

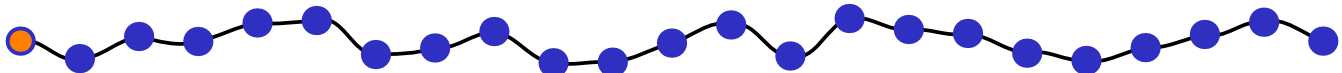
# Packet Filtering for FPGA–Based Routing Accelerator

•  
David Antoš<sup>1,2</sup>, Vojtěch Řehák<sup>1</sup>,  
and Petr Holub<sup>2,3</sup>

<sup>1</sup>Faculty of Informatics, Masaryk University, Brno

<sup>2</sup>CESNET, z. s. p. o, Prague

<sup>3</sup>Institute of Computer Science, Masaryk University, Brno



# Introduction

- Personal computer as mid-sized router
  - throughput limited by internal PC architecture
  - off-loading the traffic to an *acceleration card*
- FPGA-based hardware accelerator
  - *packet classification* and *filtering*
  - single-lookup classification engine (CAM and SRAM)
- Main parts of this talk
  - expressiveness of filters
  - Filtering Decision Diagrams (FDD)



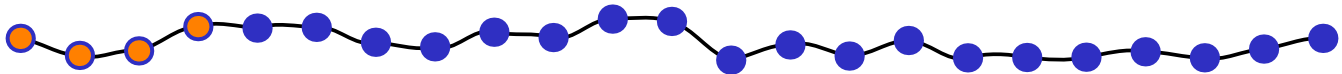
# Expressiveness of Existing Filters

- Preliminary step for proposing FDDs
  - how they can be transformed to the target hardware
  - how their “expression abilities” differ
- Analysing common packet filters
  - “BSD style:” pf, IP Filter, IPFIREWALL (IPFW)
  - “Linux style:” iptables/netfilter, ipchains
  - “commercial boxes:” CISCO, Juniper
- Inputs, outputs, “way of processing”



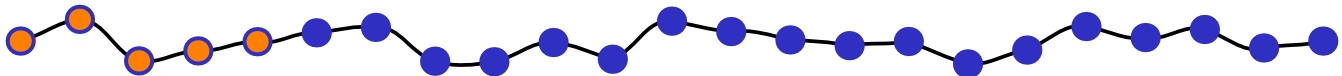
# Inputs, Outputs

- Inputs
  - packet headers and implicit packet properties
  - observables
- Outputs
  - basic actions: just one mandatory in each rule
    - ★ accept, deny
  - modifiers: extra processing, zero or more
    - ★ log, count
- $F: \text{Observables} \rightarrow \text{BasicActions} \times 2^{\text{Modifiers}}$



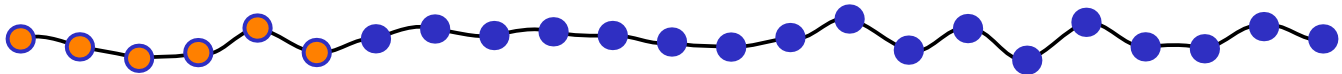
# Processing of Filters I

- Execution order: first-match, last-match (quick)
- First- to last-match
  - reversing rule order
  - or marking all rules quick
- Last-match with quick to first-match
  - “semantical matching to  $i$ -th rule”
    - i. match the  $i$ -th rule
    - ii. do not match any following rule
    - iii. do not match any preceding quick rule



# Processing of Filters II

- Named blocks of rules used as subroutines (and similar features)
  - can be expanded
- Filter application in the system
  - per-interface or common
    - ★ → concatenating the filters, adding “and interface  $i$ ” to appropriate rules
    - ★ ← distributing the rule set to all interfaces





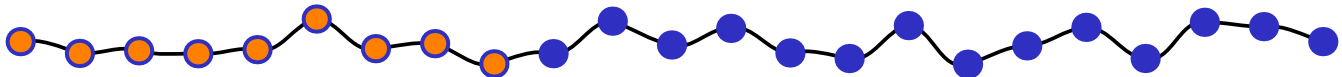
# Processing of Filters IV

- INPUT, FORWARD, OUTPUT → input/output
- expressing addresses of the host: me
  - input = INPUT with “and destined to me”
  - output =
    - OUTPUT with “and sent by me”
    - FORWARD with “and not sent by me”
- testing both interfaces necessary in the output filter



