adopting software-defined networking: challenges and recent developments

Kirill Kogan

IMDEA Networks Institute

20.12.2017
How should we introduce new services?

**Answer:** standardization with per service resolution
Traditional service interoperation

Standardization of southbound API makes possible to manage network elements from heterogeneous vendors.

Logically centralized management can be implemented in distributed manner (some experience from structured p2p networks).

New players, new relations, new abstractions.
Two questions

What should be flexible to represent desired behaviors in computer networks?

How to represent efficiently these behaviors on control and data planes?
Data plane:

- single packet processing (SIGCOMM'14, HOTI'14, ICNP'16-17);
- packet stream processing (INFOCOM'15-17, ANCS'16, ICNP'17);

Control plane:

- network virtualization (INFOCOM'17).
single packet processing
A classifier $\mathcal{K}$ is a prioritized set of rules, where each rule is a pair of a ternary bit string (filter) and an action.

<table>
<thead>
<tr>
<th>$\mathcal{K}$</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$A_1$</td>
</tr>
<tr>
<td>$R_2$</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>$A_2$</td>
</tr>
<tr>
<td>$R_3$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>$A_3$</td>
</tr>
<tr>
<td>$R_4$</td>
<td>1</td>
<td>*</td>
<td>0</td>
<td>*</td>
<td>$A_4$</td>
</tr>
</tbody>
</table>

A lookup is a process of finding the highest priority rule matched by a given header.

There are two main uses for packet classification:

**Traffic forwarding** between certain points in a communication network.

**Service policies** that guarantee desired traffic properties.

Both can be represented as a tuple-match with actions (OpenFlow, P4) but have different invariants.
SW-based vs. TCAM-based solutions

**SW-based**

\( N = 4 \) filters, \( K = 2 \) fields

- prefixes
- ranges

\[ R_1 = (100*, 001*) \]
\[ R_2 = (1010, 0001) \]
\[ R_3 = (000*, ****) \]
\[ R_4 = (001*, ****) \]

<table>
<thead>
<tr>
<th>Memory</th>
<th>Lookup time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O(N) )</td>
<td>( O(\log^{k-1}N) )</td>
</tr>
<tr>
<td>( O(N^k) )</td>
<td>( O(\log N) )</td>
</tr>
</tbody>
</table>

**TCAM-based**

\( N = 3 \) filters, \( K = 3 \) fields

- prefixes
- ranges

\[ R_1 = ([1, 3], [4, 31], [1, 28]) \]
\[ R_2 = ([4, 4], [2, 30], [4, 27]) \]
\[ R_3 = ([7, 9], [5, 21], [3, 18]) \]

<table>
<thead>
<tr>
<th>Binary Encoding</th>
<th>SRGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>42+28+50=120</td>
<td>24+8+32=64</td>
</tr>
</tbody>
</table>
Filter-order independence

Adding new fields keep order-independence
At most one rule is matched and it can be false-positive.
We can reduce space by skipping new fields

Problem (FSM)

Find a minimal subset of fields that keeps order-independence of a given filter-order independent classifier.

FSM is reducible to SetCover in \( O(kN^2) \) with approximation factor \( 2 \ln N + 1 \)
Classifiers can also be viewed as Boolean expressions.

For example,

\[ R_1 = (100*, 001*) \]
\[ R_2 = (1010, 0001) \]
\[ R_3 = (000*, ****) \]
\[ R_4 = (001*, ****) \]

is equivalent to

\[
f(x_1, \ldots, x_8) = (x_1 \land \overline{x}_2 \land \overline{x}_3 \land \overline{x}_5 \land \overline{x}_6 \land x_7) \\
\lor (x_1 \land \overline{x}_2 \land x_3 \land \overline{x}_4 \land \overline{x}_5 \land \overline{x}_6 \land \overline{x}_7 \land x_8) \\
\lor (\overline{x}_1 \land \overline{x}_2 \land \overline{x}_3) \\
\lor (\overline{x}_1 \land \overline{x}_2 \land x_3).
\]
Classifiers can also be viewed as Boolean expressions.

