Implementation and Modification for CPE Routers: Filter Rule scan Optimization, IPsec Interface and Ethernet switch

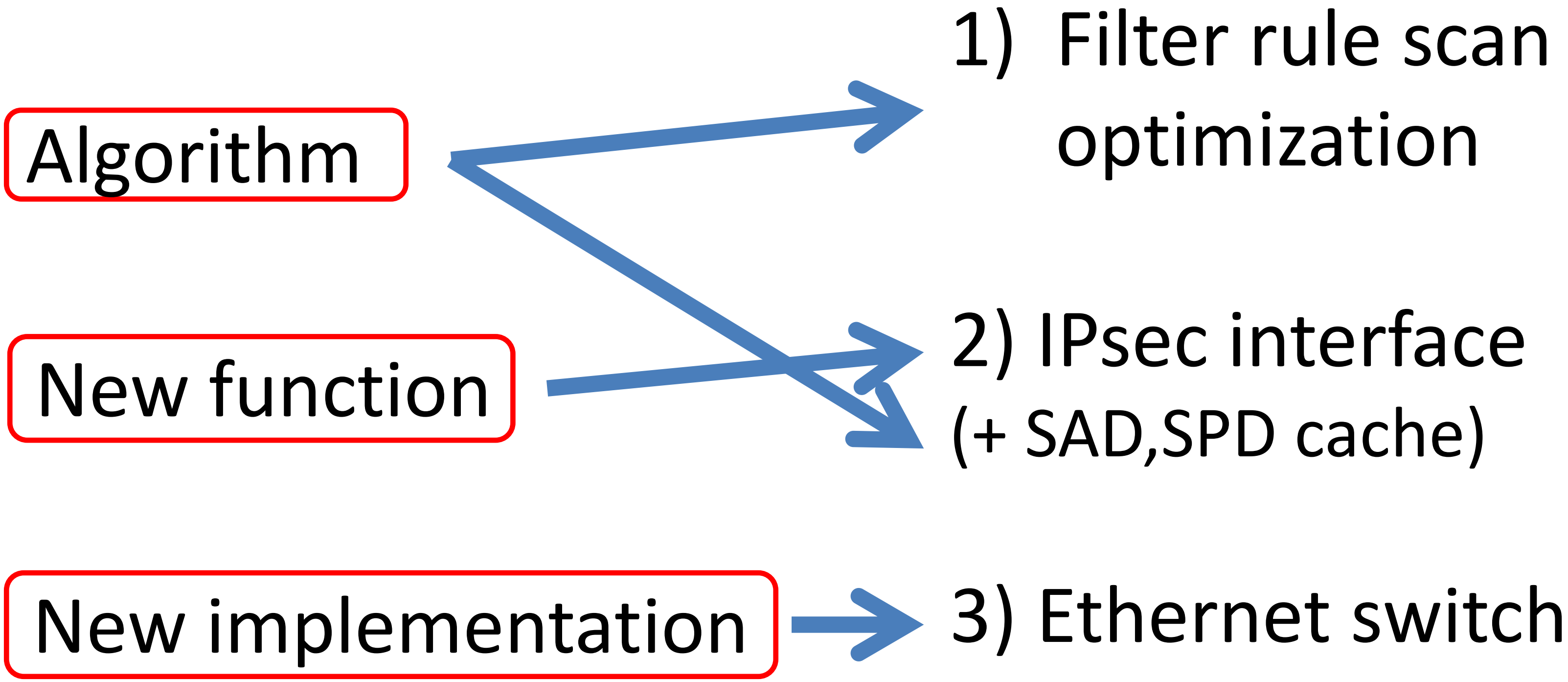
Masanobu SAITOH <msaitoh@netbsd.org>

Hiroki Suenaga <hsuenaga@iij.ad.jp>

A lot of work which might be useful for others

- Algorithms which can be useful to other implementations
- New functions that *BSD don’t have them yet.
- New implementations that existing implementation didn’t match our requirement.

Some cases



Algorithm

New function

New implementation

1) Filter rule scan optimization

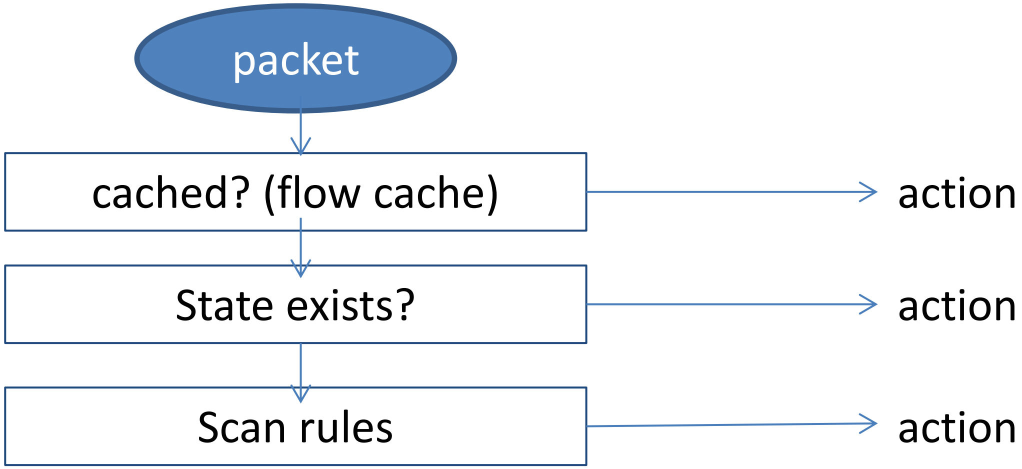
2) IPsec interface (+ SAD,SPD cache)

