# WebRTC 视频会议的轻量级呼叫信令与对等控制

Kundan Singh Intencity
Cloud Technologies 美国加利
福尼亚州旧金山
kundan10@gmail.com

## 摘要

我们介绍了我们基于 Web 的多方视频会议应用的软件架构与实现。该应用不使用媒体服务器。对于呼叫信令，它可以通过轻量级通知服务器，利用现有推送通知传递信令；也可以使用电子邮件，进一步摆脱对该服务器的依赖。对于会议控制和数据存储，它会为参与通话的客户端建立一个对等网络。我们的原型客户端网页应用可安装为浏览器扩展，也可作为渐进式 Web 应用安装在桌面设备和移动设备上。该应用使用 WebRTC 数据通道和媒体流作为控制路径和媒体路径，实现具备音频、视频、文本和屏幕共享功能的全功能视频会议。我们在创建这一轻量级或无服务器系统时遇到的挑战及采用的技术，对其他旨在节省多方视频通话服务器维护或付费订阅成本的低端 WebRTC 应用也有借鉴意义。

## 关键词

WebRTC、软件架构、视频会议、端点驱动、点对点、无服务器、JavaScript。

## 1. 引言

WebRTC，即网页实时通信 [1][2]，是一组标准 API（应用程序编程接口）和协议，用于在网页浏览器实例之间建立低延迟媒体路径。这些浏览器客户端需要一个通知服务器，以便在客户端之间交换特定的信令消息 [3][4]。对于典型的多方音频/视频会议，它还可选择使用媒体中继、媒体网关和/或媒体服务器来实现分布式媒体路径。

此类系统的很大一部分成本通常用于运营托管在云端／数据中心的这些服务器。此外，由于应用逻辑复杂，构建此类服务器的资本支出也很高。这种复杂性很大程度上源于专有厂商特定实现，以便严格控制会议中的用户访问、信令消息和媒体服务器。如果我们能够完全避免这些重型应用服务器和媒体服务器及其相关成本，会怎样?

[CC BY] Permission to make digital or hard copies of all or part of this paper is granted provided that copies bear this notice and the full citation on the first page.
Copyright © 2026, Kundan Singh, kundan10@gmail.com

我们描述了一种端点驱动的软件架构和实现，它能够在很少或完全不依赖服务器的情况下实现多方视频通话。它使用免费可用的移动端和网页推送通知进行呼叫信令，以在两个端点之间建立 WebRTC 数据通道。后续信令（例如媒体路径协商）通过该数据通道完成。随着更多用户被邀请加入通话，它会创建一个由使用成对数据通道参与的客户端组成的点对点（P2P）网络。它利用该 P2P 网络，完全在端点中存储和同步会议数据及相关的会议应用逻辑。

我们开发的客户端网页应用概念验证实现 [5] 名为 *Ezcall*，如图 1 所示，可安装为浏览器扩展或渐进式 Web 应用（PWA）。默认情况下，我们的应用使用一个轻量级通知服务器，其 PHP 代码仅有几百行。此模式使用 Google 的 Firebase Cloud Messaging （ FCM ） [6][7] 进行推送

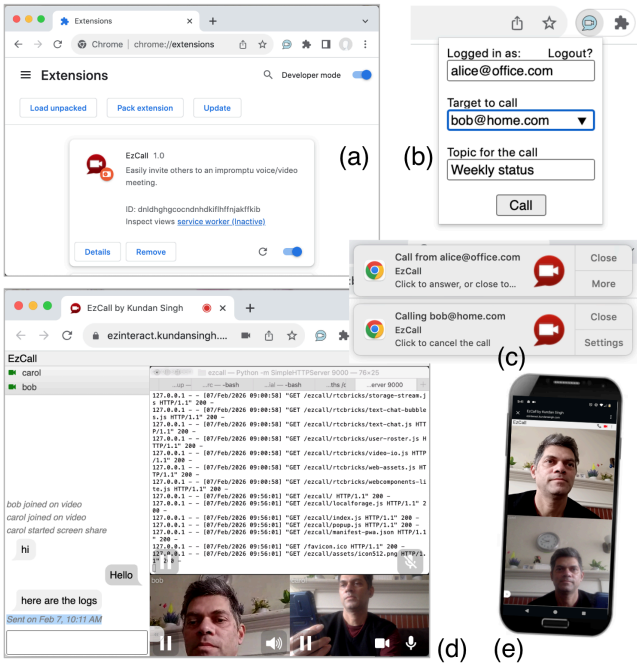



**图 1** Ezcall 截图：(a) Chrome 中显示应用的浏览器扩展页面；(b) 点击扩展程序图标后弹出的发送呼叫邀请窗口；(c) 来电和去电邀请的通知；(d) 浏览器中加载的封装应用内的视频会议 Web 应用，提供音频、视频、屏幕/应用共享和文本聊天；(d) 已安装的移动应用（渐进式 Web 应用）中的视频通话。

通知。用户也可以选择在客户端应用中使用电子邮件或即时消息进行呼叫信令，从而移除对该服务的依赖。在这种模式下，初始数据通道设置需要发送一条消息并接收一条响应，两者均包含一些呼叫信令数据。

我们的实现完全采用 JavaScript 编写，不依赖任何客户端框架。初始呼叫建立通过推送通知或电子邮件完成。除从静态网站首次下载并安装 Web 应用，以及可选使用轻量级通知服务器外，运行时无需依赖其他持久化服务。

本文结构如下：第 2 节介绍背景和相关工作。第 3 节深入描述架构和实现细节，第 4 节讨论设计决策、安全方面、其他挑战以及实用技术。第 5 节总结全文并提出未来工作的构想。

## 2. 背景和相关工作

典型的视频通话或会议包含三个步骤：查找、会话协商和维护（图 2）。首先，在查找阶段，呼叫方通过呼叫邀请联系已注册的被叫方，被叫方应答后即可加入通话；或者，组织者安排会议并通知参与者，例如发送日历邀请，以便他们加入会议，或打开同一网页以彼此连接。这一步要求用户能够通过某种方式联系到，和/或连接到网络上的某个应用。

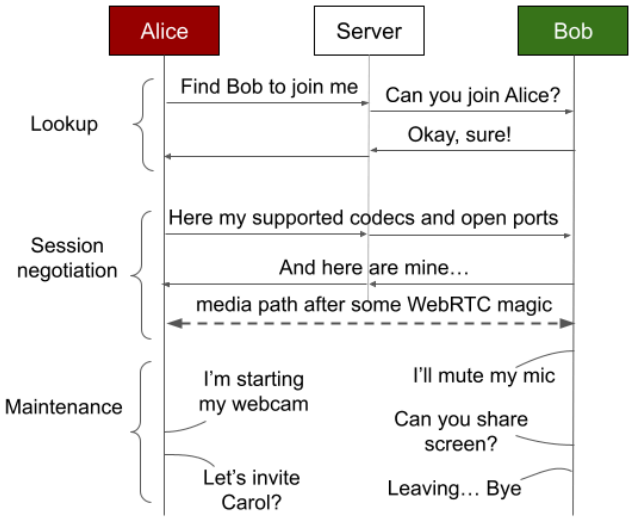



**图 2** 基于 WebRTC 的视频通话主要步骤。

其次，在会话协商过程中，两个端点，或一个端点与媒体服务器通过交换信令消息来建立媒体路径。SDP（会话描述协议）提议-应答以及渐进模式下的 ICE（交互式连接建立）候选项可能需要交换多条消息 [2]。不过，至少需要交换两条消息，且两个方向各一条；在提议-应答中不使用渐进式 ICE 即可实现这一点。

前两个步骤可以像传统的基于 SIP/VoIP（会话发起协议/互联网语音）的呼叫流程那样合并。第三个步骤是对正在进行的呼叫和会议进行控制与维护，例如在参与者加入或离开时通知端点，或添加或移除媒体通道，例如用于文本或内容共享的媒体通道。

WebRTC [1][2] 仅处理上述第二步，并有意将第一步排除在协议规范之外。现有的 Web 视频会议应用采用了多种技术；有些使用会话发起协议 (SIP) [8] 或其他开放协议，但许多应用使用专有机制，以便严格控制应用逻辑。此外，移动应用通常使用推送通知或云消息传递 [6] 来发送初始来电通知，从而唤醒应用并连接到应用服务器，以执行后续步骤。

我们将此方案扩展为在移动设备和桌面设备上双向使用推送通知，以建立控制路径的初始数据通道。Google 的 Firebase 云消息传递 (FCM) 提供了一个包装器，可在多个移动和桌面平台（包括安卓、iOS 和 Web）上发送推送通知。它使用 Web 服务工作线程 [7], ，这是一种在浏览器后台运行、与网站关联的进程，其生命周期由浏览器控制，但收到推送通知时会自动激活。浏览器扩展程序和已安装应用（渐进式 Web 应用）也可以使用它。渐进式 Web 应用 [9] 是一种可安装在桌面或移动设备上的 Web 应用，可根据需要在离线模式下运行，其运行方式和表现与原生应用相同。