Hence the MinDNF problem arises.

**Problem (MinDNF)**

*For a given Boolean function, find its minimal size DNF representation.*
MinDNF vs. FSM

MinDNF by itself is not as powerful as FSM.

In this example:

\[ R_1 = (100*, 001*) \]
\[ R_2 = (1010, 0001) \]
\[ R_3 = (000*, ****) \]
\[ R_4 = (001*, ****) \]

MinDNF yields

\[ R_1 = (100*, 001*) \]
\[ R_2 = (1010, 0001) \]
\[ R_{3,4} = (00**, ****) \]
MinDNF vs. FSM

MinDNF by itself is not as powerful as FSM.

In this example:

\[ R_1 = (100*, 001*) \]
\[ R_2 = (1010, 0001) \]
\[ R_3 = (000*, ****) \]
\[ R_4 = (001*, ****) \]

FSM with per-bit resolution:

\[ R_1^{-6} = (10) \]
\[ R_2^{-6} = (11) \]
\[ R_3^{-6} = (00) \]
\[ R_4^{-6} = (01) \]
Irreducible or Rule-order dependent classifiers

A classifier is rule-order dependent or irreducible:

assign a subset of rules;

use the level of pseudo-parallelism $\beta$.

Problem

*Find an assignment of rules to at most $\beta$ disjoint groups that keeps order-independence on $l$ fields, minimizing the size of uncovered rules.*
Semantically equivalent representations on all fields: more fields imply less efficient representation.

Representation with false-positive check: more fields more efficient representation.

Structural properties can significantly improve time-space trade-off.

No restrictions on representation of every group (we define only additional abstraction layer).

Additional applications: operation with broken communication channels, optimizations in PLs, lookups in databases.
Impact of scalability and expressiveness

Forwarding tables (FIBs) map addresses to ports in every switch.
Impact of scalability and expressiveness

Forwarding tables (FIBs) map addresses to ports in every switch.

FIB table sizes are growing (up to $10^6$ filters)

**compact routing:** (shortest path)

$$|route(x, y)| \leq stretch \times dist(x, y)$$

to minimize max local table size
Impact of scalability and expressiveness

Forwarding tables (FIBs) map addresses to ports in every switch.

FIB table sizes are growing (up to $10^6$ filters)

**compact routing:** (shortest path)

$$|route(x, y)| \leq stretch \times dist(x, y)$$

to minimize max local table size

more expressiveness in clean slate solutions - wider tables

(IP $\rightarrow$ IPv6 $\rightarrow$ OpenFlow)
Impact of scalability and expressiveness

Forwarding tables (FIBs) map addresses to ports in every switch.

FIB table sizes are growing (up to $10^6$ filters)

**compact routing:** (shortest path)

\[ |route(x, y)| \leq stretch \times dist(x, y) \]

to minimize max local table size

more expressiveness in clean slate solutions - wider tables

(IP $\rightarrow$ IPv6 $\rightarrow$ OpenFlow)

transition from LPM to **general rule priorities** (IP $\rightarrow$ OpenFlow)
Impact of scalability and expressiveness

Forwarding tables (FIBs) map addresses to ports in every switch.

FIB table sizes are growing (up to $10^6$ filters)

**compact routing:** (shortest path)

$$|route(x, y)| \leq stretch \times dist(x, y)$$
to minimize max local table size

more expressiveness in clean slate solutions - wider tables (IP→ IPv6→ OpenFlow)

transition from **LPM** to **general rule priorities** (IP→ OpenFlow)
Forwarding tables (FIBs) map addresses to ports in every switch.