3) Ethernet switch

1. Filter rule scan optimization

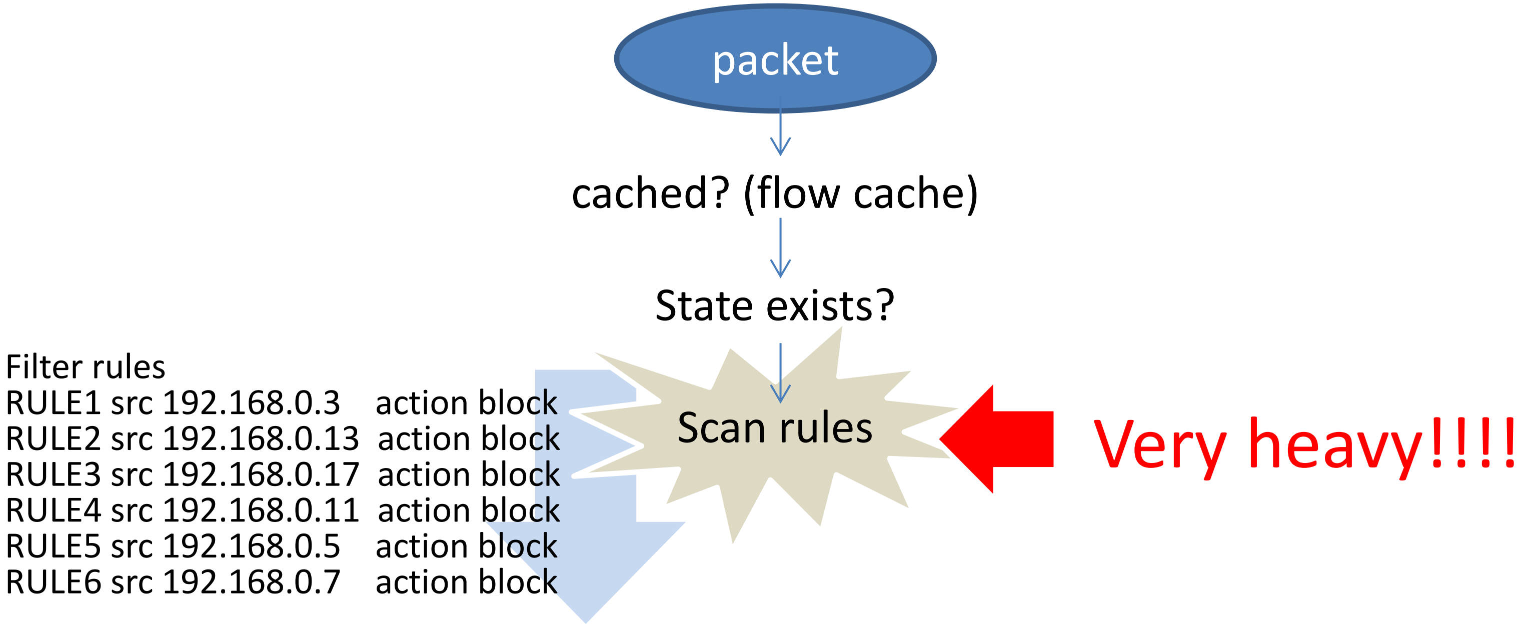
filter optimization

- What is our packet filter
  - Compare addresses, ports, protocols, etc of a packet and rules.
    If match, do action of rule (pass, block)
- How it works?



filter optimization

- Problem
  - It’s very slow to scan and evaluate many filter rules
    - State? Yes, already used.
    - Cache result? Yes, already used.
    - Otherwise?



packet

cached? (flow cache)

State exists?

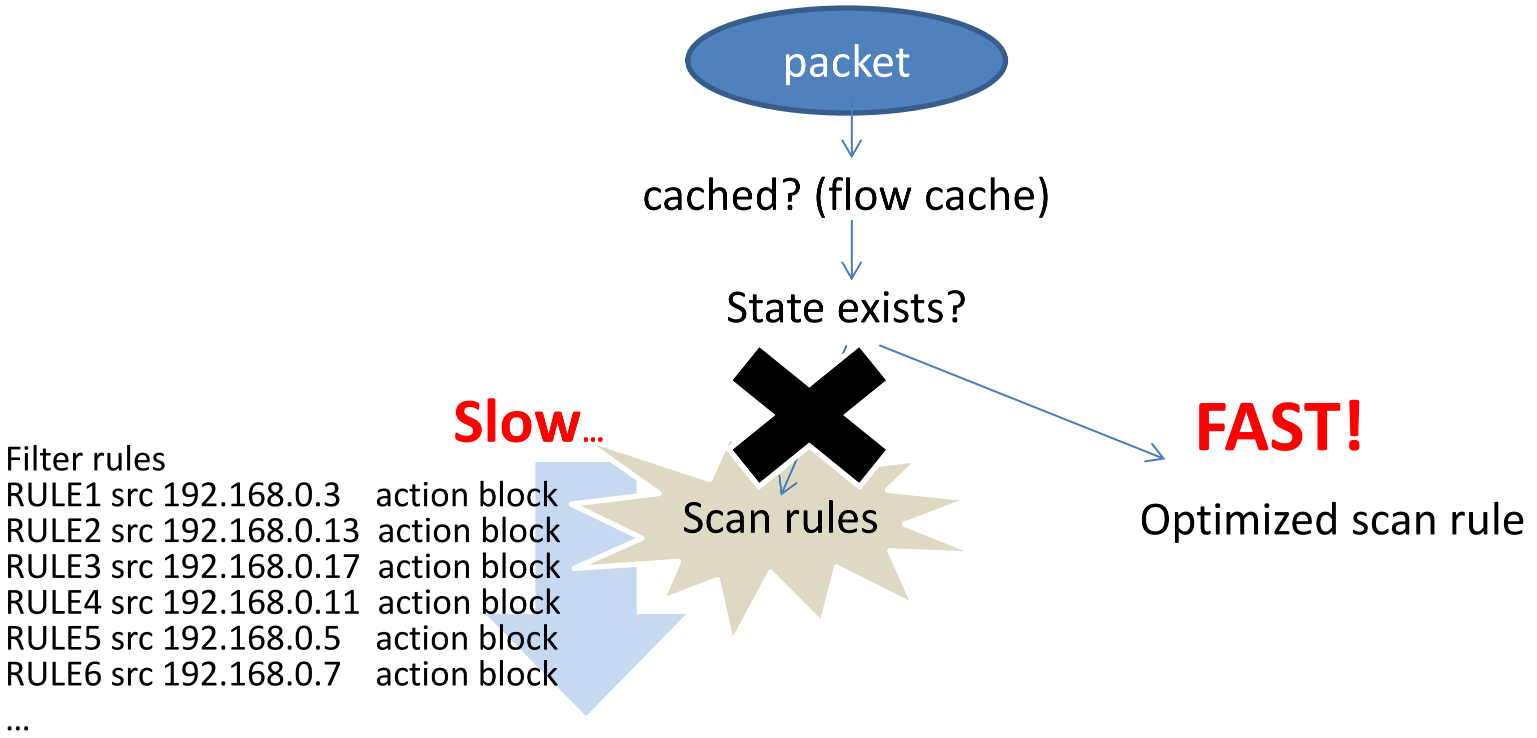
Scan rules

Very heavy!!!!

Filter rules
RULE1 src 192.168.0.3 action block
RULE2 src 192.168.0.13 action block
RULE3 src 192.168.0.17 action block
RULE4 src 192.168.0.11 action block
RULE5 src 192.168.0.5 action block
RULE6 src 192.168.0.7 action block
…

filter optimization

Optimize filter rule scan when configured



packet

cached? (flow cache)

State exists?

Slow…

FAST!

