PEER-TO-PEER REAL-TIME CLOUD COMMUNICATION SOLUTIONS

George-Alex STELEA, Radu CURPEN, Ștefania ȘURARIU, Alexandru PAPOI
Transilvania University, Brașov, Romania (george.stelea@unitbv.ro, curpenradu@hotmail.com, stefania.surariu@gmail.com, alexandru.papoi@yahoo.com)

DOI: 10.19062/1842-9238.2016.14.2.12

Abstract: *Telecommunications are part of a developing field that tries to improve constantly. Related services require accuracy, speed, and integration simplicity in complex applications. New technologies aimed at efficient and low cost communication, involving the internet and taking place in real-time. This paper is dedicated to WebRTC (Web Real-Time Communication) technology, presenting its architecture, development environment, advantages, targeted markets as well as the implementation of a real-time communication application using Unify Circuit, Node.js, Express Framework and OpenSSL.*

Keywords: *real-time communications, peer-to-peer, application programming interface, plugin free communications, internet of things (IoT).*

1. INTRODUCTION

WebRTC, short for Web Real-Time Communications, is a new standard that the W3C (World Wide Web Consortium) and the IETF (Internet Engineering Task Force) have defined to enable web browsers to conduct peer-to-peer real time communications through a series of common protocols.

Before WebRTC, in order to achieve real-time audio, video or data communications on the web browsers, the users had to install third party plugins that made the experience complicated and troublesome, and those plugins needed to be updated very often because of security vulnerabilities.

With WebRTC (an application and plugin free technology), HTML5 and JavaScript, can be used to get access to the webcam and the microphone from a computer, and send audio and video content to another browser as well as data content [1]. A web browser can become a multimedia endpoint revolutionizing the way users communicate over the Internet. Now a WebRTC gateway can be used to obtain a connection to a traditional carrier network or a PBX (Private Branch Exchange). This gateway uses SIP (Session Initiation Protocol), an arduous protocol to talk to the traditional network, but provides a RESTful (Representational state transfer) API based on HTML5 and JavaScript [2] facilitating the process of signaling. A web developer can use a PaaS (Platform as a Service) to embed real time communication into a web application with web RTC. WebRTC enables point-to-point or point-to-multipoint [3] communications without plugins as it can be simply embedded into browsers - a web server is only needed to exchange information and control data between peers.

The idea of RTC isn't new in itself - Technologies are new! Recently, Huawei performed a 5G demonstration providing ”around 1 Gbps per UE”(User Equipment) claiming ”a latency of 0.5 milliseconds”[4].

2. WEBRTC ARCHITECTURE

Traditional web architectures are built on client-server model where communication is made unidirectional, from server to client over HTTP.

WebRTC architecture improves the client-server model by adding the concept of peer-to-peer communication between web browsers. In WebRTC the most common model is the triangular architecture - Fig.1(a) (or trapezoidal architecture Fig.1(b) in some cases), browsers accessing a web application on a server through which signaling messages are transmitted via HTTP or WebSocket [5]. The server is used to connect the data control, but then the media stream is only peer-to-peer between browsers.

