SpaceOps Conferences
16-20 May 2016, Daejeon, Korea
SpaceOps 2016 Conference

Open Source based Voice Communication for Mission Control

Markus Töpfer[1], Anja Sonnenberg[2] and Dr. Rolf Kozlowski[3]
German Aerospace Center (DLR e.V.), Cologne, Germany

Voice communication is a critical service within Mission Control Rooms, as voice communication is the most practical and precise way to transmit information between human beings. It is used in any control room environment for space mission support to enable collaboration within the room and to external connected entities. Today's systems are highly specialized turnkey-based vendor solutions, which are inflexible in terms of adaptability and extension capabilities. These systems are based on vendor related proprietary protocols and are built on top of voice distribution technologies. In this paper we describe the core of a Voice Communication System (VoCS), which directly integrates the special requirements needed for Mission Control Room environments. Our solution will enable an overall system design with component-based implementations and interoperability between different vendors for different parts of the system. In addition our definition of the VoCS core will allow different levels of integration into existing infrastructures, as well as client implementations in the range from fully dedicated hardware-based infrastructures up to the provision of VoCS functionality within Web-based implementations. We will describe the idea using a prototype implementation based on FreeSWITCH, an Open Source media backend with a carrier grade cross-platform multi-protocol soft-switch, and the most differentiating client solution to common VoCS clients, a full virtual Web-based client.

Nomenclature

API	=	Application programming interface
IoT	=	Internet of Things
IP	=	Internet Protocol
LDAP	=	Lightweight Directory Access Protocol
MCC	=	Mission Control Center
NAT	=	Network address translation
PDH	=	Plesiochronous digital hierarchy
QoS	=	Quality of Service
RBAC	=	Role-based access control
SIP	=	Session Initialization Protocol
STUN	=	Session Traversal Utilities for NAT
TCP	=	Transmission Control Protocol
TDM	=	Time-division multiplexing
TDMoIP	=	Time-division multiplexing over IP
TURN	=	Traversal Using Relays around NAT
VoCS	=	Voice Communication System
W3C	=	World Wide Web Consortium

[1] Scientific Researcher, Space operations & Astronaut Training | Communication & Ground Stations, Münchener Straße 20, 82234 Wessling
[2] Scientific Researcher, Simulation and Software Technology | Distributed Systems and Component Software, Rosa-Luxemburg-Str. 2 10178 Berlin
[3] Department Head, Space operations & Astronaut Training | Communication & Ground Stations, Münchener Straße 20, 82234 Wessling

I. Introduction

MISSION Control communication is structured, formal, and communication paths are organized in context related communication groups. The information flow is canalized throughout the team, up to the flight director in charge. Only relevant information for the audience group is transmitted within a communication channel, called the Voiceloop[1]. Voiceloops are organized as communication groups with a dedicated purpose. For example a Flight Director Voiceloop is used to lead the team, whereas a Power Sources Voiceloop is used to collaborate specifically within the topic area of Power Sources. Operators choose a Voiceloop to talk only in the context of the channel. This way each Operator is able to select channels within his interest area and listen to topics relevant to him and his current tasks. These tasks and interests may change and the selection of the Voiceloops may be different at another time. Being able to configure different permission sets, using Voiceloops for group communication, as well as parallel participation in multiple channels at the same time, are the most differentiating parts of Mission Control Room communication systems in comparison to standard conferencing systems.

Initial telecommunication systems were used to transmit voice data over distances. They are the roots of any long distance data transmission systems. Originally data transmission was an additional asset in circuit switched telecommunication networks. Nowadays this status changed. Current telecommunication systems are built to transmit data and voice communication became an additional asset in these packet-oriented networks. Telecommunication providers are in a process to change or already changed their core networks. Customer interfaces based on PDH are phased out and the roll out of IP-based external connections to carrier networks will be finished within the next years.

Current voice system designs mitigate the need to change with a provision of external interfaces to adapt to IP-based interconnections. It is possible to use TDMoIP converters to connect a time division multiplexing-based core to a packet-based long distance transmission network. However, next generation Voice Communication Systems (VoCS) for Mission Control Rooms need to take into account a packet oriented core design.

Redefining the design and architecture of VoCS requires a breakdown of the core functionality of these systems.