Scan rules

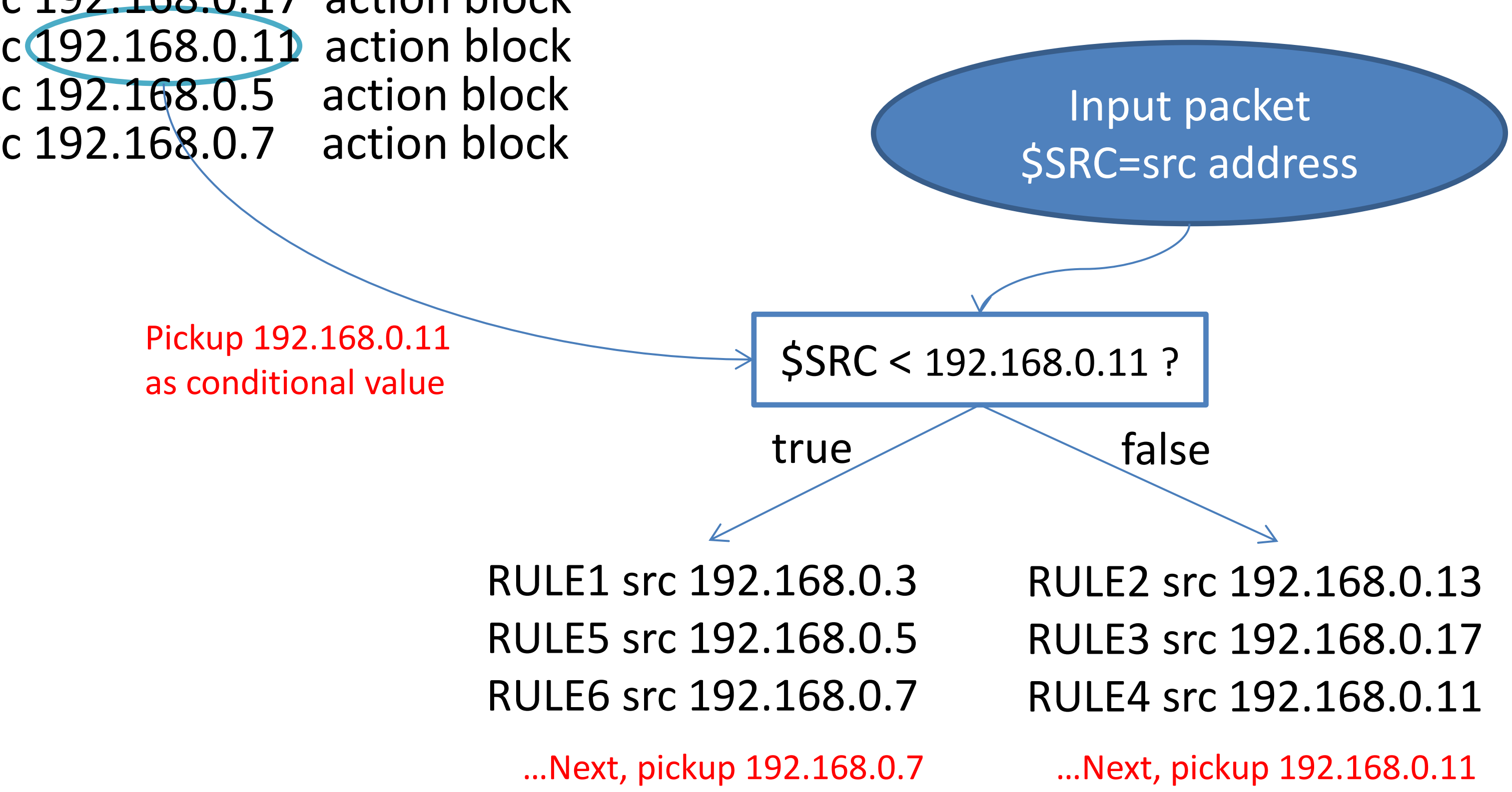
Optimized scan rule

Filter rules
RULE1 src 192.168.0.3 action block
RULE2 src 192.168.0.13 action block
RULE3 src 192.168.0.17 action block
RULE4 src 192.168.0.11 action block
RULE5 src 192.168.0.5 action block
RULE6 src 192.168.0.7 action block
…

simple case

6 filter rules to scan

RULE1 src 192.168.0.3 action block
RULE2 src 192.168.0.13 action block
RULE3 src 192.168.0.17 action block
RULE4 src 192.168.0.11 action block
RULE5 src 192.168.0.5 action block
RULE6 src 192.168.0.7 action block



Pickup 192.168.0.11
as conditional value

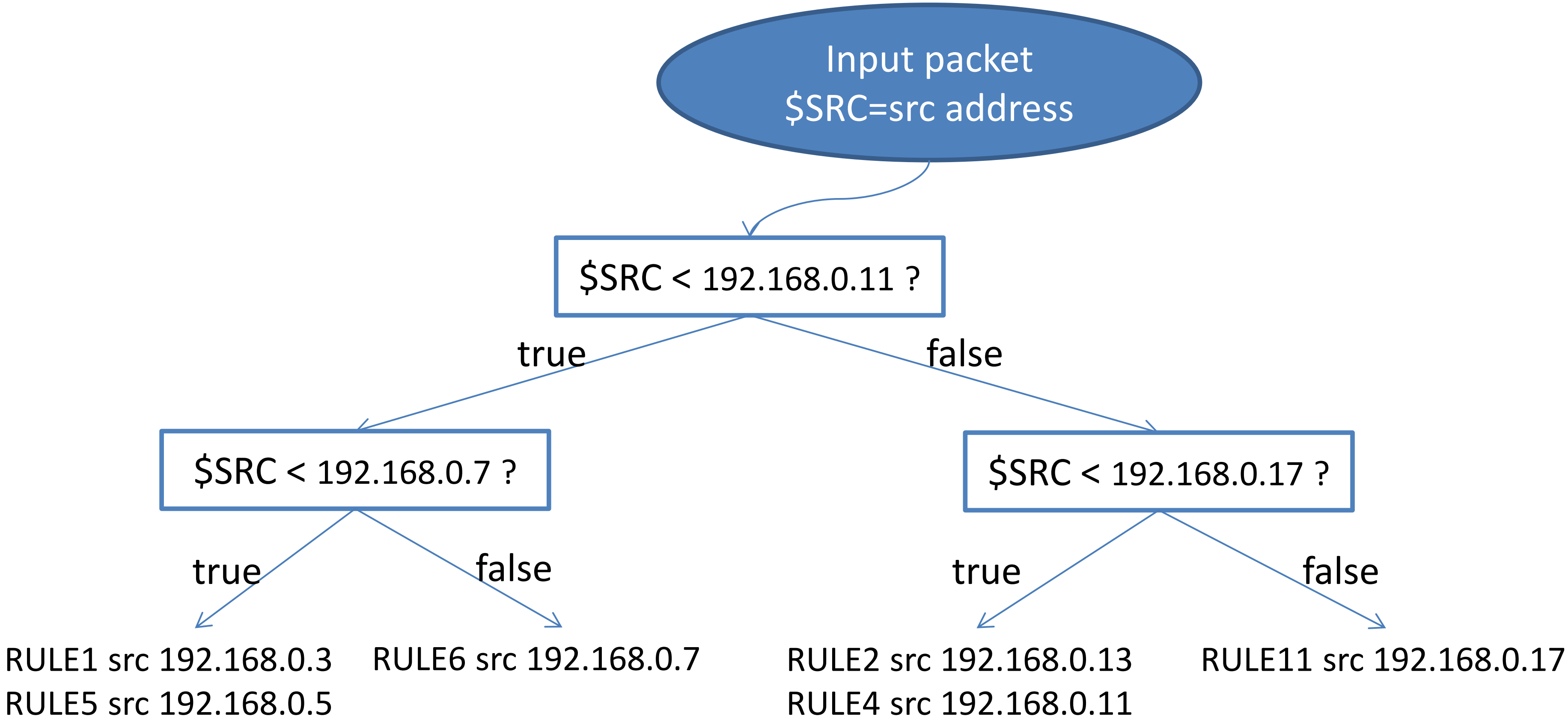
RULE1 src 192.168.0.3
RULE5 src 192.168.0.5
RULE6 src 192.168.0.7

