16th International Conference on Space Operations, Virtual Edition - 3 - 5 May 2021.

SpaceOps-2021,8,x1304

How web-based voice communication system clients can change operation in mission control

Anja Bertard[a]*, Markus Töpfer[a]

[a] *Space Operations and Astronaut Training, German Aerospace Center (DLR e.V.), Oberpfaffenhofen, 82234 Wessling, Germany*, {anja.bertard, markus.toepfer}@dlr.de

* Corresponding Author

Abstract

Current voice communication system (VoCS) solutions are primary build for mission control rooms and with mission critical phases in mind. This leaves much potential of VoCS unused. We would even argue that this complicates day-to-day operation work outside of mission critical phases. In this paper we discuss the integration of lightweight web browser-based voice clients into VoCS as secondary, remote access points for less mission critical use cases. This will not only simplify the day-to-day life of control room personal, it will also make operation with third parties easier and VoCS solutions more attractive for space flight related research institutes and training centers.
Keywords: voice communication system, voice over IP, web client, architecture

Abbreviations

CSS	= Cascading Style Sheets
DTLS	= Datagram Transport Layer Security
GSOC	= German Space Operation Center
GUI	= graphical user interfaces
HTML	= Hypertext Markup Language
IP	= Internet Protocol
JSON	= JavaScript Object Notation
PTT	= push-to-talk
QoS	= Quality of Service
RTP	= Real-time Transport Protocol
SRTP	= Secure Real-time Transport Protocol
STUN	= Session Traversal Utilities for network address translation
TURN	= Traversal using Relays around network address translation
VoCS	= voice communication system
VoIP	= Voice over IP
WebRTC	= Web Realtime Communication

1. Introduction

Voice communication systems (VoCS) in mission control rooms are one of the prime interfaces between control rooms all over the world. VoCS play a major part in guaranteeing mission success. It does not matter if the mission is a small local one or an international one spanning several space agencies. VoCS allows and enables all participants of a mission to communicate efficiently and in real-time with each other.

VoCS need to be failsafe and secure to guarantee error-free operations during mission-critical phases of e.g. rocket launches. They are therefore built with these critical phases in mind. VoCS networks are dedicated infrastructures with insular characteristics. VoCS terminals as access points are installed in control rooms. However, there is more to operations than highly critical phases. In fact, highly mission critical phases are vastly outnumbered by day-to-day operation work. But so far day-to-day operation work is largely ignored during VoCS design.

In section 2 we describe how a VoCS design that primary focuses on reliability/safety/security requirements complicates day-to-day operation work outside of mission-critical phases. In this paper we propose to soften the system requirements for day-to-day operation work and expand the VoCS by allowing remote access via Internet Protocol (IP) and web browser-based voice clients.

In this paper we discuss:

- our idea, and how operation and cooperations with third parties like universities or other space flight related research institutes will benefit from our proposed system expansion (section 3),
- the technical liberations and constrains posed by the web and browser environment (section 4),
- the technical feasibility of our idea (section 5) and
- the architecture necessary to build such VoCS (section 6).

We want to further give a short excursion into current application of web interfaces in the space flight environment (section 7.1) and give a short introduction to WebRTC (section 7.2) as one of the main technologies used in our VoCS.

We conclude the paper by giving a summary of our solution (section 8), as well as give an outlook into future work (section 9).

The content of this paper reflects our considerations and design choices while working on the project openvocs. Openvocs is a novel VoCS currently developed for and by the German Space Operation Center (GSOC).

2. Why we need remote access in VoCS

For highly mission critical phases like launch and early orbit operations the VoCS setup with stationary terminals in the control room is the optimal solution. But these phases are not covering the majority of the time span of a common space mission. Launch missions are the obvious exception.

Operations ensure success of a mission over the whole mission lifetime. Different phases of a mission lifetime have different characteristics and requirements. A rocket start will have very strong requirements to ensure operations. The focus of all systems must primarily comply with reliability, safety and functionality requirements. Whereas operations of satellites within final orbits may soften these requirements. In this case the primary requirement may shift to a focus on interoperability. Security, safety and reliability requirements for commonplace operations may all focus on interconnectability as core and primary requirement. With this requirement shift we can address different needs within commonplace operation phases.

Because of current VoCS design, the personnel must be present within a control room to access the VoCS, even for daily briefings or listen only participation. Operations personal need to commute between office and control room area for each briefing to allow VoCS participation. This is time consuming and cumbersome. In addition, it requires that the operation personal is present at the control center to use the VoCS infrastructure. At least in Europe the concept of "home office" has become quite popular during the current COVID-19 pandemic. Such a concept is not possible with the current setup.

