ICFlash: Web-based Network Measurement Using Adobe AIR

CSE 534 Final Research Paper

Jian Xu[1], Benjamin X. Lin[1], Yang Sheng Fang[1]

[1]Department of Computer Science, Stony Brook University,
Stony Brook, NY 11794-4400

{jianxu1, xianlin, yafang}@cs.stonybrook.edu

http://www.ic.sunysb.edu/stu/xianlin/~cse534

ABSTRACT

In order to help ICLab [1] collect data for censorship related researches, we develop ICFlash, a network measurement application embedded on a webpage. In specific, ICFlash allows an end user to issue HTTP GET requests or DNS queries to a remote web server and upload corresponding results to the Centinel Server. We implement ICFlash by using Adobe AIR, after investigating the feasibility of HTML5, JavaScript, PHP, Adobe Flash and AIR, since AIR does not have the cross-domain restriction. We describe our design and implementation details, provide the user manual and sample results of ICFlash application, and outline the future work.

Key Words: Cross-Domain, Adobe Flash, Adobe AIR, HTTP, DNS, Centinel Server

1. INTRODUCTION

Currently, the ICLab [1] application requires an end user to download the Centinel Client and run it locally to collect enough information for research purposes. It requires too complex configurations and installation steps that may limit the number of users who are willing to participate in this censorship data collection program.

We create an Adobe AIR application named ICFlash for web browsers, enabling users to help ICLab collect HTTP and DNS data by simply typing URL links and clicking buttons. In order to implement these functions, ICFlash first issues an HTTP GET request (or DNS query) to the web server designated by the URL; then it uploads the HTTP (or DNS) response to the Centinel Server.

In the rest of this paper, we firstly review the background (**Section 2**) technologies that may be used for ICFlash and briefly describe why we reach our decision of using Adobe AIR. Next, we introduce the design of our application (**Section 3**), followed by its implementation (**Section 4**). We present the prototype of ICFlash in **Section 5**, discuss the future work in **Section 6**, and conclude in **Section 7**.

2. BACKGROUND

In this section, we review several web technologies for their feasibility in ICFlash.

2.1 HTML5

HTML5 is a core technology markup language of the Internet used for structuring and presenting content for the World Wide Web. It is derived from HTML4, which was standardized in 1997. Comparing to HTML4, a great improvement of HTML5 is the support of the latest multimedia technology [2].

Some new features were designed to make it easy to include and handle multimedia and graphical content on the web without having to resort to proprietary plugins and APIs. Additionally, several technologies were applied in HTML5 to support specific features, like WebSocket API, to provide full-duplex communication channels over a single TCP connection [3]. So we can apply Websocket API to issue HTTP requests to servers.

While in this case in order to process successfully, servers that accept the WebSocket protocol requests have to support the protocol by upgrading. This is hardly practical in our project since we have to urge all the web servers to upgrade to support the Websocket protocol.

Therefore, applying HTML5 with Websocket API is not practical in our project.

2.2 JavaScript

JavaScript is a language supported by all modern web browsers. It complements HTML and CSS by providing client-side scripting. JavaScript is commonly used to change the document content, control the browser window, communicate asynchronously with the web server, and do complex math calculations [4]. Consequently, many web applications and games use JavaScript.

While JavaScript provides some APIs to communicate with servers, it is restricted by the cross-domain policy. This policy does not permit sending requests to another website with a different domain name. Because our application needs to send requests to other domains, this technology is not suitable.

2.3 PHP

PHP is a server-side scripting language designed for web development, in which case PHP generally runs on a web server (e.g. Apache, Nginx, etc.)

PHP code can be mixed with HTML code to construct a dynamic webpage. It is usually processed by a PHP interpreter, which is often implemented on the web server side by a server native module or a Common Gateway Interface (CGI) executable. After the PHP code is interpreted and executed, the web server sends the resulting output to its client, usually in form of a part of the generated web page; for example, PHP code can generate a web page's HTML code, an image, or some other data [5].

According to the fact that PHP is a server-side scripting language and can only be executed on the server side, this makes us have to execute our program remotely. Therefore, servers instead of browsers will send the DNS/HTTP requests that we embed in PHP programs. Results will reflect the accessibility in servers' perspective instead of users which is not what we intend to measure.

Alternatively we can achieve our goal by deploying LAMP-like environment in client-side (e.g. LAMP [6], MAMP [7], WAMP [8]) where to let PHP scripts execute on, but this approach breaches our principle that do not install heavy packages and complex configurations on the client side.

