acision

The platform for interoperable WebRTC

Kamailio World 2014

Evolution on the web

Sir Tim Berners-Lee creates HTML. Web -pages are static

W3C produces the DOM1 specification

WebSocket and WebRTC implementations become available

1990 1996 1998 2004 2011

Microsoft and Netscape introduce different mechanisms for DHTML

Google uses Ajax in Gmail (W3C releases 1st draft in 2006) – the dawn of web-apps

Revolution in telecoms

The revolution

Before today the operators (big and small) had full control over real-time communications because it was hard to do and substantial infrastructure investment was required.

Claude Chappe invented the optical telegraph

Alexander Graham Bell patents the telephone

From the 1960s onwards digital exchanges start to appear

From the 1990s onwards voice started to be carried on technologies developed for data networks such as ATM and IP

1792 1837 1876 1919 1960s > 1990s > 2011

First commercial electrical telegraph created by Cooke and Wheatstone

Rotary dial enters service

1963: DTMF enters service

WebSocket and WebRTC implementations become available

RTCWeb

There are a number of proprietary implementations that provide direct interactive rich communication using audio, video, collaboration, games, etc. between two peers' web-browsers. These are not interoperable, as they require non-standard extensions or plugins to work. There is a desire to standardize the basis for such communication so that interoperable communication can be established between any compatible browsers.

Real-Time Communication in WEB-Browsers (rtcweb) 2013-03-13 charter

WebRTC

The mission of the Web Real-Time Communications Working Group, part of the Ubiquitous Web Applications Activity, is to define client-side APIs to enable Real-Time Communications in Web browsers.

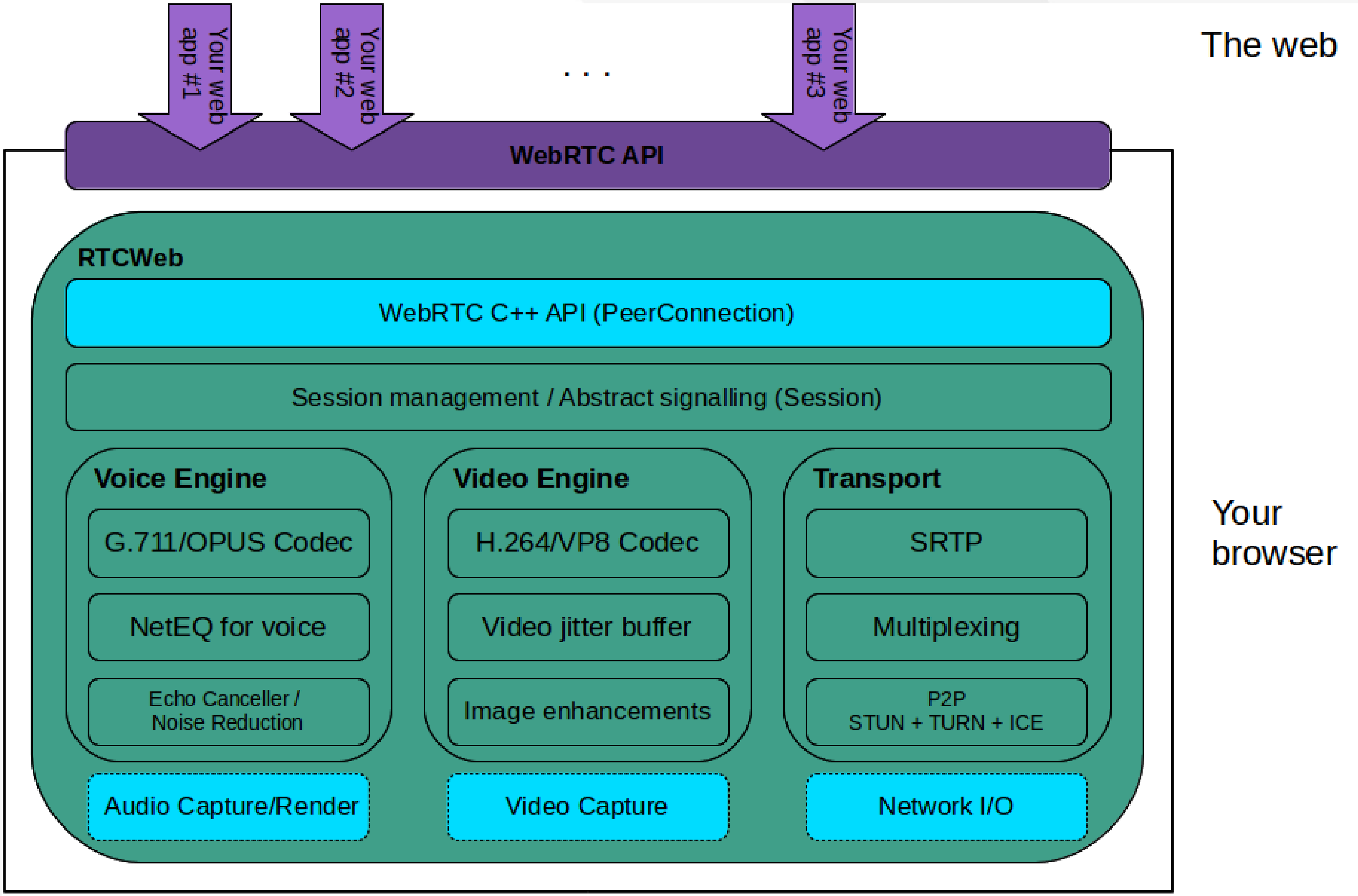
These APIs should enable building applications that can be run inside a browser, requiring no extra downloads or plugins, that allow communication between parties using audio, video and supplementary real-time communication, without having to use intervening servers (unless needed for firewall traversal, or for providing intermediary services).

