Second Screen CG Berlin F2F

May 17-18, 2018

Mark A. Foltz mfoltz@google.com
Brandon Tolsch btolsch@google.com

Outline: Day 1

- Agenda review
- Open Screen overview
- Discovery
  - Review of Chrome data
  - Mandatory vs. optional mechanisms
- Transport
  - QUIC Data Channels
- Authentication
  - J-PAKE
  - Public key based

Outline: Day 2

- Control protocol, serialization
  - CBOR vs. Protocol Messages
- HbbTV/ATSC compatibility
- Open Screen Protocol Library
- Future use cases and APIs
- Planning

Open Screen Protocol Background & Status

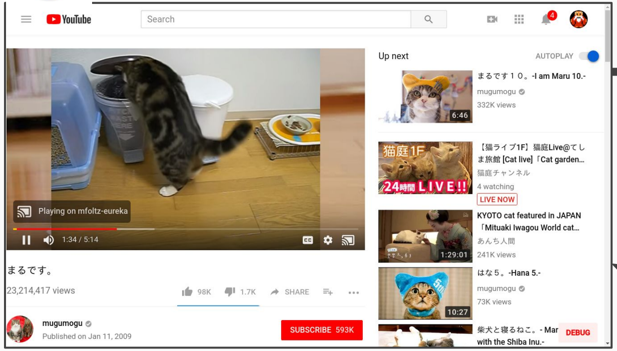
Second Screen CG History

- Nov 2013: Initial charter
- Nov 2013 - Dec 2014: Incubation of Presentation API
- Dec 2014: Presentation API transitioned to Second Screen Working Group
- Sept 2016: CG rechartered to focus on interoperability
- 2016-2017: Requirements, protocol alternatives, benchmarking plan
- Sept 2017: F2F at TPAC
- Jan-Feb 2018: SSWG rechartered. Phone conference, work plan
- May 2018: This meeting :-)

Presentation API

1. **Controlling** page (in a browser) requests presentation of a URL on a **receiver** device (on a connected display).
2. Browser lists displays compatible with the URL; the user selects one to start the presentation.
3. Controlling and receiving pages each receive a **presentation connection.**
4. The connection can be used to exchange messages between the two pages.
5. Either side may close the connection or terminate the presentation.

Presentation API: 2-UA Mode



Playing on mfoltz-eureka

1:34 / 5:14

まるです。

23,214,417 views

mugumogu

Published on Jan 11, 2009

Up next

まるです１０。-I am Maru 10.-
mugumogu
332K views
6:46

【猫ライブ1F】猫庭Live@てしま旅館 [Cat live]「Cat garden...
猫庭チャンネル
4 watching
LIVE NOW

KYOTO cat featured in JAPAN「Mituaki Iwagou World cat...
あんち人間
241K views
1:29:01

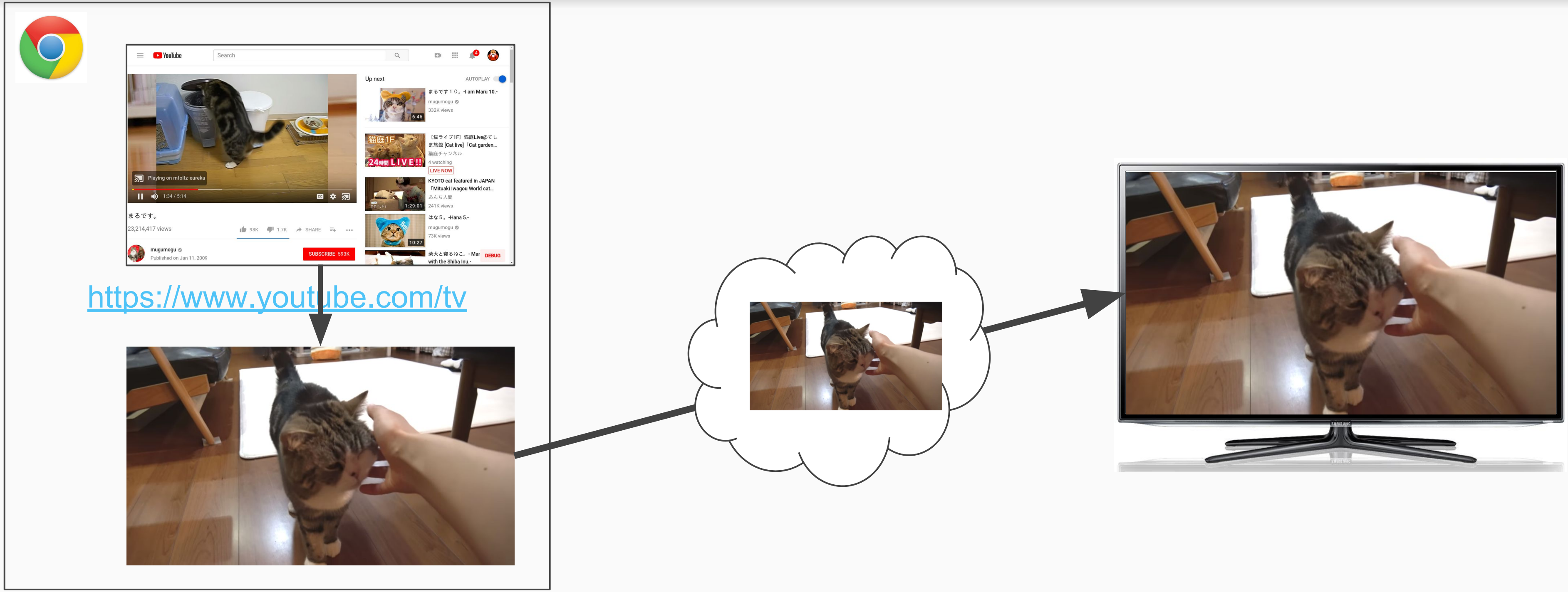
はな５。-Hana 5.-
mugumogu
73K views
10:27

柴犬と寝るねこ。- Mar with the Shiba Inu.-

https://www.youtube.com/tv

PLAY
PAUSE
...

Presentation API: 1-UA Mode

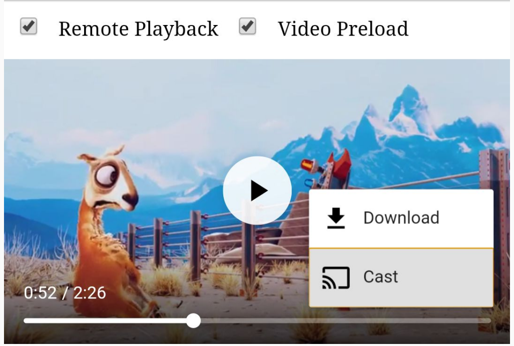


https://www.youtube.com/tv

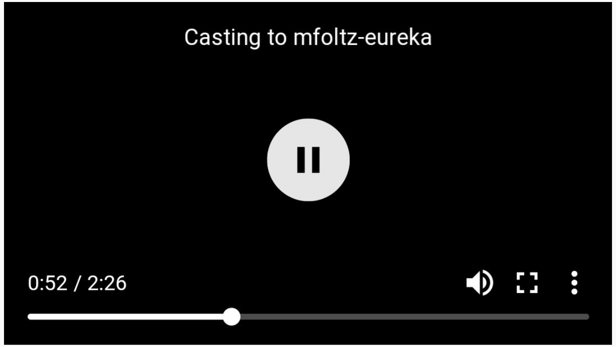
Remote Playback API

<audio> or <video> element can:

1. Watch for available remote displays
2. Request remote playback by calling `video.remote.prompt()`
3. Media commands are forwarded to the remote playback device



☑ Remote Playback ☑ Video Preload



Casting to mfoltz-eureka

Sample Code & Demos

1-UA:

https://googlechrome.github.io/samples/presentation-api/

https://googlechromelabs.github.io/presentation-api-samples/photowall/

2-UA: https://googlechrome.github.io/samples/presentation-api/cast.html

Remote Playback API:
https://beaufortfrancois.github.io/sandbox/media/remote-playback.html

Implementation Status: Chrome

	Desktop	Android
Presentation Controller (2-UA)	M51 May 2016	M48 Jan 2016
Presentation Receiver (1-UA)	M59 June 2017	
Remote Playback API		M56 Feb 2017

Community Group Rechartering & Scope

Address main feedback from TAG around **interoperability**

Controllers and receivers on **same LAN**

Presentation API: 2-UA mode ("**flinging**" **URL**)

Remote Playback API: Remote playback via **src= URL**

Extension ability for future use cases

Community Group Out Of Scope

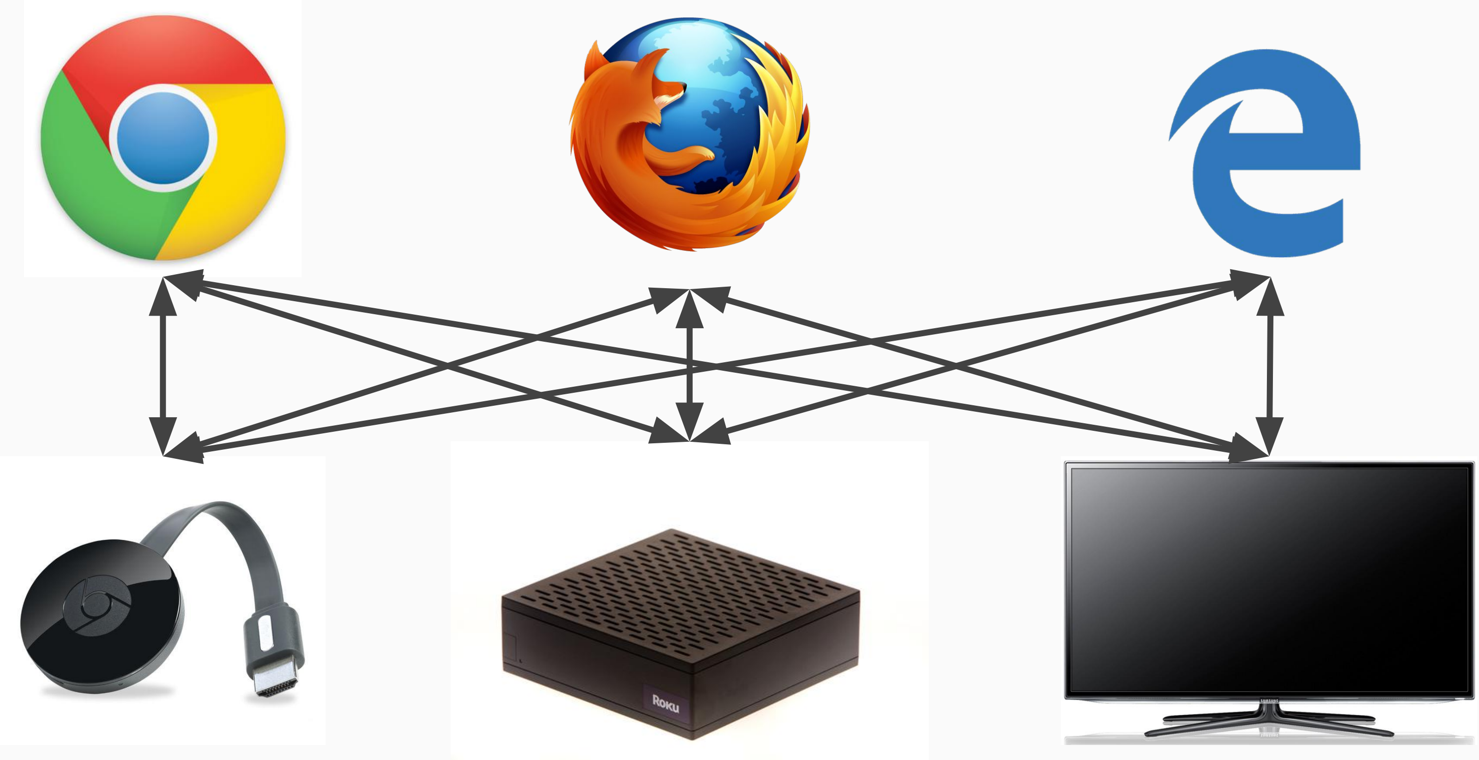
Media codecs

Streaming use cases: 1-UA mode for Presentations, MSE for remote playback

Network traversal / guest mode

Interoperability with proprietary protocols (DLNA, Google Cast, etc.)

Open Screen Protocol



Open Screen Protocol

Specify all network services & protocols needed to implement APIs

Can be deployed across a variety of devices and platforms

Re-use modern protocols and cryptography

Functional Requirements

1. Discovery of presentation receivers and controllers on a shared LAN
2. Implement Presentation API
    a. Determine display compatibility with a URL
    b. Creating a presentation and connection given a URL
    c. Reconnecting to a presentation
    d. Closing a connection
    e. Terminating a presentation
3. Reliable and in-order message exchange
4. Authentication and confidentiality
5. Implement Remote Playback API for <audio> and <video> src=

Non-functional

Usability

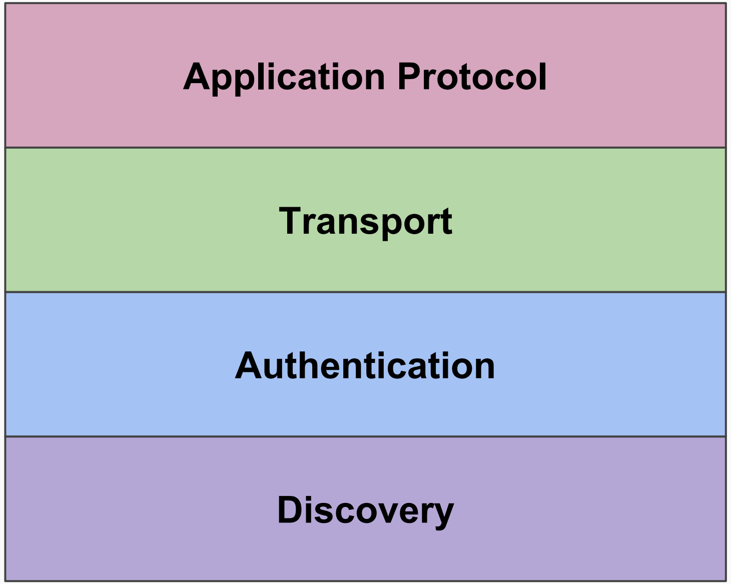
Privacy-preserving and secure

Resource efficient (battery, memory)

Implementation complexity on constrained devices

Extensibility and upgradeability

Open Screen Protocol - Stack



Application Protocol

Transport

Authentication

Discovery

Open Screen Protocol - Alternatives

Application Protocol	Custom binary, CBOR, Protocol Messages, JSON
Transport	TCP, WebSockets QUIC, RTCDataChannel, QUIC DataChannel
Authentication	TLS 1.3 (via QUIC or wss) S-PAKE, J-PAKE
Discovery	mDNS, DIAL, SSDP

Open Screen Protocol - "Modern"/"V1"

Application Protocol	Custom binary, **CBOR, Protocol Messages**, JSON
Transport	TCP, WebSockets QUIC, RTCDataChannel, **QUIC DataChannel**
Authentication	**TLS 1.3 (via QUIC** or wss:) S-PAKE, **J-PAKE**
Discovery	**mDNS**, DIAL, **SSDP**

Open Screen Protocol - WebSockets

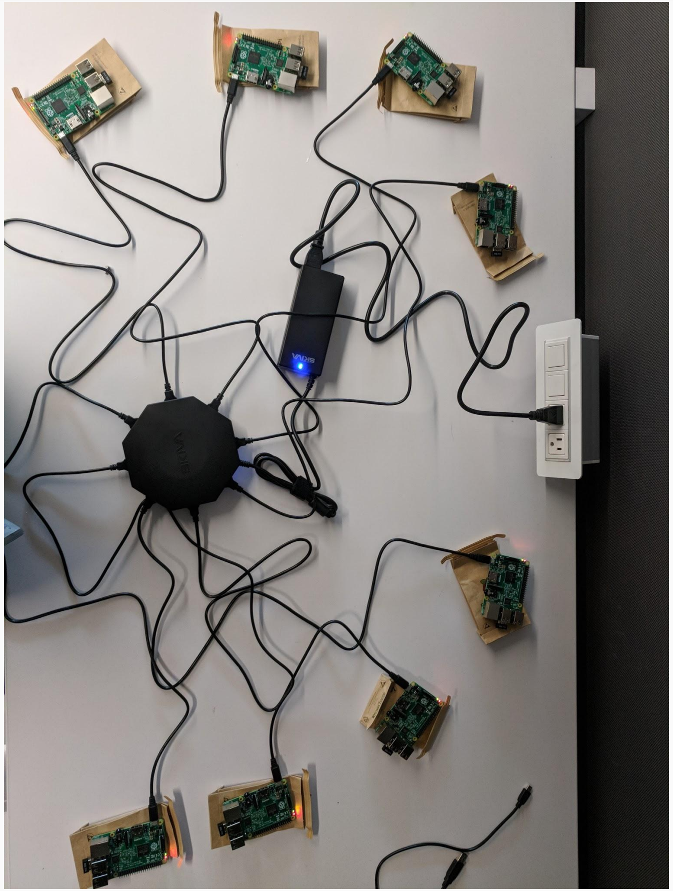
Application Protocol	Custom binary, CBOR, Protocol Messages, **JSON**
Transport	TCP, **WebSockets** QUIC, RTCDataChannel, QUIC DataChannel
Authentication	TLS 1.3 (via QUIC or **wss:**) S-PAKE, J-PAKE
Discovery	mDNS, **DIAL**, SSDP