As a consequence, PHP is not an ideal method for this project.

2.4 Adobe Flash

Adobe Flash is a common user-side presentation platform in modern web applications. Its common usage includes video player, animation, website navigation, and complex application (such as image editor). It supports cross-platforms running on Windows, Mac OS, and Linux. It not only has a wide number of platform supports, but also gains a huge user base.

Flash application is written in ActionScript with various front-end technologies. The source code is compiled into a Shockwave Flash file (i.e. SWF file).

SWF files can be embedded in web pages, and they must be running on a system with Adobe Flash installed.

Flash provides certain APIs that do not exist in browsers. These APIs include TCP streams and custom TCP requests, which are crucial to our application implementation.

However, Flash enforces certain security policies. One of such is the cross-domain policy [9]. The target server which the Flash application tries to access, must grant permission to the domain where the Flash application resides, by including a "crossdomain.xml" file on its own server. For example, if a Flash application on *www.a.com* wants to access *www.b.com*, *www.b.com* must include "crossdomain.xml" on its root that grants permission to *www.a.com*. This policy demonstrates a huge restriction on what we want to achieve, which is to send requests to any domain. And, it is impractical for us to ask web administrators to grant our application access to their web servers. Therefore, we opt not to use Flash.

2.5 Adobe AIR

Adobe AIR supports the full Adobe Flash API so that it gains all the advantages of Flash. In addition, AIR provides an API named *flash.net.dns* for DNS queries and other APIs supporting file system integration and access to connected devices. The AIR runtime supports installable applications on Windows, OS X and mobile operating systems like Android, iOS and BlackBerry Tablet OS [10].

Another advantage of Adobe AIR is allowing Adobe Flash and ActionScript code to construct applications that run outside of a web browser, and behaves like a native application on supported platforms [10].

The most important point is that AIR does not have the cross-domain restriction. Consequently it allows us to send HTTP requests to arbitrary servers [10].

According to the investigations on HTML5, JavaScript, PHP, Flash and AIR technologies, we find that AIR is a suitable platform for our application because AIR does not have the cross-domain restriction and also provides APIs accessing low level streams. Therefore, we decide to implement ICFlash using AIR.

3. DESIGN

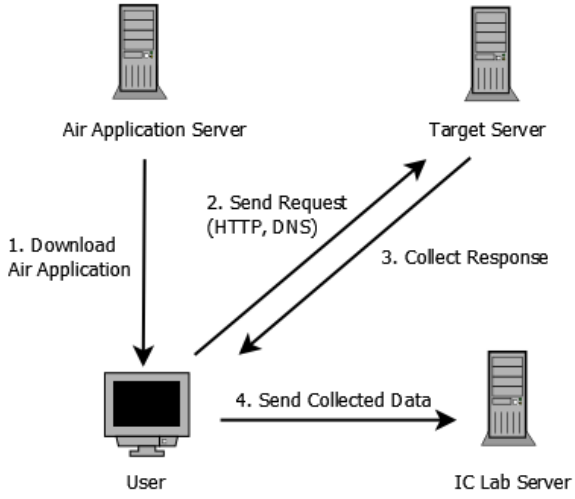

Figure 1. Interaction between ICFlash and other systems

Figure 1 presents an overview of ICFlash's workflow.

Initially, the user visits our web server to download ICFlash application. Next, the user can use a specified URL or our predefined URL list to send HTTP GET requests or DNS queries to the target server. After that, ICFlash records

the response from the server and presents the result to the user. If the request is an HTTP GET, the result will include HTTP response status code, response URL, the name-value pairs of the HTTP header and HTML content. If the request is a DNS query, the result will contain one or more IP addresses for the requested domain name. In both cases ICFlash will record the round-trip time (RTT). Finally, with the user's consent, ICFlash sends the response data to the ICLab Centinel Servers for further analysis.

4. IMPLEMENTATION

4.1 User Interface

Our user interface is written in Flex. Flex is an SDK for developing user interface for the Adobe Flash platform. One advantage of Flex is to provide consistent user experience on all major browsers, desktops, and devices [14]. Flex is based on XML technology so the code is easy to write and maintain.

An example of Flex component we used is the Datagrid, which provides a multiple column table for displaying our result. We add a row in the datagrid for each result we received. In each row, we show the original domain URL, the status code, the round-trip time, and the uploading status. We also add an event listener to the datagrid to handle users' interactions. When the user clicks on a record in the datagrid, the textbox on the right of the Result Panel will be updated to show details of that result record.