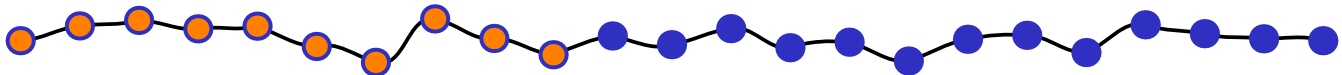
# Filtering in the Accelerator

- Forwarding filter for switched packets (FORWARD)
- Input filter for packets destined to the host computer
  - optional but advisable
- First-match filter
  - rules: conjunctions of terms
  - “or” can be expanded into several rules



# Filtering Decision Diagrams (FDD)

- A representation of packet filters
- Can be rewritten into the target hardware architecture
  - first match Content Addressable Memory
  - comparison instructions in static RAM
- *Multi-terminal Binary Decision Diagram*—rooted directed acyclic graph with two types of vertices
  - nonterminal  $v$  with a Boolean variable  $\text{VAR}(v)$  and two successors,  $\text{LOW}(v)$  and  $\text{HIGH}(v)$
  - terminal labelled with elements of a finite set



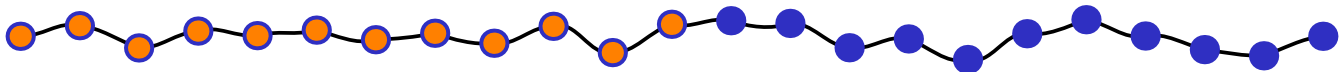
# FDD Variables

- *FDD variables*—set of all possible filtering terms of the filtering language
  - `proto tcp, saddr 147.251.54.0/24, dport 0-1023`
  - exact, prefix, range match
- *Class* of FDD variables—variables testing a single field
  - e.g., `saddr` tests
- Special symbol *HSL*—“hic sunt leones”



# FDD

- An FDD is a MTBDD over FDD variables; terminals are filtering actions and *HSL*
- Reduced FDD:
  - for nodes  $u, v$ :  $\text{VAR}(u) = \text{VAR}(v)$ ,  $\text{LOW}(u) = \text{LOW}(v)$ ,  $\text{HIGH}(u) = \text{HIGH}(v)$  implies  $u = v$ ,
  - no node  $u$  exists s.t.  $\text{LOW}(u) = \text{HIGH}(u)$
- Ordered FDD: all paths of the FDD respect an order of variables  $<$ 
  - total order/class order/no order



# Creating FDD Nodes

- $\text{FDDCreate}(n, l, h)$ 
  - returns a node  $u$  s.t.  $\text{VAR}(u) = n$ ,  $\text{LOW}(u) = l$ , and  $\text{HIGH}(u) = h$
  - standard BDD procedure
  - hash table representation
  - keeps the structure reduced



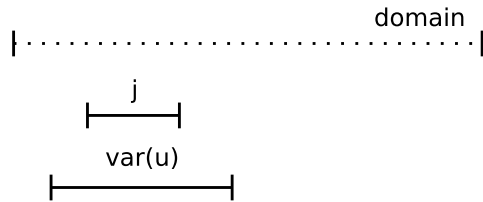
# Restriction I

- Standard BDD restriction: assign value to a Boolean variable
- FDD: relationship among variables exists
- Given an FDD with root  $u$ , variable  $j$  assigned to  $v$ , restriction  $\text{FDDRestrict}(u, j, v)$  computes an FDD with all tests following from assigning  $j$  to  $v$  eliminated
- The question in each node is “based on the fact we know the result of test  $j$ , is the test in this node necessary?”
- *Semantical restriction*

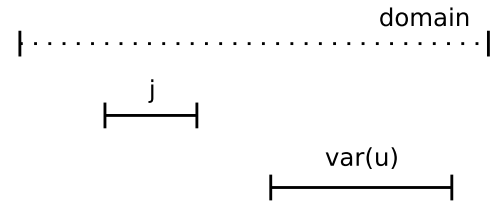


# Restriction II

j holds

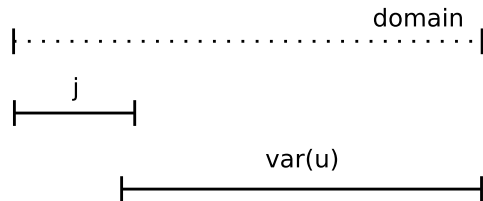


(a)

