Switches

Packet classification

Packet Classification

Modern routers perform service differentiation by looking at higher layer protocol headers (layer 4 switching or even layer 7 switching)

- firewalls
- resource reservation
- priority service
- QoS routing

Core routers easily have thousands of such rules.

In this lesson we focus on classifiers looking at layers 2 to 4.

The problem of packet classification

Packets are classified according to a set of *rules*.

- Multiple rules can be matched, so rules have a priority (cost)
- The problem is to find the least-cost matching rule for each message.

Each rule looks at K distinct header fields in each message.

- Each field H[i] is a string of bits of fixed size
- The position of fields may depend on the value of other fields!
- Examples: Source Addr, Protocol Field, TCP flags, TCP dest. port

Three kinds of match:

- exact match
- prefix match
- range match

The classifier

The classifier (or rule database) consists in a set of rules R1, … RN.

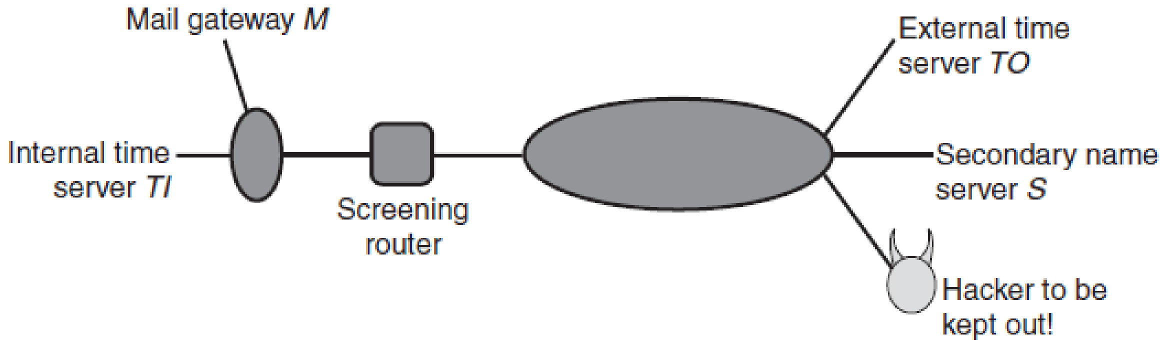
Each rule

- specifies a combination of K values, one for each field.
- is associated to a cost (generally is the position in the list)
- is associated to a directive, which specifies what to do with the packet
  - drop
  - forward according to destination address
  - forward towards a given next hop

N is the number of rules

K is the number of dimensions

Firewall example



Destination	Source	Destination Port	Source Port	Flags	Comments
M	*	25	*	*	Allow inbound mail
M	*	53	*	UDP	Allow DNS access
M	S	53	*	*	Secondary access
M	*	23	*	*	Incoming telnet
TI	*TO*	123	123	UDP	NTP time info
*	Net	*	*	*	Outgoing packets
Net	*	*	*	TCP ack	Return ACKs OK
*	*	*	*	*	Block everything!

Requirements

Lookup at wire speed with minimum sized packets

Small size of the data structure

Insertion/deletion is unfrequent, so speed is not an issue

Beware of stateful classification

- rules are added/deleted dynamically depending on some event
- in this case insertion/deletion speed is an issue

Examples:

- arrival of an UDP request triggers insertion of a rule to allow the corresponding answer
- a too large number of connection attepts from an host triggers insertion of a rule to deny all traffic from that host

Simple solutions (1)

Linear search

- slow for large databases

Caching

- experiments show that cache hit rate 80%-90% (low!)