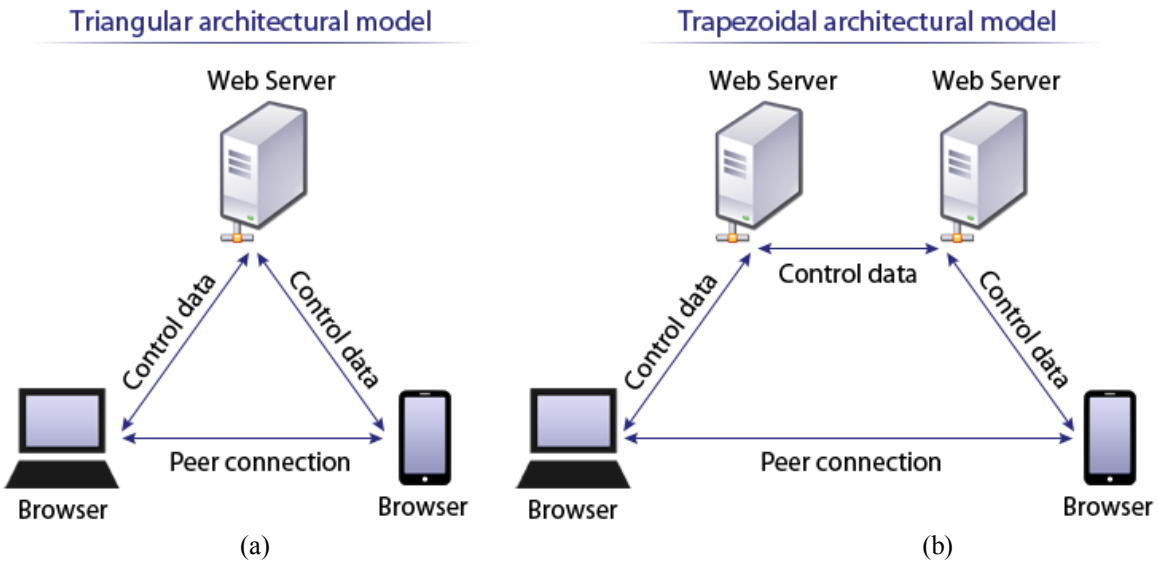

FIG. 1. WebRTC triangular and trapezoidal architectural models

3. WEBRTC APPLICATIONS

WebRTC applications are typically written in JavaScript and HTML5 and are processed by the browsers using the WebRTC API integrated into them, through which it performs functions of connection management, encoding and decoding, media control and NAT translation.

NAT is short for Network Address Translation and is the process whereby a limited set of IPs (IPv4 only support about 4 billion addresses [6]) are divided to work on more devices. In this way many computers can connect to a router interface and the router to the Internet.

4. SIGNALING IN WEBRTC

WebRTC standard does not define a specific signaling protocol other than the generic SDP (Session Description Protocol - a protocol similar to SIP because it uses the same technology) used to exchange control messages.

WebRTC signaling is left opened, allowing flexibility in design, granting developers choosing between using a gateway like WebSocket, Socket.io, XMPP, SIP or Smoke signals. The signaling process in WebRTC is based on a new standard proposed by IETF called JSEP - JavaScript Session Establishment Protocol, which is a collection of interfaces used for signaling identification.

5. WEBRTC API

The WebRTC API is an extension to HTML5, having the ability to start setting up a call information or a communication information and can be integrated into standalone or embedded web browsers and even intro smartphone applications that have the web browsing technology built intro them. Web RTC offers several standard tools used in web technologies, HTML for displaying the objects and JavaScript for programming the communication part and controlling the information objects.

The WebRTC API stack, as showed in Fig.2 is built on three concepts, MediaStream, PeerConnection, and DataChannel:

- *MediaStream* - is an interface that defines a stream media type content. A stream is represented by several audio or video tracks. Each track is defined as an instance of the MediaStreamTrack interface (Fig. 3);
- *PeerConnection* - is an interface that defines a connection between the local browser and a remote peer;
- *DataChannel* - defines methods to connect to a remote peer, monitoring and maintaining the connection and close the connection when it is no longer necessary.