Evaluation & Benchmarking

For each technology/protocol,

- Write up proposal for how it could be used
- Evaluate against requirements (performance, security, implementation)
- Write up proposal for benchmarking performance in the lab

"Open Screen Lab"



What has been accomplished

- Requirements for Presentation API and hardware specs
- Evaluations of
    - mDNS
    - SSDP/DIAL
    - QUIC
    - RTCDataChannel
- Control protocol for Presentation API ("custom binary")
- Benchmarking plans for discovery and transport

Major work items remaining for "V1"

- Discovery mechanisms - required vs. alternative
- QUIC DataChannel
  - Mapping control protocol
  - ICE integration
- Control protocol
  - Consensus on serialization
  - Update control protocol
- Authentication mechanisms
  - Integrate J-PAKE
  - Support PKI-based authentication with TLS

Discovery

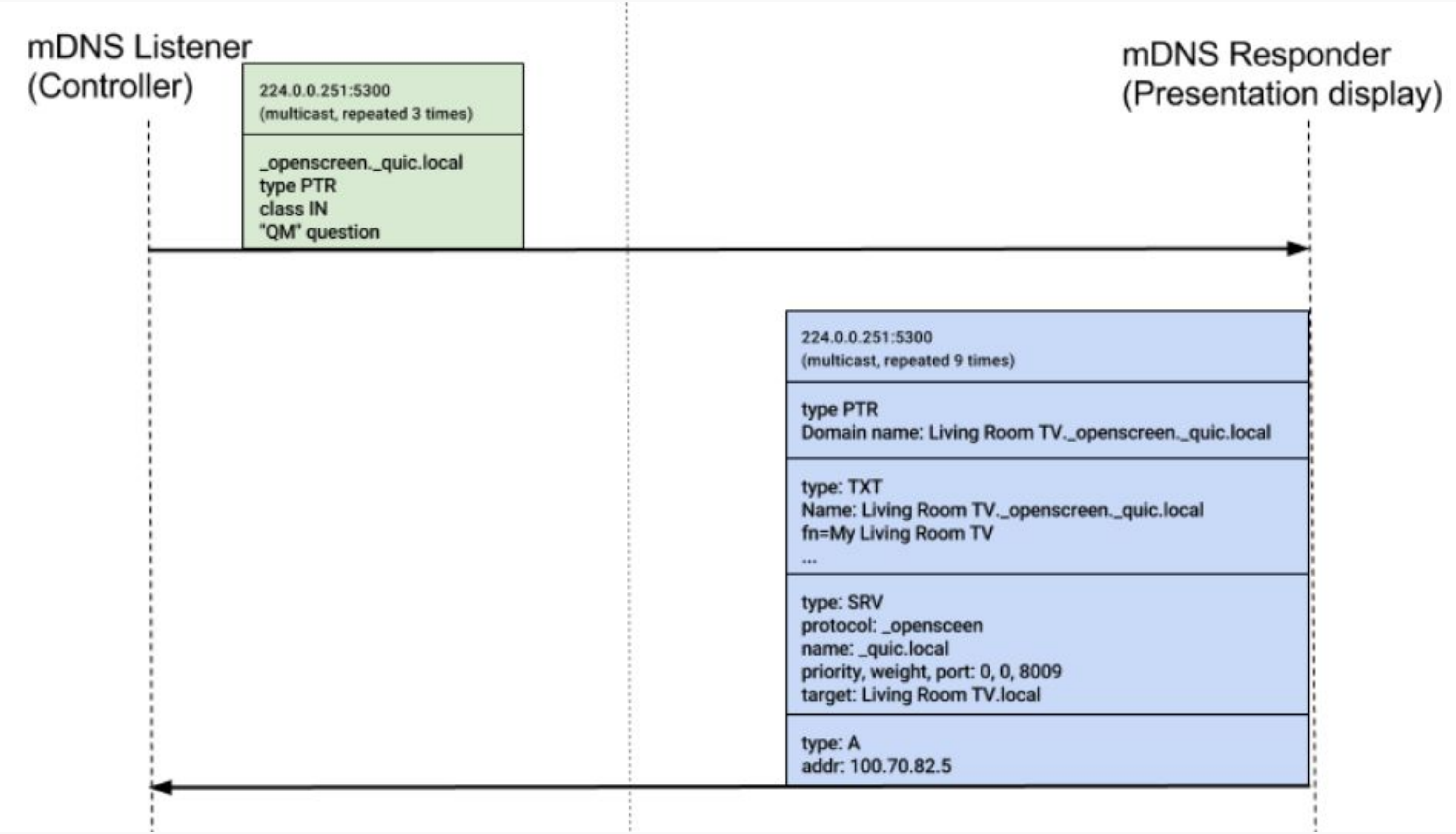
Discovery Topics

- Requirements, goals
- mDNS overview
- SSDP overview
- Implementation feedback & Chrome data
- GitHub issues
- Recommendations & next steps

Discovery: Requirements and Goals

- Allow Open Screen devices to discover each other on the LAN
- Publish enough data to bootstrap connections
    - IP, port, friendly name
- Responsive to receiver addition and removal
- Power efficient and scalable
- Secure: prevent device compromise

mDNS - Query & Response



mDNS Listener
(Controller)

mDNS Responder
(Presentation display)

224.0.0.251:5300
(multicast, repeated 3 times)

_openscreen._quic.local
type PTR
class IN
"QM" question

224.0.0.251:5300
(multicast, repeated 9 times)

type PTR
Domain name: Living Room TV._openscreen._quic.local

type: TXT
Name: Living Room TV._openscreen._quic.local
fn=My Living Room TV
...

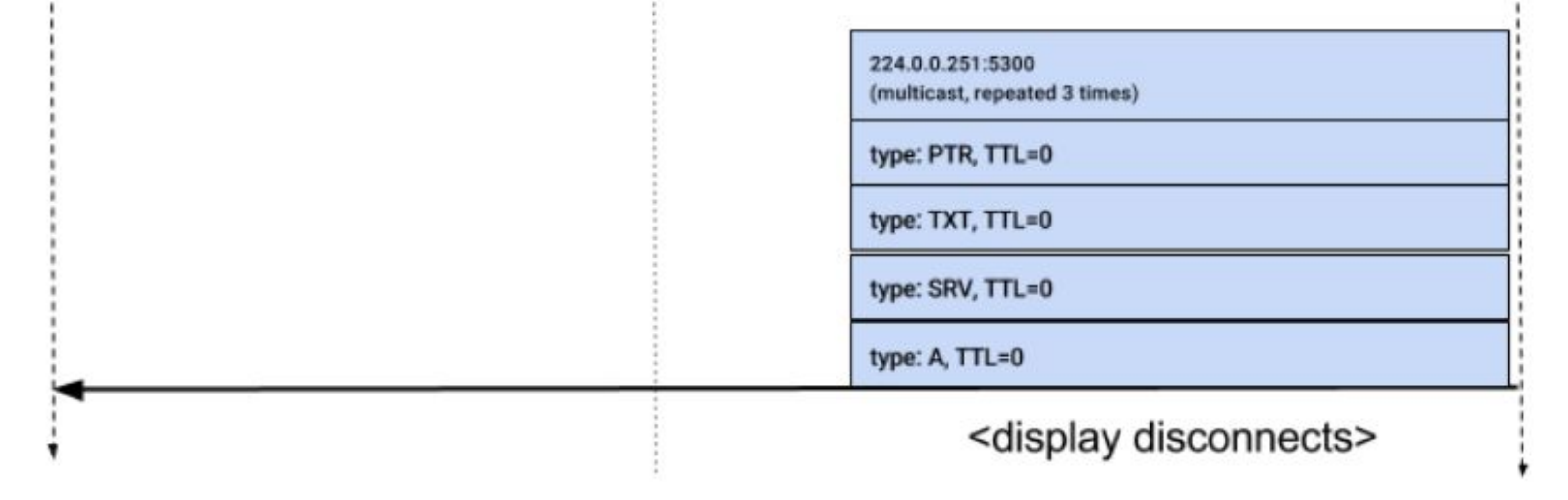
type: SRV
protocol: _opensceen
name: _quic.local
priority, weight, port: 0, 0, 8009
target: Living Room TV.local

type: A
addr: 100.70.82.5

mDNS - Disconnection

mDNS Listener

mDNS Responder



224.0.0.251:5300
(multicast, repeated 3 times)

type: PTR, TTL=0

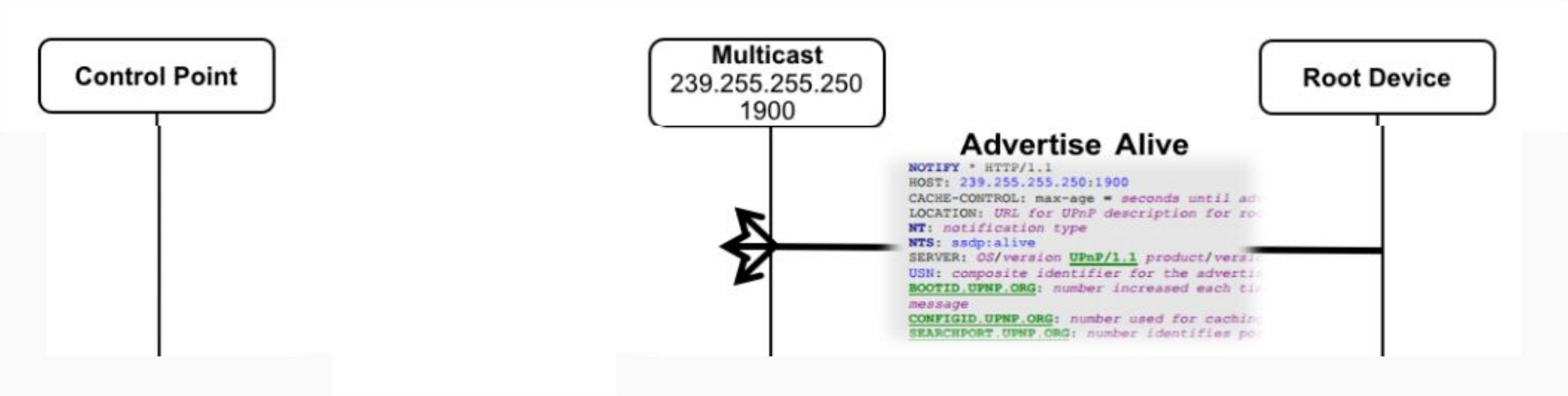
type: TXT, TTL=0

type: SRV, TTL=0

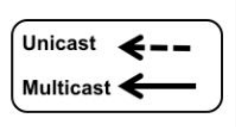
type: A, TTL=0

<display disconnects>

SSDP: Advertisement



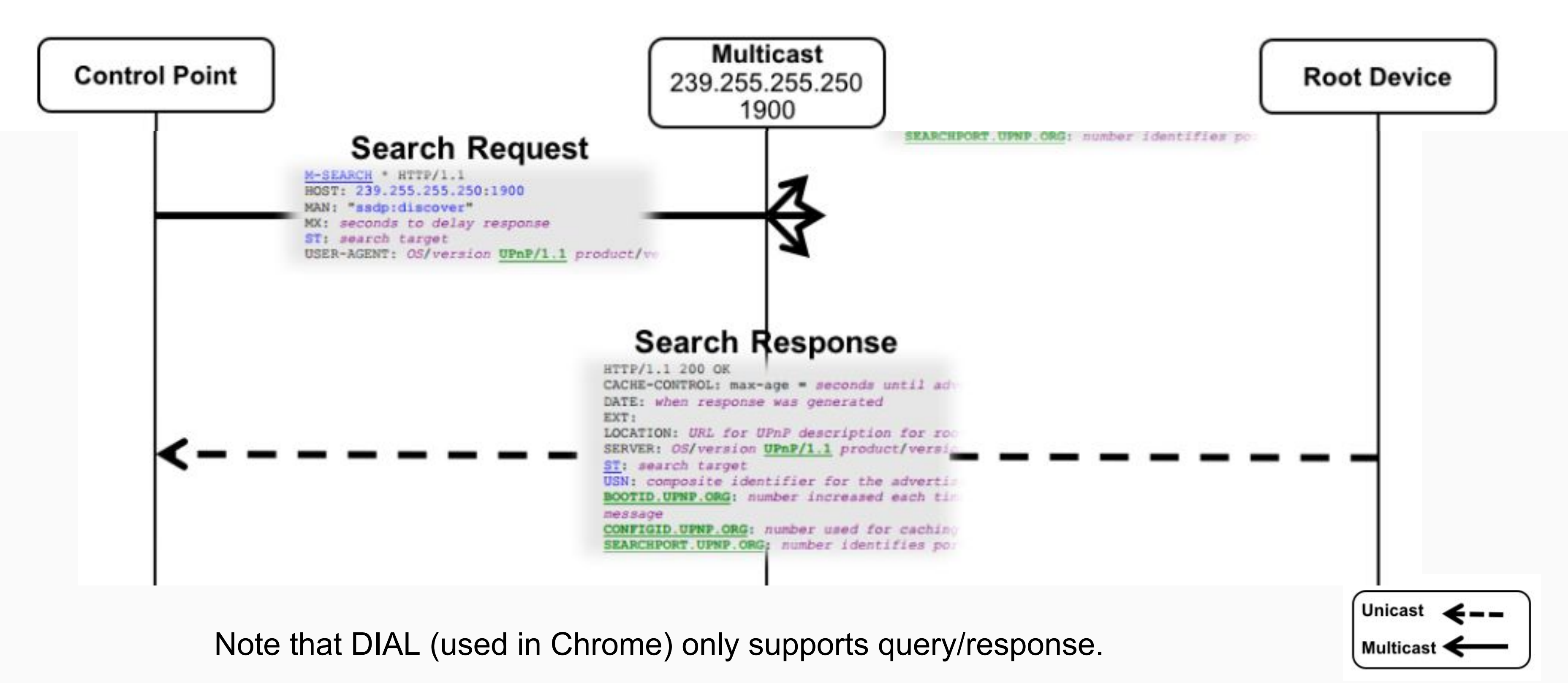
```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age = seconds until ad
LOCATION: URL for UPnP description for ro
NT: notification type
NTS: ssdp:alive
SERVER: OS/version UPnP/1.1 product/versi
USN: composite identifier for the adverti
BOOTID.UPNP.ORG: number increased each ti
message
CONFIGID.UPNP.ORG: number used for cachin
SEARCHPORT.UPNP.ORG: number identifies po
```



SSDP: Advertisement

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age = 1800 [response lifetime]
NTS: ssdp:alive
SERVER: OS/version product/version
USN: XXX-XXX-XXX-XXX [UUID for device]
NT: urn:openscreen-org:service:openscreenreceiver:1
FRIENDLY-NAME.openscreen.org: TXkgUHJlc2VudGF0aW9uIERpc3BsYXk= [My
Presentation Display]
RECEIVER.openscreen.org: 192.168.1.100:3000
```

SSDP: Query/Response



Search Request

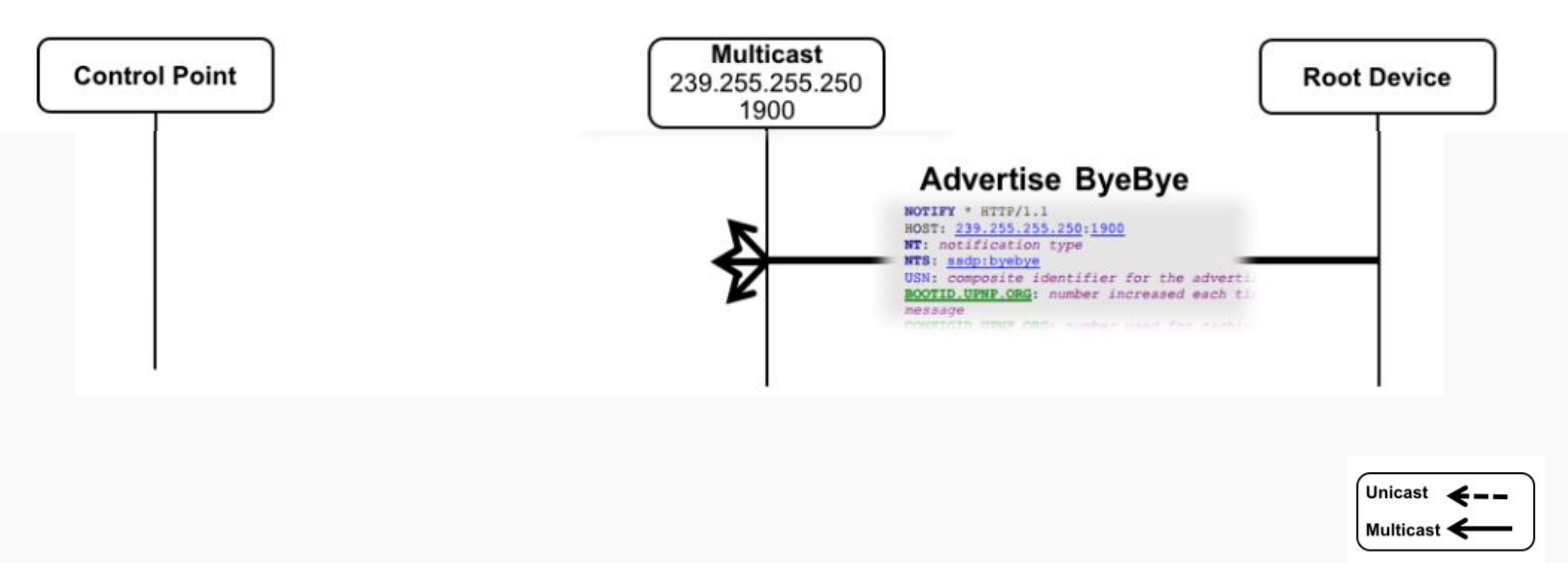
```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: seconds to delay response
ST: search target
USER-AGENT: OS/version UPnP/1.1 product/v
```

Search Response

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until ad
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description for roo
SERVER: OS/version UPnP/1.1 product/versi
ST: search target
USN: composite identifier for the adverti
BOOTID.UPNP.ORG: number increased each ti
message
CONFIGID.UPNP.ORG: number used for cachin
SEARCHPORT.UPNP.ORG: number identifies po
```

Note that DIAL (used in Chrome) only supports query/response.