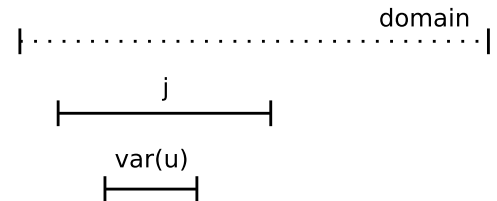


(b)

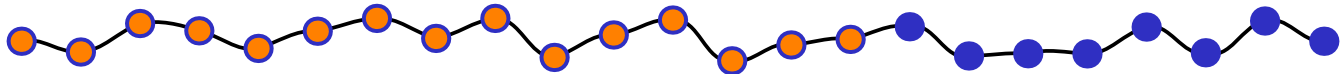
j does not hold



(c)

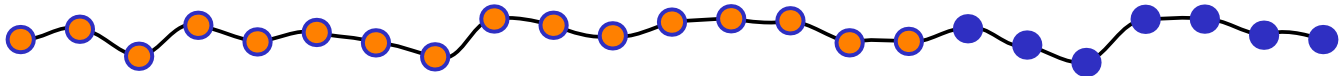


(d)

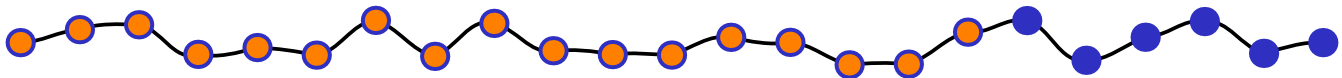
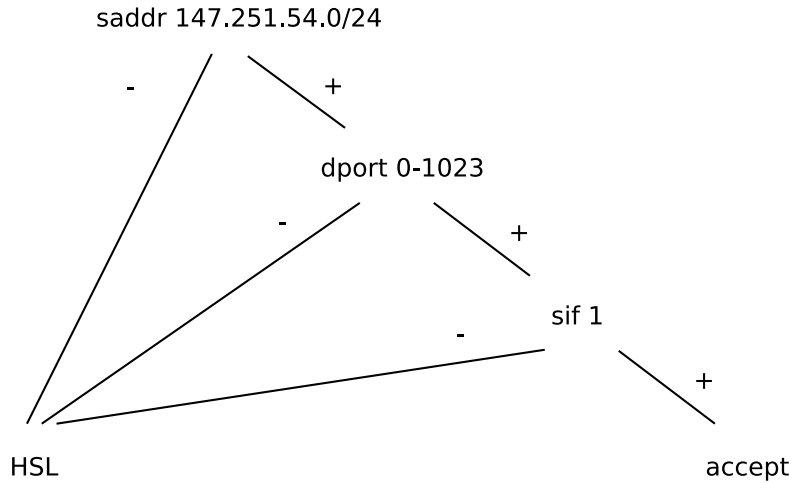
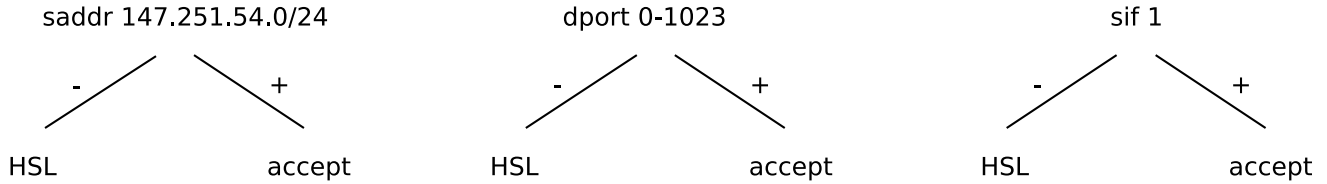


# Restriction III

```
function FDDRestrict( $u, j, v$ )
  if FDDIsTerminal( $u$ ) then return  $u$ 
  fi
  Return appropriate FDDRestrict(HIGH/LOW( $u$ ),  $j, v$ )
  if  $\text{VAR}(u) < j$  then /* the search can be stopped */
    return  $u$ 
  else
    return FDDCreate( $\text{VAR}(u)$ ,
                    FDDRestrict(HIGH( $u$ ),  $j, v$ ),
                    FDDRestrict(LOW( $u$ ),  $j, v$ ))
  fi
```



