On Kirill's Contribution to Packet Classification and an Example Why It Matters

Gábor Rétvári and Stefan Schmid
Motivation

● A joint presentation by Gabor and Stefan: two friends and colleagues of Kirill

● We have a joint national project inspired by and building upon Kirill’s work
  ○ Dependable Network Data Plane for the Cloud (DELTA)
  ○ Funded by NKFIH and FWF

● Emerging programmable data plane: new opportunities for innovative algorithms as the ones devised by Kirill
Agenda

- Algorithms in the data plane and example: Tuple Space Search
- Why designing good algorithms matters: a case study
- One solution by Kirill and Gabor
Packet Classification: Basics

Given an ordered list of wildcard (ternary) rules, find the first rule that matches a given packet header

Exact-match and longest-prefix-match (LPM) are simpler subproblems

**Indispensable in packet processing:** IP packet forwarding (only LPM), firewalls/ACLs, QoS shapers/rate-limiters/classifiers, OpenFlow/P4 match-action processing & policy routing, accounting & billing, etc. [Gupta, 2001]

**Example:** allow HTTP and DNS traffic from select networks, deny everything else

<table>
<thead>
<tr>
<th>Src</th>
<th>Dst</th>
<th>Proto</th>
<th>Dst port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.10.0.0/16</td>
<td>192.168.1.100</td>
<td>6 (TCP)</td>
<td>80 (HTTP)</td>
<td>Allow</td>
</tr>
<tr>
<td>10.0.0.0/8</td>
<td>192.168.1.53</td>
<td>17 (UDP)</td>
<td>53 (DNS)</td>
<td>Allow</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>Deny</td>
</tr>
</tbody>
</table>
Packet Classification: Algorithms

“Easy” in hardware (TCAMs), notoriously difficult in software: "a packet classifier with \( n \) rules and \( k>1 \) fields uses either \( O(n^k) \) bits space and \( O(\log n) \) time, or \( O(n) \) space and \( O((\log n)^k) \) time" [Feldman 2000, Gupta, 2001, Kogan 2014]

Difficulty stems from that (1) rules can have wildcard bit ("don’t care" bit *) and so (2) may overlap, but (3) we need to find the first matching rule

Software implementations typically use heuristics: linear search, hierarchical tries, tuple space search & decision trees (see later), geometric/cut-based algorithms (HiCuts/Efficuts), etc.

Kirill was highly active in this area [Kogan 2013, Kogan 2014, Nikolenko 2016, Demianiuk 2021]
Tuple Space Search (TSS): Idea

Hash-tables work for exact-match but a generic packet classifier has wildcards in the rules: we need something more clever

TSS: decompose a $w$ bit wide ruleset into at most $2^w$ exact-match instances

1. Find all combinations of wildcard bit positions in the rules (called tuples)
2. For each tuple, create a hash on the non-wildcard bit positions (“mask”)
3. Mask & match each incoming packet against all hashes/tuples
4. Return the highest priority match (if any)

Heuristic “prerequisite”: $O(2^w)$ hash lookups in the worst case, but typically much fewer
Tuple Space Search (TSS): Example

An IPv6 forwarding table: rather wide (w=128), but only prefix rules

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next-hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80::/4</td>
<td>a</td>
</tr>
<tr>
<td>0x40::/2</td>
<td>b</td>
</tr>
<tr>
<td>0xc0::/2</td>
<td>c</td>
</tr>
<tr>
<td>0x80::/1</td>
<td>d</td>
</tr>
</tbody>
</table>

3 tuples, a separate hash table for each one

<table>
<thead>
<tr>
<th>filter</th>
<th>#0</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$F_2$</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$F_3$</td>
<td>1</td>
<td>1</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$F_4$</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Input 0111 matches only the 2nd hash only, 1100 matches both 2nd and 3rd, $F_3$ “wins”

Good news: the number of tuples (3) is much smaller than the worst case ($2^4=16$)

Observe that (1) **each rule maps to a single tuple**, and (2) rules per each tuple admit an exact-match lookup (i.e., be order-independent, [Kogan 2014])
A Threat: Algorithmic Complexity Attacks