As alternative, operations are required to switch to a different kind of collaboration system that works independent of the control room. This results with operation communication being all over the place. The original VoCS intention was to create uniform and structured communication for mission operation. Communication over several different media channels is the complete opposite.

The three main reasons why remote participation is currently not possible are:

- insular characteristic of the VoCS networks,
- cost intensive and bulky VoCS terminals as well as
- dedicated and vendor specific connection capabilities for alternative clients within turnkey based VoCS solutions.

3. Advantages of remote access in VoCS

Meeting in online conferences independent from where we actually are, with people all over the world is now an everyday occurrence for many of us. We are no longer bound to specific locations to meet. Thanks to Laptops we can participate in meetings from everywhere. When we started to design and work on a new VoCS for the GSOC we soon released that with the way we want to build our system, we can actually transfer the "online meeting from everywhere" concept to control center operations.

In this paper we want to describe our considerations on how we can make online participation on VoCS conferences possible, without compromising security. To do this, we need to add two new components to a classical VoCS setup:

- First of all, we need lightweight software-based clients e.g. web browser-based voice terminals as secondary access points.
- To allow a connection between the web app and the VoCS we need network interfaces to allow IP connections from outside of dedicated VoCS networks.

Web-based applications remove the necessity of dedicated physical access terminals. They even remove the necessity to install dedicated software applications. End-user devices like laptops, tablets, or smartphones with modern browsers can be used as voice clients. A web-based VoCS client would therefore allow the operation personnel to participate in VoCS conferences/calls directly from their office or remote locations.

This would not only simplify the day-to-day life of control room personnel, but also the work for distributed teams and bigger control centers that expand beyond one location. For bigger satellite control centers a dedicated landline is appropriate. But there are also cooperations in place with research groups that need VoCS access. Current systems required for research groups to invest in their own, dedicated VoCS terminal. This terminal is connected to the main control center via a landline within dedicated networks. These landlines are not providing connections from outside to the VoCS network, but are extending the network to include places outside the control room. Not every research group is able to invest in such a system, especially for short to midterm cooperations. The introduction of an IP interface and browser-based voice clients, allows control centers to easily grant temporary access to the VoCS. In addition, no extra spending is necessary. Saved costs may be used for actual mission or experiment support.

Mobile research control centers could also profit from using IP based interconnects. Mobile control centers need to connect to major stationary control centers to coordinate lunches.

Our system extension will make VoCS solutions more attractive for space flight related research institutes and training centers. There are several examples where such facilities also require aid of a VoCS. But their requirements can be vastly different from classical control centers. For example, during astronaut training the trainers need to communicate with future astronauts when they are inside their space suite during simulations. Most of the time the trainers need to able to move around, e.g. to view a simulation inside a water tank from different sides of the tank. The ability to wireless connect a portable device like a smartphone or tablet to the VoCS would simplify the set-up and ease of use.

4. Voice client in a browser: technical liberations and constrains

Browser based voice clients are usable on a wide range of end user devises, like laptops, smartphone or tablets. No heavy client installation or configuration is necessary. It is enough to open a web browser and input a URL. Web interfaces are highly adaptable and flexible, and can be used to create interfaces for a wide range of use case. They are cost-efficient and versatile.

The touch controls of the SpaceX spacecraft Crew Dragon (see section 7.1) has proven that the web programming stack of HTML, CSS and JavaScript, is mature enough to program graphical user interfaces (GUI) for spaceflight environment [1]. It is important to note that the web stack is exclusively used to display the GUI and handle touch input. The touch controls in the Crew Dragon are a backup system and are designed and build with redundancies to make the interface more robust.

Browser based voice clients should have similar characteristics. They should be as lightweight as possible. All the logic should be handled by the backend. This would mean a redesign of the current systems. We can provide the same or similar interface redundancies as the Crew Dragon touch interface, not by interface but by system design. We describe how such a system could look like in section 6.

Nevertheless, we already indicated several times that we intend the web client to be a secondary keyset terminal. The main reason why we feel reluctant to use web clients for critical environments is the web client’s dependency on a browser. The usage of browsers introduces a dependency of VoCS clients to a third party. Especially during critical phases, it is not advisable to allow operation to use just any device. Certain browsers may collect and send data back to their manufactures. An outdated or "too new" browser may contain security vulnerabilities. In general, voice clients should only run on managed devices. Only this way can we ensure the usage of appropriate (open source) browses with appropriate settings.