4.2 Library

In order to implement the function of sending HTTP GET requests, we use the *flash.net* API [11] from the Flash library. We first build a *URLRequest* object, which stores various parameters related to the HTTP request. We set the method property to GET, and the *contentType* to "text/html". Then by using an instance of *URLLoader*, our application sends an HTTP request to a remote server. An event handler is used to asynchronously process the response. We extract the data and then present it.

We use the *flash.net.dns* API to send DNS queries. With a *DNSResolver* object, we use the *lookup* method to resolve a domain name to obtain its IPv4 address. In the event handler, we process the result in the *DNSResolverEvent* object and display the data in our user interface.

4.3 Result Uploading

In order to let ICLab easily decode the result, we choose the JSON encoding. JSON is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs [15]. Some examples we used include: the attribute *body* paired with the result HTML body, the attribute *time* paired with the round-trip time, and so on.

We then upload this JSON encoded result onto the Centinel Server. Because the Centinel Server uses REST [17] architecture, we can upload the result by sending an HTTP POST request. We use the method similar to HTTP GET request detailed as mentioned previously except the *method* property being changed to POST instead of GET. We embed the JSON string into the *data* property separated by boundaries as complied with the HTTP standard [16].

4.4 Embedding Air into a Web Page

We compile our code into an AIR installer and then deploy it to our current server. We use a collection of sample html files named "Badge" from the official Adobe site [12]. Then we replace the AIR file with ours, and modify the AIR version and the absolute URL to our AIR file.

Then we upload the whole folder containing "Badge" files to our server, allowing users to test our application.

5. PROTOTYPE

5.1 User Interface

The user interface of ICFlash looks like *Figure 2*.

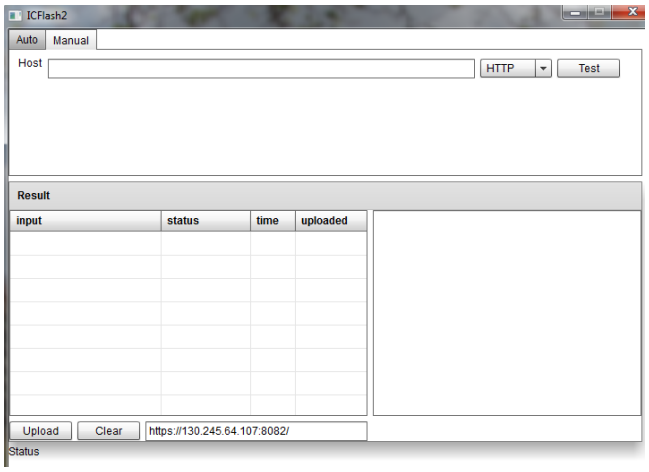

Figure 2. The user interface of ICFlash

5.2 User Manual

Step 1: The user needs to visit the webpage with ICFlash embedded, as shown in *Figure 3*.

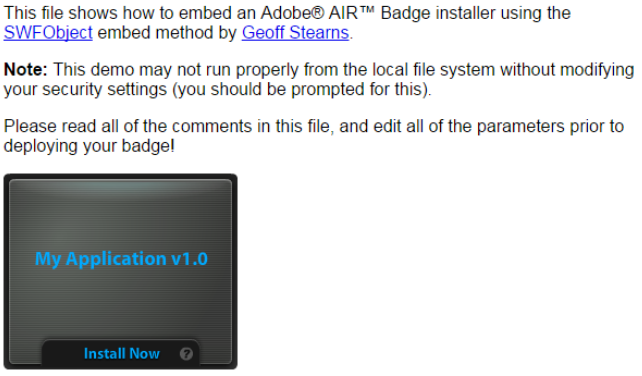

Figure 3. The initial webpage to start ICFlash installation

Step 2: The user clicks the "Install Now" button. ICFlash asks the user whether or not he wants to open and run it as shown in *Figure 4*.

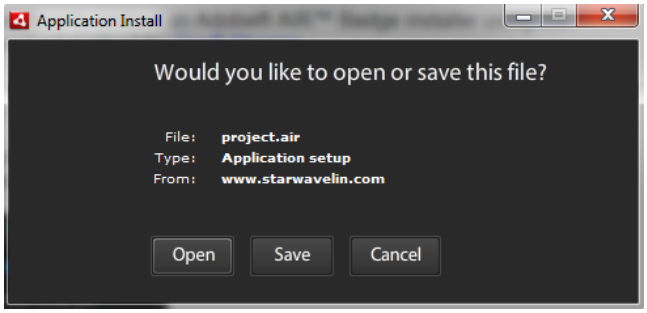

Figure 4. Installation Prompt

Step 3: The user will see the warning message in *Figure 5*. By clicking "Install" button, he confirms his decision of installing.

