ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 5, Issue 5, May 2016

The Web-Telecom Capsule: Bridging heterogeneous technologies for telephony services

Dnyaneshwari Nirwan, Dr. Shashikant Ghumbre

***Abstract*— Several approaches are emerging for exposing Telecom capabilities on the web and them being accessed through web browsers. Enterprises resort to ReST APIs in order to allow communication services on the Web. A majority of the web applications are architected using the ReST style. A shift in this paradigm came with the introduction of WebSockets. WebSockets enable two-way real-time communication between clients and servers in web-based application. The socket that is connected to the server stays open for communication allowing data to be pushed to the browser in real-time on demand. As different solutions were being proposed, IETF came up with Session Initiation Protocol (SIP). Session Initiation Protocol (SIP) is becoming the signaling protocol of choice in the IP telephony, VoIP and unified communications (UC) industries. Our work aims at in exploring how these technologies when used in unison leverage the experience of telephony services. The contribution of this work is three-fold. Firstly we introduce the Web-Telecom Capsule. We take a look into the composition of the Web-Telecom Capsule, a part of the Telecom ecosystem being discussed here, consisting of Telecom functionalities built using the heterogeneous components, ReST, SIP and WebSockets. Secondly, to overcome the challenge of testing any telephony based application having components based on different technologies, we designed and developed a prototype which acted as a middleware and aided in testing the components in isolation, away from the integrated system. Lastly, we evaluate the performance of all three technologies, ReSTful webservices, WebSockets and SIP.**

Keywords: Telephony, ReST API, WebSockets, SIP, Web-Telecom Capsule, Performance evaluation.

I. Introduction

Over the years, methods used by businesses for communication have undergone major paradigm shifts and they are still undergoing changes continuously. A decade ago, most business communications, internal or external, were centered on the telephone. Emails brought in a paradigm shift which made life convenient. But email has its own shortcomings and hence couldn't become a vehicle for instant communication. Email is now making way for real-time communications. Centered around a wide variety of devices which can be used on daily basis, including PCs, tablets, smartphones etc. real-time communication is supported on the base of different technologies like ReSTful webservices, WebSockets, Comet, SIP and many more. Thus the document-centric nature of the Web is increasingly becoming more interactive and collaborative. Considered as a best practice to build distributed hypermedia systems is the Representational State Transfer (REST) [1] design style. The research and industrial communities have proposed REST-based APIs [2], which expose communication services implemented according to the Session Initiation Protocol (SIP) specifications [3] in order to facilitate the development of web applications leveraging Web-based and Telecom-centric converged services. Recent standardization efforts by the Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C) are defining specifications aimed at web browsers evolving towards natively support voice and video communication (i.e., "Real-time Communication Between Browsers" [4] [5]). To set up and handle voice and video sessions interworking with legacy SIP User Agents, as discussed in [6] and [7], a signaling path between web browsers and intermediaries (i.e., web servers) is required.

In this context, this works aims at investigating how the technologies like ReST, WebSockets and SIP are becoming the pillars for the development of web-based Telephony services and overall enhancing the experience of real-time communications. The approach makes use of set of ReST APIs to control the signaling message exchange for the audio/video call setup between two web browsers/ devices. This envisions that the browser/device will contain all the necessary functions needed to run a Web application which will work in conjunction with back-end servers to implement telephony functions as required. ReST is based on HTTP — the only protocol web clients speak which is inherently mono-directional. Thus requests can originate only at the client in the ReST style architecture. For part of the web application where client initiated communication is required, we make use of WebSocket, a messaging protocol providing methods to send and receive messages for both ends of a communication. It uses a special HTTP request which can be sent to the server and, after the initial handshake, the client and server can freely and asynchronously send frames to each other. Thus WebSocket takes care of asynchronous notifications leveraging the real-time communication. We have designed these interfaces with reference to the SIP signaling protocol specifications [3] for two main reasons: first, SIP is a major specification for interactive multimedia communication, and, second, we want to ensure interoperability with SIP User Agents. The integrated testing of the components of the

technologies mentioned above is not straightforward and inherently complex. To bring about seamless interaction of these components and aid testing of the application as a whole, we have implemented a prototype using node.js. This prototype acts as an intermediary component and is accessible by web browsers as well as SIP User Agents. This prototype leverages the WebRTC specifications [4] for the media channel setting between two web browsers and the WebSocket protocol [8] for handling asynchronous notifications and also the SIP channel.
This paper is organized as follows: Section II which discusses the background of the technologies. Section III discusses the architecture of the system and implementation of the prototype. Section IV discusses the results of the performance testing of the components mentioned above. Section V concludes the paper with some insight on future work.

