Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 10, 2012

H. Tschofenig
Nokia Siemens Networks
B. Aboba
Microsoft Corporation
J. Peterson
NeuStar, Inc.
D. McPherson
Verisign
May 9, 2012

Trends in Web Applications and the Implications on Standardization
draft-tschofenig-post-standardization-02.txt

Abstract

Advancements in the design of web browsers have introduced fundamental changes to the architecture of application protocols. The widespread availability and growing sophistication of JavaScript interpreters in browsers enables web servers to push to browsers all of the application logic required to implement a client-server protocol. Consequently, many client-server applications that once required an installed client on a host computer now can rely simply on a modern browser to act as a client for the purposes of a particular application. For example, where once email clients required a custom application to access an inbox, increasingly a web browser can serve this purpose as well as the purpose-built applications of the past. Similarly, HTTP with the assistance of JavaScript can subsume the functions performed by the protocols like POP3 and IMAP. The need for Internet standards beyond HTTP to implement an email inbox application consequently diminishes - why author standards and worry about interoperability of clients and servers when the server can simply push to the client all the code it needs to be interoperable?

Many client-server applications on the Internet could potential migrate to this code distribution methodology.

[Note: A separate mailing list has been created for discussions related to this document and it can be found here: https://www.ietf.org/mailman/listinfo/webapps]

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The generic nature of the personal computer has enabled application providers to write general purpose programs and to make it available for download. This flexibility has lead to lots of innovation on the Internet but has also introduced security challenges since it is difficult for end users to judge the trustworthiness of downloaded programs in any reasonable way. Consequently, many users are very suspicious about any download they are asked to accept. An important goal of those deploying applications is to reach a widespread deployment as fast as possible and to react to changing needs as quickly as possible, which to a large extent requires the ability to continously update code on end devices. With operating system updates happening less frequently and the acceptance for software downloads decreasing the browser was seen by many as an ideal platform for dynamically downloaded running code. JavaScript was initially perceived as being quite limited in functionality but has been supported by all browsers. This perception has changed over the last couple of years when it became the scripting language implemented in the majority of browsers, also referred as the 'assembly language of the Internet'.

For application developers writing code running on Web servers as well as for applications that are downloaded to the end device the desire was always to develop the application once without having to consider all the different runtimes (operating systems or browsers). Now, with the PC and the cellular phone segments getting increasingly blurry this desire is stronger than ever considering the increased number of obstacles that have to be dealt with. For example, it is highly unlikely that an application will work on various different devices even if all the devices were produced by a single mobile phone vendor. Getting users to download new applications, and to install software updates also leaves software developers in a difficult situation.

How can software be developed so that it can (1) be updated instantly when a new version becomes available, (2) be used across a wide range of devices, and (3) be as powerful as regular desktop applications? This sounds almost impossible but with the increased capabilities of Web browsers, and JavaScript in particular, it seems that the Internet community has gotten a couple of steps closer to achieve this goal.

This document describes these developments, highlights impacts for the standardization community, and provides recommendations for those developing applications.

Note that the writeup heavily refers to JavaScript as a mechanism for

mobile code distribution. There is, however, nothing special about JavaScript as a language by itself and it may well be possible that other languages will be developed for usage in other environments offering similar or even superior capabilities.

2. Impact for the Standardization Community

In the application area communication protocols often follow the pattern where an end host utilizes some application service provider for communication setup and sometimes also for message routing towards the other communication end point. Examples of such a standardized communication protocols are the Post Office Protocol (POP) [1], the Internet Message Access Protocol (IMAP) [2], as well as the Session Initiation Protocol (SIP) [3] and the Extensible Messaging and Presence Protocol (XMPP) [4].