SSDP: Disconnection

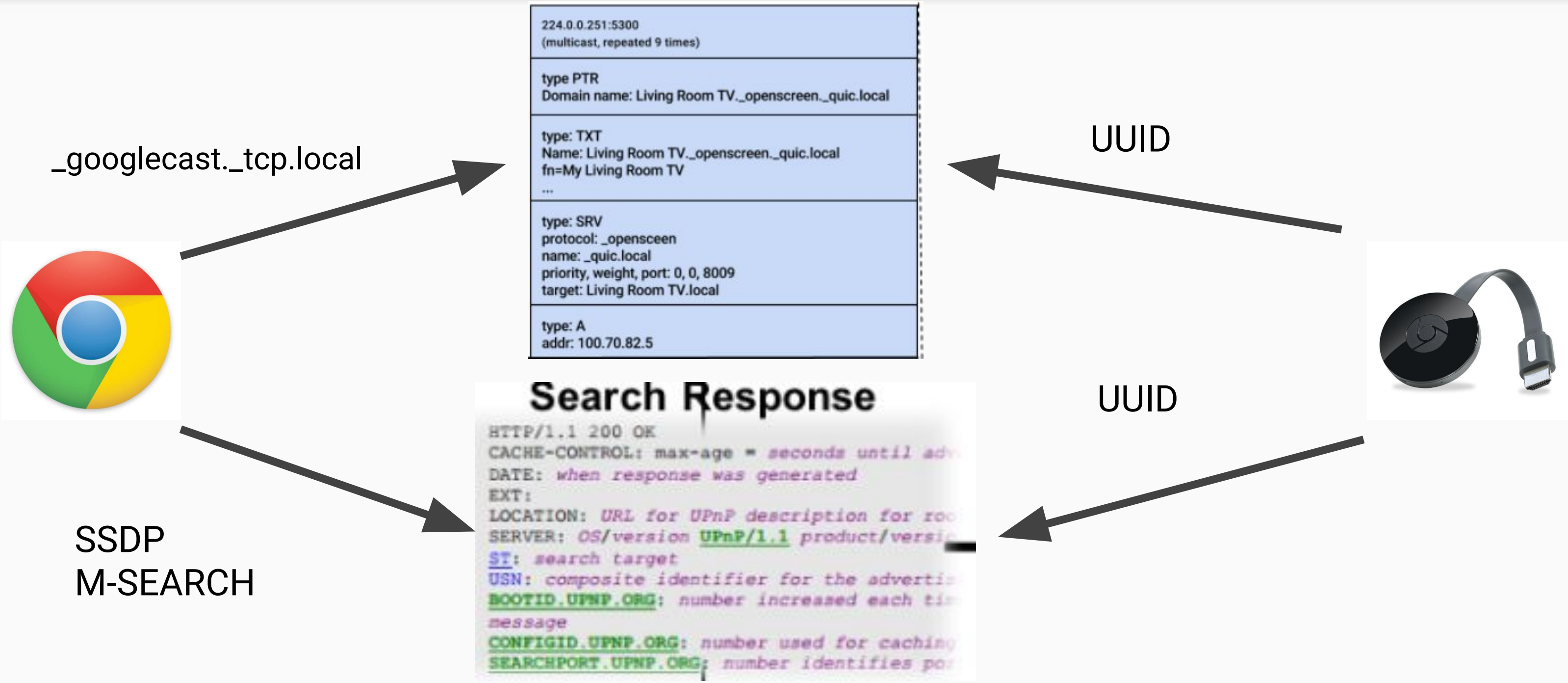


```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
NT: notification type
NTS: ssdp:byebye
USN: composite identifier for the adverti
BOOTID.UPNP.ORG: number increased each ti
message
CONFIGID.UPNP.ORG: number used for cachi
```

Discovery: Problems!

- Firewalls by OS and security software
- Routers/middleboxes configurations
- Other software/services block ports
- Enterprise policies
- ???

Chromecast Dual Discovery

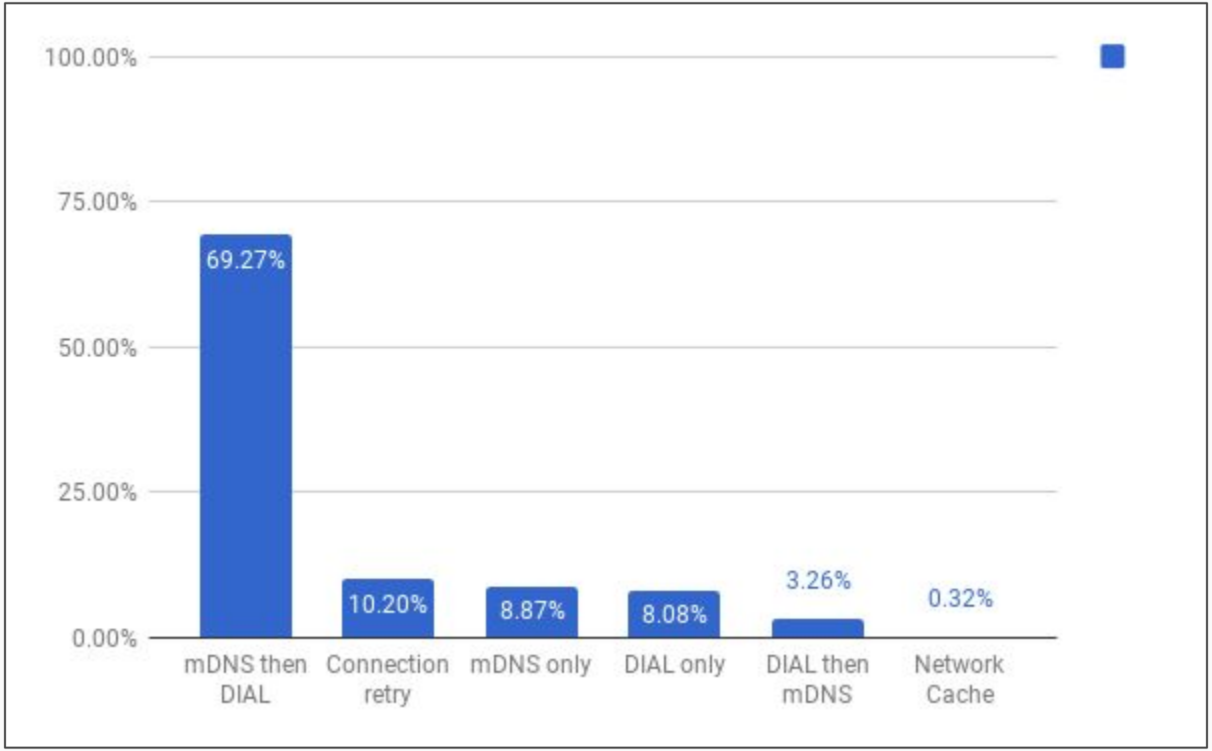


```
224.0.0.251:5300
(multicast, repeated 9 times)

type PTR
Domain name: Living Room TV._openscreen._quic.local

type: TXT
Name: Living Room TV._openscreen._quic.local
fn=My Living Room TV
...

type: SRV
protocol: _opensceen
name: _quic.local
priority, weight, port: 0, 0, 8009
target: Living Room TV.local

type: A
addr: 100.70.82.5
```