…Next, pickup 192.168.0.7

RULE2 src 192.168.0.13
RULE3 src 192.168.0.17
RULE4 src 192.168.0.11

…Next, pickup 192.168.0.11

simple case



Input packet
$SRC=src address

$SRC < 192.168.0.11 ?

true

$SRC < 192.168.0.7 ?

true

RULE1 src 192.168.0.3
RULE5 src 192.168.0.5

false

RULE6 src 192.168.0.7

false

$SRC < 192.168.0.17 ?

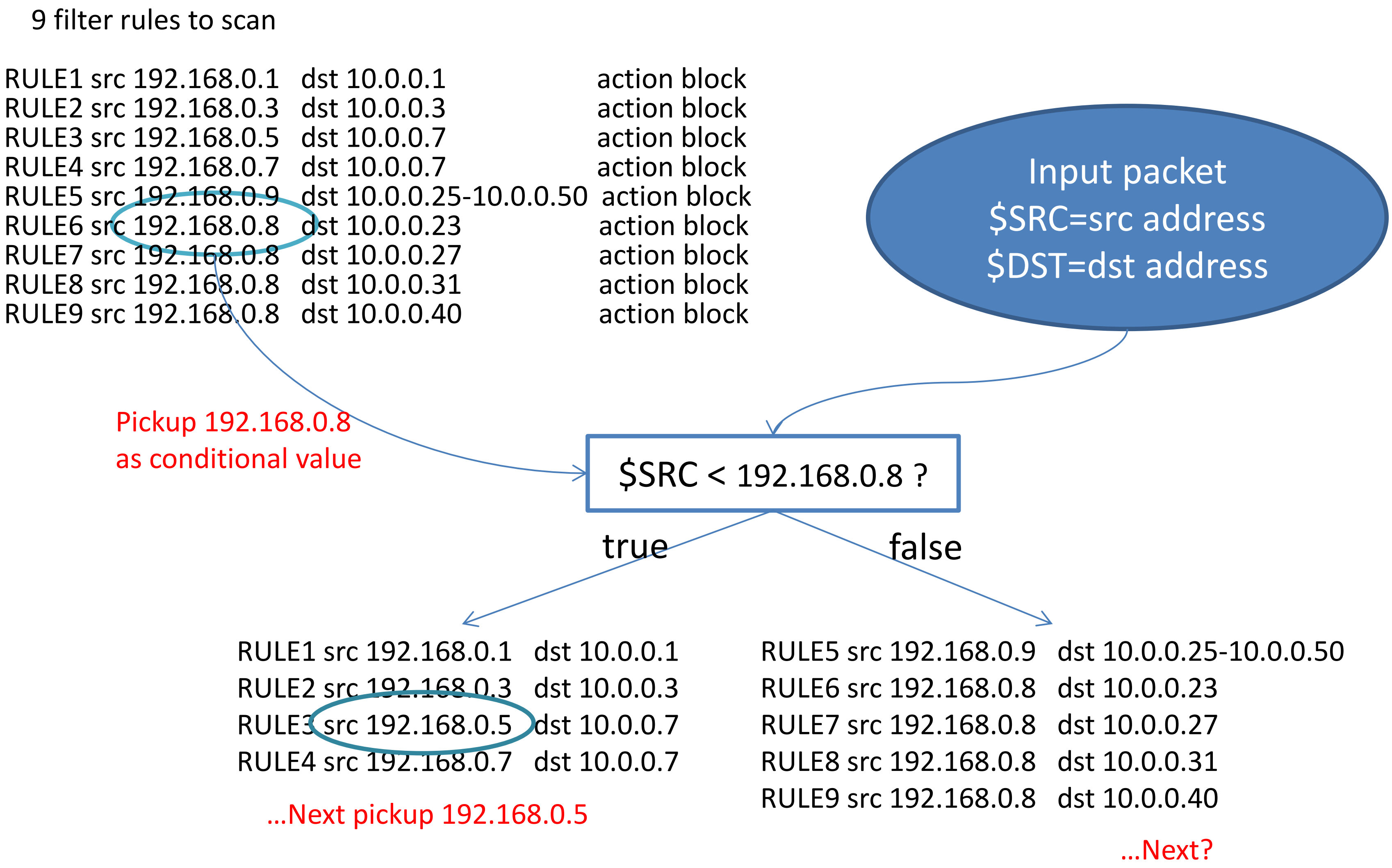
true

RULE2 src 192.168.0.13
RULE4 src 192.168.0.11

false

RULE11 src 192.168.0.17

src and dst with address range



9 filter rules to scan

RULE1	src 192.168.0.1	dst 10.0.0.1	action block
RULE2	src 192.168.0.3	dst 10.0.0.3	action block
RULE3	src 192.168.0.5	dst 10.0.0.7	action block
RULE4	src 192.168.0.7	dst 10.0.0.7	action block
RULE5	src 192.168.0.9	dst 10.0.0.25-10.0.0.50	action block
RULE6	src 192.168.0.8	dst 10.0.0.23	action block
RULE7	src 192.168.0.8	dst 10.0.0.27	action block
RULE8	src 192.168.0.8	dst 10.0.0.31	action block
RULE9	src 192.168.0.8	dst 10.0.0.40	action block

Input packet
$SRC=src address
$DST=dst address

Pickup 192.168.0.8
as conditional value

$SRC < 192.168.0.8 ?

