control Group (Figure 3). In our experiments, we share this information every four seconds. Note that the system is not synchronized and different nodes may see different states for a client at a given time. We use a combination of client timestamps (available in the SIP and RTP packets) and controller logical timestamps to correctly identify the most recent state of a client.

B. PTT session management

A client joins a PTT session by initiating a VoIP conversation with the virtual 3pcc server as described in Section IV, independently of its network location. In our protocol a PTT session is coordinated by a controller node, whose presence is continuously monitored by other nodes. The controller relinquishes its role to another mesh node upon determining that this node is better situated (network-wise) to control the PTT session, based on the current location of the clients participating in the session. Three multicast groups are used to manage a PTT session in a distributed manner.

PTT Controller Group (PTT_CONTROLLER). For each PTT session, there is a single mesh node — the *controller* — responsible for managing the floor at a given moment in time. It receives and arbitrates requests and grants the right to speak. In our architecture, when a node becomes the controller for a PTT session, it joins an overlay multicast group associated with that session. Maintaining an overlay multicast group with the controller as the only member allows any mesh node in the network to reach the controller node without actually knowing its identity. All client floor requests are sent by their access points (mesh nodes) to this group and are stored by the controller in a FIFO queue.

PTT Controller Monitoring Group (PTT_CMONITOR). This overlay multicast group is used to monitor the controller node. A mesh node joins the monitoring group of a PTT session if it is the access point of a client that participates in that session. In addition, the controller joins this group to detect the presence of another controller during a network merge. A ping message is periodically sent by the controller to this group, allowing its members to monitor controller's presence and take action if the controller is no longer available.

PTT Data Group (PTT_DATA). This multicast group is used to deliver the actual voice data to the clients. A mesh node joins the PTT Data Group of a session if it is the access point of a client in that session. Thus, we completely separate floor arbitration — coordinated by a single controller node — from data dissemination. This allows us to optimally route data from the sender node to all the participants in a PTT session.

To simplify the management of names for these three multicast groups, we generate their IP multicast addresses using a hash function of the PTT session identifier, such that any mesh node in the network knows which groups are associated with each PTT session without coordination.

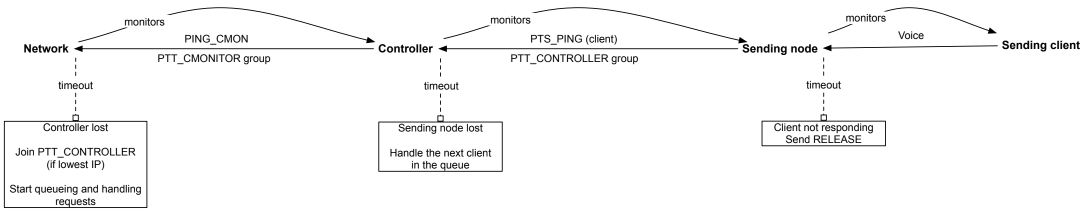


Fig. 4. Monitoring mechanisms employed to provide protocol robustness.

Similarly, the Client Control Group is generated as a hash of the client IP address.

C. Floor control

1) Requests handling: When a PTT client requests the floor, a REQUEST_FLOOR message is sent by its access point to the PTT_CONTROLLER group. The controller queues the request and sends back an acknowledgment. Release floor requests are sent to the controller in a similar manner. When a RELEASE_FLOOR is received, the controller node grants the right to speak to the next client in the queue by sending a PTS (Permission-to-Speak) message. This message is sent to the client using the Client Control group. If the client is no longer available, a simple timeout mechanism allows the controller to move to the next request in its queue.

2) Migrating the controller: While there is a single controller node for a PTT session at a given time, the system may change the controller over time, depending on participants' placement in the network. The idea is to avoid situations such as when a majority of the clients in a PTT session are localized in some part of the network while the controller node is in another. Placing the controller closer to where most participants are reduces the latency and the amount of control traffic in the network. In addition to improved performance, this migration increases the availability of the service in the face of network partitions because it keeps the controller in the "center of gravity" of the clients in the PTT session. Specifically, the system computes the cost that each node would incur if it was the controller as the sum of the costs to reach each member of PTT_DATA group. In our experiments we computed this cost every minute. By cost we refer to a wireless metric that may incorporate latency or the number of hops, for example[2]. Note that any node in the mesh network can be chosen to be a controller, even if it does not services PTT clients.