Figure 1 shows a typical scenario where two hosts, Alice and Bob, interact with an application provider. A desired interoperability goal often has been to let a software vendor develop the software clients at the end hosts to interact with a random application provider offering the specific protocol implementation.

```
             .................
             |               |
             |  Application  |
             |  Service      |
             |  Provice      |
             |  Example.com  |
             |_______________|
                 _,    .
               ,'       `.
            _,'           `.
          ,'                `._
        -'                     -
     ,''''''''|           ,''''''''|
     |  End   |           |  End   |
     |  Host  |           |  Host  |
     |  Alice |           |  Bob   |
     |........'           |........'
```

Figure 1: Communication Partners from a Single Domain

Many protocols developed in the IETF also offer the ability to let users from different application service providers (via their end hosts) to communicate. Figure 2 shows this architecture graphically, where additional interoperability needs are created between the application service provider domains.

```
              .................              .................
              |               |              |               |
              |  Application  |              |  Application  |
              |  Service      |--------------|  Service      |
              |  Provider     |              |  Provider     |
              |  Example.com  |              |  Example.org  |
              |_______________|              |_______________|
                  _,                                .
                ,'                                   `.
             _,'                                       `.
           ,'                                            `._
         -'                                                 -
      ,''''''''|                                        ,''''''''|
      |  End   |                                        |  End   |
      |  Host  |                                        |  Host  |
      |  Alice |                                        |  Bob   |
      |........'                                        |........'
```

Figure 2: Communication Partners from Multiple Domains

These two figures did not make the attempt to differentiate signaling message exchanges from the actual data traffic exchange. The data traffic may be exchanged directly between the end hosts themselves and therefore creates additional interoperability requirements when those software clients shall be developed by independent parties.

While many standardization efforts in the IETF have considered the possibility for using proprietary protocols along the end host to application service provider leg, this has usually been considered as exception or a transition case. It is typically assumed that the desired end state of standardization is to move from a proprietary protocol to the standardized alternative in the long run, which allows client software vendors to interact with all forms of application service providers. Such an approach increases the need for standardization and requires far more interoperable network elements to exist.

With a mobile code distribution platform as the Web with JavaScript offers it is possible to leave the end host to application service provider interaction largely non-standardized. Only very few standardization actions are required, to for example, enhance the capability of JavaScript to perform additional functions, such as the access to underlying hardware functions (e.g., microphone, GPS module or a camera).

Quite clearly applications can be designed in a way that fewer standardized client-server protocols are needed. The question therefore remains for those actively pursuing standardization as to

where the limitations of the JavaScript-based mobile code distribution approach is. Section 3 tries to explore this aspect in more detail.

3. Limitations of Mobile Code Distribution

The usage of JavaScript is, however, not always the right choice for application developers and even though a number of new building blocks are being made available, such as HTML5 [5] and various JavaScript extensions, there are still a number of limitations in today's browser environment. We list a couple of those challenges, some of which will be resolved in the near future as standardization and deployment progresses, while others will remain a challenge for a long time.

3.1. Performance Limitations

Early JavaScript implementations did not offer high performance. Over many years very little attention was paid to boost the performance until recently when the Google JavaScript engine V8 [6] started to compile JavaScript code directly into machine code when it is first executed. More details about the design can be found at [7].

A more serious limitation is the graphics capabilities in browsers. Efforts are under way to enhance the API capabilities, for example WebGL [8] bringing 3D graphics to the browser with features similar to OpenGL ES 2.0 that can be used in HTML5 canvas elements but expensive computations on the end host need to migrate from the Central Processing Unit (CPU) to the Graphics Processing Unit (GPU) for proper performance. Simple 3D games (similar to the recently demonstrated Quake II port to HTML5 [9] utilizing JavaScript, the WebSocket API [10] and the Web Storage API [11]) can now be implemented but state-of-the-art games and virtual worlds are out of reach. The problem is with the number of polygons that many games and virtual worlds need to process and display. Games, like Quake, use a limited number of textures, and the complexity of the scene graph is small.

