An Empirical Evaluation of Web-Based Fingerprinting

Amin Faiz Khademi and Mohammad Zulkernine, Queen's University

Komminist Weldemariam, IBM Research—Africa

// *An analysis of fingerprinting techniques and tools revealed the fingerprinting workflow. This helped define fine-grained properties that are indicators and can be used for developing a client-side fingerprinting-detection tool.* //

FINGERPRINTERS EMPLOY various techniques to uniquely identify Web users—for example, by collecting fine-grained information from the browser, communication protocol, or OS.[1–3] The collected information can serve as the browser's fingerprint. Such fingerprints can be exploited to track users across websites to infer various insights such as their browsing behavior over time. Although fingerprinting has various legitimate applications, it can be used to obtain and exploit users' personal information without their consent.

We analyzed fingerprinting methods and tools to determine the fingerprinting workflow (sequence of actions) and metrics that precisely indicate fingerprinting attempts. This analysis led us to design, implement, and evaluate *Fybrid*, a Web-based hybrid fingerprinting tool. Using Fybrid, we collected fingerprints for 1,523 browsers and quantified each fingerprinting method's effectiveness. In addition, we integrated Fybrid with a social-networking service, enabling us to identify individuals rather than browsers.

The Challenges of Fingerprinting

One reason for collecting information through fingerprinting is that most websites need to know about the underlying browser and device to run properly. For instance, a website might need to customize its appearance on the basis of the dimensions of the user's screen. It might also need to determine the presence of a particular browser plug-in (for example, Adobe Flash Player) to load additional resources (for example, Flash movies and games).

To provide such properties to developers, browsers have exposed various high-level APIs. For example, `navigator` and `screen` are a browser's built-in JavaScript objects that expose APIs to Web applications for accessing device- and browser-related properties. Furthermore, browser plug-ins contain APIs for accessing low-level system properties such as I/O, screen resolution, and the OS name.

Although these APIs can add functionality to websites, they expose valuable information and data to malicious fingerprinting providers. This poses three challenges. First, fingerprinters (at least those we investigated) are invisible to users, so the users are unaware of being fingerprinted.

Second, attentive users might observe fingerprinting activities but can't stop or reduce disclosure of the information.[4] The fact that most Web users have limited or no knowledge about the collected data's use aggravates the situation.

Third, many websites, including

fingerprinting websites, can be integrated with social-networking services (for example, Facebook) to increase their popularity and attract traffic. So, integrating fingerprinting with such services can link the browsers' fingerprints with every user profile. This further opens a door for fingerprinters to easily reveal users' identity and compromise their security and privacy.

To preserve users' privacy, fingerprinting providers often provide an opt-out feature for users who don't want to share their browser's fingerprint. For example, AddThis, a popular content-sharing platform, employs this approach.[5] However, such solutions are inadequate because most users aren't knowledgeable about being fingerprinted, as we just noted. Moreover, a recent study reported that AddThis doesn't stop data collection even if the user opts out of fingerprinting.[6] Similarly, we observed that most fingerprinting methods can easily evade existing fingerprinting detection and prevention mechanisms. So, users need a protection solution that detects and prevents fingerprinting attempts on the client side without relying on fingerprinting providers that focus on server-side solutions.

Users can also try to avoid trackers by using tools such as Adblock-Plus and browser features such as private mode.[7–9] These methods stop tracking attempts by exploiting client-side technologies such as browser and Flash cookies. However, some fingerprinters can easily evade these defenses.[6]

Fingerprinting Methods

We analyzed four major Web-based fingerprinting methods used in large-scale experimental studies:[6,10,11]

- JavaScript object fingerprinting,
- JavaScript-based font detection,
- canvas fingerprinting, and
- Flash-based fingerprinting.

We also examined popular fingerprinters such as BlueCava (http://bluecava.com/opt-out), ThreatMatrix (www.threatmetrix.com), and AddThis. Besides determining each method's workflow, we identified nine method-specific metrics (see Table 1).

Using these metrics, researchers could exploit observed fingerprinting activities to develop a tool that detects fingerprinting attempts at run time. Such a tool would monitor and record a webpage's activities on the user's browser. Next, it would extract the metrics related to each fingerprinting method and build a pattern or signature for the webpage. Then, using signatures based on past fingerprints, the tool would identify the new signature as either normal or that of a fingerprinter.

