Determining the Signalling Overhead of two common WebRTC methods:JSON via XMLHttpRequest and SIP over WebSocket

Michael Adeyeye[1], Member, IEEE, Ishmeal Makitla[2], and Thomas Fogwill[2], Member, IEEE

Abstract—Web Real-Time Communication (WebRTC) introduces real-time multimedia communication as native capabilities of Web browsers. With the adoption of WebRTC the Web browsers will be able to use WebRTC to communicate with one another (peer-to-peer), and with WebSocket servers such as Mobicents SIP Servlets and other server technologies that support WebSocket communication to enable SIP-to-WebRTC communication. This paper outlines the potential of WebRTC and discusses the two common methods of doing real-time communication in Web browsers through WebRTC. The methods are JavaScript Object Notation (JSON) via XMLHttpRequest (XHR) and Session Initiation Protocol (SIP) via WebSocket. A three-user WebRTC video chat prototype application was developed and used to evaluate both methods. Additional signalling overhead introduced into a browser by each method was determined. The results showed WebRTC-SIP/WS has more overhead than WebRTC-JSON/XHR. These signalling overhead findings are useful in that they could help application developers make decision on their choice of technologies and protocols when developing WebRTC-supported applications.

Index Terms—WebRTC, IMS, SIP, Browser communication

I. INTRODUCTION

The Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) are currently tasked with bringing WebRTC among browsers to an acceptable level in both the industry and the academia. WebRTC is an open framework that offers web application developers the ability to write rich real-time multimedia applications (e.g. video and gaming applications) on the web without requiring plugins or extensions. Its purpose is to help build a strong Real Time Communication (RTC) platform that works across multiple web browsers and platforms. In an implementation, the WebRTC API will abstract several key components for real-time audio, video, networking and signal [1], [2]. While the IETF is standardizing the signalling protocols and media technologies (e.g. codecs) required in WebRTC, the W3C is standardizing the APIs and browsers for real time communication.

There are many implementations of RTC among web browsers [3], [4], [5], [6], [7] and the WebRTC itself [8], [9], [10]. The standard signalling protocol for WebRTC is JavaScript Session Establishment Protocol (JSEP), however, Remote Object Access and Replication (ROAR) has also been used in some existing implementations.

[1]Department of Information Technology, Cape Peninsula University of Technology, South Africa, e-mail: adeyeyem@cput.ac.za
[2]The Next Generation Network and Architecture Research Group, CSIR, South Africa, e-mail: imakitla, tfogwill@csir.co.za.

The reason JSEP is the preferred protocol is because it moves control or negotiations from a browser to JavaScript (in an application). In addition, there is a need to make the WebRTC implementation look similar to the SIP Offer and Answer. Although the VP8 codec seemed to be the preferred codec for WebRTC, it is faced with royalty problems. Hence, codecs for the WebRTC are also being addressed.

WebRTC is built on the PeerConnection API and represents what browser vendors will implement and expose to web application developers. Web application developers can choose an underlying protocol depending on their project requirements. The underlying protocols, also called the sub-protocols, include SIP and XMPP (with Jingle). The Libjingle library, like a SIP stack, supports (Session Transversal Utilities for NAT (STUN) and Transversal Using Relays and NAT (TURN). Both these Interactive Connectivity Establishment (ICE) techniques, namely STUN and TURN, make communication possible when the communicating endpoints are behind a firewall.

The motivation for this work is that application developers will soon begin to create innovative WebRTC-supported applications with little or no consideration on the total cost of usage of their applications. An application with a high signalling overhead would incur more cost with poorer quality of experience for users having low Internet bandwidth and paying high cost for Internet access. This work examined the additional signalling overhead introduced by WebRTC applications. The contribution of this research is therefore the development of a three-user WebRTC video chat application with a report on the signalling overheads introduced by the two common methods of doing WebRTC.

The remainder of this paper is arranged as follows: Section II briefly presents the potential of WebRTC; Section III discusses the common methods of doing WebRTC within compliant Web browsers and the current ways of implementing video streaming using WebSocket. Section IV presents the three-user WebRTC vide chat prototype which was used to evaluate the resultant signalling overhead. Section V then presents and discusses the resultant signalling overhead of the two common methods of doing WebRTC. Section VI hints at the related future work which proposes a protocol-agnostic WebRTC Back-to-Back User Agent (B2BUA) extending the work reported in this paper. Section VII contains the conclusion.

