F (v2)

A complete system integration of stream-based IP flow-record querier

Vaibhav Bajpai

(Interim) Masters Thesis Presentation

Computer Networks and Distributed Systems
School of Engineering and Sciences
Jacobs University Bremen
Bremen, Germany

April 2012

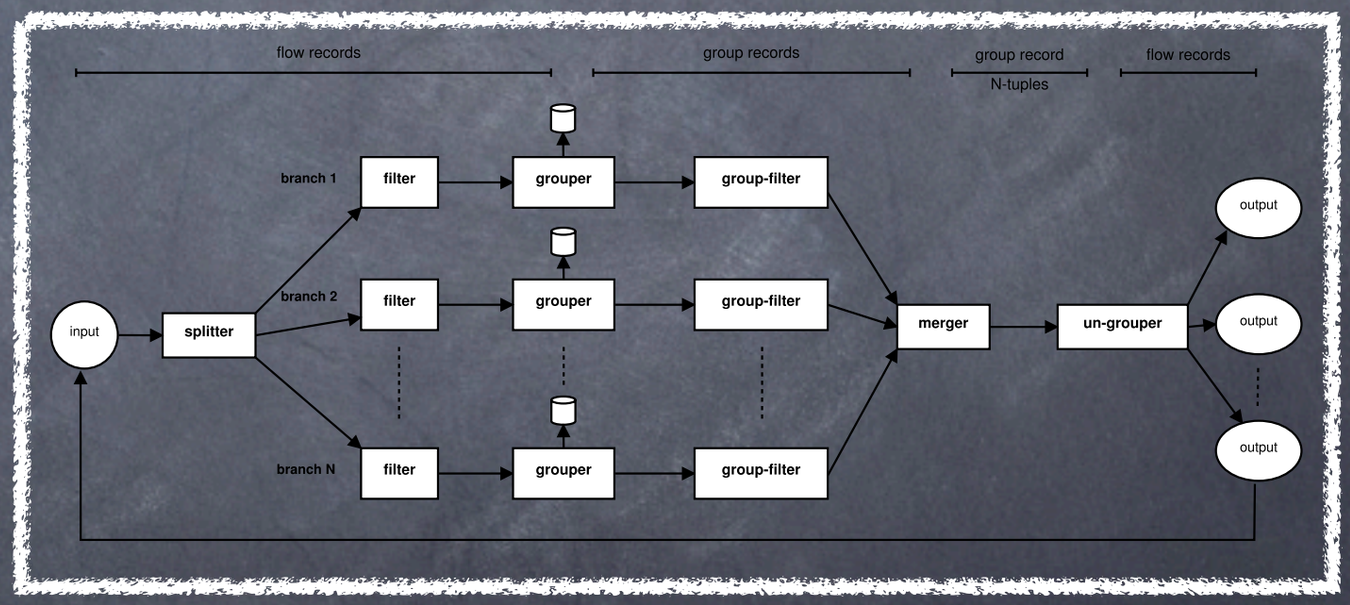
Overview

Introduction

Network Flow-Query Language (NFQL)

an in-house flow-query language, designed to cap flow-record traces to their full potential.

- filter flow-records
- combine them into groups
- apply relative filters
- aggregate their flow-fields
- invoke allen interval algebra



F (previously Flowy)

prototype implementation of NFQL.

Evolution

- Flowy (Python) [1, 2]

parser

  - PLY for parsing and validating the flowquery
  - flow-record storage using PyTables and HDF
  - deep copy of flow-records
  - deep nested loops

- Flowy Improvements using Map/Reduce [3]
  - investigative, but theoretical

- Flowy → F (C) [4]

engine

  - read flow-records into memory
  - rewrite of the execution pipeline in C (not functional)
  - efficient rule processing with dedicated function pointers
  - reduced grouper complexity using qsort and bsearch

Engine Concerns

- flow query hardcoded in pipeline structs
  - functions assume specific uintX_t offsets
- pipeline stages
  - numerous grouper segfaults
  - no group filter
  - commented out merger (segfaults when uncommented)
  - no ungrouper
- minor issues
  - code dependent on GNU99 extensions
  - some headers missing include guards
  - unused extraneous source files and headers

v0.1

it works!

