Cookieless User Tracking with Browser Fingerprint

Traffic Protection Team

August, 2013

Agenda

- Background
- Problem Statement
- Data Collection
- Experiments
  - solution one
  - solution two
- Comparison of different browser identification techniques

Background

- Cookie plays an important role in the internet industry.
  - Advertising, personalization, fraud detection rely heavily on cookie-based user tracking
- Encountering with privacy and abuse problem, (third-party) cookie faces retention challenges.
  - Some modern browsers(e.g. Firefox, Safari) block third-party cookie by default.

Browser Fingerprinting

- A new technique to identify browser, using browser features such as user agent, plugins, fonts etc.
- It is reported that, by combining several browser features, it is able to identify browser uniquely [Eckersley 2010].

Problem Statement

- Uniqueness
  - Different browsers should have different fingerprints.
- Stability
  - The fingerprint should remain stable over time, even the browser upgrades.

Data Collection

- uid (cookie): used as baseline
- http accept header
- user agent
- plugins: extensions install in the browsers.
- system fonts
- timezone
- video (resolution)
- local storage enable or not

Method to get the Data

Fingerprinting Method	Features	Blockable	Accuracy
Protocol	user agent; etags;	easy	most accurate
Javascript	plugins; Timezone;Mime-type;	hard	less accurate
Flash	system fonts; LSO	normal	most accurate
Java Applet	system fonts	easy	least accurate

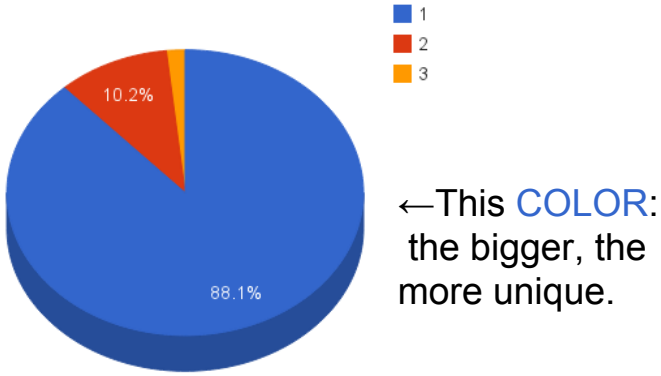
Table 1: Comparison of Fingerprinting Method

Uniqueness Evaluation

- Duplicated logs <uid, features>are removed.
- Fingerprints of different uid may collide.
  - the number of collision is calculated
  - the small the number is, the more unique the fingerprint is.

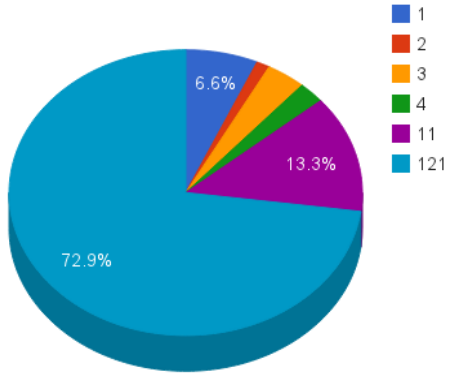
文本

clean_bfp

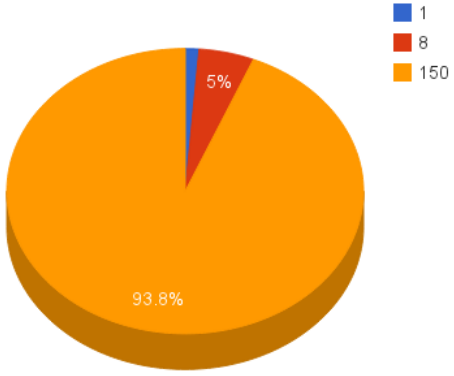


←This COLOR: the bigger, the more unique.