II. WEBRTC POTENTIALS

It will be important for the IP Multimedia Subsystem/Rich Communication Suite (IMS/RCS) world to interoperate with the WebRTC world so that voice communication can become more affordable. In addition, the interoperability would make the IMS/RCS become pervasive. Currently, browsers are in a walled-garden and are not connected to the IMS/RCS. There are huge potentials for their interoperability. The Over The Top (OTT) applications will be able to use directory services available in the social applications, such as Skype, Gtalk and WhatsApp. Directory services would become critical sources of value in connecting different identities, such as telephone numbers, SIP IDs (IDentifiers), web session IDs and other OTT IDs [11]. In addition, PSTN would become the communication path of last resort, when the other party is not data connected or the call quality is too low due to the Internet connection. A company's website can now become its call centre front end, and a weblog can become a personal communications assistant for its owner. Communication Service Aggregators (CSA) would help customers that run multiple clients on their phones save cost by running their services in cloud and offering them control via a browser. As earlier envisaged in [7], a click-to-call would not require an operator's voice network in order to setup a voice call,. VAS (Value Added Services) would be extended to web developers that can create value and solve problems for customers. Gaming, Advertisement and Customer Relationship Management (CRM) will take new communications space with innovative business models.

Section III discusses the two common methods of doing real-time communication in Web browsers through WebRTC.

III. WEB REAL-TIME COMMUNICATION (WEBRTC) METHODS AND ISSUES

The two prominent ways of doing WebRTC are using pure SIP via WebSocket (WebRTC-SIP/WS) and JavaScript Object Notation via XMLHttpRequest (WebRTC-JSON/XHR). While the former uses a WebRTC-SIP proxy/gateway as its application engine and SIP over WebSocket for signalling, the latter uses a custom engine (e.g. the Google App. Engine) as its application engine and JSON over XHR for signalling. For the Google App. Engine, the JSON/XHR signalling is done via its Channel API. However, both approaches are based on JSEP, which mimics the SIP Offer and Answer signalling.

There are however other implementations developed to meet specific requirements. An example is the Ericsson WebRTC implementation [8], which uses ROAR. In this example, some changes were made to the webkit libraries in the Epiphany web browser in order to support WebRTC. There are other kinds of implementation in the form of an extension to a browser. An example is the IEWebRTC extension (which uses Chrome-Frame) for Internet Explorer [12]. As web browsers are being extended, the number of WebRTC applications and frame-works, such as SIPML5 (which uses SIP over WebSocket) [13] and SIP-JS (with support for Flash-network) [5], are rapidly increasing. At the time of this research, Google Chrome is taking the lead in the WebRTC implementation. Mozilla Firefox is yet to have a version that has the PeerConnection or getUserMedia API. A SIP stack (called SIPCC) is now being integrated into it [14]. Hence, it does not currently support WebRTC. Other browser makers, such as Microsoft and Opera, are also contributing to the WebRTC standardization.

Fig. 1 shows the signalling between two UAs (User Agents) or devices; the sequence of events starts from top to bottom. Some of the processes (such as PeerConnectionFactory, ProcessSignalingMessage and OnSignalingMessage) are peculiar to Google Chrome, which uses the libjingle. A caller first creates a new peerconnection and adds stream using the PeerConnection API as shown in Figure 1. In addition, a local session description (for audio and/or video) is applied. ICE is then started in order to get available IP (Internet Protocol) address and port number for media transfer (these additional processes are not shown for simplicity). A peerconnection and remote session description (for the callee) are later created. When a callback at the callee's notifies that a stream is added (via a channel), an offer is created. It is sent and processed by the caller. An answer is then sent back and a local session description (for the callee) is created. The answer, which contains the remote session description and some hints, is sent to the callee. A local session description (for audio or/and video) is then set and applied in the callee's browser. Lastly, ICE is also started in order to get available IP (Internet Protocol) address and port number for media transfer. The caller later applies the remote session description in order to present video/audio from the callee. Encryption of WebRTC media in Google AppEng is achieved by sending the UDP (User Datagram Protocol) data via SCTP (Stream Control Transmission Protocol) and DTLS (Datagram Transport Layer Security).

