ICE Technical Operations

Product guide for prerelease

Document version 1841, produced on Friday, September 06, 2024.

main 90adc8bf40040649230176bbdd465f6261a2d8e0

Contents

List of Figures

List of Tables

1 Document History

Publication Date	Product Release	Notes
May 29, 2024	3.5.1	No updates.
April 15, 2024	3.5.0	Added supported codecs for Radio Gateway interoperability.
September 20, 2023	3.4.0	Updates for releases 3.3.0 and 3.4.0.
July 24, 2023	3.3.0	Added MELPe to available codecs.
December 1, 2022	3.2.0	New release.
September 26, 2022	3.1.2	No updates.
August 24, 2022	3.1.1	Replaced the term ‘engagebridge’ with the term ‘patch’ in most instances.
June 1, 2022	3.1.1	Updated ‘Ports used by ICE Telephony’ section with new port numbers. Updated ‘Audio Bridging’ section to specify that patching requires ICE Rallypoint to be enabled, and that patching multicast channels is not supported.
March 15, 2022	3.1.0	Document created.

2 Introduction

This document describes how Instant Connect Enterprise (“ICE”) works and how elements of the system interoperate: what system components comprise ICE, what their general purpose is, how they communicate with one another, and how they can be deployed inside your organization. This document does not describe how to use or administer these components. Refer instead to the ICE Server Administration Guide as well as the ICE Desktop and ICE Mobile user guides for this information.

This document contains four primary sections:

- **Deployment architecture:** A discussion of special considerations related to Kubernetes.
- **ICE Server:** How it works, and how clients interact with it.
- **ICE media engine:** A discussion of the set of software libraries and technologies used “under

the hood" throughout Instant Connect that are responsible for the conveyance of audio between users.

- **Ancillary server components:** ICE Telephony, ICE Rallypoint, ICE Static Reflector and ICE Patch Server that operate independently of ICE Server and which can be deployed outside of Kubernetes or in multiple places in the network.

2.1 ICE system components

The Instant Connect Enterprise system is comprised of the following components. Each is described in greater detail later in this document.

Table 2: Instant Connect Enterprise system components

Component	Description
ICE Server	A Kuberenetes-deployed server system used to manage and provision the ICE system.
ICE Desktop	Cross-platform desktop application that runs on Windows, Linux and macOS offering all the features of the ICE Mobile client, plus the ability to configure and administer patches, reflections and the ICE Server.
ICE Desktop for Web	A browser-based version of ICE Desktop that provides a nearly identicial user interface and featureset as the macOS, Windows or Linux-native ICE Desktop. ICE Desktop for Web is a pure web applicaiton and does not require the installation of any browser plugins or other special software on the client device.
ICE Mobile	Cross-platform mobile application for Android and iOS supporting push-to-talk, public and private telephone communications, instant replay, and user presence/location.
ICE Telephony	Connects the ICE voice platform to a telephone network. Enables ICE Mobile and ICE Desktop users to make and receive phone calls from the PSTN and/or a corporate PBX.
ICE Rallypoint	A traffic "packet forwarder" linking the communications of multiple users across disparate networks. Rallypoints can be meshed together for scale, resiliency, and network efficiency.
ICE Patch Server	Bridges audio traffic between channels, allowing a talker on one channel to be heard by participants of other channels.

Component	Description
ICE Static Reflector	Mirrors traffic from a unicast channel onto a multicast network, useful when providing interoperability with other systems (like donor radios and IP desk phones) that communicate using RTP over IGMP multicast.
ICE Archiver	Enables an administrator to mark certain channels in the system for recording. The ICE Archiver "listens" to these channels, records talk-bursts heard on them, and makes those recordings available for future review.

2.2 Third party Kubernetes components

While not authored or maintained by Instant Connect, these components may be installed, by default, into the Kubernetes cluster when installing ICE. They are primarily intended as a convenience for customers new to Kubernetes or building a cluster specifically for ICE.

When deploying ICE onto a Kubernetes cluster that is being shared with other applications, it's likely the cluster will already have these (or equivalent) tools installed. Administrators are free—and encouraged—to use their own monitoring and analysis tools whenever appropriate. Consult the Administration Guide for details about omitting these components during installation or utilizing equivalent tools in their place.

Table 3: Third party Kubernetes components

Component	Description
Nginx-Ingress Controller	*Required*. Routes traffic received by the cluster to the service that it's intended for. Every cluster running ICE Server requires an ingress controller of some kind to forward traffic to ICE, but administrators are not required to use this specific ingress—others are available on the market—nor are they required to use the default configuration that ships with ICE.
Prometheus / Loki / Grafana	*Optional*. An open-source analytics and monitoring platform useful for searching and viewing log messages generated by application components running inside Kubernetes. It also provides data visualization and dashboards of system health and activity (like active network traffic, cluster node memory, CPU utilization, etc).

Component	Description
Swagger	*Optional.* An open-source web application providing technical documentation of the ICE REST web services. This tool is primarily useful for software developers interested in building custom integrations with Instant Connect.

2.3 Architectural design goals

Instant Connect Enterprise has been written from the ground-up with these goals in mind:

2.3.1 Scalable, distributed media processing

Unlike most voice collaboration platforms, virtually all media processing performed by Instant Connect Enterprise is handled at the edge of the network rather than inside a centralized server. Audio mixing, *vocoding*, and encryption are processed by each user's client device. The ICE Rallypoint, the component whose function is closest to a central media server, behaves like a network switch forwarding audio packets to clients who expect to receive them, without regard for their payloads.

This architecture has several benefits:

1. Enables a "tactical" mode of operation in which ICE clients can communicate with one another without need for a provisioning server. In environments where IGMP multicast is available between users, then not even a Rallypoint is needed.
2. In environments employing an ICE Server, users can still communicate even when the server connection is lost. Further, when a channel is configured to use a Rallypoint and the connection to that Rallypoint is lost, the channel will failover to use multicast.

3. It makes large scale environments possible without complex or excessive server hardware.

2.3.2 Network simplicity

Instant Connect utilizes "Internet friendly" protocols throughout its ecosystem. Clients connect to the ICE Server using HTTPS and to Rallypoints using SSL. Thus, most system operators who need to provide access to users outside of their firewall only need to open three ports: 443 for HTTPS, 7443 for Rallypoints and 8443 for Rallypoints used by ICE Desktop for Web.

Telephony functions that are available in ICE Desktop and ICE Mobile are performed entirely through these interfaces—there is no need for complex firewall rules, SIP session border controllers, STUN/-TURN servers or similarly complicated network setups common to architectures that expose SIP or UDP traffic to users outside your firewall.

2.3.3 Platform ubiquity and longevity

Instant Connect is built upon modern, leading technology frameworks: platforms that we believe will become de-facto standards in the years to come, if they're not already. Doing so allows our software to run on virtually any hardware or operating system:

- Docker and Kubernetes make cloud-scale applications a reality while providing a layer of abstraction between ICE Server and the operating system and hardware it's running on. Any compute platform capable of running Docker/Kubernetes can run ICE.
- Electron, React and Redux offer a modern application framework for web and cross-platform desktop applications. Used by a rapidly-growing number of popular applications.
- Dart and Flutter are the new kids on the block for cross-platform mobile development. A single codebase that can be shared between iOS and Android means that ICE has the same feature set and user experience across all devices. And a bug fixed on one platform is a bug fixed on both!

2.3.4 Flexibility in scale

Instant Connect's ability to serve large-scale deployments is accomplished through the combination of a distributed media architecture; a message-oriented provisioning server that is not dependent on transactions and locks; and Kubernetes—an orchestration platform to grow and shrink the system in response to load. However, ICE was not designed exclusively for cloud-scale environments: A single server Kubernetes system can be deployed for small environments at little cost. And when using tactical mode over an IGMP multicast network, no server components whatsoever are needed.

2.4 Simplified system diagram

The diagram below illustrates how key components interconnect in a single-cluster deployment of Instant Connect Enterprise.

ICE Mobile and ICE Desktop clients can be deployed on any network with reachability to the ICE Server system via ports 443 (HTTPS) and 7443 (SSL). When those clients are able to communicate with one another over IGMP multicast, then they will be able to communicate in *failover mode* if their link to the

Rallypoint fails. Note that ports 443 and 7443 are only defaults; the actual ports used for provisioning and voice traffic can be changed to any value by modifying the cluster's ingress controller (not illustrated).

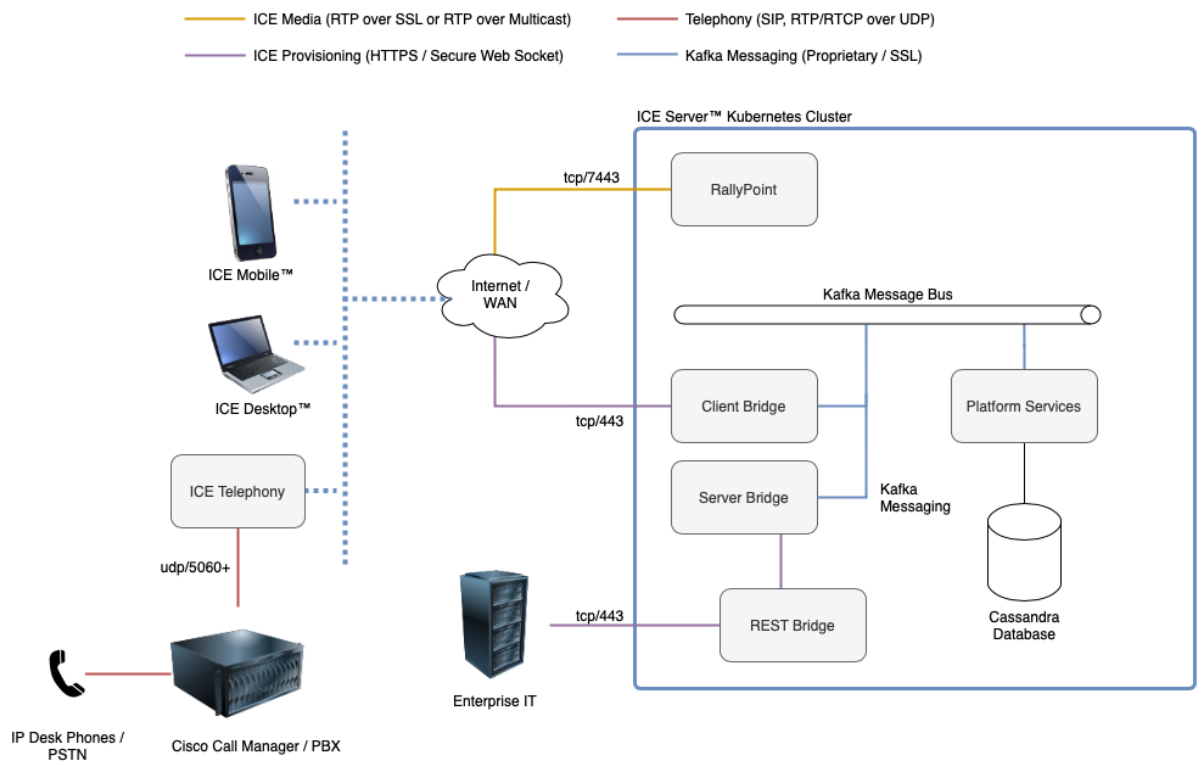

Figure 1: Simplified, single-node deployment diagram

The ICE Telephony component is deployed as a Docker container on a separate server outside the Kubernetes cluster. This host must have reachability back to the ICE Server cluster as well as unfettered voice access to the Call Manager which it is bridging. The actual protocols and ports used in this SIP/UDP link are negotiated at registration and call time. External IT systems which intend to interact with ICE Server's RESTful web services API need to be able to reach port 443 on the server.

3 ICE OS

ICE Server is packaged and distributed as a *Helm chart* referencing *Docker containers* that are intended for deployment into a Kubernetes cluster. Building and maintaining a Kubernetes cluster can be a challenging task, especially for administrators not already familiar with Kubernetes and its related ecosystem of tools and utilities.

ICE OS refers to a Linux operating system running a pre-configured, single-node Kubernetes cluster and other system utilities and software required to run ICE Server. ICE OS was developed as a way to easily deploy ICE Server in environments where an existing Kubernetes cluster is not available. ICE OS was designed with these goals in mind:

- Easy to install as a virtual machine (or directly on bare metal) with no knowledge of Kubernetes.
- A highly-secure, "lights out" operating environment that does not require regular OS-level patches or system maintenance.
- An easy-to-use configuration wizard web application that enables an administrator to customize their installation without hand-editing configuration files or memorizing complex command-line tool syntax.
- Support for small to medium sized installations with up to several thousand simultaneously active users (depending on hardware).
- Single-site and geo-redundant (dual-site) deployments.

Customers may choose to install ICE Server using ICE OS or directly onto a Kubernetes cluster. See the "Kubernetes deployment models" section, below, for details.

3.1 LinuxKit

Linux is an open-source operating system kernel initially developed by Linus Torvalds. It serves as the foundation for numerous operating systems, collectively known as Linux distributions. Renowned for stability, security, and customization, Linux operates on a wide range of devices, from personal computers to servers and embedded systems. Its modular nature allows users to tailor the OS to their needs, replacing or modifying components as desired. Its collaborative development model encourages community contributions, fostering innovation and continuous improvement. Linux has become a cornerstone of modern computing, powering critical infrastructure, mobile devices, and user-friendly desktop environments worldwide.

LinuxKit is a lightweight, open-source toolkit that enables building and running secure, container-focused Linux systems. It empowers developers to create custom, minimal Linux distributions optimized for containerized applications. With its flexibility and simplicity, LinuxKit aids in constructing efficient, portable, and isolated environments for modern software deployment.

ICE OS is built using LinuxKit and contains only the system components and utilities required to run ICE Server. This implies that ICE OS is a Linux operating system, but not a standard Linux distribution like Ubuntu, CentOS or RedHat. Additionally, ICE OS is an "immutable" operating system like those used in physical appliances: It runs from a read-only file system and cannot be modified, patched, or updated by an administrator or malicious user.

ICE OS is distributed as an disk image (in `.iso` format) and is intended to run directly from this disk image. Unlike traditional operating systems, the disk image is not an operating system "installer." ICE OS is never copied from the disk image onto a writeable file system. Instead, the ICE OS disk image is intended to be installed physically or virtually as a DVD in the host system and paired with a read/write filesystem where ICE application data and system configuration will be written. Consult the server installation and administration product documentation for details.

3.2 Kubernetes

Software containerization encapsulates applications and their dependencies into isolated units, called containers. These containers provide a consistent and portable environment, ensuring that software runs reliably across different computing environments. Containerization simplifies deployment, scaling, and management of applications, fostering consistency and efficiency in software development and operations.

Kubernetes (often stylized as *k8s*), originally developed by Google, is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides tools for container deployment, load balancing, and self-healing, making it easier to manage complex microservices architectures. Kubernetes abstracts the underlying infrastructure, enabling seamless scaling and resource optimization while enhancing application reliability. It has become a cornerstone in modern cloud-native application development, allowing efficient utilization of resources and simplifying the management of containerized workloads at scale.

Many of the world's largest cloud-hosted applications use Kubernetes to achieve their scale and power. Kubernetes enables applications to achieve this scale by orchestrating their execution across a *cluster* of computers. A cluster refers to one or more physical computers (called *nodes*) working together to form a single, logical system. A cluster could be just a few servers, or tens of thousands of them.

Kubernetes is analogous to the operating system that ICE Server runs on. In much the same way that an application that runs on Linux, Windows or macOS utilizes components provided by that operating system, ICE Server depends on and interoperates with elements provided by the Kubernetes ecosystem.

3.3 Benefits of Kubernetes

Kubernetes offers numerous benefits for managing containerized applications. It automates deployment, scaling, and updates, optimizing resource utilization and ensuring high availability. Its self-healing capabilities automatically restart or replace failed containers, enhancing application reliability. Kubernetes abstracts underlying infrastructure, enabling portability across cloud providers. With

load balancing and auto-scaling, it accommodates varying workloads. It enhances security through isolation and role-based access controls. Overall, Kubernetes streamlines complex application orchestration, reducing operational complexities and accelerating development in cloud-native computing.

- **Portability:** Kubernetes eliminates the complexities of dealing with variations in underlying infrastructure, so the same Instant Connect Enterprise software can be run on different operating systems; either on "bare metal" or as virtual machines; on-prem, in the cloud, or across clouds. This capability allows Kubernetes to adapt to changes in your IT department's software and hardware infrastructure.
- **Resiliency:** Kubernetes manages the applications running within it to ensure that they are available to end-users. If an application or any of its components fail (for example, due to a hardware failure), Kubernetes automatically redeploys those components and ensures they remain available.
- **Scalability:** Applications like Instant Connect Enterprise that have been purposely designed and built to be managed by Kubernetes can scale dynamically in response to fluctuations in demand. This capability reduces the resource footprint of an application during normal usage while maintaining the ability to serve spikes in demand.