II. BACKGROUND

A. ReST in a nutshell

The main concept of the Representational State Transfer (REST) [1] architectural style is related to "resource". Resources are nothing but abstract information entities. As per the ReST style, objects and data sets handles by the client-server application logic are modeled as resources. ReST is most commonly used with HTTP protocol.
ReST rests on 5 of these key principles: i) Servers use URIs to expose resources making URIs resource identifiers; ii) Manipulation of these resources is done using fixed set of primitives, i.e., create, read, update and delete (in HTTP they are mapped to the PUT, GET, POST and DELETE methods); iii) only the information required for its management is present in each message, i.e., the messages are Self-descriptive; iv) Interactions must be stateless i.e., the client request must contain all the information necessary to understand the current request, independently of any previous request. The session state and resource state can be distinguished as per this principle. The session state is maintained and taken care of at the client only, no request data is maintained at the server. The resource state is taken care of by the server that exposes it; v) Hypermedia, as the engine of application state, a hypermedia system is characterized by servers transferring resource representations that contain links, while the client can proceed to the next step in the interaction by choosing one of these links.

B. WebSockets in a nutshell

WebSockets is the technology which provides full-duplex communication, bi-directional channels over a single transmission control protocol socket. It is a framing protocol over TCP/IP protocol. Initiated by a client using a special HTTP request to the server and, after the initial handshake, the client and server can freely and asynchronously send frames to each other. The API to use the WebSockets protocol is defined by the HTML5 WebSockets specification and W3C and IETF have standardized the WebSockets API and protocol respectively. The protocol can be thought of as divided in two phases: i) The initial handshake; ii) the data transfer. The initial handshake involves a message being sent from the client and receiving the handshake response from the server. WebSockets provide a standard that can be used to build scalable, real-time web application and it provides a socket that is native to the browser. During the initial handshake while establishing a WebSocket connection, the client and server upgrade from the HTTP protocol to the WebSocket protocol. After the WebSocket connection is established, WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode. Both text and binary frames can be sent in either direction at the same time. Simulating bi-directional browser communication over HTTP is error-prone and complex and all that complexity does not scale and thus WebSockets are gaining popularity over HTTP. WebSockets have a unique ability to traverse firewalls and proxies. A WebSocket detects the presence of a proxy server and automatically sets up a tunnel to pass through the proxy. The following figure shows a basic WebSocket-based architecture in which browsers use a WebSocket connection for full-duplex, direct communication with remote hosts.

C. SIP in a nutshell

The IETF came up with a standard called SIP. As the name suggests; the Session Initiation Protocol (SIP) deals with initiation of interactive communication between users in form of sessions. Primarily, SIP is a standard for establishing, managing, and terminating Internet sessions, including voice and video calls and conferences. SIP itself doesn't defines what a "session" is; it uses other standards such as Session Description Protocol (SDP) for offer/answer of session negotiation, Real-Time Transport Protocol (RTP) for media path transport, Real-Time Transport Control Protocol (RTCP) for feedback and control of the media path, optionally Secure RTP (SRTP) with keys negotiated in SDP for media path security, and optionally interactive connectivity establishment (ICE) for traversal through intermediate network address translators (NATs) and firewalls. The request-response pattern forms the basis of SIP. The caller known as the User Agent Client (UAC) initiates a session by sending a request known as in INVITE addressed to the receiver of interest. SIP uses URLs for addresses and the URL format is very similar to the popular mailto URL. In many ways, the SIP is patterned after HTTP. SIP is request-response just like HTTP. Thus SIP can be well integrated with other web applications as well.