The sequence of steps performed for migrating the controller are as follows: First, the current controller enters a block state, in which it does not respond to any floor requests or releases and does not grant the right to speak to any client. Next, the controller sends an INVITE message to the selected node — the one with the lowest cost to be a controller — which includes the queue of the pending requests. Upon receiving such a message, the invited node joins the PTT_CMONITOR group — in case it was not already a member — and also joins the PTT_CONTROLLER group. It now has the queue of requests and can safely begin controlling the session, queuing new requests and issuing PTS. An acknowledgment is sent back to the initial controller so that it can leave the PTT_CONTROLLER group. In case of a timeout during this process, the original controller unblocks and continues to manage the PTT session.

D. Protocol robustness

A PTT session requires a controller and a sending node (that is, a node with a client with permission to speak). If one of these is missing, either there is nobody to arbitrate the floor or nobody is currently speaking as the system waits for a node which is no longer available. Thus, we introduce the following mechanisms to monitor the operation of each of these two nodes (Figure 4). Note that asymmetric links are eliminated by the routing protocol.

1) Controller node monitoring: The controller node periodically sends a keep-alive message (PING_CMON) to the PTT_CMONITOR group, allowing other nodes that service PTT clients for that session to monitor its presence. When the controller crashes or is partitioned away, the node with the lowest IP address on the PTT_CMONITOR group volunteers to be the controller by joining the PTT_CONTROLLER group. However, its queue of requests is empty. We use a special flag in the subsequent PING_CMON messages to notify everybody that a new controller was instantiated. All the nodes with pending PTT requests must re-send their requests as if they were new. Thus, the controller's queue is reconstructed in a best-effort way, with the requests from the current partition. Note, however, that the order of the requests in the new queue may be different than the one from the original controller. With minimal changes, the protocol can be adapted to recover part of the original order established by the previous controller.

Another situation from which we have to recover is when there are multiple controllers in the network. This occurs after a network merge but also when the controller is lost and multiple nodes decide to control the session (unlikely but possible, as the nodes can temporary have a different view of the network's topology). Since the controller node is the only one sending keep-alive messages on the PTT_CMONITOR group, receiving a keep-alive that is not its own indicates to the controller that there is at least one additional controller in the network. Once this situation is detected, the node with the lowest IP address remains the controller, while the other(s) must leave the controller's group. A redundant controller sends a LEAVE_REQUEST message to the PTT_CONTROLLER group with the content of its queue as it leaves the group. Upon

[2]Additional functionality from that provided by SMesh was added to retrieve topology and membership information from the link-state and group-state updates in Spines [19], which in turn allows a controller to compute the Euclidean distance from every node to a given PTT group.

Type	Sent by	Sent to	When
REQUEST_FLOOR	mesh node	PTT_CONTROLLER group	client requests floor
RELEASE_FLOOR	mesh node	PTT_CONTROLLER group	client releases floor
REQ_REL_ACK	controller	mesh node	
PTS	controller	CLIENT_CONTROL group	
PTS_ACK	node	controller	
INVITE	controller	mesh node	controller changes
INVITE_ACK	node	controller	
LEAVE_REQUEST	controller	controller	multiple controllers
LEAVE_REQUEST_ACK	controller	controller	multiple controllers
REVOKE	controller	node	multiple speakers
PING_CMON	controller	PTT_CMONITOR group	every second
PTS_PING	node	PTT_CONTROLLER group	every second when client has PTS

TABLE I
TYPES OF MESSAGES HANDLED BY THE CONTROLLER NODE.

receiving such a message, the controller with the lowest IP appends the queue to its own, removing duplicate requests if necessary, and acknowledges the leave.

2) Sending node monitoring: While the members of PTT_CMONITOR group monitor the controller, the controller in turn is responsible for monitoring the sending node (Figure 4). The sending node periodically issues a keep-alive message (PTS_PING - Permission-to-Speak Ping) on the PTT_CONTROLLER group. This allows the controller to quickly move to the next client in the queue in case of a timeout. An alternative to this approach would be to simply wait for the maximum allotted time per speaker to expire; however, system responsiveness is important in emergency situations, ruling this option out.

When two or more network partitions merge, there will be multiple controllers, but also multiple sending nodes in the network. The controller of the newly established network withdraws the right to speak to all additional clients by sending a REVOKE message to their access points (mesh nodes), which in turn notify their associated clients.