3.3.1 Drawbacks of Kubernetes

Kubernetes, while powerful, has drawbacks. Its complexity necessitates a significant learning curve and dedicated personnel for efficient deployment. Managing clusters demands substantial resources, both in terms of infrastructure and human expertise. The dynamic nature of microservices can lead to intricate networking issues and configuration complexities. Security concerns arise from misconfigurations, potentially exposing vulnerabilities. Additionally, upgrading and migrating applications can be challenging. Its diverse ecosystem might complicate tool selection. Inadequate monitoring and resource management might result in cost inefficiencies.

Despite these downsides, proper planning, training, and diligent management can help organizations navigate and mitigate these challenges while leveraging Kubernetes' benefits.

On-premises deployments can avoid these drawbacks by relying on ICE OS, which comes optimally prefigured for security and compatibility.

3.4 Kubernetes deployment models

Kubernetes is a powerful and flexible platform for executing highly scalable and distributed applications. This section describes various deployment models that can be successfully used with Instant

Connect Enterprise.

3.4.1 On-premises or cloud hosted

Instant Connect Enterprise can be deployed onto a Kubernetes cluster that is either physically located at a customer's site ("on-prem") or onto a Kubernetes cluster running in a public cloud, like Amazon Web Services or Microsoft Azure. Although the installation procedure is different, once installed, Instant Connect operates identically no matter the environment. The choice of which to utilize depends on the needs of the organization.

When deploying in a public cloud, we recommend utilizing a managed Kubernetes service (such as Amazon EKS, Google GKE or Azure AKS) in which Kubernetes is provided as a service and the administrator is freed of the burden of managing the control plane elements of the cluster. While not advisable, it's possible to construct your own cluster using raw cloud compute resources (like Amazon's EC2 instances). Most organizations will achieve best results utilizing a managed service where networking, ingress, and load balancing concerns have been addressed by the provider.

Some elements of your Instant Connect system will likely need to remain on-premises even when the ICE Server is hosted in a public cloud: ICE Telephony should be deployed on-prem, adjacent to your organization's call manager. Similarly, donor radios linked to ICE channels will require a Static Reflector deployed in the same multicast domain as the radio gateway (also on-prem).

3.4.2 Single-node cluster (ICE OS)

A single node cluster refers to a Kubernetes system comprised of a single physical server and is the only deployment model supported by ICE OS. While this effectively defeats the resiliency benefits provided by redundant hardware, it offers a low cost and simple deployment model for applications where large scale and maximum uptime are not required.

A single node cluster running on a 16 core, 2.0 GHz Intel Xeon-based server with 32GB RAM is sufficient for serving approximately 800 concurrently active users (producing a voice load of 19,200 calls per hour). Scale can be increased vertically by adding additional memory and CPU cores.

3.4.3 Multi-node cluster

A multi-node cluster consists of three or more physical servers assembled together into a single Kubernetes system. A three node cluster is capable of losing an entire physical node with little or no visible impact to end users—administrators can demonstrate this ability by literally pulling the power on a server while the system is in use.

> **Three or more nodes are required.** A two-node cluster cannot provide redundancy because of quorum requirements inherent to the Kubernetes platform.

When multiple nodes are present in the cluster, client connections can be received on any node; see the "Cluster Load Balancing" section (below) for a description of different techniques that can be used to route incoming traffic to different servers.

A three-node cluster utilizing the same hardware as described previously (16 cores, 32GB RAM) is sufficient for serving 2,500 concurrently active users. By adding additional nodes to the cluster, virtually unlimited scale is achievable.

> **Careful:** Simply using a three (or more) node cluster does not "auto-magically" produce a resilient or highly-scalable system. To achieve these goals special configuration of Instant Connect is required to assure system components are sufficiently distributed across nodes. Consult the product guide for details.

3.4.4 Loss of a data center

A multi-node Kubernetes cluster provides a very high level of resiliency, even when suffering a hardware failure. But no matter how many redundant nodes are added to the cluster, if the data center hosting it is lost then the software running on it will fail. In specialized environments it's important to architect a solution resilient to the loss of an entire data center.

There are two primary mechanisms for achieving this extreme level of resiliency. A more detailed description of these approaches is described in the section "Geographic redundancy."

3.4.4.1 Distribute the nodes of a single cluster between data centers This approach solves the problem at the Kubernetes-level. That is, Instant Connect (and other Kubernetes applications) are unaware that they are being distributed between different locations.

Nodes in a cluster expect to be able to communicate with one another using a high-speed, low-latency network link. Similarly, persistent volumes (disks) will need to be replicated across this link, too. Achieving this level of network connectivity between physical sites may be difficult or impossible.

For the same quorum-related reason that a cluster cannot have only two nodes, a single geo-redundant cluster must be distributed across three (or more) data centers. Some solutions achieve this by running two primary data centers and hosting a third, tie-breaking element in a small "closet."

3.4.4.2 Create two clusters and configure Instant Connect to replicate data between them

This is Instant Connect's preferred approach to providing data center level resiliency.

This approach, which is custom to Instant Connect, uses two independent Kubernetes clusters—be they single-node or multi-node—which are installed in different locations. Instant Connect runs on both clusters and replicates data and signaling messages between them to make the system appear as a single, logical entity. Because this solution involves application-layer data replication, network bandwidth and latency requirements are substantially relaxed; the two clusters can function (albeit in a "split brain" mode) for hours, days or even weeks without permanent impairment.

3.4.5 Choosing a deployment model

The two tables below illustrate the benefits and drawbacks of choosing a deployment model and cluster architecture.

Table 4: Hosting model benefits and drawbacks

Deployment	Benefits	Drawbacks
Cloud hosted	Cost and complexity of building, administrating and maintaining a Kubernetes cluster is delegated to the cloud provider. Installation can be completed in minutes.	Assumes ICE clients are able to reach the Internet. Some IT departments may perceive solution as more expensive than utilizing their own hardware.
On premises	System can be deployed on remote or highly secured ("air gapped") networks that are not connected to the Internet.	Substantially increases administrative and maintenance burden on the administrator. Administrator becomes responsible for creating the Kubernetes cluster and assuring those machines comply with the organization's security and IA policies.

Table 5: Cluster type benefits and drawbacks

Cluster	Benefits	Drawbacks
Single node cluster (ICE OS), single site	Simplest, least resource-intensive deployment model. Ideal for small scale systems, labs, demo environments and mobile systems (i.e., vehicle mounted).	Provides no hardware resiliency. Should the physical machine running ICE fail, the system will suffer a full outage.
Multi node cluster, single site	Allows organizations to increase system scale over time by adding nodes to the cluster. Provides full resiliency between nodes in the cluster: Any node can fail with little or no user-visible impact to operation.	Additional complexity and compute resources: A minimum of three physical servers are required to provide hardware level resiliency.
Two clusters, geographic redundancy	Provides extra resiliency when an entire data center goes offline. ICE clients automatically reconnect to the surviving data center and continue operating normally.	Most complex and resource intensive deployment; requires double the compute resources of a single-cluster setup and demands a high-speed link between clusters. Not all functions of Instant Connect are fully resilient, even when using this deployment model.

3.4.6 IP ports used by the system

Instant Connect Enterprise uses two primary ports for communication between clients (ICE Desktop and ICE Mobile) and server-side components (RallyPoints and ICE Server):

Port	Use
443/tcp	Establishes a secure web socket to the ICE Server system for management and provisioning functions. Clients send and receive messages over this link to receiving information about channel configurations, incoming telephone calls, person presence status, location, etc. This port is also used by third part IT systems that interact with ICE Server via REST web services and web hooks.
7443/tcp	Establishes a secure SSL connection to the RallyPoint (or, potentially, many RallyPoints depending on deployment architecture) used to convey audio traffic between users on a channel that is configured to use the RallyPoint.

> Specific port numbers—443 and 7443—are defaults; system administrators may configure the system to utilize alternate port numbers as appropriate.

3.4.6.1 Ports used by ICE Telephony The ICE Telephony component behaves like a desktop or mobile client in its communication with the rest of the ICE server infrastructure: That is, it uses port 443 and 7443 for communication with the ICE Server and RallyPoints. However, for communication with a call manager or PBX, ICE Telephony utilizes the session initiation protocol (SIP) over the following ports:

Port	Use
5060/udp/tcp	SIP system signaling between ICE Telephony and the call manager or PBX.
5061/tcp	If TLS/SRTP is enabled.
5070-5270/udp	SIP signaling for registered dial numbers. Each dial number requires two ports; ICE Telephony supports up to 100 registered dial numbers. Not all ports in this range will be used unless a system is configured with 100 registered dial numbers.
16384-20480/udp	Realtime transport protocol (RTP) traffic conveying call audio between ICE Telephony and the call manager or PBX. Not all ports in this range will be used simultaneously; per the SIP standard, actual ports are negotiated during call setup and are used only for the duration of a call.

3.4.6.2 Ports used by external static reflectors and patch servers Both static reflectors and patch servers deployed outside the cluster behave like ICE Desktop and ICE Mobile endpoint devices:

They establish SSL (7443/tcp) connections to one or more RallyPoints (depending on the configuration of the system), plus a single secure web socket (443/tcp) to the ICE Server in order to receive provisioning messages.

3.5 Cluster load balancing

The Kubernetes cluster is designed to accept traffic received by any node in the cluster. Even if an instance of the service for which the traffic is intended is not running on that node, the Kubernetes ingress controller will automatically forward the connection to a node capable of servicing the request.

System administrators have several options to choose from when deciding how to route traffic into their ICE Server. Consider a hypothetical cluster with three nodes, `n1.cluster.com`, `n2.cluster.com` and `n3.cluster.com`:

- Chose any one physical node in the cluster, say `n1.cluster.com` and route all traffic to it by using its hostname as the server address when logging in. Doing so lets this node function, implicitly, as the cluster's load balancer. This provides the simplest configuration but at the cost of introducing a single point of failure: Should `n1.cluster.com` fail, the ICE Server software will remain running but clients won't be able to reach it on one of the remaining, live nodes.
- Configure each physical node in your Kubernetes cluster as an "ingress address" on the "Organization" settings screen in ICE Desktop. In this example, you'd provision `n1.cluster.com`, `n2.cluster.com`, `n3.cluster.com` as ingress addresses. When logging into ICE Server, users may enter any one of those hostnames as the server address. Should that node fail, clients will automatically try to reconnect to the other nodes. This configuration solves the single-point-of-failure problem described above, but may not evenly distribute ingress load across the cluster (nor does it provide a way for an administrator to forcibly migrate users to or from a specific node).
- Front the cluster with an external load balancer. This configuration can be as simple or as complex as your needs demand—even offering global routing of traffic to geo-redundant clusters distributed around the world.

3.6 Geographic redundancy

Recall that Kubernetes inherently provides a great deal of resiliency across the nodes of its cluster. A properly designed cluster can lose one or more nodes without impacting the operation of the software running on it. However, achieving this same level of resiliency to address the possibility of an entire data center going offline is notably more complicated. We refer to this level of resiliency as "geographic redundancy."

A geo-redundant system must be designed to handle a loss of all nodes in either data center, meaning Kubernetes control plane elements must have sufficient replication and distribution between the sites, and each site should be sized to accommodate the entire load of users in the event that the other site should fail. If the size of deployment requires, say, four nodes to serve all users, then a geo-redundant system would require eight total nodes: four at each site.

As previously described, there are two approaches for creating geographic redundancy in Kubernetes:

3.6.1 Single cluster that spans two physical data centers

The first option is to create a single Kubernetes cluster, but distribute the physical nodes of that cluster across data centers.

In this configuration, a loss of a data center appears to Kubernetes as a loss of redundant nodes in the cluster. Kubernetes has no understanding of “data centers” or the physical distance between nodes in the cluster. It sees a loss of a data center no differently than one or more machines in a single rack failing. This is, essentially, how many cloud providers offer “availability zones” and other such cross-site resiliency assurances. Of course, the cluster ingress design (as discussed in the previous section) must allow for traffic entering both data centers.

The challenge with this approach is that it places a substantial burden on network engineers: Because this architecture treats all nodes as belonging to the same cluster, there is nothing to optimize or recognize LAN vs WAN network traffic or storage. Each node is expected to operate as through it was on the same switch as every other node, connected to the same underlying NAS, SAN or equivalent storage medium. This implies that the data centers will need to be linked with a high-bandwidth, low latency network connection between them. The exact requirements of this link depend entirely on the cluster’s load, CNI plugin, storage method, etc.

3.6.2 Replicate ICE data between independent clusters

The second option is to build two or more entirely separate Kubernetes clusters and configure Instant Connect to replicate data and messages between them (forming a single, logical instance of ICE). This is the mechanism officially supported by Instant Connect (although Instant Connect only supports a two-cluster deployment).

Clusters communicate with one another through a virtual private network (VPN) using WireGaurd. Setup and configuration details are described in the geo redundant cluster setup guide.

> Instant Connect is unaware that it is operating across clusters. In much the same way that ICE does not know, monitor, or care how many physical nodes its software components may be orchestrated across, it has no knowledge of this dual-cluster "plumbing."

In this design, network requirements are much more lax than operating a single cluster across sites. Instead, ICE requires two forms of data to be replicated between clusters:

- Kafka messages produced on one site must be relayed to the other site. For example, Bill is connected to Site A and tries to call Brenda connected to Site B. The replication of Kafka messages allows the real-time signaling of the call flow to reach both parties connected to different sites.
- Data persisted in the Cassandra database must be visible to users on others sites. Cassandra supports multi-node, multi-data center replication out of the box.

These tools—Kafka and Cassandra—are designed to be replicated over lower-bandwidth, higher latency network links and can easily adapt to intermittent and short-term partitions (referred to, broadly, as "eventual consistency"). Low bandwidth and/or high latency network links between sites will produce a degraded user experience but not a system outage.

3.6.3 Cluster partitioning

When geo-redundant clusters become partitioned from one another—that is, unable to communicate with one another over the network—both sites will continue to operate "normally," albeit in a split-brain configuration.

Recall that the ICE Server is unaware that it is even executing across different clusters. A partitioning event simply means each site will see the other as idle (generating no messages or configuration changes). End users connected to either site will be unaware that anything has happened on the other. Configuration changes or signaling initiated on one site will not propagate to the other and this will have the following observable side effects:

3.6.3.1 A private telephone call made from a user on Site A to a user connected to Site B will not go through. The calling party will hear ring-back but the callee will not observe an incoming call. As soon as the sites reconnect, these signaling messages will be delivered. During a brief partition, this may manifest itself in a delay between when the call is placed and when the receiving party rings. If the call attempt is made and then aborted while the sites remain partitioned, the callee may experience a brief "ring" / "call declined" state on their device as both enqueued messages are eventually delivered, back-to-back.

3.6.3.2 Public telephone calls placed or received may be delayed or fail. The ICE Telephony component—Instant Connect's bridge to the telephone network—attaches itself to one cluster or the other. Thus, all telephone calls to the PSTN are treated as terminating on one site. Calls placed or received from the site opposite that which ICE Telephony is attached will fail in the same way described for private calls.

At this time, ICE does not support geo-redundant telephony integration.

3.6.3.3 An intercom established between users of different sites will not immediately appear on all dashboards. Until the sites reestablish their link, the message indicating a user has been added to an intercom will not be delivered to users connected to another site. The initiating user will be unaware that the other users' dashboards have not been updated with the intercom channel.

3.6.3.4 Configuration changes made on one site will not appear to users of the other. Any changes to system configuration (including channels or users that are added, removed or modified) that are made on one site will not be visible to users on the other. When modifying a channel configuration, this can result in different users utilizing different configuration values—Rallypoint, encryption mode, etc—and this may result in a partitioning of audio traffic even when the audio's bearer network is not itself partitioned.

3.6.3.5 Users may see inconsistent presence and location information. Online/offline status indications as well as users' GPS location visible on the map will diverge between users connected to different sites. For example, consider a scenario in which Bill (connected to Site A) shares a channel with Brenda (connected to Site B). At some point, the sites become partitioned and while they are partitioned, Bill logs out of ICE. Brenda will continue to see Bill's status as "online" since the message that Bill logged out cannot be delivered to Brenda's site.

3.6.3.6 What happens to audio? It depends entirely on how a channel has been configured and the nature of the network used to convey its audio.

- In the case of multicast, users connected to different clusters will still be able to hear one another provided the multicast network used for voice traffic has not also become partitioned somehow.
- In the case of unicast (Rallypoint) traffic, the behavior will depend on the design of the Rallypoint mesh. In the simplest case, a Rallypoint is present in both clusters and together they are meshed. If a channel is configured to use this cluster-to-cluster Rallypoint mesh then channel audio will likely be partitioned in the same way as provisioning messages: Users of Site A will

be able to hear one another and users of Site B will be able to hear one another both a Site A user will not hear a Site B user and vice versa.

> **Careful:** There is not a relationship between the ICE Server and Rallypoint utilized by an end user's device. A client device could be connected to Site A for ICE Server provisioning messages and also to Site B for Rallypoint traffic.