In comparison to virtual worlds where the content is put together by users, in many games the playing field is carefully designed by experts. This has implications for the complexity of the scene graph. On the other hand, most virtual worlds do not rely on rapid communication updates in the same way that many action and tactic games do. Joshua Bell illustrated this with an example of 'a quiet scene with a single user running around in SecondLife [12]. A teleport to a region can easily have a scene graph with 2000 nodes, a couple hundred 3D textures, 4000 vertexes, and 20 MByte of vertex data. This corresponds to the maximum a graphics developer would typically like to have in a state-of-the-art game. In a busy scene with lot of user generated content and avatars the volume easily jumps up by a factor of five.' [13]. The size of the game itself

(often due to the high quality textures) and software updates is impressive; often reaching beyond several 100 Mbytes. Utilizing persistent storage and caching in combination with more aggressive client-server interactions demands a different style of programming and therefore also puts different constraints on the protocol design. This might also stress the current Mbyte limits for Web storage.

Initial work to deal with more sophisticated graphics computation has started already, as described in the recently published article [14] about elevating JavaScript performance through offloading processing to the GPU. As stated in the announcement of the Jetpack 0.5 contest [15]: 'By giving webpages and add-ons easy access to the raw processing power available on most computers, the range of abilities that the web can have greatly increases.'.

3.2. Transport Protocol Limitations

In [16] Jonathan Rosenberg argued that the new waist of the Internet hourglass is UDP and TCP, rather than IP as in the initial design. Today, application protocol designers may, however, get the impression that tunneling inside HTTP or even HTTPS is required to get an application running in a large number of environments, especially to reach a customer base that is connected to the Internet through an enterprise network. Needless to say that more complex tunneling leads to more complexity, the data transport adds overhead and the initial environment sensing phase adds delays. This is certainly true for the VoIP context where the payload data is comparatively small to the overall header size (including the TCP/ HTTP headers). The work on Interactive Connectivity Establishment (ICE) [17] is relevant for the sensing phase and this functionality may need to be replicated in the browser environment. For this purpose it is more and more common to limit the number of individual connections and to instead multiplex them over a single transport connection. See, for example, SPDY [18] and developments in the VoIP context [19]. Worse than inefficiency is that some real-time applications do not behave well with the retransmission behavior of TCP. For real-time voice and video applications, for virtual worlds, and for many games it is acceptable to loose video and voice frames from time to time without waiting for retransmission.

Adding the support for UDP to browsers again adds complexity, as the experience with Voice over IP showed, particularly when the protocols are not multiplexed together, so that it is necessary to identify multiple working end-to-end paths for the traversal of Network Address Translators (NATs) and firewalls. With the transition to IPv6 the number of NATs is likely to increase. Furthermore, in many cases it might be desired to perform route optimization for data traffic and to exchange it directly between the two endpoints

whenever possible to reduce the financial costs and the added delay of using an anchor point. For example, Google Talk only requires the involvement of relays for 8% of their calls, as reported in [20] by utilizing ICE.

It should be noted that audio and video streaming capabilities have been available in the browser for a while with plug-in support. More sophisticated audio support, such as tagging audio with x/y positions for 3D audio, is not even possible with the Adobe Flash application today. The challenge with video support in browsers is based on the lack of universal support of a specific video codec. The lack of hardware support is secondary although relevant for increased performance and lower energy consumption. Naturally, supporting different codecs makes the work of web developers and content distributors difficult.

3.3. Security, Privacy, and Cryptographic Processing Limitations

Many protocol mechanisms have several built-in cryptographic primitives and and the same capabilities must be available in the browser in order to migrate applications that use these capabilities. For example, JavaScript allows cryptographic operations to be implemented (see [21] for a JavaScript AES or other cryptographic functions [22] implementation) but access to hardware crypto-processors, smart cards [23] or to key storages from JavaScript is still at an early stage and, at the time of writing, not available as a standardized JavaScript API.