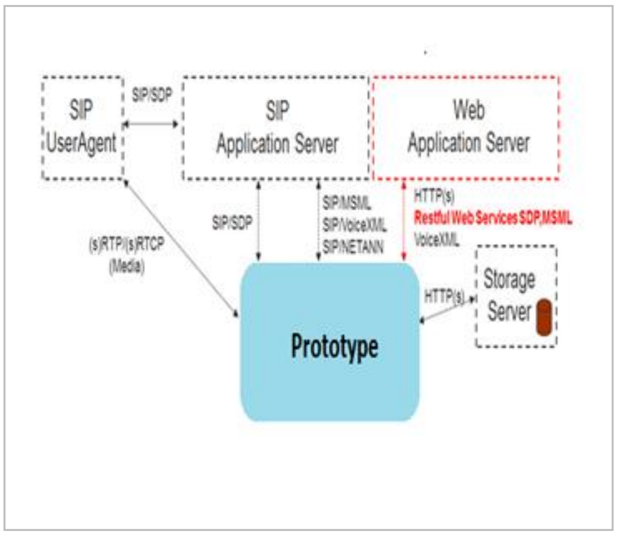


Figure (a): High level system architecture

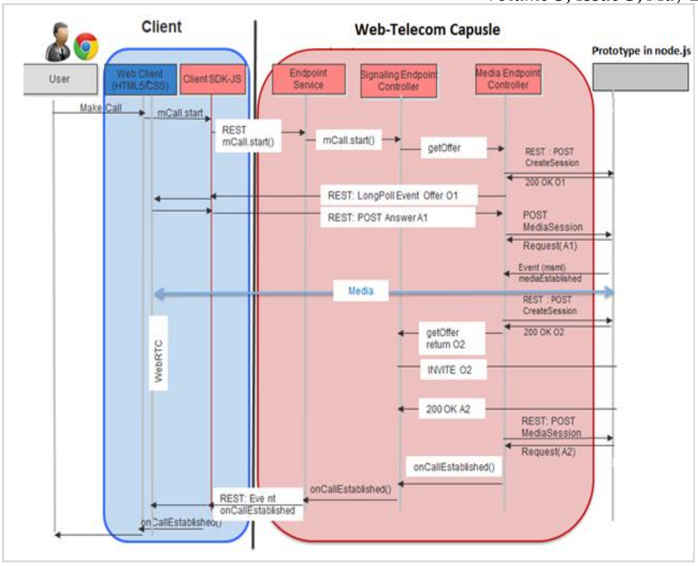


Figure (b): Outgoing Call flow Test (depicts the Web-telecom capsule)

III. System Architecture

Figures (a) & (b) shows us the overall system architecture along with the components of the Web-Telecom capsule. Figure (a) depicts the high level system architecture and how the system is actually modeled. The components SIP Application Server and Web Application Server are actually blocks of the Web-Telecom Capsule. Figure (a) shows how the SIP and Web components are connected to each other as well the prototype developed by us. It also gives us a glimpse of few other components of the system which do not lie in the scope of the paper and hence not discussed here. The Web-Telecom Capsule as a whole block can be seen in figure (b) and it is clear from this figure how the components, together form a capsule-like block and hence the name Web-Telecom capsule is justified. Figure (b) also shows the call flow for an outgoing call test. It begins with the user dialing call from a device/ browser. This gets translated as a ReST API call which is handled by the Web application component of the Capsule. The request is then forwarded to the SIP component of the Capsule.

In order to initiate a SIP session, negotiation w.r.t. the media capabilities and other factors is handled by the media endpoint component of the Capsule and the prototype developed in node.js. After successful negotiation, the SIP session is initiated which is conveyed back to the user via the ReST component from the SIP. Thus one leg of the call is successfully established. Similarly, the other end of the call is established on the same lines.

SYSTEM ORGANIZATION

The system providing server based telephony and media service is organized as follows and consists of components as follows:

i. The Web-Telecom Capsule

ii. The prototype – developed in node.js

A. The Web -Telecom Capsule

The Web – Telecom Capsule provides Server based Telephony and media service. It encapsulates the following components:

i) The SIP application server / SIP endpoint adapter;

ii) The Web application server;

iii) The media end point adapter;

B. The SIP application Server

The SIP application server provides the business logic to implement the telephony features and handles the SIP signaling. It interconnects all the different interfaces. Thus it will receive commands from the ReST API, it will exchange and process information from the media end point adapter and perform the required operations on the call.