3.6.4 Problems associated with long-lasting partitions

Instant Connect's geo-redundant clustering mechanism is not intended to operate in a partitioned environment for extended periods of time. In addition to the previously discussed anomalies, the longer two sites are partitioned, the greater the chance for data inconsistencies to emerge, namely:

- Kafka messages are discarded after approximately two hours. Messages used to signal a user's client to take some action (for example, a message to display an intercom channel on the dashboard) will simply never be delivered if sites are partitioned for a long period of time. Users will need to log out and log back in to "resync" these state inconsistencies.
- Cassandra nodes may need to be repaired by invoking the `nodetool repair` command on them. Instant Connect Enterprise automatically performs this step periodically but system administrators may wish to initiate this immediately following a long partition.

- Elasticsearch results (those displayed in people/channel search operations) may produce different results and/or incorrect online/offline status indications.

4 ICE Server

The ICE server is responsible for user and channel management within an Instant Connect Enterprise system. Clients connect to the ICE Server to be told of which channels they're able to participate on; be notified (signaled) of incoming private and PSTN telephone calls; be made aware of other users of the system; and learn the presence and location of teammates.

All application-layer configuration of the ICE Server is performed through the ICE Desktop client application. There is no command line interface or special browser-based portal for administering the system. Of course, only users with the "Administrator" role will see and have access to these configurations (within ICE Desktop, log into ICE Server then click the gear icon to view settings and configuration).

As described in greater detail below, ICE Server is a not a traditional REST or SOAP-based services oriented architecture (*SOA*). ICE Server is, rather, a message-oriented system that sends and receives

messages asynchronously to clients connected to it. This gives clients near-instant response times to changes in configuration, telephone signaling, etc.

4.1 Client connections to ICE Server

ICE Desktop and ICE Mobile clients connect to the ICE Server by establishing a WebSocket over which they exchange messages with the server. This design provides for a fully reactive system: Clients do not have to ask (poll) the server to know if someone might be trying to call them or if they were recently added to a channel. Instead, they are told of such changes asynchronously and react to them immediately. While the ICE Server is not a traditional REST-based system, it does provide a facade for REST-style external access—more on that later.

Most deployments are configured to use SSL/HTTPS on port 443 to create a secure (encrypted) web socket, although ports and protocols are ultimately configurable through changes to the cluster's ingress controller. No technical limitation prevents ICE to be used over an unsecured connection, doing so is highly discouraged as passwords and other personal information are easily captured by hackers and eavesdroppers.

4.1.1 Establishing the connection

To establish an initial connection, the end user must input the address of the ICE Server in the form of a URL, along with their username and password. When a non-standard port is being used (i.e., values other than 443 for HTTPS or port 80 for HTTP), then the port must be explicitly provided in the URL (`http://demo.instantconnectnow.com:9762`). When HTTPS on port 443 is in use, the user need not enter a scheme or port; just the hostname (`demo.instantconnectnow.com`).

The client initiates the login process by performing an HTTP `POST` to the server address at the path `/login` (i.e, `demo.instantconnectnow.com/login`. The username and clear-text password is transmitted in the body of the request in query string format, along with the following headers:

- `X-Client-DeviceID` — An ICE-proprietary hardware device ID that's used by the system to license the client.
- `User-Agent` — A string indicating the type of client (to indicate, for example, mobile or desktop).
- `X-Client-Version` — The software version of the connecting client.

If the user's credentials are accepted by ICE Server, the response to the `POST` request will contain an authorization token that, from this point on, represents the user's session identity. No further communication with the server involves the use of the user's username or password. The authorization token is a string of random alphanumeric characters. It does not encode any information about the

user or their password, and the token may be revoked at any time from within the “License” settings screen in ICE Desktop by clicking the “End Session” button next to the user’s name.

Using the authorization token provided in the response, the client then “upgrades” the connection to a WebSocket (the details of which are defined by RFC-6455). Once the WebSocket connection has been established, the ICE Server sends an initial message to the client containing information about the client’s session. This data includes (among other things) information about the API level of the sever, cryptographic keys and activation codes to license the device, the connection addresses the client may use when reconnecting, and some initial information about the connected user’s profile.

4.1.2 Determination of online/offline user presence

ICE Server uses its knowledge of which clients maintain a WebSocket with it to infer online/offline status and track license utilization. That is, as long as a user’s agent (ICE Desktop or ICE Mobile) is connected to the server, ICE will infer that the user is online and this presence will be broadcast to other users. To detect stale connections, ICE uses a ping/pong message (defined by the WebSocket standard) to quickly detect when the client has become detached from the server. Stale connections are closed; the impacted user’s presence state returns to “offline” and their session license is returned to the pool.

Be aware that ICE Server tracks only a single, global online/offline status value for each user. A user is considered online if they are connected to ICE Server from one or more devices.

Each WebSocket connection will consume a mobile or desktop license. A single user logged into multiple devices will appear to consume multiple licenses. Administrators can forcibly end a user’s session from the license settings screen in ICE Desktop.

4.1.3 Determination of a user’s location

ICE Server maintains a single GPS location for each user actively logged into the system. Each user’s location is reported by the client application they’re using. This location is shared with other users having at least one channel in common with the reporting user.

Having a single GPS location associated with a person can produce confusing results when a user is logged in to multiple devices, especially when those devices are not in the same location. ICE assumes that a user is currently in the location that was last reported by a device they are connecting from. Thus, if a user is connected through different clients each reporting different locations, other users on the system may see the user’s location jump back and forth between the location reported by one device and the other.

The ICE Mobile client uses Android and iOS' in-built location services to determine the user's spatial coordinates. The accuracy and algorithm used to determine location is specific to each device and operating system and might be as simple as the reporting the value returned from the device's GPS receiver, or as complicated as inferring location from IP address, nearby WiFi access points, etc.

ICE Desktop, which is designed for use on devices that typically do not have in-built GPS hardware, delegates to Google Maps Geolocation APIs to infer GPS coordinates. The location returned by this service is usually *close* but highly variable in its accuracy. ICE Desktop users can manually specify their location (or disable location reporting altogether) when working in environments where Google's results are inaccurate or unavailable.

4.1.4 Client reconnect behavior

Each time that a client connects to the ICE Server, the system notifies the client of alternate points of ingress that it may use to connect to the system the next time a connection needs to be made. Clients "save" this information to persistent storage so that once a client successfully connects to a system it can utilize these alternate sites or addresses even after logging out or quitting the application.

How the client goes about choosing which ingress address to connect to is configurable in the system's "High Availability" settings (on the "Organization" settings screen in ICE Desktop). In environments where no server ingress addresses have been configured, clients will always use the "Identity" reconnection strategy.

Table 8: Client reconnect strategies

Reconnection Strategy	Description
Preferred	The client will choose from the list of available ingress addresses in the order they appear (top-to-bottom) in the setting screen.
Nearest	The client will choose from the list of available ingress addresses in the order of their physical distance from the client (calculated using the location assigned to the ingress in the Organization settings screen, and the reported location of the client).
Identity	*Default.* The client will reconnect only to the server ingress address entered by the user in the login screen. All other ingress addresses specified in the Organization screen are ignored. This choice has the effect of disallowing clients to connect to any server address other than the one explicitly entered on the login screen.

Reconnection Strategy	Description
Random	The client will randomly choose from the list of available ingress addresses.

As an example, consider a system deployed in the United States with three ingress addresses: `west.demo.com`, `central.demo.com` and `east.demo.com`: When the connection strategy is set to "Preferred" clients will try to connect in the order: `west`, `central`, `east`, `west`, etc. When the strategy is "Nearest", a client in Chicago will connect to `central`, then `east`, then `west` whereas a client in Los Angeles will connect to `west` then `central`, then `east`. When the strategy is "Identity" all clients will ignore the configuration made in the "High Availability" section of "Organization" settings and connect only to the server the user entered in the login screen.

Clients connections to the ICE Server are "sticky." Clients do not try to reconnect to their original ingress or to a more ideal ingress once a successful connection is made. Once a client establishes a connection to an ICE Server, it maintains that connection until the user quits, logs out, or the connection is broken.

> **Careful**: Each client maintains both a single connection to the ICE Server, plus a connection to every Rallypoint needed by all active channels. Loss of ICE Server connectivity does not necessary imply loss of Rallypoint connectivity. Channels have their own Rallypoint reconnect and failover strategy (described in detail later in this document).

4.1.4.1 Establishing an initial connection

If the user is attempting to log into a system that has been previously connected to, the client will have "remembered" the alternate ingress addresses associated with that system and use them, if necessary, to re-establish a connection. Under certain circumstances this can result in connecting to an address other than the one entered explicitly on the login form. When establishing an initial connection, the client will try each alternate site only once before "giving up" and reporting an error.

Using the example from above, consider a scenario in which a user had logged out of, and is now logging back in to `central.demo.com`. On the login screen, they enter the server address: `central.demo.com`, but when they click the "Login" button their client, having remembered alternate ingress addresses, may choose a different ingress address to connect to.

Be aware that alternate ingress addresses are associated with the primary ("identity") address used to initially connect to the system. In the previous example, had the user re-entered a different server address on the login screen (say, `east.demo.com`) and that alternate site was unavailable, the client *would not* attempt to connect to `west` or `central`.

4.1.4.2 Detecting a connection interruption Recall that clients maintain two kinds of connections to the server environment:

- A web socket connection the ICE Server for management and provisioning functions. When this connection has failed, the user's avatar (visible in the top-right corner of the ICE Desktop and ICE Mobile applications) will appear drawn with an animated glowing yellow ring.
- An SSL connection to the RallyPoint (or, potentially, many SSL connections to many RallyPoints depending on deployment); used to convey audio traffic on channels configured to use a RallyPoint. When this connection has failed channels reliant on the connection will display a "Limited connectivity" message with a slash through the channel's cloud icon.

In the simplest case, clients are made aware of a connection interruption whenever their underlying operating system informs them that the socket has been closed.

However, it's possible for a loss of service to occur that does not cause the socket connection to be terminated at all, or terminated in a timely manner. To recover from such failures, both connections utilize a form of ping-pong messages that are sent back and forth between the client and server components to detect such outages. When a client sends a ping to the server and does not receive a timely "pong" message in response the client will presume the server/RallyPoint to be unreachable or out of service and attempt to reconnect to an alternate ICE Server ingress address or to an alternate RallyPoint. Similarly, requests made of the ICE Server that are not fulfilled also trigger the client to assume an outage has occurred.

The amount of time required by the client to detect a connection interruption depends on the type of failure and the component whose link has failed. Failures in which the client's operating system detects a socket closure typically occur quite quickly—within a matter of seconds (if not milliseconds). Failures in which the socket remains open but the server becomes unresponsive typically take 30 seconds to a minute to detect. In the case of an oversubscribed server component whose ping-pong messages may be received intermittently can result in marginal service or delays in detecting an outage.

4.1.4.3 Reconnecting after a connection interruption If a client becomes detached from the ICE Server for any unexpected reason (expected reasons include quitting the app or logging out), the client will begin attempting to reestablish a connection to the ICE Server. It will choose the next appropriate available ingress address using the configured strategy; if the connection attempt fails for any reason, the next address will be tried and so on. The client will continue this process (using a backoff timer to rate-limit requests) until a connection is established or a terminal connection response is received by the client.

There are two terminal reconnect HTTP status codes that, when received, stop the client from trying to reconnect any more:

- `HTTP 401 - Not Authorized` - Implies the user's session has been terminated by the administrator or system. The user must log in with their username/password credential to regain access to the system. ICE clients *never* store the user's password on the device.
- `HTTP 601 - No License Available` - Implies that all licenses are currently in use.
- `HTTP 514 - Session Expired` - The user's session has timed out or been terminated by an administrator; the user must reconnect using their username and password credentials.

4.2 Special considerations for ICE Desktop for Web

ICE Desktop for Web is a pure web application requiring no special software or browser plugins to operate. It provides nearly the identical feature set of the native ICE Desktop application.

4.2.1 Application limitations

When using ICE Desktop in a browser, some features are unavailable or behave differently than when using the native ICE Desktop application.

- **Hot keys** (keyboard shortcuts): For security reasons, a web page cannot monitor key-press events when the user does not have the window in focus. Because of this limitation, the hotkeys feature is unavailable in the web version.
- **Audio steering**: Users will not be able to direct audio to different speakers on their computer, or choose the microphone device to use. These selections are made through the browser itself (that is, a user will choose the speaker and microphone device in their browser's settings; the process of doing so is specific to each browser).
- **IGMP multicast channel connections**: Web browsers do not support transmitting or receiving IGMP multicast traffic. Channels not configured to use a Rallypoint will not work in a browser (and will display a "Channel not available on web" message). Furthermore, configuration options related to choosing a network device for multicast traffic are not available in the web application.
- **Gravatars**: Due to cross-origin security requirements and a limitation in the library used to render Gravatars, the web client will only display a user's initials as their avatar, even when a Gravatar is available.
- **MELPe Codec**: Channels configured to use this codec will not function on the web and will appear with the "Channel not available on web" message. This feature may be added in a future release.
- **FIPS 140-2** (Wolf SSL): All SSL connections utilize the browser's internal cryptography engine for security. Customers interested in FIPS-compliant security should utilize a FIPS-compliant

browser. Note that the "Build" dialog will always display "FIPS 140-2 not validated" in the web client, irrespective of the FIPS compliance of the browser.

- **App sounds**: Some browsers will squelch audio (like ring tones) that are produced by a web application when the browser window is minimized or in the background.

4.2.2 Browser limitations

When using ICE Desktop for the web, be aware of limitations and restrictions imposed by the browser.

4.2.2.1 Use of older browsers ICE Desktop for the Web uses the WebAssembly ("WASM") technology for its media engine and relies on the `SharedArrayBuffer` feature in Javascript. These are *relatively* new (but not cutting edge) features in web browsers; customers may find that ICE Desktop for Web is incompatible with certain out-of-date browsers and operating systems.

ICE Desktop for Web has been tested and qualified for use on these browsers:

- Chrome: 114 and newer for Windows
- Edge: 114 and newer (Internet Explorer, all versions, NOT supported) for Windows

ICE Desktop for Web is also believed to work (but has not been fully tested) on these browsers:

- Chrome/Edge 114 and newer on other platforms
- Safari: 16.5 and newer; all platforms
- Firefox: 113 and newer; all platforms
- Opera: 100 and newer; all platforms

4.2.2.2 Mobile browsers ICE Desktop for Web is not supported on mobile (iOS and Android) web browsers. Many mobile browsers do not support WebAssembly or the `SharedArrayBuffer` feature, and even those that do will find that ICE Desktop has not been optimized ("responsive") for small screens.

4.2.2.3 HTTPS is required ICE Desktop for Web will only work when hosted from a server using a secured connection (HTTPS). This implies the ICE Server must be configured with a fully-qualified domain name and accompanying SSL certificate.

Why? Our media engine (Engage) relies on a feature called `SharedArrayBuffer` to operate. As a result of the Meltdown and Spectreflaw discovered in many CPU architectures, the `SharedArrayBuffer` feature can be used by attackers to access information and data that should

be private. To eliminate this risk, browser vendors have restricted the use of this feature to web pages executing within a "secure context" which demands (among other things) that the page be loaded over HTTPS. Google provides a detailed description and history of this limitation, here: https://developer.chrome.com/blog/enabling-shared-array-buffer/.

4.2.3 Connection limitations

Channels in the browser must be configured to use a Rallypoint connection (not multicast/RTP) or else they will appear disabled and display a "Channel not available on web" message.

However, unlike native clients, the web browsers cannot establish raw TLS sockets with a server. Instead, ICE Desktop for Web connects to the Rallypoint using a secured web socket. Since the web socket cannot share the same port as the TLS socket, the Rallypoint now listens on two ports: typically 7443 for TLS and 8443 for web sockets.

ICE Desktop for Web requires the following conditions to be met in order for a channel to operate within a browser:

- Only Rallypoint versions 1.236 and newer are compatible with ICE Desktop for Web. Web clients will not be able to establish a connection to external or tactical Rallypoints running older software.
- Rallypoints must be configured to use port 7443 in ICE Desktop (under Settings → Rallypoints).
- Tactical Rallypoints (those configured manually) must be configured to enable web socket connections on port 8443 and accept client connections on 7443.

4.2.3.1 Rallypoint failover The Engage media engine in the web client does not support Rallypoint failover. When a channel is configured to use a Rallypoint with multiple ingress addresses, the web client will connect only to the first address configured, even if that Rallypoint is unreachable.

This limitation may be fixed in a future release.

4.2.4 Certificate concerns

The Rallypoint acts as a server that the browser must establish a connection to using a secured HTTPS connection. Successfully establishing a connection requires that the Rallypoint present a *trusted* server identity certificate to the browser. This implies that the Rallypoint must be configured with a certificate and private key and the certificate must be issued by an authority trusted by browsers connecting to it.

For Rallypoints deployed inside the ICE Server (i.e., those created at the time of installation), the ingress server identity certificate and key will be used as the identify certificate of the Rallypoint. Recall that the server identity certificate is supplied as the first certificate in the PEM bundle entered in the ICE OS Configuration Wizard (on the TLS page).

When using a widely accepted ingress certificate (one issued by a common, commercial certificate authority) this should work out of the box. However, for administrators using an "enterprise" certificate issued by their IT department (sometimes mistakenly called a "self-signed" certificate), things get complicated.