Web Real-Time Communications Working Group Charter

RTCWeb and WebRTC: not the same thing

- RTCWeb is the on-the-wire protocol as defined by the IETF and may be used in many applications and systems
  – Within VoIP phones
  – On network servers
  – Includes MTI codecs for audio and video
- WebRTC is the browser API as defined by the IETF

RTCWeb/WebRTC Architecture



Based on the diagram from http://www.webrtc.org/reference/architecture

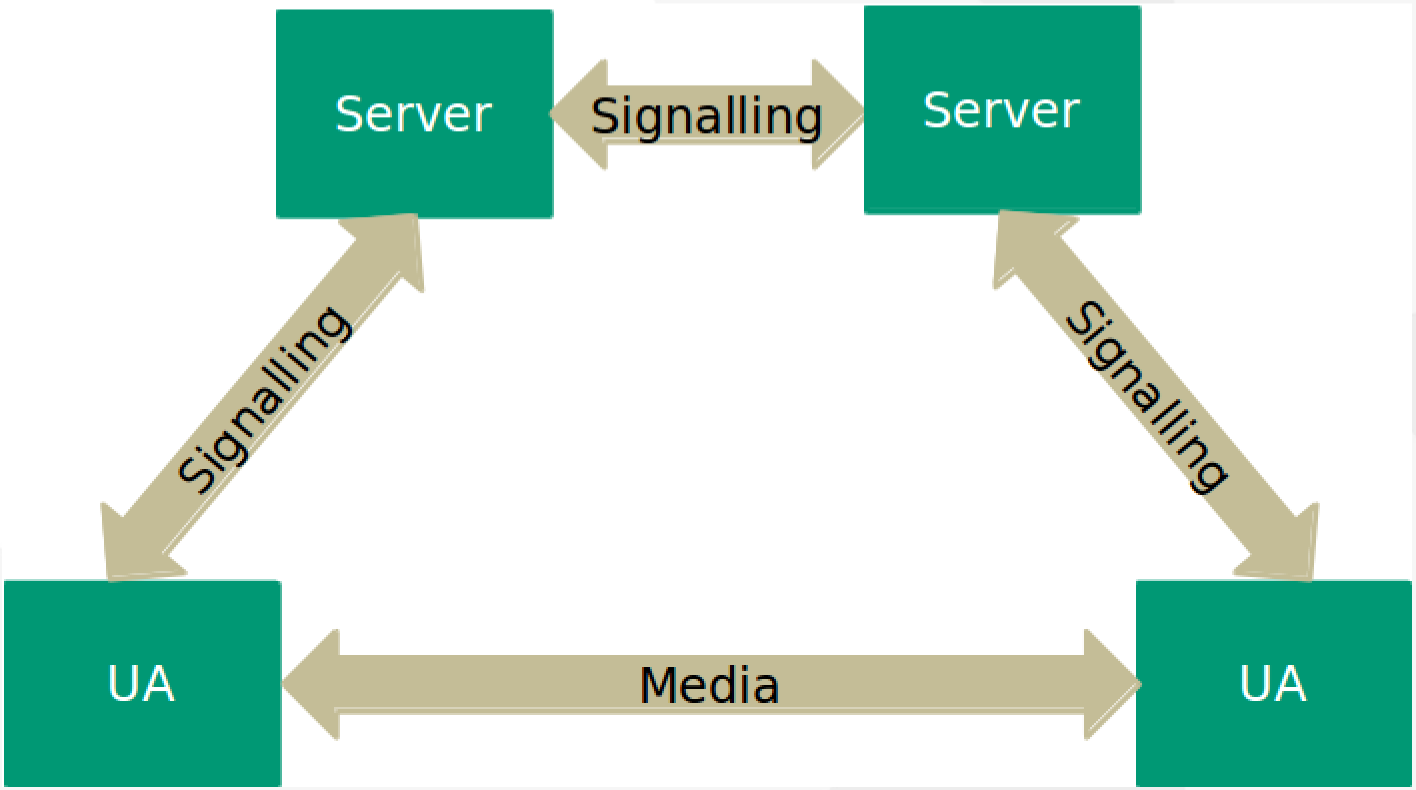
Signalling not included

- Google made a controversial (but very wise) decision not to specify how the signalling should work
- Signalling is required
  - To discover who to communicate with
  - To exchange information on what the communication should be (audio, data, video, and codecs)
- Even the simplest, proprietary, RESTful exchange is signalling

Interoperability is sometimes required

- These use-cases are typically ones where the point of the application is communication
- For example:
  - Conferencing – calls in and out of legacy networks are required
  - Call Centres – calls in and out of legacy networks are required
  - Virtual PBX – calls in and out of legacy networks are required

The signalling trapezoid



Signalling options

- Open standards are usually best
  - SIP over WebSocket, http://tools.ietf.org/html/rfc7118
  - XMPP over WebSocket, http://tools.ietf.org/html/draft-moffitt-xmpp-over-websocket
  - OpenPeer, http://openpeer.org/
- The WebRTC API is easy but signalling is often hard
  - There are many open-source libraries that do the signalling
  - The library APIs vary in complexity to meet every need
  - Hosted infrastructure lets you add real-time communications to your website without having to build a network yourself

HOWTO: Kamailio as a SIP over WebSocket server

- Download, build, and install Kamailio 4.1
