**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA CÔNG NGHỆ PHẦN MỀM**

---

**BÁO CÁO TUẦN 6**

**NGUYỄN VIẾT DANH – 12520053**

**ĐỖ HOÀNG PHƯƠNG – 12520324**

**TÌM HIỂU CÔNG NGHỆ WebRTC**

**Lớp: KTPM2012**

**Giảng viên hướng dẫn: Ths. PHẠM THI VƯƠNG**

# KIẾN TRÚC WebRTC

WebRTC yêu cầu các nhà phát triển ứng dụng phải có khả năng phát triển các ứng dụng dạng rich application, truyền thông đa phương tiện thời gian thực trên nền web mà không cài đặt các plugin liên quan. Mục đích của nó nhằm hỗ trợ xây dựng một nền tảng RTC mạnh mẽ có thể làm việc với nhiều trình duyệt khác nhau, nhiều nền tảng khác nhau.



Có 2 lớp riêng biệt:

###### Các nhà phát triển trình duyệt sẽ quan tâm đến C++ API của WebRTC và các thành phần xử lý ngoài.

###### Các nhà phát triển ứng dụng Web quan tâm đến Web API.

Những sản phẩm liên quan đến chat video hay audio do các nhà phát triển của bên thứ ba phát triển sẽ được cung cấp các web API để giao tiếp thời gian thực.

**Web API** là một API được sử dụng bởi các nhà phát triển của bên thứ ba để phát triển các ứng dụng chat.

**WebRTC Native C++ API** là một lớp API được cung cấp cho các nhà phát triển trình duyệt để họ có thể dễ dàng kế thừa **Web API**.

**Transport/Session** là các thành phần phiên được xây dựng bằng cách tái sử dụng libjingle mà không sử dụng hay yêu cầu giao thức xmpp/jingle.

**RTP Stack** là một chồng mạng cho RTP, giao thức thời gian thực.

**STUN/ICE** là một thành phần cho phép gọi sử dụng kỹ thuật STUN, ICE để thiết lập kết nối qua nhiều kiểu mạng.

**Quản lý phiên (Session Management)** là một lớp phiên trừu tượng, cho phép thiết lập cuộc gọi và quản lý lớp. Giao thức thực hiện của ứng dụng được quyết đinh bởi nhà phát triển.

**VoiceEngine** là một bộ khung sử dụng cho chuỗi truyền thông âm thanh, từ card âm thanh đến mạng.

**iSAC/iLBC/Opus:**

###### **iSAC** là một bộ biên/giải mã âm thanh băng thông rộng và siêu rộng cho VoIP và âm thanh trực tuyến. iSAC sử dụng mẫu tần số 16 kHz hoặc 32 kHz với khả năng thích nghi và tỉ lệ biến thiên từ 12 đến 52 kbps.

###### **iLBC** là là một bộ biên/giải mã âm thanh băng thông hẹp cho VoIP và âm thanh trực tuyến. Sử dụng tần số 8 kHz với bitrate 15.2 kbps cho 20ms frame và 13.33 kbps cho 30ms frame.

###### Opus hỗ trợ mã hóa bitrate liên tục và biến thiên từ 6 kbit/s đến 510 kbit/s, kích thước khung hình từ 2,5 ms đến 60 ms và tỷ lệ biến thiên mẫu từ 8kHz (với băng thông 4kHz) đến 48 kHz (với 20 băng thông kHz ).

###### Một bộ jitter buffer động và thuật toán giấu lỗi để giấu đi các hiệu ứng xấu của mạng jitter và mất gói. Giữ độ trễ càng thấp càng tốt trong khi duy trì chất lượng âm thanh cao nhất.

**Acoustic Echo Canceler (AEC)** là một phần mềm dựa trên thành phần xử lý tín hiệu sẽ loại bỏ những âm vang từ giọng nói.

**Noise Reduction (NR) (Giảm ồn)** là một phần mềm dựa trên thành phần xử lý tín hiệu sẽ loại bỏ các tiếng ồn nền được kết hợp với VoIP (như tiếng huýt gió, rít, tiếng ồn máy quạt,…).