As of this writing, the Rallypoint is limited to providing browser clients only the identity certificate, and not the chain of authorities used to issue to it. This has the following limitation:

> **The root CA certificate AND every intermediate certificate in the signing chain must be installed and trusted by the browser.**

The process for installing and trusting a certificate is platform-dependent (see this article for Windows and macOS instructions).

Careful: If you do not install/trust all intermediate certificates in the chain, the ICE Desktop for Web application will appear to work without visible warning, but all channels will display "Channel not available on web" irrespective of how they are configured.

This problem occurs because of the way the Engage media engine establishes trust with the Rallypoint.

Typically, an administrator only needs to install and trust the root CA certificate on a client system to allow the browser to navigate to a site without receiving a security error. This works because most web servers will present the browser with the entire certificate chain required to validate the identity certificate. So long as the root certificate in the chain is already present and trusted on the client system, the browser will be able to verify each link in the signing chaining from the server identity to the root CA.

However, the Rallypoint only supplies its server identity certificate to the client, NOT the entire chain. If the identity certificate was issued directly by a trusted root CA installed on the client system, this works fine. But if there is a delegated authority (an "intermediate certificate") in the chain, as is commonly the case, the client browser will NOT be able to verify the Rallypoint's certificate unless each of the intermediate certificates and root certificate are all installed and trusted by the client. This is true even when the web server (the server hosting the ICE Desktop for Web application) has presented the entire chain to the browser when loading the page. This is because the connection between the channel and its Rallypoint is distinct from the connection to the web server and the trust between the web application and the Rallypoint must be established independently.

4.3 ICE Server architecture

ICE Server is authored primarily in Java and makes use of Kafka (an open-source messaging platform) and Cassandra (an open-source non-relational database).

ICE Server is comprised of the following subcomponents:

Table 9: ICE Server system subcomponents

Resource	Description
Apache Cassandra	All data persisted by ICE Server is stored in Apache Cassandra, a distributed, wide column store, NoSQL database management system designed to handle large amounts of data across severs and data center, providing high availability with no single point of failure.
CassKop Cassandra Operator	Provides Kubernetes-level administration and management over Apache Cassandra (database) resources running inside the cluster.
Client Bridge	Client bridge terminates ICE Desktop and ICE Mobile WebSocket connections to the ICE Server, placing messages received from clients on the platform Kafka message topic (for processing by *Platform Services*). Forwards relevant messages from the platform Kafka message bus to attached clients.
Elasticsearch	Elasticsearch is a distributed search and analytics engine providing powerful, lightning-fast search capabilities. Used by ICE Server to power people and channel search functions in the application. Be aware that the Kubernetes EFK monitoring and analysis tools ships with its own Elasticsearch that operates independently of ICE Server's.
Apache Kafka	A stream-processing platform providing a unified, high-throughput, low-latency platform for handling real-time data feeds. Services inside of ICE Server communicate with one another using Kafka message topics.
Strimzi Kafka Operator	Provides Kubernetes-level administration and management over Apache Kafka (messaging) resources running inside the cluster.

Resource	Description
Model Manager	Responsible for establishing required keyspaces, tables, columns and topics within Cassandra and Kafka. Manages database table versioning and migrating data during in-place ICE Server software updates.
Platform Services	Primary message processing and application business logic. Platform Services receives messages from a Kafka message topic, processes them (which may include performing database lookups or writes), and places a response message back on the topic.
REST Bridge	A REST-style facade over the system. Takes incoming HTTP requests, converts them to Kafka messages destined for Platform Services, and converts the resulting response message into an HTTP response.
Server Bridge	Similar to Client Bridge, Server Bridge terminates WebSocket connections established by ancillary server components (like ICE Telephony, ICE Static Reflectors and ICE Patch Servers).
Apache Zookeeper	Apache Zookeeper is a centralized service to maintain naming and configuration data and to provide flexible and robust synchronization within distributed systems. Zookeeper keeps track of status of the Kafka cluster nodes and it also keeps track of Kafka topics, partitions etc, and keeps them in sync.

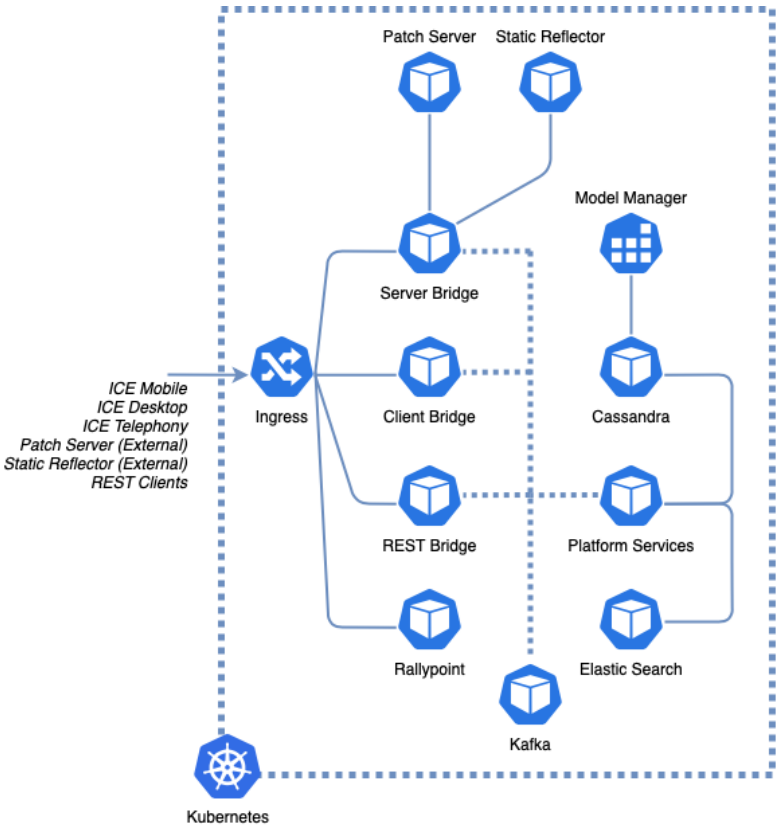

Figure 2: ICE Server system component diagram

4.4 Messages

The ICE Server is a message-oriented system modeled on the concept of *services* and *models*. While messages are not exposed to the end user or the system administrator, this information is provided to offer a better understanding of how these services operate "under the hood."

A message encapsulates information sent between entities in the system. Messages can be sent, point-to-point between a single sender and a single receiver or *broadcast* from a single sender to any entity interested in receiving the message.

A message is a JSON-formatted string consisting of:

- A message type string, representing the kind of message.
- A header object indicating the address of the entity that the message is directed to; whether this message represents a request being made of a service method; and a sender-generated correlation ID (used to associate a request message with its response message).
- A payload object consisting of JSON-formatted attribute/value pairs.

4.4.1 Services

A service is a logical component that provides one or more *service methods* that can be invoked by another entity in the system by sending a message addressed to it. A service method listens for *request* messages addressed to it, performs some process, and emits a *response* message in acknowledgement. Every method on on every service can be invoked through a REST service API.

4.4.1.1 Example of a message sent to a service An example of a message sent to the Person Service requesting a query be performed and the first ten persons matching the search string `def` be returned

```
{
  "type": "person:FindPersons",
  "headers": {
    "isRequest": true,
    "destination": "SERV:person:",
    "correlationId": "c8fb3787-32e4-4296-b8cd-ab8f13c58133"
  },
  "payload": {
    "messageType": "person:FindPersons",
    "attributes": {
      "searchString": "def",
      "limit": 10
    }
  }
}
```

4.4.2 Models

A model represents an addressable entity in the system that is analogous to an object in an object-oriented programming language. A model encapsulates data (in form of attribute/value pairs) with a set of operations that can be performed on the data.

4.4.2.1 Capabilities ICE models are able to support a multiple inherence model through *capabilities*. Models are composed of two or more *capabilities*; a *base capability* common to all models, and one or more type-specific capabilities.

Each capability defines its own set of attributes and methods, the name of each is prefixed with the type (or namespace) of the capability. This prevents two capabilities from inadvertently aliasing the attributes or functions of another capability applied to a single model.

Through the base capability, every model contains the following:

- A permanent, universally unique ID (`base:id`) that differentiates the object from every other object in the system. For example, `e071175e-2bf8-4c06-96cd-88115ed3e295`.
- An address (`base:address`), consisting of the object's namespace and ID concatenated together with a colon and prefixed with `SERV:`. For example, `SERV:person:e071175e-2bf8-4c06-96cd-88115ed3e295`.
- A method (`base:GetAttributes`) for retrieving the attributes of the model and a method (`base:SetAttributes`) for setting attributes.

4.4.2.2 Example of a message sent to a model

An example of a message addressed to a Channel model requesting the list of persons participating in the channel:

```
{
  "type": "chan:ListPersons",
  "headers": {
    "isRequest": true,
    "destination": "SERV:chan:5fc7653f-5e57-4569-b74f-1fcd24ff22d6",
    "correlationId": "7dffa31c-9aef-4a2d-b9da-713784035962"
  },
  "payload": {
    "messageType": "chan:ListPersons",
    "attributes": {}
  }
}
```

4.5 Tactical and enterprise modes of operation

Instant Connect offers two primary modes of operation: *enterprise-mode* in which the system's configuration (user identities, channels, etc) are managed by a centralized server, and *tactical-mode* in which users share channel configurations—called a *mission*—with one another through a QR code or a `.ice` file.

While these are often described as *modes*, the term is misleading: They interoperate together seamlessly in both ICE Desktop and ICE Mobile applications. A user can be logged into an ICE Server and also communicate on a tactical channel that was shared with them via email, for example. In both operating models, the same ICE media engine in used to transport audio between users. The primary difference is how configuration and presence data is distributed across teams.

The table below describes high level features :

Table 10: Tactical versus enterprise feature support

Feature	Enterprise	Tactical
Centralized management and control of channel access	Supported	Not supported
Export and distribute channel configurations via file/QR code	Not supported	Supported
PTT over IGMP multicast	Supported	Supported
PTT over SSL using Rallypoints	Supported	Supported, but requires deployment of a Rallypoint component reachable on the network by tactical clients.
End-to-End PTT encryption	Supported	Supported
User online/offline presence	Supported	Supported within the context of a mission (i.e., can tell if a user is participating in a shared mission, can't determine if the user is "online" in a different mission)
User location	Supported	Supported
Private telephone call between ICE users	Supported	Not supported
Public telephone call to PSTN via ICE Telephony	Supported	Not supported
Telephone dial-in to channel	Supported	Not supported

A tactical user has no login credential (since there is no server to log into) but can establish their own display name. There is no concept of privileges or "administrator" access in tactical mode since there is no centralized point of enforcement. Any user has the ability to create and modify channels which they can freely share with others. To prevent unauthorized users from eavesdropping on a channel, the creator of the tactical channel may assign an encryption key to the channel. Only users with whom the channel configuration has been shared will have the key necessary to decrypt audio transmitted on it.

4.5.1 Sharing user presence in tactical mode

In tactical-mode, user presence information is conveyed to other users through a special kind of non-audio channel called the *mission control channel*. This special channel is shared by all users of a given mission. Each mission has one and only one mission control channel associated with it. However, clients can participate in multiple missions at the same time. Enterprise mode uses ICE Server to convey this information between users; there is no mission control for ICE-managed channels.

Instead of communicating audio traffic, mission control conveys presence, location and other metadata and signaling (like emergency alerts). Just like an audio channel, the mission control channel may be encrypted using a key known only to members of the mission and can be configured for multicast or Rallypoint network operation.

4.6 Mission file format specification

A mission describes the configuration of zero or more channels including those channels' transmit and receive ports, codecs, and Rallypoints. Missions are intended to be shared between ICE users as files or QR codes.

Third parties interested in writing software to produce their own, ICE-compatible mission files can do so by using this section as a guide. A `.ice` mission file contains a serialized representation of a single mission object. Object format specification and serialization algorithm is described below.

4.6.1 Mission object

A valid mission consists of a single JSON object with the following properties:

Table 11: Mission file format, mission object definition

Property	Type	Required	Description
`meta`	Meta object	Required	Provides metadata about this mission including its creation date and format version. See the Meta Object section for details.

Property	Type	Required	Description
`id`	String	Required	A UUID4 (conforming to RFC4122) that uniquely identifies this mission from all other missions. The mission ID MUST be regenerated any time there is a change to the mission name. The generated value must be unique for any given name, and missions configured with the same name should generate the same ID using a common hashing algorithm. See the Mission / Group ID Generation section for details
`name`	String	Required	The user-assigned name of the mission, typically displayed as a title in the user interface. Ideally 16 characters or less; longer names may be truncated.
`description`	String	Optional	A short, user-assigned description of the mission. Should be 256 characters or less; longer descriptions may be truncated.
`groups`	Array (group objects)	Required	An array of groups (channels) specified by the mission, sorted, descending, by group name. See the Group Object section for details.

4.6.1.1 Meta object Describes the creation date, mission format version and other metadata associated with the mission.

Table 12: Mission file format, meta object definition

Property	Type	Required	Description
`type`	String	Required	The file format identifier, always equal to the string `ICE-MISSION`.
`version`	Number	Required	The format version of the mission document; should be 1 for the initial file format.
`created`	String	Required	An ISO-8601 formatted date and time string representing when the mission was created. For example, `2019-02-07T18:44:43.511Z`

4.6.1.2 Group object

Describes the set of channels defined by this mission.

Table 13: Mission file format, group object definition

Property	Type	Required	Description
`type`	Number	Required	The channel type indicator. Use 1 for talk (audio) channels; use 2 for mission control (presencing) channels.
`id`	String	Required	A UUID4 (conforming to RFC4122, optionally surrounded by braces) that uniquely identifies this group from all other groups. Unless `idOverride` is **`true`**, the group ID **must** be regenerated any time there is a change to the group configuration. The generated value must be unique for any given configuration, and groups configured with the same values should generate the same ID using a common hashing algorithm. See the Mission / Group ID Generation section for details.
`idOverride`	Boolean	Required	Allows the group creator to enable or disable automatic ID generation. `idOverride: true` means that it's up to the user to provide the UUID manually.
`description`	String	Optional	A short, user-defined, human readable description of the channel.
`name`	String	Required	The user-assigned name of the channel, typically displayed as a title in the user interface. Ideally 16 characters or less; longer names may be truncated.

Property	Type	Required	Description
`cryptoPassword`	String	Optional	The cryptographic baseline key material used to generate the symmetric encryption key. A 32-byte binary value, represented in hexadecimal text. May be generated by hashing a user-generated password. When this attribute is not present, audio is transmitted in the clear. It is not acceptable to pass an empty string to indicate no encryption; either the attribute is present with a valid, 32-byte hex value, or it must be absent from the JSON document.
`rx`	Address Object	Required	The receive multicast address and port associated with this channel. Talk bursts transmitted on this address and port will be received by this channel. Typically the transmit and receive addresses are the same, but are not required to be. See the Address Object section for details.
`tx`	Address Object	Required	The transmit multicast address and port associated with this channel. Transmitted talk bursts will be received by channels whose `rx` property specifies this same address and port. Typically the transmit and receive addresses are the same but are not required to be. See the Address Object section for details.
`txAudio`	Transmit Audio Object	Optional	The transmit audio configuration (codec, duplex, etc.) used when sending talk bursts. See the Transmit Audio Object section for details. This property is only valid on talk channels (Type 1).

Property	Type	Required	Description
`presence`	Presence Object	Conditionally Required	Mission control configuration parameters. See the Presence Object section for details. Required for mission control (type 2) channels; ignored for audio (type 1) channels.
`rallypoints`	Array (Rallypoint Object)	Optional	A list Rallypoints that this group should connect to. An empty array or undefined implies that the channel does not operate using a Rallypoint (multicast only).

4.6.1.3 Address object

Describes a multicast address and port.

Table 14: Mission file format, address object definition

Property	Type	Required	Description
`address`	String	Required	An IPv4 or IPv6 multicast address. Note that IPv6 has not been fully tested.
`port`	Number	Required	A non-administrative port number (in the range 1000-65535)

4.6.1.4 Transmit audio object

Describes the codec, duplex mode and framing specifications for transmitted audio.

Table 15: Mission file format, transmit audio object definition

Property	Type	Required	Description
`encoder`	Number	Required	Specifies the codec used to transmit audio; acceptable values are: `1`: G.711 alaw (64 kbps) `2`: G.711 ulaw (64 kbps) `3`: GSM 6.10 (13.3 kbps) `10`: AMR Narrowband (4.75 kbps) `11`: AMR Narrowband (5.15 kbps) `12`: AMR Narrowband (5.9 kbps) `13`: AMR Narrowband (6.7 kbps) `14`: AMR Narrowband (7.4 kbps) `15`: AMR Narrowband (7.95 kbps) `16`: AMR Narrowband (10.2 kbps) `17`: AMR Narrowband (12.2 kbps) `20`: Opus Narrowband (6 kbps) `21`: Opus Narrowband (8 kbps) `22`: Opus Narrowband (10 kbps) `23`: Opus Narrowband (12 kbps) `24`: Opus Narrowband (14 kbps) `25`: Opus Narrowband (16 kbps) `26`: Opus Narrowband (18 kbps) `27`: Opus Narrowband (20 kbps) `28`: Opus Narrowband (22 kbps) `29`: Opus Narrowband (24 kbps) `30`: MELPe Narrowband (600 bps) `31`: MELPe Narrowband (1.2 kbps) `32`: MELPe Narrowband (2.4 kbps)
`fdx`	Boolean	Optional	Full duplex mode when true; half duplex mode when false. Default **`false`**.
`framingMs`	Number	Optional	The number of milliseconds of audio to be encapsulated into each transmitted UDP packet. Typically `20`.
`maxTxSecs`	Number	Optional	The longest transmit duration (in seconds) allowable on this channel. When `0` no limit is applied. Default `0`.