Within this paper we will describe a basic and simple architecture that provides the core functionality of VoCS in an infinite abstracted and virtual way. We reduced VoCSs from a dedicated highly redundant hardware infrastructure to a virtual system embeddable in other infrastructural components. This system is presented in Part III of this paper. Part II contains a basic introduction to VoCS and short descriptions of essential technologies for our system design. This paper closes with a short discussion and a conclusion.

II. State of the Art

A. Voice Communication System (VoCS)

In Mission Control Center (MCC) context VoCS is defined over two independent functional core areas. The organizational area is used to manage communication groups. The technical area on the other hand transmits voice data streams and implements the organizational setup.

The organizational component defines a group communication context. It is predefined who needs to talk to whom to ensure the success of a mission. This predefinition includes activities and entities. Activities need to be performed. Entities are related to the activities and perform the tasks within the group context. An entity may be a role, an organization, a location, or a human being. Communication needs to be transmitted between the different entities within a communication group. In MCC context this is done by building a Voiceloop. A Voiceloop is similar to a traditional telecommunication conference, but differentiates in a very important fact: A Voiceloop is not hosted at one specific organization or location. Instead a Voiceloop may be shared between different organizations. Each organization may use its own VoCS to distribute the Voiceloop within itself. A Voiceloop may connect multiple telecommunication conferences over different locations.

Communication provision needs to be adaptable to activity related setups, as the communication is organized within task or activity related communication groups. Each activity within Mission Control Room scenarios can involve multiple sub activities in parallel. An operator needs to be able to adapt his current communication setup to all sub activities involved. This means he needs to be able to switch on and off relevant Voiceloops in respect to his current task set.

Abstracting the organizational perspective of a VoCS leads to a provision of a multi-party, multi-conferencing environment with user adaptable participation state selection capabilities for predefined permission restricted voice conferences, which are connected over multiple locations.

From a technical point of view VoCS needs to be able to transmit and mix voice data. It must provide selection and interconnection capabilities for different Voiceloops, and ensure enforcement of permission restrictions. These restrictions are defined within the group context.

i. VoCS architectures

Current VoCS architecture can be classified by their transmission technique as analogue, TDM, or IP-based. Voiceloops are implemented over analogue voice circuits, TDM time slices, IP-based Multicast Groups, or IP-based SIP conferences.[2]

Analogue, TDM, and IP multicast-based systems are using dedicated clients in dedicated environments of VoCS. Communication capabilities are exclusively restricted to supported clients only. Networks, transmission paths, and other infrastructural components are usually fully dedicated VoCS components. Some kind of dedicated back end is used to configure and monitor the environment and to host the group definitions and permission restrictions. All of these systems are turnkey-based vendor solutions. Interconnection capabilities with other VoCS environments are provided over different interfaces e.g. a T1/E1 TDM-based interface, an analogue connection, or a SIP trunk between different environments.

Voice transmission technology is either hub-based or bus-based. In hub-based environments Voiceloops are mixed at the hub and the mixed stream is transmitted to the client. They need to implement a state signaling mechanism to signal to the hub which Voiceloops the user has selected and should therefore be contained in the mix. In comparison bus-based environments use the systems bus to transmit all Voiceloops to all clients. They don't need a state signaling mechanism. The state selection of a user is client related and mixing is performed directly at the clients.

In both environments permission enforcement may be performed at the client or within the backend. Hub-based technology may directly implement permission enforcement, because the Voiceloop mixing is performed within the backend. Bus-based environments on the other hand need to encrypt each Voiceloop stream individually if they want to ensure permission enforcement. As both types of systems rely on a dedicated infrastructure, permission enforcement is outsourced form the backend to the clients for any VoCS environment known to the authors.

ii. User repositories

User repositories are used to store authentication credentials and permission sets of a specific user. This is the same for VoCS. VoCS uses role-based access control (RBAC). Permission sets are bound to a role instead of a user and the user is assigned to the role. In Mission Control Room environments the most common used user repositories are Active Directory from Microsoft or the Open Source alternative OpenLDAP. Both support a common protocol: Lightweight Directory Access Protocol (LDAP).

B. Voice over IP (VoIP)

VoIP is the current standard technology for voice transmissions. This terminology only states that voice packets are transmitted over IP packets. Nonetheless VoIP is often used as a synonym for the combination of the Session Initialization Protocol (SIP) for signaling with the Realtime Transport Protocol (RTP) for media distribution. But there are several signaling protocols, which may be used to initialize a media path e.g. H.323, ISDNoverIP, MGCP, Megago, XMPP, Jingle, and more.