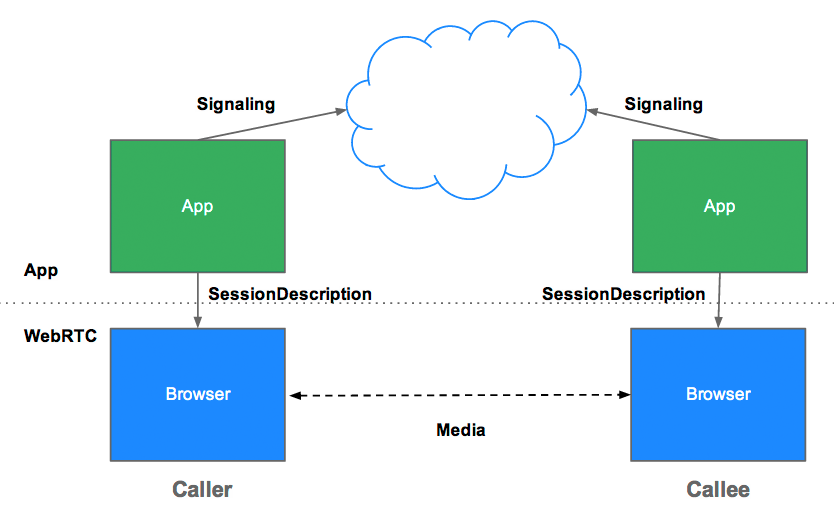
**VideoEngine** là một chuỗi khung video, từ camera đến mạng, từ mạng đến màng hình.

**VP8** là bộ biên – giải mã từ dự án án WebM.

**Video Jitter Buffer (bộ đệm Jitter)** được sử dụng cho video, giúp che giấu những ảnh hưởng của jitter và mất gói.

**Image Enhancements (xử lý ảnh tăng cường)** giúp giảm tiếng ồn video từ việc chụp ảnh bởi webcam.

# PHƯƠNG THỨC HOẠT ĐỘNG

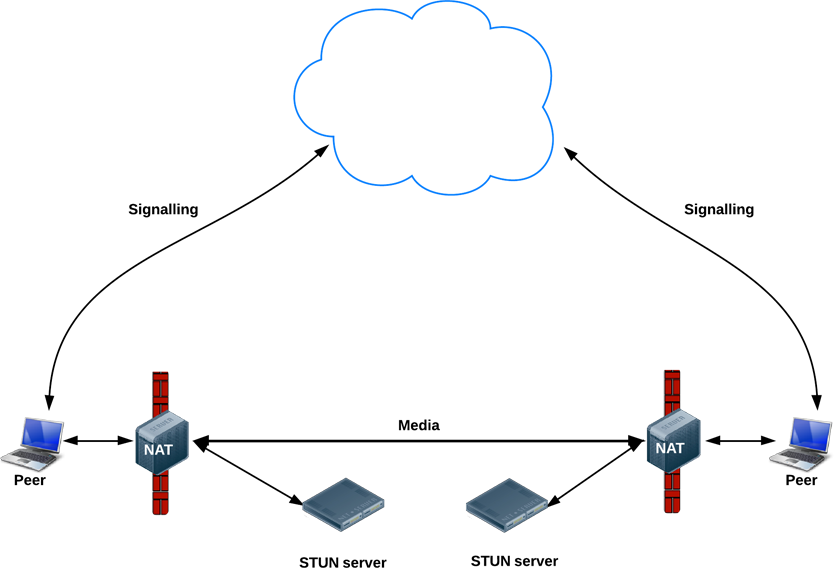


Ví dụ đơn giản về Video chat sử dụng WebRTC:

1. Caller và Callee khi bắt đầu sử dụng ứng dụng phải gửi các tín hiệu về ứng dụng mà mình đang sử dụng tới server (session description)
2. Caller gọi: Caller request tới server để đăng ký một cuộc gọi với Callee, server sẽ trả về cho Caller địa chỉ IP của Calle cùng với port ứng dụng hiện tại của Callee
3. Callee sẽ nhận một bản tin của server báo về một cuộc gọi từ phái caller
4. Caller bắt đầu truyền dữ liệu từ phía mình trực tiếp sang Callee
5. Caller/callee ngắt kết nối: thông báo về server và dừng truyền dữ liệu