Table I summarizes the messages exchanged in our protocol.

VI. EXPERIMENTAL RESULTS

A. Setup

We implemented our protocol within the open source SMesh wireless mesh system [6] and evaluated it in a testbed of 14 Linksys WRT54G wireless routers deployed across several floors in three buildings. Other than our PTT system executables that implement the protocol described throughout this paper, no other changes were made to SMesh.

Each of the mesh nodes is equipped with one radio configured in ad-hoc mode. The data rate was set to 18 Mbps, the transmission power to 50 mW, and the 802.11 link-layer retransmission limit to 7. Unless specified, the topology of the mesh, depicted in Figure 5, was stable.

In all experiments, when a client is granted permission to speak it transmits a 64 Kbps VoIP stream as 160 bytes UDP packets every 20 ms.

Some experiments require a large number of simultaneous clients. To support such experiments, we implemented a client emulator that generated the appropriate control and data traffic associated with the emulated client. From the 802.11 network and from the PTT system perspective, there was no difference between an emulated client and a real client in terms of control and data traffic. Each client is instantiated on a mesh node,

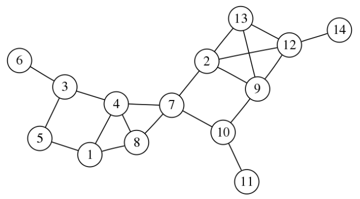


Fig. 5. Wireless mesh network testbed.

and packets between the emulated client and its access point are always transmitted over the wireless medium. In spite of generating the appropriate amount of traffic in the network, our metrics (such as latency and loss rate) are reported from the mesh nodes perspective. A real client should perceive slightly higher values. Despite this shortcoming, we resort to this method in order to evaluate our system in a real testbed, with a large number of users.

B. Measurements

We present four types of experiments. First, we demonstrate the PTT system's normal operation with a small number of clients. Second, we demonstrate the ability of the system to scale with the number of clients in a PTT group. Third, we demonstrate the ability of the system to scale with the number of PTT groups. Last, we demonstrate the robustness of the system through its ability to handle network partitions and merges correctly while PTT sessions are in progress.

1) Normal operation: This experiment involves four mobile clients, each of them connected to a different mesh node in the network (nodes 1, 2, 12, and 14 in Figure 5). All four clients join a PTT session and continuously request to talk. When a client is granted the floor, it immediately speaks for 20 seconds, releases the floor, and then renews its request. Thus, the PTT session's queue of requests is never empty.

Figure 6 depicts the VoIP data throughput and our protocol overhead, as seen by Node 1. The overhead includes the control traffic of the PTT protocol as well as the SMesh traffic associated with maintaining the mesh and the multicast groups. This overhead ranges between 1.5 Kbps and 5.8 Kbps, with an average of 3.4 Kbps, which is reasonable considering that each VoIP session is 64 Kbps. Detailed analysis shows that it takes a few tens of milliseconds from the moment the last packet from a client is received to the moment the first packet from the next client's voice stream arrives. This demonstrates that only a small part of the time is consumed for synchronizing the PTT clients.

2) Scaling with the number of clients in a PTT session: To test the scalability of our system, we gradually increase the number of clients participating in a single PTT session. Each client connects to one of the 14 mesh nodes in the network according to a round-robin order of their identifiers (i.e., the first client connects to Node 1, the second to Node 2, ..., the 14^{th} to Node 14, the 15^{th} to Node 1, etc.) and requests to speak. Upon acquiring the floor, each client speaks for 10 seconds, releases the floor, waits for another 10 seconds, and requests to speak again. Therefore, at any point, some client is authorized to speak.

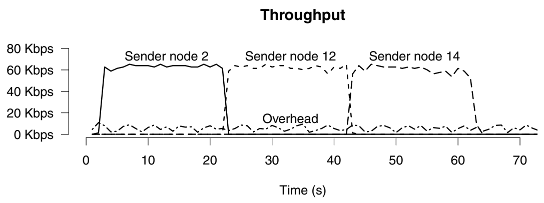


Fig. 6. Normal operation of the system, running in the 14-nodes testbed, with 4 clients on one PTT group.