Property	Type	Required	Description
`noHdrExt`	Boolean	Optional	When **`true`** no custom RTP headers wills be appended to transmitted audio packets; useful to disable on interop channels. Default **`false`**.

4.6.1.5 Presence object Configures the behavior of the mission control channel.

Table 16: Mission file format, presence object definition

Property	Type	Required	Description
`format`	Number	Required	The presence channel format identifier. Should be `1`.
`intervalSecs`	Number	Optional	Number of seconds between presence packets. Default `30`.
`forceOnAudioTransmit`	Boolean	Optional	Forces the client to send a presence update each time they transmit audio. Default **`false`**.
`listenOnly`	Boolean	Optional	When **`true`** indicates that the client should never send its own presence updates on the channel. Default **`false`**.

4.6.1.6 Rallypoint object An object whose keys represent the unique ID of a defined Rallypoint; the value of each key is a Host Object representing the Rallypoint's address and port.

Table 17: Mission file format, Rallypoint object definition

Property	Type	Required	Description
`host`	Host Object	Required	Specifies the address and port of the Rallypoint. Note that the X.509 certificate and key used for mutual authentication with the client is "baked into" the ICE client applications and are not specified inline here.

4.6.1.7 Host object

Describes a multicast address and port.

Table 18: Mission file format, host object definition

Property	Type	Description
`address`	String	An IPv4 or IPv6 multicast address. Note that IPv6 has not been fully tested.
`port`	Number	A non-administrative port number (in the range 1000-65535)

4.6.2 Serialization specification

Once a JSON-formatted mission object has been defined, it can be serialized using the following algorithm:

1. Create a payload string consisting of the following elements concatenated together:
    1. Magic header value: `&*3$e1@E`
    2. Version header value: `001`
    3. The JSON-formatted mission (as described above), encoded in UTF-8. The equivalent of calling, in Javascript, `JSON.stringify(myMissionObject)`.
2. GZIP compress (RFC-1952) the payload string.
3. Optionally encrypt the compressed result using the technique described below.
4. Encode the compressed and optionally encrypted result in Base91 format. This value is the *serialized mission payload*.

This serialized mission payload can either be written to a file (ending with the `.ice` suffix), or used to produce a QR code. Be aware that QR codes can encode no more than 2,953 bytes of data; if the length of the serialized mission payload exceeds this constant, the mission is too complex to be represented as a QR code.

1. When a deflection URL is desired, the binary data to be encoded by the QR code is equal to the deflection URL, followed by a forward slash and two question marks (`/??`), then the base91-formatted result generated in the previous step. For example, `http://www.dillonkane.com/??<base91-data>`.
2. When no deflection URL is needed, simply create a binary QR code from the base91 encoded data.

> A deflection URL hides the mission data inside of an ignored query string parameter of a URL. This has the effect of making it seem to standard QR code that the QR code simply encodes a URL. Most will simply open a browser to site referred to by the deflection URL.

4.6.3 Automatic mission and group ID generation

ICE tactical clients support a feature called “automatic channel identification” which generates channel IDs automatically by hashing relevant components of the channel configuration any time they change. This has the effect of allowing any two equivalently-configured channels to be considered the same channel.

Any time a group or mission configuration is changed, the client must recalculate the ID of the group using a common hashing algorithm. It is important that all clients calculate the same ID for any given configuration. To achieve this, clients must be able to implement RFC-4122, version 5, commonly referred to as *UUIDv5* (see section: *4.3. Algorithm for Creating a Name-Based UUID*).

For backward compatibility, any Mission or Group being imported (from durable storage, QR codes, etc) must ignore any specified ID and instead favor recalculating the ID based on the provided configuration.

UUIDv5 produces an ID from *name* and *namespace* input strings. In order for different tools to produce the same ID these values must be exactly, byte-for-byte identical. The *namespace* input must be the literal value `d15fbc09-ad8b-4080-be3c-15ab56d08d00`. The *name* input must be the following group configuration values concatenated (without a delimiter) in this specific order:

1. `name`
2. `type`
3. `cryptoPassword` (omit when the group does not use encryption)
4. `rx.address`
5. `rx.port`
6. `tx.address`
7. `tx.port`
8. For each Rallypoint in the list, concatenate the following:
    1. `rp.address`
    2. `rp.port`

5 ICE Media Engine

The ICE media engine is a set of software libraries written in C/C++ and cross-compiled for all ICE-compatible platforms. ICE clients achieve high-performance, low-latency audio by using a native library that runs "close to the metal."

The ICE media engine has several key responsibilities:

- **Managing Audio Hardware:** It controls the microphone and speaker hardware on the user's device.
- **Audio Encryption:** It encrypts and decrypts audio packets to ensure secure communication.
- **Audio Encoding and Decoding:** It decodes received audio from any supported codec and encodes transmitted audio using the codec configured for the channel.
- **Network Communication:** It handles receiving and transmitting audio over the network. This includes establishing and monitoring connections to configured Rallypoints, managing low-latency jitter buffers to prevent gaps or stuttering audio, and joining or leaving IGMP multicast groups.

5.1 Encryption

5.1.1 Symmetric Encryption

The ICE media engine uses AES (Advanced Encryption Standard) with 256-bit keys for all symmetric encryption. The AES algorithm operates in CBC (Cipher Block Chaining) mode to ensure data security.

5.1.2 Asymmetric Encryption

During the TLS encryption setup phase, ICE clients use asymmetric encryption to establish a secure connection between a client application and a Rallypoint. The specific algorithm used is determined by a cipher agreed upon by both the Rallypoint and the client, based on the X.509 certificates they exchange during the handshake process.

5.1.3 Symmetric Key Derivation

ICE never stores or transmits encryption keys directly. Instead, keys are algorithmically derived using the PBKDF2algorithm, approved by the NIST. Here's how the process works:

1. **Baseline Key Material (BKM):** The incoming passphrase provided by the user, referred to as the *Baseline Key Material* (BKM), serves as the foundation for key derivation.
2. **Salt and Iterations:** The BKM is combined with a 128-bit salt, and the PBKDF2 algorithm performs 15,000 iterations to generate the derived key. (NIST recommends a minimum of 10,000 iterations.)
3. **Derived Key:** The result is a secure, 256-bit encryption key used for encrypting data.

> **Note:** To simplify the user interface, ICE Desktop and ICE Mobile label the BKM as the "encryption key." While not technically accurate, this term is used to help users understand its purpose as a parameter for deriving the encryption key.

When sharing channel configurations—either tactically via QR codes and files or through the ICE Server in enterprise mode—it is the BKM that is transmitted between clients. The derived encryption key and the salt used to generate it are never shared.

5.1.4 Traffic Encryption

Channels that are encrypted use AES256-CBC, as described above. When traffic is transmitted over UDP, the entire UDP payload is encrypted. This means that an attacker cannot easily determine if the traffic is in a known format (such as RTP) or a custom format implemented by ICE. Even if the attacker correctly assumes that the payload uses a standard format like RTP, decrypting it becomes significantly more difficult because:

1. The RTP headers are encrypted.
2. Each packet is preceded by a unique initialization vector, increasing the complexity of cryptanalysis.

When traffic is transmitted over TCP, such as in client connections to Rallypoints or between Rallypoints in a mesh network, it is secured using TLS 1.3. This TLS encryption adds an extra layer of security on top of any existing channel encryption. Therefore, if a channel's traffic is encrypted and flows through TLS, it becomes double-encrypted in transit.

It is important to note that ICE generally treats all traffic as packets transmitted over UDP. Even if packets are sent over TCP, ICE processes them as if they were UDP. Essentially, ICE views TLS connections as secure tunnels, similar to VPN connections, through which regular UDP traffic is transmitted as a TCP stream.

5.2 Required network quality of service

Instant Connect is a real-time audio communications solution that relies on an IP network to transport audio between users of the system. As with all networked, real-time communication systems, the underlying network's quality of service (QoS) is critical to the overall performance of the communication system. The table below describes the QoS required by all network segments on which ICE is operating.

Each network parameter has an "ideal", "acceptable" and "unacceptable" range of operation. The definition of these ranges is as follows:

- **Ideal**: Uninterrupted voice communication with very little or no perceptible delays, echos, drop-out, or lost talk bursts.
- **Acceptable**: Perceptible degradation in quality, which most users are likely to find acceptable (especially if network is "ideal" most of the time and enters this range only intermittently).
- **Unacceptable**: Most users will find the system inoperable or difficult to use.

Table 19: Network quality of service parameters

Parameter	Ideal	Acceptable	Unacceptable	Notes
Bandwidth				Varies based on usage and configuration; described in detail in the following section. For planning, assume 64kbps per user.
Latency *(single-direction)*	<40ms	<250ms	>250ms	When talkers are in close proximity to one another, sub-40ms latency is important to prevent a "stadium effect" in which the same audio from different sources sounds as though its echoing from multiple directions. Latency above 250ms introduces perceptible delay and begins to impair normal human speech patterns.

Parameter	Ideal	Acceptable	Unacceptable	Notes
Jitter	<5ms	<30ms	>30ms	Jitter increases effective latency in the communication path between talker and listener. High jitter values combined with high latency produce noticeable delay in the audio path and will begin to impact speech patterns. Further, jitter measurements above 30ms may result in degraded or lost audio.
Packet Loss	0%	<2%	>2%	Even small amounts of packet loss produce noticeable degradation and artifacts, especially when using Rallypoints.

5.3 Network bandwidth considerations

There's a lot that goes into determining how much traffic is going to be produced on your network, when that traffic is going to be produced, and who's going to be producing it. Every environment is different.

Fundamentally, you'll need a pretty good handle on how many users you'll have, how many channels they'll be active on (listening as well as talking), where they're located, and what the network looks like between them.

A key criteria, though, is in knowing what your network transport capabilities are—specifically whether your network supports bi-directional IP multicast or not. This distinction is critical. In a multicast environment, the underlying network infrastructure takes care of efficient distribution of packet traffic; there's no centralized servers that process and/or forward traffic between users. Rather, users' traffic propagates in what is effectively a one-to-many setup.

If your network does not support multicast, you'll have to have some way in which to create a *multicast overlay* to create a simulated multicast environment. While there's a few ways to do this using 3rd-party tools; our recommended way is to use Rallypoints as high-performance packet forwarders that are optimized for ICE-specific operations.

5.4 Packet streams

Understand that when someone talks on an ICE channel, they produce a stream of packets. Those packets contain voice data encoded ("compressed," if you will) with a CODEC (COder/DECoder). The output from the coder is chopped up into blocks (we call them "frames") of audio, and sent out on the network. In any voice conversation, there will always be at least one stream of packets.

5.4.1 Multicasting

Imagine that we have 4 other people listening on the channel that our user is talking on. In a multicast environment, the amount of traffic on your network is a single stream, because the multicast network infrastructure makes sure not to needlessly duplicate traffic to the receiving endpoints. So, whether we have 4 people listening—or 400, or 4,000—we still only have one stream. And that's terrific because it means we can scale our user base without worrying too much about how our network is going to be affected.

Don't forget, though, that this is for a stream representing a *single* channel. Each additional channel where someone is transmitting will create another stream. So, for example, if we have someone talking on the *Alpha* channel and another person talking on *Bravo* channel; we'll have 2 streams on the multicast network. If someone talks on *Charlie* at the same time; we'll have 3 streams (and so on).

> In a multicast environment, the number of streams (and therefore bandwidth utilization) is directly proportional to the number of people speaking.

5.4.2 Unicasting with Rallypoints

But what if our network doesn't support multicast? Maybe the enterprise network isn't engineered for multicast, or we have to traverse public networks (such as the Internet) where multicast isn't available. What happens then?

Here, the distribution of traffic from the person speaking to the people listening has to go via some sort of central point that all the users are connected to. We call this "unicasting." The ICE Rallypoint component provides this centralized point of interconnectivity.

Things like access control, encryption, floor management, audio transcoding, and other functions usually conducted by servers are delegated to the ICE clients themselves because ICE operates under the assumption that the network is multicast in nature and that no centralized server architecture is necessary. ICE Rallypoints are a means to create a "multicast-like" environment on non-multicast networks. With the understanding, though, that when using Rallypoints, network traffic utilization is pretty similar to what you'd see in traditional, server-based unicast systems.

In a unicast environment, a stream coming from one user and being sent to others through a central point creates multiple streams on the network. The count of those streams is comprised of the stream coming from the person transmitting, and one stream each for each person receiving it.

> In a unicast environment, the number of streams (and therefore bandwidth utilization) is directly proportional to the number of people speaking *and* listening.

Obviously unicasting is far less efficient than multicasting and will not scale cleanly in the same way that multicasting does. In a unicast environment, it's important that we concern ourselves with how many people may be sending traffic as well as how many people will be receiving it. And, of course, we have to be concerned (as with multicasting) with how many channels we have in the system since it's a big multiplier in calculating bandwidth utilization.

5.5 Bandwidth calculations

5.5.1 Packet Structure

We're going to talk a bit below about payloads and headers and "taxes" and other stuff. Keep these drawings in mind as you read through:

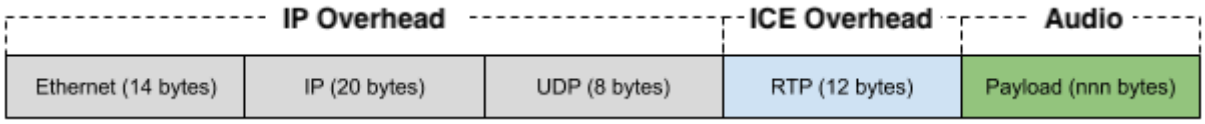

Figure 3: Clear (unencrypted) UDP packet

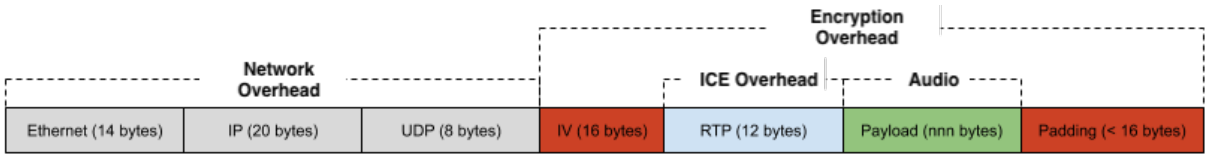

Figure 4: Secure (encrypted) UDP packet

5.5.2 A Variety Of CODECs

Different CODECs produce different outputs of varying quality traded against bandwidth utilization. The choice of which CODEC to use is based on a number of of things including computational load placed on the devices processing the audio, the desired quality of the audio, the network bandwidth required to transport it, and interoperability with systems that are limited to using certain CODECs.

CODEC design must provide a balance between these conflicting goals: How to represent audio in its highest fidelity, using a few bits as possible, with as little burden on the end-user device CPUs as possible ("computational complexity"). Historically, computational complexity was always a big factor as CPUs were pretty slow by today's high-performance hardware standards, and, therefore, engineers often chose CODECs that would not tax hardware. Thus, we have a historical proliferation of CODECs (such as G.711) that uses 64 Kpbs of bandwidth with very low computational complexity. With newer hardware, we're free to choose more computationally intensive CODECs that provide a better audio experience with reduced bandwidth requirements.

To wit: G.711 is fixed at 64 Kbps and produces high-quality audio at low complexity. Opus produces audio quality that is even better than G.711 with a much-reduced bandwidth footprint. In fact, Opus operating at 16kbps produces higher audio quality than G.711 requiring four times the bandwidth. With ICE, you can go all the way down to 6kbps on Opus.

At first glance, you might assume that someone transmitting audio using an Opus 16 CODEC is going to use 16kbps on the network. But there's a whole lot of devil in the details.

5.5.3 Packet overhead

Every time a packet of audio (the *payload*) is sent from a device, it includes information that assists the network in moving the packet around as well as information that the receiving device needs in order to know how to handle the payload. It's not unlike writing a letter to someone: The letter is the payload, and the envelope it's mailed in is the information the postal service needs to get your letter to the receipt.

This information is tacked on to the beginning of the transmitted packet in the form of *headers*.

5.5.3.1 Packet headers There are quite a few headers that travel with the payload. First is a header that the receiving endpoint uses to know what do to with the payload. For most of the ICE media engine's traffic in this header is the Realtime Transport Protocol, or RTP, header. Then, when using IP multicast, that resulting RTP packet (consisting of the payload "wrapped" in RTP) is further wrapped in a User Datagram Protocol, or UDP, header. Then, that UDP packet is wrapped in an Internet Protocol, or IP, packet. Finally, the IP packet is wrapped in a packet (sometimes called a frame just to confuse everyone) specific to the underlying transport. This transport could be wired or wireless Ethernet or some other construct. Each of these headers uses a certain number of bytes for each packet, obviously, and therefore bandwidth calculations need to take this into account.