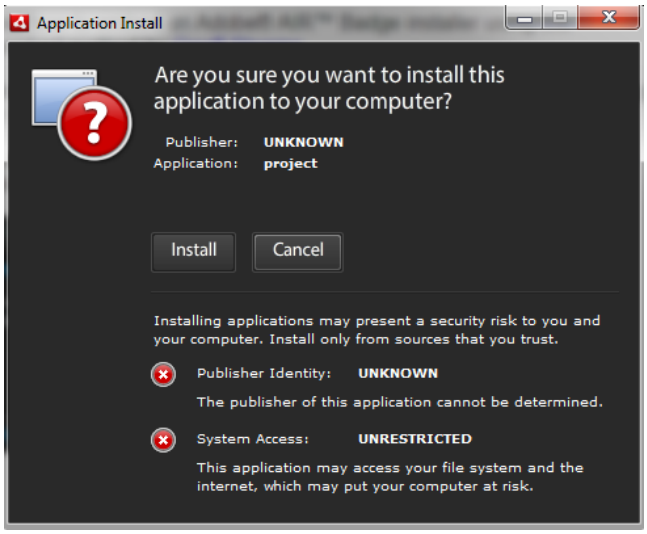

Figure 5. Request uer confirmation

Step 4: Then ICFlash asks the location where he wants to install it as shown in *Figure 6*.

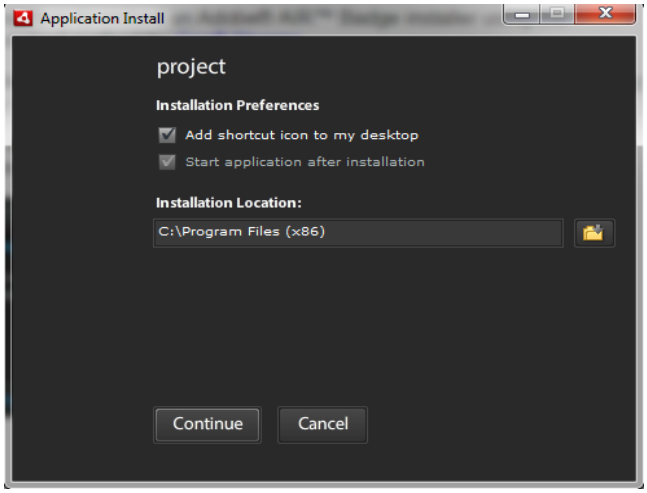

Figure 6. Installation Path

Step 5: By clicking the "Continue" button, the installation of ICFlash will start.

After the installation is finished, the interface of ICFlash will pop up, as shown in *Figure 2*.

Then the user can click the "Manual" tab on the interface to start his testing.

5.3 Result

5.3.1 Result of an HTTP GET Functionality

After the user inputs a correct website URL, selects HTTP in the dropdown menu, and then clicks "Test" button, he will see the HTTP status code, the round-trip time (RTT) and the status for this result to be uploaded in the grid on the left of the Result Panel; after clicking the record in the grid, he will further see the HTML content of the website on the right of the panel, as shown in *Figure 7*.

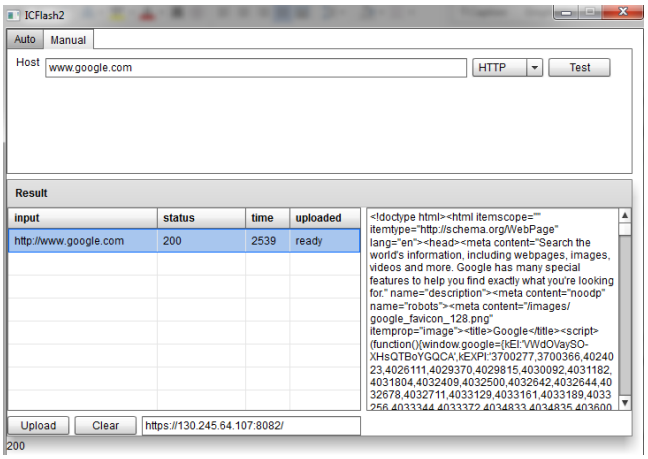

Figure 7. Result of an HTTP GET functionality

5.3.2 Result of an DNS Query Functionality

Similarly, if the user inputs a site URL, selects DNS in the dropdown menu, and then clicks "Test" button, he will see the RTT to run this DNS query and its uploading status; after he further clicks the record, ICFlash will display the detailed DNS query results on the right of the Result Panel, as shown in *Figure 8*.

