Comparing PubSubHubbub and Twitter

Introduction to Information Retrieval
INF 141/ CS 121
Donald J. Patterson
Remember this?

Politeness when crawling

<table>
<thead>
<tr>
<th>Statistics for:</th>
<th>dp3.net</th>
</tr>
</thead>
</table>

**Summary**
- When:
  - Monthly history
  - Days of month
  - Days of week
  - Hours

**Who:**
- Countries
- Hosts
  - Full list
  - Last visit
  - Unresolved IP Address

**Robots/Spiders visitors**
- Full list
- Last visit

**Navigation:**
- Visits duration
- File type
  - Full list
  - Entry
  - Exit
- Operating Systems
  - Versions
  - Unknown
- Browsers
  - Versions
  - Unknown

**Referers:**
- Origin
  - Referring search engines
  - Referring sites
- Search
  - Search Keyphrases
  - Search Keywords

---

**Last Update:** 14 Jan 2008 - 02:59

<table>
<thead>
<tr>
<th>Reported period:</th>
<th>Year</th>
<th>2007</th>
<th>OK</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Robots/Spiders visitors</th>
<th>Hits</th>
<th>Bandwidth</th>
<th>Last visit</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 different robots</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Googlebot</td>
<td>1393868+104</td>
<td>5.11 GB</td>
<td>31 Dec 2007 - 23:50</td>
</tr>
<tr>
<td>Inktomi Slurp</td>
<td>36668+221</td>
<td>554.25 MB</td>
<td>31 Dec 2007 - 23:55</td>
</tr>
<tr>
<td>MSNBot</td>
<td>19522+2</td>
<td>699.90 MB</td>
<td>28 Dec 2007 - 08:01</td>
</tr>
<tr>
<td>Unknown robot (identified by 'crawl')</td>
<td>15949+13</td>
<td>89.34 MB</td>
<td>31 Dec 2007 - 22:24</td>
</tr>
<tr>
<td>AskJeeves</td>
<td>7016+1</td>
<td>106.29 MB</td>
<td>31 Dec 2007 - 23:49</td>
</tr>
<tr>
<td>Google AdSense</td>
<td>2701</td>
<td>100.26 MB</td>
<td>31 Dec 2007 - 22:10</td>
</tr>
<tr>
<td>psbot</td>
<td>2268+1</td>
<td>80.48 MB</td>
<td>31 Dec 2007 - 09:59</td>
</tr>
<tr>
<td>Unknown robot (identified by 'robot')</td>
<td>930+1</td>
<td>19.10 MB</td>
<td>31 Dec 2007 - 09:34</td>
</tr>
<tr>
<td>Turn It In</td>
<td>350+1</td>
<td>6.32 MB</td>
<td>03 Sep 2007 - 15:44</td>
</tr>
<tr>
<td>BaiDuSpider</td>
<td>300</td>
<td>10.22 MB</td>
<td>26 Nov 2007 - 07:32</td>
</tr>
<tr>
<td>GigaBot</td>
<td>243</td>
<td>5.27 MB</td>
<td>30 Dec 2007 - 05:06</td>
</tr>
<tr>
<td>Scooter</td>
<td>90+3</td>
<td>288.75 KB</td>
<td>27 Nov 2007 - 14:30</td>
</tr>
<tr>
<td>PhpDig</td>
<td>91</td>
<td>2.28 MB</td>
<td>21 Oct 2007 - 09:51</td>
</tr>
<tr>
<td>WISENutbot</td>
<td>76</td>
<td>1.94 MB</td>
<td>13 Jan 2007 - 14:04</td>
</tr>
<tr>
<td>Magpie</td>
<td>25</td>
<td>43.48 MB</td>
<td>24 Dec 2007 - 00:51</td>
</tr>
<tr>
<td>Unknown robot (identified by hit on 'robots.txt')</td>
<td>0+16</td>
<td>4.38 KB</td>
<td>14 Nov 2007 - 03:43</td>
</tr>
<tr>
<td>EchoO!</td>
<td>14</td>
<td>287.09 KB</td>
<td>27 Dec 2007 - 13:56</td>
</tr>
<tr>
<td>Internet Shinchakubin</td>
<td>13</td>
<td>385.03 KB</td>
<td>27 Nov 2007 - 15:23</td>
</tr>
<tr>
<td>BBot</td>
<td>10</td>
<td>146.35 KB</td>
<td>13 Jun 2007 - 15:17</td>
</tr>
<tr>
<td>arks</td>
<td>8</td>
<td>142.24 KB</td>
<td>27 Nov 2007 - 12:25</td>
</tr>
<tr>
<td>MSIECrawler</td>
<td>8</td>
<td>263.02 KB</td>
<td>26 Dec 2007 - 11:16</td>
</tr>
<tr>
<td>The Python Robot</td>
<td>5</td>
<td>429.91 KB</td>
<td>23 Nov 2007 - 08:04</td>
</tr>
</tbody>
</table>
Politeness?