FIB table sizes are growing (up to $10^6$ filters)

**compact routing:** (shortest path)

$$|route(x, y)| \leq stretch \ast dist(x, y)$$

to minimize max local table size

more expressiveness in clean slate solutions - wider tables

(IP $\rightarrow$ IPv6 $\rightarrow$ OpenFlow)

transition from LPM to general rule priorities (IP $\rightarrow$ OpenFlow)

To address memory vs. performance tradeoff, find abstractions:

independent from clean slate solutions
specific platform representations
Impact of scalability and expressiveness

Forwarding tables (FIBs) map addresses to ports in every switch.

FIB table sizes are growing (up to $10^6$ filters)

**compact routing:** (shortest path)

$$|route(x, y)| \leq stretch \times dist(x, y)$$

to minimize max local table size

more expressiveness in clean slate solutions - wider tables

(IP $\rightarrow$ IPv6 $\rightarrow$ OpenFlow)

transition from **LPM** to **general rule priorities** (IP $\rightarrow$ OpenFlow)

To address memory vs. performance tradeoff, find abstractions:

independent from clean slate solutions

specific platform representations

Impact of distributed platforms?
Two classifiers are *equivalent* if both have the same result on any lookup key.

All address bit identities participate in RX classification.
Both schemes are equivalent on the system level.

Can we do it better? Yes, balancing bits between RX and TX is required.
Intuition: if the filters of a classifier do not intersect, their order is not important.
Action-order independence

Idea: a classification result is the action of the rule with the highest priority. Two rules with filters $F_1^B$ and $F_2^B$ are action-order-independent if either (a) they have the same action or (b) $F_1^B$ and $F_2^B$ are disjoint.

Example

<table>
<thead>
<tr>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>→ Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>→ $A_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>→ $A_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>→ $A_2$</td>
</tr>
</tbody>
</table>

two bits are required for filter-oi.

<table>
<thead>
<tr>
<th>#1</th>
<th>#3</th>
<th>→ Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>→ $A_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>→ $A_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>→ $A_2$</td>
</tr>
</tbody>
</table>

only one bit for action-oi.

(b) Two-stage with RX action-order independence and TX lookup to find an action.
Non-conflicting rules

Definition

Two rules $R^X_1$ and $R^X_2$ with different actions, $R^X_1 \prec R^X_2$, are conflicting with respect to bit indices $B \subset X$ if there is a header $H^X$ matching $R^X_2$ that is not matched by $R^X_1$ and $R^B_1$ matches $H^B$.

Example

In the following classifier with $|X| = 3$

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>→ A₁</th>
<th>→ A₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^X_1$</td>
<td>(*)&amp;</td>
<td>1</td>
<td>0</td>
<td>→ A₁</td>
<td></td>
</tr>
<tr>
<td>$R^X_2$</td>
<td>1</td>
<td>1</td>
<td>*</td>
<td>→ A₂</td>
<td></td>
</tr>
</tbody>
</table>

$R^X_1$ and $R^X_2$ are conflicting with respect to $B = \{1, 2\}$ ($R^B_1$ matches $H^B = (1 1)$ but $R^X_1$ does not match $H^X = (1 1 1)$ when $R^B_1$ matches $H^B$ and it matches $R^X_2$); for $B = \{2, 3\}$, $R^X_1$ and $R^X_2$ are not conflicting with respect to $B$.

<table>
<thead>
<tr>
<th>#2</th>
<th>#3</th>
<th>→ A₁</th>
<th>→ A₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^B_1$</td>
<td>(1</td>
<td>0)</td>
<td>→ A₁</td>
</tr>
<tr>
<td>$R^B_2$</td>
<td>1</td>
<td>*</td>
<td>→ A₂</td>
</tr>
</tbody>
</table>

(c) Two-stage with RX non-conflicting rules and TX lookup to find an action.
Equivalent representations

(a) One-stage forwarding

RX LCs

01000 → A1, LC1
01001 → A2, LC1
00111 → A3, LC2
11100 → A4, LC2
11000 → A4, LC2
***** → drop

switch fabric

TX LCs

LC1 A1
A2

LC2 A3
A4

: x7

(b) Two-stage forwarding

RX LCs

0100* → LC1
00111 → LC2
11*00 → LC2
***** → drop

switch fabric

TX LCs

LC1

LC2

00111 → A3
11*00 → A4

: x7

(a) One-stage with RX filter-order independence and false-positive check on TX.