We consider three scenarios: **(1)** Each mesh node is equipped with a single radio. **(2)** Each mesh node is equipped with dual radios such that the mesh traffic does not interfere with the mesh-to-client traffic, as the radios can be set on non-interfering channels. We emulated this dual-radio scenario in our single-radio environment by generating the client's messages locally on the corresponding mesh nodes and by avoiding sending data packets from the mesh nodes towards the clients. **(3)** Each mesh node is equipped with a single radio and the mobile clients have no PTT support[3].

For the case where clients have no PTT support, there are two main disadvantages that considerably affect performance. First, such clients continuously send VoIP packets, even when not having the floor. These packets are dropped by the mesh node serving the client, except for the durations when the client has acquired the floor. This case incurs considerable overhead as clients send unnecessary packets in their vicinity. The second disadvantage is that a node needs to send individual packets to all the clients directly connected to it, even if they are on the same PTT session. In contrast, with PTT support clients can use the same multicast address and local port, allowing a single stream of multicast packets to be sent by the mesh node to all of them.

Figure 7 and 8 present the latency and loss rate of the VoIP packets received by the mesh nodes, in each of the above three scenarios, averaged over 10-minute tests. With 42 clients (three clients connected to each mesh node), the average latency of the received packets was 28.42 ms for a single radio and 25.52 ms for dual radio. For the third scenario the system can scale only up to 28 clients (114 ms latency) before the loss rate goes above 0.5%. The experiment shows how the system scales and demonstrates that when utilizing a PTT enabled phone or a dual-radio configuration, the system can scale to at least 42 clients in the mesh network with minimal impact on latency and loss rate.

Figure 9 presents the overhead traffic, as seen by a single mesh node (Node 1 in Figure 5). This overhead depends on the distribution and the density of the clients in the network. For better analysis, we separate the overhead into three distinct components: **(1)** mesh control traffic (i.e., link state updates generated by topology changes and control traffic for managing multicast groups). The amount of this traffic is very small, less than 1 Kbps. **(2)** Control traffic generated by our PTT protocol (e.g., requests, releases, ping messages, acknowledgments, etc.) Since the size of these messages is

[3]In our system, one can simply participate in a PTT session using a standard VoIP phone or application.

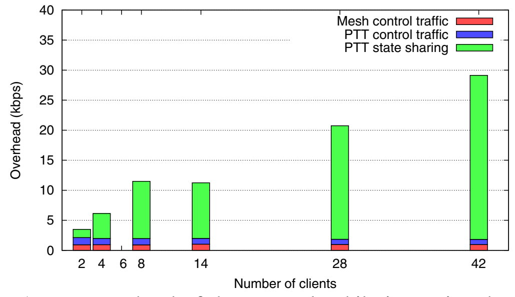


Fig. 9. Average overhead of the protocol, while increasing the number of clients on a single PTT group.

very small, this overhead is also low, less than 1 Kbps on average in our experiments. **(3)** Traffic required to locally share clients' PTT state (i.e., traffic on the Client Control groups). This represents the majority of the overhead, increasing from 1.3 Kbps for 2 clients to 27 Kbps for 42. This overhead depends on the density of the mesh (how many mesh nodes can hear a client) and the number of clients. The experiment shows that the overhead of the system as the number of clients grows is minimal, below 1 Kbps per client.

To test the scalability of our system in another dimension, we gradually increased the number of simultaneous PTT sessions in the system. Each PTT session includes four clients connected to random mesh nodes in the network. Each PTT session contributes a single VoIP stream (50 packets per second, total of 64 Kbps).

Figures 10 and 11 show the latency and loss rate as the number of PTT groups increases. With a single radio, the system scales to 6 PTT groups, while with dual-radio, the system scales to 8 PTT groups. Noting that the scalability of the system was impaired by the high overhead associated with sending small packets in 802.11 networks, we tested the same two scenarios with packing 160 ms of VoIP packets into one network packet at the mesh node. This approach allows us to trade some latency (20 ms x 7 packets = 140 ms) for an 8 fold reduction in the number of packets in the mesh. Note that PTT systems used by first-responders [13] employ a slightly higher packing scheme of 180 ms. Packet aggregation allowed us to support up to 18 PTT sessions before the latency jumped to over 500 ms. The experiment shows that it pays to trade some latency with scalability.

3) Robustness test: This experiment demonstrates the system's behavior when there is a partition and a merge in the wireless mesh network.