Passing labels

- someone else performs classification and adds a label to the packet
- requires protocol changes (e.g. MPLS, DiffServ)

Demultiplexing Algorithms (e.g. BPF)

- the problem is a bit different because in demultiplexing exact match is much more common than prefix match or range match
- composability is necessary to reduce complexity

Simple solutions (2)

TCAMs

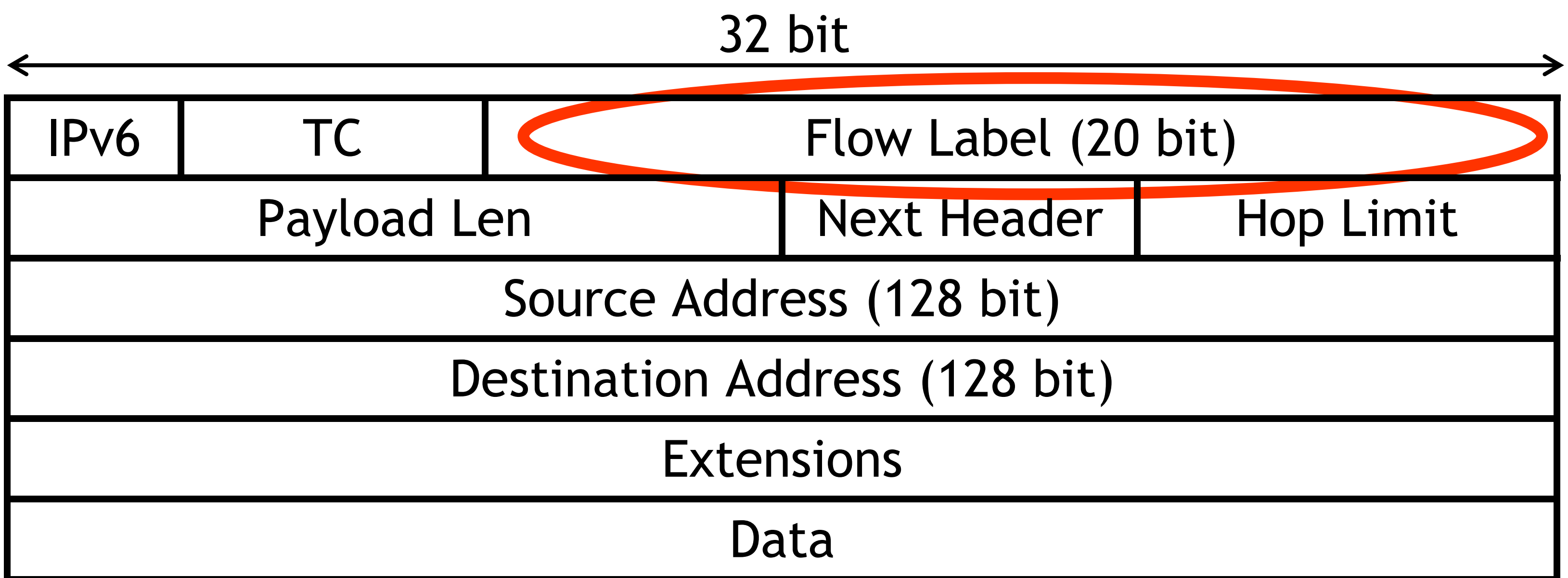
- large area
- high power consumption
- high cost
- no support for range match (it is necessary to multiply rules)

Intermezzo
Flow label in IPv6

IPv6 has a 20-bit flow label. Usage is optional

It is assigned by the source to identify all the packets “in a flow”

How could it be used for packet classification?

