ClassBench: A Packet Classification Benchmark

By: Mehdi Sabzevari
Outline

• INTRODUCTION
• ANALYSIS OF REAL FILTER SETS
  - Understanding Filter Composition
  - Application Specifications
  - Address Prefix Pairs
  - Scope
  - Additional Fields
• PARAMETER FILES
• SYNTHETIC FILTER SET GENERATION
  - Smoothing Adjustment
  - Scope Adjustment
  - Filter Redundancy & Priority
• TRACE GENERATION
INTRODUCTION

- primary bottleneck in high-performance routers
- Packet classification at physical link speeds
- packet classifier
  - compare header fields of every incoming packet against a set of filters in order to assign a flow identifier
  - apply security policies, application processing, and quality-of-service guarantees
- Where is?
  - in enterprise firewalls or edge routers
- no publicly available benchmarking tools or filter sets exist for standardized performance evaluation
- access to large, real filter sets has been limited to a small subset of the research community
- ClassBench
• **ClassBench**
ClassBench goals:

- Researchers seeking to evaluate new classification algorithms.
- Classification product vendors seeking to market their products.
- Classification product customers seeking to verify and compare classification product performance on a uniform scale.
• ANALYSIS OF REAL FILTER SETS:
  ➢ metrics and characterizations of filter set structure
  ➢ analyses on 12 real filter sets (in ISPs)
  ➢ filter sets range in size from 68 to 4557 entries
  ➢ following formats:
    - Access Control List (ACL)
    - Firewall (FW)
    - IP Chain (IPC)
• **Understanding Filter Composition**
  - complex packet filters typically appear in firewall and edge router filter sets
  - additional applications
    - Virtual Private Networks (VPNs), resource reservation
  - filter sets are created manually
    - CiscoWorks VPN/Security Management Solution (VMS)
    - Lucent Security Management Server (LSMS)
    - model of filter construction:
      - communicating subnets
      - the application or set of applications

**Results:**
- each filter as having two major components:
  - an address prefix pair (source address prefix and a destination address prefix)
  - an application specification (transport protocol, source port number, and destination port number)
• **Application Specifications:**
  - analyzed the application specifications in the 12 filter sets
  1) Protocol: TCP (49%), UDP (27%), wildcard(13%), ICMP (10%)
  - The following protocols were specified by less than 1% of the filters: General Routing Encapsulation (GRE), Open Shortest Path First (OSPF), Interior Gateway Protocol (IGP), Enhanced Interior Gateway Routing Protocol (EIGRP), IP Encapsulating Security Payload (ESP) for IPv6, IP Authentication Header (AH) for IPv6, IP Encapsulation within IP (IPE).
  2) **Port Ranges:**
     • WC, wildcard
     • HI, ephemeral user port range [1024 : 65535]
     • LO, well-known system port range [0 : 1023]
     • AR, arbitrary range
     • EM, exact match
• **Application Specifications:**
  3) Port Pair Class: (PPC)

![Port Pair Class Matrix](image-url)  
*Fig. 2. Port Pair Class Matrix for TCP, filter set fw4.*
• **Application Specifications:**

4) Address Prefix Pairs:

![Prefix length distribution for address prefix pairs in filter set ipc1.](image)
• **Notices in Address Prefix Pairs:**
  ✓ The number of unique source/destination prefix pair lengths is typically less than 32
  • For example, the largest filter set contained 4557 filters, 11 unique source address prefix lengths, 3 unique destination address lengths, and 31 unique source/destination prefix pair lengths
  ✓ **By considering the prefix pair distribution:**
    - characterize the *size of the communicating subnets*
    - characterize the relationships among address prefixes and the amount of address space covered by the prefixes
      - a binary tree constructed from the IP source address prefixes
        - *Branching Probability*
        - *Skew*
        - *correlation*
two binary tries from the source and destination prefixes

Fig. 4. Example of complete statistical characterization of address prefixes.
• Branching Probability

(a) Source address branching probability; average per level.
• Skew

\[
skew = 1 - \frac{\text{weight(light)}}{\text{weight(heavy)}}
\]
• **Skew:**

![Skew Graph]