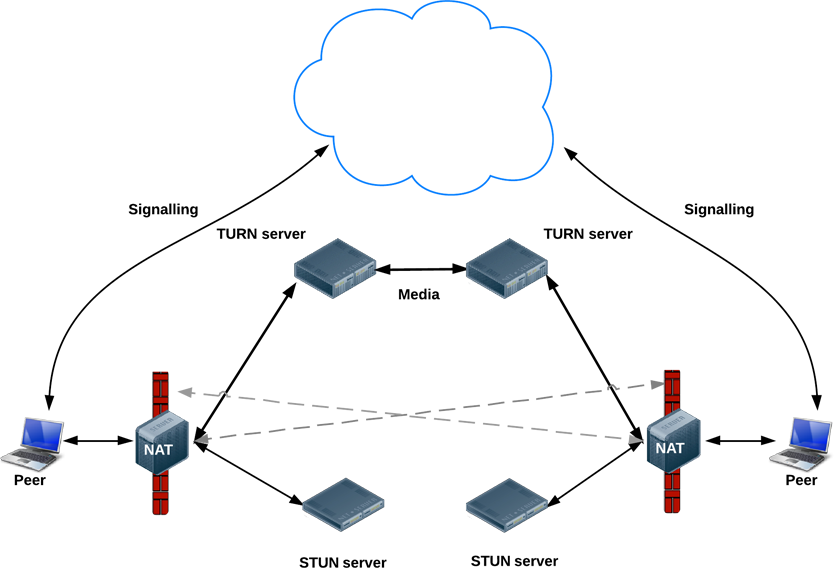
Tuy nhiên, trên thực tế, các peer đều sử dụng các địa chỉ IP private (thuộc các dải: 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8) và sử dụng cơ chế NAT để kết nối internet, do đó việc kết nối trực tiếp giữa các peer sẽ gặp khó khăn nếu sử dụng dải mạng nội bộ. Do đó, chúng ta phải có cơ chế hỗ trợ phân giải sang địa chỉ IP public.

**STUN**



Để tiết kiệm địa chỉ IP, rất nhiều các nhà điều hành mạng sử dụng cơ chế NAT (Network address translation) để chuyển đổi từ địa chỉ IP mạng nội bộ sang địa chỉ IP Public. Về cơ bản địa chỉ IP sẽ được chuyển đổi như sau : IP private <-> IP public: port trong đó port ở đây là các cổng ứng dụng nằm trên Firewall hay router. Điều này giúp cho nhiều địa chỉ IP nội bộ có thể sử dụng chung IP public. STUN server sẽ đóng vai trò trung gian giúp cho các peer có thể lấy được địa chỉ của peer khác. Hiện nay, google đã xây dựng một số STUN server hỗ trợ hoàn toàn miễn phí như stun:stun.l.google.com:19302

**TURN**



Trong mô hình OSI, tại lớp vận chuyển (Transport layer) có hai giao thức nổi tiếng là TCP và UDP. Trong khi UDP cung cấp khả năng vận chuyển nhanh nhất có thể thì TCP lại đảm bảo truyền tin cậy. Hầu hết trong các ứng dụng thời gian thực, tín hiệu điều khiển sẽ sử dụng TCP trong khi dữ liệu sẽ truyền bằng UDP. Tuy nhiên, UDP không phải là cơ chế truyền tin cậy, dễ làm tăng số lượng các gói tin bị mất (Packet lost), do vậy đôi khi TCP được ưu tiên sử dụng.

Ở đây, dữ liệu thay vì được gửi trực tiếp tới các peer thì các peer sẽ gửi dữ liệu tới các TURN server và TURN server sẽ đóng vai trò trung gian vận chuyển gói tin. Điều này nâng cao không những chất lượng dịch vụ của ứng dụng mà còn đảm bảo an toàn thông tin khi truyền dẫn.

Về cơ bản, TURN là phần mở rộng từ STUN, vì vậy, khi phần mở rộng này không được sử dụng , thì chúng ta vẫn kết nối được thông qua STUN.Trong lúc khai báo server được sử dụng, nếu có cả địa chỉ server STUN và TURN được thêm vào. Thông thường thì STUN được ưu tiên sử dụng trước nhằm mục đích tăng tốc độ kết nối để lấy thông tin candidates của người dùng nằm sau NAT.Nếu các phương tiện không trực tiếp kết nối và làm việc với nhau, lúc này cần TURN để trao đổi trung gian bởi máy chủ Proxy.

# cáC API CƠ BẢN

1. [**MediaStream**](https://dvcs.w3.org/hg/audio/raw-file/tip/streams/StreamProcessing.html)

Một **MediaStream** là một stream dữ liệu âm thanh và/hoặc hình ảnh. Khi làm việc cục bộ, stream này được khởi tạo bằng cách gọi hàm **getUserMedia**. Sau khi một kết nối **WebRTC** tới một máy tính khác được thiết lập, chúng ta có khả năng truy cập vào stream của máy tính đó.