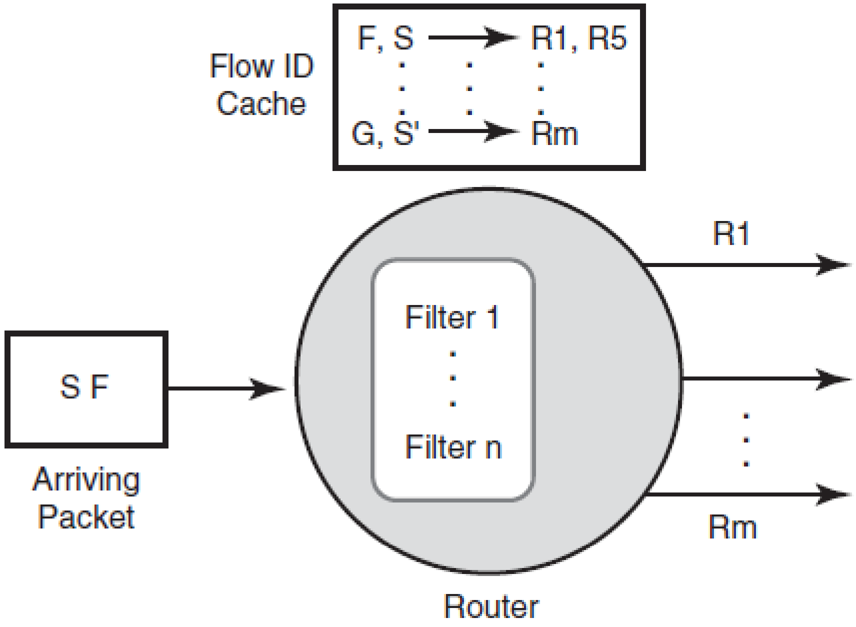


Intermezzo
Using IPv6 Flow Label

The router caches the mapping between (source id, flow id) and the matching rules

If the source is trusted (i.e. it does not change the K headers) classification is fast

What additional costs if the source cannot be trusted?



Algorithmic solutions

Several algorithms. No one outperforms the others.

- Hierarchical Tries
- Geometric
- Bit Vector Linear Search
- Cross Producting
- Decision Trees

In general classification requires either a large amount of memory or a large amount of time

Set-pruning tries

Extension of one-dimentional tries.

Efficient only for two dimensions (source and destination addresses)

Build a trie for the destinations.

For each leaf (having prefix D)

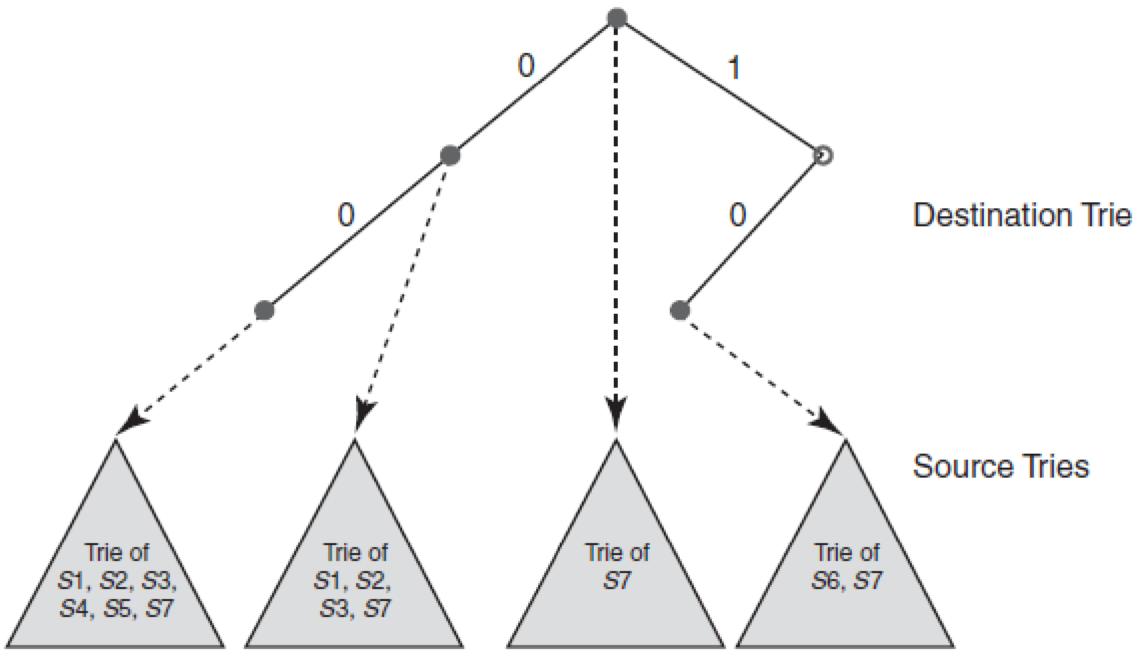
- remove from the ruleset the rules incompatible with D
- build a trie for the sources in the remaining ruleset.