视频会议通常需要服务器，不仅用于呼叫信令协商，也用于媒体路径 [3]。媒体中继可为位于受限防火墙或 NAT（网络地址转换器）之后的客户端提供媒体路径。媒体服务器可促进多方会议中的媒体分发，并提供呼叫录制或转录等其他功能。当媒体路径中使用位于公共互联网的媒体服务器，且该服务器支持对称 RTP 和 ICE lite 时，通常无需媒体中继。媒体服务器既可以充当网关，用于转码或与其他系统互操作；也可以充当 SFU（选择性转发单元），将某个客户端的媒体分发给一个或多个其他客户端；还可以充当 MCU（多点控制单元），先混合多个客户端的媒体再发送给各客户端；也可以组合使用这些方式，例如转发视频但混合音频 [10][11][12]。

利用 SFU 和 MCU 功能，可以构建多种媒体路径拓扑，规模可从两方或三方通话扩展到大型会议。此外，对于小规模会议，运行在浏览器中的 Web 应用可以充当 SFU 或 MCU，取代专用或托管的媒体服务器 [13][14]。这些技术可直接应用于我们的系统。

端点驱动的视频会议系统在端点中实现会议与呼叫控制应用逻辑


© 2026, Kundan Singh

并使用实时数据库服务进行结构化数据存储，以及通知数据变更 [15][16][17]。我们进一步拓展这一思路，避免使用服务器，并在参与的客户端组成的非结构化P2P网络上实现此类数据存储。

总而言之，我们项目的所有组成部分——用于呼叫信令的推送通知/云消息传递 [6][18][19], 用于会话协商的电子邮件或即时消息 [20][21], 与数据存储分离、由端点驱动的呼叫/会议应用逻辑 [16][17][22], 用于呼叫与会议控制的 P2P 网络 [29][30], 以及 Web 应用中的无服务器媒体路径 [13][14] ——都已分别有人提出或研究。我们的工作重点在于将这些技术结合起来，打造出具有吸引力的软件架构和切实可行的实现。

## 3. 软件架构与实现

我们首先介绍端点软件的高层框图，然后详细说明通知服务，包括其数据库模式和 API。接着介绍基于推送通知的呼叫信令，包括示例消息流和状态机。我们还将展示如何使用电子邮件实现无服务器呼叫信令。随后介绍多方会议的组织流程，包括共享数据模型以及音频/视频路径的建立。然后说明如何在参与呼叫的客户端之间构建点对点（P2P）网络。最后，我们将探讨会议中实时媒体分发的各种媒体路径拓扑，包括全网状、集中式和混合架构。

### 3.1 端点软件框图

图 3 展示了端点中软件的分层架构。Ezcall 是一个 Web 应用，可以作为网页打开，也可以安装为浏览器扩展或应用（渐进式 Web 应用）。它使用推送通知或最终用户收发的电子邮件/即时消息进行信令。对于前者，它使用 Firebase 云消息传递，以及一个轻量级通知服务器，名为 contacts.php，如下所示。

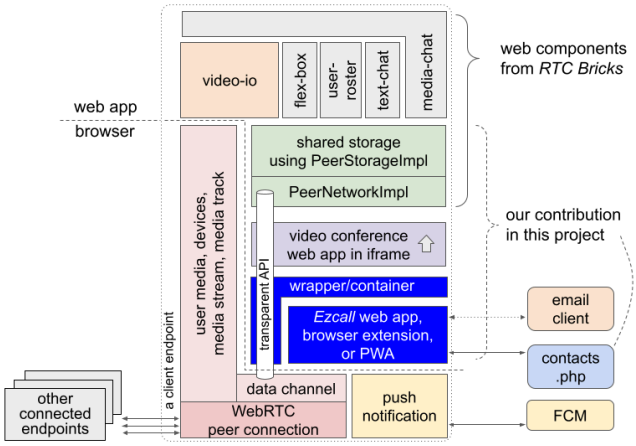



**图 3** 端点软件中各层组件的框图，同时标示了本项目的贡献。

在呼叫信令期间，它会启动一个新的浏览器标签页，其中运行一个名为 wrapper/container 的 Web 应用，以便与远程端点协商并建立 WebRTC 对等连接和数据通道。在多方通话中，wrapper 管理多个数据通道，每个连接的远程端点或对等方对应一个。

包装器还会协商并在该浏览器标签页的iframe中加载一个或多个 Web 应用。它为这些应用提供垫片 API 层，使它们能够通过数据通道与其他端点收发消息。用户可以配置所协商的应用。默认情况下，包装器会加载我们使用*RTC Bricks*项目中的 Web 组件构建的视频会议 Web 应用media-chat-peer.html [22]。该项目包含若干可用于构建实时多媒体通信与协作应用的 Web 组件。它采用基于资源的软件架构，将应用逻辑与数据存储服务分离，同时将应用逻辑保留在端点中 [16][17]。

我们用新的 P2P 网络及其配套的分布式数据存储，替换了原有的默认集中式数据存储。我们构建了两个组件：PeerNetworkImpl 和 PeerStorageImpl；前者实现非结构化 P2P 网络节点，后者则利用该节点实现共享数据存储。我们保留了shared-storage组件现有的 API，因此该项目中所有现有的 Web 组件都能继续使用我们的 P2P 存储。media-chat-peer和包装器协同向PeerNetworkImpl提供类似 WebRTC 数据通道的 API。这样一来，我们的新组件便可轻松复用于其他使用数据通道的项目，而不仅限于Ezcall这一用例。

### 3.2（可选）通知服务

Although in our application, we have email and FCM, we have excluded it from the scope of Ezcall. 在我们的应用中，我们将其排除在 Ezcall 的范围之外。可以使用任何现有机制进行注册，即在通知服务器上建立 身份验证令牌 与用户标识符之间的映射。默认情况下，我们使用电子邮件地址作为用户标识符。身份验证令牌 是随机分配的 UUID（通用唯一标识符），在单次安装实例中保持固定。一个用户标识符可以关联多个令牌，而一个令牌对应一个应用安装实例。此外，客户端应用从 FCM 服务获取的 FCM令牌用于标识已安装的应用，以便发送推送通知。

#### 数据库模式

通知服务器使用以下数据库模式，追踪已注册用户的标识符、身份验证令牌和 FCM 令牌。虽然用户标识符的名称是电子邮件，但其值可以是用户希望他人用来识别自己的任何信息，例如电话号码、社交媒体资料链接，或用户公钥的指纹。

© 2026, Kundan Singh

| 字段 | Type | 描述 |
|---|---|---|
| 令牌 | 文本（键） | 注册时随机分配的 UUID<br>代表一次应用安装 |
| 电子邮件 | text | 用户的电子邮件地址或其他<br>识别 |
| 创建 日期/时间 | | 用于记录外部注册 |
| 联系人 文本（唯一） | | 当前活动的 FCM 令牌，<br>关联到安装或端点 |
| 过期日期/时间 | | 自动刷新令牌以联系联系人<br>通过端点进行映射 |

客户端-服务器 ***API***

我们的通知服务器提供 Web API，用于更新映射关系，并根据目标用户的用户标识符向其 FCM 令牌发送推送消息。登录时，客户端会在 HTTP 授权标头中发送身份验证令牌，验证成功后会收到关联的电子邮件地址，如下所示。

=> GET /contacts Authorization: Bearer <身份验证令牌>...<= { "email": "alice@office.com" }

此 API 仅验证身份验证令牌是否存在，以及它是否与已注册用户的标识符相关联。

FCM令牌用于标识可接收 FCM 服务推送通知的设备或应用，并代表用户的联系人或可联系状态。每个身份验证令牌或应用安装实例可能只有一个联系人。要注册 FCM令牌，客户端会发送 HTTP PUT 请求，请求正文如下所示。

=> PUT /联系人 Authorization: Bearer <身份验证令牌>... { "联系人": "Firebase 云消息传递令牌..." }<= { "实例": "abc12", "过期时间": ... }

服务器会在数据库中更新与请求的身份验证令牌相关联的 FCM 令牌。因此，同一身份验证令牌的新 FCM 令牌会覆盖之前的令牌。响应中包含此联系人的唯一实例标识符，发送方可使用它来指定某个联系人，而不是面向所有联系人。服务器内部通过对用户标识符、身份验证令牌和 FCM令牌的组合值计算简单的 32 位 CRC（循环冗余校验）校验和，以生成实例标识符。

