LiveSwitch

White Paper

Real-Time Messaging

by Anton Venema

Chief Technology Officer

info@liveswitch.io

1-888-379-6686

LiveSwitch

Anton Venema
Chief Technology Officer

Anton is a leading expert on Real Time Communications solutions, and the visionary lead architect behind IceLink, WebSync and LiveSwitch.

Real Time Messaging for Chat and More

Notifications, Chat and Messaging for WebRTC Apps

Introduction

Nobody would argue that WebRTC has had a significant impact on industries that depend on real-time communications. From education to healthcare to telecommunications, WebRTC jumped into the forefront of business development plans as a cost-effective and user-friendly option for video streaming. In a world where plugins were viewed increasingly as annoying and intrusive security risks, web developers suddenly had a brand new plugin-free tool in their belt allowing them to create immersive applications that brought people together in new and exciting ways.

It sounds great (and it is), but there is a reason that it took so long for the web development community to agree on a standard for peer-to-peer media streaming. It's hard - very hard. Streaming live video from a camera to another device may sound simple, but there are many moving parts in a real-time peer-driven media application. Networks are messy and complicated to navigate. Codecs are often proprietary and incompatible. Each application's requirements are as unique as the next. There is a continual drive to reduce connectivity time and improve performance. It has been years since WebRTC first surfaced, and the web community is still not yet in agreement on all points.

In this series of white papers, we will examine WebRTC and ORTC in detail in the hopes of coming to a better understanding of the technology itself. We will look at the technologies used, the design decisions made, the impact on real-world applications, and how it has and is evolving into new areas.

Real-Time Messaging

An often-overlooked and perhaps under-appreciated feature, sending and receiving text-based messages is a critical piece of many WebRTC-based applications. Audio and video tend to steal the limelight, but the ability to exchange messages adds an extra layer of expressiveness and interactivity that is simple and reliable, even on old devices and slow networks.

In some apps, like Slack, text messaging is the primary feature, with audio and video taking a secondary role.

In most cases, the signaling system used to establish WebRTC peer connections can also be used to send arbitrary message payloads among clients in a call or conference. LiveSwitch, for example, uses WebSync as

its internal signaling system, which also doubles as a broker for messages, allowing text to be published to an entire channel or to a specific user, device, or client.

There are a variety of use cases for messaging, but most of them will fit into one of three categories - chat messages, system messages, or application messages.

Chat Messages

The most common type of text message is a chat message. Chat messages originate from the users of your system, and are typically delivered to either another user or group of users in a conversational style. These conversations can be:

1. Private, or invite-only, as is the case with iMessage.
2. Semi-private, where users in the same account can join or leave at will, as with Slack.
3. Public, where any user can join or leave, like IRC.

Chat messages are typically persistent, in the sense that they need to be stored in an application database. This permanent storage is required to provide context when a user opens their app on a new device. Unless the most recent messages are displayed, it is difficult for a user to tell where they were in the conversation when they last left.

How messages are stored, and for how long, is largely an application-level decision. Some applications may want messages to be encrypted when stored for privacy reasons, whereas others might have a legal basis to avoid storage completely. Some applications may choose to retain all messages ever sent, while others might only retain a fixed number to control storage costs.

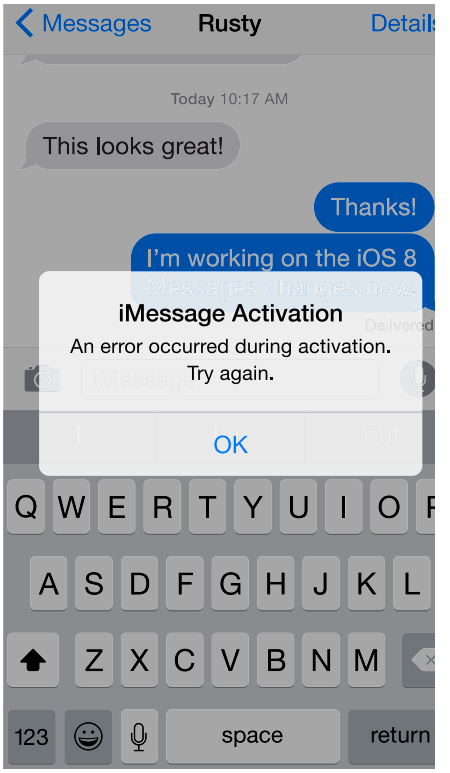
LiveSwitch and WebSync both leave these decisions to the application. Messages are kept in memory for only as long as is necessary to complete delivery. Storage and retrieval of historical messages is the application's responsibility, with simple hooks in place to make this easy, and with no restrictions on business logic.

System Messages

System messages bear much in common with chat messages, but unlike chat messages, they originate from the server or a bot. System messages could be as simple as a one-way welcome message sent to new clients joining a channel for the first time, or they could be as complex as a series of statistical reports for logging. Most of the time, system messages are simply notifications letting users know about some change of state in the application.

System messages can be:

1. Application-wide, targeting all users, as would be the case with a system downtime or maintenance notification.
2. Channel-wide, targeting users in a particular channel/ room/session, as would be the case with an end-of-session notification.
3. User-specific, as would be the case with a ban notification from an administrator.
4. Device-specific, as would be the case with a notification that a newer version of your client app is available in the device app store.
5. Client-specific, as would be the case with a kick notification sent to an app left idling in an active session.
6. Connection-specific, as would be the case with a mute or unmute notification sent to clients receiving media from the stream.



Like chat messages, system messages can be persisted, or they can be ephemeral, only delivered to active clients with no way to retrieve afterwards. Again, both LiveSwitch and WebSync leave this decision to the application developer.