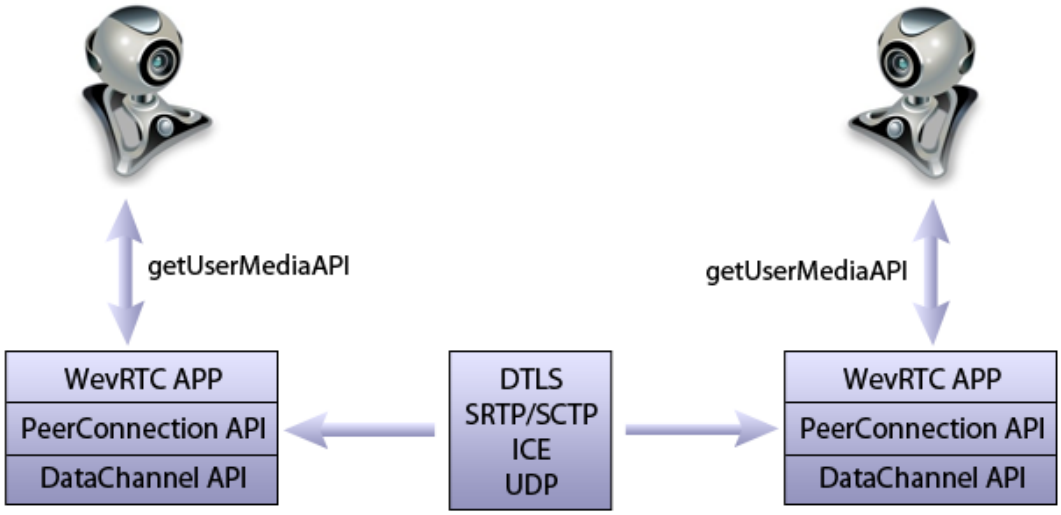

FIG. 2. WebRTC API Stack

```
<video id="videoSource" autoplay></video>
<script type="text/javascript">
    var video = document.getElementById('videoSource');
    navigator.getUserMedia('video', success, error);
    function success(stream) {
        video.src = window.URL.createObjectURL(stream);
    }
</script>
```

FIG. 3. MediaStream *getUserMedia()* function

ICE, short for Interactive Connectivity Establishment, is one of the lower layers in the protocol stack, above UDP, used to solve the problem of establishing the connection through a NAT device like a router or a firewall, without setting up port forwarding to get packets that are coming from unknown hosts outside the network and to be forwarded to a specific machine inside the network.

ICE is an amalgamation of two protocols: STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT). ICE is specified in a way supposed to be independent from SIP (Session Initiation Protocol), and uses SDP (Session Description Protocol) like an interchange format describing the local addresses to encode and send them through a third party to the peer needed to connect to, as shown in Fig.4.

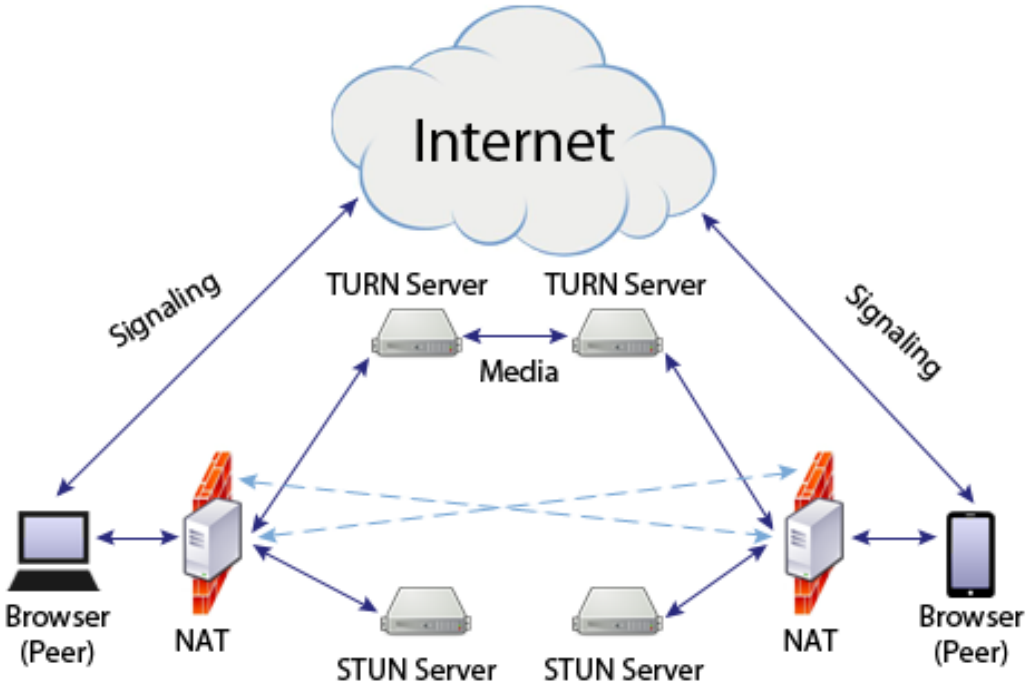

FIG.4. ICE - Interactivity Connection Establishment using STUN and TURN

6. WEBRTC MARKETS

The WebRTC technology offers practical and decentralized tools for communication and in conjunction with the widespread browser support, generates feasible applicability to a variety of market niches such as:

- *Healthcare* - there could be created derived products used to develop telemedicine providing encrypted connections in P2P system and ensuring enhanced security connections that allow for the development of physician patient healthcare services, remotely transmitting medical information, and even wearable devices providing remote diagnosis. A patient can communicate with a hospital in a website like a "nurse hot-line" - speak or have a videoconference - without having an arrangement with the hospital, an appointment or filling in an application. The patient can go to a web page and have instant live (real time) access to medical services through the click of a button;