Traditional VoIP applications use a hub-based approach. Users connect to a server using a signaling protocol to announce their current destination. Servers provide the signaling backend for calls. The media path can be setup over an intermediate server, or as a direct connection between VoIP endpoints. The latest VoIP trend is the WebRTC Protocol.

1. Web Realtime Communication (WebRTC)[3]

WebRTC was developed to enable web browsers, mobile platforms, and IoT devices to communicate over a standardized protocol. Web-based applications can make use of this API to start audio and video calls. This protocol suite is defined by the World Wide Web Consortium (W3C) and is supported by the web browsers Chrome, Firefox, Edge, and Opera and as native implementations on the platforms Android and iOS[4]. For native use under Linux the sources of the Chromium or Android project may be used.

Unlike traditional VoIP protocols, WebRTC is not defining a protocol for signaling. Connection endpoints identify their external IP addresses themselves over the protocols STUN (Session Traversal Utilities for NAT) and TURN (Traversal using Relays around NAT). With these protocols a web browser is able to gather information

[4] https://webrtc.org/ [visited 29 Mar. 2016]

about the external IP address of its own connection to the Internet. This address is sent to a communication partner over a signaling protocol, which is not further defined by the WebRTC protocol suite.

i. Simplified comparison of traditional VoIP protocols and WebRTC

SIP as example for a traditional VoIP protocol[4]:

(1) Client A and B authenticate themselves at a server
(2) TCP connection between clients and server is established
(3) Client A signals the server a connection request to client B
(4) The server forwards that request to client B
(5) Some media path and codec negotiation will be done between client A and B over the server as intermediate
(6) A media connection is setup between client A and client B using the negotiated path and codecs (which may be a direct connection or via the intermediate server)

WebRTC - Simplified Scenario STUN[5]:

(1) Client A and B requests their external IP at a STUN server
(2) Client A and B use the connection to the STUN to keep the communication path to the Internet open
(3) Some other unspecified protocol is used between client A and B to do the media path negotiation
(4) On successful negotiation the initial path to the STUN servers will be used to transmit data directly between the clients

WebRTC - Simplified Scenario TURN[6]:

(1) Client A and B open a connection to the TURN server
(2) Client A and B keep the connection to the TURN server open
(3) Some other unspecified protocol is used between client A and B to do the media path negotiation
(4) On successful negotiation the media path is setup with the TURN server as intermediate between the clients

The direct connections used in traditional VoIP and WebRTC with STUN make use of hole punching of network devices to setup the media path. This technology usually decreases the delay of the connection as fewer hops are used within the path.

WebRTC and traditional VoIP technology are pretty much the same, except for the absence of a signaling protocol in WebRTC.

III. System Architecture

A. Preliminary considerations

VoCS functionality is based on multiple parallel telecommunication conferences. Within VoCS users will not call another user endpoint, instead they access a Voiceloop. Unlike in traditional VoIP the communication between participants is not setup dynamically, but rather there is a dynamic participation within conferences. Available conferences may be implemented as call endpoints and a numbering plan may be used to identify a Voiceloop as a number to call. An example of such a setup using FreeSWITCH and traditional PABX functionality was described in Ref. [7]. Nevertheless such a call-based solution is not fitting to differentiate between participation states (Talk, Monitor). Call implementations of traditional telephony systems use different states like setup, proceeding, alerting, connect, disconnect, release, and release complete (example ISDN).[8] Their highest state is connected. Once a call is connected the media path between endpoints is established in both directions and media flow in both directions is possible. Media attributes of a call can be controlled to deaf or mute a call endpoint. This makes it possible to differentiate between a Talk and a Monitor only participation.Fehler: Referenz nicht gefunden Signaling of participation states on the other hand can be implemented as an extension of traditional VoIP protocols.

Using extensions of existing VoIP protocols and attribute changes on a call object to differentiate between states makes it feasible to implement the required functionality for one Voiceloop. But VoCS provides multiple Voiceloop participations in parallel.