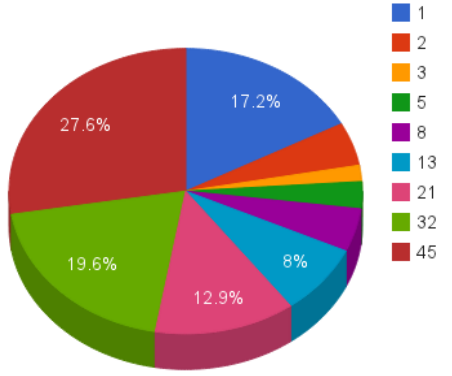
video



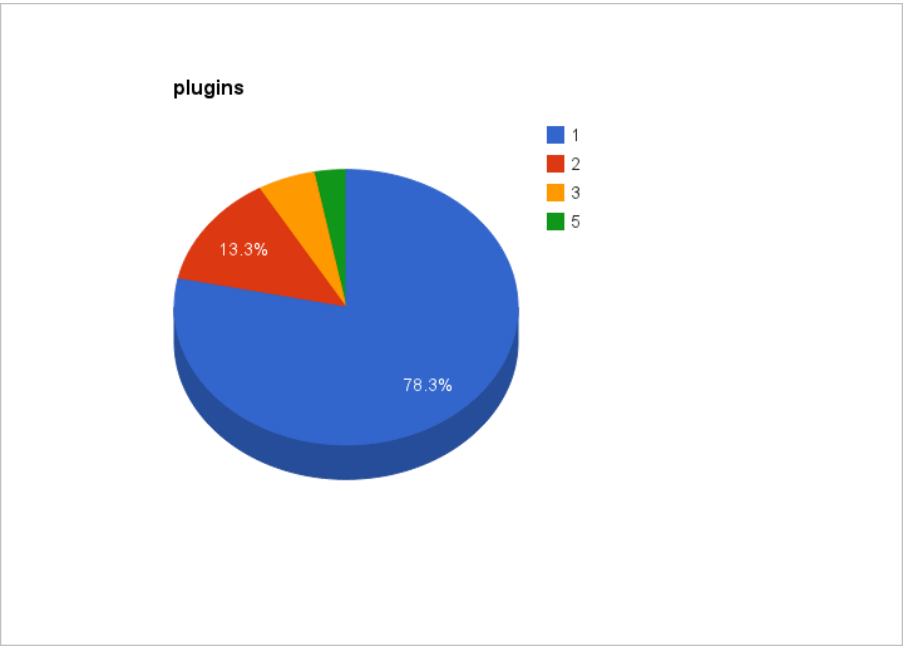
timezone

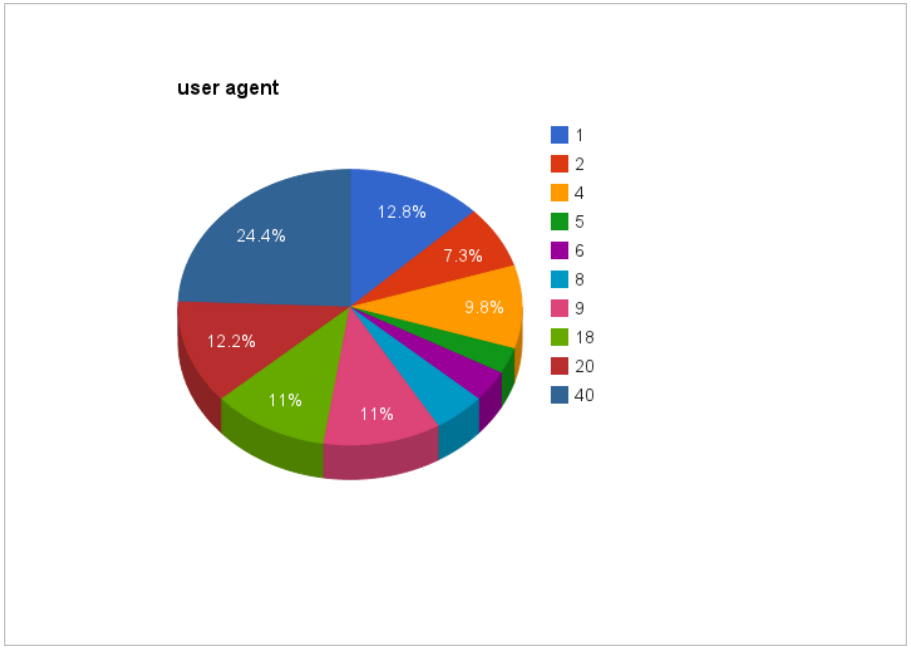


fonts



Uniqueness Distribution Diagram(1 / 2)