### Properties

* **MediaStream.active** - Trả về true nếu MediaStream đang hoạt động, hoặc false.
* **MediaStream.ended** - Trở về true nếu kết thúc *sự kiện* đã được đưa vào đối tượng, có nghĩa là dòng đã được đọc hoàn toàn, hoặc nếu kết thúc của dòng đã không đọc được.
* **MediaStream.id** - Một định danh duy nhất cho các đối tượng.
* **MediaStream.label** - Một định danh duy nhất của các danh sách người dùng chỉ định.

### Event Handlers

* **MediaStream.onactive** - Một handler cho một sự kiện được phát sinh khi một đối tượng MediaStream bắt đầu hoạt động.
* **MediaStream.onaddtrack** - Một handler cho một *addtrack* sự kiện đó phát sinh khi một mới *MediaStreamTrack* đối tượng được thêm vào.
* **MediaStream.onended** - Một handler cho một *kết thúc* sự kiện đó phát sinh khi streaming được chấm dứt.
* **MediaStream.oninactive** - Một handler cho một *hoạt động* sự kiện đó phát sinh khi một *MediaStream* đối tượng không hoạt động.
* **MediaStream.onremovetrack** - Một handler cho một *removetrack* sự kiện đó phát sinh khi một *MediaStreamTrack* đối tượng được lấy ra từ nó.

### Methods

* **MediaStream.addTrack()** - Thêm các *MediaStreamTrack* đối tượng nhất định như là đối số cho MediaStream. Nếu theo dõi đã được thêm vào, không có gì xảy ra.
* **MediaStream.clone()** - Trả về một bản sao của đối tượng MediaStream với một ID mới.
* **MediaStream.getAudioTracks()** - Trả về một danh sách các âm thanh *MediaStreamTrack* đối tượng từ *MediaStream* đối tượng.
* **MediaStream.getTrackById()** - Trả lại ca khúc của ID. Nếu đối số là rỗng hoặc các ID không được tìm thấy, nó sẽ trả về null. Nếu một số tracks có cùng một ID, nó trả về đầu tiên.
* **MediaStream.getTracks()** - Trả về một danh sách của tất cả các*MediaStreamTrack* đối tượng từ *MediaStream* đối tượng.
* **MediaStream.getVideoTracks()** - Trả về một danh sách các video *MediaStreamTrack* đối tượng từ các *MediaStream* đối tượng.
* **MediaStream.removeTrack()** - Loại bỏ các *MediaStreamTrack* đối tượng nhất định như là đối số từ MediaStream. Nếu theo dõi đã được gỡ bỏ, không có gì xảy ra.

### RTCDataChannel

### **RTCDataChannel** là một kênh hai chiều để gửi dữ liệu bất kỳ qua kết nối **WebRTC**. Nó hoạt động tương tự web sockets, nhưng dữ liệu được truyền tải trực tiếp giữa hai trình duyệt, và nó cho phép trao đổi thông tin nhanh và đáng tin cậy (giống như giữa **UDP** và **TCP**).

### Properties

* **RTCDataChannel.label** - Trả về một chuỗi chứa tên kênh dữ liệu.
* **RTCDataChannel.ordered** - Trả về true nếu lệnh gọi của các tin nhắn được đảm bảo hay sai nếu nó không được đảm bảo.
* **RTCDataChannel.protocol** - Trả về một chuỗi chứa tên subprotocol sử dụng cho kênh này.
* **RTCDataChannel.id** - Trả về một id duy nhất cho các kênh mà được đặt ở việc tạo ra các đối tượng RTCDataChannel.
* **RTCDataChannel.readyState**- Trả về enum RTCDataChannelState đại diện cho trạng thái của kết nối.
* **RTCDataChannel.bufferedAmount** - Trả về số byte đã được xếp hàng để gửi. Đây là số lượng dữ liệu mà đã chưa được gửi qua RTCDataChannel.send ().
* **RTCDataChannel.bufferedAmountLowThreshold** - Trả về số byte mà tại đó các RTCDataChannel.bufferedAmount được đưa lên là chậm. Khi RTCDataChannel.bufferedAmount giảm dưới ngưỡng này, sự kiện bufferedamountlow là fired.
* **RTCDataChannel.binaryType** - Trả về loại của dữ liệu nhị phân truyền qua kết nối. Có thể được "blob" hoặc "arraybuffer".
* **RTCDataChannel.maxPacketLifeType** - Trả về một unsigned short cho biết chiều dài trong mili giây của cửa sổ trong khi nhắn tin sẽ ở chế độ không chính xác.
* **RTCDataChannel.maxRetransmits** - Trả về một unsigned short cho biết số lần tối đa một kênh sẽ truyền lại dữ liệu nếu nó không được chuyển giao.
* **RTCDataChannel.negotiated** - Trả về một boolean cho biết nếu các kênh đã được đàm phán bởi các đại lý người dùng, hoặc các ứng dụng.
* **RTCDataChannel.reliable** - Trả về một boolean cho biết các kết nối có thể gửi tin nhắn trong chế độ không đáng tin cậy.
* **RTCDataChannel.stream** - Từ đồng nghĩa với RTCDataChannel.id