TABLE 1 Metrics for characterizing fingerprinting attempts.

Metric	What the metric specifies
1	The number of accesses to the **navigator** and **screen** objects' properties
2	The number of accesses to the **Plugin** and **MimeType** objects' properties
3	The number of fonts loaded using JavaScript
4	The number of accesses to the offset properties of HTML elements
5	The presence of writes and reads to a canvas element
6	If a canvas element is hidden or created dynamically
7	The presence of methods for enumerating system fonts and accessing system-related information in a Flash file's source code
8	The presence of methods for transferring or storing information using Flash
9	If a Flash file is small or hidden or if it's added dynamically

JavaScript Object Fingerprinting

As we mentioned before, the JavaScript objects found effective for fingerprinting are **navigator** and **screen** (informative objects).[1,12] The **navigator** object represents the device and browser environment properties—for example, browser name, plug-ins, and MIME (Multipurpose Internet Mail Extensions) types. The **screen** object contains information about the settings of the screen displaying the browser (for example, resolution and color depth).

Moreover, **navigator**'s **plugins** property contains the list of installed plug-ins on the browser in the form of **Plugin** objects. Each **Plugin** object has properties such as a plug-in's name, description, and file name. In addition, the **mimeTypes** property contains the list of browser-supported MIME types in the form of **MimeType** objects. Each **MimeType** object has properties such as a MIME type's description and type.

To collect these properties, a fingerprinter must access them by calling them. One metric for recognizing JavaScript object fingerprinting is the number of accesses a webpage makes to the navigator and screen objects' properties (Metric 1 in Table 1). Another metric is the number of accesses to the Plugin and MimeType objects' properties (Metric 2).

JavaScript-Based Font Detection

An HTML document consists of HTML elements that can each have properties for styling purposes. One such property is the font family, which HTML elements use to specify their font. This property has a fallback system indicating that the property can hold several font names. If the first font isn't available on the user's system, the browser tries the next font; this process continues for all fonts in the fallback list. If none of those fonts are available, the browser renders the element with a default font.

Each character has a different style and size in different fonts. For example, a <div> HTML element containing a sentence with a particular font size will have different width and height properties for different fonts. Exploiting the style and size differences of fonts and the font family's fallback system, fingerprinters can check for the presence of a particular font on the user's system.

To do this, they first apply an unknown font (for example, "newfont") that's unavailable on the user's system to an arbitrary sentence (often a pangram) in an HTML element. (A pangram is a string of text containing all the letters of the alphabet—for example, "The quick brown fox jumps over the lazy dog.") Second, they record the default values corresponding to the offset properties (width and height) of the HTML element. Third, they apply the font to the given sentence and measure the current (new) values for the offset properties. Finally, they compare the new and default values. An unavailable font has new values equal to the default values; an available font has new values that are different.[11]

The metrics for JavaScript-based font detection are the number of loaded fonts (Metric 3) and number of accesses to the offset properties (Metric 4).

Canvas Fingerprinting

The HTML5 canvas element draws graphical content on the fly using JavaScript. To fingerprint a browser, fingerprinters first draw an arbitrary context (usually a pangram) on a canvas element. Then, they collect the canvas element's image data (pixels), which depends on the underlying software and hardware configurations (font libraries, graphics hardware, and OS). So, the image data is different for different combinations of browsers and OSs and thus could be used for browser identification.[2] So, a metric for canvas fingerprinting is the presence of writes and reads to a canvas element (Metric 5).

In addition, fingerprinters will hide canvas elements in a document or dynamically add them to a document (for example, by using the JavaScript document.createElement method). So, another metric is if the canvas element is hidden or created dynamically (Metric 6).

Flash-Based Fingerprinting

Two Flash classes that are mainly accessible through APIs for acquiring system information are Capabilities and Font.[12] Capabilities provides information about the underlying system (for example, the OS name) and running application (for example, the Flash plug-in version). Font provides APIs for managing embedded fonts in Flash files. For example, it has a method (Font.enumerateFonts) for getting all available embedded and system fonts. So, Flash can be used for enumerating system fonts and collecting system information.