- Create a kamailio.cfg file based on the following code snippets

http://www.kamailio.org/

Handling WebSocket handshakes in Kamailio

```
...
tcp_accept_no_cl=yes
...
event_route[xhttp:request] {
    set_reply_close();
    set_reply_no_connect();

    if ($hdr(Upgrade)=~"websocket"
        && $hdr(Connection)=~"Upgrade"
        && $rm=~"GET") {

        # Validate as required (Host:, Origin:, Cookie:)

        if (ws_handle_handshake())
            exit;
    }

    xhttp_reply("404", "Not Found", "", "");
}
```

WebSocket connections are <u>always</u> behind a NAT

- Javascript applications cannot see the real IP address and port for the WebSocket connection
- This means that the SIP server cannot trust addresses and ports in SIP messages received over WebSockets
- nathelper <u>and/or</u> outbound can be used to solve this problem

Using nathelper on SIP over WebSocket requests

```
modparam("nathelper|registrar", "received_avp", "$avp(RECEIVED)")
...
request_route {
	route(REQINIT);
	route(WSDETECT);
	...
route[WSDETECT] {
	if (proto == WS || proto == WSS) {
		force_rport();
		if (is_method("REGISTER")) {
			fix_nated_register();
		} else if (is_method("INVITE|NOTIFY|SUBSCRIBE")) {
			add_contact_alias();
		}
	}
}
...
route[WITHINDLG] {
	if (has_totag()) {
		if (loose_route()) {
			if (!isdsturiset()) {
				handle_ruri_alias();
			}
			...
```

Using nathelper on SIP over WebSocket responses

```
onreply_route {
    if ((proto == WS || proto == WSS) && status =~ "[12][0-9][0-9]") {
        add_contact_alias();
    }
}
```

WebSocket connections are <u>always</u> behind a NAT

- Use mediaproxy-ng from SIPWise

  https://github.com/sipwise/mediaproxy-ng

- Companion Kamailio module: rtpproxy-ng

  http://kamailio.org/docs/modules/stable/modules/rtpproxy-ng.html

- SIP Signalling is proxied instead of B2BUA'd (that is, not broken)