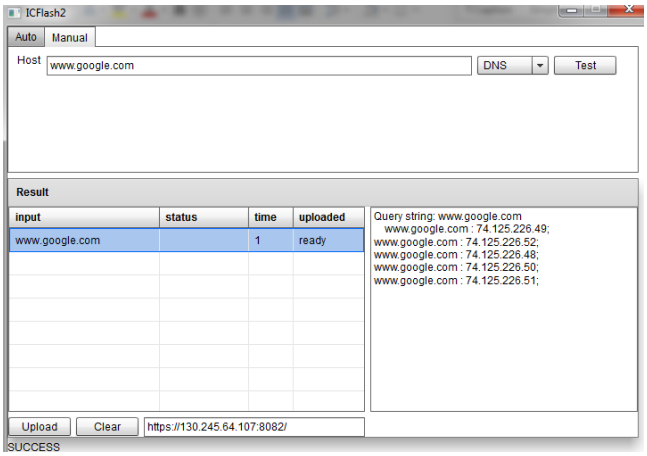

Figure 8. Result of an DNS query functionality

By clicking the "Clear" button below the grids, the user can always remove the current results displayed on the panel.

5.3.3 Send Collected Results to a Local Centinel Server

We have enabled the functionality for the user to upload the previously obtained result to a locally deployed Centinel Server in JSON format. The user simply needs to click the result record shown in the grid, modifies the IP Address Field to the right of the "Clear" button, and then click "Upload" button. After clicking the "YES" button to prompted security alerts, he will see the uploading status of this result changes from "ready" to "uploading" then to "done", as shown in *Figure 9.1*. Simultaneously, the local Centinel Server logs the result sent by the user, as shown in *Figure 9.2*. If error occurs in the uploading process, the status code for the result would display "error", as shown in *Figure 9.3*.

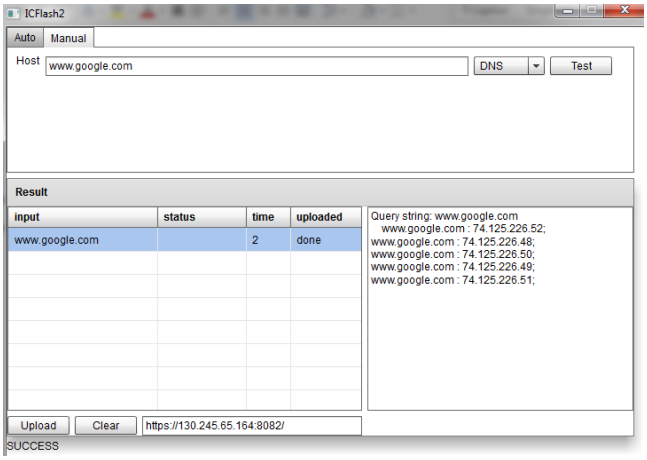

Figure 9.1. A successful result uploading from the user side

Figure 9.2. The corresponding DNS query result on the server side

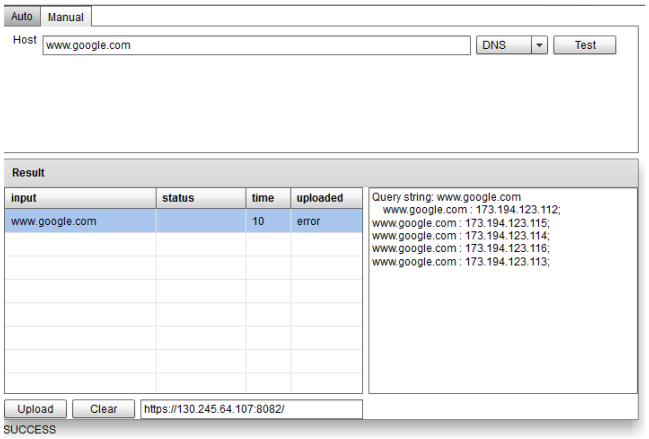

Figure 9.3. A result when uploading failed

5.3.4 Allow Users to Automatically Test a List of URLs

Besides the required functionalities we have implemented, as illustrated in *Section 5.3.1* to *5.3.3*, we have also implemented a feature to supply a list of URLs for a user to test in sequence, as shown in *Figure 10*. Similarly, when the user clicks any result record in the grid, he will see the detailed result on the right of the Result Panel. The functionality of uploading results to the local Centinel Server is also available.