当客户端要向目标用户发送推送通知时，会使用 HTTP POST 请求，并在请求正文中包含目标用户的标识符，如下所示。

=> POST /contacts Authorization: Bearer <身份验证令牌>... { "to": "bob@home.com", "data": {...} } <= { "count": 2 }

“to”属性中的目标可以是上面显示的用户标识符，也可以是带有实例标识符的特定联系人设备，例如“bob@home.com/abc12”。如果成功，响应会包含成功用于发送推送通知的目标联系人的数量。

如果失败，则会返回包含更多详细信息的 404 响应正文。请注意，如果存在多个联系人，只要成功发送给其中任意一个联系人，响应就会成功。

通知服务器收到此请求后，会验证发送方的身份验证令牌，并确认 "data" 属性是一个对象。然后，它会从数据库中查找特定目标用户的所有联系人，并尝试对每个联系人使用 FCM 推送通知 API 发送 data 值。在每次推送尝试中，它都会在数据对象中添加或替换 "To" 和 "From" 属性，分别指明目标用户的特定联系人和发送方，如下所示。这样，推送通知的接收方就能确认自己是否是正确的消息接收联系人，并向发送方的特定联系人发送响应。

=> POST https://fcm.googleapis.com... {"message": {"data":{..., "收件人":"bob@home.com/abc12", "发件人":"alice@office.com/xyz34" } }

默认情况下，我们的推送通知会在一分钟后过期，因为它仅用于实时呼叫信令。我们使用基于 OAuth 的 FCM v1 API，而非已弃用的旧版 API。

## 3.3 推送呼叫信令流程

启动 Ezcall 应用后，会显示一个界面（见图 4），提供两个选项：输入身份验证令牌登录，以便使用推送通知进行呼叫信令；或者使用电子邮件进行无服务器模式下的呼叫信令。

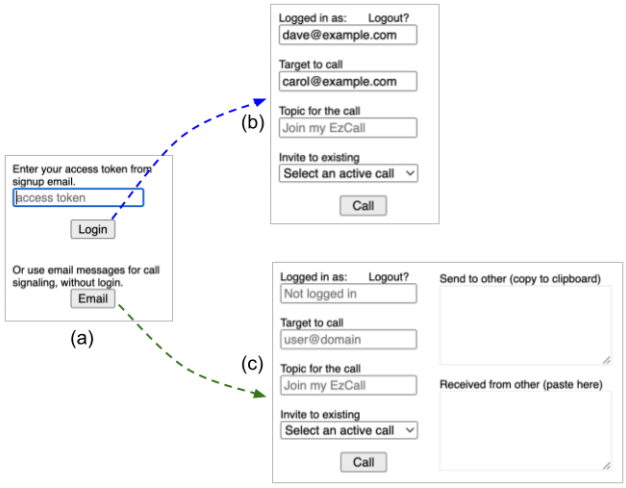



**图 4** (a) 登录（或电子邮件）选项；(b) 登录后，使用推送通知进行呼叫；或 (c) 未登录时，使用电子邮件发送或接收呼叫邀请——支持复制和粘贴消息。

如果用户选择登录，客户端会发起前文所述的登录 API，并将 身份验证令牌 保存在浏览器本地存储中，以避免日后再次提示登录。随后，它会获取 FCM 令牌（图 5 中的步骤 1），并在通知服务器上注册该令牌（步骤 2）。


© 2026, Kundan Singh

### 发送呼出通话邀请

图 5 展示了在两个端点之间建立双向数据通道的呼叫信令。在此示例中，当 Alice 向 Bob 发起呼叫时，应用会创建一个新的随机 邀请标识符。它会为此呼出邀请显示一条通知，以便用户与之交互，例如在邀请被应答或超时之前随时取消待处理的邀请。随后，应用会启动一个带有 包装器 应用的新浏览器标签页（步骤 3）。应用选取一个随机生成的 节点标识符，并将其提供给包装器。该节点标识符在整个呼叫以及 P2P 网络中用于标识此客户端端点。它在单个会议内的所有客户端之间必须唯一。应用还会创建一个新的随机会议或 会议标识符，并指示 包装器 使用此会议标识符加载视频会议 Web 应用。请注意，该 Web 应用包含一个 PeerNetworkImpl 对象，它代表 P2P 网络中的一个节点，并使用先前选定的节点标识符。最后，应用指示包装器创建对等连接，并使用此邀请标识符为新的数据通道创建会话“提议”。

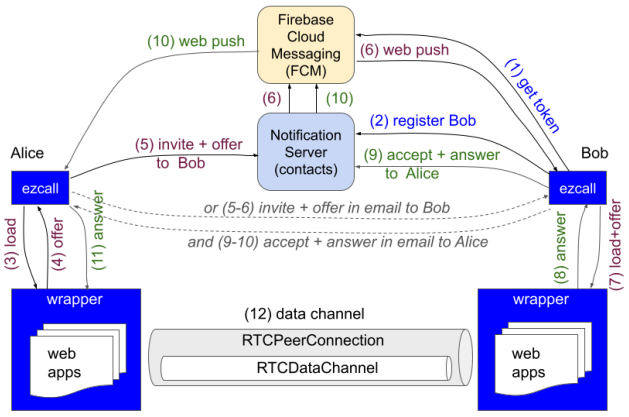



**图 5** 建立数据通道的呼叫信令步骤。

包装器使用非 Trickle 模式生成一个包含数据通道 ICE 候选项的单一提议 SDP，并将其传递给应用（步骤 4）。该消息包含邀请标识符，以便应用将提议与待处理的呼出邀请关联起来，并支持同时发起多个呼叫邀请。随后，应用会构建一条邀请消息，其中包含呼叫的所有详细信息，包括邀请标识符、会议标识符、发送方的节点标识符、可选的呼叫主题以及提议 SDP。

如果呼叫方 Alice 已登录，客户端会使用推送通知，通过通知服务器向目标用户 Bob 发送邀请消息（步骤 5–6）。推送通知会唤醒接收端 Bob 的客户端服务工作线程，并向用户显示通知（见图 1(c)）。接收用户 Bob 可以通过与通知交互来接听或拒绝呼叫邀请。如果一分钟后仍未进行任何操作，应用便会超时并移除呼叫邀请通知。

### 接受或拒绝呼入呼叫邀请

如果用户拒绝呼叫，系统会向呼叫方发送一条新的拒绝推送通知，其中包含邀请标识符等必要数据，并使用通知消息中的“发件人”值。因此，该通知会返回到发送邀请的呼叫方应用的特定实例。

如果接收方用户 Bob 接听了呼叫，应用会尝试创建数据通道的接收端。请注意，WebRTC API 无法在服务工作线程中创建对等连接或数据通道。因此，如果用户通过点击通知接受呼叫邀请，而主 Ezcall 应用窗口尚未打开，服务工作线程会先打开主应用窗口。然后，它会启动一个带有 包装器 的新浏览器标签页，并提供呼叫邀请中的必要信息，包括会议标识符、邀请标识符、节点标识符、可选主题和提议 SDP（步骤 7）。

与呼叫方一侧类似，接收方应用也会生成一个新的唯一随机节点标识符，并将其分配给每个 包装器。应用还会指示包装器使用会议标识符加载视频会议 Web 应用，并为此邀请中收到的数据通道“提议”创建对等连接和会话“应答”。包装器与呼叫方一侧一样，使用非 Trickle 模式生成包含数据通道 ICE 候选项的单个 SDP，并将其交给应用（步骤 8）。此消息包含邀请标识符，以便呼叫方一侧进行关联。

随后，应用会构建一条接受消息，其中包含邀请标识符、生成的应答 SDP 等所有详细信息，并根据初始通知消息中的“发件人”值，将其发送回呼叫方对应的联系人。接收方通过另一条推送通知将此消息发送给呼叫方（步骤 9-10）。呼叫方应用通过推送通知收到消息后，会更新现有通知，显示目标用户已接受通话，并将此应答 SDP 传递给包装器（步骤 11）。这样便完成了呼叫方一侧对等连接的提议-应答协商。被叫方一侧的协商已提前完成（步骤 8 之后）。

如果提议-应答协商成功，并且两个端点上的 包装器应用能够找到网络路径来建立 WebRTC 数据通道，那么双方都会宣布连接成功（步骤 12）。随后，他们会使用此数据通道进行后续数据交换，例如建立实际的音视频媒体路径和进行会议控制。后文将对此进行介绍。

图 6 展示了呼叫信令过程中端点上每个邀请所对应的完整状态机。请注意，拒绝和活跃状态几乎相同，并且都会在短暂超时后转为空闲状态。由于此状态机仅适用于呼叫邀请，因此它只在