Catch 488 to invoke mediaproxy-ng

```
modparam("rtpproxy-ng", "rtpproxy_sock", "udp:localhost:22223")
...
route[LOCATION] {
          ...
          t_on_failure("UA_FAILURE");
}
...
failure_route[UA_FAILURE] {
     if (t_check_status("488") && sdp_content()) {
          if (sdp_get_line_startswith("$avp(mline)", "m=")) {
               if ($avp(mline) =~ "SAVPF")) {
                    $avp(rtpproxy_offer_flags) = "froc-sp";
                    $avp(rtpproxy_answer_flags) = "froc+SP";
               } else {
                    $avp(rtpproxy_offer_flags) = "froc+SP";
                    $avp(rtpproxy_answer_flags) = "froc-sp";
               }
               # In a production system you probably need to catch
               # "RTP/SAVP" and "RTP/AVPF" and handle them correctly
               # too
          }
          append_branch();
          rtpproxy_offer($avp(rtpproxy_offer_flags));
          t_on_reply("RTPPROXY_REPLY");
          route(RELAY);
     }
}
...
```

Handle replies to the retried INVITE

```
modparam("rtpproxy-ng", "rtpproxy_sock", "udp:localhost:22223")
...
failure_route[UA_FAILURE] {
     ...
     t_on_reply("RTPPROXY_REPLY");
     route(RELAY);
}

onreply_route[RTPPROXY_REPLY] {
     if (status =~ "18[03]") {
          # mediaproxy-ng only supports SRTP/SDES - early media
          # won't work so strip it out now to avoid problems
          change_reply_status(180, "Ringing");
          remove_body();
     } else if (status =~ "2[0-9][0-9]" && sdp_content()) {
          rtpproxy_answer($avp(rtpproxy_answer_flags));
     }
}
...
```

Current mediaproxy-ng limitations

- No support for SRTP/DTLS
  - SRTP/DTLS is a <u>MUST</u> for WebRTC and SRTP/SDES is a <u>MUST NOT</u>
  - mediaproxy-ng works with Google Chrome today (but Google will be removing SRTP/SDES over the next year)
  - mediaproxy-ng does not work with Firefox at this time
- Does not support “bundling”/”unbundling”
  - WebRTC can “bundle” audio and video streams together, but mediaproxy-ng does not support this yet
  - Google Chrome does not currently support “unbundling”
  - You can have an audio stream, or a video stream, but not an audio <u>and</u> video stream at this time