(b) Two-stage with RX action-order independence and TX lookup to find an action.

(c) Two-stage with RX non-conflicting rules and TX lookup to find an action.
Equivalent vs. non-equivalent representations

(a) One-stage with RX filter-order independence and false-positive check on TX.

(b) Two-stage with RX action-order independence and TX lookup to find an action.

(c) Two-stage with RX non-conflicting rules and TX lookup to find an action.

(a) One-stage with RX action-order independence and no false-positive check on TX.

(b) Two-stage with RX action-order independence and TX lookup to distinguish between actions.

(c) Two-stage with RX non-conflicting rules and TX lookup to distinguish between actions.
Network-wide representation
We considered FIB representations on distributed platforms based on various structural properties.

- transparent to PD representations
- no effect on original objective vs. compact routing
- transparent to clean-slate solutions
- improvements in memory and/or lookup time

Additional directions:

- How to run clean-slate solutions on existing IPv4 infrastructure (ICNP’17)
- Approximate classifiers with controlled errors.
processing multiple packets
Buffering architectures define how input and output ports of a network element are connected.

**Queue:**
- admission policy (tail-drop, RED)
- processing policy (FIFO, SRPT)

**Port:** scheduling policy (DRR, WFQ)

**Buffer:** admission policy (LQD)
Why is a buffer management important?

- Imbalance between incoming and outgoing packet rates.
- Overprovisioning of buffer capacity is not viable.
- Buffering architecture and management impact performance and cost of network elements.
Traditional networks allow only a predefined set of policies. Incorporation of new management policies requires CP/DP code changes and respin of implementing hardware. Objectives beyond fairness and additional traffic properties lead to new challenges (pFabric SIGCOMM14, pHost CoNEXT15). Traditional approaches and current SDN deal with efficient representation of packet classifiers.
Requirements for Software-Defined buffer management

**Expressivity**: should be expressive enough

**Simplicity**: policies should be expressible concisely with a limited set of basic primitives

**Performance**: implementations of policies should be efficient

**Dynamism**: specification and provision of new policies should be possible at run-time without any code changes and HW respins
From single packets to packet streams

Primitives:

**OF,P4**: packets, fields, flows, tables, etc.

**OpenQueue**: packets, queues, buffer, ports

<table>
<thead>
<tr>
<th>$K$</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$A_1$</td>
</tr>
<tr>
<td>$R_2$</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>$A_2$</td>
</tr>
<tr>
<td>$R_3$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>$A_3$</td>
</tr>
</tbody>
</table>

Problem

*What should be flexible to implement admission, processing, and scheduling?*

*How to represent this expressiveness?*
Queue complexity

How complex should be a queue primitive?
“Towards Programmable Packet Scheduling” HotNets’15

Observation
For any deterministic (or probabilistic) MQ policy ALG, there exists an SQ policy that for any input sequence transmits exactly the same set of packets (or a random set of packets with the same distribution) as ALG.

Observation
In the worst case, MQ architecture with m queues has time complexity of operations at least $O(m/\log B)$ times better than SQ simulating the same order.
Expressiveness vs. Simplicity

Buffer management policies mostly deal with extreme cases. Implementation: priority queues (operating on internal state and packet’s info).

