What is WebRTC?

Web Real-Time Communication - WebRTC

- A collection of well-established protocols for establishing and maintaining real-time, peer-to-peer communication.
  - Protocols for establishing connection between peers
    - Interactive Connectivity Establishment (**ICE**)
    - Session Traversal Utilities for NAT (**STUN**)
    - Traversal Using Relays around NAT (**TURN**)
  - Protocol for describing media to transmit
    - Session Description Protocol (**SDP**)
  - Protocol for establishing a secure connection
    - Datagram Transport Layer Security (**DTLS**)
  - Transport Protocols
    - Stream Control Transport Protocol (**SCTP**)
    - Secure Real-Time Transport Protocol (**SRTP**)
- Primarily designed for Video Conferences, but has now extended to many different applications.
- WebRTC requires, but **does not** specify, a signaling channel.

WebRTC Signaling

Intentionally not part of the spec to allow flexibility.

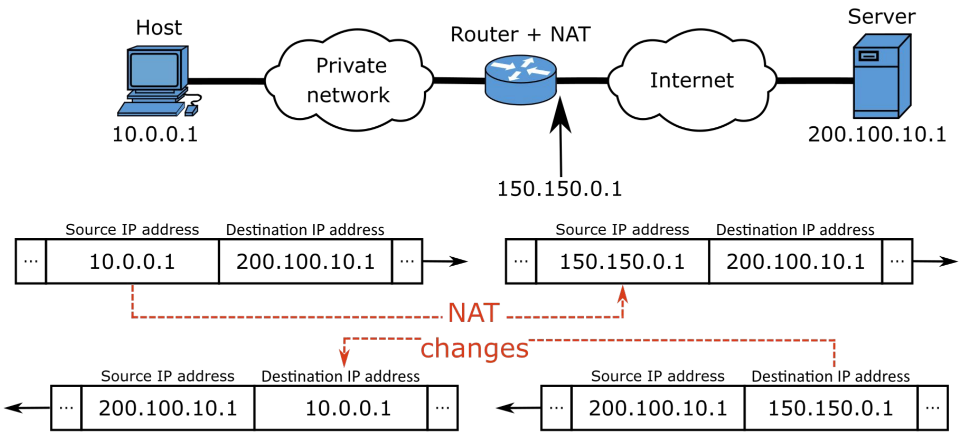
- Handles sending and receiving core messages for establishing the WebRTC media connection.
- Does not transmit or receive any media.
- In Orchid, we use WebSockets for this task.

WebRTC High-level Process

- *Peer #1* creates an ***offer SDP*** and sends it to *Peer #2* using the signaling channel.
- *Peer #2* receives the ***offer SDP***, generates an ***answer SDP,*** and sends it to *Peer #1* using the signaling channel.
- Both peers generate information about their network configuration using the **ICE** protocol and send this information over the signaling channel.
- This connection information is used to generate UDP connections that are tested to find paths that work.
- Once a valid connection is found, the communication channel is secured and media flows via RTP protocol.

Establishing Connection - NAT Refresher

- Translates your internal/local IP:Port to an external Internet-routable IP address.
- Typical pain-point for VoIP applications.



Interactive Connectivity Establishment (ICE)

A protocol for discovering a way for two peers to communicate. Addresses common NAT-related connection issues.

Overview of Technique

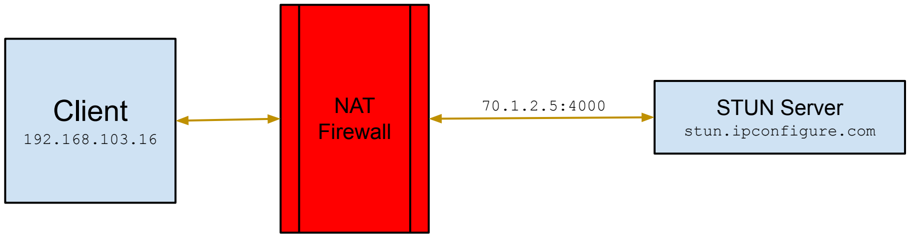
1. Gather - Each peer generates Connection Candidates
   - Host candidates (eth0, eth1, tun0)
   - Server reflexive candidates (Candidates received from STUN server).
   - Relay candidate (Candidates received from TURN server).
   - Peer reflexive candidates (Candidates generated during the connection test process).
2. Exchange Connection Candidates
3. Generate connection pairs and sort by priority
4. Connectivity Checks
   - Iterate through the combinations of peer candidates to determine which ones can actually transmit media.
5. Conclude ICE

Session Traversal Utilities for NAT (STUN)

A tool for hosts to discover the presence of a NAT and discover their external connection information.