We defined three metrics for Flash-based fingerprinting. The first is whether the Flash file contains methods for enumerating system fonts or accessing system-related information (Metric 7). The second is the presence of methods for transferring or storing information using Flash (Metric 8). The third is if the Flash file is small or hidden or if it's added dynamically (Metric 9).

Fybrid

Fybrid combines the methods we just discussed to create a fingerprint for a given browser instance. We deployed it as a Web application; it's also available as an open source tool (https://github.com/ammosh/fybridjs).

Figure 1 shows an overview of our approach. Fybrid has two main components. The *collector* collects the browser's properties by exploiting various APIs; the *ID builder* transforms those properties into the fingerprint.

For JavaScript object fingerprinting, we identified 20 properties observable through navigator and 12 properties observable through screen. Also, for Plugin and MimeType, we concatenated each object's properties and considered them as one property. We thus considered the plugins array as one property and the mimeType array as one property. Fybrid queries for the 32 properties, extracts them, and employs them for fingerprinting.

For JavaScript-based font detection, Fybrid executes the workflow we described earlier. To create a list of fonts for checking their presence on the user's system, we investigated BlueCava's code and found 460 fonts used for fingerprinting. Fybrid checks for the presence of these fonts, which we classify by OS (231 Windows fonts, 167 Mac fonts, and 62 fonts for other OSs).

For canvas fingerprinting, Fybrid draws a pangram on a canvas element (300 × 177 pixels) and then extracts the generated image data as a Base64 encoded string.

For Flash-based fingerprinting, Fybrid uses the `Font` API to extract system fonts by calling the `enumerateFonts` method. It uses the `Capabilities` API to collect and extract 39 system-related properties.[13]

The ID builder applies the MD5 hash function on the collected information and generates a fingerprint. Fybrid sends that fingerprint and the collected information to remote storage.

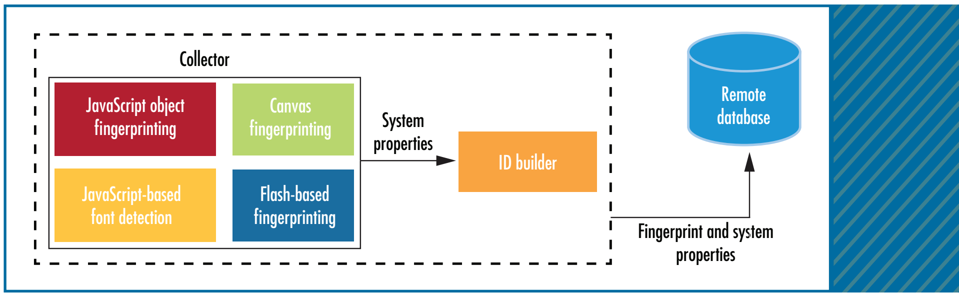
Collector
JavaScript object fingerprinting
Canvas fingerprinting
JavaScript-based font detection
Flash-based fingerprinting
System properties
ID builder
Remote database
Fingerprint and system properties

FIGURE 1. An overview of Fybrid, a hybrid browser-fingerprinting tool. The *collector* collects the browser's properties by exploiting various APIs; the *ID builder* transforms those properties into a fingerprint.

TABLE 2 A breakdown of the 74 features extracted.

Method	No. of features	Description
JavaScript object fingerprinting	32	Properties accessible through `navigator` and `screen`
JavaScript-based font detection	1	Number of available fonts (out of 460) on the user's system
Canvas fingerprinting	1	Image data of the canvas element
Flash-based fingerprinting	40	One feature for the available fonts on the user's system; 39 for the system-related properties

Data Collection and Feature Evaluation

We asked members of the microWorkers marketplace (https://microworkers.com) to visit the Fybrid Web app and submit their browsers' fingerprints (with their consent upon clicking on the submit button). This let us collect 1,523 unique fingerprints. We stored each fingerprint as a vector of 74 properties (or features) of the user's system (see Table 2).

To evaluate the importance of features in a fingerprint, we calculated their entropy values, for which we used the Shannon entropy equation.[14] Given feature F in the fingerprint with n different values (for all 1,523 fingerprints), the entropy (produced information) is

$$H(F) = -\sum_{i=1}^{n} p_i \times \log_2 p_i \,,$$

where p_i is the probability of the occurrence of the ith value. With this equation, a feature that has different values for all 1,523 browser fingerprints has the maximum entropy value of 10.57 bits and can serve to uniquely identify each browser instance.