Preliminary Improvements

- painless single step parser installation [1]

```
$ pip install -r requirements.txt
```

- reverse-engineered parser to generate UML [2]

*depends on pylint and graphVIZ

```
$ pyreverse -o png -p parser parser/
```

- reverse-engineered engine to generate UML [3]

*depends on graphVIZ

```
$ doxygen Doxyfile
```

[1] http://goo.gl/yTCTZ

[2] http://goo.gl/HTpxN

[3] http://goo.gl/SXjbv

Preliminary Improvements

- multiple verbosity levels in the engine.

```
$ bin/flowy-engine $PARAMS --verbose=$LEVEL
```

  - --verbose=1: results of each stage
  - --verbose=2: intermediate results of each stage
  - --verbose=3: original flow-record trace

- command line parsing using getopt_long(...)
  - prints usage on insufficient arguments
  - tracks invalid options
  - tracks invalid verbosity levels
- misc
  - conditional compilation macros for each stage
  - consistency checks before reading flow-records in memory

Grouper

Grouper Internals

```
grouper g1 {
  srcIP = srcIP
  dstIP = dstIP
}
```

- × näive approach $O(n^2)$
- × smart approach O(n) using a HT

SrcIPaddress
209.132.180.131
209.132.180.131
131.155.140.135
128.30.52.37
128.30.52.95
195.37.77.138
195.37.77.138
195.37.77.138
195.37.77.138
93.184.220.20
93.184.220.20
93.184.220.20

grouper operators

- equalTO
- nequalTO
- lThan
- gThan
- lThanequalTO
- gThanequalTO

SrcIPaddress
209.132.180.131
209.132.180.131
131.155.140.135
128.30.52.37
128.30.52.95
195.37.77.138
195.37.77.138
195.37.77.138
195.37.77.138
93.184.220.20
93.184.220.20
93.184.220.20

Grouper Internals

```
grouper g1 {
  srcIP = srcIP
  dstIP = dstIP
}
```

- sort : O(n*lg(n))
- remove duplicates : O(n)

preprocessing

- for each item : O(n*lg(k))
  do binary search

SrcIPaddress
209.132.180.131
209.132.180.131
131.155.140.135
128.30.52.37
128.30.52.95
195.37.77.138
195.37.77.138
195.37.77.138
195.37.77.138
93.184.220.20
93.184.220.20
93.184.220.20

grouper operators

- equalTO
- nequalTO
- lThan
- gThan
- lThanequalTO
- gThanequalTO

unique recordset

SrcIPaddress
93.184.220.20
128.30.52.37
128.30.52.95
131.155.140.135
195.37.77.138
209.132.180.131

Grouper Features

- aggregations as separate (cooked) v5 record.

```
No. of Groups: 32 (Aggregations)

...    SrcIPaddress    ...      DstIPaddress      OR(Fl)    Sum(Octets)

...    4.23.48.126     ...      192.168.0.135     3         81034
...    8.12.214.126    ...      192.168.0.135     2         5065
...    80.157.170.88   ...      192.168.0.135     6         18025
```

  - ignores aggregations on fields touched by filter/grouper
  - returns a SET for aggregation on uncommon fields

- club records into 1 group if no grouper rules defined

```
No. of Groups: 1 (Aggregations)

...    Sum(Octets)

...    2356654
```

Merger

Merger Internals

merger pseudocode:

```
get_module_output_stream(module m) {
  (branch_1, branch_2, ..., branch_n) = get_input_branches(m);
  for each g_1 in group_records(branch_1)
    for each g_2 in group_records(branch_2)
      ...
        ...
          for each g_n in group_records(branch_n)
            if match(g_1, g_2, ..., g_n, rules(m))
              output.add(g_1, g_2,..., g_n);
  return output;
}
```

nesting level NOT known until RUNTIME

iterate over all the possible permutations of the group tuples

```
/* initialize the iterator */
struct permut_iter *iter = iter_init(binfo_set, num_branches);

/* iterate over all permutations */
while(iter_next(iter)) {...}

/* free the iterator */
iter_destroy(iter);
```

input: (b1, b2, b3) = (3, 2, 2)
output: 12 group tuples, that are checked for a match