true

RULE1	src 192.168.0.1	dst 10.0.0.1
RULE2	src 192.168.0.3	dst 10.0.0.3
RULE3	src 192.168.0.5	dst 10.0.0.7
RULE4	src 192.168.0.7	dst 10.0.0.7

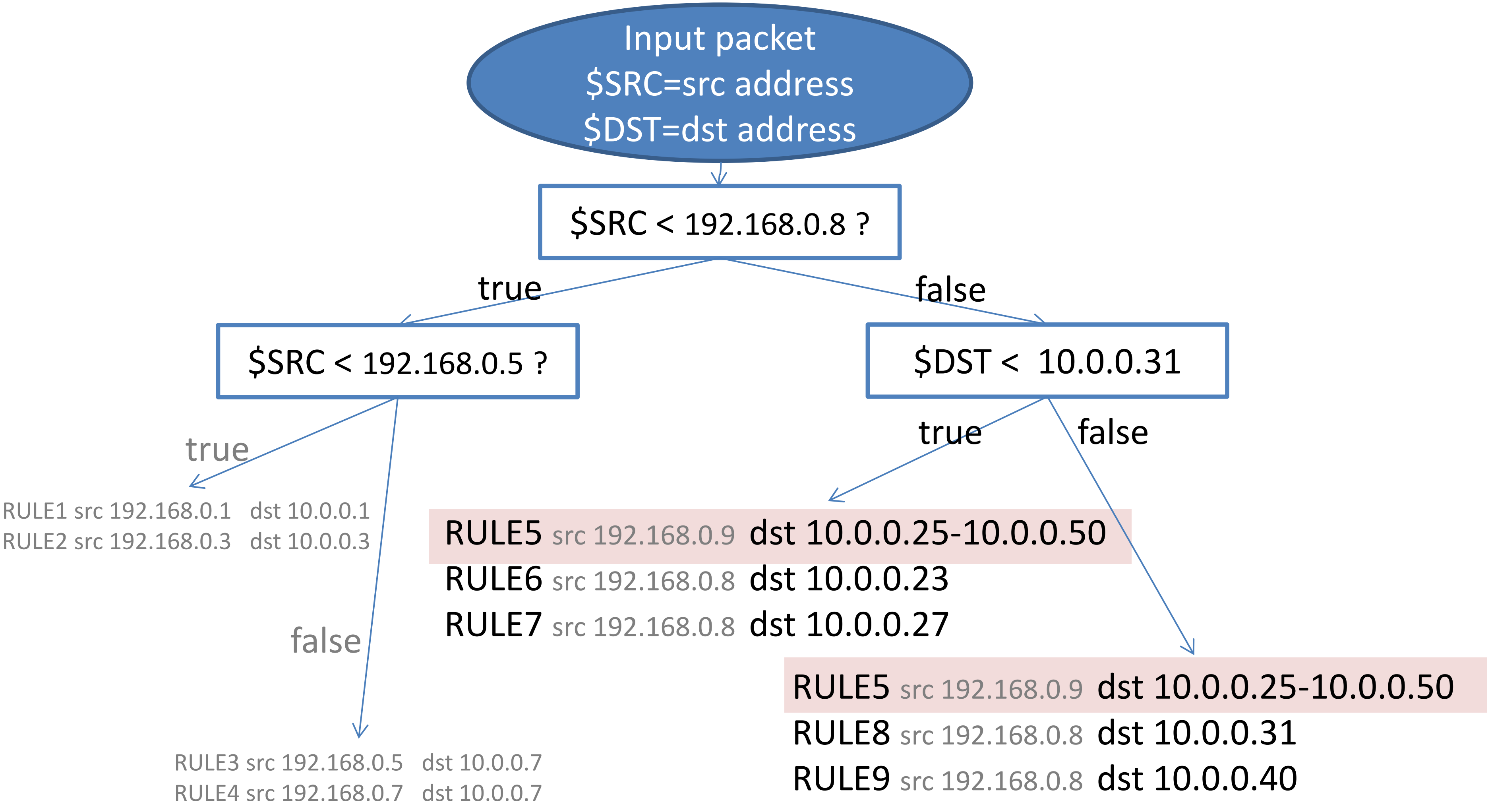
…Next pickup 192.168.0.5

false

RULE5	src 192.168.0.9	dst 10.0.0.25-10.0.0.50
RULE6	src 192.168.0.8	dst 10.0.0.23
RULE7	src 192.168.0.8	dst 10.0.0.27
RULE8	src 192.168.0.8	dst 10.0.0.31
RULE9	src 192.168.0.8	dst 10.0.0.40

…Next?

src and dst with address range



Input packet
$SRC=src address
$DST=dst address

$SRC < 192.168.0.8 ?

true

$SRC < 192.168.0.5 ?