The browser environment also influences how some functions of the user interface need to be handled. Contrary to dedicated client terminals, a web application is not the only program open on a device or even the only tab open in the browser. This means the VoCS client will not always be in focus or even visible. This could have an influence on push-to-talk (PTT) mechanisms in VoCS.

If the voice client is not in focus, a software PTT signal does not arrive at the browser and can therefore not trigger a transfer of a local voice stream. This problem mainly concerns the usability of the voice client user interface and is out of scope in terms of system liberation, as described within this paper. This is a problem that needs further research and will be briefly covered in section 9.

5. Technical feasibility

5.1 Browser capabilities

A lightweight keyset in form of a web browser must be able to provide the basic functionality of a traditional voice terminal.

The core capability of voice communication is the transmission of audio. In detail, we need capabilities of voice communication within two application areas:

- audio input from user side, as well as
- audio transmission and reception over networks.

Both are covered within browser-based environments by the WebRTC (Web Realtime Communication) [2] stack. We introduce WebRTC in section 7.2 WebRTC explicitly targets media transmission. From a user perspective this stack may be used to request and select audio and video input devices. This enables the first application area, connection of microphones and headsets, as well as transmission of local audio streams. The WebRTC stack also implements and defines network transmission capabilities in terms of sending and receiving remote audio streams. This combination comprises the major high-level requirements of any VoCS keysets.

It is further required to provide a user interface with options to select and switch channels (voice loops [3]) as well as other commanding capabilities. This is possible with standard HTML, CSS and JavaScript within each browser. Touch capabilities have been added in recent years [4] and are standard for browser environments nowadays.

In addition, the browser is able to capture other operation system inputs, e.g. connecting a game pad and using the game port input [5] as trigger mechanism for PTT.

A common web browser therefore provides all functionality a VoCS keysets needs:

- able to handle operation system input,
- mature GUI and touch capabilities, as well as
- the WebRTC stack for local and remote handling of audio streams.

This is enabling any hardware that is able to run web browsers, network connections as well as providing audio ports to be potential VoCS terminal devices.

Abstraction of the functionality to a logical level and using unknown and unspecified hardware as VoCS keysets limits capabilities of control of the actual hardware and local end user environment. End users will be responsible for their own setup, e.g. connecting and selecting the right headset in the web application that is running in the browser.

5.2 Server side and network requirements

With the introduction of the web stack and WebRTC the network becomes generic. No dedicated network connection is used. The whole system changes from insular and dedicated to open for anyone. It is even possible for basically everyone to write their own clients. Only condition is to implement a communication protocol for switching. We outlined a suitable protocol in one of our previous papers [6].

Such a setup must be controlled at different levels. Mainly at authentication/authorization level to control participation. This has to be done at server side with a new kind of server and service infrastructure. Network audio streams and user sessions to control these streams have to be connected within the server environment.

The network part of the WebRTC protocol defines functionality to identify and connect media streaming and signaling protocol components [2]. The server environment must be able to use this protocol to relate switching and media transmission of keyset connections. This is similar but also different to standard Voice over IP protocols (VoIP) e.g. Session Initiation Protocol (SIP). The WebRTC stack is implemented with capabilities to connect traditional VoIP endpoints [2].

VoIP as technology to transmit audio over IP networks is dependent on the network in terms of transmission quality (e.g. delay). Some clients may connect over networks without Quality of Service (QoS) loss at network level. QoS at network level means that audio communication is favored over standard data in terms of forwarding to reduce the delay [7]. Other clients e.g. in mission critical scenarios may mitigate delays by QoS enabled or with dedicated network components. A dedicated network component would be similar to traditional setups and has no disadvantages in terms of network level transmissions over traditional systems.

Enabling the implementation of different clients through the utilization of open communication protocols as well as the capability to use these clients within any kind of network environment poses the biggest advantage of our solution. Clients are flexible in terms of implementation. How new clients are implemented depends on system requirements from operations perspective. For example, a mission control room setup may use dedicated client implementations on dedicated hardware within dedicated networks, while a connection from a university to listen to the communication during experiments could be done using standard web browsers over the internet.

This flexibility comes at a cost on server side. Servers must implement the protocols for the stack and ensure secure, authenticated and authorized transmission of audio channels.