Removing Assumptions

- flexible stages (no uintX_t assumptions)

```
-    { 0, trace_data->offsets.srcaddr, aggr_static_uint32_t },
-    { 0, trace_data->offsets.dPkts, aggr_sum_uint32_t },
+    { 0, trace_data->offsets.srcaddr, RULE_STATIC | RULE_S1_32, NULL },
+    { 0, trace_data->offsets.dPkts, RULE_SUM | RULE_S1_32, NULL },
```

```
switch (op) {
        ...
        case RULE_SUM | RULE_S1_32:
          X.func = X_uint32_t;
          break;
        ...
```

- grouper
- grouper aggregations
- group filter
- merger

- performance recap
  - filter (worst) — O(n)
  - grouper (average) — O(n*lg(n)) + O(n) + O(n*lg(k))
  - grouper aggr (worst) — O(n)
  - group filter (worst) — O(n)
  - merger (worst) — $O(n^m)$ where m = num(branches)
  - ungrouper (worst) — O(n)

Summary

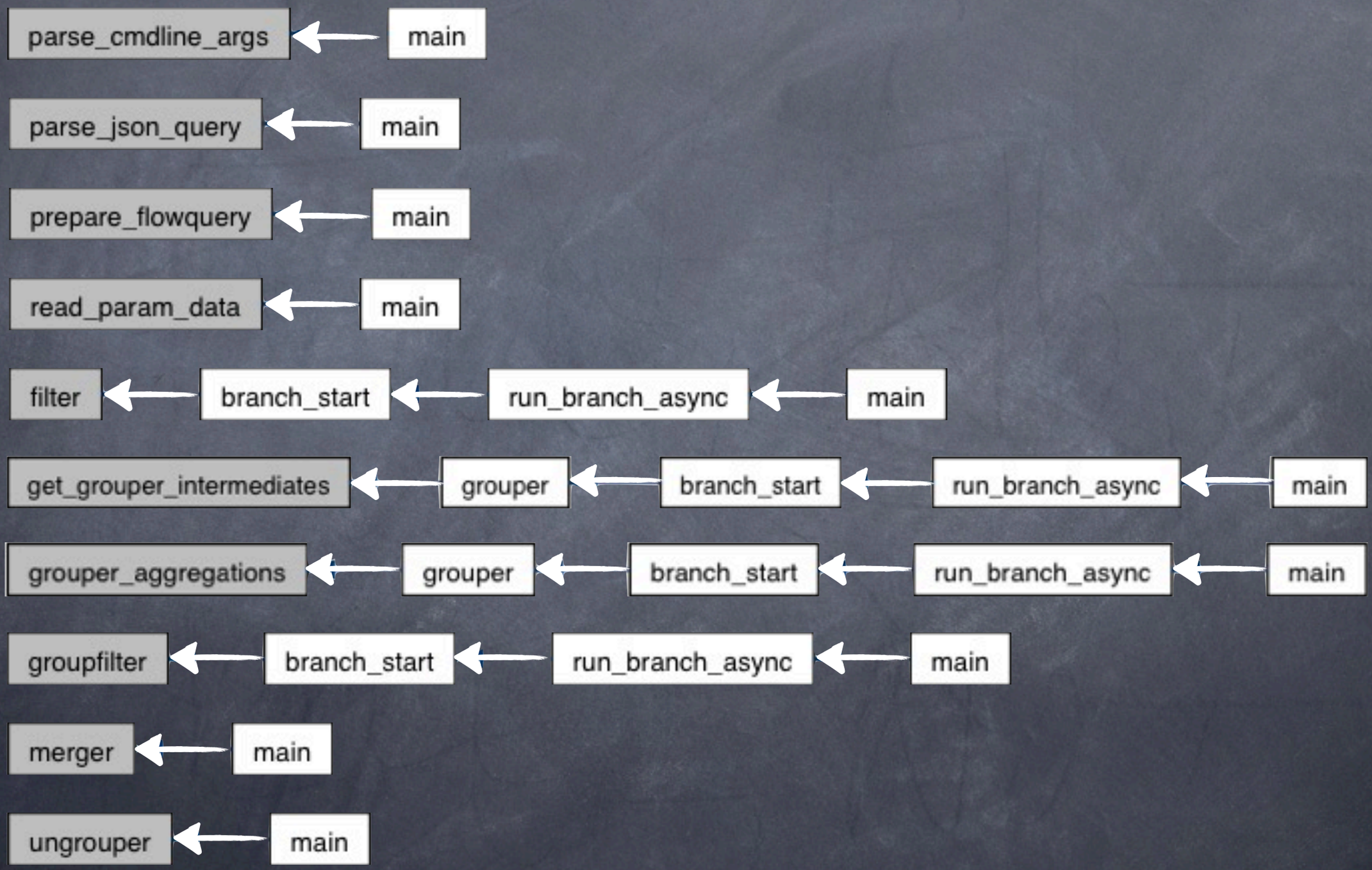
v0.1

- reverse engineered parser to generate UML.
- single step installation of the python parser using pip
- doxygen documentation of the engine
- replaced GNU99 extensions dependent code with c99
- resolved numerous segfaults in grouper and merger
- group aggregations as a separate (cooked) v5 record
- flexible group aggregations with no uintX_t assumptions
- first ever group filter implementation
- cleaner src/ directory structure layout
- multiple verbosity levels in the engine
- first-ever merger implementation
- flexible filters and group filters with no uintX_t assumptions
- first-ever ungrouper implementation

v0.2

it is robust!

Complete Engine Refactor



Complete Engine Refactor

- all rules are clubbed in X_ruleset
- each stage returns X_result
- rulesets are dealloc as soon as X returns

```
struct flowquery {
  size_t                          num_branches;
  struct branch**                 branchset;

  size_t                          num_merger_rules;
  struct merger_rule**            merger_ruleset;

  struct merger_result*           merger_result;
  struct ungrouper_result*        ungrouper_result;
};
```

```
struct filter_result {
  size_t                          num_filtered_records;