On the other hand, there are still NAT and firewall issues in the WebRTC. In addition, SBCs (Session Border Controllers) could be required to handle connections between two or more domains. ICE techniques (STUN and TURN) are likely to increase duration to set-up a call. Security of media and permissions are also hot issues in the WebRTC, though there are a couple of solutions that can be used. Other issues of interest include recording video, supporting other SIP/SIMPLE features, such as presence and messaging, and doing multi-user video chat using the WebRTC framework.

IV. A THREE-USER WEBRTC VIDEO CONFERENCE PROTOTYPE

To determine the additional signalling overhead introduced by WebRTC applications, a three-user WebRTC video chat application was developed as a prototype for this research using the PeerConnection API. The prototype application used both WebRTC communication methods discussed in Section III namely WebRTC-SIP/WS and WebRTC-JSON/XHR. Google Chrome Web browser which integrated libjingle (with XMPP) was used for experimentation as it is the only browser with an acceptable level of WebRTC support required for this research. As at the time of writing, this is the only work that has considered signalling overhead introduced by the two WebRTC methods, In addition, the three-user WebRTC video chat application is one of the few WebRTC video chat applications that support three or more users.

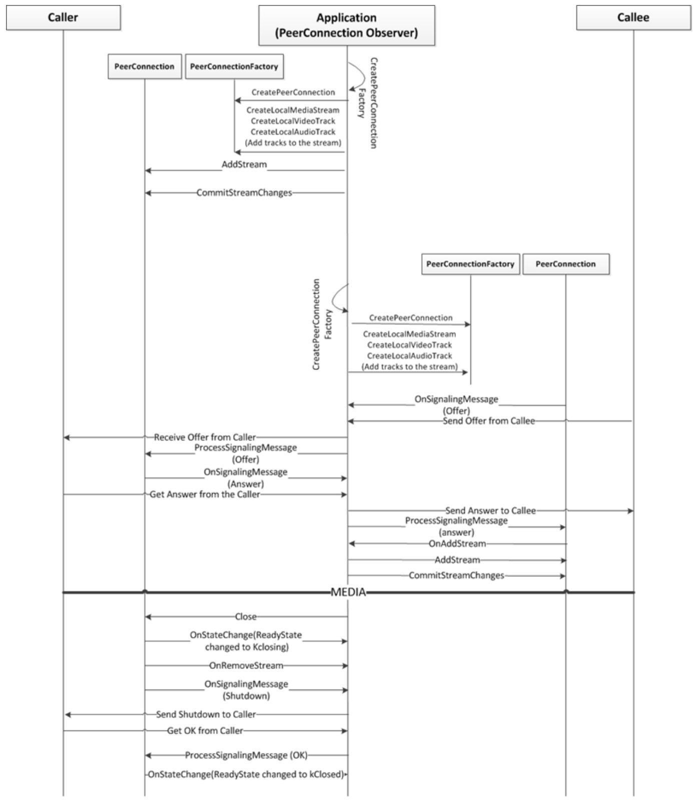
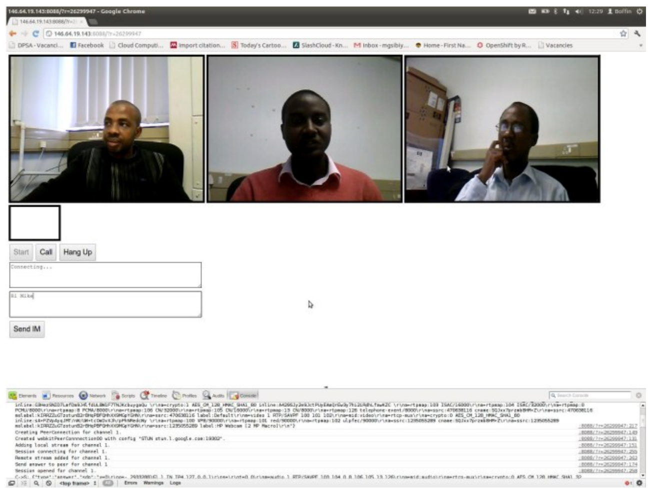