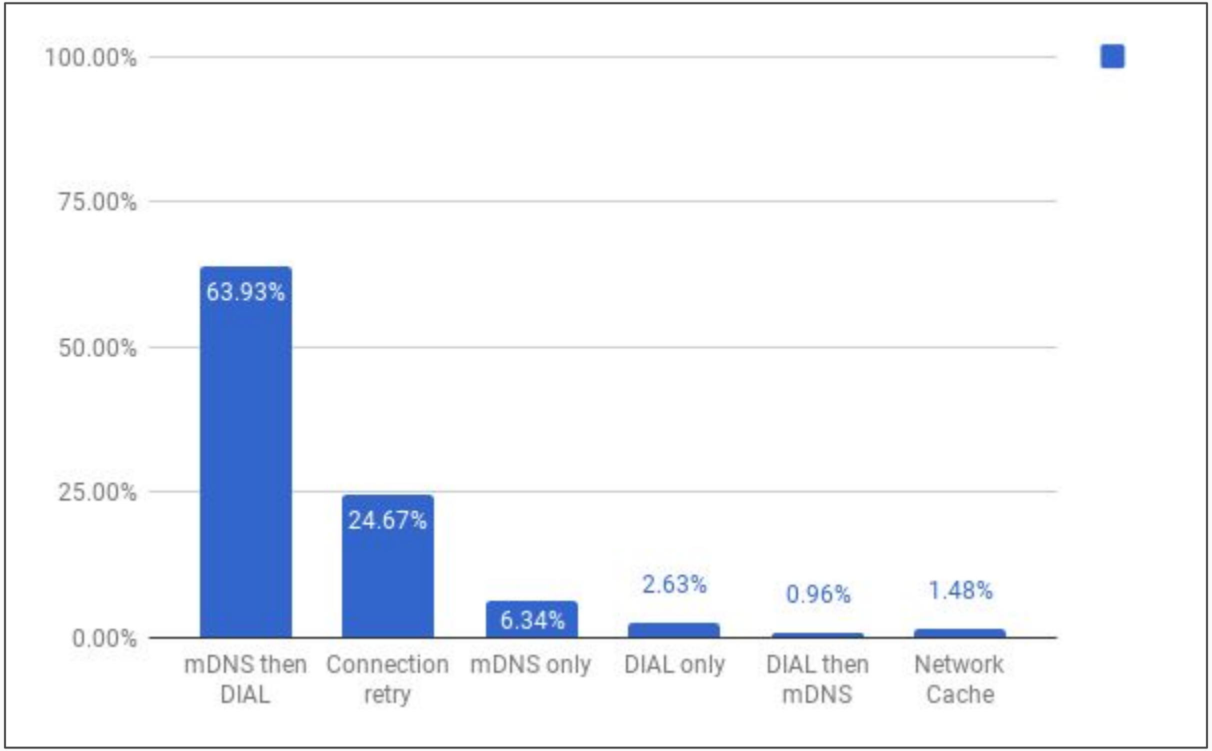
Search Response

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until ad
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description for roo
SERVER: OS/version UPnP/1.1 product/versi
ST: search target
USN: composite identifier for the adverti
BOOTID.UPNP.ORG: number increased each ti
message
CONFIGID.UPNP.ORG: number used for cachin
SEARCHPORT.UPNP.ORG: number identifies po
```

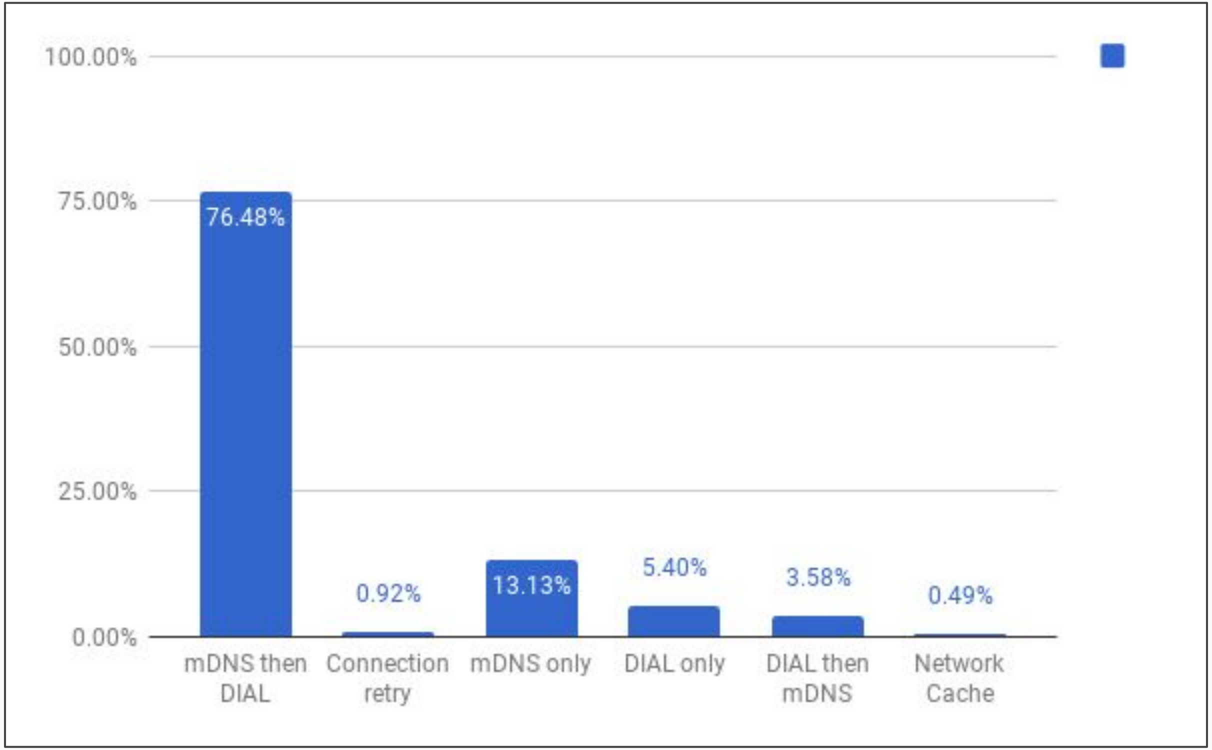
Dual Discovery: Windows



Dual Discovery: Mac



Dual Discovery: ChromeOS



Dual Discovery: Summary

How many would you find if you found 100 by dual discovery?

	mDNS only	**DIAL only**	**Dual**
Windows	91	90	100
Mac	96	91	100
ChromeOS	95	87	100

Dual Discovery: Conclusions

1. Across platforms, mDNS is more likely to find a given device.
2. About 5% of failures can be attributed to network issues.
3. Windows has a failure rate of 10% for both mDNS and DIAL.
4. Adding DIAL improves reliability by 5-10%.

Discovery: Recommendations

1. Across platforms, mDNS is more likely to find a given device.
2. About 5% of failures can be attributed to network issues.
3. Windows has a failure rate of 10% for both mDNS and DIAL.
4. Adding DIAL improves reliability by 5-10%.

mDNS should be mandatory for controllers and receivers.

SSDP should be specified as an alternative, but not moved forward as part of the core protocol.

Evaluate additional discovery mechanisms (including SSDP) for the future.

GitHub issues

Issue #81: [SSDP] Update implementation information

Issue #57: [SSDP] Update proposed use of SSDP to specifically prevent SSDP amplification attacks

Issue #21: Investigate mechanisms to pre-filter devices by Presentation URL

Postpone to v2?

Transport

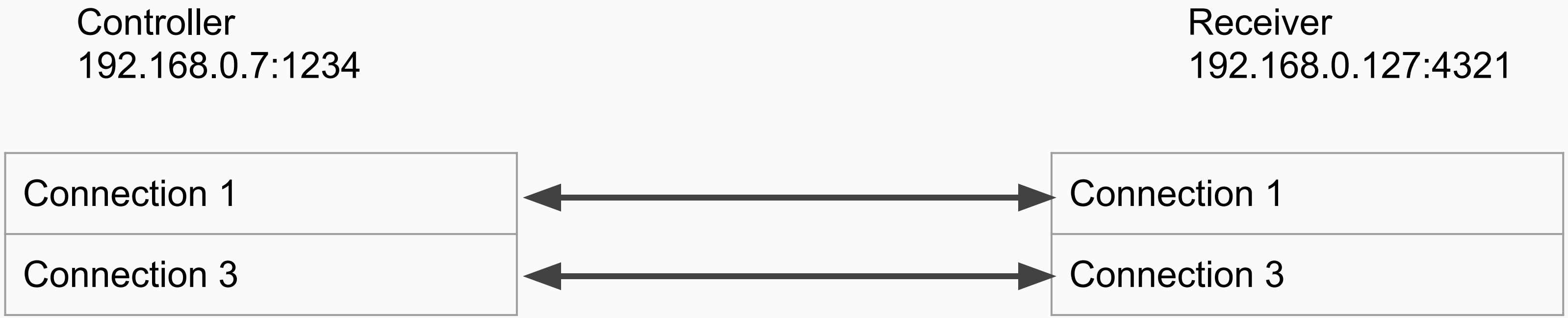
Transport topics

- Requirements, QUIC overview
- QUIC DataChannel background
- Application protocol mapping
- QUIC DataChannel bootstrapping and authentication
- ORTC API
- GitHub issues
- Proposals, next steps

QUIC Overview

- Reliable, connection-oriented byte streams over UDP
- Multiple streams can be sent without head-of-line blocking
- Streams support message based or streaming payloads (media)
- Supports pluggable authentication handshake
- Supports alternative congestion control (BBR)
- Supports 0-RTT TLS 1.3 session resumption

QUIC Connections



Controller
192.168.0.7:1234

Receiver
192.168.0.127:4321

Connection 1

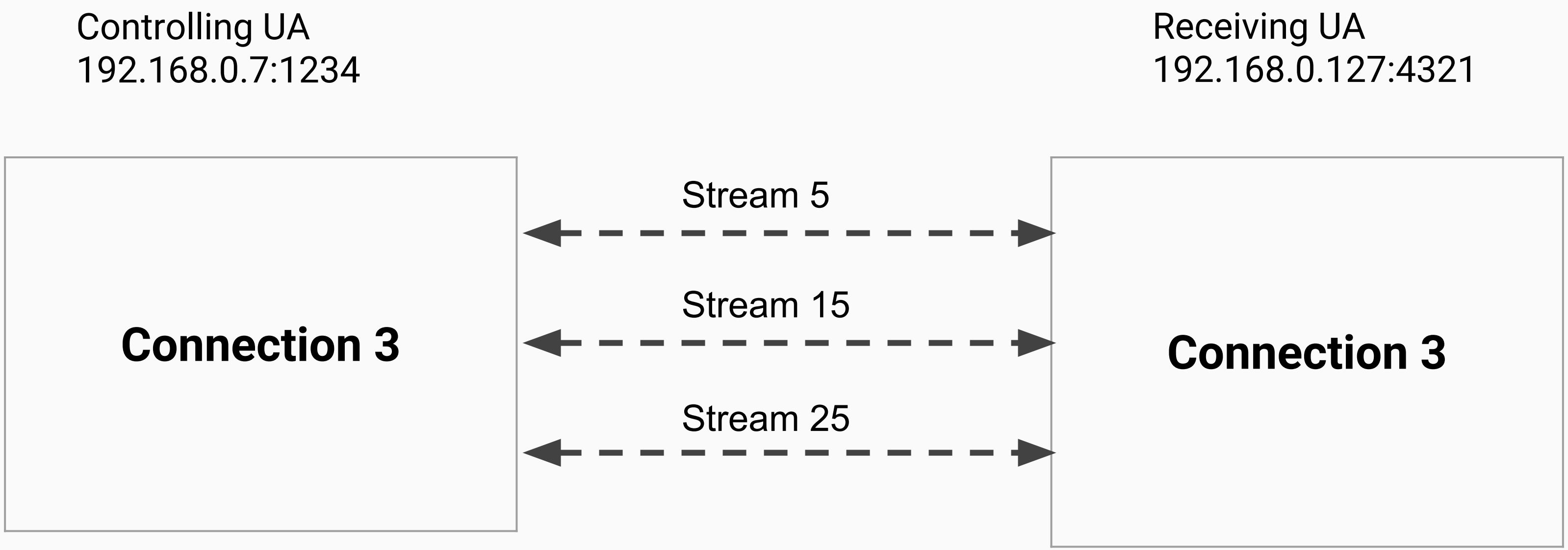
Connection 3

Connection 1

Connection 3

Each connection uses a separate crypto handshake.
This assumes port sharing which may not be in v1.
https://github.com/quicwg/base-drafts/issues/714

QUIC Streams



Very lightweight; can be 0 to 2^62 bytes and spread among packets.

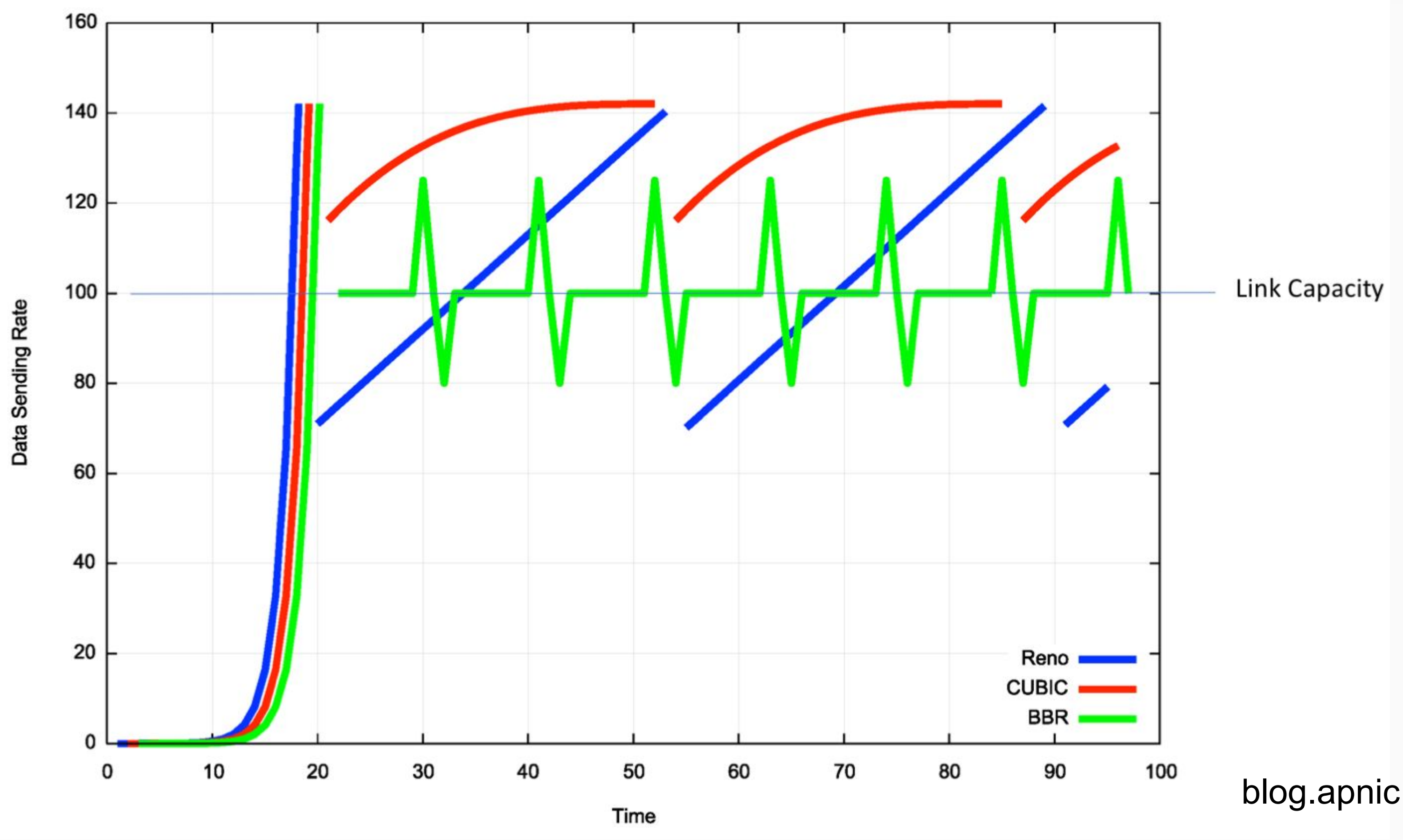
QUIC protocol mapping (multi connection)

Control channel between controlling and receiving user agent	QUIC connection
Control channel command/response	QUIC stream (id for ordering)
PresentationConnection between controlling page and presentation	Separate QUIC connection
PresentationConnection message	QUIC stream (id for ordering)

QUIC protocol mapping (single connection)

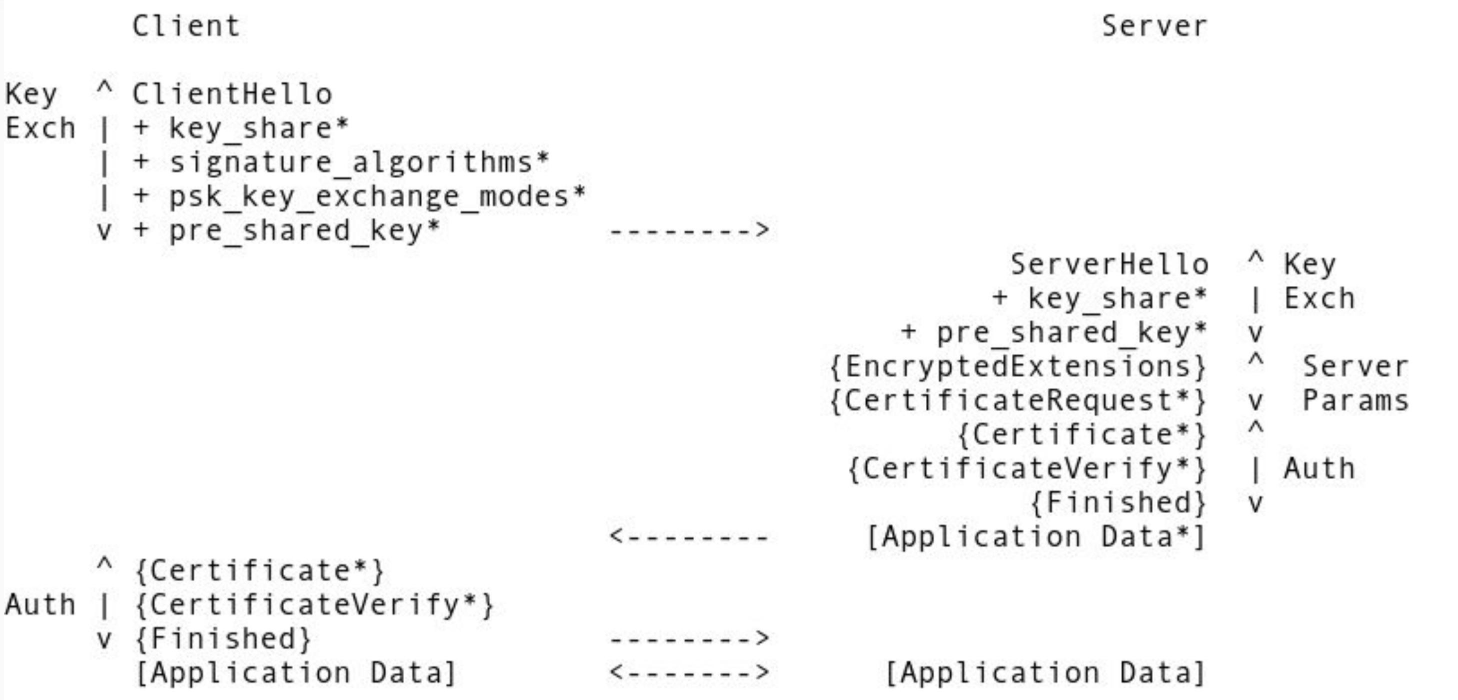
Control channel between controlling and receiving user agent	Fixed QUIC stream id
Control channel command/response	Separate QUIC stream
PresentationConnection between controlling page and presentation	Separate QUIC stream
PresentationConnection message	Uses existing stream for connection

QUIC Congestion Control (BBR)



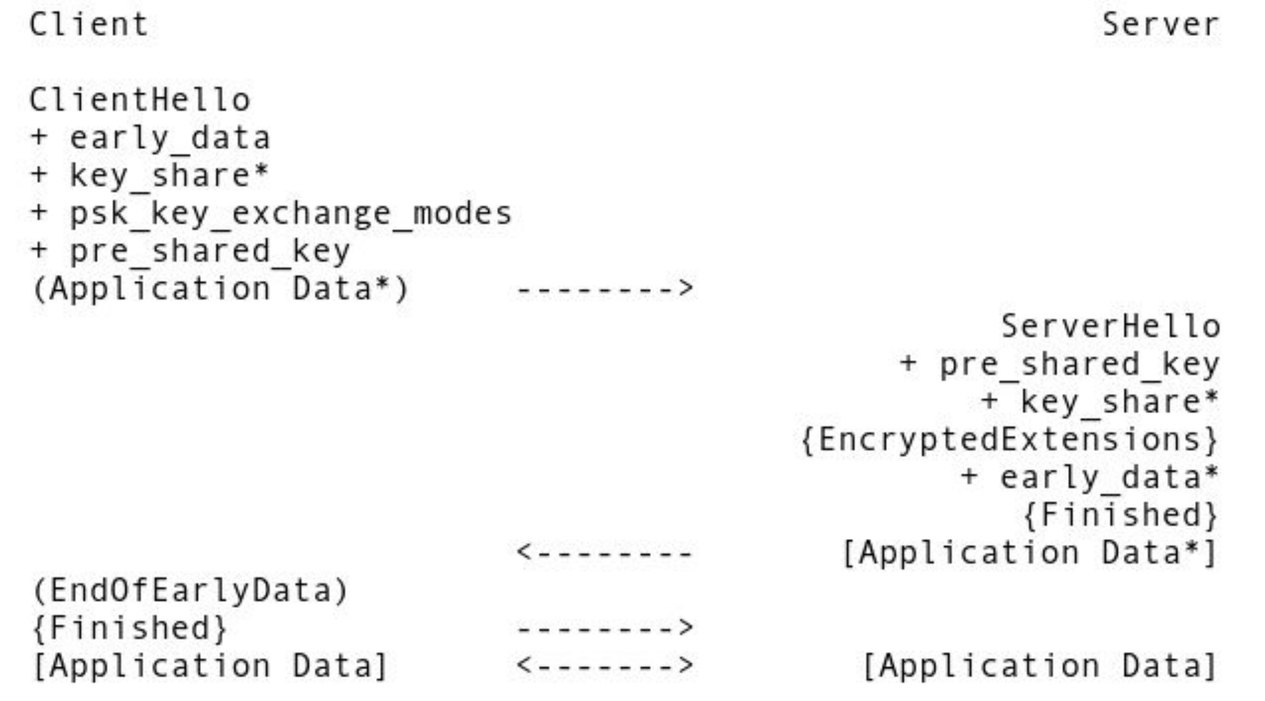
blog.apnic.net

QUIC Handshake (1 RTT)



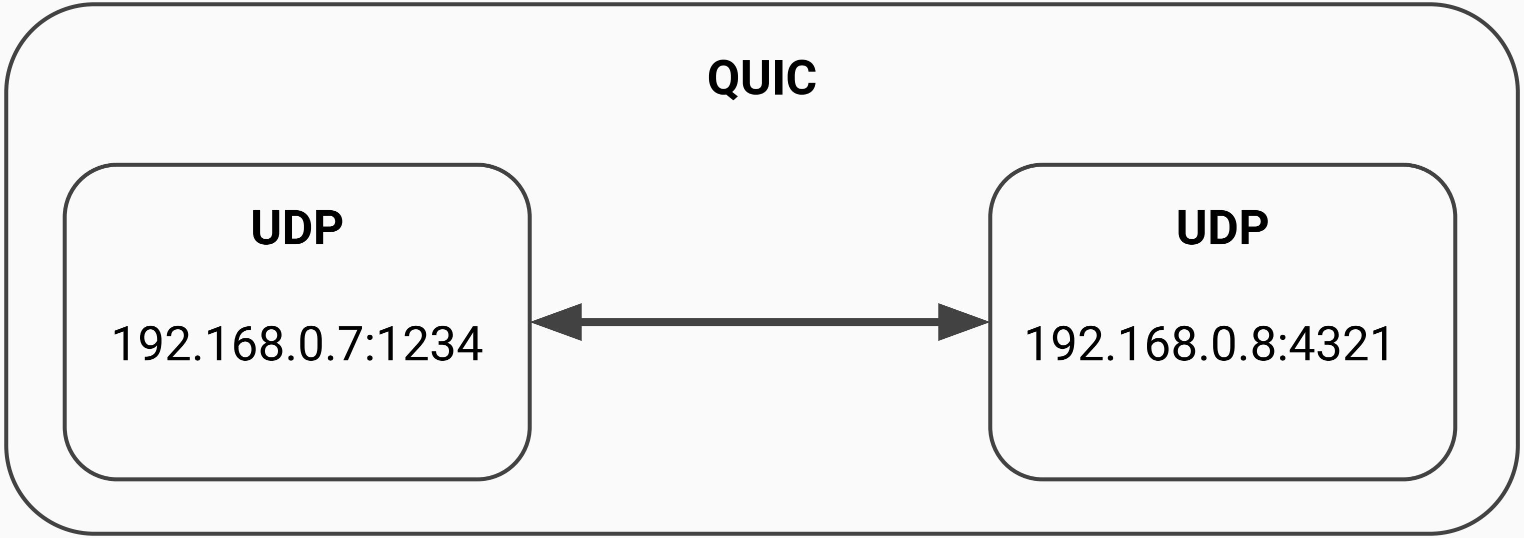
```
       Client                                               Server