© 2026, Kundan Singh

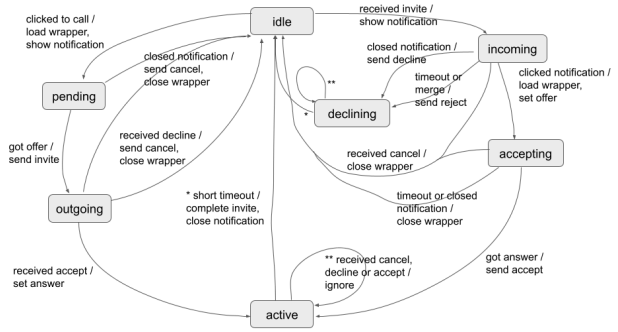



**图 6** 端点处呼叫邀请的状态机。

初始呼叫建立阶段保持活跃，而非贯穿整个通话，这与 SIP 代理上的 INVITE 事务类似 [23]。

### 将收到的邀请合并到现有呼叫

如果接收用户 Bob 在收到传入邀请时已经在通话中，他可以选择将其作为新呼叫接听（前文已述），也可以将其合并到现有呼叫中。如果 Bob 决定合并，就会向呼叫方发送特殊的呼叫拒绝响应，表明接收方将通过新的邀请回拨。在这种情况下，Bob 的应用会按照呼叫方一侧的消息流程，在现有呼叫中邀请 Alice 的特定联系人，使用新的呼叫邀请和新的状态机，但使用与 Bob 现有呼叫相同的会议标识符。

从用户体验的角度来看，如果接听呼叫时浏览器中聚焦的是一个活动呼叫，则可以自动选择合并选项。类似地，如果在浏览器中聚焦着一个活动呼叫时发出呼出通话邀请，该呼叫邀请会自动分配相同的会议标识符，以邀请新用户加入现有会议。或者，用户也可以使用特定的通知按钮进行响应（图 1(c)）。

### 将呼叫邀请分叉到多个设备

应用支持同一用户标识符进行多个登录。这在一定程度上基于 SIP 分叉代理的行为 [23]，如图 7 所示。图中显示，呼叫方 Alice 使用两台设备（x 和 y），接收方 Bob 使用三台设备（a、b 和 c）。

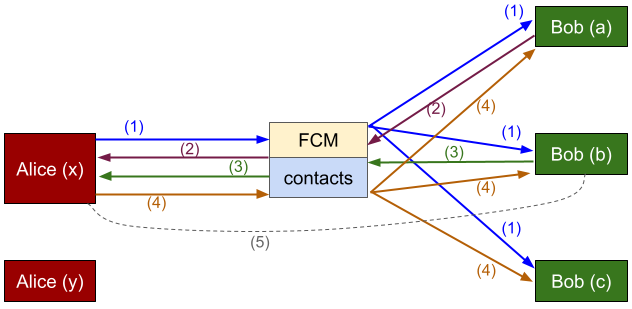



**图 7** 当两名用户拥有多个已登录设备时，建立数据通道的呼叫信令步骤。

消息由通知服务器按以下方式分叉。如果服务器检测到目标用户标识符对应多个联系人，就会生成多条推送通知，每个联系人一条。因此，Alice 发给 Bob 的呼叫邀请会分叉发送给全部三个联系人（图 7 中的流程 1）。在此示例中，第一台设备（a）超时并发送拒绝消息（流程 2），呼叫方一侧会忽略该消息。接收方 Bob 使用第二台设备（b）接受呼叫邀请（流程 3）。这会使呼叫方一侧发送取消请求（流程 4），以便所有其他待处理设备都能取消传入邀请。接受消息送达呼叫方后，两个指定设备之间会建立 WebRTC 数据通道（流程 5），并使用此前完成的提议与应答协商（流程 1 和 3）。

是否发送取消请求的决定可以基于已发送的推送通知数量，如服务器的响应中所示（流程 1 之后）。但为了简单和稳健，我们的实现在呼叫被接听或拒绝时始终发送取消请求。实际接听呼叫（或拒绝、或超时）的接收客户端会直接忽略收到的取消请求，如状态机所示。其他待处理的接收方客户端可以利用该请求来移除呼叫邀请通知。

请注意，我们将最终用户明确拒绝呼叫的情况与超时、忙线或合并等其他拒绝原因区别对待。具体而言，拒绝会导致呼叫方立即取消邀请，而其他拒绝原因则让呼叫方或最终用户决定是否取消呼叫邀请。此外，被拒绝的呼叫不会显示未接来电通知。

请注意，在图 7 中，接收方发往呼叫方的消息始终使用呼叫方设备的特定联系人，因此只会送达发送初始邀请的特定客户端（例如，流程 2 和 3）。另一方面，呼叫方发往接收方的消息，包括初始邀请（流程 1）或之后的取消消息（流程 4），默认发送到接收用户的标识符，并分发给该用户的所有联系人。

## 3.4 电子邮件呼叫信令流程

无服务器呼叫信令流程的行为与之前使用推送通知的呼叫信令流程几乎相同。两者存在一些差异，如图 5 所示，具体如下。

在之前介绍的呼叫信令流程中，第 4 步之后，如果呼叫方尚未登录，Ezcall 应用会使用“mailto:”URL 打开用户的电子邮件客户端。to、subject 和 body 中会预先填入通过电子邮件发送邀请所需的信息。系统会指示用户将电子邮件发送给目标用户 Bob（第 5–6 步）。invite 数据也会显示在应用的发送框中（图 4(c)），这样 Alice 就可以通过其他即时通讯软件发送邀请。当用户 Bob 通过电子邮件收到邀请后，他会启动自己的 Ezcall 应用


© 2026, Kundan Singh

如果应用尚未打开，他会将电子邮件正文粘贴到应用中相应的已接收消息区域（见图 4(c)）。这会在应用中触发传入邀请。其余流程，例如加载包装器、创建各种标识符，或显示有关外呼或传入通话的通知，与之前相同。

用户点击传入通话通知以接受邀请后，应用从包装器接收到应答 SDP（图 5 中的步骤 8），便会使用“mailto:” URL 打开用户的电子邮件客户端。正文中已预先填入通过电子邮件发送响应所需的信息。与之前一样，应用会在发送框中显示接受数据，方便 Bob 使用其他消息工具发送。系统会提示用户回复来自呼叫方 Alice 的电子邮件（步骤 9-10）。呼叫方 Alice 通过电子邮件收到响应后，会将电子邮件正文复制并粘贴到应用中相应的已接收消息区域。这会触发应用中的通话接受。

应用的用户界面（见图 4(c)）有两个框：一个用于复制要发送给远程用户的消息，另一个用于粘贴从远程用户收到的消息。请注意，为了成功完成呼叫信令，呼叫方和被叫方都需要各使用这两个框一次：呼叫方先发送电子邮件，再接收响应邮件；接收方则先收到电子邮件，然后发送响应邮件以接受通话。

基于电子邮件的流程与基于推送的流程还有一些不同之处，例如需要使用额外的标识符，将响应与请求关联起来。通常没有取消或拒绝流程，因为可以通过超时使邀请失效。这使得此类情况下的状态机更加简单。此外，通话邀请不会分叉。由于呼叫邀请发送到用户的电子邮件，而非直接发送到设备，接收用户可以明确选择合适的设备，将收到的邀请粘贴到该设备上，从而接受通话。

## 3.5 包装器与会议网页应用

如前所述，两个端点之间的数据通道一旦建立，尤其是由包装器建立后，初始呼叫建立便告完成（图 3 和 5）。之后，两个端点之间实际的多媒体通话建立由视频会议 Web 应用media-chat-peer完成；该应用由包装器加载到同一浏览器标签页的iframe中。

### 包装器与 *Web* 应用之间的接口

包装器和 Web 应用通过跨源 postMessage 进行通信。如果用户进行了相应配置，包装器便可在其 iframe 中从其他网站或源加载应用。包装器会通过一些 URL 参数向 Web 应用传递初始数据：data.path、data.self 和 data.displayname，分别用于标识共享数据存储中的会议资源、本地用户标识符和她的屏幕名称；topic 用于表示通话主题；id 用于标识此 P2P 网络中的节点。Web 应用加载完成后，会向 包装器 发送一条 "ready" 消息。或者，包装器也可以监听 iframe 的 "load" 事件，以确定应用何时加载完成。

请注意，包装器会维护所有处于活动状态的 WebRTC 数据通道，但这些通道由 Web 应用使用。当两个端点之间建立数据通道时，它们的 包装器应用会先交换各自的节点标识符。包装器会通过“add”或“remove”消息，分别将任何数据通道连接或断开情况通知本地 Web 应用；消息中包含另一个节点的标识符。

Web 应用负责维护 P2P 网络和共享数据存储。它通过活动数据通道与其他节点收发消息。发送消息时，它会使用“send”消息，其中包含目标节点的标识符以及作为字符串的实际消息。当包装器在数据通道上收到消息时，会通过“message”消息将其传递给 Web 应用，其中包含发送方节点的标识符以及作为字符串的接收消息。