Some system messages, like maintenance notifications, could be sent over a cloud messaging service as well, like Apple's Push Notification Service (APNS), Google Cloud Messaging (GCM), or Firebase Cloud Messaging (FCM), to ensure delivery to all devices where the client app is installed, even if the app is not active.

Application Messages

Application message originate from users of your application, but are not chat-based. For example, a simple shared whiteboard application could use the signaling system to send each stroke on the board as a message that includes the coordinates, colour, and stroke characteristics. An online chess platform might use application messages to send move information between participants.

Choosing how to send application messages is a bit of a gray area.
They can easily be sent over the signaling engine in most cases, but in some cases, using WebRTC data channels would result in an improved user experience and/or more optimal use of server resources. For

example, while sending player moves in a chess game over the signaling engine is quite practical, player moves in a real-time racing game are much more frequent and require lower latency for a good gameplay experience, and so data channels make more sense for that case.

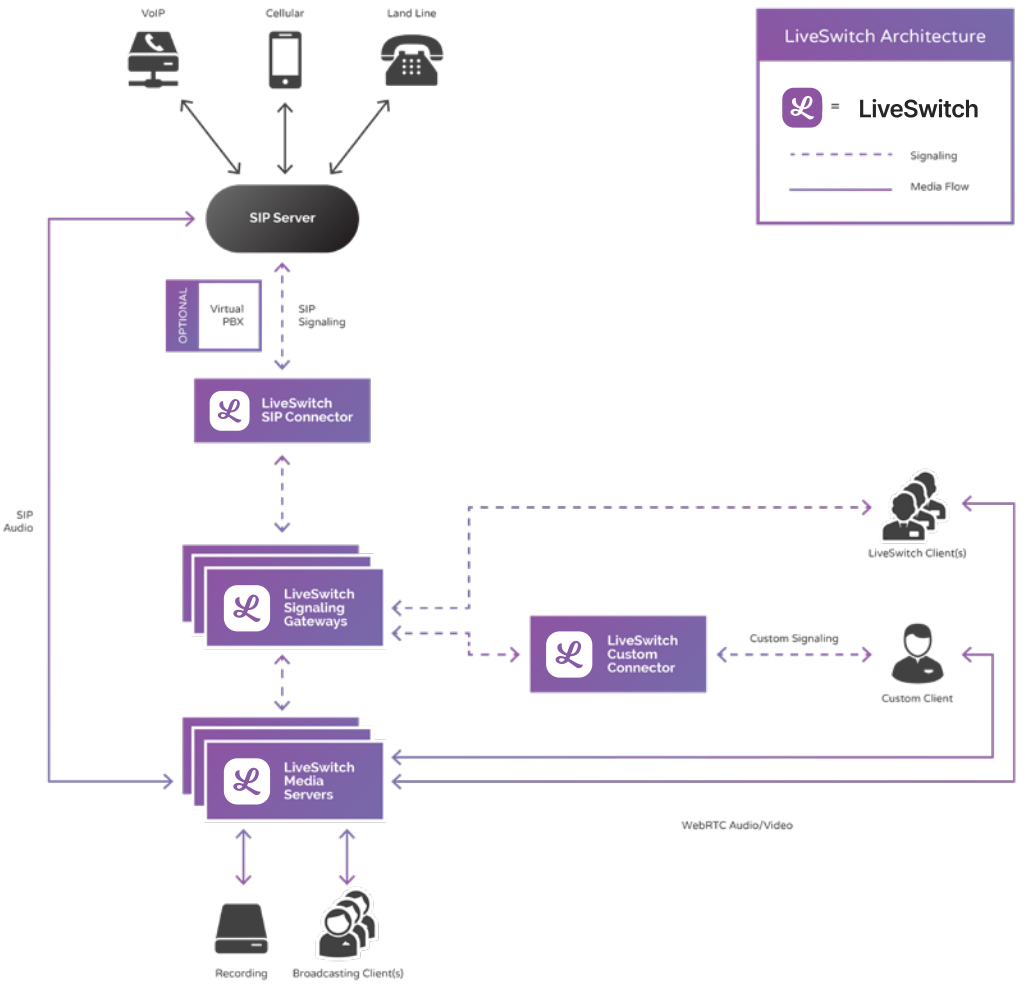
Even the whiteboard example used above could benefit from the lower latency and streaming nature of data channels. Sending the stroke data piece by piece as it happens instead of as a single message after the stroke completes allows the receiving end to reconstruct the stroke piece by piece as well, changing the user experience entirely.

In this case, assuming we want to persist the whiteboard drawings, we could make use of both techniques. Data channels stream the stroke data "as it happens" for the live user experience, while signaling sends the completed stroke to the server to be stored for late-joining participants or historical viewing.

Wrap-Up

Hopefully you have found this informative in understanding the different types of messaging and how they can be used in real-time applications. If you are in the process of building an application that can benefit from the types of messaging described here, consider using LiveSwitch.

Building on the success of IceLink and WebSync, LiveSwitch can help you build your real-time, WebRTC-compatible application quickly and effectively. With thoughtful support for text messaging, audio/video streaming, and channel-based communications, our products are designed to empower your developers to do more and in less time.



Our client SDKs feature a consistent, powerful API and support for popular development frameworks/platforms like .NET, Java, macOS, iOS, Android, Windows Phone, Xamarin, Unity, and of course JavaScript. The powerful pipe-driven media engine in LiveSwitch and IceLink allows you to do just about anything imaginable with your audio and video data, and the flexible publish/subscribe architecture in LiveSwitch and WebSync lets you deliver real-time text and binary messages to your entire client base quickly and efficiently.