Real Time Internet Peering Protocol*

draft-rosenbergjennings-dispatch-ripp-03

IETF 105 – July 22nd, 2019

J. Rosenberg

Five9

C. Jennings

cisco

A. Minnesale

SignalWire

FreeSWITCH

J. Livingood

comcast

J. Uberti

Google

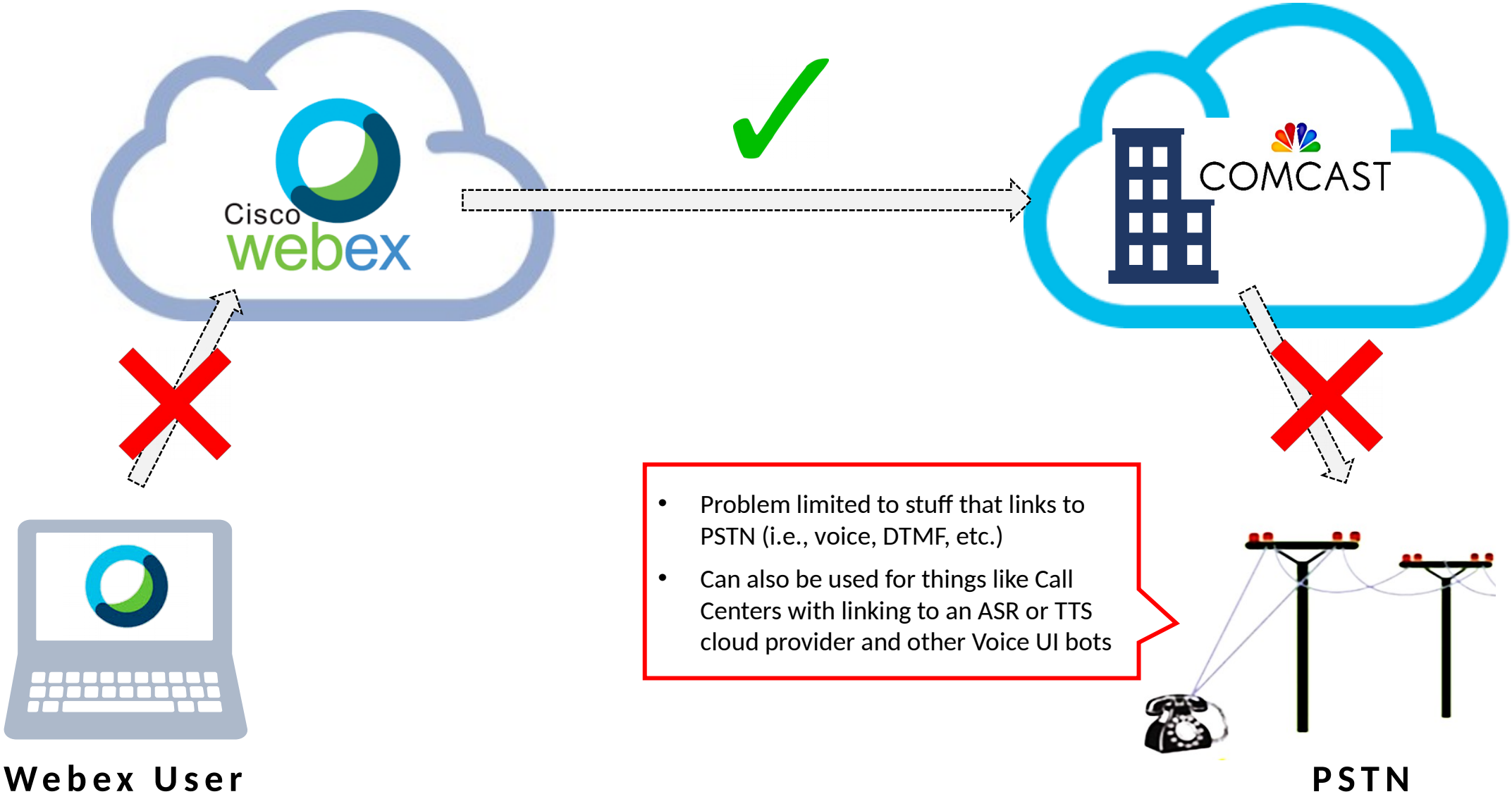
V6

* Five9 may have IPR on this draft: https://datatracker.ietf.org/ipr/3643/

Goals for Today

- Share the problems we’re trying to solve
- Gauge interest from others in solving these problems
- See if there is interest in implementing and tuning
- Technical details aren’t important right now (in fact, the co-authors themselves don’t all agree!)
- We’re not asking for a BoF yet
- We want to implement and prototype and then return to the next IETF with a more baked proposal