To keep an overview of all participation states of one client using multiple calls with to VoCS adapted functionality requires a backend which tracks all interactions between client and VoIP server to generate a state overview out of all exchanged signaling information. This would be a state tracking backend, not a state enforcement backend. In this context state enforcement can be implemented by using the logic of the protocol adaptions. However, distribution and reestablishment of the complete client state between different backends will be challenging and make a redundancy concept pretty hard. In addition are the actual media states not tightly coupled to the signaling states. Instead they are only loosely coupled based on the logic of the implementation.

The previous section describes a bottom up system design. It takes available protocols and functionality of a VoIP system and implements the required functionality of VoCS on top of it. This is the traditional way VoCS are designed. In our design approach we intent to avoid such a setup. We rather want to build our system with a top down approach, where the special functionality for Mission Control Room conferencing is the base for the VoCS design.

B. Abstract system design

In the beginning of the paper the setup of a Voiceloop was described. Within a user repository all information about Voiceloops are available and predefined. It is defined which users are allowed to use the VoCS, which Voiceloops are available, and which participation states a user is allowed to use within a Voiceloop.

In principal the VoCS only has to deliver and present this information to a user over some client interface, to enforce the predefined rules, and to deliver media connections based on current states for the currently used Voiceloops. VoCS is basically a state machine for media connections with a state selection capability for human end users. This state machine needs to be tightly coupled to a media input and a media output resource, to send media to a Voiceloop (Talk) or to receive media from a Voiceloop (Monitor). Therefor we implemented an Input-Processing-Output-based environment for permission descriptions of VoCS. Also a media backend needs to be able to mix different audio streams and deliver it to any interested party.

With this VoCS design, both, the input to the system, in terms of authorization and communication patterns, as well as the output, in terms of presentation of these patterns, are highly customizable and can be easily adapted to the specific needs of an organization or mission.

C. System Concept

The system concept is designed to be as simple as possible. It is connecting a permission description (LDAP definition of a Voiceloop) with a permission presentation (client user interface). Over the client's user interface a user is able to communicate with the state machine. The state machine on the other hand triggers the media backend to switch media connections. The current prototype implementation of the media plane offers two unidirectional channels to the media backend: one channel to send audio data to the media backend and one channel to receive audio data from the media backend.

The state machine is implemented within a proxy. This proxy connects LDAP, client, and media backend. This way VoCS specific functionality can be connected to available media mixing and distribution backends.

We created a protocol for state and information distribution based on the functionality required for VoCS. This protocol is described in section D.

For reverse gathering of a permission definition we created a LDAP scheme file. As the design and implementation is focusing on a blueprint architecture based on Open Source, the repository of interest in this paper is OpenLDAP. OpenLDAP already provides a couple of useful ObjectClass definitions for RBAC within the core.schema[5], for example the Object Class organizationalRole. It includes an attribute list with user entries, which is called roleOccupants. To be later able to manage the Voiceloop permission sets, a Voiceloop is defined as a custom ObjectClass within the LDAP scheme. It is possible to import this scheme to any OpenLDAP implementation.

A Voiceloop object has three attributes: a name, a list of members who are allowed to monitor this Voiceloop, and a list of members who are allowed to talk in this Voiceloop. It uses the same semantics as an organizationalRole. This makes administration of VoCS with LDAP straight forward. An administrator just connects a role to either the monitor or to the talk list to enable this Voiceloop for the role. Subsequently a request for all objects of the Object Class Voiceloop is sufficient to reverse build the whole permission structure for VoCS from LDAP.

D. State exchange protocol

The state exchange protocol is based on customized event-based remote procedure calls for the VoCS state machines. A state machine is used at client side, as well as within the backend.

Client and server functionality are triggered using a JSON protocol with the base elements <EVENT>, <TYPE> and <PARAMETER>. An event identifies the actual procedure to be called. The type identifies the distribution level and Parameter sets are used to provide relevant information sets for the procedures.

The following procedures are defined:

(1) *Mission context selection* is used to identify available missions within the system. This functionality is interesting for use cases with multi mission support. Each or several missions may use their own dedicated LDAP backend. In this case selecting the mission context may switch the whole backend,

[5] http://www.openldap.org/doc/admin23/schema.html [visited 29 Mar. 2016]

including the media backend, within the proxy. Within multi mission environments a user may work in more than one mission and in each mission he may have similar roles. In this case he may want to select the appropriate mission to easier access the relevant role information for the tasks he is going to perform. To identify available missions a client must send an event <AUTH_PROJECTS> and the server will reply with a list of reachable, respectively configured projects from the backend. Mission projects are user selectable and a project of interest must be sent as attribute within the authentication process.