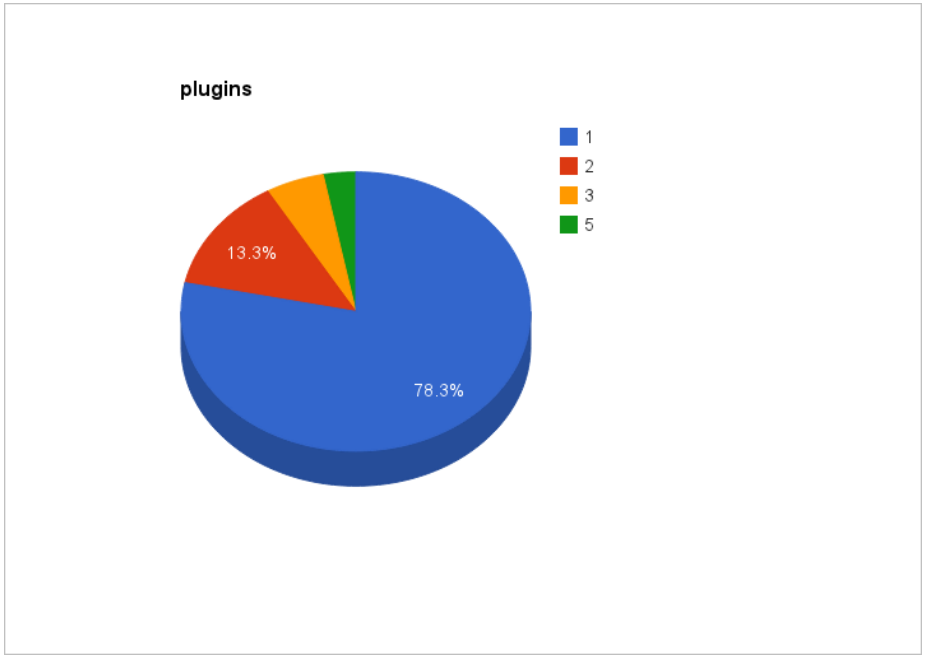




Uniqueness Distribution Diagram(2 / 2)

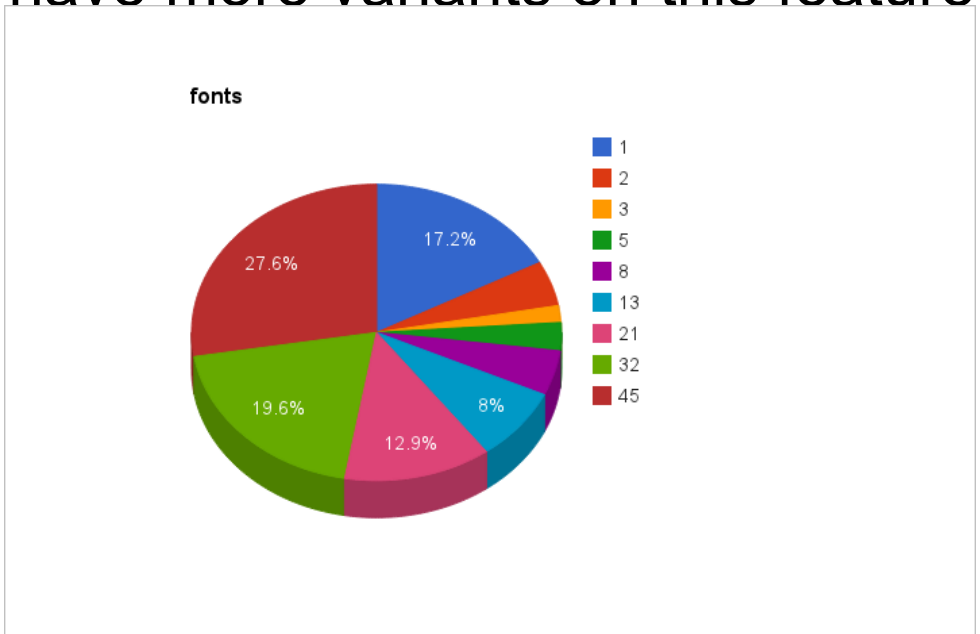
Uniqueness: Plugins

- 78.3% of 166 plugins fingerprint is unique, while 13.3% is collided with another one.
- Plugins show the greatest ability to distinguish visitors, mostly due to it’s diverse combination and highly personalized flavor.