We first present a small-scale scenario with 4 clients (A, B, C and D) joining the network in 4 different places (Node 1, Node 5, Node 9 and Node 10), with A and B in one "side" of network, and C and D in the other side, as illustrated by Figure 12. In the beginning, the controller of the PTT session is Node 1 and client A is granted permission to speak. We then partitioned the network, such that Node 9 and Node 10 became unreachable from Node 1 and Node 5's side of the network. Figure 12 shows the voice traffic as received by client B in the first partition and by client D in the second one. We can see that in the first partition the data packets are generated by client A, and this does not change, as expected, even after

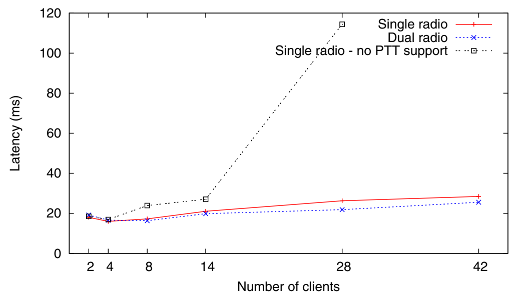


Fig. 7. Average latency of the packets received by the mesh nodes, while increasing the number of clients on a single PTT group.

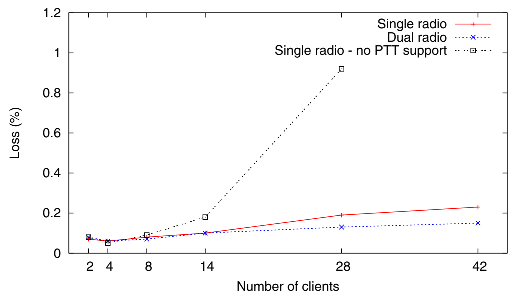


Fig. 8. Average loss rate of the packets received by the mesh nodes, while increasing the number of clients on a single PTT group.

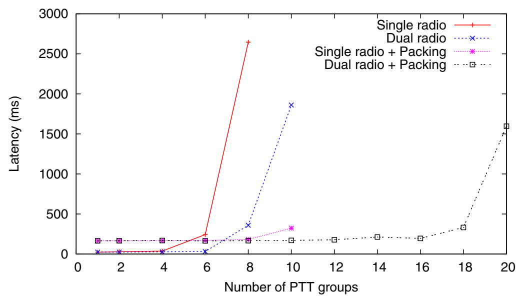


Fig. 10. Average latency of the packets received by the mesh nodes, while increasing the number of PTT groups (4 clients per group).

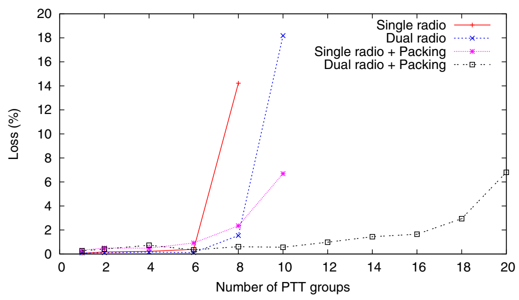


Fig. 11. Average loss rate of the packets received by the mesh nodes, while increasing the number of PTT groups (4 clients per group).

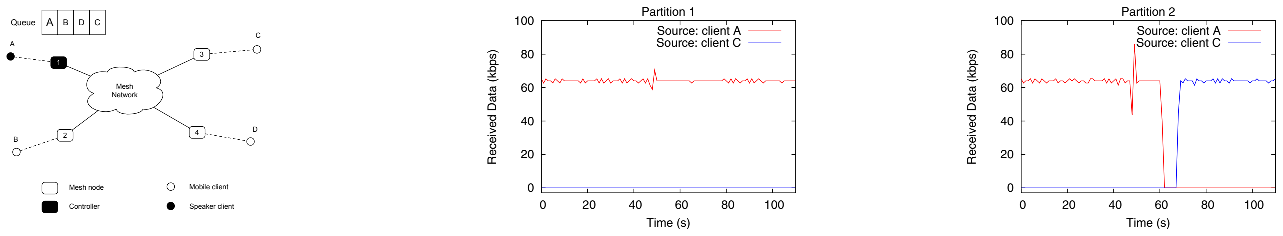
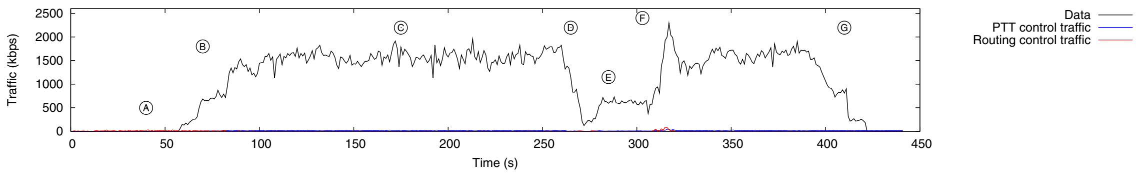


Fig. 12. Scenario showing a network partition with 4 clients in one PTT group. After the partition, the 14-nodes network is split into two independent PTT systems.