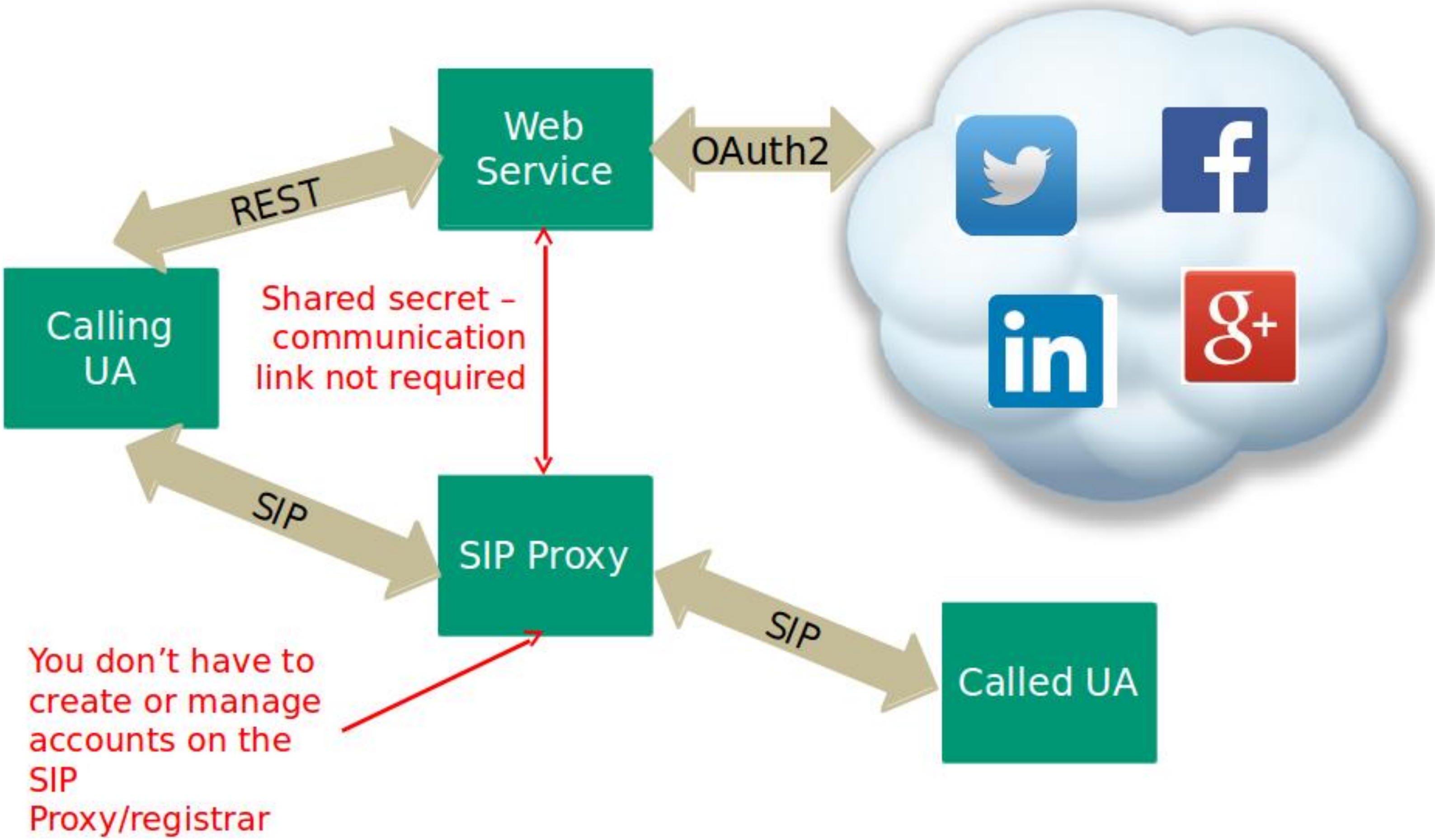
RTPEngine is coming

HOWTO: Authenticate SIP using a web-service

- No communication required between authentication server and Kamailio
- Credentials expire (the expiry time is chosen by the authentication server)
- Extract username and password from the "GET" used for HTTP handshake and authenticate there, or
- Use the credentials for digest authentication of SIP requests
- Check the From-URI or To-URI in SIP headers match the user part of the credential

http://kamailio.org/docs/modules/stable/modules/auth_ephemeral.html

Ephemeral Authentication



Authenticating the handshake

```
...
tcp_accept_no_cl=yes
...
modparam("auth_ephemeral", "secret", "kamailio_rules")
...
modparam("htable", "htable", "wsconn=>size=8;")
...
event_route[xhttp:request] {
     ...
          # URI format is /?username=foo&password=bar
          $var(uri_params) = $(hu{url.querystring});
          $var(username) = $(var(uri_params){param.name,username,&});
          $var(password) = $(var(uri_params){param.name,password,&});
          # Note: username and password could also have been in a Cookie: header

          if (!autheph_authenticate("$var(username)", "$var(password)")) {
               xhttp_reply("403", "Forbidden", "", "");
               exit;
          }

          if (ws_handle_handshake()) {
               $sht(wsconn=>$si:$sp::username) = $var(username)
               exit;
          }
     ...
event_route[websocket:closed] {
     $var(regex) = $si + ":" $sp + ".*";
     sht_rm_name_re("wsconn=>$var(regex)");
}
```

Checking SIP requests

```
...
request_route {
     route(REQINIT);
     route(WSDETECT);
     ...
     if (!(proto == WS || proto == WSS))
          route(AUTH);
     ...
route[WSDETECT] {
     if (proto == WS || proto == WSS) {
          $var(username) = (str) $sht(wsconn=>$si:$sp::username);
          if ($var(username) == $null || $var(username) == "") {
               send_reply("403", "Forbidden");
               ws_close(1008, "Policy Violation");
               exit;
          }

          if (!autheph_check_timestamp("$var(username)")
                    || (is_method("REGISTER|PUBLISH")
                              && !autheph_check_to("$var(username)"))
                    || (!has_totag() && !autheph_check_from("$var(username)"))) {
               send_reply("403", "Forbidden");
               ws_close(1008, "Policy Violation");
               exit;
          }

          force_rport();
          ...
```

Questions?

Email: peter.dunkley@acision.com
Twitter: @pdunkley