[Image of a chart showing monthly history with columns for Month, Unique visitors, Number of visits, Pages, Hits, and Bandwidth. The data is presented in a table format for each month from January to December 2007, with total figures for the year as well.]
The problem

- Crawling incurs a huge waste of resources:
  - server bandwidth
  - crawler bandwidth
  - server CPU
  - electricity
  - hard drive space (for logs)
The problem
PubSubHubbub

- Basic idea is to eliminate polling sites for changes
- = eliminate crawling web sites blindly
PubSubHubbub

- A "topic" declares its PuSH server
- A topic is a URL
  - Could be a webpage
  - Could be an {Atom or RSS} XML file associated with a blog
  - Could be anything that is referenced by a URL

- A PuSH server is a "hub"
A PuSH server is a "hub"
- a 3rd party server
- manages your subscribers
- receives your updates
- rebroadcasts your updates
A “topic” can declare who its PuSH server:

- In the HTTP response
  - rel=hub (indicating URL of hub)
  - rel=self (indicating URL of self)
- If it is an html page, in can be in the header

```html
<link rel="hub" href="http://myhub.example.com/endpoint" />
```
PubSubHubbub

- A subscriber first gets the “topic” as always
- for example a news feed reader
- a webpage
• A subscriber then subscribes to the “hub”
• The hub then tells the subscriber when there is an update
• This avoids endless polling the feed by the subscriber for changes
• When there is an update, the publisher tells the hub
The hub gets the url and sends it to the subscribers
PubSubHubbub

Subscription flow

Draft 0.1
1. Subscriber polls Publisher's feed. The feed contains a forward link to the Hub.

Publisher

Subscriber

Give me your latest content for feed X, please.

I've delegated distribution of that content to this Hub:
2. Subscriber POSTs subscription request to the Hub. The request contains the endpoint URL where the Hub should POST new updates.
3. Hub POSTs to the endpoint URL to verify the request was authentic; Subscriber responds with confirmation to the Hub.

Publisher

Hub

Subscriber

Hey there! Did you really send this request?

Yup, that was really me, not a DoS attacker.
4. Publisher notifies Hub about updates by POSTing feed URLs to the Hub; Hub pulls the feed again to find new entries.

Publisher

Hub

Subscriber

I have new content for feed X for you!

Give me your latest content for feed X, please.

Here you go.
5. If the Publisher does not inform the Hub, the Hub will periodically poll the Publisher’s feed for new updates.

Publisher -> Hub

Publisher: Give me your latest content for feed X, please.

Hub: Here you go.

Hub -> Subscriber

Hub: Here you go.
6. When Hub receives new update to feed X, it POSTs the update to the Subscriber’s endpoint URL.