- *Training and higher education* - students can bring their own devices into the campus environment enabling those students to communicate with university staff and professors or with other students, not having to worry about different operating services or applications support, basically they can have access through a web application (without plugins or downloading and install native applications to their devices);
- *Conferencing* - a very large use in the conferencing marketplace, especially on the business environment, there are already applications developed for web conferencing through WebRTC enabled web pages starting conferencing by a click distance without pre-installed software or downloading extra code;
- *Customer service* - the use of WebRTC could revolutionize and democratize the customer service solutions and support using applications enabled with voice, video and data communication, especially to e-commerce business or technical support services enabling the possibility to do co-browsing assisting to help a customer resolve a problem;
- *Social media* - social networks continue to grow rapidly creating a large marketplace for new applications and products. Most of the popular social media platforms already started to use and adapt WebRTC.

7. ADVANTAGES AND REASONS TO USE WEBRTC

7.1. Need for rapid development - in a very fast moving world, the possibility to build quick and reliable applications become an important asset in the growth on the business sector. Using WebRTC built-in technology into the web browser allows to rapidly deploy and develop new scalable applications that can be accessed by users instantly;

7.2. Embedded communication - because the web browsers implement an underlying technology HTTP and HTTPS and underneath that is HTML which represents the format of the information that comes down, it can be embedded anywhere, not only on the web browsers - web technology is the ability to go to the web servers to grab the information and to display (render) it and can be built into any native applications on smartphones or other modern devices;

7.3. Viruses, Malware, Business and industrial Spying and Ransomware - People are growing reluctant to download plugins and installing new software, companies are locking down systems, not allowing employees to install programs without the accept of the organization because downloading programs to the devices can open security risks, making the IT department overloaded. With WebRTC the IT costs can be lowered, not having to worry about downloading new software because the technologies needed for communication are built into the web browser itself and into the protocol that the browser uses. In WebRTC communication the signaling encryption can be achieved with DTLS (Datagram Transport Layer Security) and the media can be encrypted using SRTP (Secure Real-time Transport Protocol) avoiding "man-in-the-middle" attacks [7];

7.4. Internet of Things (IoT) - the IoT concept assumes controlling via the Internet al the internet enabled devices that surround us [8] - smartwatches, phones, appliances, lighting systems and even cars [9]. WebRTC technology can attain greater connectivity, easy access to data, interconnected ecosystems, achieving lower costs and optimizing resource efficiency.

8. BUILDING A WEBRTC APPLICATION

8.1. Application development environment is extensive, comprising installing software development services and software platforms for security and encryption such as Java SDK (Java software development kit), Node.js, Python, Circuit and OpenSSL.

8.2. Call control module: in Node.js, all modules export an item that can easily be called anywhere in the code [10, 11] and for logging an instance of Express module called app was used. Express is a rapid web development module that provides various functions for viewing, testing, routing, negotiation content in a web page, but also for generating an executable application.

After calling the logger function the app.get function was used to get users from database with user.js subprogram. App.get function is structured as (route, reply) and is used for routing the endpoints (URIs) and responding to customer demand through HTTP GET method. In this case, the function points to the directory users, demanding the user list as a response.

The user.js subprogram, which is the software solution driver, connects the sub composing web application and creates a number of variables Java Script connecting them in different packages, dependencies, Node functionality or routes, when a new Express Node.js project starts.

The logon() function compares the login email and password with the database from the Circuit platform, then creates a Client instance and through events method grants user access in the application. (Fig. 5.)

```
_client.logon($email.value, $password.value).then(function(user){
_localUser = user;
$email.disabled = true; //default filled field
$password.disabled = true; //default filled field
$login_wall.style.display = 'none';
$content.style.display = '';
$logonButton.style.display = 'none';
document.getElementById('name').innerHTML = user.displayName;
document.getElementById('right_side').style.display = '';
})
```

FIG.5. *logon()* function

The second operation after the login, is grabbing conversations already taken by the user. For exceptions, the catch function was used to verify if the logging takes place successfully, or if the username or password are incorrect. The log out function sets the Client module off, leaving the application and returning to the original page.