The security model of JavaScript is different than the one offered by Widgets [24] (available with different platforms/operating systems, such as Mac OS X (via the dashboard), Windows 7, Opera, etc.) or classical operating systems. JavaScript code does not declare what operations it is intended to perform. Even with Widgets there is the question of who verifies any of these privileges. It can hardly be assumed that the end user will be bothered with such a responsibility (due to the lack of his or her expertise. Furthermore, the semantic of end-to-end security is challenged when the distinct communication legs support protocols with different semantics, and dissimilar encodings. Imagine a browser that sends location data encoded in JSON [25], for example using [26], to a web server, which converts it to XML, for example into the PIDF-LO format [27] to interoperate with another application service provider. Consequently, this server then uses XMPP to deliver notifications to its users, for example using [28]. No two of these encodings offer the same privacy mechanisms nor security properties.

The privacy implications of a heavily JavaScript-centered Web environment are not yet well understood. For example, the SIP

privacy mechanisms, described in [29], [30], and [31]) rely to a large degree on the end point to select independent RTP/SRTP relays, and to obfuscate important header fields based on the context provided by the user. One could argue that these standardized SIP privacy extensions represent a community design even though those who deploy ultimately make the final decisions about what policies to use. When the executable code itself is provided by the application service provider then the privacy functionality for data minimization can change at any point in time with little possibility that the user will notice. Only the application service provider makes decisions about what functionality it desires without having to consult or agree with anyone else.

3.4. Source Code Hiding Limitations

In many commercial environments it is not desirable to make source code available to the public. With JavaScript the source code is sent from the server to the browser and only compression and obfuscation tools are available [32]. However, the only way to protect code is to not expose it to observers, instead leaving the important code on the server-side and have a minimal public Javascript code segment use asynchronous message exchanges with the server.

4. Recommendations

This section lists a few basic questions for protocol authors. We hope that in answering these questions honestly a thought process will be triggered that may lead you to re-consider your design before starting the standardization effort that may not lead to successful deployment. Note: We use the term 'protocol' below to refer to a protocol extension, a single protocol, or to a complete protocol suite, or an entire architecture.

1. Does your standardization effort fall priminarily into the client-to-server interaction described in this document? If the answer is "yes", is there a story how the involved stakeholders can innovate at a high speed?

2. Are you attempting to offer functionality typically found at the application layer at the lower layers? If so, have you carefully investigated the cost vs. benefit tradeoff?

3. Does your protocol design involve other stakeholders whoes goals are either not known or potentially not aligned with the goals of your envisioned deployment, i.e. for successful deployment do you require cooperation of stakeholders who may have disincentives (or unclear incentives) to deploy your protocol?

4. When designing your protocol have you considered the Web application environment? Do you understand Web development yourself or do you have experts from the Web development community involved in your work?

5. Does your protocol design offer the ability to carry payloads on HTTP/HTTPS?

6. Why is the current Web framework unable to meet your application requirements? Have you documented the reasons?

7. Have you implemented your protocol in a tyipcal Web development programming language? Hands-on experience may help you to detect problems with using your application design in a Web context in early stages of the design.

8. Is your protocol deployed already? If not, who do you envision to implement and deploy it?

5. Conclusions

This document aims to highlight recent trends in Web application development with impact to Internet standardization. In a nutshell, there is a certain class of applications for which the standardization need is diminishing: chances are good that your standardization work will not be relevant relevant in such an environment.

A lot of this change is driven by mobile code distribution using JavaScript executed on the end host (typically in the Web browser) while server-to-server communication is not yet impacted.

> We are, however, already seeing server-side JavaScript implementations. NodeJS [33] is such an example that is built on top of the V8 JavaScript engine. It runs multiple concurrent JavaScript execution engines in one thread allowing to develop a massively concurrent Web server in JavaScript, addressing a typical pain point for server developers when implementing distributed systems. As another example, CommonJS [34] defines APIs that handle many common application needs, including those that go beyond the usage in Web browsers (such as regular command line programs).

Hence, just as the barriers for rapidly deploying code have dropped on the client side; the server side will likely follow.