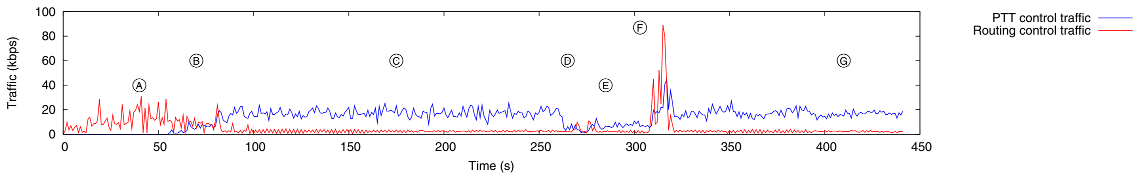
the partition occurs (around second 60). However, the right side of the partition lost the controller. After approximately 7 seconds, a new controller is generated (Node 9), the requests are recovered, and client C is granted permission to speak, as shown by the second partition's view in Figure 12. When the network merged (graph not shown), Clients' voice traffic was corrupted — due to multiple voice streams — for about 686 ms (35 packets), demonstrating that our system quickly eliminates redundant controllers. This demonstrates that the system gracefully handles network partitions and merges.

Finally, we benchmark the system in a large scale partition and merge scenario, with 40 participants in 10 simultaneous PTT sessions. Partition was performed by disconnecting one of the routers. Similar to the scalability experiment, the sending client in each group changes every 10 seconds. Figure 13 shows the overall traffic in the vicinity of Node 1 (as observed by setting Node 1 in promiscuous mode and counting all the packets in its vicinity). To better understand the system's behavior, we present in Figure 13(a) both the data and overhead traffic, and separately, in Figure 13(b), two components of the overhead traffic: routing control traffic (link state, multicast group management) and PTT protocol control traffic. For clarity, we do not show the overhead traffic associated with sharing the state of a client within his vicinity, as it was already shown in Figure 9.

Following the overhead traffic, we can see the route updates that are generated when clients join the network (point A), as well as the overhead related to clients joining a PTT group and asking for permission to speak (point B). The system operates normally until second 265 (point D), when the network partitions. Then, many of the sessions in Node 1's partition lose their speaker or their ability to route to some PTT members. When the connectivity stabilizes, new speakers are granted permission to speak (point E). Note that the amount of VoIP traffic is smaller, as some PTT sessions no longer have members in the current partition, or do not have to route through Node 1's vicinity PTT session messages. Around second 310 (point F), the network merges, causing a spike in both data and overhead traffic. Shortly after that, network routes stabilize and normal operation is resumed. Lastly, around second 380 (point G), all the clients stop speaking and the data rate drops to zero. Since clients did not leave their PTT groups, the overhead associated with maintaining PTT sessions remains constant through the end of the experiment. This elaborate scenario demonstrates the robustness of the system to network connectivity changes



(a) Data and overhead traffic.



(b) Overhead traffic.

Fig. 13. Large case scenario showing a network partition and merge. 40 clients join the 14-nodes system on 10 PTT groups: (A) clients joins, (B) clients request to speak, (C) regular operation, (D) network partitions, (E) network stabilizes after the partition, (F) network merges, (G) clients stop speaking. The marks indicate approximately the middle of each stage.

while supporting a large number of distinct PTT sessions.

VII. Conclusion

In this paper we presented the design and implementation of the first robust Push-to-Talk protocol for wireless mesh networks. The architecture seamlessly integrates standard VoIP phones as well as regular phones and cell phones from the PSTN network to support heterogeneous environments. The protocol provides high availability while efficiently arbitrating the PTT sessions and efficiently disseminating voice traffic. We implemented our protocol within an open source wireless mesh system and evaluated it in a 14 node testbed. Experimental results demonstrate the scalability of our system and its operation in the presence of network connectivity changes such as partitions and merges. This preliminary evaluation of the system shows that PTT is a viable application for self-organizing mesh networks.