- **Correlation:** the probability that the source and destination address prefixes continue to be the same for a given prefix length.
• **Scope:**
  - more specific cover a small set of possible packet headers
  - less specific cover a large set of possible packet headers
  - number of possible packet headers covered by a filter is characterized by its *tuple*
• standard 5-tuple as a vector containing the following fields:
  • $t[0]$, source address prefix length, $[0::32]$
  • $t[1]$, destination address prefix length, $[0::32]$
  • $t[2]$, source port range width, the number of port numbers covered by the range, $[0::216]$
  • $t[3]$, destination port range width, the number of port numbers covered by the range, $[0::216]$
  • $t[4]$, protocol specification, Boolean value denoting whether or not a protocol is specified, $[0; 1]$

\[
\text{scope} = \log\left\{ (2^{32} - t[0]) \times (2^{32} - t[1]) \times t[2] \times t[3] \times (2^{8(1 - t[4])}) \right\} \\
= (32 - t[0]) + (32 - t[1]) + (\log t[2]) + (\log t[2]) + 8(1 - t[4])
\]
PARAMETER FILES:
Parameter files include the following entries:

• **Protocol** specifications and the distribution of filters over those values
• **Port Pair Class Matrix** for each unique protocol specification in the filter set
• **Flags** specifications for each protocol and a distribution of filters over those values
• **Arbitrary port range** specifications and a distribution of filters over those values for both the source and destination port fields
• **Exact port** number specifications and a distribution of filters over those values for both the source and destination port fields
• **Prefix pair length** distribution for each Port Pair Class Matrix
• **Address prefix** branching and skew distributions for both source and destination address prefixes
• **Address prefix** correlation distribution
• **Prefix nesting** thresholds for both source and destination address prefixes.
Fig. 7. *Parameter files* represent prefix pair length distributions using a combination of a total prefix length distribution and source prefix length distributions for each non-zero total length.
• SYNTHETIC FILTER SET GENERATION
  ✓ read the statistics and distributions from the parameter file
  ✓ get the four high-level input parameters:
    • size: target size for the synthetic filter set
    • smoothing: controls the number of new address aggregates (prefix lengths)
    • port scope: biases the tool to generate more or less specific port range pairs
    • address scope: biases the tool to generate more or less specific address prefix pairs
FilterSetGenerator()

    // Read input file and parameters
1    read(parameter file)
2    get(size)
3    get(smoothing)
4    get(port scope)
5    get(address scope)
    // Apply smoothing to prefix pair lengths
6    If smoothing > 0
7        For i: 1 to MaxPortPairClass
8            TotalLengths[i] = smooth(smoothing)
9        For j: 0 to 64
10            SALLengths[i][j] = smooth(smoothing)
    // Allocate temporary filter array
11   FilterType Filters[size]
    // Generate partial filters
12   For i: 1 to size
        // Choose an application specification
13       rv = Random()
14       Filters[i].Prot = Protocols→choose(rv)
15       rv = Random()
16       Filters[i].Flags =
                  Flags[Filters[i].Prot] = choose(rv)
rv = RandomBias("port scope")
PPC = PPCMatrix[Filters[i].Prot]→choose(rv)
rv = Random()
Filters[i].SP =
    SrcPorts[PPC.SPClass]→choose(rv)
rv = Random()
Filters[i].DP =
    DstPorts[PPC.DPClass]→choose(rv)
    // Choose an address prefix pair length
rv = RandomBias("address scope")
TotalLength = TotalLengths[PPC]→choose(rv)
rv = Random()
Filters[i].SALength =
    SrcLengths[PPC][TotalLength]→choose(rv)
Filters[i].DALength =
    TotalLength - Filters[i].SALength
    // Assign address prefix pairs
AssignSA(Filters)
AssignDA(Filters)
    // Remove redundant filters
RemoveRedundantFilters(Filters)
    // Prevent filter nesting
OrderNestedFilters(Filters)
PrintFilters(Filters)
**SYNTHETIC FILTER SET GENERATION**

