Protocol Compliance Study in Popular RTC Applications

Woo Xie, Peng Qiu, Peiqing Chen, Zaoxing Liu
RTC Apps are Wild

- Do NOT fully comply with RFCs.
- We will measure and discuss:
  - To what extent do RTC apps comply?
  - Why do RTC apps modify, or augment the standards?

<table>
<thead>
<tr>
<th></th>
<th>FaceTime</th>
<th>Zoom</th>
<th>Discord</th>
<th>Messenger</th>
<th>WhatsApp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension of standard features</td>
<td></td>
<td></td>
<td></td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>Proprietary implementation of standard features</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>Invention of non-standard features</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>
Real-Time Communication (RTC) Apps

- FaceTime
- Discord
- zoom
- Messenger
- WhatsApp
RFCs for RTC

- **Session Establishment**
  - Session Traversal Utilities for NAT (STUN)
  - Traversal Using Relays around NAT (TURN)
  - Interactive Connectivity Establishment (ICE)

- **Transport Protocols**
  - UDP/TCP/QUIC

- **Media Transport Protocols**
  - Real-Time Transport Protocol (RTP)
  - RTP Control Protocol (RTCP)
Data Collection & Preprocessing

- One-to-One Call Setup
  - 5 Apps: FaceTime, Zoom, Discord, Messenger, WhatsApp
  - Device/Tools: Two iOS client devices + Wireshark on Mac for capturing packets
  - Networks: cellular network + Wi-Fi

- Test Procedure
  - 5s period for each user action:
    - Turn on/off media devices (e.g. video and audio)
    - Switch networks (Wi-Fi vs Cellular)
    - Reconnect the call after temporary exiting

- Preprocessing
  - Remove background traffic
  - Classify call sessions into P2P or relay mode
  - Differentiation between media and non-media data
## Highlights

<table>
<thead>
<tr>
<th>Applications</th>
<th>Features</th>
</tr>
</thead>
</table>
| FaceTime     | ● Co-existence of QUIC and non-QUIC in one UDP session.  
● Proprietary RTP Encapsulation.  
● various types of ping-pong messages |
| Zoom         | ● 1000-byte filler packets.  
● Proprietary RTP Encapsulation.  
● Proprietary ping-pong packets with time-based counters. |
| Discord      | ● Absence of P2P mode in one-to-one calls.  
● Proprietary STUN-like packets for client public IP discovery.  
● Proprietary ping-pong packets with a time-based counter. |
| Messenger    | ● Proprietary STUN attribute as timestamp.  
● Proprietary STUN attribute as client network cost and type.  
● Proprietary TURN attribute as client identifier. |
| WhatsApp     | ● 5/9 STUN messages are proprietary  
● 5/11 STUN attributes are proprietary  
● 3/11 STUN attributes are not compliant |
FaceTime: multiplexing

- QUIC (.9%)
  - complete QUIC handshake
  - one QUIC session per network interface
  - client to relay server, not caller to callee
- RTP (96.7%)
  - Locate the standard RTP header
- STUN (1%)
- Others (1.4%)
FaceTime: Proprietary RTP Encapsulation

- RTP with a proprietary header (96.7%)
  - Locate the standard RTP header
  - This proprietary header varies from 5 to 43 bytes.
FaceTime: Ping-Pong at various protocols

- **QUIC**
  - 5-8 QUIC packets every 15s

- **Proprietary protocol**
  - Cellular only and server to client only
  - datagrams of 44 bytes @ 17 pkt/s
  - starts with “0xdeadbeefcafe”

- **STUN binding requests**
  - last for exact 60s, at a constant pps
Zoom: 1000-Byte Filler Packets

- Payload bytes are identical (“0xda”)
  - it lasts 10~20s, after network changes
  - pps ramps up to ~500 pkt/s
- Speculation: Proprietary protocol for bandwidth probing
Zoom: Proprietary Encapsulation

- Proprietary header before and RTP*
  - 27~33 bytes
  - 1-byte indicator for packet direction
  - 1-byte indicator for media type

- Speculation: Additional media info layer.

---

Zoom: Proprietary Ping-pong

- After call established, 2 ping-pong transactions every 4 seconds,
  - Client ping; Server pong (Same UDP payload size: 81 bytes; No counter)
  - Server ping; Client pong (Different UDP payload size; Both with a time counter)
- If network switches, the counter will be reset.

- Speculation: Proprietary implementation of ICE connectivity check
Discord: Absence of P2P Mode in Calls

- From our experiments:
  - Discord has no P2P calls
  - Both one-to-one and group calls use relay server.
- Speculation: Support one-to-one call with group call tech stack (relay)

```
Caller 10.0.0.140
       +-------------+     +-------------+
       |             |     |             |
       |             |     |             |
       +-------------+     +-------------+
       | Relay Server |
       +-------------+     +-------------+
                     | 66.22.214.131 |
                     |             |
                     |             |
                     +-------------+

P2P

Callee 10.0.0.251
```
Discord: Proprietary STUN Protocol

- Once a client starts a call,
- Client sends a proprietary packet to server.
- Server’s response contains client’s public IP address in ASCII.
- Speculation: Proprietary implementation of STUN protocol for client IP discovery
Discord: Proprietary Ping-pong Packets

- After call established, every 5s, 8-byte packets are exchanged between client and server.