(2) *Authentication* is used to identify a user within the system. To authenticate a user the client must sent a <AUTH_LOGIN> event to the server including the authentication credentials and the mission context as parameters.

The server routine for the authentication event is:

a. The server uses the credentials to authenticate the user over LDAP.
b. The server will request LDAP for all of the authorization levels (roles) assigned to the user.
c. The server will send an event <AUTH_ROLES> back to the client. Available roles are sent as parameter.
d. The server will attach the authenticated session to a list of all active sessions of the user, as the server keeps state about all sessions a user has.

In case of successful authentication, the client will receive the list of roles for the user and will use this information to display them for role selection.

(3) *Authorization* is used to select an available authorization level (role). To use an authorization level a client must sent an <SWITCH_ROLE> event to the server.

The process to switch a role in the server state machine is the following:

a. The server will request LDAP for all Voiceloops assigned to the role (or may use a cached previous search).
b. The server will send a <AUTH_VOICELOOPS> event back to the client session with an attribute list including all Voiceloops and permission levels of the role as well as currently active roles within the Voiceloops.
c. The server will broadcast the <SWITCH_ROLE> to all active sessions of the user.
d. The server will broadcast the <SWITCH_ROLE> to all active sessions of the role.
e. The server will attach the client session to a list of all active sessions of the role, as the server keeps state about all active sessions within a role.

If a client receives a <AUTH_VOICELOOPS> event it will use the information to draw the user interface. If a client receives a <SWITCH_ROLE> event over a user broadcast, it may use that information for user state tracking within different clients. If a client receives a <SWITCH_ROLE> event over a role broadcast it may use this information to show login or logout messages of users within this role.

(4) *States for Voiceloop* participation states are “OFF”, “MONITOR” and “TALK”. In addition audio related states are used for “VOLUME” and “TALKING”. As soon as a user interaction results in a state change e.g. the user is pushing a Voiceloop button from off to monitoring, the event <SWITCH_STATE> will be send from client to server.

The server will perform the following steps on state selection events:

a. The server will trigger the media backend to switch to the desired state.
b. The server will broadcast <SWITCH_STATE> to all sessions of the user.
c. The server will broadcast <SWITCH_STATE> to all sessions connected to the Voiceloop.

If a client receives a <SWITCH_STATE> message as a loop broadcast it may use that information to inform the user about new members entering a Voiceloop or members leaving a Voiceloop. If a client receives a <SWITCH_STATE> message as a user broadcast it can be used to synchronize all the clients the user is currently authenticated at.

The event <SWITCH_VOLUME> will be forwarded by the server to the media backend to change the volume of the media within the audio summation.

The event <SWITCH_TALKING> is used to indicate the start or end of the TALKING state. It indicates the user is actually using the VoCS system to talk within a Voiceloop. The server is broadcasting the <SWITCH_TALKING> message to all sessions connected to the respective Voiceloop. If a client receives a <SWITCH_TALKING> it may use the information to indicate who is currently talking in a Voiceloop.

(5) In addition we defined some other event types e.g. for playback of recordings, but these are in the context of this paper out of scope.

With such a protocol and state mechanism at client and server side the system is able to connect and implement:

(1) A permission behavior description within a LDAP repository.
(2) A customizable state behavior description within the server.
(3) The possibility to implement permission visualization within the client.
(4) A fully customizable state behavior description within the client.
(5) The possibility to implement state synchronization between different clients based on user logins.
(6) State awareness of Voiceloop participation states at each component of the system.

In addition, with this protocol it is possible to implement all signaling related scenarios for VoCS using only six different events: context signaling, authentication, authorization, state switching, volume switching, and talking. Half of the events are authorization related and already include the ability to switch the whole permission backend to a different environment. The protocol is able to perform full state synchronization based on authorization levels (role broadcast) as well as full state synchronization based on user logins (user broadcast) and in addition full state synchronization over room-based subscriptions (Voiceloop broadcast).

