Packet Classification

Adnan Aziz

The University of Texas

References:

- P. Gupta. Algorithms for routing lookups and packet classification. PhD Thesis, Stanford University, 2001.
  - Excellent survey, pages 106–128; recursive flow classification
- F. Baboescu and G. Varghese. Scalable packet classification. ACM SIGCOMM, 2001.
- P. Agrawal and J. Erickson. Geometric range searching and its relatives.
  - Rigorous, mathematical treatment.

Motivation

IP networks have very limited support for QoS, security, accounting

- customers unwilling to commit to IP-based infrastructure

In principle, packet classification could help solve these problem

- differentiate between customers, type of traffic

In practise, performance drops dramatically

- number of algorithms & architectures for fast packet classification

Syntax and semantics of rules

Will study classification based on header fields

- IP src/dest addresses, protocol, TCP src/dest ports

Syntactically, rule is *(predicate, action)* pair

- predicate is sequence of prefixes, one per header field
  - in some formulations, use intervals for succinctness
- action is an integer in some range $[0, A-1]$

Semantically, packet satisfies predicate iff each header field from packet matches corresponding prefix

- rule-set — collection of rules
- each rule is given a priority, action taken is that of (any) highest priority rule satisfied by packet

Working example

Rule	F1	F2
R1	00∗	00∗
R2	0∗	01∗
R3	1∗	0∗
R4	00∗	0∗
R5	0∗	1∗
R6	∗	1∗

Rules ordered from top to bottom

Relationship with computational geometry

A 2-D rule represents a rectangle parallel to the axes

- rule in d-dimensions — d-dimensional hyper-rectangle
- packet with d fields in header — point in d-dimensional space

Classifier is collection of rectangles, each labeled with priority

- classification: find highest priority rectangle containing packet

Point location problem:

- given set of nonoverlapping regions in d-dimensional space, find region containing point

Claim commonly made that packet classification is as hard as point location problem

- not clear (to me): coming from prefixes

Best bounds for point location: N regions, d dimensions (can do better in 2-D and 3-D)

- $\Theta(\log N)$ time with $\Theta(N^d)$ space
- $\Theta((\log N)^{d-1})$ time with $\Theta(N)$ space

Conclusion: need heuristics that exploit statistics of rule-sets seen in practise

Computational complexity

First implementation of firewalls: linear scan, with some tricks

- works fine for 3 Mb Ethernet, 10 rules in rule-set

Innate complexity: performing classification is akin to searching geometrical structures

- recall from IP forwarding: prefix corresponds to interval of $[0, 2^{32} - 1]$

Given set of N disjoint ranges $\{[l_1, u_1], \ldots, [l_N, u_N]\}$ partitioning $[0, 2^{32} - 1]$ range lookup problem is to find range corresponding to point P

- assumption: lots of lookups, can spend time preprocessing set
- various solutions: binary search on endpoints, prefix matching (convert range to set of prefixes)
  - $[4, 7]$ — 01$**$; Interval $[3, 8]$ — 0011, 01$**$, 1000; Interval $[1, 14]$ — 0001, 001$*$, 01$**$, 10$**$, 110$*$, 1100
  - worst case: range on W-dimensions becomes $2W - 2$ prefixes

How to lookup $(v_1, v_2, \ldots, v_d)$?

- proceed recursively in each dimension
- at **each** F1-trie node, follow next-trie pointer recursively
- encounter a rule iff rule matches $\Rightarrow$ keep record of highest priority rule

Complexity:

- space — $\Theta(N \cdot d \cdot W)$ "rake"
- time — Gupta says $O(W^d)$??? Looks like DFS, $\Theta(N \cdot d \cdot W)$???