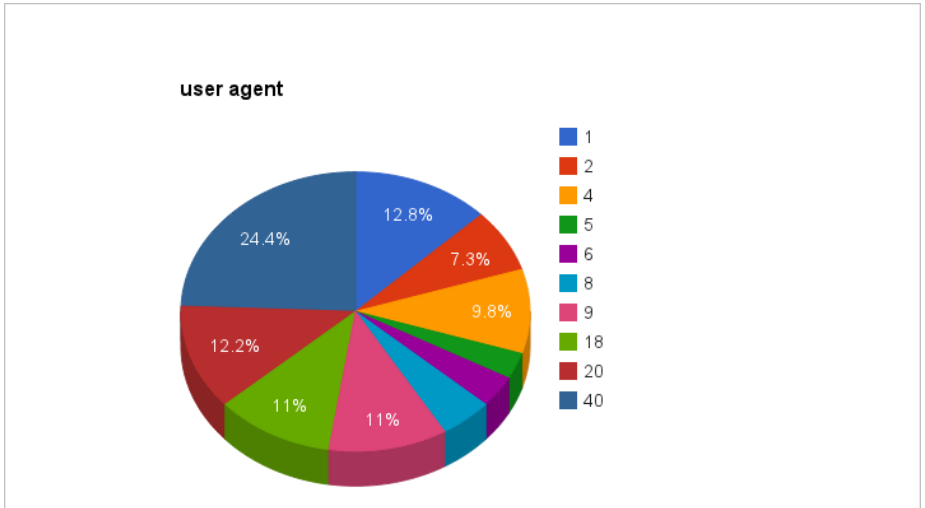
Uniqueness: Fonts

- Only 17.2% shows uniqueness.
- This can be explained as the samples were collected from a highly homogenized environment, most of the subjects were using rMBP with pre-setup.
- On the contrary, the subjects from real-world should be expected to have more variants on this feature dimension.