E. Session Handling

There are several ways to introduce session handling in the system. The simplest is to use so-called long running TCP sessions, where the connection itself is used as a session. With this approach it is not necessary to introduce an additional session handling or session distribution protocol. For TCP connections this means as long as the connection is active and established, there is a session between client and server. If the connection is served, the session is closed and has to be reinitiated. To keep a TCP session open a keep-alive mechanism needs to be put in place. This has the additional advantage that all active clients can be monitored and it can be verified that they are all indeed available. Another advantage of an open TCP session is network related. The session has already passed all network filtering, firewall rule checking, and initialization routines at the server and all involved devices of the path.

Long running TCP sessions are the base of the HTTP/2 specification[9] and are also used for WebSocket connections. WebSockets are a protocol for bidirectional communication within Web environments and allow server push technologies. They also have a build in keep-alive mechanism within the protocol.

For media communication we need a media path between client and server and a media path to the media backend. Here we use the capabilities of TURN to relay between endpoints. A TURN server is implemented within the proxy and relays media connections between clients and media backends. Implementing the TURN server in combination with a WebSockets server allows a tightly coupling between media and signaling related information.

F. Media switching functionality

The media path is extended from the intermediate TURN-based proxy server to the media backend. Each user login at the proxy will result in two user logins at the media backend FreeSWITCH. The first login at FreeSWITCH is used to send audio from client to server. This channel is called TALK. The second one is used to send audio back to the client. This is called MONITOR. Both channels are independent from each other and unidirectional.

The TALK channel is connected to a traditional voice conference within FreeSWITCH. Each client has a dedicated TALK conference and a dedicated MONITOR conference. Media setup from FreeSWITCH over the TURN functionality back to the client is done via WebRTC. The media path is setup from FreeSWITCH to the client. Therefore the FreeSWITCH server always initiates a call, never the client. The attribute <TAGNAME> is used to identify the TALK and MONITOR calls. Next to <TAGNAME> another important attribute necessary to setup a call is <SDP>. It contains the media description, the codec, and the connection path to be used. A client connects the <SDP> for TALK with the microphone and the <SDP> for MONITOR to the loudspeakers. Media initialization finishes with two unidirectional media connections from a client to two client dedicated voice conferences (TALK and MONITOR) within the media backend.

The clients MONITOR conference builds a conference of all Voiceloops activated for monitoring. The clients TALK conference builds a conference of all Voiceloops activated for talking. With this it is possible to provide multi talk and multi listening capabilities within the system.

If the proxy server receives a <SWITCH_STATE> event an internal mechanism using the FreeSWITCH event socket interface will connect the client conference TALK or MONITOR with the Voiceloop conference. Internally a muted connection is used between the clients MONITOR conference and the Voiceloop conference. This way a listen only connection is enforced. For TALK state changes the clients TALK conference is connected with the Voiceloop conference. In case of a <SWITCH_STATE> to off, all client conferences will be internally disconnected from the Voiceloop conference.

If either the WebSocket or the TURN media connection to a client is lost, the proxy will automatically disconnect all internal connections to its conferences in FreeSWITCH.

G. Summary of components and protocols

The system architecture includes the components client, proxy, authorization backend, media backend, and IP network. Core component of the architecture is the proxy that relays media streams between clients and a media backend, switches media states, and provides capabilities to connect external authorization backends.

Application level protocols used within the core of the system are: JSON, WebSockets, TURN, (S)RTP, SDP, OPUS, G.711, LDAP, and the state exchange protocol defined within this paper. TURN, SRTP, OPUS, G.711, and SDP are part of the WebRTC suite.

A proxy application needs to implement WebRTC, JSON, WebSockets, LDAP, and the state exchange protocol with six different event procedures and five distribution types (user broadcast, role broadcast, loop broadcast, server broadcast and server unicast).

H. Flexibility of the System

The system architecture was designed to be as flexible as possible. Based on the Input-Processing-Output design pattern, components are flexible and loosely coupled. Protocols on the other hand are tightly coupled within the system.

Clients may implement the state exchange protocol in a way to use more than one proxy in parallel. This allows to setup redundant active-passive media communication paths to the backend and implement logic to switch the active path based on network or proxy conditions. The proxy implementation may use different kind of media backends to mix audio streams or to connect external media paths to other systems. The overall design is motivated by the intention to allow all components to be replaceable.