Basic Concept:

- A host communicates with a well-known server (STUN Server) *outside* the host's NAT.
- The STUN Server reports back with the host's external connection info.
- This forms a *server reflexive* WebRTC Candidate which can be given to a peer.
- Media information does **not** flow through the STUN server.



Traversal Using Relays around NAT (TURN)

A tool for traversing media through difficult NATs using relays.

Basic Concept:

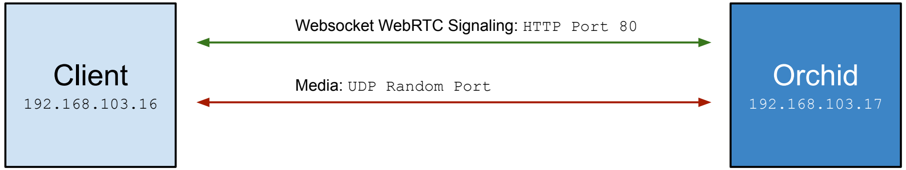
- A host communicates with a well-known server (TURN Server) *outside* the host’s NAT.
- The TURN Server provides it’s connection info to give to a peer and proxies the media.
- This forms a *relay* WebRTC Candidate which can be given to a peer.
- Media information flows through the TURN server.
- Can be resource heavy.

WebRTC Network Scenarios

WebRTC Network Scenarios

Direct Connection between client and Orchid Server

- Client communicates over an HTTP websocket to negotiate media channel.
- Media directly flows on local network.