  char**                          filtered_recordset;
};
```

```
struct grouper_result {
  size_t                          num_unique_records;
  char**                          sorted_recordset;
  char**                          unique_recordset;

  size_t                          num_groups;
  struct group**                  groupset;
};
```

```
struct groupfilter_result {
  size_t                          num_filtered_groups;
  struct group**                  filtered_groupset;
};
```

```
struct merger_result {
  size_t                          num_group_tuples;
  size_t                          total_num_group_tuples;
  struct group***                 group_tuples;
};
```

```
struct ungrouper_result {
  size_t                          num_streams;
  struct stream**                 streamset;
};
```

```
struct branch {

  /* ------------------------------------------------------------------*/
  /*                              inputs                               */
  /* ------------------------------------------------------------------*/

  …

  size_t                          num_filter_rules;
  size_t                          num_grouper_rules;
  size_t                          num_aggr_rules;
  size_t                          num_gfilter_rules;

  struct filter_rule**            filter_ruleset;
  struct grouper_rule**           grouper_ruleset;
  struct aggr_rule**              aggr_ruleset;
  struct gfilter_rule**           gfilter_ruleset;

  /* ------------------------------------------------------------------*/



  /* ------------------------------------------------------------------*/
  /*                              output                               */
  /* ------------------------------------------------------------------*/

  struct filter_result*           filter_result;
  struct grouper_result*          grouper_result;
  struct groupfilter_result*      gfilter_result;

  /* ------------------------------------------------------------------*/

};
```

Complete Engine Profiling

before:

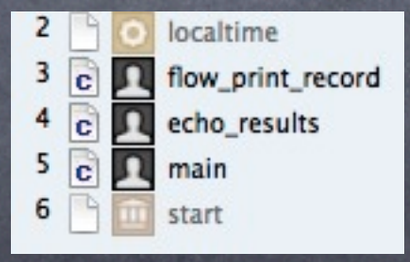
```
$ git checkout v0.1
$ valgrind bin/flowy-engine $TRACE $QUERY

==19000== HEAP SUMMARY:
==19000==     in use at exit: 131,519 bytes in 1,182 blocks
==19000==   total heap usage: 2,609 allocs, 1,427 frees, 1,631,199
bytes allocated
==19000==
==19000== LEAK SUMMARY:
==19000==    definitely lost: 6,912 bytes in 472 blocks
==19000==    indirectly lost: 0 bytes in 0 blocks
==19000==      possibly lost: 0 bytes in 0 blocks
==19000==    still reachable: 124,607 bytes in 710 blocks
==19000==         suppressed: 0 bytes in 0 blocks
...
```

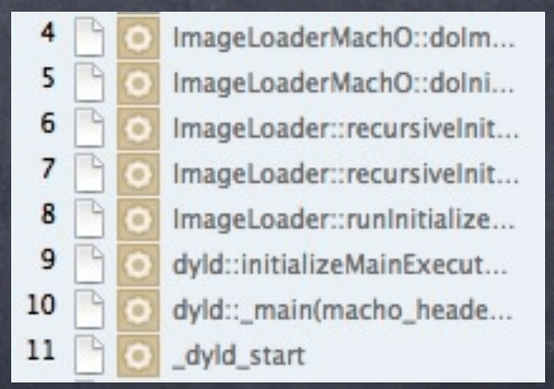
after:

```
$ git checkout master
$ valgrind bin/flowy-engine $TRACE $QUERY

==19164== HEAP SUMMARY:
==19164==     in use at exit: 20,228 bytes in 37 blocks
==19164==   total heap usage: 3,646 allocs, 3,609 frees, 1,647,767
bytes allocated
==19164==
==19164== LEAK SUMMARY:
==19164==    definitely lost: 0 bytes in 0 blocks
==19164==    indirectly lost: 0 bytes in 0 blocks
==19164==      possibly lost: 0 bytes in 0 blocks
==19164==    still reachable: 20,228 bytes in 37 blocks
==19164==         suppressed: 0 bytes in 0 blocks
...
```



libsystem_c
(10 mallocs)



dyld
(81 mallocs)

Issues Closed

lazy rule->func assignments

```
assign_filter_func(struct filter_rule* const frule) {…}

assign_grouper_func(struct grouper_rule* const grule) {…}

assign_aggr_func(struct aggr_rule* const arule) {…}

assign_gfilter_func(struct gfilter_rule* const gfrule) {…}

assign_merger_func(struct merger_rule* const mrule) {…}
```

greedily dealloc non-filtered records in O(n) before merger

```
 struct ft_data {
+  struct record**                 recordset;
+  int                             num_records;
 };

 struct record {
+  char*                           record;
+  bool                            if_filtered;
 };
```

flexible grouper with no uintX_t assumptions

```
struct grouper_type* get_gtype(uint64_t op) {
  ...
  switch (op) {

    case RULE_S2_8:
      gtype->qsort_comp = comp_uint8_t;
      gtype->bsearch = bsearch_uint8_t;
      gtype->alloc_uniqresult = alloc_uniqresult_uint8_t;
      gtype->get_uniq_record = get_uniq_record_uint8_t;
      gtype->dealloc_uniqresult = dealloc_uniqresult_uint8_t;

      break;
    case RULE_S2_16:
      ...
      break;

    case RULE_S2_32:
      ...
      break;

    case RULE_S2_64:
      ...
      break;
  }
  return gtype;
}
```

Summary

v0.2

- complete engine refactor
- complete engine profiling (no memory leaks)

- greedy dealloc non-filtered records in O(n) before merger(...)
- all filtered records make 1 group with NO grouping rule
- aggregation on common fields hit by filter/grouper is ignored
- no uintX_t assumption for field offsets anywhere.
- each stage functions receive bare minimum parameters
- func parameters are safe using [const] ptr and ptr to [const]
- lazy rule->func assignment only when the stage is hit

v0.3

it is flexible!