The function that changes the connection status of the application is performed by addEventListeners() method, attaching an event to the Client element, without overwriting other already created events, as showed in Fig.6.

```
function addClientEventListeners() { //adding a client
 if (_client) {
    _client.addEventListener('connectionStateChanged',
    function onConnectionStateChanged(evt) { //change status
 prin crearea unui eveniment
    console.log('Received connectionStateChanged event-state=',
 evt.state)
   $logonState.textContent = evt.state;
   //adding "connected" or "disconnected"
   if (evt.state === Circuit.Enums.ConnectionState.Disconnected){
   //in case of disconnection returns to initial interface
        resetCallUI(); //interface reset
     }
 });
```

FIG.6. *addClientEventListeners() function*

8.3. Connection management: to make the connection between the two entities, the data link layer: path and app.js were set. The app.set function (name, value) sets the name and value of the local port where the ('port') application is running.

The index.js subprogram contains the request-response function which exports the web page that includes the application. Creating the virtual server that hosts the application, in this case https://localhost:8080 is made by the function showed in Fig.7.

```
//creating virtual server
var httpsServer = https.createServer(credentials, app);
httpsServer.listen(app.get('port'));
```

FIG.7. *addClientEventListeners()* function

Another connection aspect is adding the <script> tag containing the sub-interface elements index.ejs. The tags refer to a link either to a script document type or to a local client.js file. The code in Fig.8 connects the index.ejs interface to all subprograms and functionality of the main program called client.js.

```
<script src='https://circuitsandbox.net/circuit.js'></script>
<script src='js/client.js'></script>
```

FIG.8. *index.ejs* interface

8.4. Flow management: the start() function first checks if the user is logged on and creates average time variables, then a conversation is selected and media is streaming. The join() function also checks if the user is logged in, create variables for media types and allows the user to choose a conversation. To attend a conversation, the function must state "started". The leave() function has the same elements as the join() function, but user the leave() module and depending on the ID of the conversation, removes the user from

the conference. Changing the conference status when the conference ends takes place through the updateList(call) function as showed in Fig.9.

```
function updateList(call) { //updates conversations list
   if (call.state === Circuit.Enums.CallStateName.Terminated) {
   //if the conference ends
       getListOption(call.convId).textContent =
                   _conversations[call.convId].title;
   } else {
       getListOption(call.convId).textContent =
       _conversations[call.convId].title + '('+ call.state +')';}
   //if not ended further display the status
```

FIG.9. *updateList()* function

8.5. Graphical user interface (GUI): A setCallUI() function was created and used every time when it came to setting or resetting the user interface. In the following function the button which accesses the conference is changed.

The button has two options: "Start" - which refers to the start of a conference, or 'Enter' - which refers to join a conference that has already begun. (Fig. 10.).

```
function setCallUI() {
 updateButtons(); //text based on conversation
 var convId = document.getElementById('convList').value;
  if (!_conversations[convId]) {
     return;
 }
 var call = _calls[convId]; //returns the ID
 $convId.textContent = call.convId;
 $callState.textContent = call.state; //conversation status
 $callId.textContent = call.callId;
```

FIG.10. *setCallUI()* function

When joining a conversation, audio and video stream is automatically activated and sent to onVideoChange() function. The resetCallUI() function, showed on Fig. 11, is called for every change made by other functions while accessing the application or conversation.

```
function resetCallUI() {
 updateButtons(); //change button after joining conversation
 $callState.textContent = ''; //button status after join
 $media.textContent = '';
 $callId.textContent = '';
 $convId.textContent = '';
 $localVideo.src = ''; //local video
 $remoteVideo1.src = ''; //video from other users
 $remoteVideo2.src = ''; //... for 6 users
 $remoteAudio.src = ''; //audio from other users
 $enableVideo.checked = false; //does not allow video stream
 $enableVideo.disabled = true; } //inactive checkbox
```