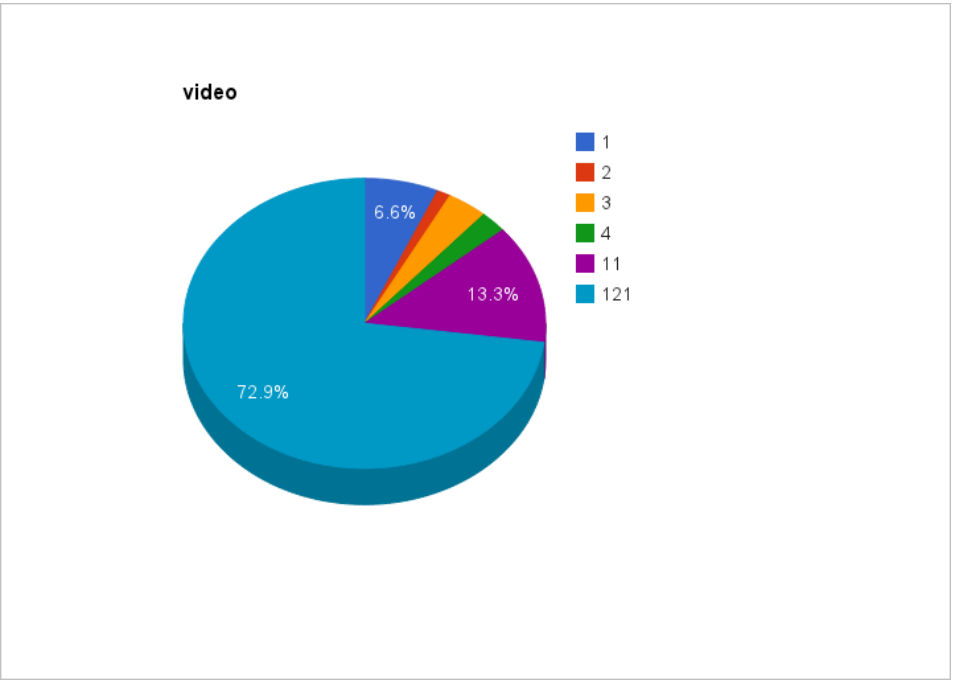
Uniqueness: User Agent

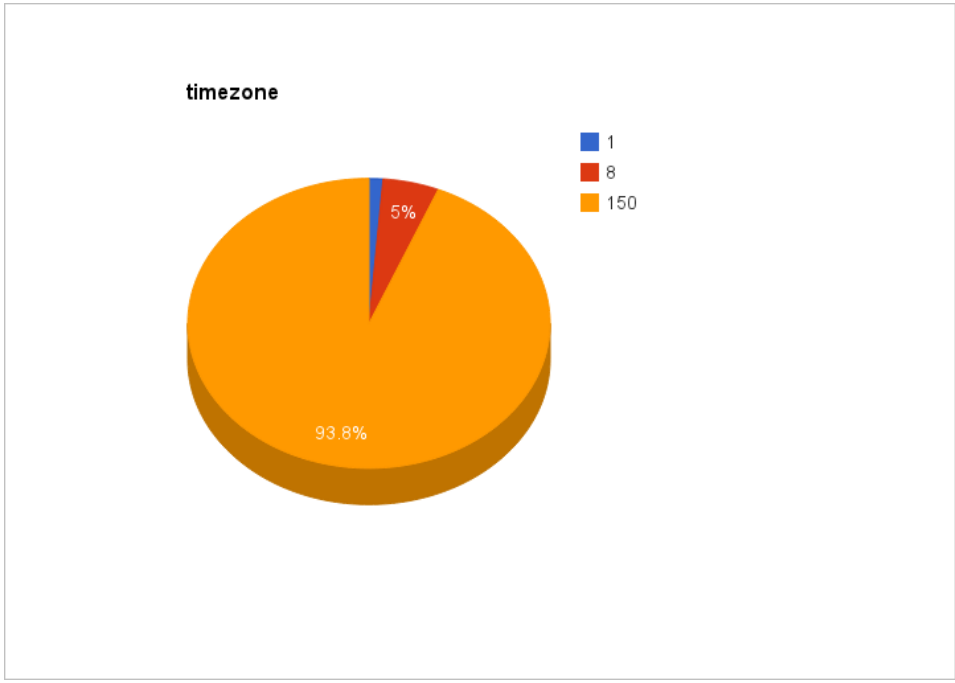
- 12.8% browsers is unique with regards to user agent.
- It might be due to the shorter string length(~107 chars), while the average string length of system fonts is ~5457.
- The highly homogenized environment also has impact on this dimension, even worse when automatic update mechanism of Chrome comes.



Uniqueness: Resolution, Timezone

- These two features have neither uniqueness percentage over 7%.





Uniqueness: Combine Them All

- Treat all dimension of features as string then simply concatenate them as one, we reached a fairly high percentage of 88.1%.
- This means that we can (almost) assert a browser is unique with the probability of nearly 90% without the aid of cookies!

Stability

- The browser fingerprint of one single user can vary over time.
- Identify the browser even if the bfp changes

Our Approach: Assumption

- Regarding uniqueness
  - The combination of various fingerprints can generate a high enough bar to distinguish different browsers.
- Regarding stability
  - The variation along the evolution of a single browser’s fingerprint is smaller than that between different browsers.

Our Approach: Framework(iterate)

```
iterate(incoming_log) {
    guessed_log = guess(incoming_log, previous_logs)

    if (incoming_log.cookies_id == guessed_log.cookies_id)
        right_num += 1;
    if (guess_log != NULL)
        return_num += 1;
    if (previous_logs.includes(incoming_log))
        should_return_num += 1;

    previous_logs[incoming_log.cookies_id] = incoming_log
}
```

Our Approach: Framework(guess)

```
guess(incoming_log, previous_logs) {
    candidates = [];
    for (log in previous_logs)
        if (at most one feature diverse between incoming_log and
log)
            candidates.push(log);
    if (candidates.size > 0) {
        best_candidate = select the best candidate;

        if (best_candidate is qualified by threshold)
            return best_candidate;
        else
            return NULL;
    }
    else
        return NULL;
}
```

Our Approach: Select the Best Candidate

- The “gestalt pattern matching” algorithm is used to measure match ratio between two strings.
- The ratio range is between 0 and 1, higher value implies more similar. The ratio between identical strings is 1.
- We select the best_candidate with the highest match ratio when there is discord.

Our Approach: Optimization On Candidate Selection

- Situation: What if there are several candidates with exactly the same match ratio? Which one to choose?
- Optimizations
  - Leverage the timestamp of the log.
  - Leverage the semantic of raw browser fingerprint.
  - ~~Leverage the browser upgrade delta-time distribution.~~

Optimization: Timestamp

- Every log essentially comes with a timestamp.
- Assumptions
  - if the candidates' bfp are all identical to that of current one, we select the latest candidate, since the likelihood to have the bfp kept invariant is much lower for a longer period.
  - if the candidates' bfp disagree with that of current one, we select the earliest candidate.