Caller
Application (PeerConnection Observer)
Callee
PeerConnection
PeerConnectionFactory
CreatePeerConnection Factory
CreatePeerConnection
CreateLocalMediaStream
CreateLocalVideoTrack
CreateLocalAudioTrack
(Add tracks to the stream)
AddStream
CommitStreamChanges
PeerConnectionFactory
PeerConnection
CreatePeerConnection Factory
CreatePeerConnection
CreateLocalMediaStream
CreateLocalVideoTrack
CreateLocalAudioTrack
(Add tracks to the stream)
OnSignalingMessage (Offer)
Send Offer from Callee
Receive Offer from Caller
ProcessSignalingMessage (Offer)
OnSignalingMessage (Answer)
Get Answer from the Caller
Send Answer to Callee
ProcessSignalingMessage (answer)
OnAddStream
AddStream
CommitStreamChanges
MEDIA
Close
OnStateChange(ReadyState changed to Kclosing)
OnRemoveStream
OnSignalingMessage (Shutdown)
Send Shutdown to Caller
Get OK from Caller
ProcessSignalingMessage (OK)
OnStateChange(ReadyState changed to kClosed)

Fig.1. The WebRTC Signalling in a Call Session

Most WebRTC video chat applications are only for two users, since WebRTC is currently being standardized. The Three-user WebRTC video chat is depicted in Fig. 2.



```
[illegible]
Creating PeerConnection for channel 1.
Created webkitPeerConnnection00 with config "STUN stun.l.google.com:19302".
Adding local stream for channel 1.
Session connecting for channel 1.
Remote stream added for channel 1.
Send answer to peer for channel 1
Session opened for channel 1.
[illegible]
```

Fig. 2. The Three-user WebRTC demo

Although the application was built for no more than three users (as shown in Fig. 2), the signalling between two users is shown here (Fig. 1) for simplicity reasons. The application in-between the two User Agents (UAs) acts as a B2BUA and is common feature among multi-user conference applications. The video conference application was first developed and deployed in Google AppEngine (i.e. the WebRTC-JSON/XHR).

The three-user video conferencing application used the Channel API in the Google AppEngine for WebRTC signalling and the getUserMedia and PeerConnection APIs in the Google Chrome browser for media streaming. Since the PeerConnection API only works for two devices, each device created two instances of "we-bkitPeerConnnection00" and each instance was used to set-up a peer-to-peer connection with the other device. In order to demonstrate WebRTC-SIP/WS, a SIP servlet application was modified and deployed into the Mobicents AS (Application Server) as a SIP proxy in an IMS. The Mobicents SIP Servlets AS used Apache Tomcat 7.0, which supports WebSocket. The SIP proxy acted as a B2BUA, which sets up a video chat among the three users. The source of the application is published on the Internet for contributions from interested parties and the Open Source (OS) community [15]. It is one of the few WebRTC works on the Internet that support more than two users. Figure 2 also shows the signalling in a browser using the browser's developer tools.

V. WebRTC-JSON/XHR and WebRTC-SIP/WS Signalling Overhead

In order to report the performances and differences between WebRTC-JSON/XHR and WebRTC-SIP/WS method of doing WebRTC, an experiment was performed using each method. The signalling overhead in a peer-to-peer connection was measured. Three laptop computers with identical hardware and software specifications and each running a WebRTC-enabled Chrome Web browser were used. The upload and download speed for the network were 0.15Mbps and 0.81Mbps, respectively. The test was carried out on a Local Area Network (LAN), and the WebRTC-SIP/WS application played the role of both a WebRTC-SIP proxy/gateway and a SIP Registrar. Hence, there were no outbound connections. Like every application, its QoS (Quality of Service) depends on the network speed. Connection time (latency) and signalling overheads are two factors that can be used to evaluate the performance of the two WebRTC methods. The connection time and delay were determined by running Network Time Protocol (NTP) on all machines used in the experiment. While connection time among peers in a video chat was infinitesimal or not noticeable (being a test performed on a LAN), the signalling overhead was noticeable. As a result, this work focuses on the signalling overheads of each WebRTC method.

The payload of each application was not included in the values of signalling overheads. Table I shows the signalling overheads in a web browser when the browser runs the WebRTC prototype applications for the three-user video chat. In addition, the values were compared with overheads introduced by a regular SIP client - PJSIP. The result shows the signalling overheads as they increases in both WebRTC approaches. The experiment was repeated multiple times in order to report mean values and, for each value, its variance in brackets for the overheads.