true

RULE1 src 192.168.0.1 dst 10.0.0.1
RULE2 src 192.168.0.3 dst 10.0.0.3

false

RULE3 src 192.168.0.5 dst 10.0.0.7
RULE4 src 192.168.0.7 dst 10.0.0.7

false

$DST < 10.0.0.31

true

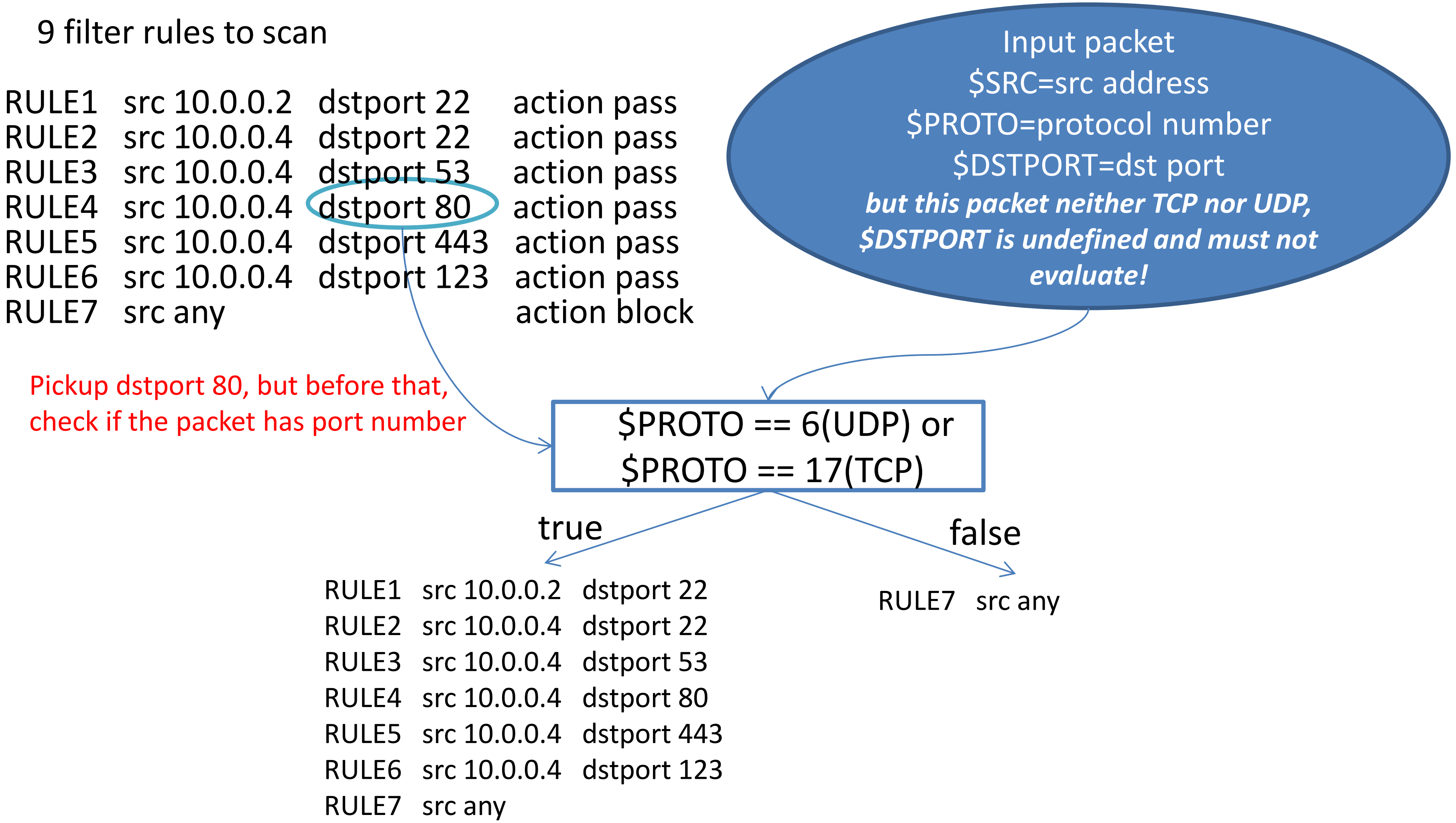
RULE5 src 192.168.0.9 dst 10.0.0.25-10.0.0.50
RULE6 src 192.168.0.8 dst 10.0.0.23
RULE7 src 192.168.0.8 dst 10.0.0.27

false

RULE5 src 192.168.0.9 dst 10.0.0.25-10.0.0.50
RULE8 src 192.168.0.8 dst 10.0.0.31
RULE9 src 192.168.0.8 dst 10.0.0.40

RULE5 belongs to both,
because RULE5 matches
whether true or false.

with port number



9 filter rules to scan

RULE1	src 10.0.0.2	dstport 22	action pass
RULE2	src 10.0.0.4	dstport 22	action pass
RULE3	src 10.0.0.4	dstport 53	action pass
RULE4	src 10.0.0.4	dstport 80	action pass
RULE5	src 10.0.0.4	dstport 443	action pass
RULE6	src 10.0.0.4	dstport 123	action pass
RULE7	src any		action block

Input packet
$SRC=src address
$PROTO=protocol number
$DSTPORT=dst port
but this packet neither TCP nor UDP, $DSTPORT is undefined and must not evaluate!

Pickup dstport 80, but before that,
check if the packet has port number

$PROTO == 6(UDP) or
$PROTO == 17(TCP)

true

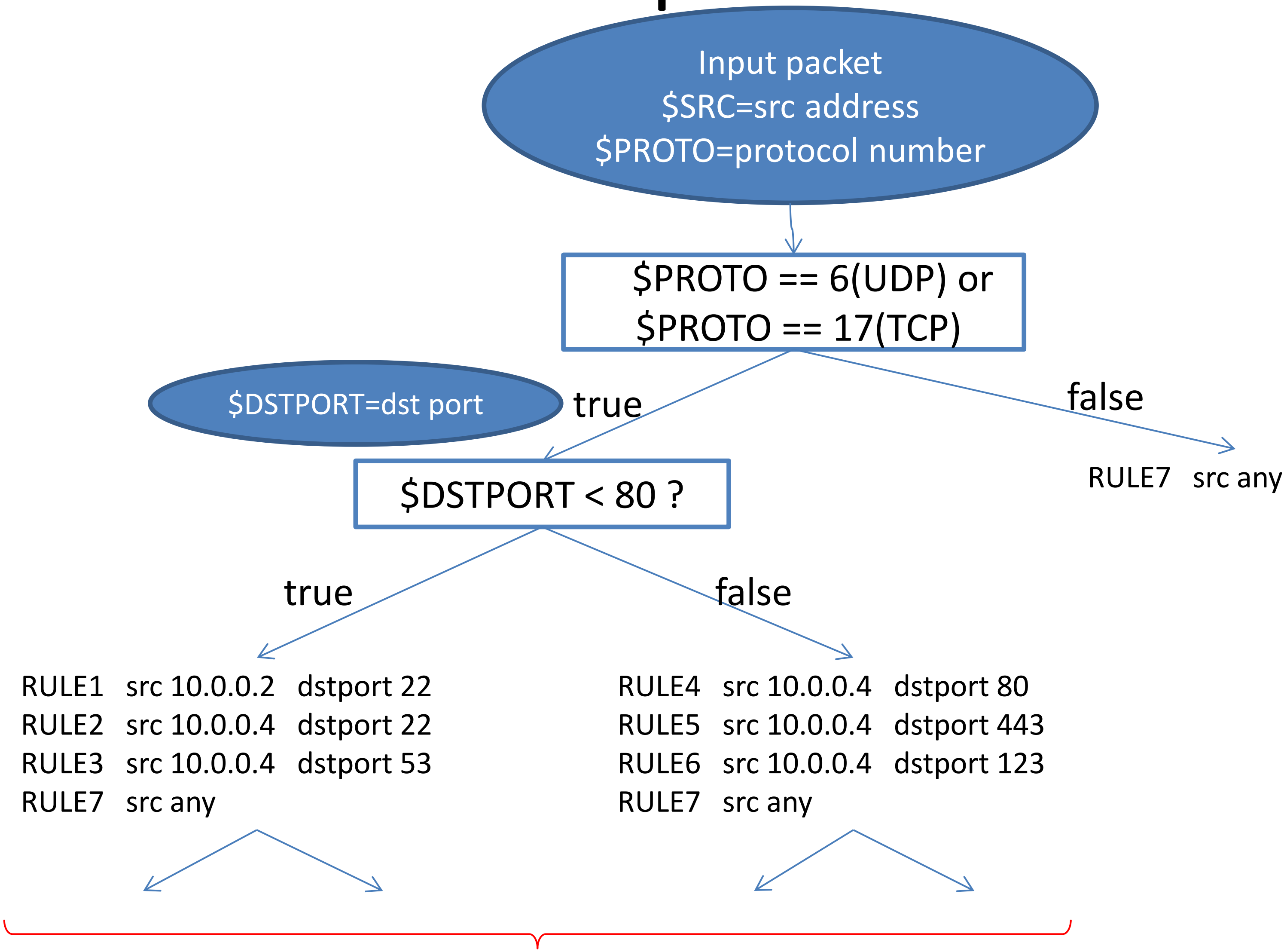
RULE1	src 10.0.0.2	dstport 22
RULE2	src 10.0.0.4	dstport 22
RULE3	src 10.0.0.4	dstport 53
RULE4	src 10.0.0.4	dstport 80
RULE5	src 10.0.0.4	dstport 443
RULE6	src 10.0.0.4	dstport 123
RULE7	src any	