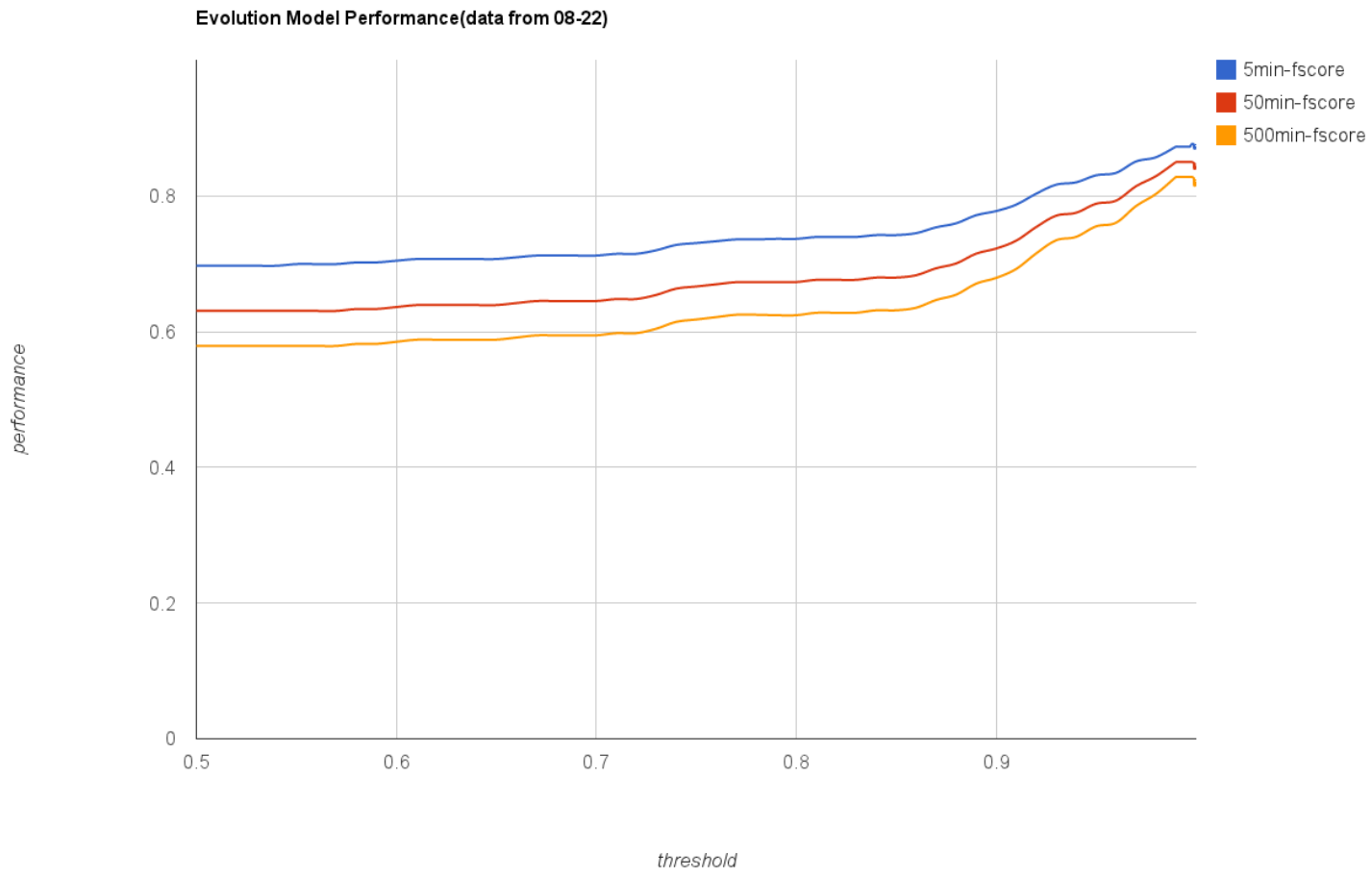
Optimization: Semantic of bfp

- Different fields of fingerprint have their own formats.
- A typical user-agent is like this:`Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.1500.95 Safari/537.36’
- We can do more that treat the whole string solely as a string.
- We leverage the monotonicity of the version number of each component. For example, if the above user-agent belongs to one of the candidates, and current log gets `Mozilla/4.9 ...’, that candidate will fall short. Whereas one with `Mozilla/5.1 …’ qualifies.