RIPP Problem Space



- Problem limited to stuff that links to PSTN (i.e., voice, DTMF, etc.)
- Can also be used for things like Call Centers with linking to an ASR or TTS cloud provider and other Voice UI bots

Why don’t we just stick with what we already have (SIP + SDP + RTP)?

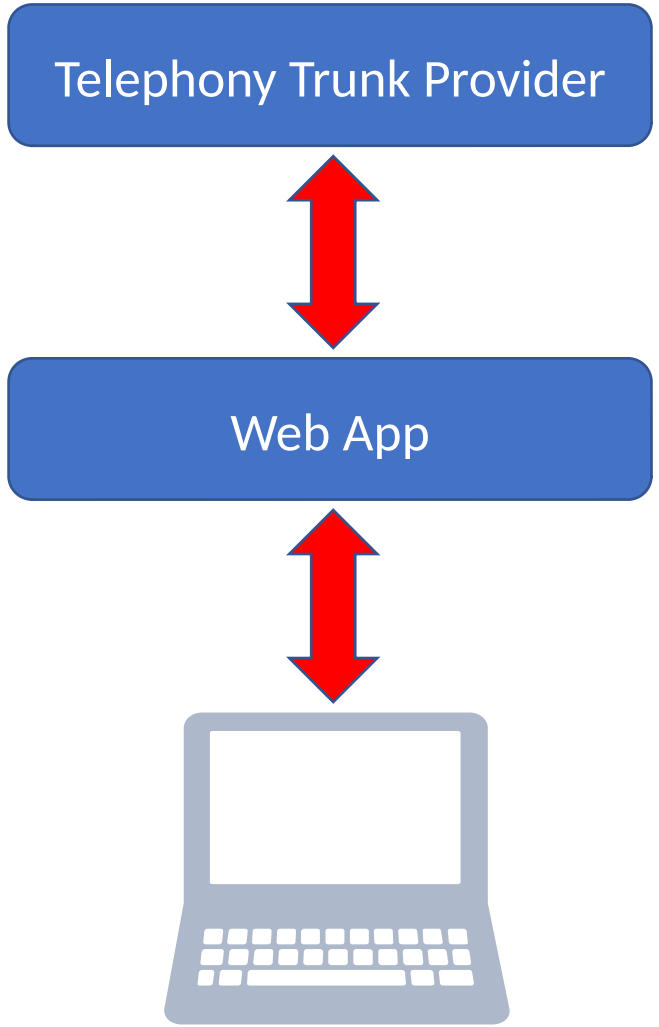
A cloud service like Webex can use existing cloud services for all of the following:

- Autoscaling
- Load balancing
- High reliability
- API gateways
- DDos mitigation tools
- Service Mesh (e.g., Istio)
- APM
- Tools the workforce already knows how to use

We *could* build all of these in a way that achieves all of this, but it would be:

- More work
- Harder to deploy
- Harder to use
- Sub-optimal because wouldn’t leverage the vast changes that have occurred to support HTTP-based APIs

This is the other half of WebRTC



RIPP – cloud to cloud

WebRTC – client to cloud

Use Cases

- Add browser voice customer care to eCommerce sites
  - WebRTC solves browser to web servers, but we need to trunk the call to the contact center from there
  - This requires softswitches, SBCs, and special network considerations today – high complexity, not done often or broadly
  - With RIPP – it is “just another web app”
- Telecom apps on web PaaS
  - AWS, Azure, Google – increasing sets of services making it very easy to deploy and run non real-time apps (e.g., Google AppEngine, HTTP LB, Istio on GKE, APIGee, etc)
  - They could modify all of their code to ALSO work for SIP – but they haven’t, and likely wont
  - Goal – make it so that once they go HTTP3, real-time will just work