false

RULE7 src any

In this case, the packet has port number.
It is able to compare $DSTPORT.

with port number



Input packet
$SRC=src address
$PROTO=protocol number

$PROTO == 6(UDP) or
$PROTO == 17(TCP)

$DSTPORT=dst port

true

false

RULE7 src any

$DSTPORT < 80 ?

true

false

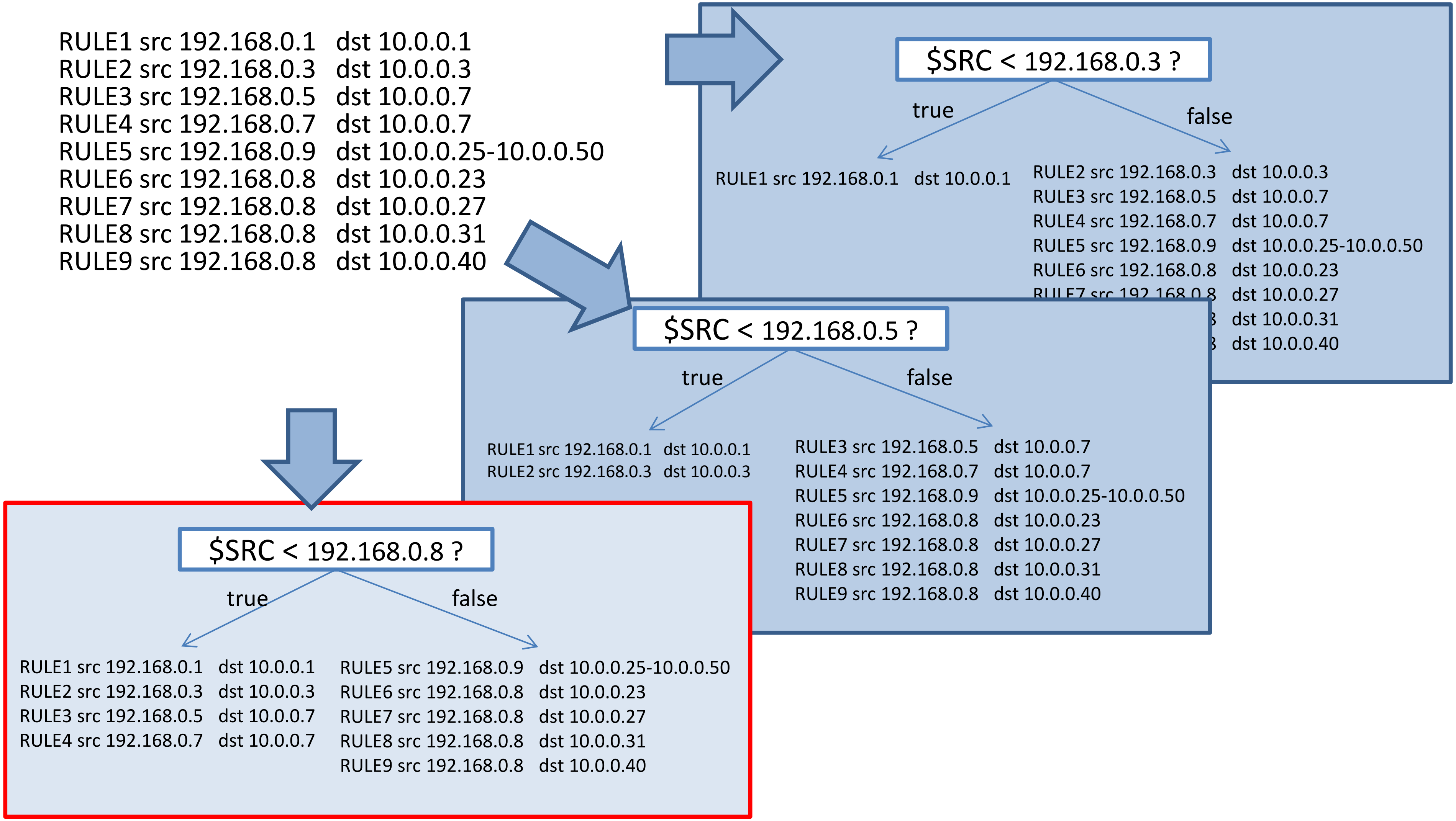
RULE1 src 10.0.0.2 dstport 22
RULE2 src 10.0.0.4 dstport 22
RULE3 src 10.0.0.4 dstport 53
RULE7 src any

RULE4 src 10.0.0.4 dstport 80
RULE5 src 10.0.0.4 dstport 443
RULE6 src 10.0.0.4 dstport 123
RULE7 src any

In this case, the packet has port number.
It is able to compare $DSTPORT

How to select conditional value

RULE1 src 192.168.0.1 dst 10.0.0.1
RULE2 src 192.168.0.3 dst 10.0.0.3
RULE3 src 192.168.0.5 dst 10.0.0.7
RULE4 src 192.168.0.7 dst 10.0.0.7
RULE5 src 192.168.0.9 dst 10.0.0.25-10.0.0.50
RULE6 src 192.168.0.8 dst 10.0.0.23
RULE7 src 192.168.0.8 dst 10.0.0.27
RULE8 src 192.168.0.8 dst 10.0.0.31
RULE9 src 192.168.0.8 dst 10.0.0.40



$SRC < 192.168.0.3 ?

true

RULE1 src 192.168.0.1 dst 10.0.0.1

false

RULE2 src 192.168.0.3 dst 10.0.0.3
RULE3 src 192.168.0.5 dst 10.0.0.7
RULE4 src 192.168.0.7 dst 10.0.0.7
RULE5 src 192.168.0.9 dst 10.0.0.25-10.0.0.50
RULE6 src 192.168.0.8 dst 10.0.0.23
RULE7 src 192.168.0.8 dst 10.0.0.27
3 dst 10.0.0.31
3 dst 10.0.0.40