### 对等网络和分布式数据存储

Web 应用将 PeerNetworkImpl 和 PeerStorageImpl 分别用作 P2P 网络节点和关联共享数据存储的实现。此数据存储由通话中所有已连接端点（即 P2P 网络中的节点）共享。所有端点在各自的存储中看到的数据都相同。同步可能会有轻微延迟，也可能因任何数据通道断开而导致数据存储分区。

该分布式数据存储以 共享存储 对象的形式提供给其他 Web 组件，用于接下来介绍的各种会议功能。这些 Web 组件原本是为使用集中式数据存储而设计的，现在只需替换其 共享存储 引用，就能透明地使用我们的 P2P 数据存储。

## 3.6 使用 media-chat 的多方会议

实际的多媒体通话建立和会议控制是通过 RTC bricks 项目的 media-chat Web 组件完成的。下面我们将介绍此 Web 组件的多方会议运作方式。

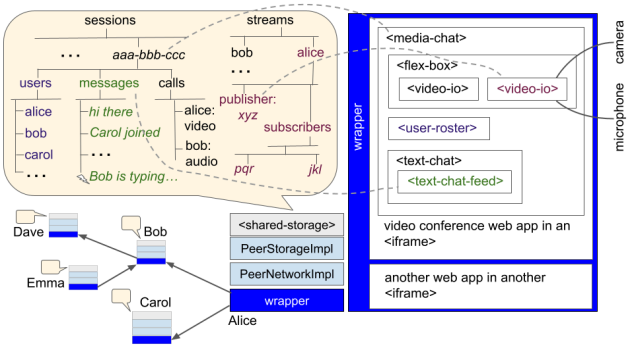



**图 8** RTC Bricks 的 Web 组件及其在多方会议共享存储中的逻辑数据结构。

© 2026, Kundan Singh

该项目中的 Web 组件使用 共享存储组件来创建和管理层次数据结构（即资源），如图 8 所示。应用可以更新资源，并在另一个端点上的其他应用更新该资源时，实时收到任何更改的通知。

一个 media-chat 实例代表多方音频、视频和文本会议中的一名用户，并绑定到一个会话资源路径，例如 /sessions/aaa-bbb-ccc，其中 aaa-bbb-ccc 是随机生成且唯一的会议标识符。该组件嵌入其他组件，例如用于排布参与者视频的 弹性布局框、用于显示参与者列表及其状态（如正在输入或网络摄像头处于活动状态）的 用户名册，以及用于多方文本对话的 文本聊天。它在自身资源路径下为这些嵌入组件提供子资源路径，例如，文本聊天 绑定到子资源 /messages，并在其下存储所有文本消息；而 用户名册 绑定到 /users，以存储通话参与者数据。/calls 子资源用于标识通话中哪些参与者已启用音频或视频，以便 media-chat 组件能够在其 弹性布局框 内添加 video-io 组件。文本聊天 组件使用 text-chat-feed 组件来显示聊天消息，并使用文本输入区域输入要发送的消息。

当用户在 文本聊天 组件的文本输入区域中输入文本消息并按下回车键时，该组件会在其资源下创建一个新的消息数据对象，例如 /sessions/aaa-bbb-ccc/messages/hya62。该对象包含消息的详细信息，例如 {"sender": "Alice", "data": "hi there", ...}。在其他已连接端点的应用中运行的 文本聊天 组件会收到此变更的通知，判定有新消息到达，并更新其 文本聊天消息流 组件，以显示来自该发送方的新消息。某些事件（例如 正在输入 指示）不会作为数据资源创建，而是作为资源路径上的显式通知事件发送，并以与资源变更事件相同的方式传递给已连接端点。

各种 Web 组件采用类似的资源操作与更新通知来实现功能完备的视频会议体验。有关这些 Web 组件行为的完整细节，感兴趣的读者可参阅 RTC bricks 项目 [22]。

## 3.7 音频/视频媒体流

视频输入输出模块负责发布或播放单个音频或视频媒体流，与端点的摄像头和/或麦克风设备进行交互，下一小节将对其进行介绍。它包含用于使用 WebRTC 发布或播放包含音频和/或视频轨道的单个媒体流的用户界面和应用逻辑。在我们的视频会议应用中，它显示单个用户的视频：可以是本地用户发布的视频，也可以是远程用户播放的视频。它还用于屏幕或应用共享。

video-io 组件连接到另一个 named-stream 组件，该组件可以通过多种方式实现，以支持会话协商。我们使用连接到 shared-storage 组件的 storage-stream 实现。它使用分层资源来表示每个命名流中的零个或一个发布者，以及零个或多个订阅者。

图 8 和 9 显示，会议中的每位用户都会根据自己的标识符创建一个命名流，例如 /streams/alice。该用户用于发布的 video-io 组件绑定到流的 /publisher 子资源。其他用户用于观看该流的 video-io 组件则绑定到 /subscribers 下的子资源。WebRTC 会话通过 SDP 提议-应答和 ICE 候选交换进行设置，方法是在这些子资源上发送包含协商数据的通知事件。与之前的数据通道不同，这里使用渐进式 ICE。根据使用音频、视频还是两者，发布者的 video-io 到订阅者的 video-io 之间建立的媒体流包含一条或两条媒体轨道。

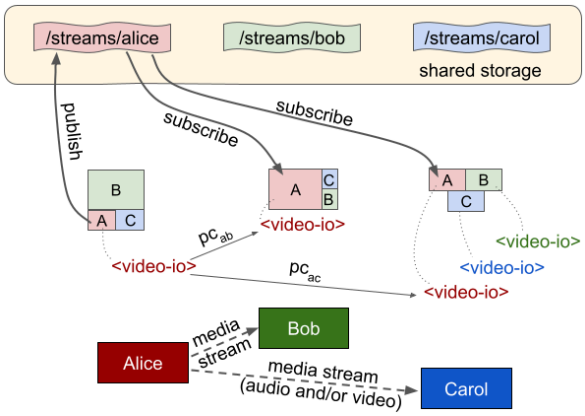



**图 9** 使用共享存储时，视频输入输出组件与命名流之间的交互及其行为

视频输入输出组件为每一对发布者和订阅者使用一个对等连接和一个媒体流。因此，在双方通话中，媒体路径上有两个用于媒体流的对等连接，分别沿两个方向传输。同样，在五方视频通话中，媒体路径共有 N(N-1) =20 个对等连接，以全网状拓扑传输 20 个单向媒体流，每个连接传输一个。

这些连接是在呼叫建立期间为控制路径建立的初始对等连接之外的额外连接。虽然可以优化并复用对等连接的数量，但我们倾向于将它们分开。将用于初始控制路径的对等连接与用于后续媒体路径的对等连接分开，可以使软件实现模块化：初始呼叫建立无需处理媒体路径拓扑或其优化，而媒体路径的建立也可以独立于控制路径进行。


© 2026, Kundan Singh

## 3.8 控制路径的点对点网络

通话参与者之间通过初始对等连接和数据通道建立 P2P 网络。这是一种无结构的自组织网络，模拟了呼叫建立过程。具体而言，该网络取决于谁邀请了谁。图 10 展示了一个由五位用户组成的网络示例：Alice 邀请 Bob 和 Carol，Bob 邀请 Dave，而 Emma 呼叫 Bob，随后将接到的来电与自己已有的通话合并。

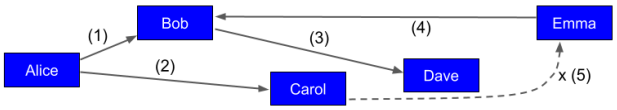



**图 10** 带有控制路径的对等网络。箭头表示呼叫邀请的方向，且数据通道为双向。

由于采用了共享数据存储，每个节点都知道控制路径的完整拓扑；如果检测到环路，便会立即解决。例如，如果 Carol 尝试在同一通话中邀请 Emma，且会议标识符与 Emma 现有通话的会议标识符相同，则网络中不会创建新的链路。但是，如果 Carol 在一个新通话中使用新的会议标识符邀请 Emma，那么这将作为另一个独立通话进行，并拥有与之前通话相分离的点对点网络。目前，我们不支持合并会议标识符不同的独立通话。

我们使用泛洪在对等网络中进行数据同步。请注意，网络中的每条链路都是双向数据通道。通常，环路不会造成危害，只会在同步共享数据存储时导致数据传输冗余。例如，当 PeerStorageImpl 检测到本地数据存储发生变化时，会将该变化发送到所有直接相连的链路。每个相邻节点又会将收到的消息继续传播到与其相连的下一个节点。我们使用随机且唯一的消息标识符以及较小的生存时间（TTL），以确保消息能够送达，或在一段时间后被丢弃。该 TTL 根据跳数设定，类似于 IP 数据包的 TTL，默认值较小。