# A Rule to FDD



# A Rule Set to an FDD I

- Having an FDD representing rules up to rule  $i$ , we add rule  $i + 1$
- Principle: rewrite the *HSL* to the rule  $i + 1$ 
  - the whole rule set can be converted repeating this step
  - *HSL* is eliminated by the default rule
    - ★ the last rule in the set



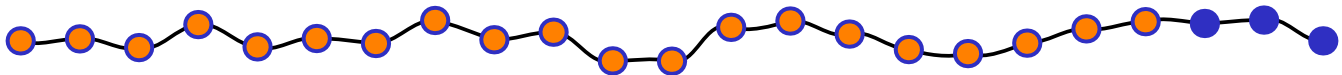
# A Rule Set to an FDD II

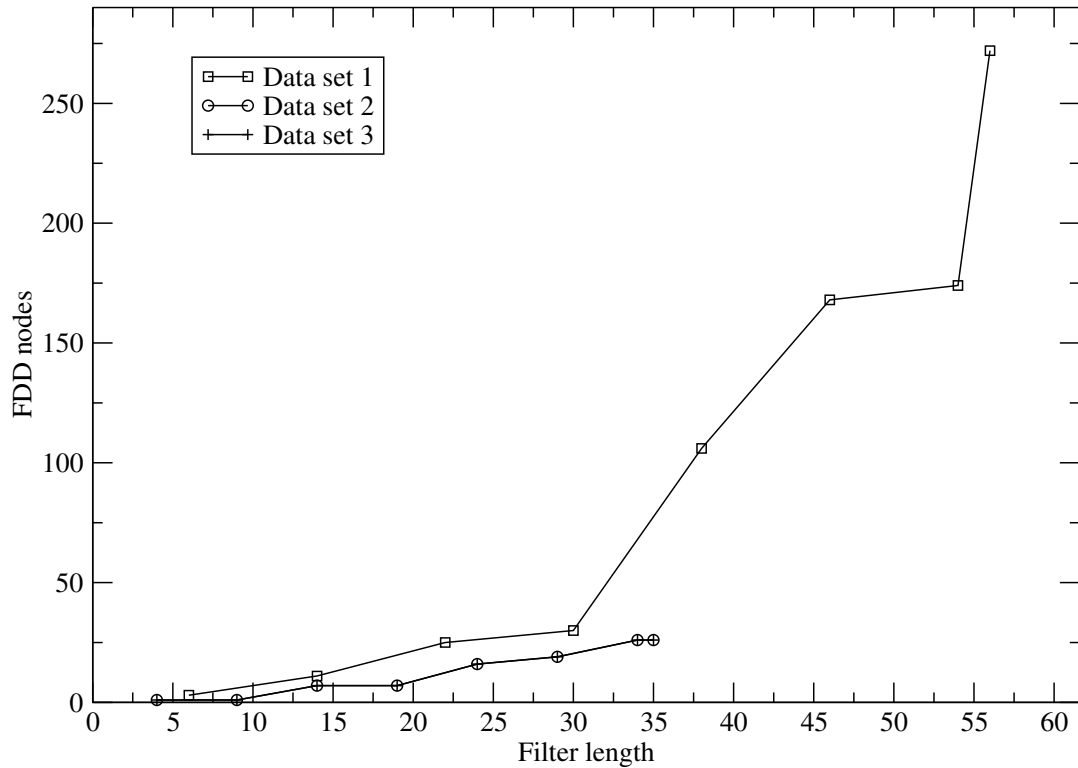
```
function FDDAppend( $u_1, u_2$ )
  if  $u_1 = HSL$  then return  $u_2$ 
  fi
  if FDDIsTerminal( $u_1$ ) then return  $u_1$ 
  fi
   $h =$  smallest of nodes  $u_1, u_2$  in relation  $<$ 
   $u_1^h =$  FDDRestrict( $u_1, \text{VAR}(h), \text{high}$ )
   $u_1^l =$  FDDRestrict( $u_1, \text{VAR}(h), \text{low}$ )
   $u_2^h =$  FDDRestrict( $u_2, \text{VAR}(h), \text{high}$ )
   $u_2^l =$  FDDRestrict( $u_2, \text{VAR}(h), \text{low}$ )
  return FDDCreate( $h, \text{FDDAppend}(u_1^h, u_2^h),$ 
                   $\text{FDDAppend}(u_1^l, u_2^l)$ )
```