Key  ^ ClientHello
Exch | + key_share*
     | + signature_algorithms*
     | + psk_key_exchange_modes*
     v + pre_shared_key*          -------->
                                                       ServerHello  ^ Key
                                                      + key_share*  | Exch
                                                 + pre_shared_key*  v
                                             {EncryptedExtensions}  ^  Server
                                             {CertificateRequest*}  v  Params
                                                    {Certificate*}  ^
                                              {CertificateVerify*}  | Auth
                                                        {Finished}  v
                                 <--------     [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}                -------->
       [Application Data]        <------->      [Application Data]
```

QUIC Handshake (0 RTT)

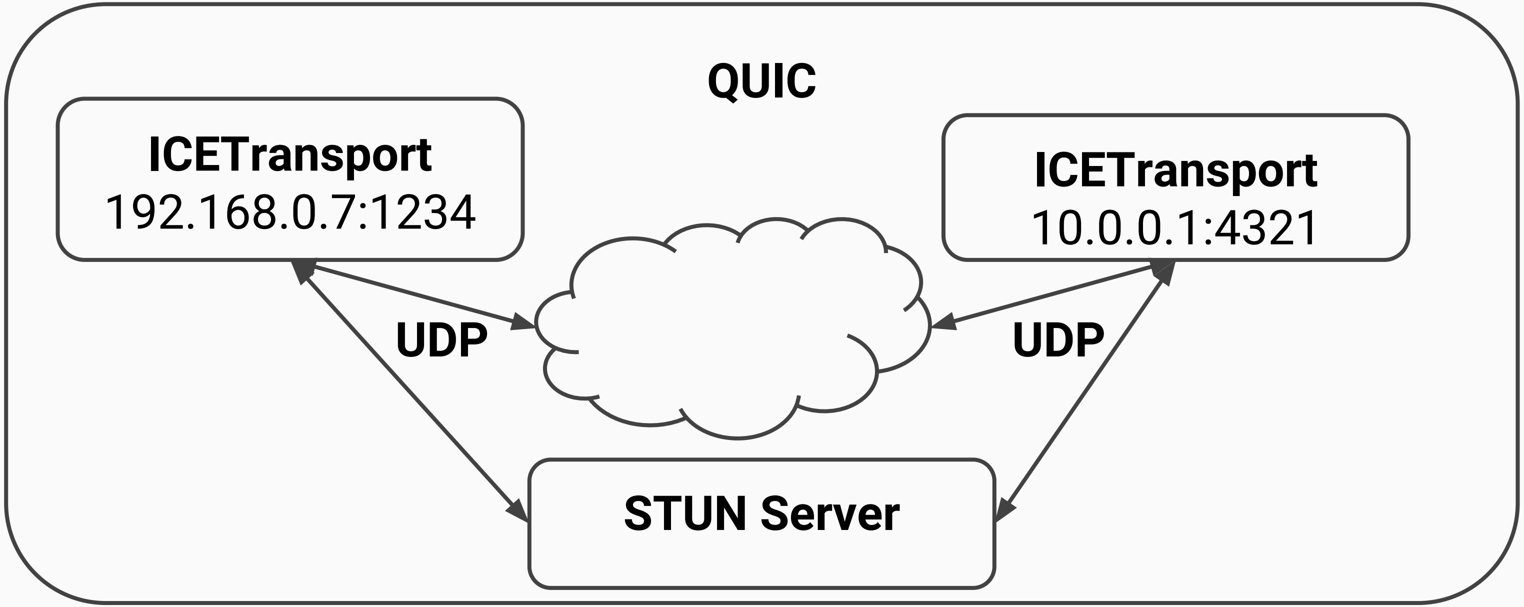


```
Client                                               Server

ClientHello
+ early_data
+ key_share*
+ psk_key_exchange_modes
+ pre_shared_key
(Application Data*)     -------->
                                                ServerHello
                                           + pre_shared_key
                                               + key_share*
                                      {EncryptedExtensions}
                                              + early_data*
                                                 {Finished}
                        <--------       [Application Data*]
(EndOfEarlyData)
{Finished}              -------->
[Application Data]      <------->        [Application Data]
```

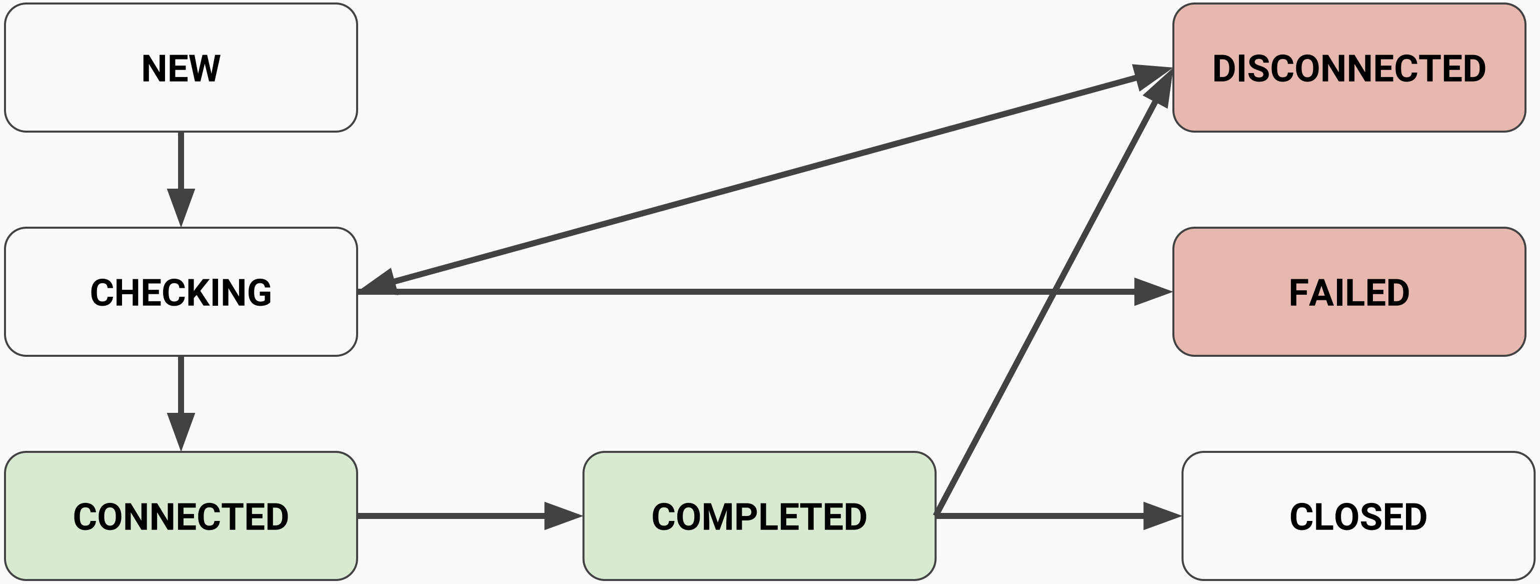
QUIC DataChannel



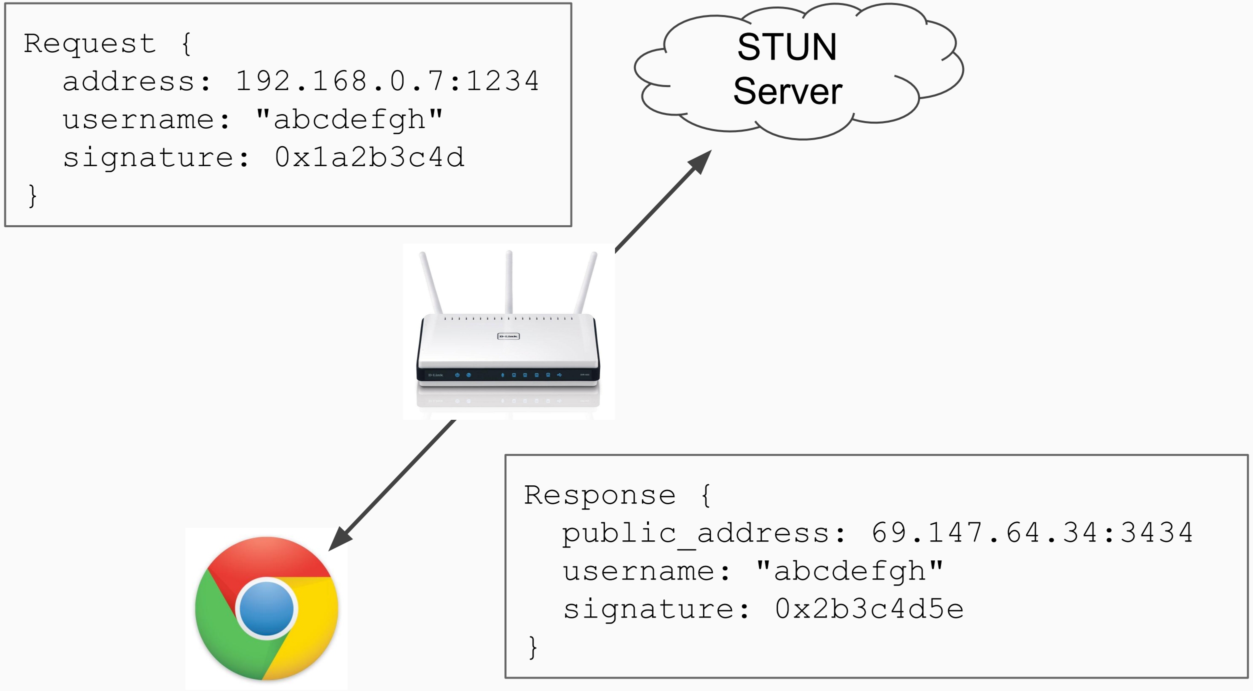
QUIC DataChannel - ICE



ICE State Machine



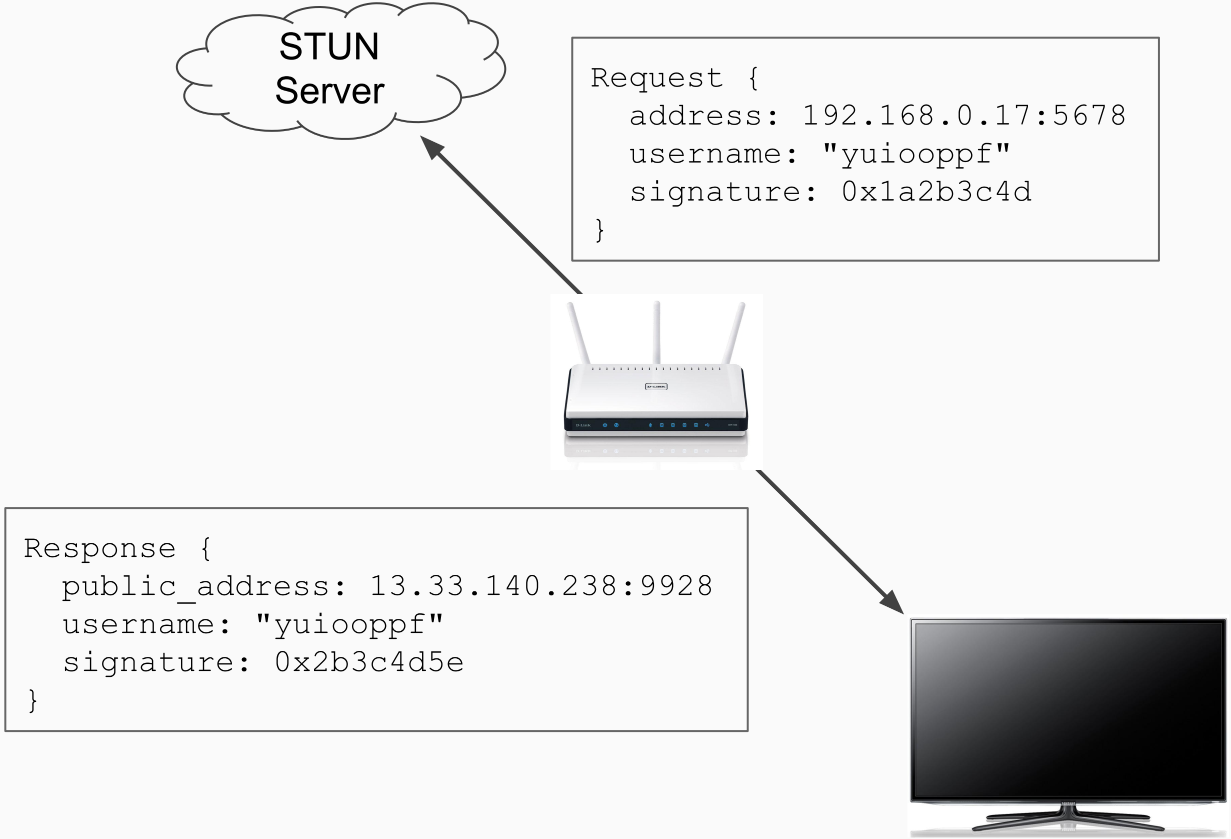
STUN Binding



```
Request {
  address: 192.168.0.7:1234
  username: "abcdefgh"
  signature: 0x1a2b3c4d
}
```

```
Response {
  public_address: 69.147.64.34:3434
  username: "abcdefgh"
  signature: 0x2b3c4d5e
}
```

STUN Binding

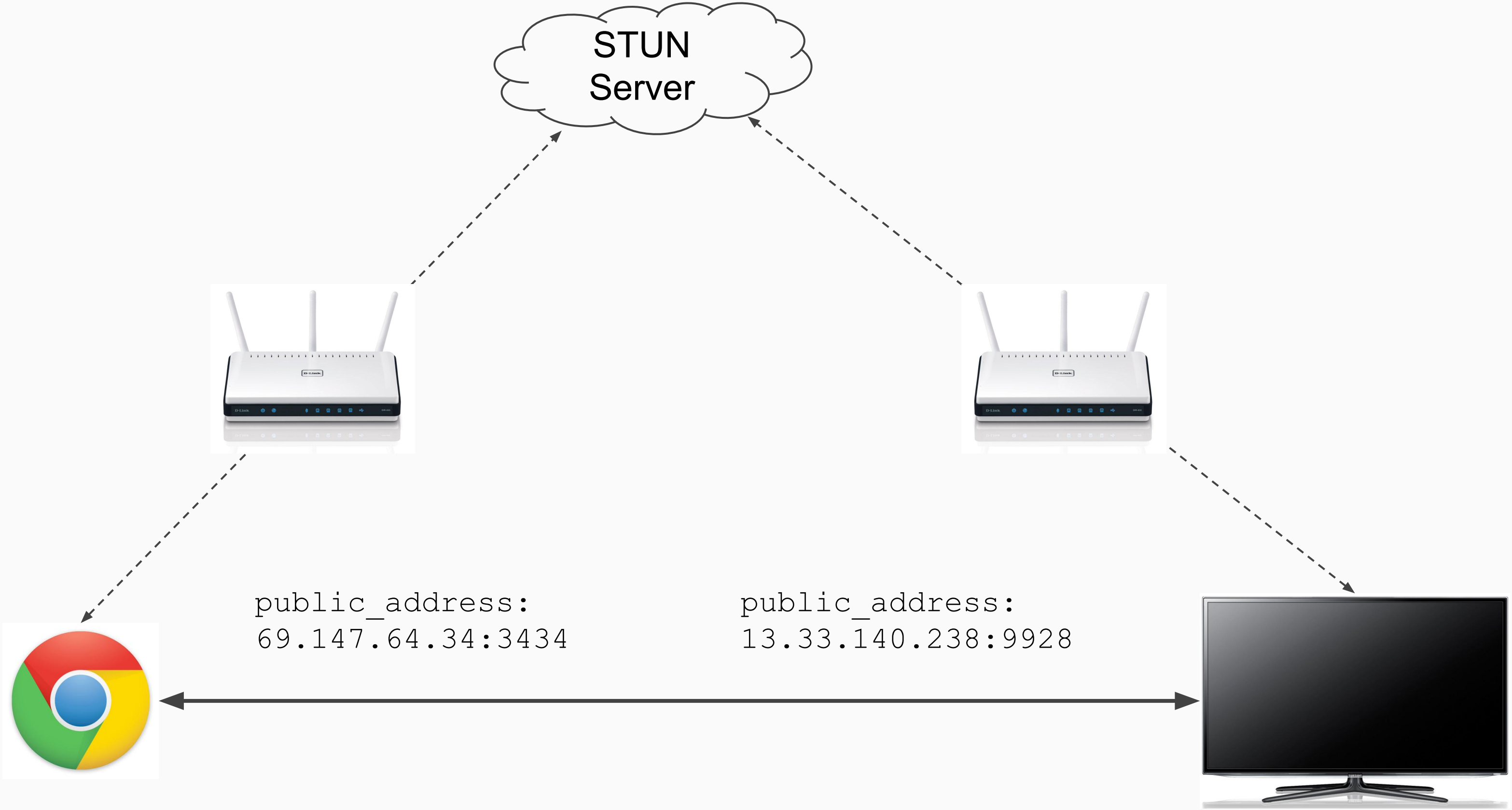


STUN Server

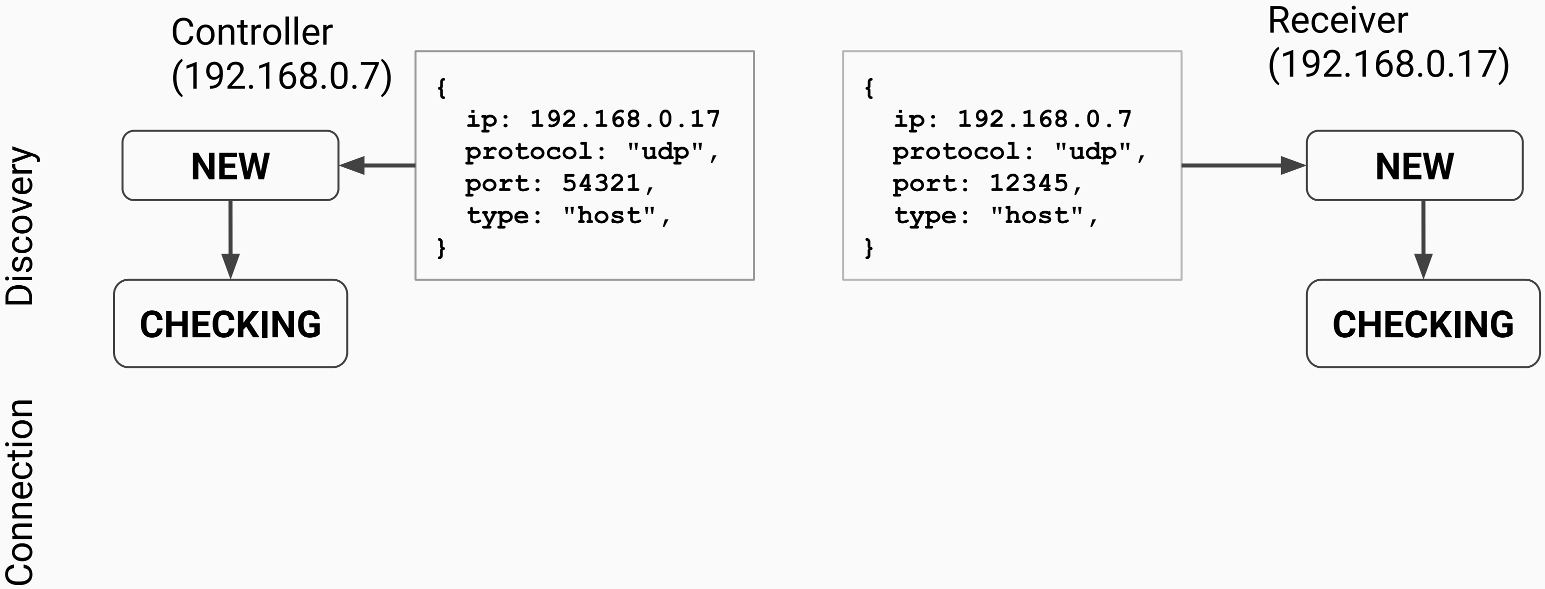
```
Request {
  address: 192.168.0.17:5678
  username: "yuiooppf"
  signature: 0x1a2b3c4d
}
```