The most prominent example is the coupling between clients and the proxy server. Both implement the state exchange protocol that defines the communication between them. Every client that implements this protocol is able to communicate with the proxy server and the other way around every server can serve as a proxy, if it implements the state exchange protocol and provides the necessary data for the client.

In traditional voice conferencing systems clients are general dedicated to specific proprietary hardware. The here described solution on the other hand is software-based with protocols as interfaces between different components. Possible client implementations range from a dedicated implementation on dedicated hardware, over integration of dedicated software clients on specific operation systems such as Linux or Android up to platform independent solutions such as Java or Web clients. This combined with the systems flexibility regarding the client-proxy relation presents a big advantage.

Usage of the state exchange protocol allows several different client implementations to use the same backend. For example, this allows running a dedicated Linux, Windows, or Android client in a dedicated voice network, such as a control room, and a Web application in another network, such as any office environment, at the same time.

Another possibility is the implementation of different specialized types of clients. For example in Ref. [10] a recording application is implemented using the here described system backend.

User interfaces are fully customizable and user input strategies may differ in different types of client applications. We have developed two example clients, one Web and one Android-based. Both user interfaces were designed to have the same look and feel to make it is easy to switch between them. Naturally, there are platform differences as Android devices are touch screen devices and Web applications need mouse interactions. They are used to show the integration level of this architecture and to test two different input strategies: click-based user navigation and multi-touch gestures.

IV. Discussion of the Design

The here introduced system design allows to implement a redundant and lightweight VoCS. It adds Mission Control Room specific VoCS functionality to standard telecommunication conferencing backends. This approach opens up the design of VoCS to different standard out-off-the-shelf IT components such as laptops and tablets and is a counter draft to turnkey-based vendor solutions.

With our approach, VoCS may be integrated to different kinds of environments. VoCS functionality is no longer a dedicated system, but rather a service on top of different infrastructural components. It therefor becomes embeddable in other types of service provisions.

A very important part of voice communication is latency and delay within the transmission. Evaluating the media transmission quality was not a focus of this paper. But running a VoCS functionality on top of a network not

optimized for voice transmissions and without QoS mechanism may decrease the overall service quality of voice communication. Our proxy functionality may be used within dedicated networks to ensure a high quality voice transmission, but it also opens possibilities to use the proxy to external network environments. This may result in a lowering of the voice quality by not optimized network environments. In exchange the VoCS is opened up to Internet-based connections. This may result in new operational scenarios. Also VoCS becomes accessible for backup communication provisions, something that is not possible with current solutions.

There are also some limitations with the here described media implementation.

A first limitation is related to the codec. Internal conference connections within FreeSWITCH can be done with G.711, the standard codec for audio transmissions in traditional PABX environments. To avoid recoding of audio streams G.711 needs to be included in the SDP descriptions for the clients. In WebRTC-based environments OPUS is the standard codec, due to its adaptability to network conditions as well as a lower bandwidth requirement and a very low encoding latency. A low encoding latency is highly recommended for media mixing. G.711 is also supported by WebRTC but only as a fallback.

A second limitation is the setup of two independent call streams. This setup is very good in terms of separation, but requires two network streams between client and TURN server. Both streams are necessary because FreeSWITCH is call-based and a VoIP call is duplex. The two calls to the two independent conferences require to deaf the MONITOR conference, while actually talking in this Voiceloop within FreeSWITCH. Otherwise audio would be looped from the TALK call, over the TALK conference, to the Voiceloop conference, over the MONITOR conference, through the MONITOR call, back to the user client. This may result in feedback tones between mic and audio or simply delayed echo of the talkers own voice.

The best solution to solve the issue is to extent the TURN relay with direct media mixing of OPUS streams. To provide connections to the external world over traditional VoIP protocols FreeSWITCH may be used as media gateway. The Voiceloop would be shared between the TURN relay and FreeSWITCH, just like for any other backend to backend connection.

In the next stage of our research we intend to include multi conferencing capabilities in the proxy to connect traditional conferencing media backends.

Also within the media relay a bidirectional audio connection shall be split in a receiving and sending channel. Media data received from a client needs to become a TALK channel and media send to the client needs to become a MONITOR channel. Depending on the state triggered within the WebSocket-based signaling component the TALK channel needs to be connected to the conference of the media backend representing the Voiceloop in TALK mode. The MONITOR channel needs to be connected to all Voiceloops selected for monitor and the streams of all Voiceloops connected shall to be mixed back to the MONITOR channel.