As reported on Table I, all results show a limited variance. A basic HTTP request-response (with no payload) is 150B. The HTTP overhead is higher than the WebRTC-JSON/XHR overhead for a completed session (104B) because the HTTP server (Apache) responded with some additional information in its response header. It is however possible for a developer to compress HTTP response headers or reduce the response information to the essential ones. The WebRTC-SIP/WS overhead can affect quality of experience, where access to the Internet is costly and the Internet connection speed is low.

Table I
SIGNALLING OVERHEAD IN WEBRTC VIA JSON/XHR AND SIP/WS

WebRTC Events	WebRTC-JSON/XHR	WebRTC-SIP/WS	A SIP Client (PJSIP)
On Register	13B (0.58)	34B (1)	2.5kB (1.1)
On Invite	39B (0.58)	204KB (0.88)	4.9kB (1.02)
During Call Session	78B (0.78)	240KB (1)	9.6kB (1)
Ending Call Session	104B (0.78)	275KB (1)	10.4kB (1.02)

VI. RELATED FUTURE WORK : PROTOCOL-AGNOSTIC WEBRTC-B2BUA

Although this paper reports the signalling overhead of both WebRTC-JSON/XHR and WebRTC-SIP/WS methods, future efforts will aim at developing a protocol-agnostic WebRTC B2BUA. The aim is to allow different user agents (which can be Web browsers or normal mobile or desktop communication applications) to interoperate with though they may each be using a different sub-protocol (e.g SIP, XMPP or H.323). The work will use the Mobicents Java API for Integrated Network Service Logic Execution Environment (JAIN SLEE) as its communication platform.

Fig. 3 depicts the anticipated solution architecture based on Mobicents JAIN SLEE, and using the JavaScript Session Establishment Protocol (JSEP) as a unifying protocol for all underlying user-agent specific sub-protocols such as SIP, XMPP or H.323. As of this writing, the only WebRTC work involving the use of Mobicents application server uses SIP Servlet and therefore assumes an all-SIP communication.

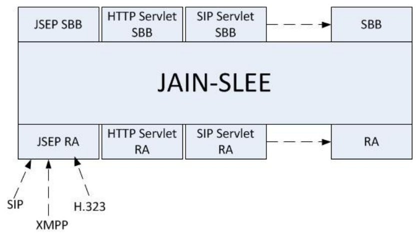


Fig. 3. The JAIN-SLEE WebRTC Implementation Plan

The proposed protocol-agnostic B2BUA is based on JSEP and any signalling protocol of choice (e.g. SIP, H.323 and XMPP) can be plugged to it. Because of WebRTC's flexibility regarding the sub-protocols used by clients over WebSocket, the application server needs to have multi-protocol support. Mobicents JAIN SLEE is chosen because it meets this requirement through its Resource Adaptor (RA) framework. Efforts are currently underway to develop the JSEP RA required to realize the proposed protocol-agnostic WebRTC B2BUA.

VII. CONCLUSIONS

A three-user WebRTC video chat has been developed and released to the OS community and researchers exploring WebRTC. In addition, the signalling overheads for the two WebRTC approaches have been reported and a protocol-agnostic user agent for WebRTC-enabled IMS is now proposed. The support for WebRTC would create an additional ways of communicating between two devices. Voice services in existing telecommunication networks may likely drop as customers will pay more for data services in order to use WebRTC. Like there are unique attributes in CSS for different browsers, developers may need use some browser-specific features, most notably in JS (JavaScript), after the standardization effort. They would have to choose what approach they want to use to develop their applications, and one of their considerations would be the signalling overhead. In order to support WebRTC in IMS, a WebRTC-to-SIP B2BUA that will make multiple implementations of WebRTC exist in it is proposed. Using JAIN-SLEE over SIP servlet, WebRTC-to-SIP B2BUA would seamlessly work with any underlying protocols and services offered in the IMS.