Set-pruning tries

Improve search time by replication

- construction similar to that of hiearchical trie
  - for **each** prefix p in $F1$, recursively construct $(d-1)$-dimensional trie T_p on those rules which specify p **or** a prefix of p in first dimension
  - conceptually "pushing" down prefixes

Hardware solution — TCAMs

Very much in spirit of IP forwarding. TCAM essential, cannot do with CAMs (great diversity in lengths)

Very much as in IP forwarding

- select highest priority matching rule with priority encoder
- need to pipeline for performance
- various architectures trading off area/time, lookup/update
- same problems (density/cost, power)

Hierachical tries

Natural extension of tries; construct recursively

- $d = 1$, very similar to usual trie (difference???)
- $d > 1$, construct $F1$ trie on prefixes from first dimension
  - use next-trie pointer to link p to T_p

Convince yourself that the hierarchical trie actually captures rule-set

Crossproducting

Basic idea — first identify which interval in each field incoming packet lies in

- index into multidimensional array to find highest priority rule

Details: use binary search on array of endpoints for interval identification

- N rules implies $\Theta(\lg N \cdot d)$ complexity: search on each field, $\Theta(\lg N)$ time since, prefixes partition number line into at most $2N$ intervals
- if G_k is set of intervals corresponding to k-th field, multidimensional array is of size $\Pi_{i=1}^{d} |G_i|$

Tuple-state space search

Very similar to binary search on hash tables for IP forwarding

Map each d-dimensional rule into a d-tuple of integers, whose i-th component denote the length of prefix in i-th dimension

- M distinct tuples $\Rightarrow$ build M hashtables, do M hash lookups
- improve upon M by using markers, perform binary search

No markers — $O(N)$ storage, with markers?

Note: Same authors continue to publish very different schemes

Search for $(v_1, v_2, \ldots, v_d)$ by finding longest matching prefix of v_1, follow its next trie pointer, recurse

- guaranteed every matching rule encountered

Complexity:

- time — $\Theta(W \cdot d)$ (no backtrack!)
- space — $\Theta(N^d \cdot d \cdot W)$

2-D classification

Can do $O(N)$ space and $O(W)$ time in two dimensions (as opposed to $\Theta(N^2)$ space from set-pruning tries)

- first dimension — prefix, second dimension — arbitrary interval

Intuition: order prefixes on first field by length, use pruning information from search on previous length on second field

Recursive flow classification

View as logic function mapping S bits in header to $T = \lceil \lg N \rceil$ bits

- directly create table $\Rightarrow$ size 2^S

RFC: basic idea is to "re-encode" input space

- suppose rule-set is on 5 header fields, and for each field rule-set engenders at most 16 intervals
  - encode interval with 4 bits $\Rightarrow$ can solve with lookup table with 2^{20} entries

Unfortunately, nontrivial rule-set won't yield so few intervals on each field

- iterate construction: map 16 bits to 8, combine pairs, repeat

Which pairs to combine: most "correlated" so as to reduce the number of equivalence classes

- no details on this

Results: 2 4Mb SRAM, 2 4-bank 64 Mb SDRAM, 125 Mhz suffices for all rule-sets

Updates: delete easy, insert hard

Relationship to BDDs?

Bitmap intersecting

Foundation: Set of rules S matching a packet is $\cap_{i=1}^{d} S_i$, where S_i is rules matching in i-th dimension

- compute S_i using binary search on interval endpoint
- take intersection efficiently by encoding subset of rule-set as N bit vector
- index rules in decreasing order of priority

Storage: $\Theta(d \cdot N \cdot N)$, time is $\Theta(d \cdot \lg N + d \cdot N/w)$, where w is word length

- parallel hardware $\Rightarrow$ reduce by a factor of d

F. Baboescu and G. Varghese SIGCOMM 01 extend this scheme by performing aggregation on the bit vectors

- individual S_i sets tend to be small, so corresponding bit vectors are sparse
  - actually, if even one S_i is small, scheme works well
- rather than AND N-bit vectors, aggregate (OR) A consecutive bits, then search only in locations where conjunction of aggregated vectors evaluates to 1
- reorder rule indices to get sparser aggregated vectors
  - heuristic: sort on rules on prefix length
  - nontrivial: given k sets $S_1, \ldots, S_k$ of N-bit vectors, it's NP-complete to find if $\exists x_i \in S_i$ such that $\cap_{i=1}^{d} x_i \neq 0$