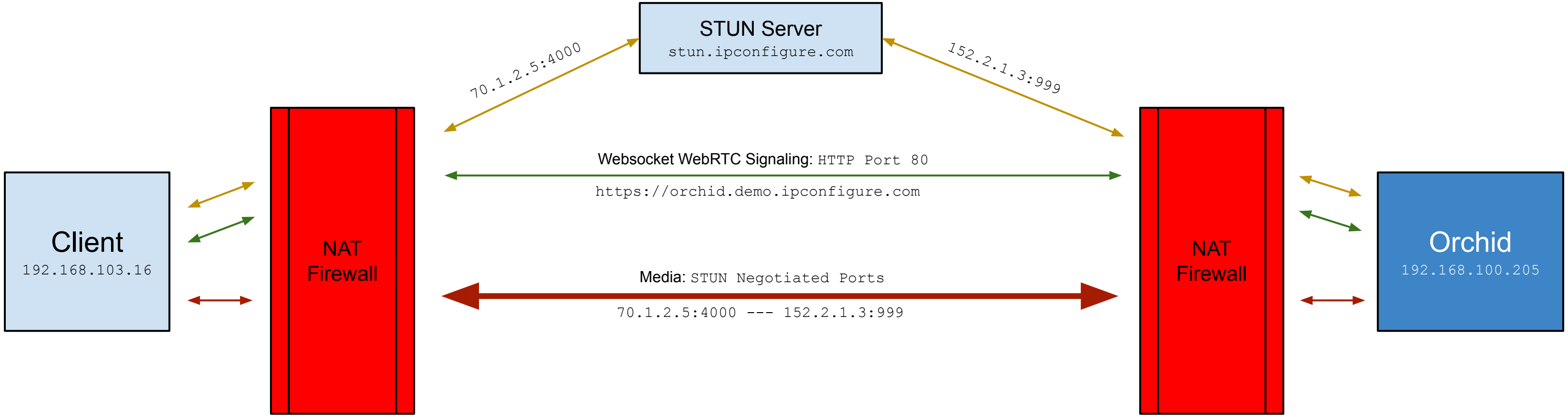


Example: https://orchid.ipconfigure.com

WebRTC Network Scenarios

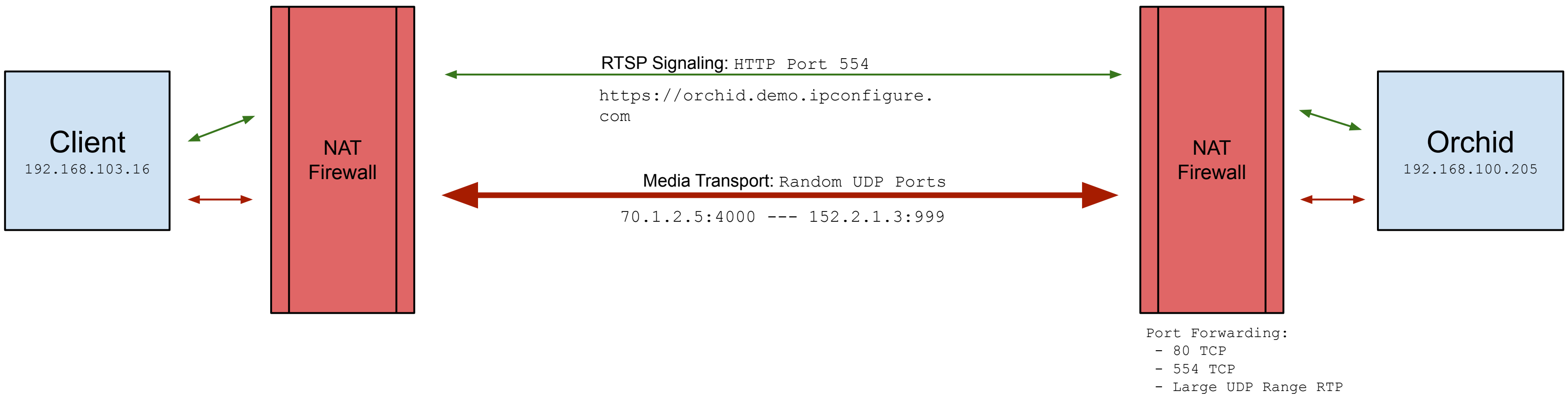
Remote Connection - Both Orchid Server and Client behind NAT

- Signaling still happens with HTTP (port 80 is forwarded).
- STUN server is used to discover peer external IP addresses and ports.
- Media directly sent between Orchid and Client.