Packet overhead (taxes) generally consumes more bandwidth than the actual payload itself!

For example: let's say your CODEC produces a payload of 10 bytes because its optimized to operate at a very low output rate. Your resulting transmitted packet size (what actually goes out over the network)

consists of these 10 bytes + 12 bytes for RTP + 8 bytes for UDP + 20 bytes for IP + 14 bytes for Ethernet. So, sending a payload of 10 bytes results in 64 bytes (10+12+8+20+14) being transmitted.

But wait, there's more… That's the fixed, guaranteed tax we have to pay the network to get our packet moved around. No matter what our payload is and regardless of size (for the most part), we're going to have these headers on our packets chewing up our bandwidth. The "more" here is encryption.

5.5.3.2 Encryption overhead What if we want to secure our data to hide as much as possible from the bad guys?

The answer here is to encrypt our payload, and, in Instant Connect's case, the RTP header as well. (ICE can't encrypt the UDP, IP, or Ethernet headers as the network then wouldn't know how to deliver the datagram.)

For performance and security reasons, ICE uses the Advanced Encryption Standard, or "AES", encryption algorithm which produces encrypted data in 16-byte blocks. So, if we encrypt 1 byte of data, our encrypted output is 16 bytes. If we encrypt 15 bytes, it is still 16 bytes. If we encrypt 16 bytes, however, the output is 32 bytes. If we encrypt between 16 and 31 bytes, the output is still 32 bytes. Encrypting 32 bytes, will produce 48 bytes of output. And so on. Basically, AES encrypts on 16-byte *boundaries*. That extra padding (which varies in size) is going to affect the size of the data we're transmitting.

Also, ICE operates AES in what's known as Cipher Block Chaining, or CBC, mode. And CBC requires that an extra blob of random data is added to the encrypted output—16 bytes of random data, in fact. This is known as the Initialization Vector, or IV.

Together, the 16-byte block operation and the IV added to each packet substantially increases the size of the packet and therefore our bandwidth utilization.

5.5.4 Packet framing

We now know that the overhead placed on our payload is quite substantial in order to get it dropped into a packet and sent over the network. But we have some wiggle room. Instead of little payloads, we can send bigger payloads. If our overhead for every unencrypted packet is 54 bytes then the best way to reduce the bandwidth utilization is to reduce the number of packets being sent.

Let's say that the 10 bytes of payload consituted 20 milliseconds of audio. That means that for every 1000 milliseconds (1 second) of audio, we'd be sending 50 packets (1000/20). Now, the payload in each of those packets is 10 bytes and the overhead is 54 bytes. Therefore, every second, we'd be sending 500 bytes of payload (10 bytes of payload * 50 packets) and 2,700 bytes of overhead (54 * 50). The total number of bytes we'd send every second is therefore 3,200 bytes (500 + 2,700).

However, let's say we changed the payload size and instead of sending audio every 20 milliseconds, we send at 40 millisecond intervals. Now, the number of packets we'd send every second halves to 25 packets. The total amount of payload we send is still 500 bytes (now 20 bytes per payload but only 25 packets in a second - i.e. 20 * 25 = 500) but the total overhead is now only 1,350 bytes (54 * 25); a full 50% reduction in overhead!

The tradeoff, though, is we now have to wait for 40 milliseconds before we send any audio; and it means receivers have to wait 40 milliseconds to receive it. Doing so increases audio latency, which is typically undesirable, but in a push-to-talk environment this modest increase in latency is not perceived by users.

There's a bigger risk here, though. If a 20 millisecond packet gets lost on the network, the receiver only looses 20 milliseconds of audio. If a 40 millisecond packet is lost, the audio dropout is larger. We can go even bigger, we can send 60 millisecond packets, or 80, or even 100 millisecond packets. Consider if we sent 100 millisecond packets: the payload size for 100 milliseconds would now be 50 bytes (10 bytes for 20 milliseconds * 5 … because 100 / 20 = 5), and we'd only be sending 10 packets. Therefore, in a second, our bandwidth consumption would be 500 payload bytes (as always) plus 540 bytes of taxes. A total of 1,040 bytes in a second vs 3,200!

All this stuff is equally relevant if our streams are encrypted—but with extras added on for the encryption initialization vector and padding. Basically, we can treat encryption as simply another, albeit optional, tax.

> **Careful:** As these numbers get bigger and networks drop or delay packets, audio quality begins suffering. So we need to be very careful on what numbers we choose.

The considerations above drive us ultimately to *packet framing* which refers to the size of the audio payload that we transmit. You'll see in the bandwidth utilization tables below how sizing changes bandwidth consumption. These numbers are as accurate as possible but, as with anything technical, there's always some wiggling that needs to be considered.

For instance, CODECs such as G.711 and GSM always produce a reliably-sized output. Other CODECs such as AMR and Opus are variable in size, meaning that they will try to actively *reduce* the amount of traffic based on how much the sender is saying, what their volume level is, the complexity of their voice, and so on. With these so-called *VBR* CODECs (Variable Bit Rate) you may sometimes see lower output sizes than listed in the tables. But you should rarely see higher values. Best, though, to add a "fudge factor" of 1 Kbps or so to the numbers used for planning purposes.

5.5.5 Comparing UDP and TCP

There's one more thing to talk about and that's whether we're using UDP (User Datagram Protocol) or TCP (Transmission Control Protocol).

When multicasting, ICE uses UDP as its transport, and all the numbers and discussion above applies. However, when unicasting via Rallypoint connections, ICE uses TCP as its transport. Like UDP, TCP has its own overhead associated it (things like as link establishment, packet acknowledgements, heartbeats, and whatnot).

When we use Rallypoints, TCP is used to convey packets using Transport Layer Security. TLS imposes further overhead which needs to be taken into account.

Unfortunately, when using TCP and TLS, calculating bandwidth utilization is somewhat nondeterministic in that bandwidth utilization can vary somewhat based on network conditions. The values illustrated in the tables below (those showing TCP over TLS) are calculated as an average, of sorts, of multiple measurements of actual traffic taken under Internet conditions. We've then combined those calculations with the observed values to arrive at what we believe to be best representative of what unicast traffic utilization looks like. Of course, your mileage may vary.

5.6 Bandwidth utilization tables

These tables show bandwidth utilization for each CODEC supported by ICE. There are four tables total: two for IGMP multicast traffic (encrypted and in the clear), and two for unicast (Rallypoint) traffic. Each table illustrates the breakdown per CODEC with different packet framing. The most efficient framing size for each codec is highlighted in **bold**.

5.6.1 Unicast (Rallypoint) bandwidth utilization

Table 20: Bandwidth (Kbps) for unencrypted unicast TCP per framing size (ms)

Codec	Rate (Kbps)	20ms	40ms	60ms	80ms	100ms
G.711 ulaw	64.00	102	86	84	84	**82**
G.711 alaw	64.00	102	86	84	84	**82**
GSM 6.10	13.30	46	37	33	31	**29**
AMR	4.75	**38**	40	40	**38**	**38**
	5.15	39	**37**	38	39	39

Codec	Rate (Kbps)	20ms	40ms	60ms	80ms	100ms
	5.90	40	40	**39**	**39**	41
	6.70	40	42	38	39	**37**
	7.40	40	42	40	**39**	40
	7.95	42	43	42	42	**39**
	10.20	**40**	**40**	43	43	45
	12.20	46	**40**	47	48	**40**
Opus	6.00	40	29	26	24	**21**
	8.00	41	32	27	25	**23**
	10.00	44	33	30	27	**24**
	12.00	46	36	32	30	**27**
	14.00	47	40	34	31	**28**
	16.00	50	41	34	34	**32**
	18.0	52	42	39	36	**34**
	20.00	54	43	41	38	**34**
	22.00	55	48	**40**	**40**	38
	24.00	58	50	45	**42**	43

Table 21: Bandwidth (Kbps) for encrypted unicast TCP per framing size (ms)

Codec	Rate (Kbps)	20ms	40ms	60ms	80ms	100ms
G.711 ulaw	64.00	100	93	90	86	**80**
G.711 alaw	64.00	100	93	90	86	**80**
GSM 6.10	13.30	55	40	35	33	**30**
AMR	4.75	48	48	**47**	48	49
	5.15	49	**47**	50	49	48
	5.90	48	49	50	47	**46**
	6.70	51	**45**	47	54	48

Codec	Rate (Kbps)	20ms	40ms	60ms	80ms	100ms
	7.40	52	53	52	55	**51**
	7.95	53	**52**	55	54	55
	10.20	**52**	54	**52**	**52**	53
	12.20	58	**52**	53	55	53
Opus	6.00	48	31	27	26	**22**
	8.00	54	33	29	27	**25**
	10.00	53	36	33	28	**27**
	12.00	56	39	33	33	**30**
	14.00	57	42	36	33	**31**
	16.00	63	43	40	36	**33**
	18.0	57	45	41	40	**36**
	20.00	65	48	43	**38**	40
	22.00	62	50	45	40	**38**
	24.00	63	53	48	44	**43**

Table 22: Supported Radio Interoperability Codecs

Profile	Codec	txPayloadType	rxExtPayloadType	rxIntPayloadType
Default	All codecs	null	null	null
Trellisware	MELPe 600bps (50)	117	117	79
	MELPe 120bps (51)	117	117	78
	MELPe 2400bps (52)	117	117	77
	All other codecs	null	null	null
Persistent Systems	OPUS - Half Duplex	112	112	118
	OPUS - Full Duplex	111	111	118
Cistech	AMR Narrowband	126	126	122
	Opus 18k	125	125	118

Profile	Codec	txPayloadType	rxExtPayloadType	rxIntPayloadType
Vocality	Opus 18k	120	120	118

5.6.2 Multicast bandwidth utilization

Table 23: Bandwidth (Kbps) for unencrypted multicast UDP per framing size (ms)

Codec	Rate (Kbps)	20ms	40ms	60ms	80ms	100ms
G.711 ulaw	64.00	85.60	74.80	71.21	69.40	**68.32**
G.711 alaw	64.00	85.60	74.80	71.20	69.40	**68.32**
GSM 6.10	13.30	34.90	24.10	20.50	18.70	**17.62**
AMR	4.75	26.35	15.55	11.95	10.15	**9.07**
	5.15	26.75	15.95	12.35	10.55	**9.47**
	5.90	27.50	16.70	13.10	11.30	**10.22**
	6.70	28.30	17.50	13.90	12.10	**11.02**
	7.40	29.00	18.20	14.60	12.80	**11.72**
	7.95	29.55	18.75	15.15	13.35	**12.27**
	10.20	31.80	21.00	17.40	15.60	**14.52**
	12.20	33.80	23.00	19.40	17.60	**16.52**
Opus	6.00	27.60	16.80	13.20	11.40	**10.32**
	8.00	29.60	18.80	15.20	13.40	**12.32**
	10.00	31.60	20.80	17.20	15.40	**14.32**
	12.00	33.60	22.80	19.20	17.40	**16.32**
	14.00	35.60	24.80	21.20	19.40	**18.32**
	16.00	37.60	26.80	23.20	21.40	**20.32**
	18.0	39.60	28.80	25.20	23.40	**22.32**
	20.00	41.60	30.80	27.20	25.40	**24.32**
	22.00	43.60	32.80	29.20	27.40	**26.32**

Codec	Rate (Kbps)	20ms	40ms	60ms	80ms	100ms
	24.00	45.60	34.80	31.20	29.40	**28.32**

Table 24: Bandwidth (Kbps) for encrypted multicast UDP per framing size (ms)

Codec	Rate (Kbps)	20ms	40ms	60ms	80ms	100ms
G.711 ulaw	64.00	96.00	80.00	74.67	72.00	**70.40**
G.711 alaw	64.00	96.00	80.00	74.67	72.00	**70.40**
GSM 6.10	13.30	44.80	28.80	23.47	20.80	**19.20**
AMR	4.75	38.40	19.20	14.93	12.80	**11.52**
	5.15	38.40	19.20	14.93	12.80	**11.52**
	5.90	38.40	22.40	17.07	14.40	**11.52**
	6.70	38.40	22.40	17.07	14.40	**12.80**
	7.40	38.40	22.40	17.07	16.00	**14.08**
	7.95	38.40	22.40	19.20	16.00	**14.08**
	10.20	38.40	25.60	21.33	17.60	**16.64**
	12.20	44.80	28.80	23.47	20.80	**17.92**
Opus	6.00	38.40	22.40	17.07	14.40	**12.80**
	8.00	38.40	22.40	19.20	16.00	**14.08**
	10.00	38.40	25.60	21.33	17.60	**16.64**
	12.00	44.80	28.80	23.47	19.20	**17.92**
	14.00	44.80	28.80	23.47	22.40	**20.48**
	16.00	44.80	32.00	25.60	24.00	**21.76**
	18.0	51.20	35.20	27.73	25.60	**24.32**
	20.00	51.20	35.20	29.87	27.20	**26.88**
	22.00	51.20	38.40	32.00	30.40	**28.16**
	24.00	57.60	38.40	34.13	32.00	**30.72**

5.7 Rallypoint meshing

Perhaps you've setup a Rallypoint and have lots of users connecting to it. But you're running out of CPU because you have thousands of users talking away like crazy. Or, you have have groups of people in different parts of the world that need to connect to their own Rallypoints—but they all still want to communicate with each other. Or, you have a need to provide additional Rallypoints for failover and redundancy purposes. Or, something else…

The solution here, generally speaking, is to just add more Rallypoints. But, you want all those Rallypoints to interconnect with each other. That's where Rallypoint meshing comes in.

For this example we're going to assume we want to setup a bunch of Rallypoints in a cloud environment such as Amazon Web Services (AWS). We'd like to make those Rallypoints available to our users spread around the world. And we don't want to have those users connect to particular Rallypoints. Rather, we want any user to connect to any available Rallypoint and have the Rallypoints take care of forwarding traffic amongst themselves to create what looks like a big, centralized, cloud-based Rallypoint to our users.

First, we're going to need something that front-ends all these Rallypoints; offering a single DNS name (or single IP address) that all our users connect to. Let's call it `cloudrp.example.com`. Also, for purposes of this discussion, we'll say we're going to do all this in Amazon Web Services, taking advantage of Amazon's Elastic Load Balancer.

In an ideal situation, we'd use Rallypoints' ability to forward traffic to a multicast backbone to create something like below:

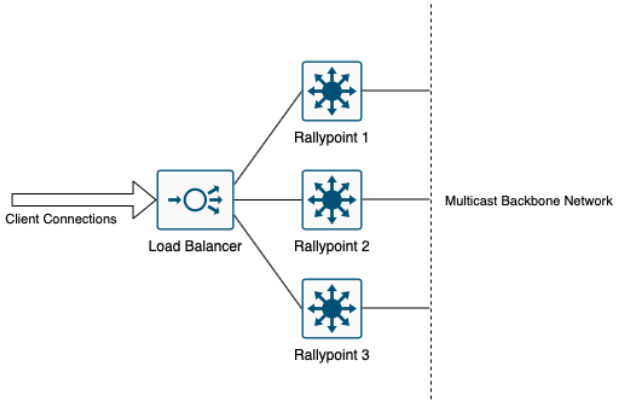

Figure 5: Rallypoint mesh using a multicast backbone

But, sadly, our cloud provider does not support IP multicast (and that is, in fact, true of AWS as well as most of cloud providers). Also, multicast networks can be difficult to setup and maintain so sometimes its just easier to go with unicast.

So, we'd still like to have the logical setup we described above but we have to figure a way to use something other than a multicast backbone. The answer is to create a Rallypoint Mesh.

A mesh is simply a configuration where Rallypoints connect directly to each other for purposes of traffic forwarding. This is not too much different from the way in which IP multicast works anyway. In this case, it's just that the Rallypoints themselves are doing "multicasting" rather than utilizing the IP network for that purpose.

As you can see below, what we've done is to have each Rallypoint in our cloud connect to every other Rallypoint. Now, when a client connects (indirectly) to a Rallypoint, it's traffic is forwarded to other Rallypoints - just like multicast. It's actually rather straightforward.

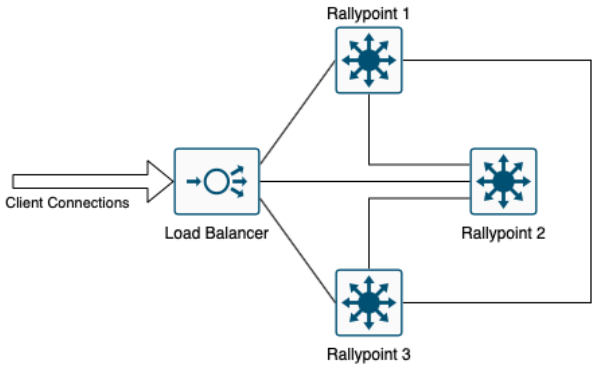

Figure 6: Rallypoint mesh without a multicast backbone

You might be wondering if all traffic from all Rallypoints is forwarded to all other Rallypoints. Doing so would be inefficient, so what Rallypoints do instead is to "subscribe" to each other's individual streams. Thus, if client 1 connected to RP1 and client 3 connected to RP3 and both are registered/subscribed for the same stream, that stream's traffic only flows between RP1 and RP3 - bypassing RP2. And what about security on these links between Ralypoints? Well, they're TLS connections just like from clients to Rallypoints and are subject to the same level of X.509 mutual authentication and TLS encryption.