C. The Web application server

The Web application server or service gateway provides the business-logic to implement REST API layer. The API exposed by this application is made up of two components.

i) The ReST API

ii) The Notifications API for notifications over WebSocket or Server-Sent Events (SSE).

D. The Media Endpoint Adapter

The Media Endpoint Adapter is responsible for handling the media transcoding between webrtc and other codes, and also communicating with the web browser (WebRTC media engine). The media endpoint adapter talks to the SIP end point and also to our prototype.

The in-depth implementation details of the above mentioned are beyond the scope of this document and hence not included here.

E. The prototype

The prototype is designed and implemented to simplify media application prototyping, development and testing. It has been designed to align with the WebRTC architecture, protocols and interfaces. It has sparked innovation by lowering the "barrier to entry" for developing media applications. It is implemented using node.js and it publishes a Restful control interface.

A. The prototype provides support for:

B. Registering web application control contexts

C. Creating new sessions

D. Refreshing sessions

E. Negotiating media via SDP offer/answer

F. Injecting MSML commands

G. Retrieving event streams

H. Retrieving media server reports and load factor

I. Destroying sessions

J. Destroying control contexts. Web applications are partitioned by utilizing a unique control context for session creation.

K. Both SIP and Rest interfaces can be used simultaneously by multiple applications.

IV. PERFORMANCE EVALUATION

The following section shows the performance results and analysis of the system. We first discuss the test bed and methodology and then move forward to the performance results captured and its analysis.

Test setup

A. JMeter load generator

In order to simulate a configurable number of clients asking for a call setup, we have used JMeter. JMeter not only aids in simulation of the clients but also helps in analyzing and measuring the performance of the webservices.

B. SIPp load generator

For SIP traffic generator, we use the open-source SIPp tool, which allows a wide range of SIP scenarios to be tested.

C. Hardware and Connectivity

The test bed environment includes a machine hosting the Call service implementation which is the ReST API component of the capsule and also SIP application server, a machine hosting the SIPp tool and also the prototype implemented in node.js. The machines are Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz, RAM 8GB.

A set of test iterations were performed to evaluate the performance of the Web – Telecom capsule. We evaluated the performance of the components of the Capsule, namely, the Web application server i.e. the ReST webservices API and the WebSocket API, the signaling component i.e. the SIP application server. The tools used to carry out the performance testing are JMeter for the Web application server and SIPp for the SIP application server.
We measured performance of the system in terms of number of calls per second the system can handle. We tested the system by making 10 calls per second. The performance of the ReST component was measured in terms of call setup delay. We tested WebSockets performance in terms of subscribe and notification delay. The SIP component's performance is measured on the number of call setup delay for ReST-SIP calls.
The following table shows the scenario we used for testing.

Test Scenario	**Test configuration**
Tested with 10 calls per second	1) JMeter: 700 users-Making 5 calls each 2) SIPP

Test scenario

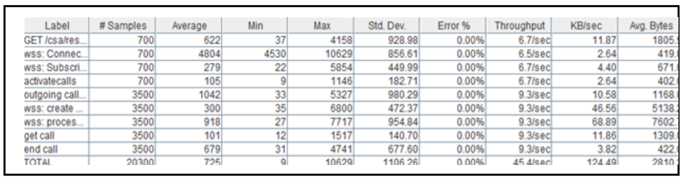
The following table shows the time delay measurements we discovered after testing the above scenario.