Example: https://orchid123.ddns.net:8443

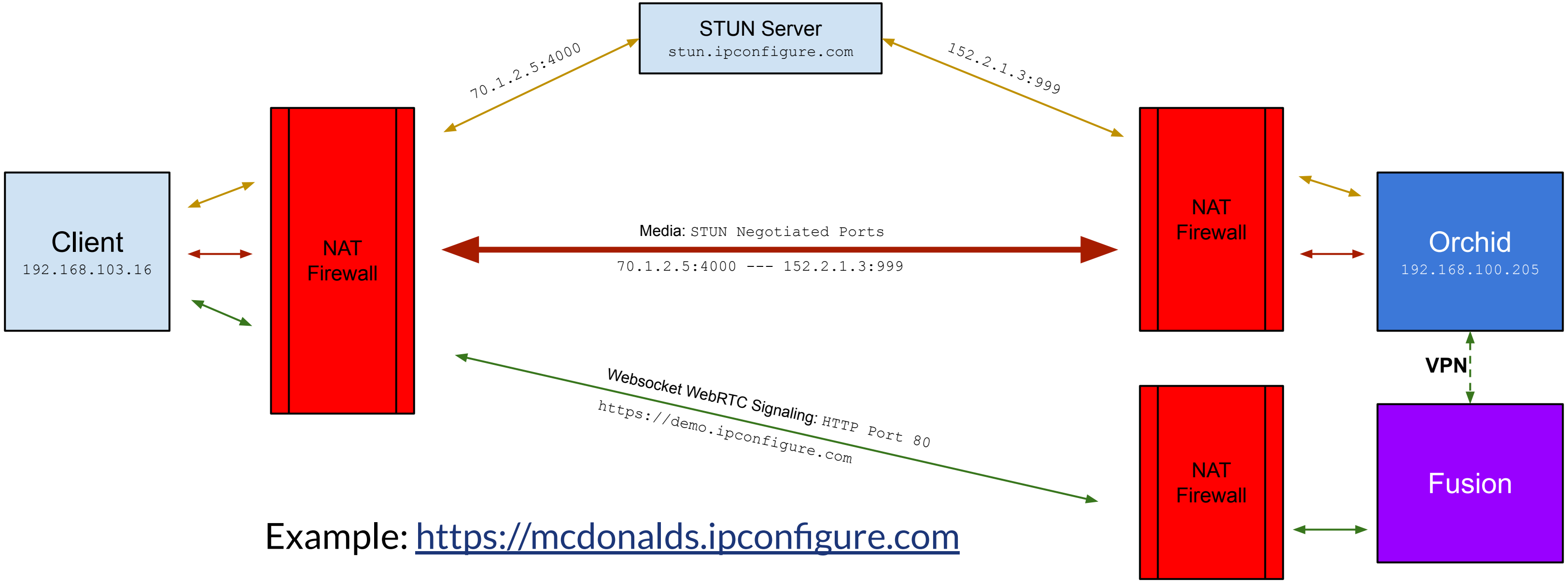
RTSP - Remote Orchid



WebRTC Network Scenarios

Remote Connection with Fusion

- Signaling now happens through Fusion Proxy (Fusion forwards port 80).
- Media still direct connects between Orchid and Client.
- No rule changes to the Orchid NAT are required.

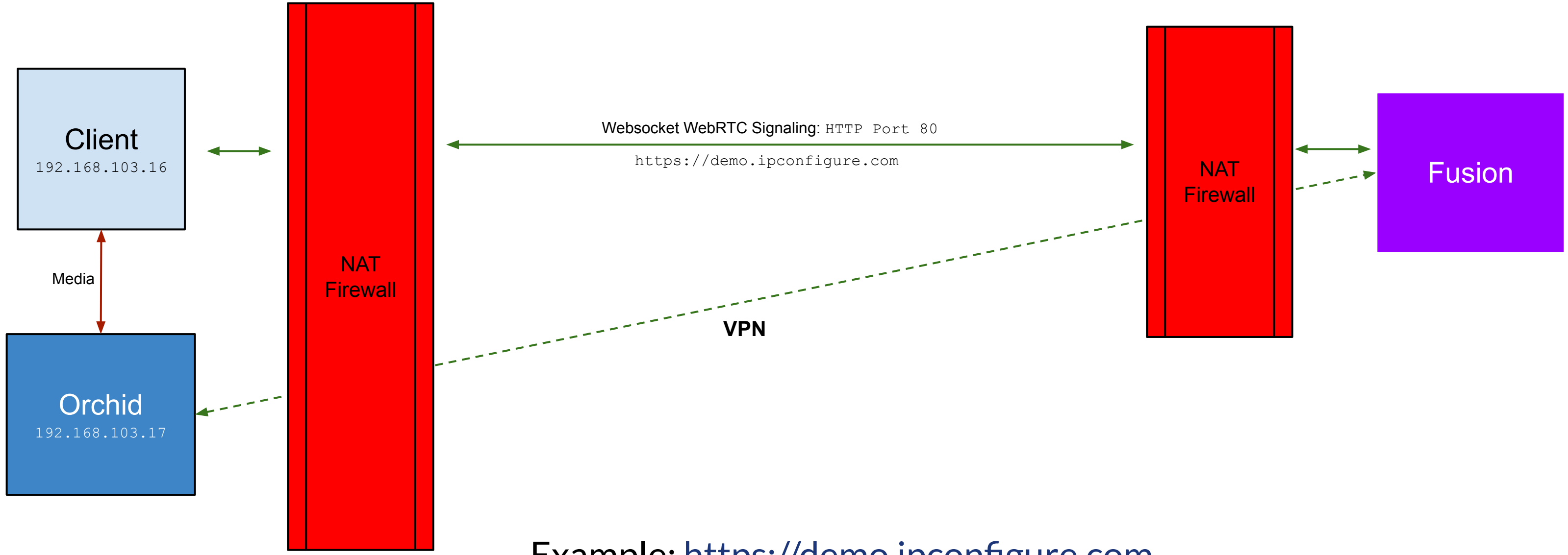


Example: https://mcdonalds.ipconfigure.com

WebRTC Network Scenarios

Semi-Local Connection with Fusion

- Signaling through Fusion Proxy.
- Media transports all on local network.
- Doesn't need STUN server in this case.



Example: https://demo.ipconfigure.com

Implementation

Backend

- Implemented using Gstreamer elements
    - libnice - `nicesrc`, `nicesink` -- Handles ICE negotiations
    - `dtlssrtpdec`, `dtlssrtpenc` - Handles DTLS interactions and SRTP transport.
    - `rtpbin` - Handles RTP payload/depayload
- Created before Gstreamer released their official `webrtcbin` element.
- The difficult work is performed within the Gstreamer elements. Orchid code glues all of the components together.
- Playback and Live streams use most of the same code paths as our RTSP server.

Frontend

- Provided by browsers natively in the Web API - RTCPeerConnection