除了通过网络泛洪进行数据同步外，该实现还允许向特定节点发送消息，例如用于音频/视频流的会话协商以及发送私人文本消息。每个节点都会维护一个简单的路由表，指明发往特定目的地的消息应经过哪个下一跳。各节点与直接相邻的节点同步路由表，并在发生任何变化时更新自己的路由表。

图11和图13展示了我们用于评估 P2P 网络上的数据同步和路由的测试应用程序。它可以在本地计算机上模拟最多 30 个节点进行测试。左上方显示的网络基于呼叫建立控制路径。所有media-chat组件中的文本消息、用户列表和视频通话状态都会同步。

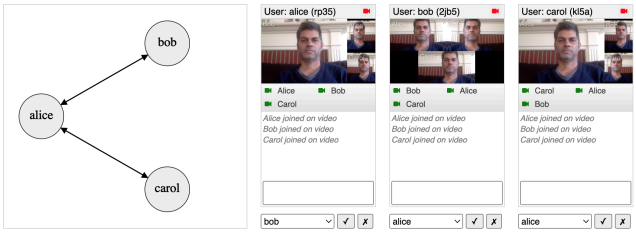



**图 11** 展示三方多媒体通话的测试应用程序。

图13展示了六位用户参与的多方会议及其中的文本聊天，并说明断开的链路如何影响会议，导致会议分区。当各分区合并时，数据会重新同步。重新同步后，使用消息标识符来避免重复。未来还可以使用消息时间戳，使所有用户看到的消息顺序保持一致。音频/视频聊天也会出现类似的分区和合并行为。

## 3.9 媒体路径拓扑

如前所述，控制路径独立于媒体路径，且媒体路径默认采用全网状拓扑。只有参与用户在多方聊天中启用音频/视频时，才会使用媒体路径。

图 12 对比了使用应用层 MCU 和 SFU 功能的几种媒体路径拓扑（类似于 [13]）。全网状拓扑使参与者之间的媒体路径延迟较低，因此适用于实时对话。然而，每个节点的网络用量会随参与者数量线性增长，也就是说，每个节点接收 N-1 条媒体流，并发送 N-1 条媒体流。低端网络的用户很容易遇到上行带宽拥塞。此外，单独的媒体流还可能增加机器 CPU/GPU 的编码器处理负载。

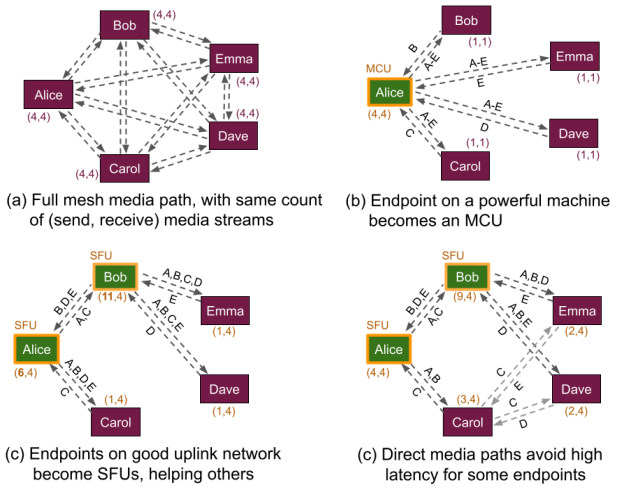
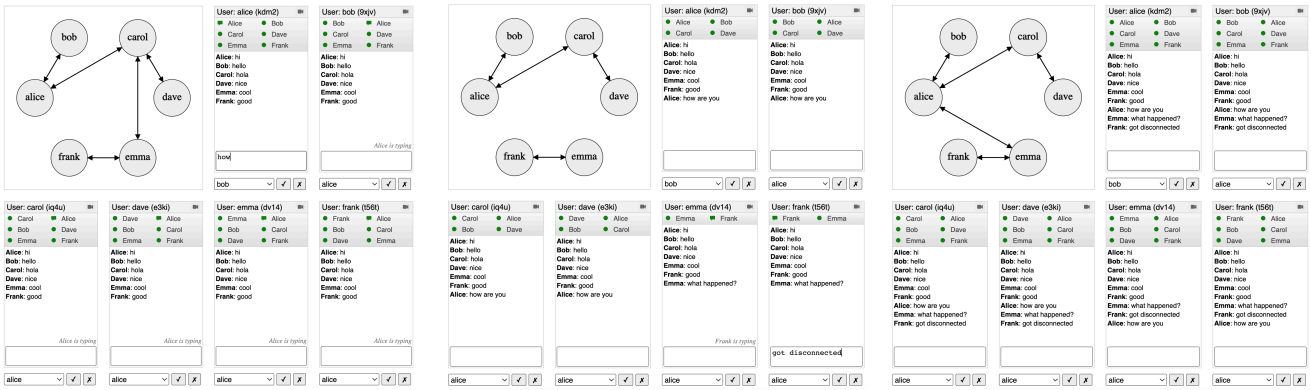



**图 12** 媒体路径的各种拓扑。箭头表示媒体流的方向。每条流都通过单独的对等连接传输。

© 2026, Kundan Singh




**(a)** 控制路径的点对点网络显示在左上方。还显示了每位参与者的 media-chat 组件。

**(b)** Carol 与 Emma 之间的链路已断开。会议发生分区，每个分区继续独立运行。

**(c)** Alice 再次邀请 Emma，合并这两个分区。数据经过重新同步，以便显示所有消息。

**图13** 一个用于模拟最多 30 名参与者的测试应用程序，可创建连接路径组成的对等网络，并评估参与者之间的消息同步和媒体路径。虽然运行在同一台机器上，但每位参与者的应用都表现为独立的 Web 应用，使用实际的 WebRTC 连接、数据通道和媒体流 API 来建立控制路径和媒体路径。

如图 12(b) 所示，具有强大 CPU 或处理能力的节点可以充当混音器或多点控制单元（MCU）。在这种情况下，其他所有节点都会将媒体流发送到该节点；该节点混合所有收到的媒体流，生成单条出站媒体流，然后将其发送给其他每个节点。这会降低其他节点的网络用量和处理负载，但会略微增加 MCU 节点的处理负载。

对于视频，多点控制单元会在 `画布` 元素上绘制各种视频元素的布局（这些视频元素来自其他参与者的媒体流，以及本地预览），并从 `画布` 捕获媒体流，生成混合视频布局。对于音频，为避免用户听到自己的声音，多点控制单元应在将混合流发回说话者的端点之前，从混合流中移除该说话者的音频。使用多点控制单元的一个缺点是延迟较高；这不仅是因为其他用户的媒体路径需经由多点控制单元多经过一跳，也因为多点控制单元的播放和混音会带来额外延迟。

如图 12(c) 所示，一个或多个上行网络状况良好的节点可以充当转发器，即选择性转发单元（SFU）。这样，其他节点只需通过其上行链路向某一个 SFU 发送一路媒体流，而 SFU 本身则可以发送多于 N-1 路媒体流。虽然这降低了低端端点的上行带宽需求，却可能显著增加媒体路径上的延迟。图 12(d) 展示了部分媒体流如何通过直接链路而非 SFU 获取，以缩短延迟较高的路径。对于规模极大的端点驱动型会议，这种结合 SFU、MCU 和全网状媒体路径的混合架构可能很有用；在这类会议中，低延迟并非那么重要，例如网络研讨会或活动直播。

请注意，适用于基于服务器的选择性转发单元的 simulcast 或可伸缩视频编码功能在选择性转发单元节点中用处较小。为了减少节点的网络用量，此类选择性转发单元节点向其他节点发送视频时，应始终只使用单个层或级别。

是否使用 SFU 或 MCU，或者是否将某个端点提升为 SFU 或 MCU，是通过分布式方式决定的。决策可依据端点的异构性，以及实时质量和性能监控结果，以检测网络状况不佳或端点处理能力过载的情况。

## 4. 进一步讨论

我们将讨论实现中的一些重要设计选择、安全考量、技术挑战及潜在解决方案。

### 推送通知

Chrome 浏览器不再支持静默网页推送通知，也就是说，必须始终向用户显示或更新通知。由于我们使用该机制，通过数据通道初始呼叫信令进行双向提议-应答，因此在呼叫建立期间，应用会在两个端点上都显示通知。我们不会显示新通知，而是更新现有通知消息，以反映呼叫建立的进度，例如呼叫已接听、拒绝或取消，从而确保每个方向上的每个邀请在任意时刻只显示一条通知。初始呼叫邀请的通知会持续显示，并要求用户进行交互；后续事件的通知则会在超时后自动隐藏。