Publisher  

Hub  

Subscriber

New update to feed X! Here you go:
7. If feed X has multiple subscribers, the Hub sends updates to all of them. This reduces load on the Publisher.
The future is distributed: There will be big hubs, many small hubs, and tons of publishers and subscribers. Publishers, subscribers, and hubs may play multiple roles.
Requirements

Publishers
• Publish Atom feeds with their content
• Include Hub forwarding information in feeds (optional)
• POST updates to Subscribers and Hubs (optional)

Subscribers
• Parse the feeds forwarded by the Hub
• POST subscription requests to Publishers and Hubs & confirm them
• Provide endpoint URLs to which Publishers and Hubs can post updates

Hubs
• Receive subscription requests from Subscribers & verify them
• Provide endpoint URLs to which Publishers can POST updates
• Poll non-push capable Publishers to turn pull into push (optional)
• POST updates to Subscribers
• Let subscribers poll it (optional)
• Scale!
Summary of terms

- **Topic:**
  - An HTTP [RFC2616] resource URL. The unit to which one can subscribe to changes.

- **Hub ("the hub"):**
  - The server (URL [RFC3986]) which implements both sides of this protocol. Any hub MAY implement its own policies on who can use it.

- **Publisher:**
  - An owner of a topic. Notifies the hub when the topic feed has been updated. As in almost all pubsub systems, the publisher is unaware of the subscribers, if any. Other pubsub systems might call the publisher the "source".
Summary of terms

- **Subscriber:**
  - An entity (person or program) that wants to be notified of changes on a topic. The subscriber must be directly network-accessible and is identified by its Subscriber Callback URL.

- **Subscription:**
  - A unique relation to a topic by a subscriber that indicates it should receive updates for that topic. A subscription's unique key is the tuple (Topic URL, Subscriber Callback URL). Subscriptions may (at the hub's decision) have expiration times akin to DHCP leases which must be periodically renewed.
Summary of terms

- **Subscriber Callback URL:**
  - The URL [RFC3986] at which a subscriber wishes to receive notifications.

- **Event:**
  - An event that causes updates to multiple topics. For each event that happens (e.g. "Brad posted to the Linux Community."), multiple topics could be affected (e.g. "Brad posted." and "Linux community has new post"). Publisher events cause topics to be updated and the hub looks up all subscriptions for affected topics, sending out notifications to subscribers.

- **Notification:**
  - A payload describing how a topic's contents have changed, or the full updated content. Depending on the topic's content type, the difference (or "delta") may be computed by the hub and sent to all subscribers.
Summary of terms

- **Topic:**
  - An HTTP [RFC2616] resource URL. The unit to which one can subscribe to changes.

- **Hub ("the hub"):**
  - The server (URL [RFC3986]) which implements both sides of this protocol. Any hub MAY implement its own policies on who can use it.

- **Publisher:**
  - An owner of a topic. Notifies the hub when the topic
• Compare PubSubHubbub with Twitter
• Thinking of them both as content distribution mechanisms
• In contrast to PuSH
  • Twitter takes control of both publish and subscribe
  • Twitter becomes a black box for multicast
  • PuSH decouples publication from subscription
    • PuSH is much more scalable
  • Not as popular
There are actually three APIs
- REST interaction
- REST search
- Stream-based
  - The Streaming API provides low-latency high-volume access to Tweets.
Twitter API

• You cannot make unlimited calls, follow requests, updates or direct message
• API usage is rate limited.
• There are limits on the number of follow requests, updates and direct messages you can make in a single day.
Twitter API

- The API is entirely HTTP-based
  - Methods to retrieve data from the Twitter API require a GET request.
  - Methods that submit, change, or destroy data require a POST.
  - API Methods that require a particular HTTP method will return an error if you do not make your request with the correct method.
Twitter API

- The API is a RESTful resource
- JSON format only as of V1.1
• Parameters have certain expectations
• Some API methods take optional or requisite parameters.
• Parameter values should be converted to UTF-8 and URL encoded.
• The page parameter begins at 1, not 0.
• There are many libraries available
Twitter API

• Examples:
  • Get your data in RSS format
  • https://dev.twitter.com/docs/api/1.1/get/statuses/home_timeline
Twitter API

- Examples:
  - Streaming API
  - https://dev.twitter.com/docs/api/1.1/get/statuses/sample