References

[1] "The federal response to hurricane katrina: Lessons Learned," February 2006, washington, DC: Office of the Assistant to the President for Homeland Security and Counterterrorism.

[2] M. R. Souryal, J. Geissbuehler, L. E. Miller, and N. Moayeri, "Real-time deployment of multihop relays for range extension," in *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*. New York, NY, USA: ACM, 2007, pp. 85–98.

[3] S. Ganguly, V. Navda, K. Kim, A. Kashyap, D. Niculescu, R. Izmailov, S. Hong, and S. Das, "Performance optimizations for deploying voip services in mesh networks," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 11, pp. 2147–2158, Nov. 2006.

[4] Y. Amir, C. Danilov, M. Hilsdale, R. Musaloiu-Elefteri, and N. Rivera, "Fast handoff for seamless wireless mesh networks," in *MobiSys 2006*. New York, NY, USA: ACM Press, 2006, pp. 83–95.

[5] O. M. Alliance, "Push-to-talk over cellular (poc), release 1.0, 2005."

[6] "The SMesh wireless mesh network," http://www.smesh.org. [Online]. Available: http://www.smesh.org

[7] L. DaSilva, G. Morgan, C. Bostian, D. Sweeney, S. Midkiff, J. Reed, C. Thompson, W. Newhall, and B. Woerner, "The resurgence of push-to-talk technologies," *Communications Magazine, IEEE*, vol. 44, no. 1, pp. 48–55, Jan. 2006.

[8] P. Koskelainen, H. Schulzrinne, and X. Wu, "A sip-based conference control framework," in *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2002, pp. 53–61.

[9] R. Malpani and L. A. Rowe, "Floor control for large-scale mbone seminars," in *MULTIMEDIA '97: Proceedings of the fifth ACM international conference on Multimedia*. New York, NY, USA: ACM, 1997, pp. 155–163.

[10] H.-P. Dommel and J. J. Garcia-Luna-Aceves, "Floor control for multimedia conferencing and collaboration," *Multimedia Syst.*, vol. 5, no. 1, pp. 23–38, 1997.

[11] S. Banik, S. Radhakrishnan, T. Zheng, and C. Sekharan, "Distributed floor control protocols for computer collaborative applications on overlay networks," *Collaborative Computing: Networking, Applications and Worksharing, 2005 International Conference on*, pp. 10 pp.–, 19-21 Dec. 2005.

[12] D. Sharp, N. Cackov, N. Laskovic, Q. Shao, and L. Trajkovic, "Analysis of public safety traffic on trunked land mobile radio systems," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 7, pp. 1197–1205, Sept. 2004.

[13] "Project 25, Association of Public-Safety Communications Officials," http://www.apco911.org/frequency/project25.

[14] A. Balazs, "Push-to-talk performance over gprs," in *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*. New York, NY, USA: ACM, 2004, pp. 182–187.

[15] K. Balachandran, K. Budka, T. Chu, T. Doumi, and J. Kang, "Mobile responder communication networks for public safety," *Communications Magazine, IEEE*, vol. 44, no. 1, pp. 56–64, Jan. 2006.

[16] J. Lennox and H. Schulzrinne, "A protocol for reliable decentralized conferencing," in *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2003, pp. 72–81.

[17] F. Maurer, "Push-2-talk decentralized," 2004.

[18] Y. Kado, A. O. Lim, and B. Zhang, "A study of wireless mesh network routing protocols for push-to-talk traffic," in *Proceedings of the 16th International Conference on Computer Communications and Networks, IEEE ICCCN 2007, Turtle Bay Resort, Honolulu, Hawaii, USA*, 2007, pp. 197–202.

[19] "The Spines Overlay Network," http://www.spines.org.

[20] J. Moy, "Multicast extensions to OSPF," IETF, RFC 1584, Mar. 1994. [Online]. Available: ftp://ftp.isi.edu/in-notes/rfc1584.txt

[21] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," *RFC 3261*, Jun. 2002.

[22] M. Handley and V. Jacobson, "SDP: Session Description Protocol," *RFC 2327*, April 1998.

[23] J. Rosenberg, J. Peterson, H. Schulzrinne, and G. Camarillo, "Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)," *RFC 3725*, April 2004.

[24] H. Schulzrinne and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals," *RFC 2833*, May 2000.