The server environment must provide a second characteristic of VoCS in control room context to simplify requirements on client side in terms of capabilities of web browsers, communication transmission, network requirements and customization options. Namely, parallel participation within multiple communication channels at the same time. This implies audio streams send to clients must each contain an audio mix of requested voice loops for each end user connection.

Our approach basically translates into a "phone call" between a web client and the server backend. We have one media connection including the audio stream of all requested voice loops as well as one signaling channel for the selection of voice loops. Any network environment able to deliver phone call level capabilities will be sufficient to connect a VoCS keyset with a server backend.

This however, is unfortunately not as simple as it sounds.

WebRTC as underlying protocol stack for media channel provisioning requires additional capabilities. Both connection endpoints of a WebRTC connection may be located within private networks. In this case a reachable remote destination is needed to identify the public connection address of the endpoints (further details in section 7.2). Since a web-based approach requires a web server to deliver webpage content anyway, keyset endpoints mandatory must have one network connection capability to a web server. A server environment for such setups may use the same device/interface combination of the web server to provide functionality for remote endpoint gathering, namely STUN [8] (see section 7.2). Network environments of the clients must allow connections of local addresses to one remote IP address.

During the WebRTC connection process media streams validate identity using a certificate fingerprint, which was previously transmitted over a signaling protocol (for details see section 7.2). This is the earlier mentioned binding between signaling and media channels. The certificate of the secured Hypertext Transfer Protocol Secure (HTTPS) connection of a web server can as well be used for this. This will only work, if the host, offering web server capabilities, is also providing WebRTC endpoints for the server environment. In fact, this is even compliant to the "same origin" policy within the security level of web browser implementations. The "same origin" policy is not necessarily required by the WebRTC stack [2].

5.3 Summary

In summary we can say that it is technical feasible to use a web-based setup for VoCS keysets. But the following four points must be considered:

(1) A VoIP connection with WebRTC requires one signaling and one media transmission protocol.
(2) A Secure Sockets Layer (SSL) certificate can be used to setup and bind a secured connection.
(3) Client networks must allow outgoing connections to one remote destination.
(4) A server must be able to host a web server and a STUN server, and have WebRTC endpoint capabilities.

6. Architecture

Based on the feasibility assessment we would like to discuss the architecture of our solution.

The system will consist of clients and servers. Servers have capabilities to connect clients and provide functionality to serve the communication requests of users to clients.

A high-level view of the system is pretty simple (see Fig. 1). The system needs two black boxes connected over a well-defined protocol stack based on web technologies. The first black box is a client instance providing local audio input and output, as well as remote communication capabilities over IP based networks. The second black box is a server instance providing data (e.g. a webpage), media and signaling interfaces, as well as actual audio mixing capabilities to serve communication request inputs to clients.

This is a star topology, with a hub as central logic component.

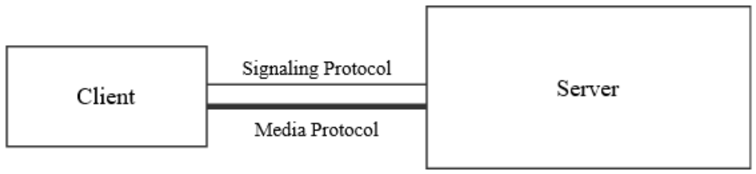

Fig. 1. High-level view of the system: Client and server black boxes are loosely coupled with a signaling and a media protocol

6.1 Protocols

Before we can discuss the functionality, we need to define the protocol stack. We base this stack on the required interconnection technology.

To generalize the architecture, we will focus our considerations on the backend and ignore the web clients for now. For the protocol stack we need:

a. one signaling protocol based on web technology over IP networks for event messaging
b. one media protocol based on web technology over IP networks for audio streaming

6.2 Signaling Protocol (a)

Our signaling protocol base stack is based on three Internet standards and allows implementation of any kind of event signaling mechanism over JSON event messaging.

(1) Transport Layer Security (TLS) [9]
(2) WebSocket [10]
(3) JavaScript Object Notation (JSON) [11]

Secure WebSockets provide a lightweight communication channel and are perfectly suited as transmission protocol for signaling messages. JSON provides a lightweight data exchange format for custom communication protocols.

An example for an actual messaging protocol within the signaling layer can be found in [6]. In [6] we identify the basic event message exchange required in the event layer for VoCS.