Memory explosion problem: in the worst case space is O(N^2)

Can be extended to K dimensions, but memory ~O(N^K)

Set-Pruning Tries Example

Rule	Destination	Source
R_1	0*	10*
R_2	0*	01*
R_3	0*	1*
R_4	00*	1*
R_5	00*	11*
R_6	10*	1*
R_7	*	00*



Geometric Scheme

A prefix is like a segment, a 2D rule is a rectangle, a 3D rule is a cube, …

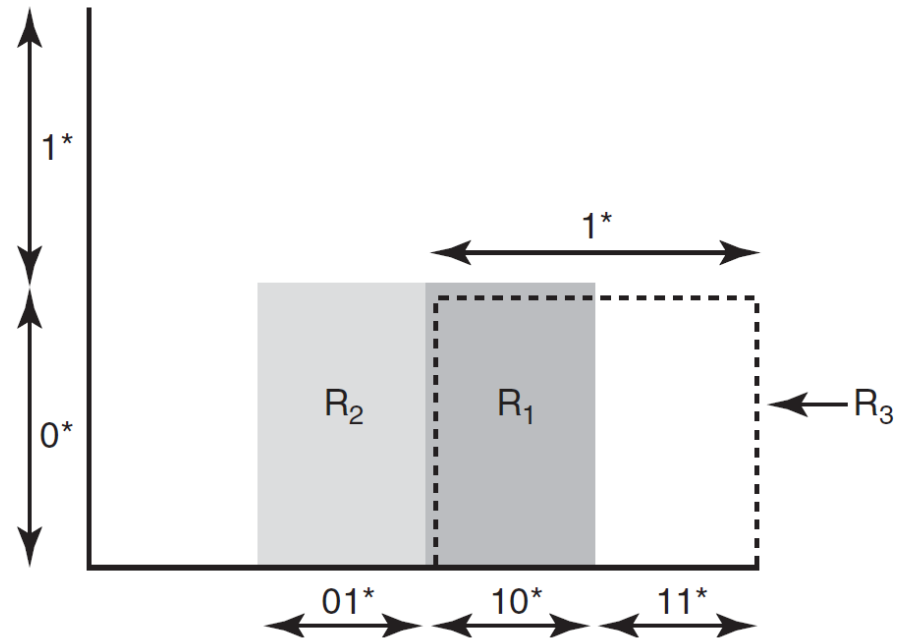
A packet is a point.

The problem is finding the least-cost box that contains the point.

If K=2 complexity is O(log N)

For K>2 either time or memory explode in the worst case

Why? Consider the number of disjoint classification regions worst case there are O(N^K) regions



Geometric Scheme

Field experience shows that complexity in the expected case is much lower than in the worst case:

- Prefix containment is rare
- Range matches are rare
- The number of disjoint regions is small O(N) instead of O(N^K)
- Packets generally match only a few source-dest pairs suggesting that the number of regions can grow more slowly than N

Geometric schemes are appealing even in the multidimentional case

Divide and Conquer Strategies
Bit Vector Linear Search

Idea:

- match one field at a time
- calculate the intersection
- search the intersection

- For each field the match returns a bit mask with a bit for each rule in the set. The *i*th bit is 1 if the match appears in the *i*th rule
- The intersection is obtained with an AND operation.
  - Time complexity of AND is linear with N, but faster than simple linear search because the number of memory accesses is smaller.

- Good for moderate size rulesets.

Bit Vector Linear Search Example

Full database

Destination	Source	Destination Port	Source Port	Flags	Comments
M	*	25	*	*	Allow inbound mail
M	*	53	*	UDP	Allow DNS access
M	S	53	*	*	Secondary access
M	*	23	*	*	Incoming telnet
TI	*TO*	123	123	UDP	NTP time info
*	Net	*	*	*	Outgoing packets
Net	*	*	*	TCP ack	Return ACKs OK
*	*	*	*	*	Block everything!