- In general, algorithmic complexity attacks exploit known cases in which an algorithm will exhibit worst-case behavior
  - E.g., *Billion Laughs*: attack on XML parsers, exponentially expanding itself (“lol”)
  - E.g., *Zip bomb*: attack on virus scanner unpacking an archive (resource explosion)
- Algorithmic complexity attacks are also a threat for network algorithms
- We have recently shown that Tuple Space Search (TSS) has such an issue
  - TSS for example used in the Open vSwitch (OvS) MegaFlow Cache (MFC)
  - OvS is the “de facto” software switch in data centers
- Simple flow table: “allow some but drop others”
Denial-of-Service Attack on OVS Packet Processing

- OVS uses a MegaFlow cache: first packet subject to full-table processing, then flow-specific rules and actions cached
- Entries matching on the same headers are collected into a hash
  - Masked packet headers can be found fast
  - However, masks and associated hashes are searched sequentially

\[
\text{Can be a costly linear search in case of lots of masks!}
\]
A Denial-of-Service Attack on TSS

- KEY FINDING: More masks -> slower packet processing
- Strategy: for each packet for the allow rule, add a packet with the relevant bits inverted
  - Each packet gives one mask
- Multiple allow rules on multiple header fields -> Exponential growth
- Matching on either 1) and 2) -> 512 masks

With less than 1 Mbps specially crafted packet sequence we get a full Denial-of-Service (OVS performance drops close to 0%).
TSS on steroids: Kirill’s beautiful idea

TSS is extremely simple but slow if the number of hashes grows huge.

How to decrease the number of tuples/hashtags?

Recall the “invariants” of TSS: (1) each rule must map to a single hash, (2) rules per each hash must be order-independent.

Kirill’s observation: rules that belong to different tuples can be assigned into a single hash as long as the above two prerequisites hold.

This may allow to map rules that belong to different tuples to a single hash.
Reduced order-independent decompositions: Idea

**Strong reduced order-independent decompositions** [Nikolenko 2016]:

1) partition the rules into the smallest number of groups, where each group is associated with a bitmask, so that

2) the rules in each group masked with the group’s bit positions are order-independent

But we may lose “valuable” bits in each group due to applying the mask

Perform a **false positive (FP) check** after each hash-lookup (cf. [Kogan 2014])

TSS is the worst-case order-independent reduction, so we may only get fewer hash lookups, this may be worth the additional false-positive check

Simulations show that usually only 20-30 hashes \((w=16)\) is enough, instead of 128
Reduced order-independent decompositions: Example

Recall the previous IPv6 forwarding table: TSS needed 3 hash lookups

<table>
<thead>
<tr>
<th>filter</th>
<th>#0</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$F_2$</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$F_3$</td>
<td>1</td>
<td>1</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$F_4$</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

A reduced order-independent decomposition with just 2 hash lookups + 1 FP check

<table>
<thead>
<tr>
<th>filter</th>
<th>#0</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$F_2$</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$F_3$</td>
<td>1</td>
<td>1</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Kirill’s crazy optimal solution: 1 bit per hash (FP checks not shown)

<table>
<thead>
<tr>
<th>filter</th>
<th>#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>0</td>
</tr>
<tr>
<td>$F_2$</td>
<td>0</td>
</tr>
<tr>
<td>$F_3$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>filter</th>
<th>#0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_2$</td>
<td>0</td>
</tr>
<tr>
<td>$F_4$</td>
<td>1</td>
</tr>
</tbody>
</table>
Closing thoughts

SW packet classification is an actively researched area
Kirill made cornerstone contributions to the field
His ideas will always be an inspiration to the community

SAX-PAC (Scalable And eXpressive PAcket Classification)

Kirill Kogan
Purdue University and NetSysAlgo
kirill.kogan@gmail.com

Sergey Nikolenko
Steklov Mathematical Institute at St. Petersburg,
National Research University Higher School of Economics
sergey@logic.pDMI.ras.ru

Ori Rottenstreich
Mellanox, Israel
orir@mellanox.com

William Culhane
Purdue University
wculhane@purdue.edu

Patrick Eugster
Purdue University and Technical University of Darmstadt
peugster@cs.purdue.edu
Nikolenko, Kogan, Rétvári et al., *How to represent IPv6 forwarding tables on IPv4 or MPLS dataplanes*, IEEE GI, 2016.