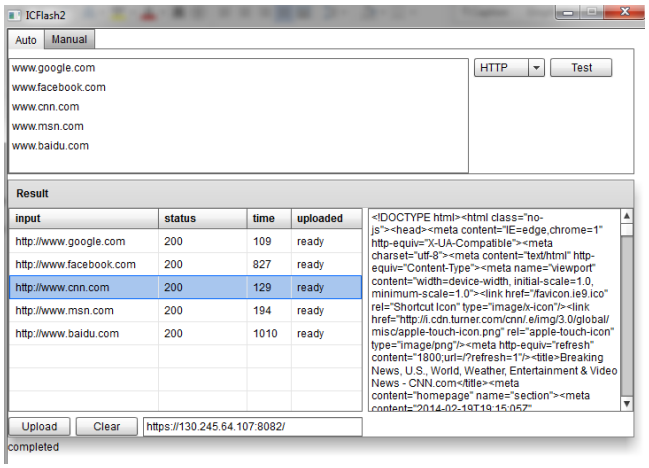

Figure 10. Result from executing a list of HTTP GET requests

6. FUTURE WORK

Till now, we have overcome the cross-domain restriction, completed the HTTP GET and DNS query modules, the uploading to a locally deployed Centinel Server functionality [13] and an extra feature of including a list of URLs for a user to test in batch.

For the future, we plan to give users more flexibility by allowing them to upload a customized list of URLs to test. Moreover, we will contact the administrator of ICLab Centinel Server Project to figure out the way of uploading users' testing results to the actual Centinel Server instead of a locally deployed one. Last but not the least, we would coordinate with ICLab to promote our ICFlash application to the volunteers who participated or are willing to participate in ICLab's data collection program over the world.

7. CONCLUSION

In order to get HTTP/DNS information from users' perspective via browsers, we implement a web-based application named ICFLash. In this paper we first investigate the feasibility of several technologies such as HTML5, JavaScript, PHP and Adobe Flash.

Then we eliminate those technologies, as they are not able to satisfy our requirements. For instance, we cannot get user-perspective information from servers executing PHP; since not all servers support WebSocket protocol, it is impossible to deploy a general application which sends WebSocket requests via HTML5; because JavaScript and Flash are subjected to the cross-domain policy, they are impractical for us to apply, either.

Hence, we decide to use Adobe AIR technology due to its capability of satisfying our project requirements and convenient deployment, for instance, to acquire user-perspective accessibility to web servers.

Next, we describe several key steps on how to realize the project, for example, how to send HTTP GET requests and DNS queries by using Adobe AIR APIs, how to upload the corresponding results to the Centinel Server in JSON format, etc.

We also demonstrate the user manual of how to install and use the ICFlash step by step.

Finally, we look into the future work of ICFlash, planning to give users more flexibility of customizing a list of URLs to test, coordinate with the ICLab administrator for uploading testing results to the actual Centinel Server, and promoting our ICFlash application to the volunteers who participate in ICLab's data collection program over the world.

8. REFERENCES

[1] The Internet Censorship Lab: http://www.internetcensorshiplab.com/

[2] http://en.wikipedia.org/wiki/HTML5

[3] http://en.wikipedia.org/wiki/WebSocket

[4] http://en.wikipedia.org/wiki/JavaScript

[5] http://en.wikipedia.org/wiki/PHP

[6] http://en.wikipedia.org/wiki/LAMP_(software_bundle)

[7] http://en.wikipedia.org/wiki/MAMP

[8] http://en.wikipedia.org/wiki/LAMP_(software_bundle)#WAMP

[9] http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7e3f.html

[10] https://en.wikipedia.org/wiki/Adobe_AIR

[11] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/package-detail.html

[12] Adobe *Getting started with the custom install badge.* https://www.adobe.com/devnet/air/articles/badge_for_air.html

[13] Discussion of our project scope with Abbas Razzaghpanah, a PhD student participating in the Internet Censorship Lab. Friday, April 03, 2015

[14] *Flex*. https://www.adobe.com/products/flex.html

[15] https://en.wikipedia.org/wiki/JSON

[16] *Form-based File Upload in HTML*. https://tools.ietf.org/html/rfc1867

[17] https://en.wikipedia.org/wiki/Representational_state_transfer