Use Cases

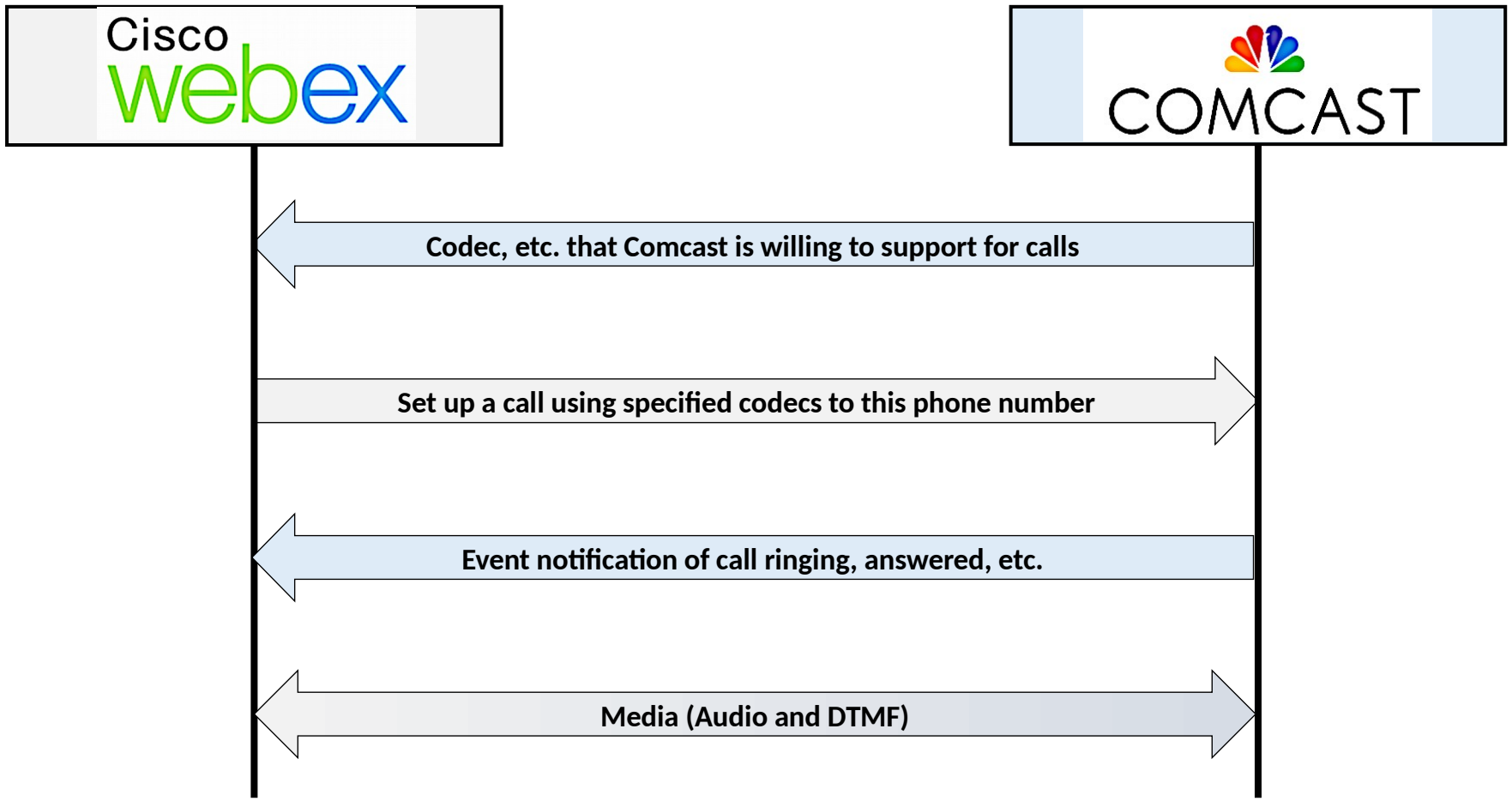
- Increase reliability SaaS to trunking provider calls
  - Webex to Comcast; Five9 to Tata; RingCentral to Verizon
  - Difficult to achieve hitless upgrades – server restarts cause calls to drop
  - Underlying VM restarts and migrations cause server restarts – calls drop
  - Cluster expansion / contractions difficult – no way to drain and migrate calls reliably and consistently
  - Shouldn’t it be as easy to achieve hitless upgrades, handle VM restarts & migrations, and deal with automated cluster expansion / contraction as it is today for web apps??