Sliced database

Destination Prefixes	
M	11110111
TI	00001111
Net	00000111
*	00000101

Source Prefixes	
S	11110011
TO	11011011
Net	11010111
*	11010011

DstPort Prefixes	
25	10000111
53	01100111
23	00010111
123	00001111
*	00000111

SrcPort Prefixes	
123	11111111
*	11110111

Flags Prefixes	
UDP	11111101
TCP	10110111
*	10110101

Bit Vector Linear Search
Comments

Moderate size of the ruleset. Can be increased if bus width increases.

Can be parallelized easily.

Time Complexity:

- K individual matches +
- N(K+1)/W for calculating the intersection (W=bus width)

Space complexity:

- K individual matches +
- N^2 K bits (worst case)

Slow updates (database must be rebuilt)

Decision trees

- Similar to the set-pruning scheme.
- The set-pruning algorithm tests all the bits of the destination, then of the source. If extended to multiple fields, each field is tested before moving to another.
- Idea 1: choose the optimal order for testing the bits
  - there are exponentially many possibilities, use some heuristic (e.g. local search)
- Idea 2: split the ruleset into multiple decision trees.
  - Increases search time, but reduces storage.
- Idea 3: to avoid too many tree levels, perform linear search in the lowest level
  - Increases search time, but reduces storage.

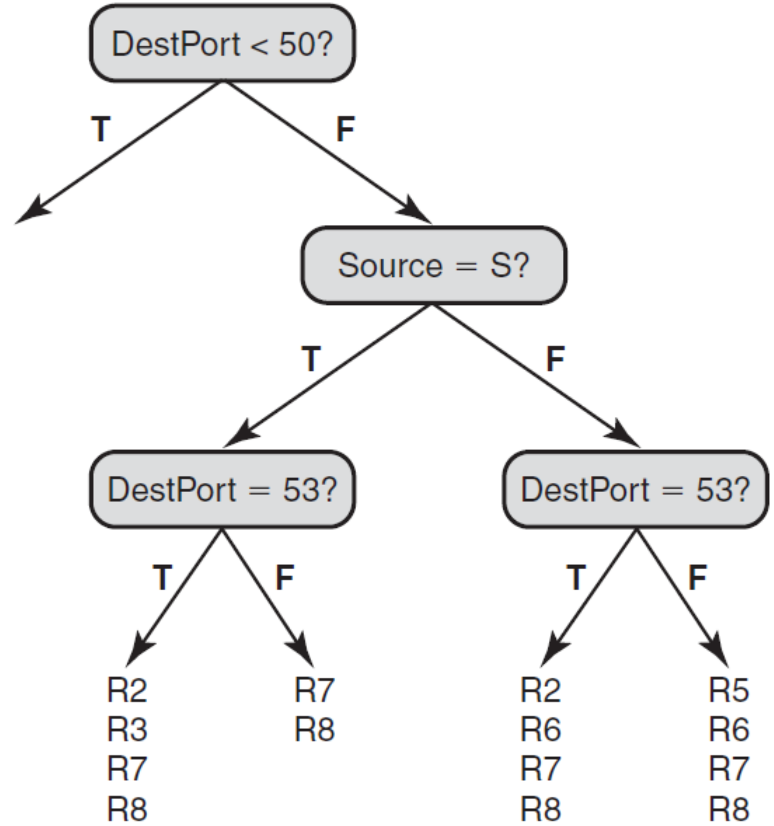
Decision Trees
Example

Strong assumption: enough distinct fields
many rule replications at the leaves

If the assumptions holds

- Time complexity nearly constant
- Space complexity nearly linear

Update is slow



Example for the Hierarchical Cuttings (HiCut) algorithm. Note the range tests instead of the bit tests.