Queue:
- admission policy: 1
- processing policy: 1

Port: scheduling policy 1

Buffer: admission policy 1

Design considerations:
- a single copy of each packet; operations on packet pointers.
- user-defined expressions for priorities are immutable.
OpenQueue primitives
Queue primitive

```plaintext
Queue {
    // user-specified at declaration
    size       // size in bytes           [r, cons]
    buffer     // allocating buffer      [r, cons]
    // primitive properties
    currSize   // current size            [r, dyn]
    // admission -- user-specified at decl.
    admPrio(p1, p2)  // pushOut comparat. [bool fun]
    congestion()     // drop(P) condit.   [drop cond]
    postAdmAct()     // [{mark, notify, modify} comp]
    weightAdm       // adm. priority     [rw, dyn]
    // processing -- user-specified at decl.
    procPrio(p1, p2)  // proc comparat.  [Packet comp]
    getHOL()         // HOL packet         [Packet fun]
    // scheduling -- user-specified at decl.
    weightSched     // scheduling prio. [rw, dyn]
}

congestion() = |
    (currSize >= .95*size, drop(1)) .|
    (currSize >= .9*size, drop(.9)) .|
    (currSize >= .75*size, drop(.5))|

procPrio(p1, p2) = (p1.arrival < p2.arrival)
```
Buffer { 
   // primitive properties
   currSize // current size [r, dyn]
   getBestQueue() // on weightAdm [Queue fun]
   getCurrQueue() // admitted one [Queue fun]
   // user-specified at declaration
   size // size [r, cons]
   // admission -- user-specified at decl.
   congestion() // drop(P) [drop cond]
   queuePrio(q1, q2) // compare q-s [bool fun]
   postAdmAct() // [{mark, notify, modify} cond]
}

queuePrio(q1, q2) = (q1.currSize < q2.currSize)
Port
{
    // primitive properties
    getBestQueue()       // on weightSched   [Queue fun]
    getCurrQueue()      // scheduled one    [Queue fun]
    // scheduling user-specified at decl.
    schedPrio(q1, q2)    // compare q-s    [bool fun]
    postSchedAct()       // [{mark, notify, modify} cond]
}

Packet
{
    size               // size in bytes     [r, cons]
    value              // virtual value    [r, cons]
    processing         // nb of cycles    [r, dyn]
    arrival            // arrival time     [r, cons]
    slack              // offset in time   [r, cons]
    queue              // target queue id [r, cons]
    flow               // flow id          [r, cons]
}
Examples in OpenQueue
Example: Single queue

Impact of admission and processing orders

```
// priorities for admission and processing
fifo(p1, p2) = (p1.arrival < p2.arrival)
srpt(p1, p2) = (p1.processing < p2.processing)
rsrpt(p1, p2) = (p1.processing > p2.processing)
// congestion conds. considered.
// trigger when occupancy exceeds size.
defCongestion() =
    lambda q, (q.currSize >= q.size, drop(1))

// buffering architecture specification
q1 = Queue(B);
out = Port(q1);
// admission control
q1.admPrio(p1, p2) = rsrpt(p1, p2);
q1.congestion = defCongestion(q1);
// processing policy
q1.procPrio(p1, p2) = srpt(p1, p2);
```

<table>
<thead>
<tr>
<th>admPrio</th>
<th>procPrio</th>
<th>OPT/ALG</th>
</tr>
</thead>
<tbody>
<tr>
<td>fifo()</td>
<td>fifo()</td>
<td>O(k)</td>
</tr>
<tr>
<td>rsrpt()</td>
<td>fifo()</td>
<td>O(logk)</td>
</tr>
<tr>
<td>rsrpt()</td>
<td>srpt()</td>
<td>1 (optimal)</td>
</tr>
</tbody>
</table>
Example: Multiple queues