Features

- read multiple traces from stdin

```
$ flow-cat ... | flowy-engine -
```

- pipeline stages can be skipped
  - each stage is smart to skip itself if NO rules are defined for it.
- stages only proceed when the previous returned results
- graceful exits on failure
  - glibc backtrace(...) to print the back trace on errExit(...)
  - gracefully exiting when arguments cannot be parsed

Query at Runtime

- engine now reads the JSON query at runtime
  - number of branches
  - number of rules in each stage
  - branchset as a JSON array
  - rulesets as a JSON array

- JSON query is generated using python script build-query.py

```
class FilterRule: ...
class GrouperRule: ...
class AggregationRule: ...
class GroupFilterRule: ...
class MergerRule: ...

branchset = []
branchset.append({'filter': filter,
                  'grouper': grouper,
                  'aggregation': aggregation,
                  'groupfilter': groupfilter,
                 })

query = {'num_branches': len(branchset),
         'branchset': branchset,
         'merger': merger}
```

```
{
  "branchset": [
    "num_branches": 2
    {
      "filter": {
        "num_rules": 2,
        "ruleset": [...]
      },
      "grouper": {
        "num_rules": 2,
        "ruleset": [...]
      }
      "aggregation": {
        "num_rules": 4,
        "ruleset": [...]
      },
      "groupfilter": {
        "num_rules": 1,
        "ruleset": [...]
      },
    },
  {
    ...
  }
  ],
  "merger": {
    "num_rules": 2,
    "ruleset": [...]
  },
}
```

Summary

v0.3

- number of branches and rules now come from JSON
- each ruleset of the stage now comes from JSON
- build-query.py to generate a JSON query
- flow-cat ... | flowy-engine $QUERY -

- glibc backtrace(...) to print the back trace on errExit(...)
- gracefully exiting when trace cannot be read
- gracefully exiting when JSON query cannot be parsed
- each stage proceeds only when previous returned results
- pipeline stages can now be skipped (need to test)

Conclusions

Future Work

tasks

- make parser spit the JSON query using build-query.py
- CMake build process
- enable allen interval operations on group metadata.
- remove duplicate records after ungrouping
- enable multiple modules in grouper and merger
- enable OR in filter rules
- enable SET operations on group filter
- validate the engine robustness with different queries.
- benchmark against Flowy and flow-tools/nfdump
- cross-check code compilation on GNU/Linux

goals

- IPFIX support
- hash tables for EQ/NE operations in grouper/merger
- binary search trees for grouper/merger
- multithreaded merger
- package as a distribution and make it available via PyPI
- sphinx and doxygen documentation for parser and engine

Resources

- Thesis Blog

  http://mthesis.vaibhavbajpai.com

- Thesis Source

  https://github.com/vbajpai/mthesis-src/

- Issue Tracker

  https://github.com/vbajpai/mthesis-src/issues

- Thesis Proposal

  http://www.vaibhavbajpai.com/documents/vbajpai-proposal.pdf

References

(1) V. Marinov, "**Design of an IP Flow Record Query Language**," Master's thesis, Jacobs University Bremen, Campus Ring 1, 28759 Bremen, Germany, August 2009.

(2) K. Kanev, "**Flowy - Network Flow Analysis Application**," Master's thesis, Jacobs University Bremen, Campus Ring 1, 28759 Bremen, Germany, August 2009.

(3) P. Nemeth, "**Flowy Improvements using Map/Reduce**," Bachelor's thesis, Jacobs University Bremen, Campus Ring 1, 28759 Bremen, Germany, May 2010.

(4) J. Schauer, "**Flowy 2.0: Fast Execution of Stream based IP Flow Queries**," Bachelor's thesis, Jacobs University Bremen, Campus Ring 1, 28759 Bremen, Germany, May 2011.