Even if there are challenges for standardization there are other areas where work is needed:

- The development of of protocol mechanisms to support a larger range of applications will have an important role to play in the future. Examples of such efforts include the currenly ongoing work on 'BiDirectional or Server-Initiated HTTP' in the HYBI working group [35]. For future work on improving the performance of the Web, for example [36], improvements in HTTP, or common security functionality for the Web as standardized in the Web Security working group [37].

- In those areas where application islands want to interact with larger eco-systems the need for cross-domain communication arises. Often, this is done in a proprietary way but for larger distributed systems and for common functions standardized solutions are valuable. This can be observed today within the VoIP environment, although much slower than expected, in the case of Voice over IP peering but also in the Internet identity management community under the umbrella of 'data portability' [38]. As recent IETF work in this area the Open Authentication

Protocol (oauth) [39] working group could be referenced. OAuth deals with more sophisticated security protocol interactions that require multiple parties to participate in an interoperable way.

- Everyone knows that protocol design is hard regardless whether it happens inside a standards developing organization, like the IETF or W3C, or in some other less structured community. For Web developers the standardization results are often only visible if they appear in form of rich JavaScript libraries and development frameworks, such as JQuery [40], the Prototype JavaScript Framework [41], MooTools [42], YUI [43] and Narwahl [44]. In order to have an impact in the Web community it is essential for working groups participants to think about how to their protocols can be deployed in a Web environment, for by making JavaScript implementations available. The desire in the standards developing community, including the IETF, to be programming language agnostic and to avoid API standardization may need to be re-visited in light of these recent developments. Extending JavaScript may, for example, require new Document Object Models (DOMs) [45] and these could serve as a valuable contribution.

Offering almost unlimited capabilities to JavaScript/HTML running in a browser (in the same style as native applications run in an operating system environment) will raise security concerns and will consequently require countermeasures (such as 'deep inspection' and blocking). This in turn will sparkle new ideas to bypass limitations introduced, for example by utilizing new scripting languages with different capabilities, etc. This is an arms race that the IT industry is already able to observe already with deep packet inspection firewalls and peer-to-peer networks during the last few years.

It is unavoidable to get the impression that the hard problems, particularly to security concerns regarding the distribution of new software in whatever form, have not been tackled. Instead, the browser becomes the new operating system, inherits the same weaknesses and is likely to share the same fate.

6. Security Considerations

This document includes discussions related to security.

7. IANA Considerations

This document does not require actions by IANA.

8. Acknowledgements

The authors would like to thank Gonzalo Camarillo, Robert Sparks, Alissa Cooper, Blaine Cook, Alexey Melnikov, Peter Saint-Andre, Jonathan Rosenberg, Lisa Dusseault, Joshua Bell, John Hurliman, Meadhbh Hamrick, Mark Nottingham, Anders Rundgren, Markus Isomaki, Spencer Dawkins, Jan Kall, Jan Ignatius and Thomas Roessler.

An early version of this document was written to provide additional background for the IETF#80 IAB technical plenary discussion in Prague, March 2011. A number of persons provided their feedback, including Dave Crocker, Pete Resnick, Leslie Daigle, Harald Alvestrand, Jonathan Rosenberg, Dave Cridland, Nico Williams, Peter Saint-Andre, Graham Klyne, Philip Hallam-Baker, Scott Brim, Henry Sinnreich, Eliot Lear, Mark Nottingham, Paul Hoffman, Ted Hardie, Cyrus Daboo, Claudio Allocchio, and Sam Hartman. We thank them for the lively discussion.

9. Informative References

[1] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, May 1996.

[2] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[3] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[4] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

[5] "W3C HTML Working Group Charter", Sep 2010.

[6] "V8 JavaScript Engine", Sep 2010.

[7] "V8 JavaScript Engine - Design Elements", Sep 2010.

[8] "WebGL", Sep 2010.