FIG.11. *resetCallUI()* function

The updateButtons(), shown in Fig.12, relates to the button that changes its text depending on the state of the conversation. If the conference has already started, the function will output "Join" and if the users starts the conference the function will output "Start" and hide the "Join" button. If the conference is not started the function will hide the "Exit conference" button.

```
function updateButtons() {
 $startButton.classList.add('hide'); //hide "Start" button
 $joinButton.classList.add('hide'); //hide "Join" button
 $leaveButton.classList.add('hide'); //hide "Leave" button
 if (!call) { //If user has not yet entered a conference

   $startButton.classList.remove('hide'); //hide "Start" button
 } else if (call.state === Circuit.Enums.CallStateName.Started) {
                //if a conversation is already "Started"
   $joinButton.classList.remove('hide'); //hide "Join" button
 } else if (call.state === Circuit.Enums.CallStateName.ActiveRemote){
     $leaveButton.classList.remove('hide'); //hide "Leave" button }}
```

FIG.12. *updateButtons()* function

4. CONCLUSIONS AND FURTHER DEVELOPMENTS

The usage of WebRTC technology reduces physical resources: cables, physical input-output ports, memory and also software maintenance resources: operating system (OS) updates, installing or uninstalling software on endpoints and even security aspects, monitoring and controlling them on a shared server. The solution was tested using "Circuit" API-s, in real-time applications of video-conferencing (as the main target were cloud communications, other forms of live streaming - e.g. instrumental acquired data flows - are intended in future developments). There are already solutions like Google Duo providing live preview of the caller's face from her/his smartphone front-camera.

A WebRTC success factor is that its core technologies (HTML, HTTP, TCP / IP) are open and implementable. Already integrated with the best solutions for voice and video, WebRTC includes firewall traversing technology using STUN (datagram user protocol through NAT - network address translation), interactive connectivity and RTP-over-TCP support for proxies. Also, the developer can use any protocol according to its usage scenario, such as XMPP, Jingle or SIP, which gives more freedom of choice.

The specific of our solution is the ”seamless” integration of the RTC in the network - in the ”Web” - assuming a high performance. We consider to include a micro-test of network state (benchmarks of bandwidth or, at least, a connectivity check) in our solution in order to react on different network states. Given the highly interactive character of our conferencing Web-RTC solution, a subjective ”down-grade” reaction is possible (blocking video to allow a better voice service or blocking both streams to allow at least a written chat).

As our proof-of concept illustrated, WebRTC can be integrated and adjusted for any user and customer demand. Currently, some of the largest companies in the world - Google, Amazon, Oracle, Facebook, Twilio, Citrix, etc - have already implemented this technology in their applications just one year after its appearance on the market.

This underlines the upsurge of developing and integrating WebRTC in web applications and the trend remote communications will take via Cloud.

REFERENCES

[1] Salvatore Loreto, Simon Pietro Romano, *Real-Time Communication with WebRTC*, O’Reilly Media, Inc., 2014;

[2] Alan B. Johnston, Daniel C. Burnett, *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, Digital Codex LLC, 2014;

[3] Ruben Picek, Samuel Picek, *WebRTC Multipoint Web Real-Time Communication*, Springer International Publishing, 2014;

[4] Bicheno Scott - Huawei claims first 4.5 GHz large scale 5G field trial - http://telecoms.com/477482/huawei-claims-first-4-5-ghz-large-scale-5g-field-trial[accessed Nov.2016];

[5] Andrew Lombardi, *WebSocket: Lightweight Client-Server Communications*, O’Reilly Media, Inc., 2015;

[6] D. Enache, M.Alexandru, „A Study of the Technology Transition from IPV4 to IPV6 for an ISP” *Review of the Air Force Academy*, nr. 1, 2016.

[7] Siani Pearson, George Yee, *Privacy and Security for Cloud Computing*, Springer-Verlag London, 2013;

[8] A. Balica, C. Costache, F. Sandu, D. Robu „ Deep Packet Inspection for M2M Flow Discrimination - Integration on an ATCA Platform” *Review of the Air Force Academy*, nr. 2, 2014;

[9] Sorin Zamfir, Titus Bălan, Florin Sandu, Cosmin Costache - *Mobile communication solutions for the services in the Internet of Things* - Proceedings of the 7th International Conference on Exploring Service Science, IESS 1.6 - Springer International, pp 619-634, 2016;

[10] Mike Cantelon, Marc Harter, T.J. Holowaychuk, Nathan Rajlich, *Node.js in Action*, Manning Publications Co., 2013;

[11] A. Mardan, *Practical Node.js: Building Real-World Scalable Web Apps*, Apress, 2014.