Real-Time Internet Peering for Telephony (RIPT)
Problem Statement and Protocol Summary

Jonathan Rosenberg
CTO Five9
26 March 2020
Deploying SIP/RTP apps in public cloud is much harder than web apps with a growing gap

**Hard/Impossible**
- elastic peering (SIP trunking IP based),
- Hitless upgrade on peering (requires call drops)
- Call survivability (dropped call on server failure)

Proxy farms, SBCs, custom DNS or IP,
Firewall configs, custom HA, custom load balancing, custom monitoring and tracing... your biz logic

**VM**

---

Biz logic

- Service Mesh
- Kubernetes
- Global Elastic LB + TLS termination + geoDNS
- Cloud Functions
- VM
Anthos Service Mesh

Overview

Alerts timeline  No service alerts. Time selection is 8:21 PM to 9:21 PM.

Requests 0.1/sec
0.0/sec (+12%)

Error rate 0.0%
-0.5% (+100%)

Latency 50th percentile 2ms
0ms (+2%)

CPU 0.1%
+0.01% (+12%)

Memory 31.4%
+0.07% (0%)

Disk 0.0%
0% (0%)

Service status: none
There are no SLOs set for this service.

CREATE AN SLO

Details

Namespace  default
Mesh  us-west1/main-gke-1
Clusters  us-west1/main-gke-1
Host  aalite.default.svc.cluster.local
What is it about SIP/RTP?

• It's not using HTTP
• IP addresses all over the place breaks load balancers and HA techniques
• RTP in the cloud (bridges, transcoders, etc)
  • Ports for session ID incompatible with kubernetes
  • Usage of IPs for routing incompatible with load balancers, modern HA techniques
  • SIP trunking requires VIP for any form of HA – works very poorly in public cloud
Problem Statement

Enable the deployment of VoIP apps on top of public cloud platforms such that they can take advantage of the capabilities provided by those platforms for web applications (such as load balancing, HA, hitless upgrades, service mesh, geo-redundancy, etc), minimizing (if not eliminating) the need for customized behaviors unique to VoIP.
Resulting Protocol Requirements

- “just an app” ontop of HTTP (and ideally http3)
- No IP addresses or ports in the protocol
- Server-centric state
- Client-Server
- Signaling and Media Together
- Separation of Call and Connection State
- OAuth
## Many Use Cases

<table>
<thead>
<tr>
<th>Same Provider</th>
<th>Browser to Cloud App</th>
<th>IP Softphone to IP PBX</th>
<th>App to App (ASR to cloud)</th>
<th>Real-time devices</th>
<th>SIP Trunking alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media Transport (/media)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Control (start, stop) (/events)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Media Negotiation (/handlers)</td>
<td>(API not protocol)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Identity Management (/providers, /consumers)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Exchanging media: /media

GET {{callURI}}/media
GET {{callURI}}/media
GET {{callURI}}/media
PUT {{callURI}}/media {media chunk}
200 OK to GET {media chunk}

Media Chunk = payload of RTP packets

Media is always sent to the call URI, which you get....
By control /events – when a call is established

Events:
- Proceeding
- Alerting
- Answered
- Declined
- Ended
- Migrate
- Moved
- Ping
- Pong

Handler describes media capabilities, which was shared...
By posting to /handlers one time event when new client comes online

Which was sent to a “tg” – a bag for policy and identity management, which was gotten...
By GETting to /providertgs
And for receiving inbound calls – POSTING to /consumertgs

"outbound":
{
"origins": "RFC 8226 cert with +14085551000 and +14085551002",
"destinations": "+1*"
}