Technical Details

Really just a starting point to help folks understand what we had in mind

General Information Flow

Ignore what protocols are used and just look at data for a call from Webex to PSTN…



Capabilities

- Capabilities allow Comcast to indicate what codecs it supports and other relatively static information about system capabilities
  - These do not change frequently – reasonable to think of as updating on perhaps a daily basis, but they do *not* contain per call information
  - Capabilities could include:
    - Supported codecs
    - Max bitrate & max sample rate
    - Force use of CBR (constant bit rate for the security crazy)
    - A few more weird things for advanced use-cases

Advertisement

- The calling side (Webex in this case) has the capabilities of the other side (in this case, Comcast). The calling side creates an Advertisement of all the exact details for setting up the call. These details must be consistent with the capabilities from the other side. The advertisement is sent to Comcast and they can *_only_* accept or reject it. There is no offer / answer like negation.
  - Information in advertisement:
    - Number to call
    - Caller ID of caller

Caller ID

We use (drum roll please)…

Passport

Protocol Mapping

- All the authors are big believers that:
  - In the near future, it will be possible to do unreliable data over QUIC
  - HTTP/3 over QUIC is a good idea
- Currently mapped to multiple HTTP requests with simple API

https://ripp.example.com/trunks/{trunkID/consumerTrunk - get an auth token

https://ripp.example.com/trunks/{trunkID/capAdv - get the Advertisement of capabilities for this trunk

https://ripp.example.com/trunks/{trunkID/calls - Get ongoing calls on this trunk

https://ripp.example.com/calls/{callID/prevEvent - Get the previous even (ringing, answer ….)

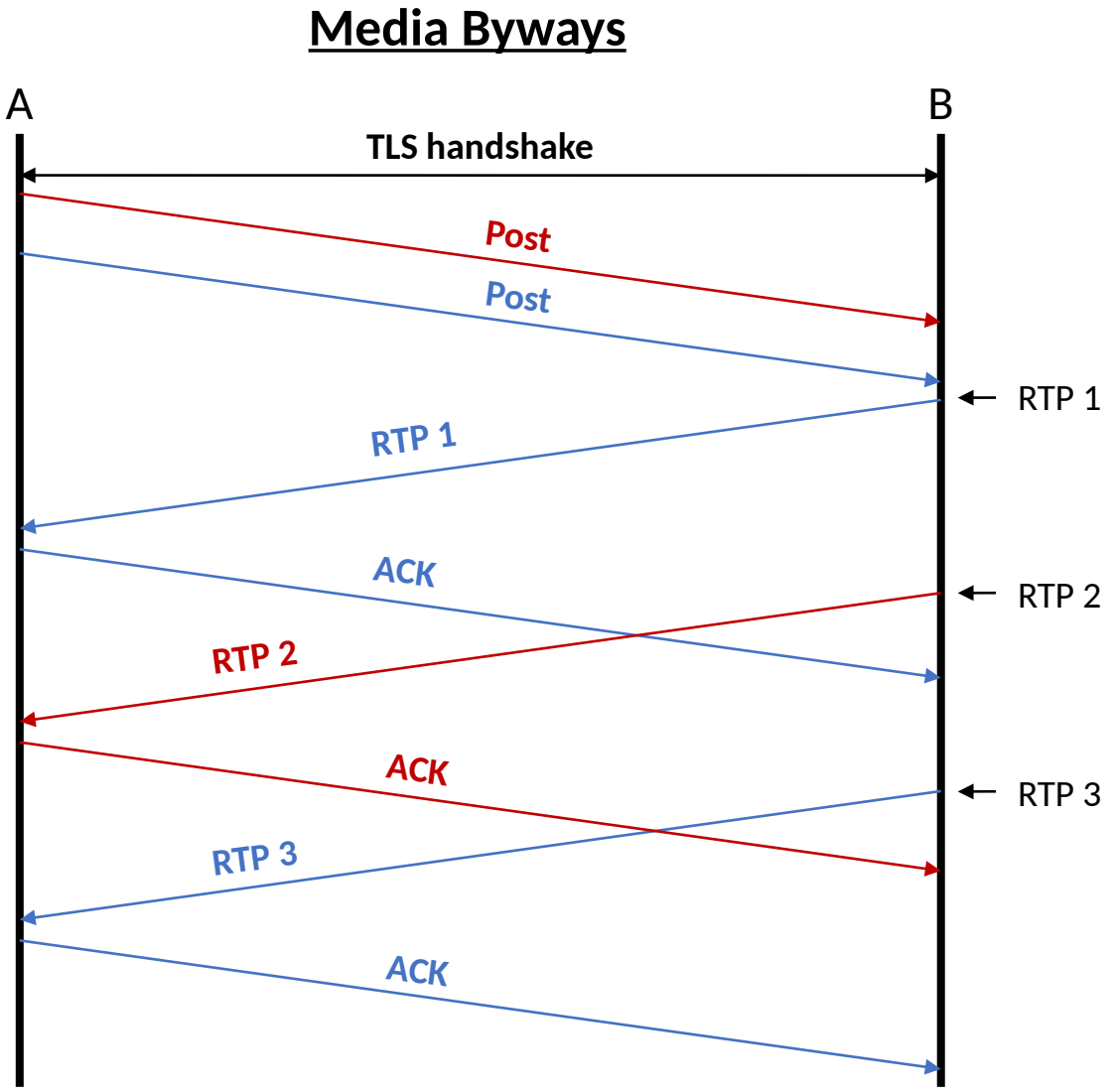
https://ripp.example.com/calls/{callID/event - Wait for next and event and return it