6.3 Media Protocol (b)

As we already mentioned, we use WebRTC as our web technology for the media stack. In terms of media connection capabilities for the black boxes, this protocol stack makes use of the following network protocols:

(4) Interactive Connectivity Establishment (ICE) [12]
    5. Session Traversal Utilities for NAT (STUN) [8]
    6. Datagram Transport Layer Security (DTLS) [13]

7. Real-time Transport Protocol (RTP) [14]
8. Secure Real-time Transport Protocol (SRTP) [15]
9. DTLS-SRTP [16]

A protocol stack using these nine protocols in combination with the messaging mentioned in [6] will be sufficient to provide all client/server communications for the system.

6.4 Client - Server coupling

So far, we have ignored protocols for webpage delivery (e.g. HTTP and HTML) on purpose. It's not required for the basic architecture. An architecture for a web client is already a specialization of the basic architecture. This specific system specialization will require HTTP protocols on server side to deliver the webpages.

A native Linux client can connect to the same server black box over the same protocol stack mentioned above. This implementation will not require a HTTP web server component. A native client will already contain the GUI and application logic inside the installation on the client hardware. Same logic will apply for other operation systems and clients e.g. iPad or Windows apps as well as dedicated integrated VoCS terminals.

Our architecture specification is an interconnection protocol stack in combination with an event protocol stack. Its main advantage is that we don’t limit ourselves to one specific implementation and to allow loose coupling between client and server environments.

6.5 Server architecture

As mentioned in subsection 5.2 a major functionality of VoCS is multi-party multi-conferencing. The server environment needs to provide these capabilities to mix audio streams based on event mechanism to serve client requests (e.g. switch on a voice loop) protected by an authorization mechanism (permission to switch on that voice loop).

Within the next section, we will provide an overview about the specific implementation of the server black boxes architecture as it is used by the openvocs project. As described in the last subsection, the basic architecture is not limited to connect web browser-based clients, but open for all kinds of client implementations. In the following we will focus on the specifics of web browser-based clients.

Our server architecture contains gateways in combination with an audio backend (see Fig. 2). Gateways are used to connect the client protocol in terms of signaling as well as media functionality. Within the audio backend, multi-conferencing capabilities are provided. All instances used are lightweight processes, also known as micro services [17]. All these services use their own specific signaling protocol for its specific purpose.

Gateways are used to terminate the interconnection protocol stack in terms of media, as well as signaling connections. The signaling event mechanism is used to control the audio backend on client requests, based on the user’s authorization level. The media gateway is used to terminate a client media connection and provide an endpoint for WebRTC communication and forward media between client and audio backend. The combination of signaling and media connection is called endpoint.

Within the system, a session is spawned for each endpoint between web gateway, media gateway and audio backend. A session is therefore the state of a user in terms of audio participation.

The services we use at architecture level are:

(1) Web gateway (client / server event communications and authentication)
- to provide the HTTP capabilities to load a web client (webpage with JavaScript)
- to provide the signaling (event messaging) protocol for client events
- to provide the authentication and authorization component of the systems

(2) Media (ICE) gateway (client / server media communications gateway)
- to provide a WebRTC media endpoint on server side
- to transcode from secure (SRTP) to insecure media content (RTP)
- to forward all media data of a client session between client and backend and vice versa

(3) Audio resource manager (audio layer orchestration)
- to request an audio mixer
- to request an echo canceler
- to request a voice loop mixer
- to provide registration and orchestration of audio micro service components

(4) Mixer
- to provide actual audio mixing

(5) Echo canceler
- to remove the users voice bevor stream is played back to user

The web and media gateways are coupled. They provide a one-to-one combination of media and signaling related connections for an endpoint. In WebRTC a media connection is established between dedicated, authenticated and secure connection endpoints. The authentication of a user over a signaling connection triggers the setup of a media connection. From the system perspective the media and signaling channels of a session are at this point combined as one endpoint.

The authorization within VoCS is role based. The endpoint itself identifies connection layer functionality but is independent of the actual state within the system session. All states in terms of participation within voice loops are related to the role the user chooses for the current participation within the VoCS.