Figure 2 shows the 10 attributes with the highest entropy values. The `plugins` property had the highest value (9.97 bits). This means that a randomly chosen browser shared the same plug-ins with at most one of the other 1,522 browsers ($2^{9.97}$). The value of `plugins` for each browser was the concatenation of the properties (name, filename, and description) of all the browser's plug-ins. When we used only `plugins` as the fingerprint, 82 percent of the fingerprints were unique. When we used all the features, 90.41 percent of the browsers had unique fingerprints. This shows `plugins`' significance in making fingerprints unique.

After `plugins`, `system fonts` (collected using the Flash plug-in) and `JavaScript fonts` (collected using JavaScript-based font detection) had the highest entropy values. As we mentioned before, we used only a set of 460 fonts for JavaScript-based font detection. Adding more fonts would have made this detection method's entropy value closer to that of font enumeration using the Flash plug-in.

Canvas fingerprinting (`canvas` in Figure 2) had an entropy value of 3.2 bits. This technique effectively

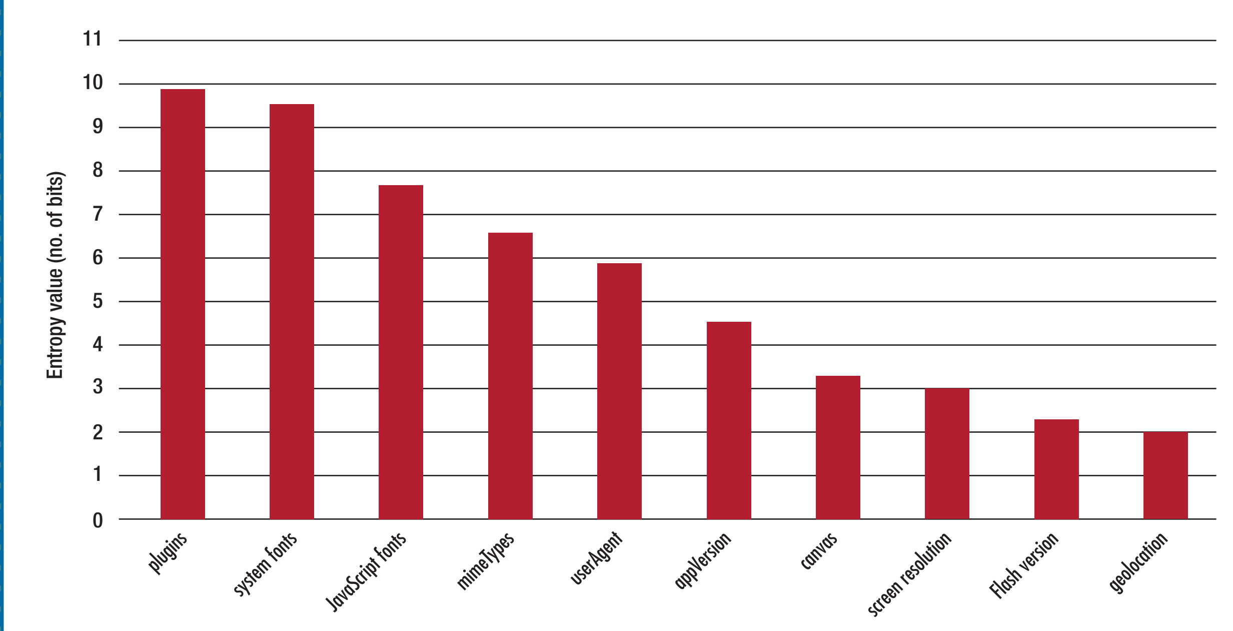

FIGURE 2. The 10 attributes with the highest entropy values. When we used only **plugins** as the fingerprint, 82 percent of the fingerprints were unique.

TABLE 3 Entropy and uniqueness values for the fingerprinting methods.

Method	Entropy (no. of bits)	Unique browsers (%)
JavaScript object fingerprinting	10.27	89.69
Flash-based fingerprinting	10.07	83.25
JavaScript-based font detection	7.64	43.79
Canvas fingerprinting	3.21	0.05
All methods	10.28	90.41

identifies hardware and software configurations of the user's system and is prevalent among the top Alexa websites.[6] But by itself it's not ideal for fingerprinting owing to its low entropy value, as our experiment demonstrated.