```
Response {
  public_address: 13.33.140.238:9928
  username: "yuiooppf"
  signature: 0x2b3c4d5e
}
```

STUN Binding



QUIC DataChannel ICE Bootstrapping (LAN)



Controller
(192.168.0.7)

NEW

CHECKING

```
{
  ip: 192.168.0.17
  protocol: "udp",
  port: 54321,
  type: "host",
}
```

```
{
  ip: 192.168.0.7
  protocol: "udp",
  port: 12345,
  type: "host",
}
```

Receiver
(192.168.0.17)

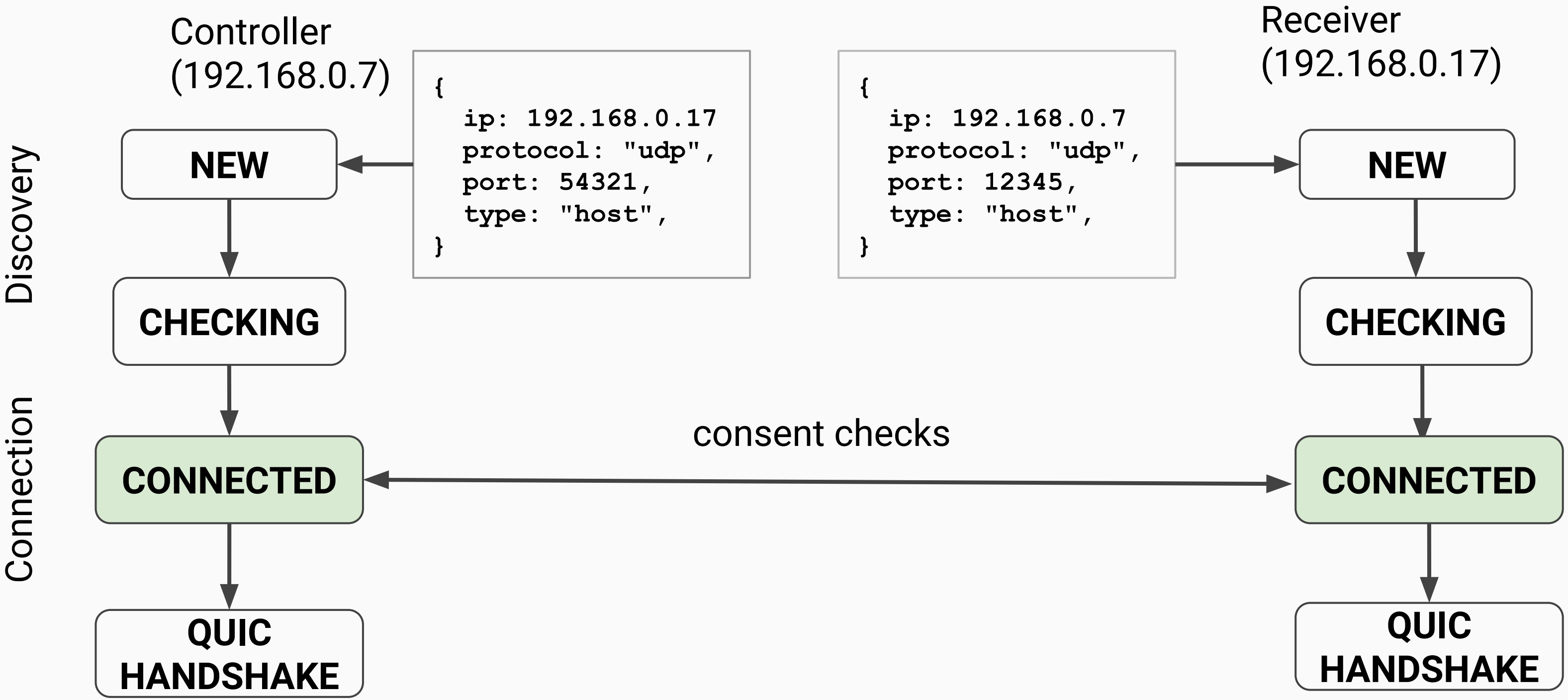
NEW

CHECKING

Discovery

Connection

QUIC DataChannel ICE Bootstrapping (LAN)



```
{
  ip: 192.168.0.17
  protocol: "udp",
  port: 54321,
  type: "host",
}
```

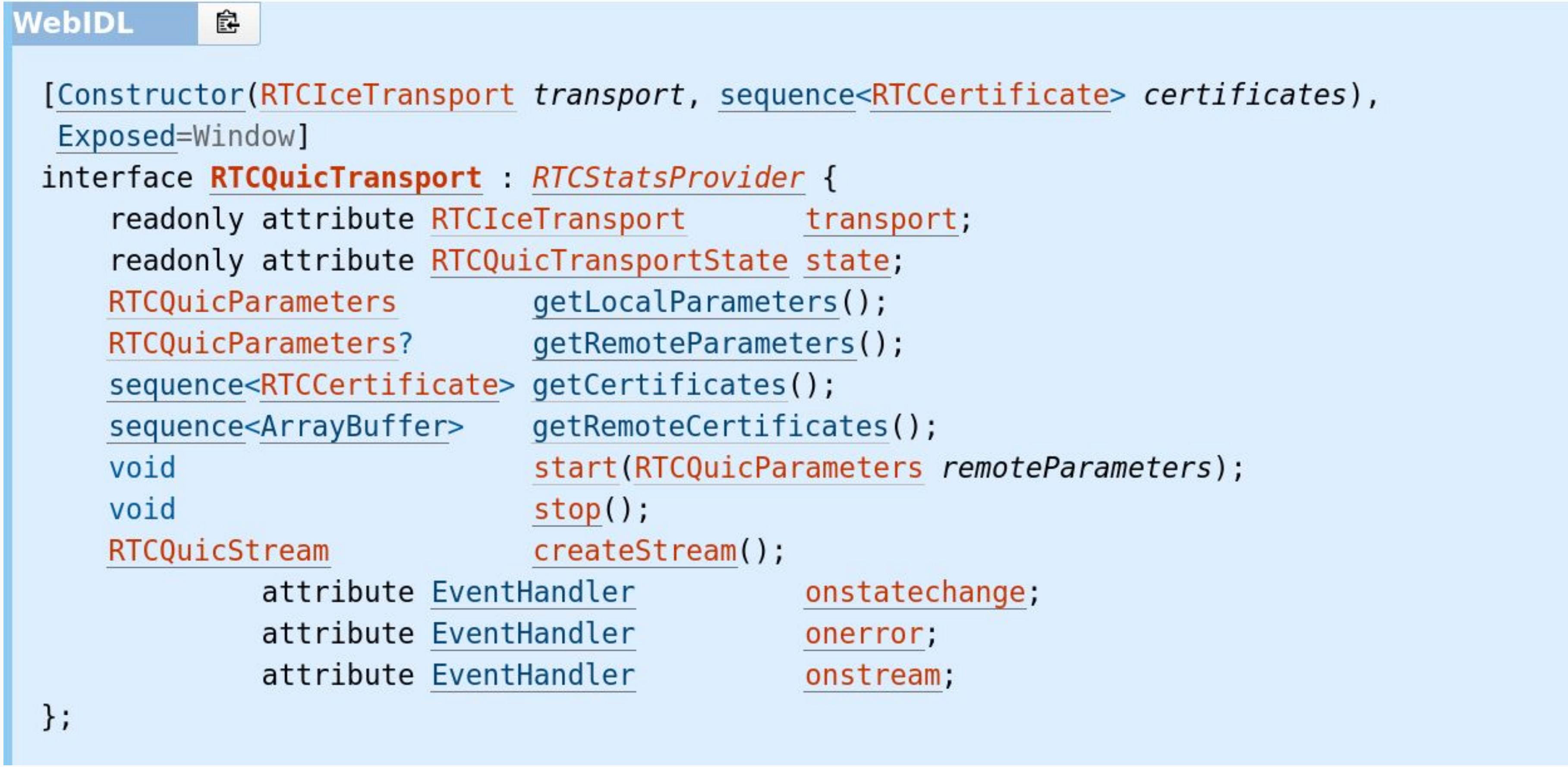
```
{
  ip: 192.168.0.7
  protocol: "udp",
  port: 12345,
  type: "host",
}
```

Q: do we need dummy STUN server for this to work?

QUIC DataChannel authentication

- Each side obtains (or generates) an RTCCertificate
- Passes the certificate fingerprint to the other party by secure signaling
- The fingerprint is passed into the data channel after ICE connects to initiate TLS handshake
- Can extract keying material from QUIC connection for separate auth step

ORTC



WebIDL

```
[Constructor(RTCIceTransport transport, sequence<RTCCertificate> certificates),
 Exposed=Window]
interface RTCQuicTransport : RTCStatsProvider {
    readonly attribute RTCIceTransport       transport;
    readonly attribute RTCQuicTransportState state;
    RTCQuicParameters         getLocalParameters();
    RTCQuicParameters?        getRemoteParameters();
    sequence<RTCCertificate> getCertificates();
    sequence<ArrayBuffer>     getRemoteCertificates();
    void                      start(RTCQuicParameters remoteParameters);
    void                      stop();
    RTCQuicStream             createStream();
             attribute EventHandler          onstatechange;
             attribute EventHandler          onerror;
             attribute EventHandler          onstream;
};
```

ORTC

```
const ice = new RTCIceTransport(new RTCIceGatherer({/* ICE options */}));

const localCert = RTCCertificate.generateCertificate(/* algorithm */);
/* Send local certificate fingerprint via signaling */

const quic = new RTCQuicTransport(ice, [localCert]);

quic.onstatechange = _ => {
  if (quic.state == 'connected') {
    const stream = quic.createStream();
    stream.waitForWritable.then(_ => write(...));
    stream.waitForReadable.then(_ => readInto(...));
    stream.finish();
  }
};

/* Await remote certificate fingerprint from signaling */
quic.start({role = "auto", fingerprints = ["deadbeef"]});
```

Implementation Status

Basic implementation in Chromium: net/third_party/quic/quartc

Supports BBR

Crypto is stubbed out

QuartcPacketTransport will only be implemented by ORTC (ref)

GitHub issues

Issue #84: [QUIC] Investigate and propose use of DataChannel framing on top of QUIC

Issue #83: [DataChannel] Investigate use of DataChannel without all of WebRTC

Issue #73: [DataChannel] Define bootstrap mechanism for RTCDataChannel

Issue #82: [QUIC] Find out timeline for TLS 1.3

Proposals

Proposal: QUIC DataChannel as the V1 transport.

Specify two modes: DataChannel over UDP or ICE with host candidates.

Integrate ICE + STUN / TURN for network traversal in V2.

Work Items (WebRTC/ORTC)

Work with WebRTC on:

- Use of ICE in a LAN-only scenario
- Possible implementation of Open Screen Protocol with ORTC
- Demuxing with other protocols (RTP, RTCP, DTLS, ICE)
- Implementation status

Work Items (QUIC)

Work with QUIC implementers on:

- Connection multiplexing
- Message ordering with stream IDs
- Server parameters
- Use of 0-RTT connections and BBR on LANs
- Pluggable authentication (J-PAKE?)

Authentication

Authentication Topics

- Requirements & threats
- J-PAKE authentication (no prior key exchange)
- Public-key based authentication
- Open questions and next steps

Requirements & Threats

- Protect integrity of the user's display selection
- Ensure presentation connections are between appropriate parties
- Ensure confidentiality and integrity of presentation URLs, ids, & messages

Threats

- Passive network observer (on-LAN, off-LAN, WAN)
- Active network attacker (injection, replay, spoofing)
- Side channels (timing attacks, telescopes?)

Additional threats to consider

- Malicious or insecure content
  - Cross-origin presentation connections
  - Phishing via presentations
- Mis-configured routers/ISPs
- Compromised displays/user agents
- Device change of ownership or theft

**Recommend a white paper analyzing all threats in more detail and proposing mitigations.
Also document what specific data on the wire should be protected.**

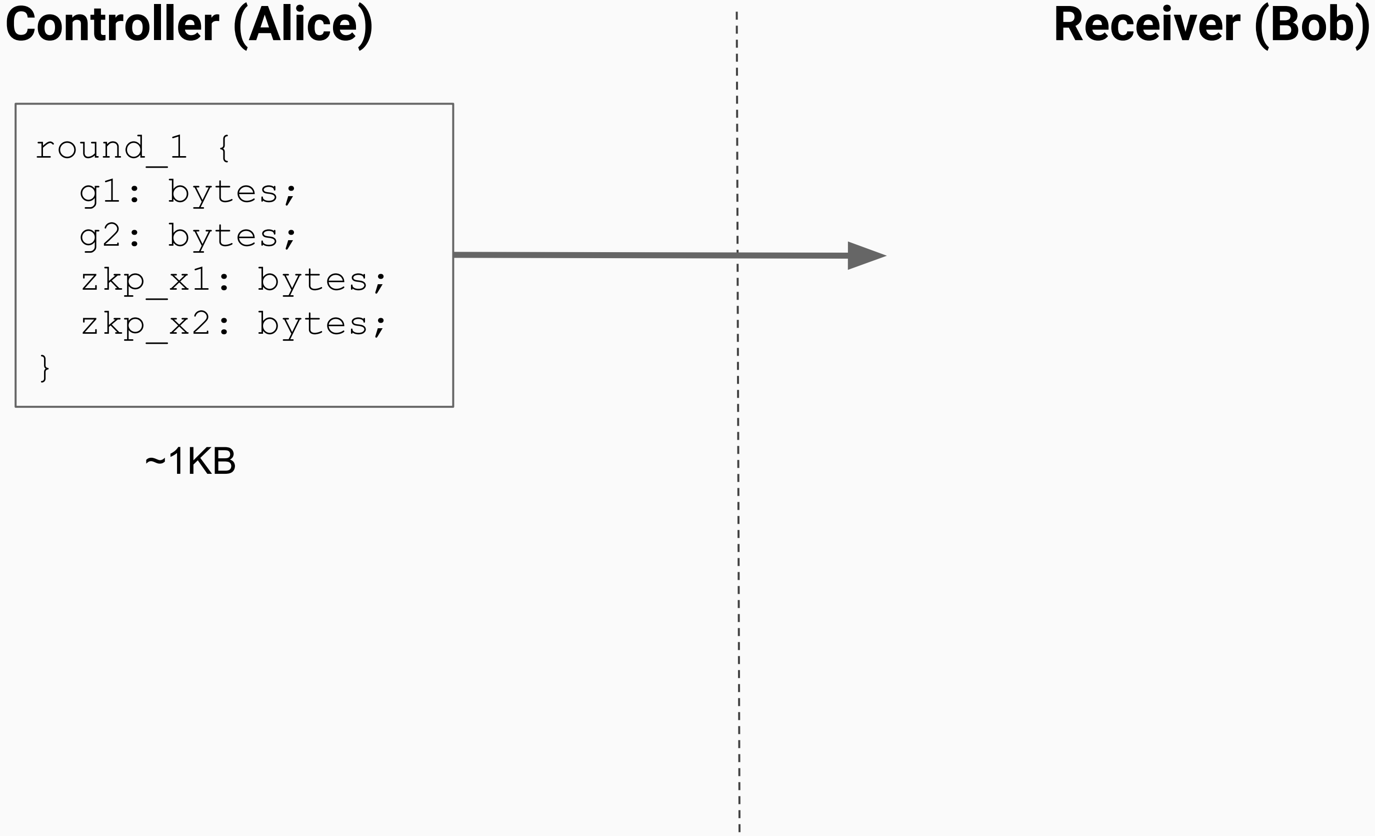
J-PAKE key exchange

Requires a shared password (no prior public key exchange required).

https://github.com/webscreens/openscreenprotocol/blob/gh-pages/j-pake.md

https://www.lightbluetouchpaper.org/2008/05/29/j-pake/

J-PAKE: Round 1



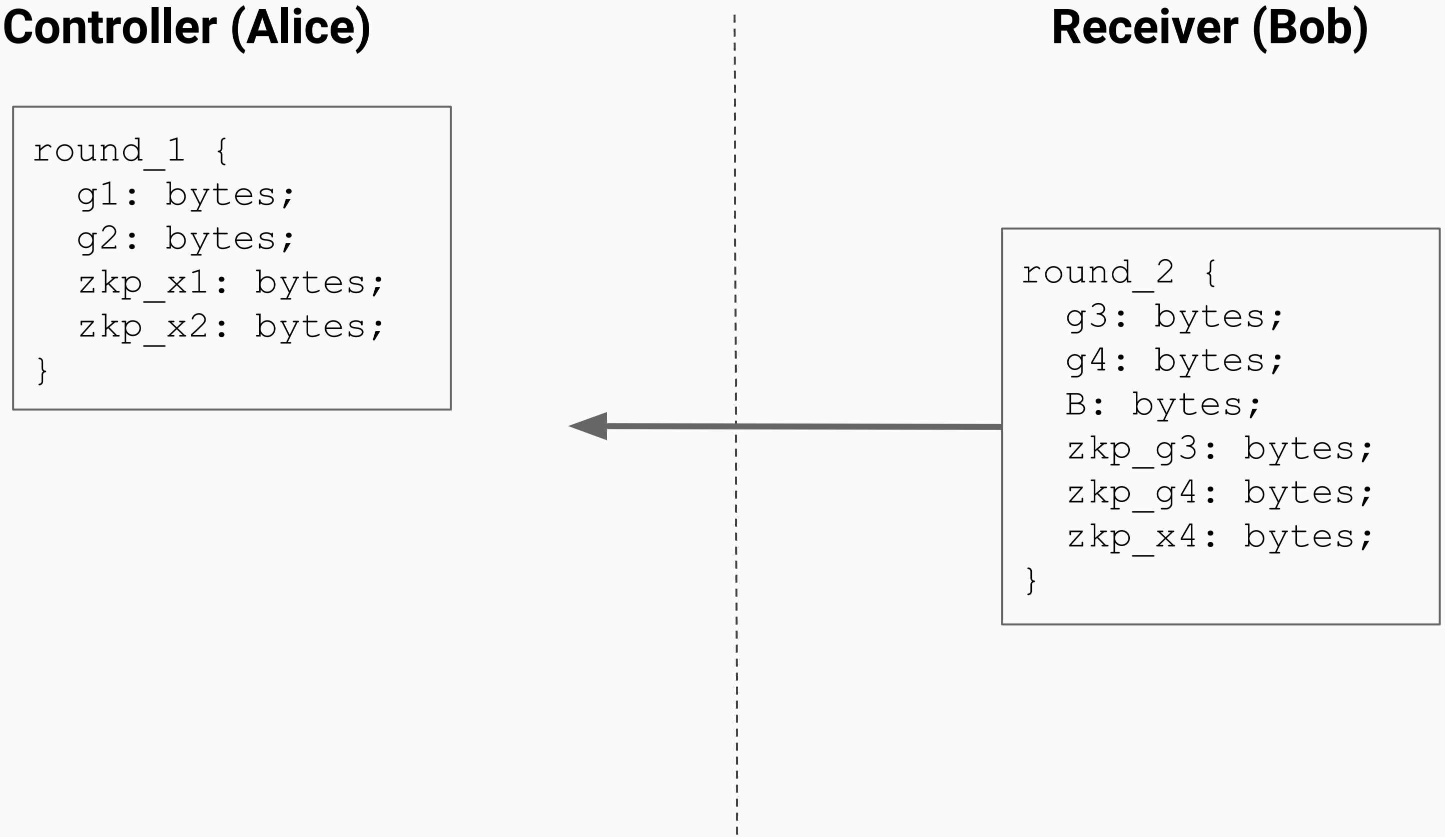
Controller (Alice)

Receiver (Bob)