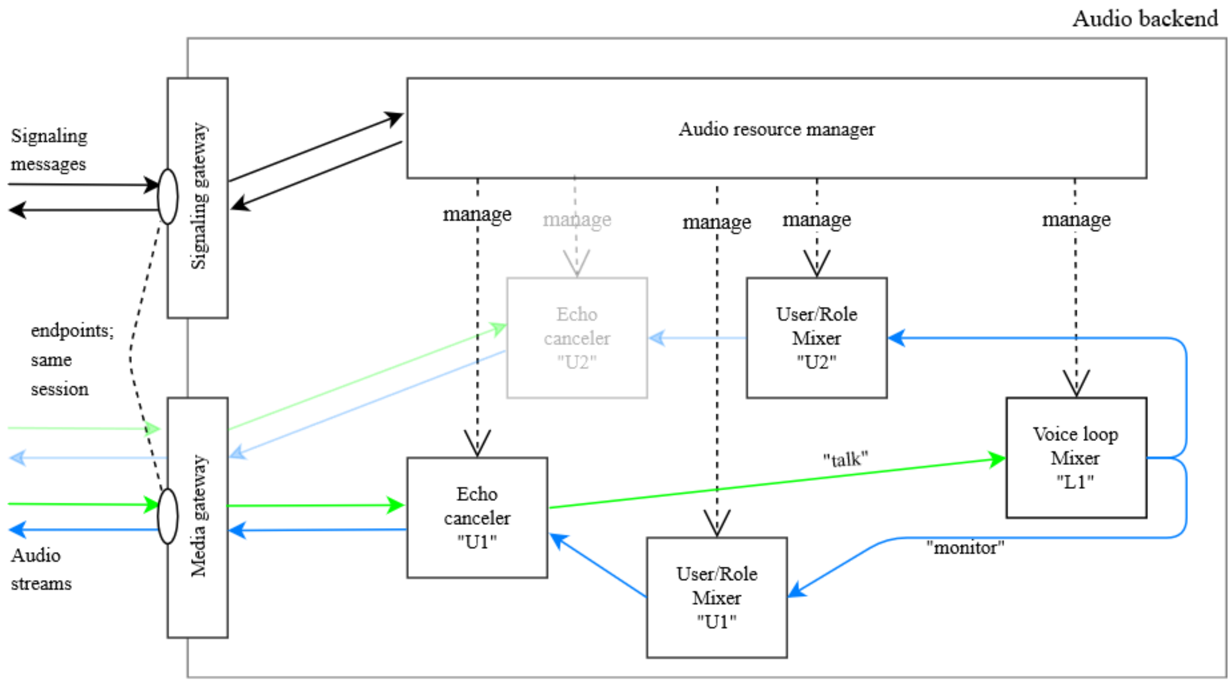

Fig. 2. Backend Architecture with two user/role and one voice loop mixers: Signaling and media stream for user/role identity "U1". “U1” is currently talking in voice loop “L1”. “U2” is listening. Signling for “U2” is not included.

6.6 User/role mixer

A user/role combination is used to identify an audio mixing component that is dedicated for this combination. The mixer will be requested by the audio resource manager and assigned within the session. In terms of participation permission this assignment combines authentication as well as authorization within a session. This assignment also defines the identity of the session. Permission within voice loop participation are bound to a role and is either receive only (monitoring) or send and receive (talk). This binding allows defining the participation permission within a session over the identity. It does not indicate the current state of participation (e.g. which voice loop is switched on).

Any other endpoint connecting with the same user/role identity will be terminated to the same audio mixer assigned to this identity. With this, the VoCS contains multiplexing on identity level. This may be used for parallel communication and transmissions of the state within the system session to different endpoints of the same identity. It basically allows redundant communication endpoints, based on login information. The user of the system may choose to use parallelism for endpoints, using parallel logins on separated devices.

6.7 Voice loop mixer

To provide multi-party multi-conferencing functionality, voice loops are assigned to a mixer of the backend. It is the same process as before, but instead of requesting a mixer for a user/role identity, the audio resource manager requests a mixer with the same identity as the voice loop.

Actual participation state changes within voice loops translate into switching a mixer's input and output channels. A request for monitoring participation will connect the output of a voice loop mixer to the input of a user/role identity mixer. A request for talk participation will connect the media part of the endpoint from the media gateway to the input of the respective voice loop mixer. All these connection requests are orchestrated by the web gateway component, which enforces participation rights before translating user-based switching events to actual switching commands for the mixer components.

6.8 System state

Each switching event will change the participation state within the system session. The state is basically the combination of all input and output channels of all mixers as well as the media gateway. It is the logical wiring of transmission channels. This is a one-to-one translation of analogue audio wiring like it was done in switchboard operator times. The only difference is that the switchboard operators are the users themselves. Users are using a web client to switch their own communication channels within a software based micro-service environment, based on their communication needs. And the web gateway controls that each user only acts according to their permissions.