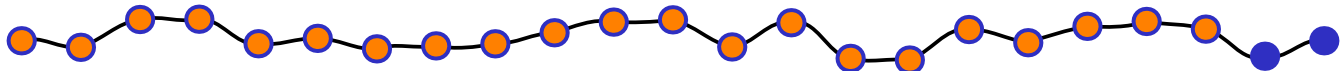
# Complexity

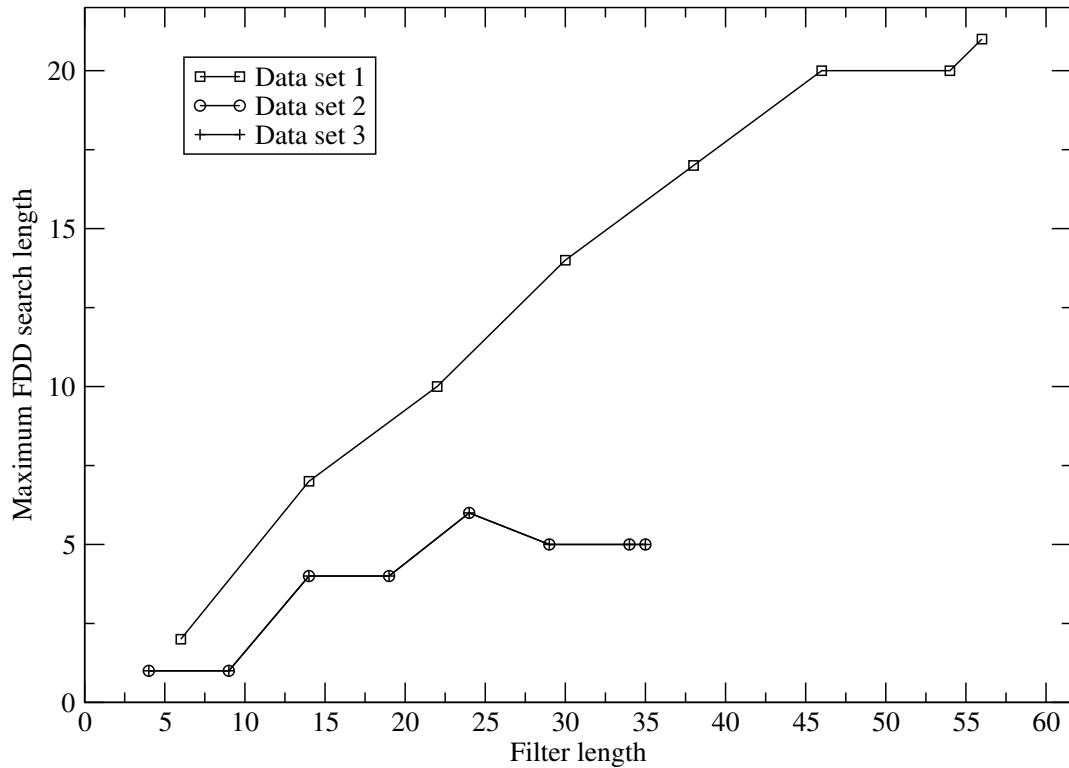
- Filter consisting of  $m$  rules, each up to  $n$  terms
- Space
  - exponential
  - BDDs just behave that way... theoretically
  - measurements: not that bad
- Time
  - evaluating the filter in software:  $O(mn)$  comparisons
  - FDD: ditto
  - measurements: at most 27% of the theoretical maximum





Data set 1: 56 filtering rules, 105 filtering terms





Data set 1: 56 filtering rules, 105 filtering terms



# Conclusions

- Analysis of expressive power of common packet filters
- FDD structure compared to BDDs and Interval Decision Diagrams (IDD)
  - granularity of tests (BDD: a single bit at a time)
  - two way branching (IDD: multi-way)
  - direct rule order encoding (via *HSL*)
- FDD can be combined with routing and rewritten to the hardware architecture
- Future work: a method to find a suitable variable ordering