Before our experiment, we assumed that all the features were necessary to uniquely fingerprint a browser or system. However, we observed that most attributes (for example, **appCodeName**) often used in the fingerprinters didn't contribute to fingerprinting activities and thus might be redundant.

To analyze how much information websites need to identify a particular browser, we applied *greedy forward feature selection* (GFFS)[15] to the 74 features. To identify a given browser, we needed only eight features: **plugins**, **userAgent**, **screen resolution** (**width** × **height**), **geolocation** (**latitude** and **longitude**), **onLine**, **system fonts**, **screenDPI**, and **canvas**. The identification accuracy of using these eight features was equal to that of using all the features.

We also calculated the entropy values for the four fingerprinting methods separately (see Table 3). JavaScript object fingerprinting had the highest value (10.27 bits); 89.69 percent of the browsers had unique JavaScript object properties. Moreover, JavaScript-based font detection can be a suitable alternative to Flash-based fingerprinting when the Flash plug-in isn't present or is disabled (for example, on mobile phones). Furthermore, unifying all fingerprinting methods could increase identification accuracy.

Linking Fingerprints with Users

We integrated Fybrid with Facebook to create iFybrid. We chose that social-media platform to determine whether we could associate a fingerprint with a user identity, thus creating a potential privacy leak.

Fingerprinting lets trackers create identifiers for browsers, not individuals. Linking browser fingerprints to individuals is complicated, if not impossible in some cases, for several reasons. For example, multiple users can share a device, a user can run multiple browsers on a single device, or multiple instances of a browser can run in different modes.

Despite that, we observed that integrating fingerprinting with a social-networking service not only can allow the linking of the browser's fingerprint to individuals but also can be a way to collect a list of browsers that a user interacts with over time. Because of social-networking services' popularity (more than one billion monthly active users[16]), they could offer the opportunity for such exploitation on a large scale if a well-designed fingerprinting tool was integrated with them.

We used the Facebook Login API for the integration. Web applications or websites that authenticate users through that API can access, at a minimum, the users' public profiles. Those that require more personal information about users and need to read or publish content on Facebook on their behalf must ask for specific permissions. In addition, applications that ask for more information than the users' public profile, friends list, and email address must go through a rigorous review before Facebook grants access to that information.

iFybrid asks for only the user's public profile and collects the user's ID, so it doesn't need to be reviewed by Facebook. Because this ID is an individual-specific identifier, IFybrid can learn the user's public-profile information, including the first name, last name, locale, gender, and so on.

In typical iFybrid fingerprinting data collection, a user

- visits the iFybrid Web application,
- logs on to the application through the Facebook Login API, using his user name and password, and
- clicks on the submit link to submit the browser fingerprint.

iFybrid sends the ID and collected data to a remote database. For privacy, the ID builder anonymizes the user's Facebook profile ID before sending it to remote storage, along with the collected fingerprint.

Instead of generating an anonymized identifier, a malicious tracker

ABOUT THE AUTHORS

AMIN FAIZ KHADEMI is a software developer at Ericsson Canada. His research interests include software engineering, software reliability and security, data mining, and machine learning. Faiz Khademi received an MSc in software reliability and security from Queen's University. Contact him at khademi@cs.queensu.ca.

MOHAMMAD ZULKERNINE is a Canada Research Chair in Software Dependability and an associate professor at the School of Computing, Queen's University. He leads the Queen's Reliable Software Technology research group. His research projects focus on software reliability and security. Zulkernine received a PhD in electrical and computer engineering from the University of Waterloo. He was a program cochair of the 5th International Congress on Secure Software Integration and Reliability Improvement; the 2012 IEEE Conference on Computers, Software, and Applications; and the 15th IEEE International Symposium on High-Assurance Systems Engineering. He's a senior member of IEEE and ACM and a licensed professional engineer in Ontario. Contact him at mzulker@cs.queensu.ca.