Lastly, while latency does increase when a mesh of Rallypoints is being used, it increases only by a minuscule amount. Remember that Rallypoints are just packets routers; they do not process traffic payloads, and therefore introduce only milliseconds worth of latency.

6 Satellite server components

A satellite server is an Instant Connect component that extends the functionality of ICE Server but does not need to be co-located with the primary ICE Server system and does not need to run as a

workflow inside a Kubernetes cluster. The ability to distribute audio processing elements throughout an operator's network improves network efficiency and scale by making it possible to locate traffic-intensive system elements at the edge of the network, closer to where they are being utilized.

Table 25: Satellite server components

Server	Support Deployment Models	Description
ICE Rallypoint	Internal, external, tactical	Audio routing network element for push-to-talk channels configured for unicast traffic distribution.
ICE Desktop	Internal, tactical	Web server for the ICE Desktop for Web application; serves HTML and other web resources to a user's browser when they browse to the web application.
ICE Archiver	Internal	Captures audio spoken on channels that have been marked for recording and when provisioned for linguistics, produces transcriptions of the recordings.
ICE Gateway	Internal, external	Bridge and gateway device for interfacing with telephone PBXs or wireline radio gateways.
ICE Patch Server	Internal, external, tactical	Network element used to connect two or more channels together such that audio heard on one is also heard on the others.
ICE Static Reflector	Internal, external, tactical	Bridges unicast and multicast audio streams typically for donor radio integrations.

6.1 Satellite deployment models

- **Internal**: The component runs as a Kubernetes workload ("pod") inside the ICE Server. Internal deployment occurs automatically at the time of installation without requiring any explicit administrator action. These components are identified within the ICE application with the name "internal" or default" (for example, "Default Rallypoint" or "Internal Patch Server") and cannot typically be removed or deleted from the system.
- **External**: The component is installed by an administrator on a Linux, Windows or macOS host and communicates with ICE Server to receive configuration data. External deployments generally behave identically to internal deployments; the distinction is simply in where the software

has been deployed.

- **Tactical:** The component is installed by an administrator on a Linux, Windows or macOS host and operates in an "offline" or stand-alone mode that does not connect to an ICE Server. Configuration of the element is done manually via a ICE Configuration Wizard web application.

6.2 Leader election

Before ICE version 3.6.0, satellite server components lacked full geo-redundancy support. Although a satellite could automatically migrate from a failed ICE Server data center to an active one, there was no automated backup for the satellite itself if it failed. Administrators had no option to provision a backup. In the event of a failure, a new server had to be deployed and manually configured, which included tasks like recreating and reassigning patches to a patch server.

Starting with version 3.6.0, ICE supports the provisioning of multiple physical servers as part of a *server group*. Within this setup:

- One *member server* actively operates on behalf of the group.
- The remaining member servers are on standby, ready to take over in case of a failure.

This update eliminates previous limitations, providing robust geo-redundancy and ensuring seamless failover and backup capabilities.

Why not allow all satellite servers within a group to be active at the same time? The nature of these servers is that they cannot operate simultaneously without producing undesired—and sometimes catastrophic—side effects. For example:

Server type	Side effect of multiple active
Patch Server	Network loops and audio loss. An audio packet bridged by one server is received by the other, and bridged back to the originating server ad infinitum. This will consume large amounts of network bandwidth and result in channel cards that appear "stuck" in a receive state.
ICE Archiver	Duplicated recordings and ops log entries. Each active archiver will archive talk-bursts resulting in duplicated records in the database.
ICE Desktop Server	None; not applicable. Multiple ICE Desktop server may run simultaneously without negative side effects.
ICE Reflector	Potential network loops, feedback and other audio corruption depending on the multicast architecture of the network.

Server type	Side effect of multiple active
ICE Gateway	Inoperative communications. Multiple active gateways registering with the radio or telephone server will result in no gateway successfully establishing communications.
Rallypoint	None; not applicable. Rallypoints are intended to run in an active/active configuration and form a mesh between them

6.2.1 How a member server is elected leader

All member servers within a server group periodically communicate with each other by sending a *ballot message* to other members in the group. This process is used to select a leader for the group. By default, this election cycle occurs every 10 seconds. Ballots can be sent using either a Rallypoint or IGMP multicast, and both methods can be used simultaneously for increased reliability and resilience.

It is important to note that leader elections in the context of Instant Connect differ from typical democratic elections in several key ways:

1. **Self-Voting:** Member servers do not vote for other member servers; they only vote for themselves.
2. **Idempotency:** Ballots are idempotent, meaning receiving more than one ballot from a candidate does not affect the election outcome.
3. **Vote Count:** The winner of the election is not solely determined by the number of votes received.
4. **Decentralized Decision-Making:** Each member server independently determines the election winner. There is no centralized registrar counting the ballots. Instead, each server follows a specific algorithm, outlined in this document, to decide the winner.

> **Note:** Leader elections do not function like conventional democratic elections.

When a member server is created, it is assigned a unique ID used for elections. This candidate ID is randomly generated as a UUID v4 and remains unchanged once assigned.

6.2.1.1 Ballot messages Every ten seconds, each member server submits a ballot to each other member server in the group. The ballot message is a JSON-formatted data structure that looks like this:

```
{
  "id": "a23b4555-e9e7-48ac-b6ce-5f79e56e9cd1",
```

```
  "votes": 1,
  "ms": 1710628477713,
  "rids": [ "4d7e92ac-8c63-4950-aeba-4520e52e4cf8" ],
  "lids": [ "4d7e92ac-8c63-4950-aeba-4520e52e4cf8" ],
  "hmac": "
     f0e5286e1dc98645d026fa7880e302cfbb25b36d9231473aa2f66ed7c94d7def"
}
```

Each property in the message is described in the table below:

Property	Description
`id`	The unique ID of the member server who is submitting this ballot.
`votes`	The number of votes this member server is submitting for itself. This value is used to represent selection priority and makes it possible for an administrator to force a member server to become active, or to have precedence or subservience in an election. A negative value means the server is out of service (represented by toggling the “Enabled” switch in ICE Desktop to the off position); it continues to send ballots for purposes of quorum, but may not ever be made leader.
`ms`	A timestamp (represented in milliseconds since the Unix epoch) of when this ballot was submitted (transmitted to others).
`rids`	Race IDs: A list of zero or more *race IDs* that this member server is competing in. While this election system supports the ability for a member server to participate in multiple elections (*races*) simultaneously, in this context, there is only one race.
`lids`	Leading IDs: A list of zero or more race IDs that this member server is currently the leader of.
`hmac`	A *hashed message authentication code* used to verify the authenticity of the ballot and to assure that a malicious attacker cannot affect the outcome of an election by submitting bogus ballots. The `ms` timestamp, in conjunction with this property prevents replay attacks. This cryptographic hash is based on a salt that is assigned to the server group and randomly generated at the time the server group is created.

6.2.1.2 Choosing a leader When a server receives a valid ballot from another member, it records that server as a potential leader candidate. A ballot is considered valid for twice the election period. For example, a received ballot will be counted in any election that occurs within 20 seconds of the timestamp on the ballot. This window helps accommodate minor clock differences between servers and issues related to Nyquist rate.

During each election, every server determines the leader by ranking candidates based on the number of votes they receive (the `votes` property of the ballot). If one server receives more votes than the others, it is declared the winner. However, if multiple servers have the same priority, they will receive the same number of votes, resulting in a tie. In such cases, the winner is chosen from the tied candidates by selecting the server with the lowest ID (sorted alphabetically).

It is important to note that winning the election does not automatically make a server the leader. Additional criteria must be met for the winning server to assume leadership:

1. **The group must not already have a leader:** If a new candidate server comes online and its ID would make it the natural leader, it will not assume leadership as long as an existing server is claiming to be the leader. The exception to this rule is if a new server has a higher priority; it will immediately take over leadership. This approach prevents unnecessary leader changes while still allowing administrators to force a server into leadership (by increasing its priority) or into standby (by decreasing its priority).
2. **The quorum must be met:** Server groups are configured with a quorum strategy, which represents the number of elector servers that must submit a ballot for the election to be valid. For example, if a server group of three servers uses a "majority" quorum strategy, and two servers fail (leaving only one), the remaining server cannot be elected leader because it does not meet the quorum requirement. In such cases, no server will be elected leader, and the system will remain in a standby/offline state. Refer to the "Quorum" section for details and special considerations.
3. **The winner must be chosen consecutively:** By default, a server must win two consecutive elections before becoming the leader. This prevents situations where a server briefly declares itself the leader due to initial conditions when coming online without receiving ballots from competitors. It also reduces leadership changes caused by temporary network glitches.

Once elected, the leader becomes active while all other servers go into standby mode. Although elections occur frequently (every 10 seconds, by default), the outcome should rarely change unless a candidate server goes offline.

6.2.2 Quorum

The quorum strategy determines the conditions under which a leader election is valid. If an election is deemed invalid, no server will be designated as the leader, causing the system to stop operating. Whether this behavior is desirable depends on the nature of the server group's activities and the design of the network in which it operates. Therefore, it's crucial to understand how these quorum strategies work and choose the appropriate one for your situation.

To address the situation where quorum may be temporarily lost when a new server is added to the

server group, but the new server has not yet submitted a ballot, a quorum is valid for the current and next election (that is, no leader will be elected because of quorum, only if a quorum has not been met for two consecutive elections).

Instant Connect supports three different quorum strategies, each described below:

6.2.2.1 None In this strategy, no quorum is required, and any member server can elect itself if communication with other servers is lost. This approach prioritizes system availability over network safety.

When to use this strategy:

- Use this when there are fewer than three member servers in the group, and having both servers active simultaneously is preferable to having neither active.
- Ideal for networks likely to become partitioned, where activities in each partitioned site should continue to function as normally as possible during a partition event.

6.2.2.2 Majority This strategy requires a majority of electors (defined as 50% + 1 for groups of 2 or more servers) to submit a ballot. Note that in a group of two servers, one surviving server will not take leadership, because 1 is not a majority of 2. However, in a group of only one server, the single server will self-elect when using the majority strategy. This approach prioritizes network safety over availability.

When to use this strategy:

- Use this when there are at least three servers in a group.

6.2.2.3 N minus 1 This strategy attempts to provide an “easy button” for small server groups that grow from one to three members. The `n-1` strategy implies that a quorum exists only when all, or all but one, member servers are submitting ballots. When the number of member servers in the group is 1 or 2 the strategy behaves the same as `none`; when the number of servers is 3, it behaves the same as `majority`.

When to use this strategy:

- Use this when you intend to deploy one, two, or three member servers and you want the system to reasonably adapt to the change in the number of member servers.

6.2.3 Election priority and out-of-service

A member server may be assigned an election priority (by ordering the server within the list of other member servers in the group, in ICE Desktop settings). The election priority determines the number of votes that the server assigns itself when balloting. A higher priority results in more votes, implying that the server will be elected ahead of other servers whenever the higher-priority server is online.

To assure a server will never be elected leader, it can be taken out of service (toggle the “Enabled” switch to off, in ICE Desktop settings). When a server has been taken out of service it will be assigned a negative priority which is interpreted by the leader election algorithm as out of service. Out of service member servers will continue to vote in leader elections, so their vote counts toward reaching a quorum.

6.2.4 Interpreting election results

Whenever the state of a member server’s election status changes it will publish a status report that looks like this:


```
{
  "version": 1,
  "state": "healthy",
  "active": false,
  "updatedMs": 1722976385590,
  "details": {
    "message": "GETTING READY: Waiting to win 2 consecutive elections (won
        1 so far).",
    "httpStatus": {
      "state": "notRunning",
      "reason": "Not initialized."
    },
    "igmpStatus": {
      "state": "notRunning",
      "reason": "Not initialized."
    },
    "rpStatus": {
      "state": "healthy"
    },
    "won": {
      "c6fa1352-7650-454c-894b-64f59011b58f": true
    },
    "elected": {
      "c6fa1352-7650-454c-894b-64f59011b58f": false
    },
    "quorum": {
      "c6fa1352-7650-454c-894b-64f59011b58f": true
    },
```

```
    "votes": {
      "c6fa1352-7650-454c-894b-64f59011b58f": {
        "8c3fdc1e-ffc0-4ea7-a4d7-4b46011490a3": 1
      }
    }
  }
}
```

Where each property is described below:

Property	Description
`version`	The version of this JSON document's schema. For internal use only.
`state`	The state of the election system, see below for election state details.
`active`	The value **`true`** if this server is active (leading) or **`false`** if its operating in standby.
`details/message`	The last election status message printed to the console on the server. See below for a description of these messages.
`details/httpStatus`	The status of the HTTP ballot transport mechanism. This is not used and will always appear in the `notRunning` state.
`details/igmpStatus`	The status of the IGMP (multicast) ballot transport mechanism. This will appear in the `notRunning` state for internal (Kubernetes-hosted) servers and any external satellite server which was started without specifying a default NIC. A warning will appear in the console of a server running without a valid IGMP leader election configuration: `WARNING ElectionConfiguration - IGMP leader election will be disabled because a default NIC was not specified. Use "agent --default-nic"to specify a default NIC.`
`details/rpStatus`	The status of the Rallypoint-based ballot transport mechanism. This should appear in the state `healthy`. If it appears unhealthy, this typically indicates a problem with the Rallypoint this server group is using for elections.
`details/won`	A map of the server group ID, to a boolean indication of whether this server won the last election.

Property	Description
`details/elected`	A map of the server group ID, to a boolean indication of whether this server was elected leader. See details above for situation where a server may win an election but not be elected leader.
`details/quorum`	A map of the server group ID, to a boolean indication of whether the election met quorum rules.
`details/votes`	A map of server IDs, to the number of votes the server submitted for itself. Recall that a negative vote count implies that the server has been taken out of service (disabled); a zero value indicates the server has lost its connection to ICE Server; and a positive value represents its election priority.

6.2.4.1 Election state descriptions The election state (show above in the `state` property) indicates the current operating mode of the server based on election status. It may have one of the following values:

6.2.4.1.1 `active` Indicates the server was elected leader and is actively performing the duties of its associated server group. When in this state, it will produce the following status message: `ACTIVE: Ballots received from <candidateCount> candidates.`

6.2.4.1.2 `standbyOutOfService` Indicates the server was taken out of server group by an administrator toggling the "Enabled" switch off in ICE Desktop settings. The server will continue to vote in elections for quorum purposes, but will never become leader, even it is the only remaining member server. When in this state, it will produce the following status message: `DISABLED: This server has been designated out of service; it will not become active.`

6.2.4.1.3 `standbyQuorumNotAchieved` Indicates that the server did not receive enough ballots from other servers to become leader. This occurs when more servers have failed than the quorum strategy allows for. For example, this state will be reached by the sole surviving member server in a group of three servers when two have failed.

When in this state, the server will produce the following status message: `STANDBY: Quorum not achieved; a leader will not be elected.`

6.2.4.1.4 `standbyGettingReady` Indicates the server has recently won the leader election and is waiting to win a consecutive election before taking over as leader. When in this state, the server will produce the following status message: `GETTING READY: Waiting to win 2 consecutive elections (won 1 so far).`

6.2.4.1.5 `standbyNotWinner` Indicates the server did not win the leader election and should remain in standby. When in this state, the server will produce the following status message: `STANDBY: Ballots received from $candidateCount candidates. Leading server: <winning server ID>` (where `<winning server ID>` is replaced by the ID of the member server which was elected leader).

6.2.5 Special considerations

6.2.5.1 Date and time synchronization Leader election in a server group requires that all member servers transmit timestamped ballots to each other. Therefore, it is crucial that all member servers agree on the time, regardless of their local time zones.

To ensure leader election functions correctly, all member servers must use the Network Time Protocol (NTP) or a similar method to stay synchronized within a few seconds of each other. If a server receives a ballot with an invalid timestamp, it will discard it, which is equivalent to the server not submitting a ballot at all. This may result in all servers being stuck in either active (leading) or standby mode, depending on the quorum strategy selected.

6.2.5.2 Loss of connection to ICE Server If a satellite server loses its connection to the ICE Server, it enters a "not ready" state. In this state, the server continues to send ballots to other servers in the group but with a vote count of zero. If another server is alive, connected to the ICE Server, and not marked out of service, it will win the leader election and become the leader. Otherwise, if there are no other servers in the group or if none are ready and in service, the current "not ready" server will remain the leader.

When a server is "not ready" but still active (acting as leader), it will *coast*, whereby:

- The server retains the configuration it had when it lost connectivity to the ICE Server. For example, if a patch server was previously set up to bridge "Channel Alpha" with "Channel Beta," it will continue to do so. However, this is a "best effort" approach. Some capabilities may cease to function when disconnected from the ICE Server, such as an ICE Archiver not archiving recordings while disconnected.