At first glance the usage of WebRTC and WebSockets seem to be a Web optimized version of the design. Indeed Web application-based implementations are pretty easy with such an approach. But the functionality with WebRTC and WebSockets can be used within native implementations too. WebRTC native means a secured RTP-based communication channel between two endpoints. To start the RTP-based channel a SDP packet need to be available. The media part is pretty similar to media implementations of other protocols and the media channel is only a RTP stream, which is created by a SDP description. This is exactly the functionality we need for external connections to the proxy. Thanks to the WebSockets keep-alive specification, we are able to trigger if the thread or process providing the WebSocket is answering. With such a mechanism it is possible to identify if the client application is able to answer without checking at operating system level. This functionality is needed for monitoring and to check if a client or proxy implementation is in a good working condition.

Also building a core design based on protocols and protocol suites like WebRTC, WebSockets, HTTP/2 technologies, LDAP, JSON, and SRTP is one of the best options to ensure interoperability and a widespread tested system base. These technologies run in nearly any top tier web browser implementation (except Safari). A wider field test of the base protocol than nearly every available consumer IT device is not possible. At the same time each of these devices becomes a potential device for a VoCS client terminal. From the client side our approach can't even called a vendor lock-in break out anymore, it is more a tear down.

If a proxy implementation is able to perform the multi conferencing functionality within the media relay (just like we described in the short outlook above), the same will happen for media backends. Any standard voice conferencing backend may be used to provide the distribution of Voiceloops over their interfaces.

V. Conclusion

We showed it is possible to abstract the system requirements for VoCS in Mission Control Room environments to a very small functional core stack. This core allows the definition of a broader system design including redundancy concepts, specific client interaction patterns, external interface provisions, hardware and operating system definitions, integration depth to existing infrastructures, and everything else which is important for a complete system design, but unimportant for the core functionality of VoCS. We reduced and simplified VoCS to the Mission Control Room specific use case. This reduction allows and opens up new operational scenarios for VoCS. Interoperability between different vendor implementation can be easily archived using the state exchange protocol and related procedures presented in this paper.

References

[1] Patterson, E. S., Watts-Perotti, J. and Woods, D. D., "Voice Loops as Coordination Aids in Space Shuttle Mission Control", *Computer Supported Cooperative Work (CSCW)*, Vol. 8, No. 4, Dec. 1999, pp. 353-371

[2] The Consultative Committee for Space Data Systems (CCSDS), *CCSDS Report Concerning Voice Communications*, Informational Report CCSDS 706.2-G-1, Issue 1, CCSDS Secretariat, Space Communications and Navigation Office, Space Operations Mission Directorate, NASA Headquarters, Washington, DC, Sep. 2010

[3] Bergkvist, A., Burnett, D. C., Jennings, C., Narayanan, A. and Aboba, B., "WebRTC 1.0: Real-time Communication Between Browsers", *W3C*, Jan 2016, URL: https://www.w3.org/TR/2016/WD-webrtc-20160128/ [cited 29 Mar. 2016] (working draft)

[4] Rosenberg, J., et. al., "SIP: Session Initiation Protocol", *IETF*, RFC 3261, Jun. 2002

[5] Rosenberg, J., Mahy, R., Matthews, P. and Wing, D., "Session Traversal Utilities for NAT (STUN)", *IETF*, RFC 5389, Oct. 2008

[6] Mahy, R., Matthews, P. and Rosenberg, J., "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", *IETF*, RFC 5766, Apr. 2010

[7] Peter, D. L. and Töpfer, M., „Mission Control Room Conferencing using Standard PABX Systems", *SpaceOps 2014 Conference Pasadena, CA*, AIAA, 2014

[8] Telecommunication Standardization Sector of ITU (ITU-T), "ISDN user-network interface layer 3 specification for basic call control", ITU-T Recommendation Q.931, May 1998

[9] Belshe, M., Peon, R. and Thomson, M., "Hypertext Transfer Protocol Version 2 (HTTP/2)", *IETF*, RFC 7540, May 2015

[10] Junge F., Schwien, N. and Töpfer, M., "A Time Domain based Playback User Interface for Voice Communication Systems in Mission Control Room Environments", *SpaceOps 2016 Conference Daejeon, Korea*, AIAA (submitted for publication)