当安卓上的令牌因任何原因发生更改时，FCM 会触发令牌刷新事件。但网页推送已不再提供此类事件。通常，令牌会保持有效很长时间，长达数月，但也可能因客户端网页应用无法控制的原因而被删除或失效。由于无法保证令牌


© 2026, Kundan Singh

刷新事件，我们的 Web 应用的服务工作线程会在每次启动或激活时尝试获取 FCM 令牌。即使用户很长时间没有打开应用，这也能确保应用仍可触达。此外，客户端应用可以使用服务工作线程中提供的周期性后台同步 API，或扩展程序中的 alarms API，安排定期刷新令牌，例如每天刷新一次。用户必须授予此权限，以及现有的摄像头、麦克风、通知和弹出窗口权限。

由于控制路径和媒体路径彼此分离，因此也可以将呼叫切换为混合拓扑：控制采用点对点方式，而媒体集中处理，反之亦然。对于集中式控制路径，可以使用 RTC bricks 项目的集中式资源服务器。对于集中式媒体路径，可以部署外部 SFU 或 MCU 服务器。即使采用集中式控制路径，推送通知对于发送初始呼叫邀请通知仍然很有用。

通常需要服务器调用 FCM API 来发送推送通知。虽然 Web 应用可以通过 JavaScript 调用该 API，但这需要将 FCM 身份验证密钥保存在应用中，因此不建议这样做。下面介绍如何通过轻量级通知服务器实现这一功能。另一种方法是使用云托管的 API 网关和 Lambda 函数来实际发送推送通知 [24][25]。也可以使用一项免费服务 [26]，它可以利用 Firebase 或 BitTorrent 等技术，实现无服务器的呼叫建立或匹配。

最后，对于仅在受信任各方之间用于通信的私有可用应用，将 Firebase 云消息传递密钥保存在端点应用中并不是问题。不过，此类系统仍需跟踪用户标识符与 FCM 令牌之间的映射。可以使用可公开访问的文档存储或社交媒体个人资料来实现此类映射。例如，remoteStorage 可以使用专为此目的设计的 WebFinger [27][28], 协议；用户标识符可以包含用户可公开查看的 Google Docs 或 Dropbox 文件 URL，并由用户更新以包含联系人令牌；或者用户标识符可以包含 LinkedIn 个人资料，个人资料页面上的预定义区块或链路可以包含联系人令牌。此类想法将用户可达性的存储委托给外部服务。

或者，也可以使用 P2P 网络以完全分布式的方式存储可达性数据。已有研究提出将此类 P2P 网络用于基于 VoIP/SIP 的通信中的呼叫信令 [29][30]。然而，Web 应用无法在 JavaScript 中创建侦听套接字。因此，若没有外部套接字网关、插件或外部 P2P 网络，就无法仅通过 Web 应用实现此类用于呼叫信令的 P2P 网络。不过，在大型会议中，可以将此类结构化 P2P 网络或分布式哈希表（DHT）用于控制路径网络（替代图 10）。

无论是将 FCM 令牌公开存储在公共数据存储中，还是存储在 P2P 网络中，都存在风险：恶意实体可能向用户的设备发送任意推送通知。有害通知可能包含附件或指向恶意软件的链路。另一方面，使用受控的通知服务器发送推送通知，可以限制通知内容，并将访问权限限制为仅授权发送方。

### 无服务器模式

在无服务器模式中，尽管我们的应用演示了电子邮件的用法，但信令消息的实际发送和接收不在该应用的范围内。更重要的是，可以使用任何其他流行的消息传递服务。由于消息大小限制，使用普通短信较为棘手，在这种情况下，建议使用带有包含该消息的附件的彩信。

基于电子邮件的流程有一个缺点：它不是实时的，也就是说，发送和接收电子邮件可能会有明显延迟。因此，在此模式下，邀请或接受消息的有效期默认是两分钟。如果响应延迟超过两分钟，应用可能不会使用该消息。交换呼叫信令数据时，如果可以使用即时消息，优先选择即时消息，而不是电子邮件消息。

在应用与消息客户端之间手动复制粘贴呼叫信令消息可能很麻烦。Web 应用程序支持拖放。可以创建包含信令消息的可拖动附件，让用户在此应用与其消息客户端之间拖放这些附件。不过，出于安全考虑，现有的电子邮件客户端（包括基于 Web 的客户端）不允许跨应用拖放非文本和非图像内容。作为一种变通方法，可以将信令消息呈现为图像，并将实际消息文本存入 PNG 图像文件的元数据内容中；也可以将信令消息编码为二维码并呈现为图像，以支持拖放。我们尚未测试这种方法是否确实可行。

对于无服务器的提议-应答协商，一个挑战是确定包含 ICE 候选项的 SDP 提议和应答的有效时长。这取决于端点附近的网络设备，例如，如果没有流量，用户的本地路由器或网络地址转换会将端口保持开放多久。在路径中不存在网络地址转换的情况下，例如在同一局域网内的端点之间建立连接，提议和应答的有效时间超过两分钟，长于我们应用的呼叫建立超时时间。不过，许多路由器会关闭处于非活动状态的端口，其超时时间为 30 秒到 5 分钟不等。如果浏览器的 WebRTC 实现能够在内部刷新提议和应答的 STUN 绑定，直到对等连接成功建立或关闭，那么 Web 应用就不必担心开放端口的有效性。

© 2026, Kundan Singh

### 视频会议 *Web* 应用

我们的客户端端点运行着三个独立的 Web 应用：Ezcall、包装器和media-chat-peer。Ezcall包含一个服务工作线程，其用途是执行初始呼叫信令并发送通知；而media-chat-peer Web 应用则用于提供视频会议用户体验。负责将两者连接起来的包装器功能，可以与另外两个应用合并，从而减少端点中的 Web 应用数量。不过，我们将其保持独立，以便在需要时，实际的视频会议 Web 应用可以在受控或沙盒化的iframe环境中运行。这样便可使用其他第三方应用，同时用户仍可以对这些应用的操作施加某些安全限制。

视频会议应用位于一个与呼叫信令应用分开的窗口中。这样一来，呼叫信令应用可以调整为较小的窗口，同时显示全尺寸的视频会议窗口。它还支持类似即时通讯工具的标签式对话：将各个会议放在不同的浏览器标签页中，从而支持多个会议。它也能与桌面安装应用（渐进式 Web 应用）良好配合。它也很适合与浏览器扩展配合使用：点击扩展程序图标时，主信令应用会以小弹出窗口的形式启动。此功能还可以扩展为第三方网站上的点击呼叫按钮。在移动设备上，我们使用 iframe 加载会议应用的包装器，而不是打开单独的窗口，以避免后台窗口导致运行变慢。

如前所述，我们将初始控制路径的对等连接与媒体路径的对等连接分开。可以对此进行优化，例如建立一个对等连接工厂，在应用的不同层之间重复使用。创建初始数据通道，并使其与媒体流分开，可以确保初始呼叫信令的消息大小不超过推送通知允许的上限。后续的协商消息可以通过该数据通道发送。

RTC bricks 项目的 Web 组件还可以进行优化，以支持多个媒体流复用对等连接。这也将使我们的项目受益：不仅适用于全网状媒体路径，也适用于在端点中使用 SFU 或多点控制单元（MCU）的情况。目前，media-chat 组件在媒体路径中仅实现了全网状拓扑。应对其进行扩展，使其支持在端点中使用 SFU 和/或多点控制单元（MCU）。特别是，video-io Web 组件应确保在发布媒体流时仅调用一次视频编码器，即使有多个订阅者通过各自独立的对等连接接收该发布者的媒体流。

### 身份验证令牌与安全性

我们使用简单的身份验证令牌来进行初始用户查找和呼叫建立。外部注册流程应确保与令牌关联的用户标识符属于该用户，例如，如果使用电子邮件地址作为标识符，则可在注册时通过电子邮件发送令牌。用户应将身份验证令牌视为密钥。注册系统应允许轻松撤销任何已泄露的令牌。泄露的令牌可能导致此应用中的身份盗用，恶意行为者可冒充其他用户。其风险与密码泄露类似，而多因素身份验证或通过第三方身份验证器实现单点登录等额外安全措施可以降低风险。我们可以在应用安装期间使用客户端证书，并将身份验证令牌与证书关联，例如将其作为公钥的指纹。在这种情况下，只要与证书关联的私钥未泄露，令牌泄露就不会危及安全。随后，系统会使用客户端证书与服务器建立所有安全连接（传输层安全协议），并在服务器和其他客户端处使用该证书进行用户身份验证。