- Packet Format:
  - 4-byte constant payload (0x1337cafe)
  - 4-byte counter increasing over time

- Speculation: Proprietary implementation of ICE connectivity check with additional counter for client status detection
Messenger: Alien STUN Attribute (0xdaba)

- Exists in 83% of STUN Binding
- Proprietary STUN Attribute (0xdaba)
  - 4-byte counter increasing over time
  - Speculation: Latency computation

<table>
<thead>
<tr>
<th>Oxdaba values</th>
<th>97</th>
<th>1037</th>
<th>1983</th>
<th>3010</th>
<th>4017</th>
<th>5054</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.02s</td>
<td>1.00s</td>
<td>1.02s</td>
<td>1.00s</td>
<td>1.00s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Messenger: Alien STUN Attribute (0xc057)

- Exist in 41.6% of STUN Binding
- WebRTC STUN attribute (0xc057): GOOG-NETWORK-INFO
  - Google Network ID (2 bytes)
  - Google Network Cost (2 bytes)
- Speculation: Network type indicator

Google Network Cost values in WebRTC source code:

<table>
<thead>
<tr>
<th>Cost</th>
<th>Max</th>
<th>Cellular2G</th>
<th>Cellular3G</th>
<th>Cellular</th>
<th>Cellular4G</th>
<th>Cellular5G</th>
<th>Unknown</th>
<th>Low</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>999</td>
<td>980</td>
<td>910</td>
<td>900</td>
<td>500</td>
<td>250</td>
<td>50</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
Messenger: Alien TURN Attribute (0x4000)

- Exists in 27.9% of TURN Allocate requests
- Proprietary STUN Attribute (0x4000)
  - 118-byte value: a constant token over the call.
- Speculation: Client connection ID
WhatsApp: 5/9 messages are proprietary

<table>
<thead>
<tr>
<th>Proprietary STUN Types</th>
<th>Speculations</th>
<th>Observations</th>
</tr>
</thead>
</table>
| 0x0800                 | client closes a session| • only happens at the end of a call  
• client sends 5 copies to server, and server never responds  
• contains a attribute like call UUID |
| 0x0801 & 0x0802        | ping-pong             | • 0x0801 (client → server) pairs with a 0x0802 (server → client)  
• inter packet gap is ~2ms Before the call is established  
• inter packet gap is ~3 second After the call is established |
| 0x0804 & 0x0805        | ping-pong             | • 0x0804 (client → server) pairs with a 0x0805 (server → client)  
• inter packet gap is ~2ms before the call is established  
• initiated by server |
WhatsApp: 5/11 attributes are proprietary

<table>
<thead>
<tr>
<th>Proprietary STUN Attributes</th>
<th>Speculations</th>
<th>Observations</th>
</tr>
</thead>
</table>
| 0x4000                      | UUID         | ● 150 bytes,  
● constant throughout the call, unaffected by network switches.  
● used in both Allocate and Deallocate |
<p>| 0x4002                      | A timestamp  | 8 bytes, the elapsed time since 1/1/1970, in milliseconds. |
| 0x4003                      | Unknown      | 1 byte, constant value 0xff |
| 0x4004                      | Padding bytes | 440~452 bytes, with the identical byte (0x00). |
| 0x4007                      | Unknown      | 2 bytes, constant value 0x01f4 |</p>
<table>
<thead>
<tr>
<th>Attribute type</th>
<th>Standards</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0022</td>
<td><strong>RESERVATION-TOKEN</strong>&lt;br&gt;RFC8656: “The server includes this attribute in a success response”&lt;br&gt;RFC8656: “The attribute value is 8 bytes…”</td>
<td>• It is included in each Allocate request, prior server responses.&lt;br&gt;• 14 to 16 bytes.</td>
</tr>
<tr>
<td>0x802b</td>
<td><strong>RESPONSE-ORIGIN</strong>&lt;br&gt;RFC5780: “The attribute is inserted by the server and indicates the source IP address and port the response was sent from.”</td>
<td>• sent by the clients at the end of a call.&lt;br&gt;• 4 bytes, with the constant value 0x02000000.</td>
</tr>
<tr>
<td>0x0024</td>
<td><strong>PRIORITY</strong>&lt;br&gt;RFC8445: “The PRIORITY attribute MUST be included in a Binding request.”</td>
<td>• used in Binding requests and responses.&lt;br&gt;• 1 byte, with the constant value 0x1.</td>
</tr>
</tbody>
</table>
From Our Experiments & Analysis:

- RTC applications do not fully comply with RFCs
- But, their proprietary implementations share similarity
  - Proprietary RTP encapsulation (Zoom, FaceTime)
  - Call UUID (Messenger, Zoom, FaceTime)
  - Ping-pong with a time counter (Messenger, Discord, Zoom)
  - ...

- Speculation: Existing standards might not be flexible enough to satisfy new demands in today’s RTC industry.
Protocol fragmentation

For Application Developers
- Development cost
- Code sharing & reusability
- Performance evaluation & optimization
- Interoperability

For Network Operators
- Traffic monitoring
- Traffic prioritization & obstruction

For Research Community
- Knowledge sharing
- Performance comparison
- Protocol development
Appendix: Links

• [SIGCOMM IMC Paper submission](#)
• [Zoom & FaceTime Wireshark plugin](#)
• [Datasets](#)
Q & A