[9] "Quake II Google Web Toolkit (GWT) Port", Sep 2010.

[10] "The WebSocket API", Sep 2010.

[11] "Web Storage", Aug 2010.

[12] "Second Life", Sep 2010.

[13] "Private communication between Joshua Bell, Hannes Tschofenig and Jon Peterson about browser performance limitations", Aug 2010.

[14] "Elevating JavaScript Performance Through GPU Power", Jan 2010.

[15] "Jetpack 0.5 Contest: A Winner", Nov 2009.

[16] Rosenberg, J., "UDP and TCP as the New Waist of the Internet Hourglass", draft-rosenberg-internet-waist-hourglass-00 (work in progress), February 2008.

[17] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[18] "SPDY: An experimental protocol for a faster web", Oct 2011.

[19] Westerlund, M., Burman, B., and C. Perkins, "RTP Multiplexing Architecture", draft-westerlund-avtcore-multiplex-architecture-01 (work in progress), March 2012.

[20] "Google Talk for Developers: Important Concepts", Sep 2010.

[21] "JavaScript Implementation of AES Advanced Encryption Standard in Counter Mode", Sep 2010.

[22] "crypto-js: JavaScript implementations of standard and secure cryptographic algorithms", Sep 2010.

[23] "JavaScript Crypto", Sep 2010.

[24] "W3C Web Applications (WebApps) Working Group", Sep 2010.

[25] "JavaScript Object Notation (JSON)", Sep 2010.

[26] "The GeoJSON Format Specification", Jun 2008.

[27] Peterson, J., "A Presence-based GEOPRIV Location Object Format", RFC 4119, December 2005.

[28] "XEP-0080: User Location", Sep 2009.

[29] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.

[30] Munakata, M., Schubert, S., and T. Ohba, "Guidelines for Using the Privacy Mechanism for SIP", RFC 5379, February 2010.

[31] Munakata, M., Schubert, S., and T. Ohba, "User-Agent-Driven Privacy Mechanism for SIP", RFC 5767, April 2010.

[32] Crockford, D., "(JavaScript) Minification v Obfuscation", Mar 2006.

[33] "nodeJS", Sep 2010.

[34] "CommonJS", Sep 2010.

[35] "IETF BiDirectional or Server-Initiated HTTP (hybi) Working Group Charter", Mar 2011.

[36] "Let's make the web faster", Sep 2010.

[37] "IETF Web Security (websec) Working Group Charter", Mar 2011.

[38] "Data Portability Project: Share and Remix Data using Open Standards", Sep 2010.

[39] "IETF Open Authentication Protocol (oauth) Working Group Charter", Sep 2010.

[40] "jQuery: The Write Less, Do More, JavaScript Library", Sep 2010.

[41] "Prototype JavaScript framework: Easy Ajax and DOM anipultion for dynamic web applications", Sep 2010.

[42] "MooTools - a compact javascript framework", Sep 2010.

[43] "Yahoo! User Interface Library 3", Sep 2010.

[44] "Narwhal - A general purpose JavaScript platform", Sep 2010.

[45] "Document Object Model", Sep 2010.

[46] "W3C Workshop on Privacy for Advanced Web APIs", Jul 2010.

[47] "W3C Geolocation Working Group", Sep 2010.

[48] "Ajax (programming)", Sep 2010.

[49] "Device APIs and Policy Working Group", Sep 2010.

[50] Doty, N., Mulligan, D., and E. Wilde, "Privacy Issues of the W3C Geolocation API, UC Berkeley School of Information Report 2010-038", Feb 2010.

[51] "Adobe Flash Player", Sep 2010.

[52] "Microsoft Silverlight", Sep 2010.

Authors' Addresses

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Email: bernarda@microsoft.com

Jon Peterson
NeuStar, Inc.
1800 Sutter St Suite 570
Concord, CA 94520
US

Email: jon.peterson@neustar.biz

Danny McPherson
Verisign
US

Email: danny@tcb.net