我们的 media-chat-peer Web 应用会将已注册用户标识符或电子邮件地址（仅使用用户部分，不包括域名）复用为用户界面中的用户显示名称。如果使用第三方 Web 应用，或采用无服务器呼叫信令，会议用户可能会冒用显示名称，从而造成不理想的用户体验。会议中显示名称的准确性由具体的会议应用负责，而许多现有的基于网页的会议应用并未施加此限制。然而，更重要的是呼叫信令中的用户标识符与视频会议应用中的用户标识符之间的关联。如果使用的第三方 Web 应用具有独立的用户识别机制，那么将两个标识符关联起来就成为该 Web 应用的责任。

RTC bricks 项目的 Web 组件使用 共享存储组件来维护会议数据结构。目前，它几乎没有访问控制或完全没有访问控制，例如，一个用户可以更新另一个用户创建的数据对象，从而篡改文本消息或其他人的呼叫状态。受控的 Web 应用运行良好，并通过此类共享存储彼此协作。但恶意应用一旦获得存储访问权限，就可能滥用它。这与 Firebase实时数据库类似，访问控制责任由客户端应用及其数据访问策略承担。与此不同，RTC Bricks 提议采用文件系统式访问控制。此类具有访问控制的共享存储可以确保对不同用户创建的各项资源设置正确的读写权限。例如，/sessions/aaa-bbb-ccc资源只能由该会议的用户访问，而子资源/users/alice归该用户 Alice 所有，她拥有完全访问权限，其他参与者则只能读取该子资源；又如，/messages子资源仅允许追加，任何用户都可以在其中添加消息对象，但不能修改已有的消息对象。我们可以在PeerStorageImpl对象中复用该访问控制机制，以便在分布式共享数据存储中启用它。

除了对共享存储上资源进行基本的访问控制外，某些资源可能还需要具备一定的数据完整性和隐私保护。例如，文本消息


© 2026, Kundan Singh

资源可以使用发送方的私钥进行签名，以便其他人验证该消息确实由该发送方发送。同样，Alice 发送给 Bob、但作为资源存储在共享存储中的私密消息，可以使用 Bob 的公钥加密，这样只有 Bob 才能解密并查看该消息。这些理念中的许多都适用于一般的网络视频会议应用，但不属于我们的Ezcall应用的讨论范围。

## 5. 结论与未来工作

我们介绍了一种用于实现端点驱动的多方视频会议应用的软件架构。我们扩展了 RTC bricks 项目的集中式视频会议应用，使其支持对等网络和数据存储，并基于推送通知实现呼叫信令。我们的工作包括一个轻量级端点实现，其中包含呼叫信令应用（Ezcall）、网络视频会议应用（media-chat-peer）、一个充当粘合层的包装器应用，以及新的 Web 组件（PeerNetworkImpl 和 PeerStorageImpl）。总体而言，端点新增部分的代码不到 3000 行，主要使用 JavaScript 编写；轻量级通知服务器的 PHP 代码不到 300 行。

通过使用电子邮件来传递呼叫信令，我们的应用可以实现无服务器化。这样就无需部署或实现专用的视频会议应用/媒体服务器。严格来说，它仍会使用电子邮件服务器，但这些服务器彼此独立、无处不在，且与我们的系统无关。我们软件的轻量特性尤其适用于低成本通信场景，同时还能保护用户隐私，不像其他大型科技公司的“免费”服务。

我们的软件仍处于早期阶段。我们在上一节中提到了一些局限性。除了着手解决其中一些局限性外，我们还计划继续探索端点创新的新方式。这与当前通过软件即服务（SaaS）创建 Web 应用的趋势背道而驰，但符合互联网的基本原则：将应用逻辑和控制权保留在端点。从用户的角度来看，长远而言，拥有已安装的应用且无需支付订阅费，比租用 SaaS 应用并持续付费更划算。

## 6. 参考文献


[1] WebRTC：浏览器间实时通信，万维网联盟推荐标准，2025年3月，http://www.w3.org/TR/webrtc/ [2] WEB 浏览器实时通信（RTCWEB）互联网工程任务组工作组，http://tools.ietf.org/wg/rtcweb [3] Johnston, A.B., Burnett, D.C.《WebRTC：HTML5 实时 Web 的 API 和 RTCWEB 协议》。Digital Codex，2012年，ISBN 978-0985978808[4] Singh, K.《WebRTC 通知系统与信令范式》。博客。2019年。https://blog.kundansingh.com/2019/06/webrtc-notification-system-and.html [5] Singh, K.《Ezcall：WebRTC 的轻量级呼叫信令》。访问于2026年2月。https://github.com/theintencity/ezcall [6] 推送技术。维基百科。访问于2026年2月。https://en.wikipedia.org/wiki/Push_technology [7] Firebase 云消息传递。维基百科。访问于2026年2月。https://en.wikipedia.org/wiki/Firebase_Cloud_Messaging [8] Singh, K., Krishnaswamy, V.《JavaScript 中采用 SIP 的理由》。《IEEE通信杂志》，第51卷，第4期，2013年4月。[9] 渐进式 Web 应用。维基百科。访问于2026年2月。https://en.wikipedia.org/wiki/Progressive_web_app [10] Singh, K.《深入探究 WebRTC 拓扑》。博客。2024年。https://blog.kundansingh.com/2024/06/a-deep-dive-into-webrtc-topology.html [11] Iyengar, M.《WebRTC 架构基础：点对点、选择性转发单元、多点控制单元与混合方案》。博客。2019年3月。https://medium.com/securemeeting/webrtc-architecture-basics -p 2 6 7 77 46 66 [12] p-sfu-mcu-and-hybrid-approaches-ed a a Husøy, T.K.《WebRTC 服务中的拓扑》。硕士论文。远程信息处理系，挪威科技大学。2015 11250 2352767 . [13] https://hdl.handle.net/ / Sandholm, T.《SnoW：基于 WebRTC 的无服务器多方通话》。2022年6月。doi:10.48550/arXiv.2206.12762，软件见 https://github.com/patompa/snow [14] Ganeko, M.《在浏览器中构建 WebRTC 多点控制单元》。幻灯片。2016年8月。https://speakerdeck.com/mganeko/build-webrtc-mcu- on-browser[15] Davids, C., et al.《面向 Web 语音和视频通信的 SIP API》。IP 电信原理、系统与应用国际会议（IPTcomm），美国伊利诺伊州惠顿。2011年8月。[16] Singh, K., Krishnaswamy, V.《利用端点和云资源服务构建通信型 Web 应用程序》。IEEE 云计算国际会议（IEEE Cloud），美国加利福尼亚州圣克拉拉。2013年6月至7月。[17] Singh, K., Yoakum, J.《Vclick：端点驱动的企业级 WebRTC》。IEEE 多媒体国际研讨会（IEEE ISM），美国佛罗里达州迈阿密。2015年。[18] Anguenot, O.《WebRTC 中的信令》。博客。2023年12月。https://www.webrtc-developers.com/signaling-in-webrtc/ [19] Holmberg, C., Arnold, M.《基于会话发起协议（SIP）的推送通知》。互联网工程任务组 RFC 8599。2019年。[20] 在 WebRTC 中使用电子邮件进行信令。Stackoverflow。12月 2022 74847473 。[21] https://stackoverflow.com/questions/ WebRTC 数据通道与手动信令的完整示例。Stackoverflow。2019年3月。https://stackoverflow.com/a/54985729 Singh, K.《RTC Bricks：运用 [22] Web 组件释放 WebRTC 的创造力》。项目（进行中）。访问于2026年2月。https://github.com/theintencity/rtcbricks [23] Rosenberg, J., et al.，《SIP：会话发起协议》。互联网工程任务组，RFC 3261。2002年6月。[24] 用于 WebRTC 连接的无服务器 WebSocket。项目。访问于2026年2月。https://github.com/ScryVR/scry-rtc [25] 使用 AWS 和 Lambda 函数实现的无服务器 WebRTC 信令服务器。项目。访问于2026年2月。https://github.com/ hakobera /serverless-webrtc-signaling-server [26] Trystero：构建无需服务器的即时 Web 应用。项目。访问于2026年2月。https://github.com/dmotz/trystero [27] Jong, M.B., Kooman, F., and Kippe, S.《远程存储 1.0》。IETF 互联网草案。draft-dejong-remotestorage-26（进行中）。2025年12月。[28] Jones, P., et al.《Web Finger》。互联网工程任务组 RFC 7033。2013年9月。[29] Singh, K., Schulzrinne, H.《使用 SIP 的点对点互联网电话》。NOSSDAV，美国华盛顿州斯卡马尼亚。2005年6月。[30] Singh, K., Schulzrinne, H.《SIPpeer：基于 SIP 的点对点互联网电话客户端适配器》。报告。哥伦比亚大学。2004年。https://kundansingh.com/ papers/sip-p2p-design.pdf


© 2026, Kundan Singh