### Event Handlers

* **RTCDataChannel.onopen** - xử lý sự kiện này được gọi khi sự kiện mở là fired. Sự kiện này được gửi khi các kết nối dữ liệu đã được thành lập.
* **RTCDataChannel.onmessage** - xử lý sự kiện này được gọi khi sự kiện tin là fired. Sự kiện này được gửi khi một tin nhắn có sẵn trên các kênh dữ liệu.
* **RTCDataChannel.onbufferedamountlow** - xử lý sự kiện này được gọi khi sự kiện bufferedamoutlow là fired. Sự kiện này được gửi khi RTCDataChannel.bufferedAmount giảm dưới đây là tài sản RTCDataChannel.bufferedAmountLowThreshold.
* **RTCDataChannel.onclose** - xử lý sự kiện này được gọi khi sự kiện là fired. Sự kiện này được gửi khi các kết nối dữ liệu đã bị đóng cửa.
* **RTCDataChannel.onerror** - xử lý sự kiện này được gọi khi sự kiện lỗi là fired. Sự kiện này được gửi khi một lỗi đã được gặp.

### Methods

* **RTCDataChannel.close()** - Đóng các kênh dữ liệu.
* **RTCDataChannel.send()** - Gửi dữ liệu trong các thông số trên kênh.Các dữ liệu có thể là một blob, một chuỗi, một ArrayBuffer hoặc một ArrayBufferView.

### RTCPeerConnection

**RTCPeerConnection** là API đóng vai trò kết nối **Media Stream** và **RTCDataChannel** để trở thành **WebRTC**.**RTCPeerConnection** cung cấp một hàm API làm nhiệm vụ hoàn thành giao ước giữa hai trình duyệt. Trong suốt quá trình giao ước này, các trình duyệt cung cấp các thông tin cần thiết để thiết lập kết nối: mô tả của phiên làm việc (những tính năng của trình duyệt) và các ứng viên ICE (thông tin về cổng và địa chỉ IP công cộng). Cần lưu ý rằng mô tả của **WebRTC** không nói gì về cách thức những thông tin này được trao đổi.

Các RTCPeerConnection API là cốt lõi của các kết nối peer-to-peer giữa mỗi trình duyệt. Để tạo các đối tượng RTCPeerConnection chỉ cần viết

var pc = RTCPeerConnection(config);

### Properties

* **RTCPeerConnection.iceConnectionState** - Trả về một enum RTCIceConnectionState mô tả trạng thái của kết nối. Một sự kiện iceconnectionstatechange phát sinh khi giá trị này thay đổi.
* **RTCPeerConnection.iceGatheringState** - Trả về một enum RTCIceGatheringState mô tả trạng thái thu thập ICE cho kết nối
* **RTCPeerConnection.localDescription** - Trả về một RTCSessionDescription mô tả phiên địa phương. Nó có thể là vô giá trị nếu nó chưa được thiết lập.
* **RTCPeerConnection.peerIdentity** - Trả về một RTCIdentityAssertion. Nó bao gồm một IDP (tên miền) và một tên đại diện cho bản sắc của các đồng đẳng từ xa.
* **RTCPeerConnection.remoteDescription** - Trả về RTCSessionDescription mô tả phiên từ xa. Nó có thể là vô giá trị nếu nó chưa được thiết lập.
* **RTCPeerConnection.signalingState** - Trả về một enum RTCSignalingState mô tả các trạng thái tín hiệu của kết nối địa phương.Trạng thái này mô tả các đề nghị SDP. Một sự kiện signalingstatechange phát sinh khi giá trị này thay đổi.