Impact of scheduling.

```
// LQF: HOL packet from Longest-Queue-First
lqf(q1,q2) = (q1.currSize > q2.currSize);
// SQF: HOL packet from Shortest-Queue-First
sqf(q1,q2) = (q1.currSize < q2.currSize);
// MAXQF: HOL packet from queue that
// admits max processing
maxqf(q1,q2) = (q1.weightSched > q2.weightSched);
// MINQF: HOL packet from queue that admits
// min processing
minqf(q1,q2) = (q1.weightSched < q2.weightSched);
// CRR: Round-Robin with per cycle resolution
crr(q1,q2) = (q1.weightSched < q2.weightSched);
crrPostSchedAct() =
    lambda port,
    let q = port.getCurrQueue() in
    (true, // condition
    modify(q.weightSched := q.weightSched+k));
// PRR: Round-Robin with per packet resolution
prr(q1,q2) = (q1.weightSched < q2.weightSched);
prrPostSchedAct() =
    lambda port,
    (let q = port.getCurrQueue() in
    (q.getHOL().processing == 0, // condition
    modify(weightSched := weightSched+k*k)));

// initializing schedWeight for CRR
q1.weightSched = 1; ...; qk.weightSched = k;
// postSchedAct updating schedWeight
out.postSchedAct = crrPostSchedAct(out);
```
OpenQueue is a concise yet expressive language to define buffer management policies.

new buffer management policies do not require control/data-plane code changes.

**Conclusion:** OpenQueue can enable and accelerate innovation in buffering architectures and their management, similar to programming abstractions as P4.
network virtualization
Motivations

Network infrastructure is an expensive resource requiring complex management. Various bandwidth allocation and routing methods are very complex and time consuming.

**Question:** How to operate on simplified network without loosing an original network capabilities transparently to bandwidth allocation methods?

**Capacity planning:** reduce unnecessary link capacity. Network topology remains unchanged. (usecase: geo-distributed data centers)

**Network topology transformations**

Network topology can be changed.

- preserves one-to-one mapping of bandwidth allocations;
- feasibility of bandwidth allocations (e.g., Fastpath transport);
- how to exploit extra capacity to further simplify networks?
Network topology transformations
Routing equivalence

Denote by $\text{Path}_G(A, B)$ the set of paths in network $G = (V, E)$ that begin in $A \subseteq V$ and end in $B \subseteq V$.

**Definition**

Two networks $G = (V, E, w, S, D)$ and $G' = (V', E', w', S, D)$ are *routing equivalent* if there is a one-to-one mapping $g : \text{Path}_G(S, D) \rightarrow \text{Path}_{G'}(S, D)$ between paths from sources to corresponding destinations on $G$ and $G'$ that preserves bandwidth allocations.

---

The transformations (a)-(b) reduce simultaneously the number of edges in the network, the number of vertices in the network, and the total capacity of all edges.

**Theorem**

*Transformations (a)-(b) outputs a routing equivalent network.*
For two networks $G = (V, E, w, S, D)$ and $G' = (V', E', w', S, D)$ with the same set of sources and destinations, a *bandwidth-preserving routing transformation* is a function $g : \text{Path}_G(S, D) \rightarrow \text{Path}_{G'}(S, D)$ between paths from sources to corresponding destinations on $G$ and $G'$.

(c) Merge parallel edges  
(d) Removing self-loops

**Theorem**

*Transformations (c)-(d) are bandwidth preserving routing transformation.*
Tradeoff between extra capacity and simplicity
Braess’ Paradox

Removing \((a, b)\)-edge leads to the simplest graph.
Using extra capacity to simplify network

Find the best possible allocation of extra capacity to simplify the resulting graph.
short summary

two types of network topology transformations;
study a tradeoff between extra capacity and simplicity

Our ultimate goal is to understand how to represent a network as a virtual switch and operate on top of it. This work is only the first step towards this goal.

Delayed information and action in on-line algorithms FOCS'98
Current projects

Towards in-network processing of data streams (design on FD.io VPP - Cisco grant).
Cost-efficient Infrastructure for Compute-aggregate Problems (taxonomy of aggregation functions vs. Microsoft cam-cube) (ICNP’17, submitted).
Virtualization of heterogeneous computing resources (CPU+FPGA) (ongoing).
Self-adapting buffer-management policies (user defines objectives and traffic properties) (ongoing).
Datacenter transports with provable guarantees (bounded delay vs. average flow compl. time) (ongoing).
Composition of heterogeneous control planes (ONS’14, ICNP’14).
Questions?
kirill.kogan@imdea.org