6.9 System scalability

This architecture allows different kinds of implementations. One implementation may define provision of a dedicated audio mixing component for each user/role as well as voice loop identity on instantiation. This way all possible combinations of audio mixers are always available. This is also how most traditional VoCS for mission control work. Another implementation could choose dynamic provision of identities and allow overbooking. This is also done in telecommunication environments. The reasoning is, that it is highlight unlikely that all possible combinations are used at once.

The architecture is highly scalable. System capabilities in terms of mixing components are related to the availability of mixing services. Mixing capabilities are bound to the registration of mixing services within the audio resource manager backend. To scale up, additional mixing services must be started. Endpoint scaling depends on the availability of web and media gateway components. Standard web concepts can be used (e.g. load balancing) to scale the amount of possible client applications.

6.10 System redundancy

Different concepts of redundancy can be used, based on the configuration level. Basic redundancy of endpoint duplications is already provided within the standard concept. With this, all client hardware issues can be eliminated. A user can login on two clients with the same user/role. All state switches done within one client are automatically replicated to the other. This is achieved due to only assigning one single mixer for a user/role identity to provide the mixing service, independent on how many clients login with this specific user/role identity.

Furthermore, the whole system can be used in a redundant manner. We only need to send the switching commands to independent system instances.

6.11 Summary

Redundancy and scalability become a factor of usage and configuration. They are not bound to the system or the systems architecture.

The whole concept abstracts any VoCS functionality to a logical level, which is hardware independent, highly scalable, and configurable and transfers system capabilities to configuration level.

7. Related Work

7.1 Dragon

Our idea would not be the first-time web interfaces are used in space flight. The GUI of the touch controls in the Dragon Module from SpaceX are implemented in HTML, CSS and JavaScript and run in a Chromium browser environment. This was confirmed by the SpaceX software team during an "ask me anything" (AMA) [1].

SpaceX's Dragon 2 spacecraft flies fully autonomous. The backend, which executes the flight navigation, is written in C and C++ [1]. The three touch screens in the cockpit mainly contain system information for the crew, but can also be used to fly the spacecraft. This controls were used in the SpaceX Demo-2 flight by astronaut Douglas Hurley [18]. Both crew members have a flight touch console in front of them. One console can work as a backup should the other misbehave. The touch controls in themselves are a backup system, should the flight computer misbehave.

7.2 WebRTC

Web Realtime Communication (WebRTC) [2] is part of the modern web stack and is defined by the World Wide Web Consortium (W3C). It is a standardized communication protocol for web browsers, mobile platforms, and Internet of Thinks (IoT) devices. The API is a VoIP technology and can be used to create audio and video communication channels. WebRTC is currently supported by the web browsers Chrome, Firefox, Edge, and Opera. There are native implementations for Android and iOS. With the sources of the Chromium or Android project native use under Linux is possible.

WebRTC does not include a predefined signaling protocol.

WebRTC as a protocol is intended to enable connections between private networks. Endpoints of the connection can have a private IP address, but it is expected that communication is set up without any knowledge of the remote station. Both endpoints of a connection are located within private networks. They identify their external IP addresses themselves over the protocols STUN [8] and Traversal using Relays around NAT (TURN). For this they need a reachable remote destination (STUN or TRUN server). This enables a connection between public addresses of WebRTC endpoints.

As soon as a WebRTC endpoint gathered its connection candidates, it will signal these candidates over an undefined communication protocol (e.g. the signaling protocol) to a remote party. The remote answers with its own candidates. WebRTC automatically sets up the connection by testing all received candidates for actual connection capabilities. This test is done using a DTLS [13] based connection setup. Within DTLS, the identity of a media streams is validated using a certificate fingerprint. This fingerprint was previously transmitted over the signaling protocol.

8. Conclusions

In our paper we presented a redesign of VoCS to integrate modern communication patterns into VoCS. The VoCS we present is not singularly focused on mission control rooms. It rather supports a wide variety of use-cases without compromising the mission control room use-case: Our system provides all the functionalities of current VoCS but is equally applicable outside the mission control room. It supports remote access and integration of remote parties without any extra costs.

The system we described is built with open standardized web technologies and protocols including WebRTC. It allows parallel participation within multiple communication channels over VoIP. Our system embraces redundancy, is scalable and highly configurable.

We also described, how the system supports the integration of browser-based voice clients, which are low cost, easy to customize and usable without extra client configuration. As the system only has lose coupling between server environment and clients, it supports the integration of practically every type of client or client setup. Only requirement is the integration of the WebRTC stack and the usage of a compatible signaling protocol.