Although the number of VOIP & IM applications on a PC would likely reduce, a browser installer file size would drastically increase as it will now support new features, such as WebRTC and WebGL. While libjingle (with XMPP) is integrated into Google Chrome, a SIP stack is integrated in Mozilla Firefox to implement WebRTC. The integration of these protocols would open enormous opportunities for developers. On one hand, web developers can develop websites and applications that would run in a browser using HTML5 with the APIs exposed to webpages. On the other hand, application developers can develop applications that work with a browser internals (e.g. a XULRunner or Chrome Application) thereby directly communicating with the underlying protocols and mechanisms in that browser. Future work includes extending the SIP Servlet application to JAIN-SLEE and comparing latency in setting up a two-way communication between an IMS-based WebRTC and the regular (non IMS-based) WebRTC

REFERENCES

[1] WebRTC. {Online]. Available: http://www.webrtc.org

[2] IETF WebRTC. [Online]. Available: http://tools.ietf.org/wg/rtcweb

[3] David Linner, Horst Stein, Ulrich Staiger and Stephan Steglich, "Realtime Communication Enabler for Web 2.0 Applications," in: Proceedings of the Sixth International Conference on Networking and Services (ICNS '10), Cancun, Mexico, March 7-13, 2010, pp. 42 - 48.

[4] SIP on the Web. [Online]. Available: http://sip-on-the-web.aliax.net

[5] SIP-JS. [Online]. Available: http://code.google.com/p/sip-js

[6] The Phono WebRTC. [Online]. Available: http://phono.com/webrtc

[7] Michael Adeyeye, Neco Ventura and Luca Foschini, "Converged Multimedia Services in Emerging Web 2.0 Session Mobility Scenarios ," in: the Springer Wireless Networks (WINET) Journal. DOI: 10.1007/s11276-011-0394-z.

[8] Ericsson WebRTC. [Online]. Available: https://groups.google.com/group/ericsson-labs-web-rtc

[9] Chrome WebRTC Implementation. [Online]. Available: http://www.w3.org/2011/04/webrtc/wiki/images/7/7f/Webrtc-chromeimpl-status.pdf

[10] IETF RTCWeb-SIP WG. [Online]. Available:

http://tools.ietf.org/html/draft-kaplan-rtcweb-sip-interworkingrequirements-01

[11] The IMS World Forum Summary. [Online]. Available: http://www.alanquayle.com/blog/2012/04/the-ims-world-forum-summary-pa.html

[12] WebRTC for Microsoft Internet Explorer. [Online]. Avaialble: http://code.google.com/p/webrtc4ie/

[13] SIPML5. [Online]. Available: http://www.sipml5.org/
[14] Ethinhugg/Ikran. [Online] . Available: https://github.com/ethanhugg/ikran

[15] WebRTC_VideoChat. [Online]. Available: https://github.com/micadeyeye/webrtc_videochat

[16] Vijay K. Gurbani, Xian-He Sun and A. Brusilovsky, Inhibitors for Ubiquitous Deployment of Services in the Next-Generation Network, in: the IEEE Communications Magazine, Vol. 43, No. 9, pp. 116-121, September 2005.

[17] Karim Sbata, Houda Khrouf, Sabine Zander and Monique Becker, Converging Web and IMS Services: Stakes and Solution Proposals, in: Proceedings of the International ACM Conference on Management of Emergent Digital EcoSystems (MEDES '09), Lyon, France, October 27-30, 2009.

[18] Haruno Kataoka, Masashi Toyama, Yoshiko Sueda, Osamu Mizuno and Kenji Takahashi, Demonstration of Web Contents Collaborative System for Call Parties, in: Proceedings of the 7th IEEE Consumer Communications and Networking Conference (IEEE CCNC '10), Las Vegas, Nevada, USA, January 9-12, 2010.

[19] .http://www.google.com/chromeframe?quickenable=true

[20] WebRTC-Sampels. [Online]. Available: http://code.google.com/p/webrtc-samples/

[21] WebRTC Reference App. [Online]. Available: https://apprtc.appspot.com

[22] Hideo Nishimura, Hiroyuki Ohnishi and Miki Hirano, "Architecture for Web-IMS Co-operative Services for Web Terminals ," in: Proceedings of the 13th International Conference on Intelligence in Next Generation Networks (ICIN '09), Bordeaux, France, October 26 - 29, 2009, pp 1-6.

[23] The PJSIP Project. [Online]. Available: http://www.pjsip.org/

[24] The Mozilla Firefox Web browser. [Online]. Available: http://www.mozilla.org

[25] M. Handley and V. Jacobson, SDP: Session Description Protocol, IETF RFC 2327, April 12, 2012.