Our Approach: Performance

- Evaluate using standard measure matrix:
  - precision(`# of return_true/ # of all_return`)
  - recall(`# of return_true/ # of should_return`)
  - f-score(`harmonic mean of precision and recall`)
- Performance:
  - recall = 0.897
  - precision = 0.857
  - f-score = 0.877
- Model parameters:
  - match ratio threshold = 0.998
  - sample interval = 5 minutes.
- Achieve a fairly high performance on our rather limited size and quality dataset.

Our Approach: Performance(cont')



Solution Two

- Treat all the information as text, we use the webpage duplication detection method to identify the text.
- As in our case, we want to hash the original text to an integer, the similar browser fingerprints will have the same hash value. we can also find the most similar ones with minimum hamming distance.

LSH:Locality-sensitive hashing

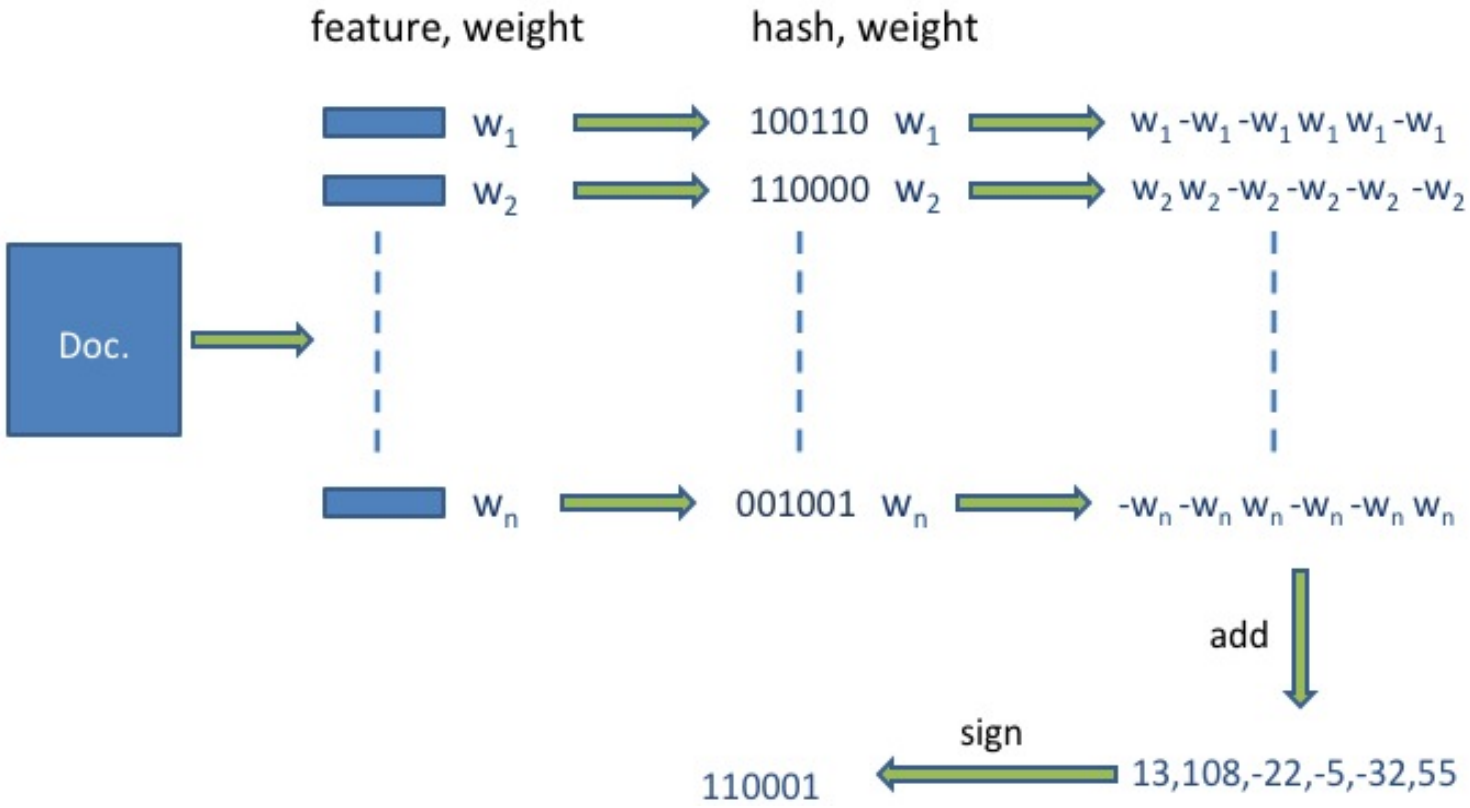
LSH is a method of performing probabilistic dimension reduction of high-dimensional data. The basic idea is to hash the input items so that similar items are mapped to the same buckets with high probability.(from wikipedia)