https://ripp.example.com/calls/{callID}/media-forward - Flow of media chunks in forward direction

https://ripp.example.com/calls/{callID/media-reverse - Flow of media chunks in reverse direction

Media Byways

- We are sending media over the HTTP/3 connection
  - Many ways to do this— still need to figure out what works best
- Draft currently purposes several simultaneous HTTP long poll type requests that are multiplex in one HTTP/3 session. The sender sends media on whichever one is currently “empty” and waiting for data. Each packet is ACK’d so that the sender knows more data can be sent on that HTTP transaction. These are called byways.
- There are multiple byways each direction



Events

The events that flow both directions on a call are:

- **Ringing**: phone is ringing
- **Accepted**: user (human or bot) answered the call
- **Rejected**: the user (human or bot) declined the call
- **Failed**: something went wrong
- **Ended**: user (human or bot) ended the call

Security

TLS

- Signaling encryption and integrity
- Media encryption and integrity
- Server authentication

OAuth

- Client authentication

End-to-End Security

- To the PSTN? Seriously? No.

Backup Slides

Inbound Calls

- Pretty much the opposite of the outbound case with an automated pre-configuration process.
- Before any calls take place, the Webex side creates a HTTP/3 connection to Comcast and uses a specific API to pass Comcast an auth token and base URI for Webex endpoint.
  - Later, when Comcast has an incoming call for Webex, it uses that URI and auth token to make the call happen.

Recover from Failed Server

- If the client finds the connection failed, closed, or just dead (no ACKs), it simply forms a new HTTP connection using the URLs from before and picks up the old call
  - 5 second window to do this before the far side terminates the call

Moving Calls

- Servers can tell client to migrate media connection gracefully to new server
- Clients can do the same

The API

https://ripp.example.com/trunks/{trunkID/consumerTrunk - get an auth token

https://ripp.example.com/trunks/{trunkID/capAdv - get the Advertisement of capabilities for this trunk

https://ripp.example.com/trunks/{trunkID/calls - Get ongoing calls on this trunk

https://ripp.example.com/calls/{callID/prevEvent - Get the previous even (ringing, answer ….)

https://ripp.example.com/calls/{callID/event - Wait for next and event and return it

https://ripp.example.com/calls/{callID}/media-forward - Flow of media chunks in forward direction

https://ripp.example.com/calls/{callID/media-reverse - Flow of media chunks in reverse direction

Gateway to SIP