- The server will continuously attempt to reestablish its connection to the ICE Server. Once reconnected, it will resynchronize its operating configuration to update any changes made while it was offline.

6.3 ICE Agent

Satellite components typically consist of a *child* process and an *agent* process. The agent is responsible for communicating with ICE Server and managing the configuration and execution of the child process according to the state of the ICE Server. The child process is responsible for audio and media processing functions like bridging audio in a patch.

The ICE Agent is a command-line tool that runs on Windows, Linux and macOS:

- Communicates with ICE Server to determine the operating state of the component and continuously update the component's configuration in response to operating state changes. For example, as a user creates, deletes or updates a patch, the agent will respond to these configuration changes by updating the patch server's child process configuration accordingly.
- Manages the lifecycle of the child process, starting and stoping it when necessary and automatically restarting it in case of a failure or crash.
- Provides a web application for configuring tactical-mode satellite components in environments where an ICE Server is not being used.
- Manages the election of a *leader* from within a group of satellite server components.

6.4 ICE Rallypoint

Rallypoints provide a common point of connection for channels that cannot rely on an IGMP multicast network to carry their audio traffic to all interested users.

Rallypoints and their meshing ability were described previously.

6.5 ICE Gateway

The ICE Gateway component links ICE push-to-talk communications with telephone and wireline-interconnected land mobile radio systems. ICE Gateway enables ICE users to place and receive telephone calls and let telephone callers can dial into specially configured channels. When dialed into a channel, telephone callers use * to begin speaking ("take the floor") and # to relinquish it. Note that the ICE Gateway component is not used for making private calls between ICE users.

6.5.1 Interface with a call manager

ICE Gateway has two primary modes of operation that determines how it interacts with the call manager:

1. By treating the call manager as a SIP registrar, and registering ICE users and ICE channels (with dial-in enabled) with the call manager. In this mode, ICE appears to the telephone system as if it were a group of desk phones or similar endpoints.
2. By establishing a SIP trunk with the call manager. In this mode, ICE Gateway appears as an auxiliary telephony system intended to handle calls terminating on a defined set of dial peers.

Which mode you choose typically depends on the capabilities of your call manager. When both modes are supported, SIP trunk mode should be preferred as it offers a more efficient signaling path.

6.5.2 Call setup with ICE clients

While ICE Mobile and ICE Desktop offer soft-phone features—like a dial pad—administrators should be aware that Instant Connect does not use telephony-native protocols (like SIP) to signal or negotiate call setup. Instead, all telephony-specific protocols are terminated at the ICE Gateway server and "converted" into ICE-native messaging and audio.

- When a call is placed or received by an ICE client, the ICE client exchanges messages with ICE Server, which, in turn, exchanges messages with ICE Gateway to signal changes in call-state like "ringing", "busy", etc. ICE Gateway translates these ICE-proprietary messages to standard SIP messages.
- Once a call is established by both parties, ICE Server dynamically creates a full duplex channel configured to operate in unicast (Rallypoint) mode to carry the audio of the channel.
- While the user experience is quite different from push-to-talk on a channel, the ICE clients are using the same ICE media engine to convey audio between the ICE Desktop or ICE Mobile application to the ICE Gateway server. Just as ICE Gateway "translates" signaling, it does the same for audio; the audio received over the full-duplex ICE channel is mixed and transcoded into whatever media format was negotiated with the call manager.

6.6 ICE Patch Server

The ICE Patch Server is responsible for linking the audio of two or more channels together such that a talk-burst transmitted on one channel can be heard on the others, too. By default, a patch server is automatically installed inside the Kubernetes cluster when ICE Server is installed, but additional patch servers can be installed elsewhere in the network.

When a patch is created in the ICE Desktop client, the patch is assigned to a specific patch server (the choice of server is made by the user creating the patch). The patch remains associated with that patch server for its lifetime. If the patch server is deleted or otherwise becomes non-functional, ICE does not attempt to automatically relocate or migrate the patch to another server. However, in the case of a patch server crash, the process will automatically restart and resume patching (the entire restart process typically takes less than 10 seconds).

6.6.1 Patch limitations

Any two or more channels in the ICE system can be patched together. Active telephone calls can be included in a patch as well. There are two primary limitations that a user should be aware of when creating a patch:

- Channels participating in a patch must be be configured to use the same duplex mode: half-duplex channels may only be patched with other half-duplex channels; full-duplex channels must be patched with other full-duplex channels. Telephone calls are treated as full-duplex channels.

- A channel (or telephone call) cannot participate in multiple *active* patches simultaneously. This restriction exists to assure that audio loops cannot occur in the network. Note that a channel may, however, participate in multiple inactive patches (enabling a patch administrator to stage various patch configurations ahead of their needed use and activate them only once necessary).

Aside from these limitations, there are no restrictions on which channels can be patched together. In particular, it's worth noting that channels do not have to be configured to use the same codec, nor do they have to use the same encryption mode or key. Be aware, however, that patching an encrypted channel with an unencrypted channel has the effect of making encrypted audio available "in the clear" as its delivered on channels configured without encryption.

6.6.2 Preventing audio loops

An audio loop exists when a cycle (an infinite loop) is created in the path of audio traversing the system. Great care must be taken to prevent audio loops since they have the potential to consume all available network bandwith, effectively creating a denial of service scenario not just for ICE users, but for any resource reliant on the network. Instant Connect prevents audio loops by preventing a channel from being an active participant in multiple patches at the same time.

Imagine a scenario where channel Alpha is patched with channel Bravo; channel Bravo is patched with channel Charlie; and channel Charlie is patch with channel Alpha (`Alpha <-> Bravo`, `Bravo`

`<->` `Charlie`, and `Charlie` `<->` `Alpha`): A user who transmits on Alpha will have their audio stream retransmitted on Bravo; Bravo's stream will be retransmitted on Charlie; Charlie's stream will be retransmitted on Alpha; and around-and-around we go, quickly consuming all available bandwidth on the network and bringing the system—and likely other systems—to a halt.

Suffice it to say that bridging/patching systems must be *extremely* careful not to allow such configurations to exist (a problem known as *loop detection*). ICE provides a simple solution to this problem, one that is both easy for administrators to understand and to correctly implement in software: Any channel can participate in only one patch. In the example above, the loop is created by allowing Alpha to participate in two patches: The Alpha/Bravo patch and the Alpha/Charlie patch. The solution is disallow the second patch.

Note that disallowing a channel to participate in two patches simultaneously *does not* mean that a channel cannot be patched to two other channels. A single patch may bridge any number of channels. Bridging audio between Alpha, Bravo and Charlie can be accomplished by creating a single patch containing all three channels in lieu of three separate patches.

6.6.3 Audio bridging

As described previously, Instant Connect Enterprise is architected around the principle that as little audio processing be done in the network as absolutely needed. However, in the case of patches, a design decision was made to centralize patching so that a patch could outlive the administrator or device that created it.

Recall that ICE clients, and not a centralized media server, are responsible for mixing the audio of simultaneously received audio together. Furthermore, clients are able to receive and decode audio streams comprised of heterogenous CODECs, too (that is, each talker on a channel could transmit using a different CODEC). As a benefit of these capabilities, the patch server does not need to decode or mix audio streams together. Instead, when processing unencrypted audio streams, the patch server joins each channel participating in the patch and simply retransmits audio packets received on one channel to all the other patched channels. For encrypted channels, the process is slightly more CPU-intensive: the patch server must decrypt and re-encrypt traffic as it gets mirrored on participating channels.

In order to use patching functionality, channels should have ICE Rallypoint enabled. The patch server functions like a client in that it connects to Rallypoints to receive and transmit patched audio. This implies that the patch server must have reachability to any Rallypoint used by a channel.

> **Note:** Patches with multicast channels is not supported.

6.6.4 External patch server deployment

A patch server can run inside the Kubernetes cluster, or outside on an external server host. When deployed outside of Kubernetes, the patch server is delivered as a child process of ICE Agent. Install ICE Agent on Linux, Windows, macOS or Docker, and start the patch server using the `agent external patch` command (consult the product guide for details).

The agent connects to the Server Bridge component in ICE Server via a mechanism similar to the one used by ICE Mobile and ICE Desktop clients. Once the patch agent registers itself with the ICE Server it begins receiving patch configuration messages intended for this patch server instance. The agent is responsible for monitoring configuration changes made on the ICE Server and updating the configuration of the bridging component accordingly. The two processes communicate with each other through configuration and status files written to a common configuration directory:

- The agent writes `conf/ice_engagebridge_bridge.json` with a set of patches (and their constituent channel configurations) and `conf/ice_engagebridge_config.json` with process-level configuration.
- The bridge writes `status/ice_engagebridge_status.json` with the patch server's operational report (viewable in human readable form in ICE Desktop "Patch Servers" settings).

The bridging component is designed to listen for changes to `ice_engagebridge_bridge.json` and `ice_engagebridge_config.json` and react accordingly. Similarly, the agent component listens to status changes written to `ice_engagebridge_status.json` and publishes them to ICE Server where they can be viewed in ICE Desktop. Changes to configuration written by the agent do not produce an outage on existing patches. That is, an administrator is free to create, delete or update patches without concern for disrupting other patches active on the system.

6.7 ICE Static Reflector

The ICE Static Reflector "reflects" audio traffic from unicast channels (those configured to use a Rallypoint) onto an IGMP multicast network. Static reflectors are primarily used to establish system interoperability, like bridging multicast traffic from a land mobile radio gateway or IP desk phone into an ICE push-to-talk channel.

Recall that ICE clients (such as mobile and desktop apps) can exchange multicast voice traffic with non-ICE entities, provided those entities support industry standard protocols such as RTP and ICE-supported CODECS like G.711, AMR, or Opus.

A common use case is when you have an entity such as a two-way radio gateway that "speaks" multicast and you and need ICE entities to talk to that system. This is easily accomplished by setting up

a channel on ICE to use the codec the gateway is configured for, and to use the same multicast IP address and port configured on the gateway. Assuming the multicast flows cleanly between the gateway and the clients then everything works fine.

The following diagram illustrates a setup in which we have ICE Mobile and ICE Desktop clients on the same multicast network as the gateway. And the land mobile radio gateway is configured to forward a single talk group/channel/frequency on the radio system to bi-directional multicast that an ICE channel has been configured to use.

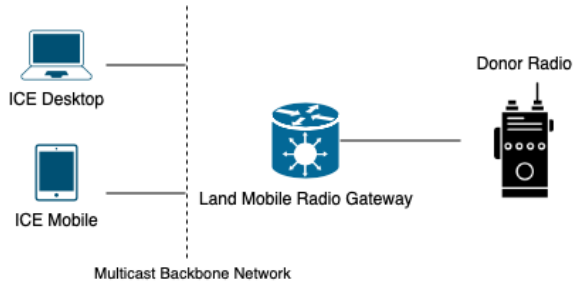

Figure 7: Radio integration with multicast

This setup works *only* if the ICE channel has been configured for multicast. In environments where multicast cannot be by all clients communicating on the channel (because, for example, some users on the channel are connecting from the open Internet), we need a system component to connect the multicast traffic produced by the radio system to the unicast/Rallypoint traffic produced by ICE. This component is the *ICE Static Reflector*.

As illustrated below, the ICE Static Reflector is placed on the same network as the radio gateway. The reflector connects to ICE Server (not illustrated) to receive information about what channels it should be reflecting and how to connect to whichever Rallypoint (or Rallypoints) are used by those channels.

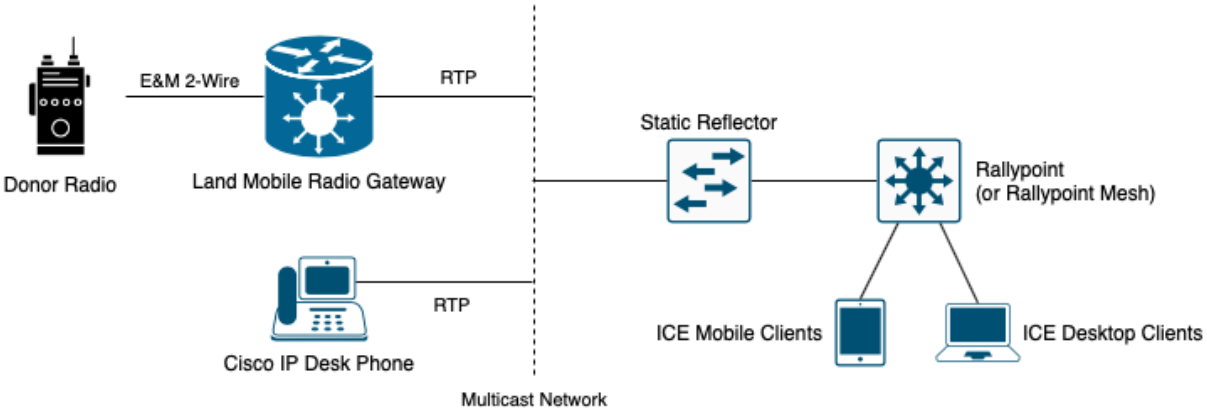

Figure 8: Radio and telephony interoperability with static reflectors

Understand that multiple reflectors can be configured in your network, and that a given channel can be reflected on multiple reflectors at the same time. By doing so, multicast traffic can be picked up at one site, carried across a WAN link, and dropped onto the multicast network of a remote site. This is sometimes called a *MUM trunk* (multicast-unicast-multicast) and can be used to interconnect disparate radio systems with or without linking them to ICE users.

6.7.1 External reflector deployment

A static reflector can run inside the Kubernetes cluster, or outside on an external server host. When deployed outside of Kubernetes, the static reflector is delivered as a child process of ICE Agent. Install ICE Agent on Linux, Windows, macOS or Docker, and start the reflector using the `agent external reflector` command (consult the product guide for details).

As is true for patch servers, the agent assumes responsibility for communications with the ICE Server and supplying configuration data to the reflector. Similarly, the two containers communicate with eachother through configuration and status files written to a shared mount point :

- The agent writes `conf/ice_reflector_peers.json` with a set of channels to be reflected and `conf/ice_reflector_config.json` with process-level configuration similar to that provided for patch servers.
- The reflector writes `status/ice_reflector_status.json` with the server's operational report (viewable in human readable form in ICE Desktop "Static Reflectors" settings).

Under the hood, a static reflector is a specially configured Rallypoint. You're advised to ignore this technical implementation detail, though. These "Rallypoints" should not be substituted for real Rallypoints in your network.

6.7.1.1 A word about multicast interfaces

When connecting to a routable address on the network, the host operating system typically chooses the physical network interface that will be used to facilitate the connection (for example, wired Ethernet versus WiFi). The choice can be made on the basis of any number of variables—detected connection speed, latency, user preference, etc.—but one element the operating system knows for certain is whether a given network interface can reach the requested address. Interfaces that cannot reach the destination are quickly disqualified when making this choice.

Multicast poses a unique challenge here: Any multicast-range IP address is joinable from any network interface on the host computer (this is not strictly true as some network interfaces do not support IGMP multicast, but we'll ignore that for this discussion). In other words, any network interface can "reach" any requested multicast group address. Unfortunately, because each network typically has its own multicast universe, joining `239.1.1.1` from the WiFi network will not necessarily yield the same results as joining `239.1.1.1` from the wired Ethernet interface. Radio multicast traffic made available on the wired Ethernet network by a radio gateway will likely not be "seen" from the WiFi interface.

Complicating this matter further, the same multicast group address may be used for different purposes on different networks. `239.1.1.1` on the WiFi network might yield radio traffic from the "Alpha" channel and `239.1.1.1` on the Ethernet network might be used for "Bravo" channel. Which channel's traffic did you intend to reflect when you asked the software to join `239.1.1.1`?

> **There is simply no way for ICE to know which is the "correct" multicast network interface.**
>
> It is incumbent on the system administrator to understand these details and choose the correct interface for their environment. Most reports of reflected multicast "not working" are caused by a network interface misconfiguration.

Lastly, be aware that a single instance of ICE Static Reflector can connect to only one network interface. If you intend to reflect traffic from multiple network interfaces, you will need to deploy multiple reflectors—one for each interface.

6.7.1.2 Multicast and Docker

Typically, a Docker container does not have direct access to the host machine's network. Software running *inside* a Docker container would not be reachable by a client running on a machine elsewhere on the network. Docker's *software defined network* must be configured explicitly to forward the incoming connection from the host machine into the server software's Docker container. This becomes quite complicated, especially when dealing with multicast and an indeterminate range of IP addresses that the software might expect to listen on.

Instead, when running a static reflector inside a Docker container, it's recommended to configure the container to run in *host networking mode* (`--net=host`). This makes the reflector appear as though

it were running natively on the host system with full access to the underlying system's network hardware and eliminating a great deal of complexity.

Unfortunately, as of this writing, Docker supports host networking mode only on Linux. Attempting to run this software on macOS or Windows might *appear* to start and run successfully, but audio will not flow.

Consult the ICE Agent product guide for additional details about using Docker with the static reflector.