Simhash

Simhash is one type of locality sensitive hash, it was first proposed by Moses Charikar in “similarity estimation techniques from rounding algorithms”. Google use it to detect near-duplicates for web crawling.

Simhash

The procedure of SimHash:



Weight

tf–idf, term frequency–inverse document frequency, is a numerical statistic which reflects how important a word is to a document in a collection or corpus.

The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to control for the fact that some words are generally more common than others.

Experiment setup

- word separator in browser fingerprint:
  space,semi-colons,left bracket,right bracket

- word weight:
  tf(t,d) = log (f(t,d) + 1)

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

  weight=tf*idf

Experiment result

Uniqueness:

- 157 of 165 BFPs are mapped to one bfp-id, 3 of the 4 conflicts caused by one digit number difference. 1 of the 4 conflicts caused by a different java applet version.

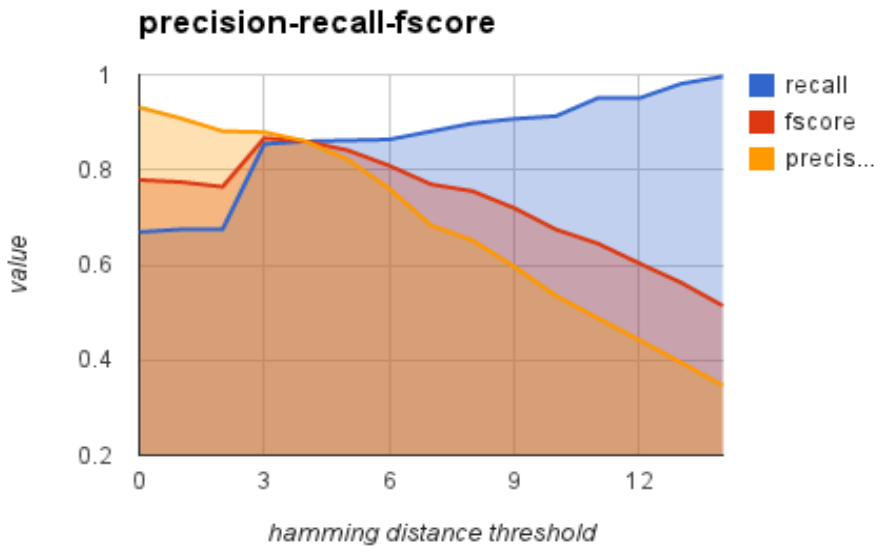
Stability:

- 161 of 160 cookies are mapped one bfp-id.
- 34 cookie has more than one distinct browser information, but 23 (67.6%) of them map to one bfp-id

Evaluation the model

Tracking User:Find nearest neighbor, if hamm distance is small than a threshold, then assign to the same id.

Sample Interval=5minutes



Others

文本

Informations that can be used as Browser Identification

Table 3: Key Characteristics of Information used as Browser Identification.

	HTTP Cookies	Flash cookies	HTML5 storage	etags	Browser fingerprint	Browser History	Browser Cache	Browser Specific Features
Storage	4KB	100KB by default	5Mb by default	none	none	none	none	none
Expiration	Session by default	Permanent by default	Permanent by default	browser cache lifetime	Permanent by default	Permanent by default	browser cache lifetime	Permanent by default
Source	In SQL file (firefox)	Stored outside the browser	In SQL file (Firefox)	Http Protocol	Http Protocol	Javascript	Javascript	Javascript
Access	Only by browser	By multiple browsers on same machine	Only by browser	Only by browser	Only by browser	Only by browser	Only by browser	Only by browser