9. Future Work

The implementation of our system is still a work in progress. The theory is sound, but we have not completely proven our system in a production environment. The primary focus for the near future is to test our system inside the control room. Beyond this point there are several components and features we develop further.

One point would be to find a PPT solution for browser-based voice clients, where the browser is not in focus. We have several ideas how to address this, but so far, no time to try it out. One idea would be to always use headsets with hardware PPT buttons, that open and close the microphone directly on the headphone. Alternatively, the usage of "Mute" instead of PTT would solve the issues. But "Mute" would be a different type of interaction, that should be analyzed over user studies in coordination with operations.

We have redesigned the system itself, but we think it is also time to rethink the VoCS client interface. It is rather easy to change the functionality and the look and feel of web applications. They are great platforms to simply try thinks out and a great opportunity to reinvent the VoCS GUI. While working on openvocs we came up with several ideas. Not only for the GUI but also for other client setups including e.g. smart watches. All these ideas need to be formalized, implemented and thoroughly tested in user studies.

We are also looking into the possibilities of integrating speech to text logging and video or text messages into our system. Our systems architecture could easily allow the inclusion of these alternative input and media channels.

References

[1] J. Dexter, J. Sulkin, W. Shimata, J. Dietrick, S. Hnaide and M. Monson, We are the SpaceX software team, ask us anything!, SpaceX, 06 June 2020, https://www.reddit.com/r/spacex/comments/gxb7j1/we_are_the_spacex_software_team_ask_us_anything/, (accessed 13.04.21).

[2] C. Jennings, H. Boström , J.-I. Bruaroey, A. Bergkvist, D. C. Burnett, A. Narayanan, B. Aboba and T. Brandstetter, WebRTC 1.0: Real-Time Communication Between Browsers, W3C Recommendation, 26 January 2021, https://www.w3.org/TR/2021/REC-webrtc-20210126/, (accessed 13.04.21).

[3] E. S. Patterson, J. Watts-Perotti and D. D. Woods, Voice Loops as Coordination Aids in Space Shuttle Mission Control, Computer Supported Cooperative Work (CSCW), vol. 8, pp. 353--371, 1999.

[4] Mozilla and individual contributors, MDN Web Docs: Touch events, 19 Feburary 2021, https://developer.mozilla.org/en-US/docs/Web/API/Touch_events#touch, (accessed 13.04.21).

[5] S. Agoston, J. Hollyer, M. Reynolds, B. Jones, S. Graham and T. Mielczarek, Gamepad, W3C Editor's Draft, 08 April 2021, https://w3c.github.io/gamepad/, (accessed 13.04.21).

[6] M. Töpfer, A. Sonnenberg and R. A. Kozlowski, OpenSource based Voice Communication for Mission Control, AIAA, SpaceOps 2016 Conferences, South Korea, 2016.

[7] A. Raake, Speech Quality of VoIP: Assessment and Prediction, UK-Chichester: John Wiley and Sons, 2006.

[8] M. Petit-Huguenin, G. Salgueiro, J. Rosenberg, D. Wing, R. Mahy und P. Matthews, Session Traversal Utilities for NAT (STUN), RFC 8489, IETF, 2020.

[9] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, RFC8446, IETF, 2018.

[10] I. Fette und A. Melnikov, The WebSocket Protocol, RFC 6455, IETF, 2011.

[11] T. Bray, The JavaScript Object Notation (JSON) Data Interchange Format, RFC 8259, IETF, 2017.

[12] A. Keranen, C. Holmberg und J. Rosenberg, Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal, RFC 8445, IETF, 2018.

[13] E. Rescorla und N. Modadugu, Datagram Transport Layer Security Version 1.2, RFC 6347, IETF, 2012.

[14] H. Schulzrinne, S. Casner, R. Frederick und V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, RFC 3550, IEFT, 2003.

[15] M. Baugher, D. McGrew, M. Naslund, E. Carrara und K. Norrman, The Secure Real-time Transport Protocol (SRTP), RFC 3711, IETF, 2004.

[16] D. McGrew und E. Rescorla, Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP), RFC 5764, IETF, 2010.

[17] J. Lewis and M. Fowler, Microservices, 25 March 2014, https://martinfowler.com/articles/microservices.html, (accessed 13.04.21).

[18] SpaceX, Crew Demo-2 | Coast, 01 June 2020, https://youtu.be/z9uAN6YNkP0?t=4201, (accessed 13.04.21).