KOMMINIST WELDEMARIAM is a research scientist at IBM Research—Africa. His research areas are software engineering, modeling and verification of safety- and security-critical systems, security and reliability engineering, business processes, technologies for information and communication technologies and development applications, and analytics techniques for Web browser and mobile security. Weldemariam received a PhD in computer science and engineering from Trento University. He received the 2015 Next Einstein Fellowship and the Experienced Level and Managerial Recognition award from IBM. Contact him at k.weldemariam@ke.ibm.com.

could silently send the actual user ID, along with the fingerprint, to remote storage without the user's knowledge by using asynchronous requests (for example, `XMLHttpRequest` in JavaScript). Here's a hypothetical scenario for revealing the user's identity through integrating fingerprinting with Facebook:

1. A user logs on to website A using her Facebook credentials. So, A has access to her public information.
2. Website A fingerprints her browser, links her profile to the collected fingerprint, and stores the collected data in a database.
3. Website A shares the database with B, a third-party Web tracker.
4. B performs the same fingerprinting algorithm as A, so it generates the same fingerprint.
5. B identifies the user (without her consent), using the database that A provided.

In this way, trackers could create a rich repository of individuals and their browsers' fingerprints over time. This would also provide the opportunity for the trackers to become aware of the number of devices a user has and keep track of the browser changes to create more stable fingerprints over time.

We're employing the metrics we discovered to develop a detection tool. This tool will monitor website activity on the user's browser and look for fingerprinting patterns at run time. Because the metrics are system-related properties, we should be able to develop a suitable prevention technique to break the fingerprinting process and defend users against fingerprinters.

References

1. J.R. Mayer, "Any Person ... a Pamphleteer: Internet Anonymity in the Age of Web 2.0," undergraduate senior thesis, Woodrow Wilson School of Public and International Affairs, Princeton Univ., 2009.
2. K. Mowery and H. Shacham, "Pixel Perfect: Fingerprinting Canvas in HTML5," *Proc. Web 2.0 Security & Privacy 2014* (W2SP 14), 2012.
3. M. Mulazzani et al., "Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting," *Proc. Web 2.0 Workshop Security and Privacy* (W2SP 13), 2013; www.w2spconf.com/2013/papers/s2p1.pdf.
4. G. Acar, "Obfuscation for and against Device Fingerprinting," *Proc. Symp. Obfuscation*, 2014; http://obfuscationsymposium.org/wp-content/uploads/2014/02/gunes-position.pdf.
5. "Terms of Service," AddThis, July 2014; www.addthis.com/tos.
6. G. Acar et al., "The Web Never Forgets: Persistent Tracking Mechanisms in the Wild," *Proc. 2014 ACM SIGSAC Conf. Computer and Communications Security*, 2014, pp. 674–689.
7. C.J. Hoofnagle, J.M. Urban, and S. Li, "Privacy and Modern Advertising: Most US Internet Users Want 'Do Not Track' to Stop Collection of Data about Their Online Activities," *Proc. 2012 Amsterdam Privacy Conf.*, 2012.
8. A.M. McDonald and L.F. Cranor, "Beliefs and Behaviors: Internet Users? Understanding of Behavioral Advertising," *Proc. 2010 Research Conf. Communication, Information, and Internet Policy*, 2010.
9. J. Turow et al., "Americans Reject Tailored Advertising and Three Activities That Enable It," 2009; http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1478214.
10. G. Acar et al., "FPDetective: Dusting the Web for Fingerprinters," *Proc. 2013 ACM SIGSAC Conf. Computer & Communications Security*, 2013, pp. 1129–1140.
11. N. Nikiforakis et al., "Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting," *Proc. 2003 IEEE Symp. Security and Privacy* (SP 13), 2013, pp. 541–555.
12. P. Eckersley, "How Unique is Your Web Browser?," *Privacy Enhancing Technologies*, LNCS 6205, Springer, 2010, pp. 1–18.
13. *ActionScript 3.0 Reference for the Adobe Flash Platform*, Adobe, 2014; http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/system/Capabilities.html#propertySummary.
14. C.E. Shannon, "Prediction and Entropy of Printed English," *Bell System Technical J.*, vol. 30, no. 1, 1951, pp. 50–64.
15. I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *J. Machine Learning Research*, vol. 3, 2003, pp. 1157–1182.
16. E. Bakshy et al., "The Role of Social Networks in Information Diffusion," *Proc. 21st Int'l Conf. World Wide Web* (WWW 12), 2012, pp. 519–528.