```
round_1 {
  g1: bytes;
  g2: bytes;
  zkp_x1: bytes;
  zkp_x2: bytes;
}
```

~1KB

J-PAKE: Round 2



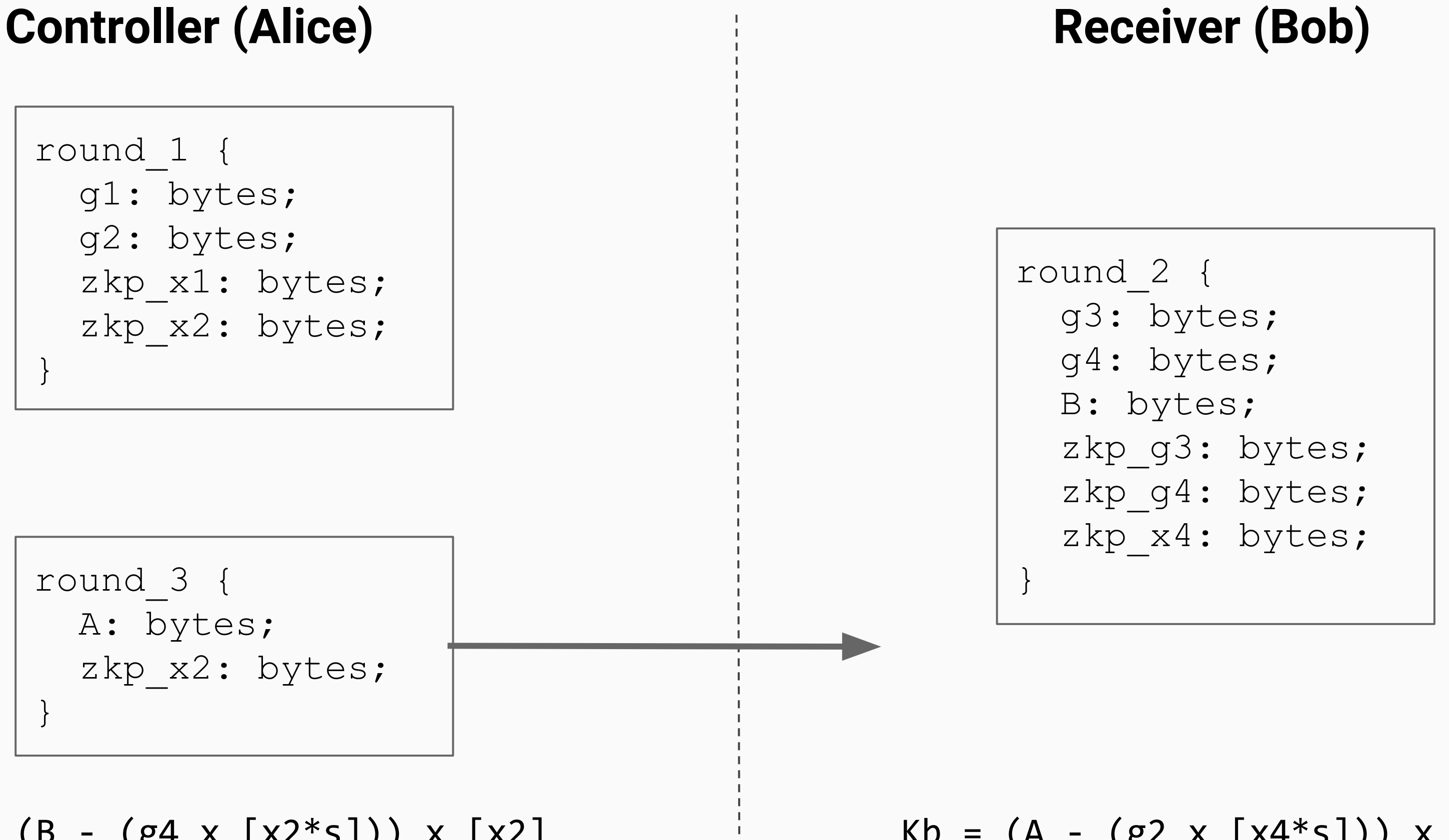
Controller (Alice)

Receiver (Bob)

```
round_1 {
  g1: bytes;
  g2: bytes;
  zkp_x1: bytes;
  zkp_x2: bytes;
}
```

```
round_2 {
  g3: bytes;
  g4: bytes;
  B: bytes;
  zkp_g3: bytes;
  zkp_g4: bytes;
  zkp_x4: bytes;
}
```

J-PAKE: Round 3



Controller (Alice)

Receiver (Bob)

```
round_1 {
  g1: bytes;
  g2: bytes;
  zkp_x1: bytes;
  zkp_x2: bytes;
}
```

```
round_2 {
  g3: bytes;
  g4: bytes;
  B: bytes;
  zkp_g3: bytes;
  zkp_g4: bytes;
  zkp_x4: bytes;
}
```

```
round_3 {
  A: bytes;
  zkp_x2: bytes;
}
```

```
Ka = (B - (g4 x [x2*s])) x [x2]
```

```
Kb = (A - (g2 x [x4*s])) x [x4]
```

J-PAKE next steps

Propose passcode requirements, possible UI, and key derivation function.

Define J-PAKE key exchange messages as part of control protocol.

Determine whether J-PAKE can be used for recurring authentication.

J-PAKE initial connection

1. QUIC connection with self-signed keys.
2. J-PAKE to derive shared secret.
3. J-PAKE secret verification.
4. Extract keying info from QUIC connection and verify with shared secret.

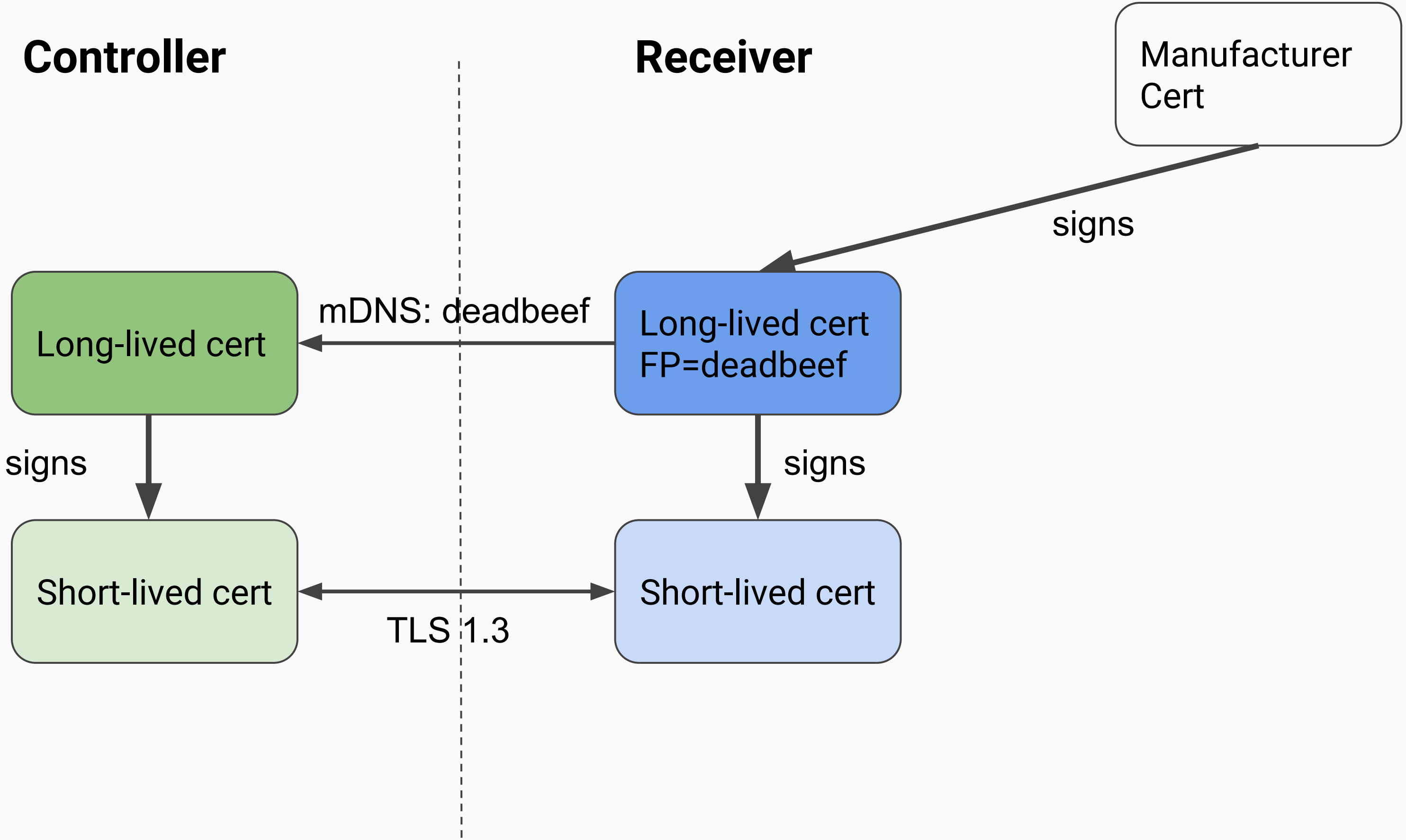
J-PAKE key exchange

1. Complete prior steps to create a J-PAKE authenticated connection.
2. Server (presentation display) generates a long lived signing certificate.
   a. For TLS 1.3 compatibility it uses this same cert for all connections.
3. Server sends public key to client (controlling UA).
   a. It may have signatures attached, e.g. from display manufacturer.
4. Client generates a long lived signing certificate.
   a. Tied to the public key fingerprint for the server cert.
5. Client sends public key of its cert to server.

PKI based authentication

1. Server advertises its signing certificate fingerprint via discovery.
2. Server and client create a short lived (~48H) certificate for TLS handshake.
3. TLS certs are signed by long lived certificates exchanged earlier.
4. Client verifies server cert was signed by server signing cert.
5. Server verifies client cert was signed by client signing cert.

After certificate exchange



Cert structure, lifetime, scope

- Selection of cipher suites and signature algorithms
  - Hardware crypto capabilities may come into play
- Identities associated with certificates
  - Display: serial number/code + friendly name + display model
- Per-PresentationConnection certificates?
- Certificate lifetime
  - Want to ensure they are reset on factory reset or user data deletion
- Client certificates and privacy
  - Want separate certificate store for private browsing
- Certificate revocation, requirements changes, transparency logs, etc.

PKI based authentication: Next steps

Full proposal on key exchange

Full proposal on certificate structure & scope

Comparative research from other IoT efforts (Nest, WoT, etc.)

Develop representative user interface for both J-PAKE and PKI based auth

Control Protocol & Serialization

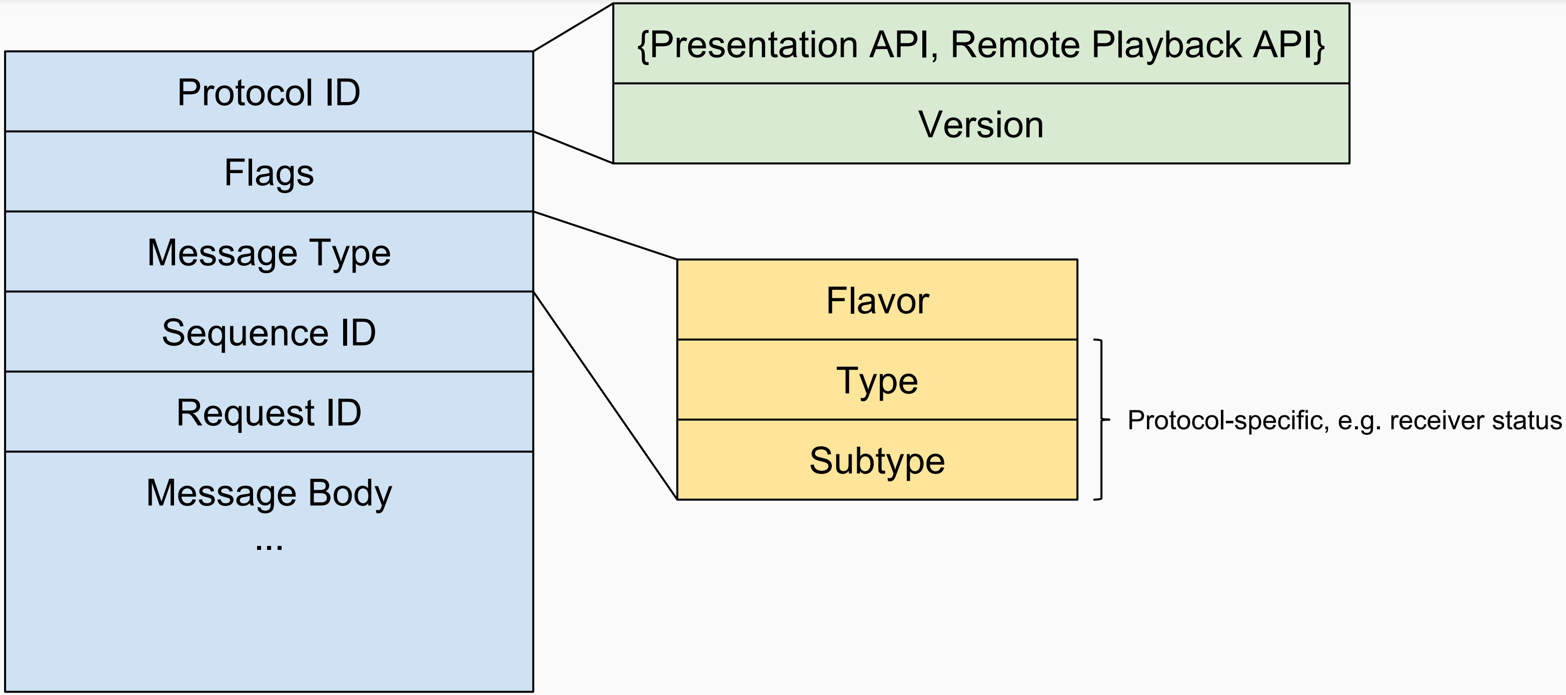
Control Protocol Topics

- Overview of control protocol
- Current custom binary serialization
- CBOR alternative
- Protocol Buffers alternative
- Discussion & recommendation for serialization
- Extensions & roles
- GitHub issues

Protocol Overview

- Broad Message Types (Flavor): Command, Request, Response, Event
- Presentation API Message Types
  - Presentation Display Availability
  - Presentation Lifecycle
  - Presentation Connection Management
  - Presentation Application Messages
  - Receiver Status

Protocol Header



Protocol ID

Flags

Message Type

Sequence ID

Request ID

Message Body
...

{Presentation API, Remote Playback API}

Version

Flavor

Type

Subtype

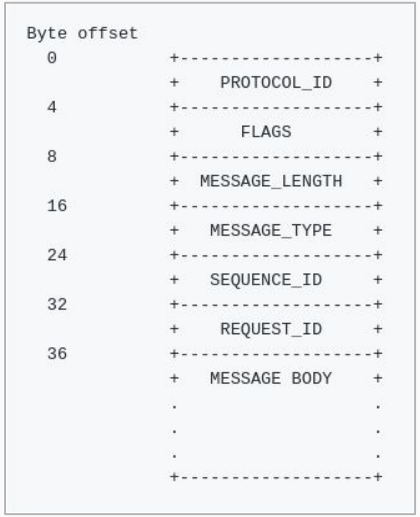
Protocol-specific, e.g. receiver status

Versioning

- Versioning is done using major and minor version numbers (X.Y)
- Two implementations can talk if they support the same major version
- Minor versions may extend another minor version but remain backwards compatible
- Discovery and connection process should negotiate version
    - TODO: Add this to the working specs for these processes.

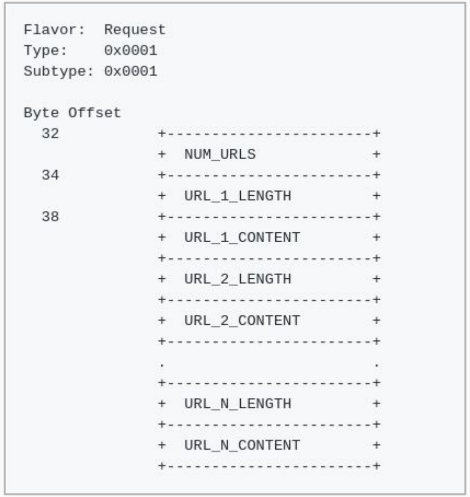
Custom Binary Format

Message Header



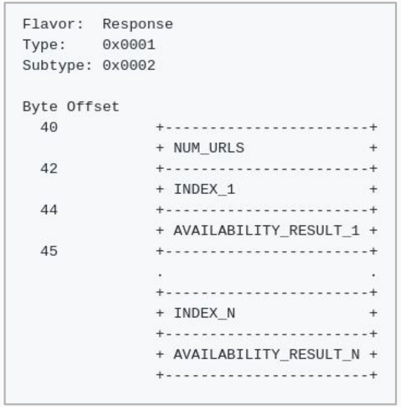
```
Byte offset
  0            +-------------------+
               +    PROTOCOL_ID    +
  4            +-------------------+
               +      FLAGS        +
  8            +-------------------+
               +  MESSAGE_LENGTH   +
  16           +-------------------+
               +   MESSAGE_TYPE    +
  24           +-------------------+
               +   SEQUENCE_ID     +
  32           +-------------------+
               +    REQUEST_ID     +
  36           +-------------------+
               +   MESSAGE BODY    +
               .                   .
               .                   .
               .                   .
               +-------------------+
```

Availability Request



