NAME
sc_hoiho — Holistic Orthography of Internet Hostname Observations

SYNOPSIS

DESCRIPTION
The sc_hoiho utility learns regular expressions that extract router names from hostnames, based on a set of training routers whose interfaces were inferred to be aliases by other alias resolution techniques. The technique is described in the paper "Learning to Extract Router Names from Hostnames" published in the ACM Internet Measurement Conference 2019. The supported options to sc_hoiho are as follows:

-6 specifies that the input training set contains IPv6 addresses, and not IPv4 addresses.

-d dump specifies the dump ID to use to analyze the collected data. Currently, ID values 1 (working-set), 2 (routers), and 3 (best-regex) are valid, which dump the working set of regexes for each suffix, the result of applying the best regex to the training data, and the best regex, respectively.

-D domain specifies the domain suffix that sc_hoiho should operate on. If no suffix is specified, then sc_hoiho operates on all suffixes.

-O option allows the behavior of sc_hoiho to be further tailored. The current choices for this option are:
  - application: Show the outcome of applying the regular expression to the application set.
  - debug: output significant volumes of debugging information. Debugging information is only printed when a sc_hoiho is using a single thread.
  - json: Output inferences using json format. This option is only applicable on dump ID 2.
  - nojit: Do not use pcre(3) or pcre2(3) just in time complication to improve regex performance.
  - norefine: do not do any refinement of regular expressions.
  - norefine-tp, refine-tp: do not, or do, execute the phase that builds literals into components that extract a string from the hostnames. This phase is described in phase 5.2 of the paper.
  - norefine-fne, refine-fne: do not, or do, execute the phase that embeds literals in a regex component that separates hostnames from their training routers. This phase is described in section 5.3 of the paper.
  - norefine-class, refine-class: do not, or do, execute the phase that embeds character classes in regular expressions. This phase is described in section 5.4 of the paper.
  - norefine-fnu, refine-fnu: do not, or do, execute the phase that builds additional regexes to use as a companion regex with another regex that did not cluster all interfaces on a router. This phase is described in section 5.5 of the paper.
  - norefine-sets, refine-sets: do not, or do, execute the phase that builds naming convention sets using the regexes in the working set. This phase is described in section 5.6 of the paper.
  - norefine-ip, refine-ip: do not, or do, execute the phase that builds regexes to filter hostnames with IP addresses in them, where the IP address would be part of an extracted name. This phase is described in section 5.7 of the paper.
  - norefine-fp, refine-fp: do not, or do, execute the phase that builds regexes to filter hostnames that, if matched, would result in false positives according to the training data. This phase is described in phase 5.7 of the paper.
- **nothin:** do not, or do, remove any redundant regexes at the end of each phase, as described in section 4.5 of the paper.
- **nothin-matchc, thin-matchc:** do not, or do, remove any regexes that do not meet the minimum number of matches to be considered capturing a convention. The conditions are described in section 4.5 of the paper.
- **nothin-same, thin-same:** do not, or do, remove redundant regexes that perform the same clustering.
- **nothin-mask, thin-mask:** do not, or do, remove redundant regexes whose clustering is entirely contained in another regex, with no additional false positives.
- **randindex:** compute the Rand Index metric on the clustering of hostnames according to the training data.

- `-regex` specifies the name of a file containing a working set of regexes, or a naming convention, to apply.

- `-thread` specifies the number of threads to use in the threadpool. By default, **sc_hoiho** will determine the number of processors online, and use all of them.

**EXAMPLES**

Given a set of routers in a file named routers.txt, and a copy of public_suffix_list.dat obtained from the Mozilla Foundation’s [https://publicsuffix.org/list/](https://publicsuffix.org/list/) website:

```
192.0.2.1  esr1-ge-5-0-0.jfk2.example.net
192.0.2.10 esr1-ge-5-0-6.jfk2.example.net
192.0.31.60
192.0.2.2  esr2-xe-4-0-0.pax.example.net
192.0.2.5  esr2-xe-4-0-1.pax.example.net
192.0.31.8
192.0.2.6  das1-v3005.nj2.example.net
192.0.2.9  das1-v3006.nj2.example.net
192.0.2.44 44.2.0.192.example.net
192.0.2.13 esr1-xe-4-0-0.pax.example.net
192.0.2.17 esr1-xe-4-0-1.pax.example.net
192.0.2.21 esr1-xe-4-0-1.pax.example.net
```

Then the following command will build a base set of regular expressions, as described in section 5.1 of the paper, and output the working set of regexes inferred for each suffix at the end of that phase.

```
sc_hoiho -d working-set -O norefine public_suffix_list.dat routers.txt
```

To obtain the best selected regular expression for example.net, use:

```
sc_hoiho -d best-regex -D example.net public_suffix_list.dat routers.txt
```

To examine how the best regex applies to the training data for example.net, use:

```
sc_hoiho -d routers -D example.net public_suffix_list.dat routers.txt
```

To examine how the best regex applies to the training data, as well as interfaces in the application set, use:

```
sc_hoiho -d routers -D example.net -O application public_suffix_list.dat routers.txt
```

To see the working set of regexes built after embedding literals in captures for example.net, use:
sc_hoiho -d working-set -D example.net -O norefine -O refine-tp public_suffix_list.dat routers.txt
To see how a manually-derived regex behaves against the training data, use:
sc_hoiho -D example.net -r "ˆ([a-z]+\d+)-.+\([a-z]\d+\).example\net\$"
-d routers -O norefine public_suffix_list.dat routers.txt

HINTS
sc_hoiho can take a long time to run, depending on the training set involved. One option to breaking up
the runtime (but not reducing it) is to capture the output from one phase, and then use that as input to the next
phase. For example, to run the first three phases:
sc_hoiho -d working-set -O norefine public_suffix_list.dat routers.txt >phase-1.re
sc_hoiho -d working-set -O norefine -O refine-tp -r phase-1.re public_suffix_list.dat routers.txt >phase-2.re
sc_hoiho -d working-set -O norefine -O refine-fne -r phase-2.re public_suffix_list.dat routers.txt >phase-3.re

NOTES
sc_hoiho follows the format of the hostnames files stored in CAIDA’s Internet Topology Data Kit (ITDK)
which stores hostnames in lower-case, and escapes characters that do not form part of the DNS’s alphabet
(A-Z, a-z, - and .) as a hexadecimal escaped string. For example, if a hostname contains an underscore char-
acter, such as foo_bar, then encode the underscore using the hexadecimal dictionary in ascii(7) as follows:
foo\x5fbar.

SEE ALSO
pcre(3), pcre2(3), sc_ally(1), sc_radargun(1), sc_speedtrap(1),
M. Luckie, B. Huffaker, and k claffy, Learning to Extract Router Names from Hostnames, Proc. ACM
Internet Measurement Conference 2019.
Mozilla Foundation, Public Suffix List, https://publicsuffix.org/list/.
Center for Applied Internet Data Analysis (CAIDA), Macroscopic Internet Topology Data Kit (ITDK),
N. Spring, R. Mahajan, and D. Wetherall, Measuring ISP topologies with Rocketfuel, Proc. ACM SIGCOMM
2002.
A. Bender, R. Sherwood, and N. Spring, Fixing Ally’s growing pains with velocity modeling, Proc.
K. Keys, Y. Hyun, M. Luckie, and k claffy, Internet-Scale IPv4 Alias Resolution with MIDAR, IEEE/ACM
Transactions on Networking 2013.
ACM/SIGCOMM Internet Measurement Conference 2013.

AUTHORS
sc_hoiho was written by Matthew Luckie.