Call setup delay ReST	**Call setup delay REST-SIP**	**Subscription & Notification delay**
37 ms	31 ms	4 s

Time delay measurements

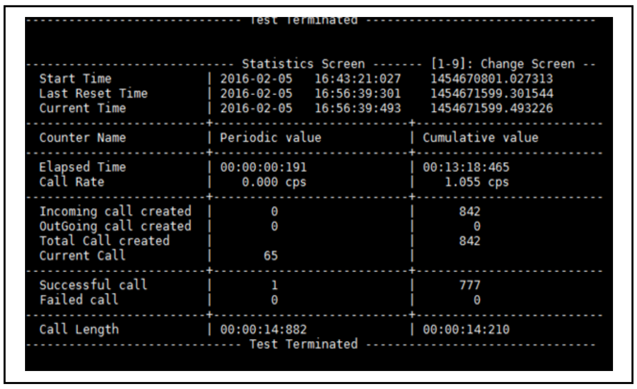
Below figures show the results captured: c) JMeter report; d) SIPp report; e) JConsole overview; f) CPU usage

Fig (c) JMeter summary report



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
GET /csa/res...	700	622	37	4158	928.98	0.00%	6.7/sec	11.87	1805.
wss: Connec...	700	4804	4530	10629	856.61	0.00%	6.5/sec	2.64	419.
wss: Subscri...	700	279	22	5854	449.99	0.00%	6.7/sec	4.40	671.
activatecalls	700	105	9	1146	182.71	0.00%	6.7/sec	2.64	402.
outgoing call...	3500	1042	33	5327	980.29	0.00%	9.3/sec	10.58	1168.
wss: create ...	3500	300	35	6800	472.37	0.00%	9.3/sec	46.56	5138.
wss: proces...	3500	918	27	7717	954.84	0.00%	9.3/sec	68.89	7602.
get call	3500	101	12	1517	140.70	0.00%	9.3/sec	11.86	1309.
end call	3500	679	31	4741	677.60	0.00%	9.3/sec	3.82	422.
TOTAL	20300	725	9	10629	1106.26	0.00%	45.4/sec	124.49	2810.

Fig (d) SIPp summary report



```
------------------------------ Test Terminated --------------------------------


----------------------------- Statistics Screen ------- [1-9]: Change Screen --
  Start Time             | 2016-02-05   16:43:21:027    1454670801.027313
  Last Reset Time        | 2016-02-05   16:56:39:301    1454671599.301544
  Current Time           | 2016-02-05   16:56:39:493    1454671599.493226
-------------------------+---------------------------+--------------------------
  Counter Name           | Periodic value            | Cumulative value
-------------------------+---------------------------+--------------------------
  Elapsed Time           | 00:00:00:191              | 00:13:18:465
  Call Rate              |    0.000 cps              |    1.055 cps
-------------------------+---------------------------+--------------------------
  Incoming call created  |        0                  |      842
  OutGoing call created  |        0                  |        0
  Total Call created     |                           |      842
  Current Call           |       65                  |
-------------------------+---------------------------+--------------------------
  Successful call        |        1                  |      777
  Failed call            |        0                  |        0
-------------------------+---------------------------+--------------------------
  Call Length            | 00:00:14:882              | 00:00:14:210
------------------------------ Test Terminated --------------------------------
```

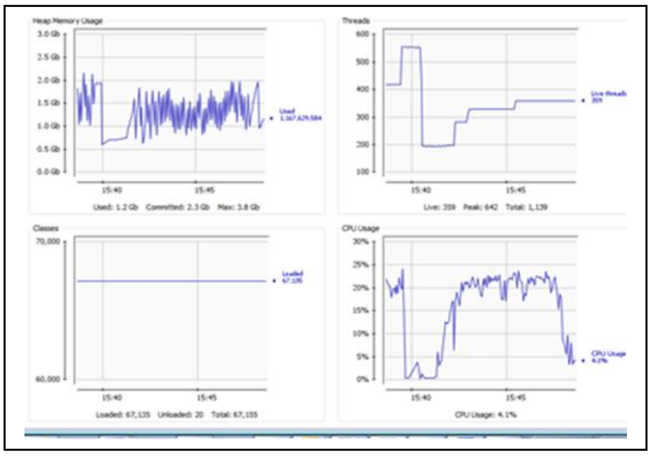

Heap Memory Usage
Threads
Classes
CPU Usage
Used: 1.2 Gb Committed: 2.3 Gb Max: 3.8 Gb
Live: 359 Peak: 642 Total: 1,139
Loaded: 67,135 Unloaded: 20 Total: 67,155
CPU Usage: 4.1%
15:40
15:45