```
Flavor:   Request
Type:     0x0001
Subtype:  0x0001

Byte Offset
  32              +----------------------+
                  +  NUM_URLS            +
  34              +----------------------+
                  +  URL_1_LENGTH        +
  38              +----------------------+
                  +  URL_1_CONTENT       +
                  +----------------------+
                  +  URL_2_LENGTH        +
                  +----------------------+
                  +  URL_2_CONTENT       +
                  +----------------------+
                  .                      .
                  +----------------------+
                  +  URL_N_LENGTH        +
                  +----------------------+
                  +  URL_N_CONTENT       +
                  +----------------------+
```

Availability Response



```
Flavor:   Response
Type:     0x0001
Subtype:  0x0002

Byte Offset
  40             +---------------------+
                 + NUM_URLS            +
  42             +---------------------+
                 + INDEX_1             +
  44             +---------------------+
                 + AVAILABILITY_RESULT_1 +
  45             +---------------------+
                 .                     .
                 +---------------------+
                 + INDEX_N             +
                 +---------------------+
                 + AVAILABILITY_RESULT_N +
                 +---------------------+
```

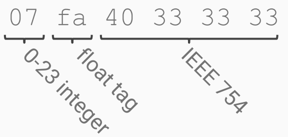
CBOR Alternative

- Concise Binary Object Representation - RFC 7049
- Based on JSON data model
- Design Goals:
    - Allow for very small code size
    - Fairly small message size
    - Extensibility without version negotiation
- Open source implementations available in many languages
    - C, C++, C#, Java, Python, Ruby, Go, JavaScript, etc.
    - Still requires type-specific encode/decode to be done

CBOR Samples

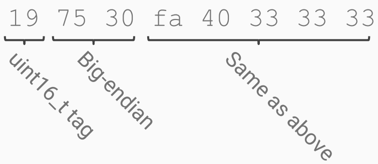
```
struct {
  int x;
  float y;
}
```

```
{ 7, 2.8f }
```



```
07 fa 40 33 33 33
```

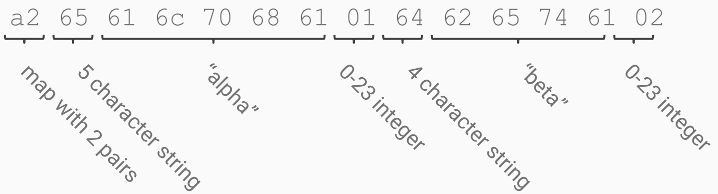
```
{ 30000, 2.8f }
```



```
19 75 30 fa 40 33 33 33
```

```
struct {
  int alpha;
  int beta;
}
```

```
{ 1, 2 }  ⇨  { "alpha": 1, "beta": 2}
```



```
a2 65 61 6c 70 68 61 01 64 62 65 74 61 02
```

23-byte strings can be encoded with single-byte tag

CBOR Optional Fields

```
A = (
  x: int,
  ? y: float,
  ? z: float,
)
```

Value-or-null:

`{int}{null|float}{null|float}`

```
B = (
  x: int,
  ? y: int,
)
```

Omission:

`{int}{int}` or `{int}`

Also works for JSON map-style encoding

Protocol Buffer Alternative

- Google’s serialization scheme (similar to XML and JSON)
- Open source implementation also available in many languages
  - proto2: Java, Python, Objective-C, C++
  - proto3: adds Go, Ruby, C#
- Uses code generation for encode/decode to typed messages

```
message ProtocolId {
  enum Type {
    PRESENTATION_API = 0;
    REMOTE_PLAYBACK_API = 1;
  }
  required Type type = 1;
  required int32 version_major = 2;
  required int32 version_minor = 3;
}
```

```
message MessageType {
  enum Flavor {
    COMMAND = 0;
    REQUEST = 1;
    RESPONSE = 2;
    EVENT = 3;
  }
  required Flavor flavor = 1;
  required int32 type = 2;
  optional int32 subtype = 3;
}
```

```
message Header {
  required ProtocolId protocol = 1;
  optional int32 flags = 2;
  required int32 message_length = 3;
  required MessageType message_type = 4;
  required int64 sequence_id = 5;
  optional int64 request_id = 6;
}
```

Benchmark Data

- Benchmarks were run with prototype PresentationAvailability{Request,Response} messages
- CBOR used untagged serialization variant shown before
- -O2 everywhere

Benchmark Data

Code size

	Test+Generated	Library
CBOR	30KB	42KB
Protobufs	94KB	3.8MB lite (30MB full)

Message Size

	Request	Response
CBOR	131 B	137 B
Protobufs	258 B	282 B

Benchmark w/ 10000 messages

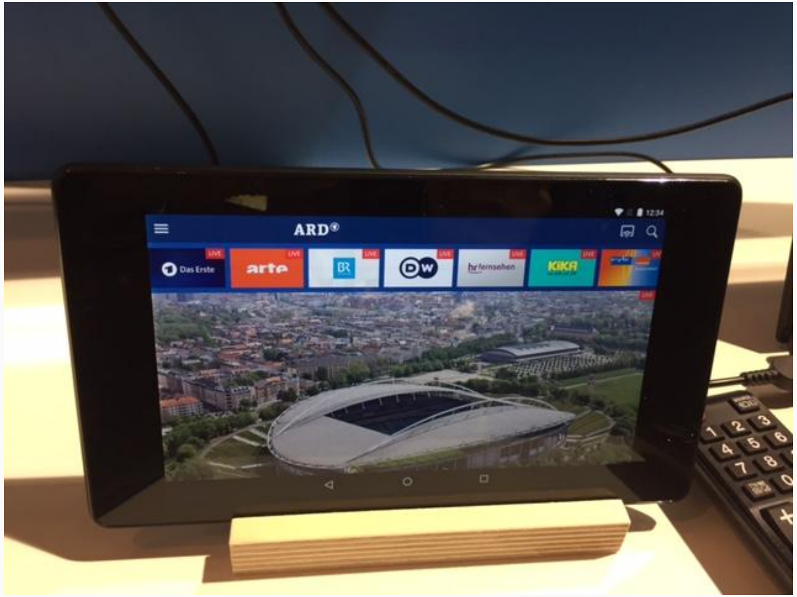
	Read	Write
CBOR	14 ms	9 ms
Protobufs	12 ms	17 ms

Discussion & Recommendation

- Performance is very similar
- CBOR is more size-efficient but possibly more error-prone
- Both are open source and available in many languages
- CBOR has decent tooling support (debugging, CDDL, validation)
- CBOR was adopted by the Web Packaging standard
- CBOR would have more efficient JavaScript support

Our recommendation at this time is to use CBOR for serialization

Capabilities & Roles



Open Screen Protocol Library

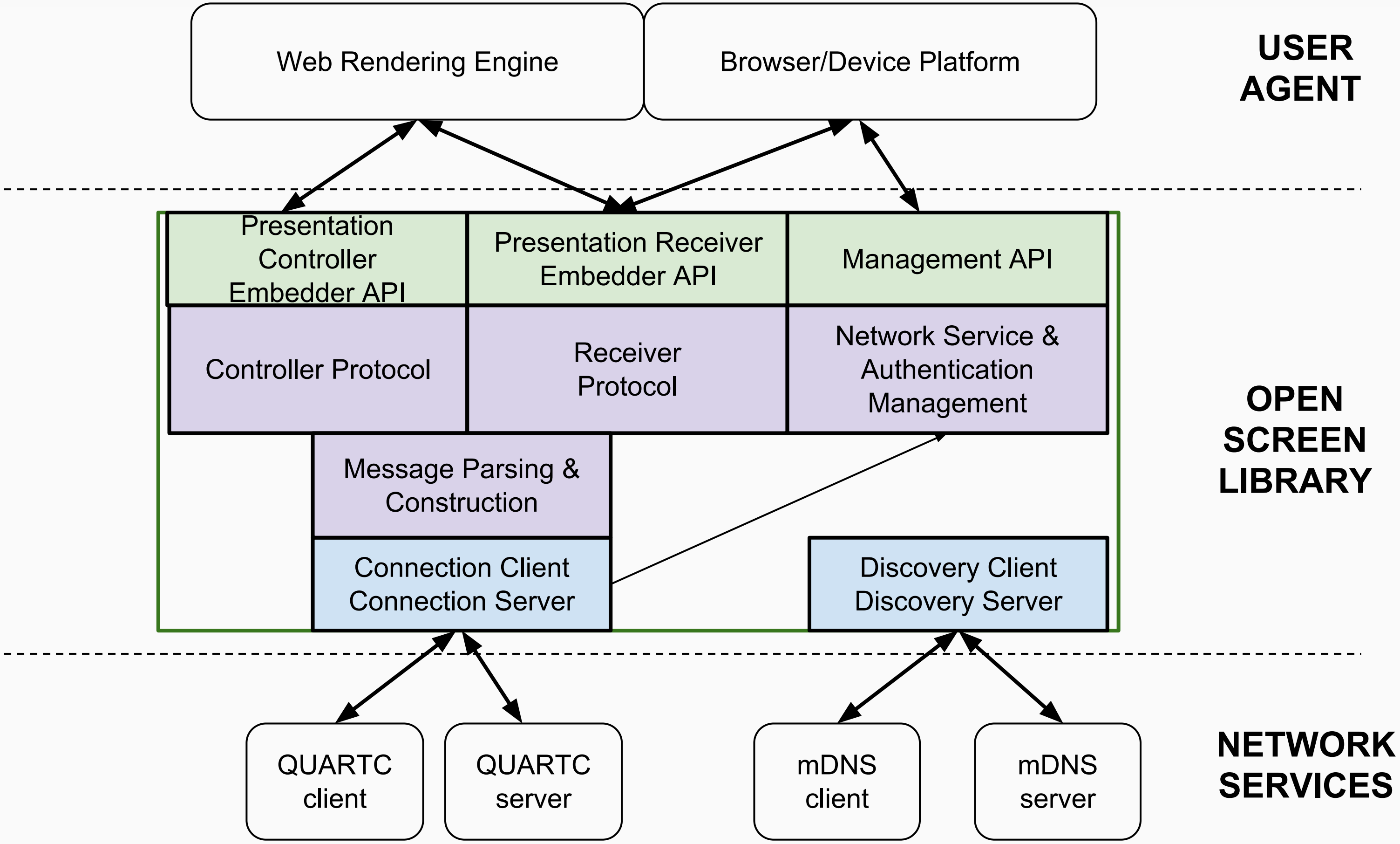
Open Screen Protocol Library Outline

- Goals and rationale
- Library architecture
- Embedder API (sample)
- Dependencies, toolchains and style
- Repository & access
- Work plan

Goals and Rationale

- Create a complete library solution to translate network protocol to Presentation API and Remote Playback API
- Platform-independent, Chromium-independent
- Allow replacement of network services (e.g. mDNS) by users

Architecture


Web Rendering Engine
Browser/Device Platform
USER AGENT
Presentation Controller Embedder API
Presentation Receiver Embedder API
Management API
Controller Protocol
Receiver Protocol
Network Service & Authentication Management
Message Parsing & Construction
Connection Client Connection Server
Discovery Client Discovery Server
OPEN SCREEN LIBRARY
QUARTC client
QUARTC server
mDNS client
mDNS server
NETWORK SERVICES

Embedder API (sample)

```
class PresentationController {
  public:
    // Requests screens compatible with |url| and registers |observer| for
    // availability changes.  The screens will be a subset of the screen list
    // maintained by the ScreenListener.  Returns a positive integer id that
    // tracks the registration. If |url| is already being watched for screens,
    // then the id of the previous registration is returned and |observer|
    // replaces the previous registration.
    uint64_t RegisterScreenWatch(const std::string& url,
                                 PresentationScreenObserver* observer);

    // Requests that a new presentation be created on |screen_id| using
    // |presentation_url, with the result passed to |delegate|.
    // |connection_delegate| is passed to the resulting connection.
    void StartPresentation(const std::string& url,
                           const std::string& screen_id,
                           PresentationRequestDelegate* delegate,
                           PresentationConnectionDelegate* conn_delegate);

    // ...
}
```

Embedder API (sample)

```
// An object to receive callbacks related to a single PresentationConnection.
class PresentationConnectionDelegate {
 public:
  // State changes.
  virtual void OnConnected() = 0;
  virtual void OnClosed() = 0;
  virtual void OnDiscarded() = 0;
  virtual void OnError(const std::string& message) = 0;
  virtual void OnTerminated(PresentationTerminationSource source) = 0;

  // A string message was received.
  virtual void OnStringMessage(const std::string& message) = 0;

  // ...
}
```

Embedder API (sample)

Controller

controller.js:

```
presentationRequest.start()
    .then(connection => {
        connection.send(“hello”);
    });
```

Embedder (e.g. Chromium):

```
connection->SendString(“hello”);
```

Receiver

Embedder (e.g. Chromium):

```
void PCDelegate::OnStringMessage(...) {
  // Forward |message| up to web engine.
}
```

hello

receiver.js:

```
connection.onmessage = e => {
  console.log(e.data);
};
```

Platform API

- Porting layer of the library for platform primitives
- Similar to Chromium base/ and WebRTC rtc_base/
- Sockets, threading, logging, network state, system power states, etc.
- Not yet designed

Get The Source

- Get the source
  - `git clone https://chromium.googlesource.com/openscreen`
  - `git submodule update --init --recursive`
- Built with `gn`
  - Contained in a Chromium checkout
  - Also available from storage.googleapis.com (see README.md)
- Gerrit for code review (also see README.md)

Open Screen Library Timeline

Jan 2018: Kickoff

Feb 2018: Hello World

June 2018: Embedder APIs

August 2018: Platform APIs, Control protocol

Oct 2018: Authentication

2H 2018: Benchmarking, E2E testing

2019: V2 features

Intentionally Blank :-)

Protocol extensions

Why we need them. Sample use cases