- *Smoothing Adjustment*
  - Injecting purely random address prefix pair lengths neglects the structure of the filter set
  - “near” to the original prefix length
  - new prefix pair lengths should be clustered around the existing spike in the distribution
  - translates “spikes” in the distribution into smoother “hills”
  - *smoothing parameter which limits the maximum radius*
  - clustering using a symmetric binomial distribution

\[ p_i = \binom{2r}{i} \left( \frac{1}{2} \right)^{2r} \]

- \( r \) may assume values in the range \([0 : 64]\)
• **SYNTHETIC FILTER SET GENERATION**
  
  - *Smoothing Adjustment*

---

**Fig. 9.** Prefix pair length distributions for a synthetic filter set of 64000 filters generated with a *parameter file* specifying 16-bit prefix lengths for all addresses and smoothing parameter $r = 8$. 
• **SYNTHETIC FILTER SET ENERATION**

   - *Smoothing Adjustment*

Fig. 10. Prefix pair length distribution for a synthetic filter set of 64000 filters generated with the `ipc1 parameter file` with smoothing parameter \( r = 4 \).
SYNTHETIC FILTER SET GENERATION

Scope Adjustment

Consider the case of selecting prefix pair lengths

recompute the cumulative distribution to make longer or shorter total prefix lengths more or less probable

more specific address prefix pairs, then we want the random variable used to sample from the distribution to be biased to values closer to 1

The reverse is true if we want less specific address prefix pairs

random number generator to choose a uniformly distributed random variable $rv_{uni}$

apply a biasing function to generate a biased random variable $rv_{bias}$

$$rv_{bias} = rv_{uni} \times (scope \times rv_{uni} - scope + 1)$$
• **Scope Adjustment**

![Cumulative Density Plot](image)

**Fig. 11.** Example of sampling from a cumulative distribution using a uniform random variable, and a biased random variable. Distribution is for the total prefix pair length associated with the WC-WC port pair class of the acl2 filter set.
• **Scope Adjustment**

• Positive values of *address scope bias the Filter Set Generator* to choose less specific address prefix pairs, thus increasing the average scope of the filter set.

(a) Biased random variable is defined by area under line with slope $s = 2 \times \text{scope}$. 
• **Scope Adjustment**

(b) Plot of scope biasing function.
• **Scope Adjustment**

Fig. 13. Average scope of synthetic filter sets consisting of 16000 filters generated with parameter files extracted from filter sets acl3, fw5, and ipc1, and various values of the scope parameters.
• **Filter Redundancy & Priority**
  ▪ comparing each filter against all other filters in the set \( O(N^2) \)
  ▪ Other approach: constructing a binary search tree of tuple set pointers
  ▪ the scope of the tuple as the key for the node
  ▪ When adding a filter to a tuple set, we search the set for redundant filters
• **TRACE GENERATION**

```plaintext
TraceGenerator()

    // Generate list of synthetic packet headers
    1. read(FilterSet)
    2. get(scale)
    3. get(ParetoA)
    4. get(ParetoB)
    5. Threshold = scale \times \text{size(FilterSet)}
    6. HeaderList Headers()
    7. While size(Headers) < Threshold
        8. RandFilt = randint(0, size(FilterSet))
        9. NewHeader = RandomCorner(RandFilt, FilterSet)
        10. Copies = Pareto(ParetoA, ParetoB)
        11. For i: 1 to Copies
            12. Headers→append(NewHeader)
    13. Headers→print

Fig. 14. Pseudocode for Trace Generator.
```
• TRACE GENERATION
  ▪ a is typically called the shape parameter
  ▪ b is typically called the scale parameter
  ▪ distribution is defined on values in $(b, \infty)$

  ❖ Low locality of reference, short tail:
    $(a = 10, b = 1)$ most headers will be inserted once
  ❖ Low locality of reference, long tail:
    $(a = 1, b = 1)$ many headers will be inserted once, but some could be inserted over 20 times
  ❖ High locality of reference, short tail:
    $(a = 10, b = 4)$ most headers will be inserted four times

\[
P(x) = \frac{ab^a}{x^{a+1}}
\]

\[
D(x) = 1 - \left(\frac{b}{x}\right)^a
\]

\[
\mu = \frac{ab}{a - 1}
\]