Fig (e) JConsole overview

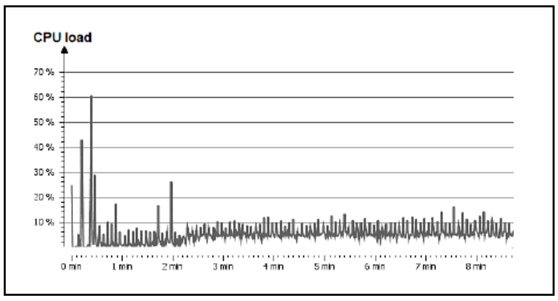


Fig (e) CPU usage graph

We used the software JConsole to analyze the Java Virtual Machine (JVM), hosting the Apache Tomcat servlet container. We used JConsole to capture the number of running threads and the CPU load. We are interested mainly to observe the behavior of the CPU. The CPU load is depicted. We can note that the CPU is utilized only in some cases corresponding to the creation of the presence resource occurring in the initial phase of tests. The next spikes are due to the processing of POST and PUT requests occupying, at approximately 5 seconds regular intervals, 10% or slightly more of CPU, with minimum CPU usage that never drops below 5%.

These primitive tests show how the resource consumption is linearly comparable with the number of users and calls to setup. In-depth performance analysis will be carried out in the near future by evaluating delays and resource consumption in more complex workload scenarios.

V. CONCLUSION

In this paper, we introduced the Web-Telecom Capsule, a block comprising of heterogeneous technologies and how it leveraged the development of web-telecom services in the concerned telecom ecosystem. We also implemented a node.js based prototype which eased the testing of the Web-Telecom Capsule. Finally we evaluated the performance of the components of the Capsule and the system as a whole in terms of CPU usage and time delays. Future work includes test activities for performance evaluation and analysis under different complex workloads and different use cases.

Acknowledgement

Authors acknowledge the support of Avaya India Pvt. Ltd., Mr. Suvarna Madhey and Mr. Varun Khanna for providing valuable inputs and suggestions for execution of this project.

REFERENCES

[1] R. Fielding, "REST: Architectural Styles and the Design of Networkbased Software Architectures", Doctoral dissertation, University of California, Irvine, 2000.

[2] C. Fu, F. Belqasmi, and R. Glitho, "RESTful web services for bridging presence service across technologies and domains: an early feasibility prototype," IEEE Commun. Mag., vol. 48, pp. 92– 100, December 2010.

[3] J. Rosenberg et al. (June 2002) RFC3261—SIP: session initiation protocol. http://www.ietf.org/rfc/rfc3261.txt.

[4] W3C, WebRTC 1.0: Real-time Communication Between Browsers, W3C Working Draft 21 August 2012.Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[5] IETF, Overview: Real Time Protocols for Brower-based Applications, Internet-Draft, December 14, 2012.

[6] D. Lozano, L.A. Galindo, and L. Garci, "WIMS 2.0: Converging IMS and Web 2.0. Designing REST APIs for the Exposure of Session-Based IMS Capabilities", in Proc. of Second Int. Conf. on Next Generation Mobile Applications, Services, and Technologies (NGMAST '08). IEEE Computer Society, Washington, DC, USA, 18-24, 2008.

[7] Davids, A. Johnston, K. Singh, H. Sinnreich, and W. Wimmreuter,"SIP APIs for voice and video communications on the web", In Proc. of the 5th Int. Conf. on Principles, Systems and Applications of IP Telecommunications (IPTcomm '11). ACM, NY, USA.

[8] IETF, The WebSocket protocol, Internet Draft, 6 May 2010, Ian Hickson,http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol

Dnyaneshwari Nirwan is currently pursuing her Masters in Technology from College of Engineering, Pune. She is working as an intern at Avaya, India. She has a work experience of 3 years prior to her Masters

.

Dr. Shashikant Ghumbre is faculty at the Dept. of Computer and IT Engineering at College of Engineering, Pune and has been teaching since 13 years. He holds a Ph.D in Computer Engineering and has authored many international papers.