$SRC < 192.168.0.5 ?

true

RULE1 src 192.168.0.1 dst 10.0.0.1
RULE2 src 192.168.0.3 dst 10.0.0.3

false

RULE3 src 192.168.0.5 dst 10.0.0.7
RULE4 src 192.168.0.7 dst 10.0.0.7
RULE5 src 192.168.0.9 dst 10.0.0.25-10.0.0.50
RULE6 src 192.168.0.8 dst 10.0.0.23
RULE7 src 192.168.0.8 dst 10.0.0.27
RULE8 src 192.168.0.8 dst 10.0.0.31
RULE9 src 192.168.0.8 dst 10.0.0.40

$SRC < 192.168.0.8 ?

true

RULE1 src 192.168.0.1 dst 10.0.0.1
RULE2 src 192.168.0.3 dst 10.0.0.3
RULE3 src 192.168.0.5 dst 10.0.0.7
RULE4 src 192.168.0.7 dst 10.0.0.7

false

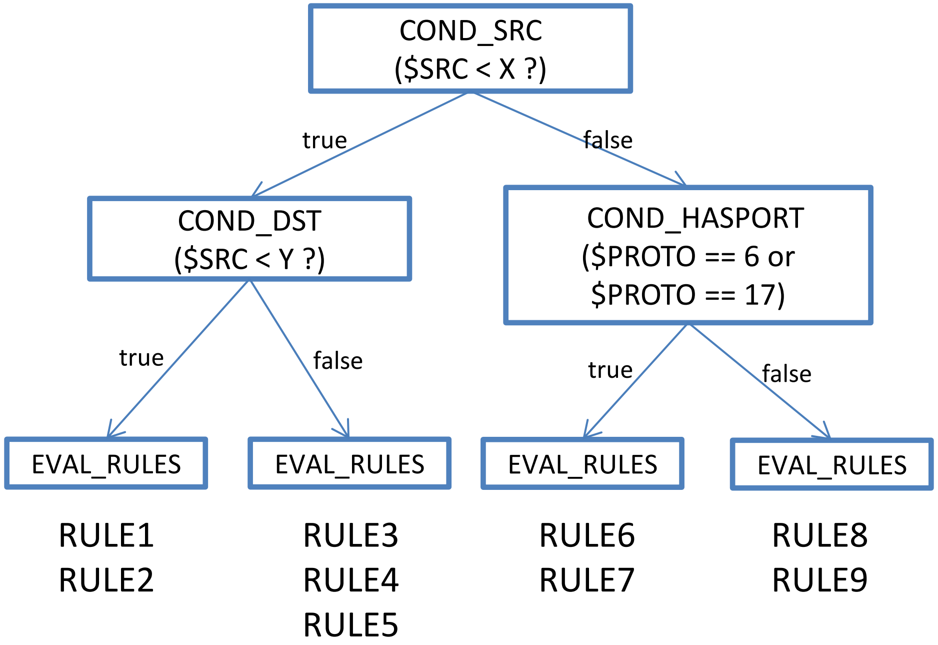
RULE5 src 192.168.0.9 dst 10.0.0.25-10.0.0.50
RULE6 src 192.168.0.8 dst 10.0.0.23
RULE7 src 192.168.0.8 dst 10.0.0.27
RULE8 src 192.168.0.8 dst 10.0.0.31
RULE9 src 192.168.0.8 dst 10.0.0.40

best balanced! Use this!

INTERNAL CODE

Type of node

COND_SRC
COND_DST
COND_PROTO
COND_SRCPORT
COND_DSTPORT
COND_IFNAME
COND_SRC6
COND_DST6
COND_HASPORT
EVAL_RULES

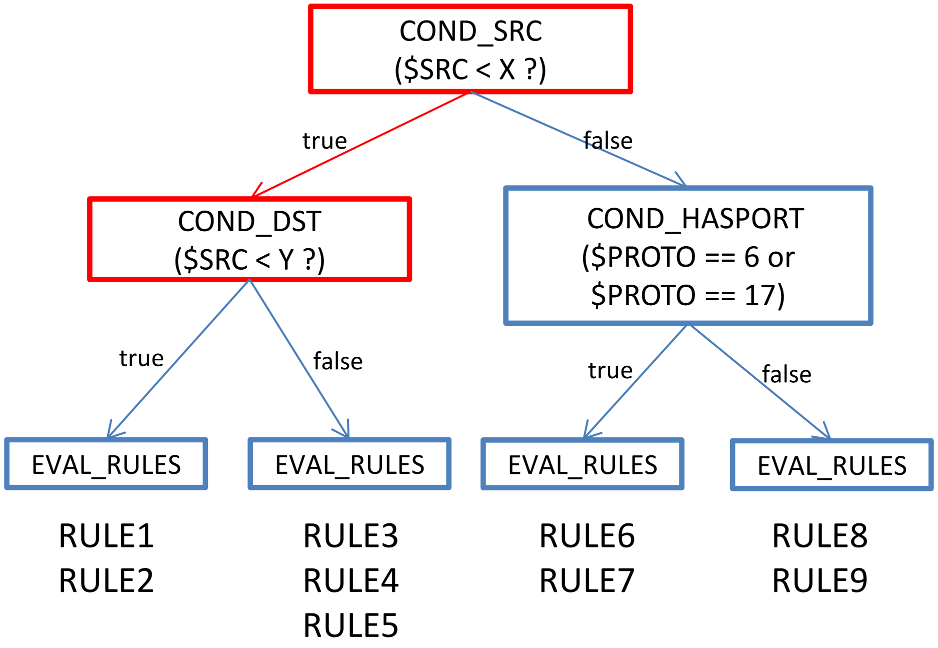


0x0000	COND_SRC	ADDRESS (X)
	TRUE (0x0010)	FALSE (0x0020)
0x0010	COND_DST	ADDRESS (Y)
	TRUE (0x0030)	FALSE (0x0040)
0x0020	COND_HASPORT	
	TRUE (0x0058)	FALSE (0x0068)
0x0030	EVAL_RULES	number of rules(2)
	index of RULE1	index of RULE2
0x0040	EVAL_RULES	number of rules(3)
	index of RULE3	index of RULE4
	index of RULE5	
0x0058	EVAL_RULES	number of rules(2)
	index of RULE6	index of RULE7
0x0068	EVAL_RULES	number of rules(2)
	index of RULE8	index of RULE9

INTERNAL CODE

Type of node

COND_SRC
COND_DST
COND_PROTO
COND_SRCPORT
COND_DSTPORT
COND_IFNAME
COND_SRC6
COND_DST6
COND_HASPORT
EVAL_RULES

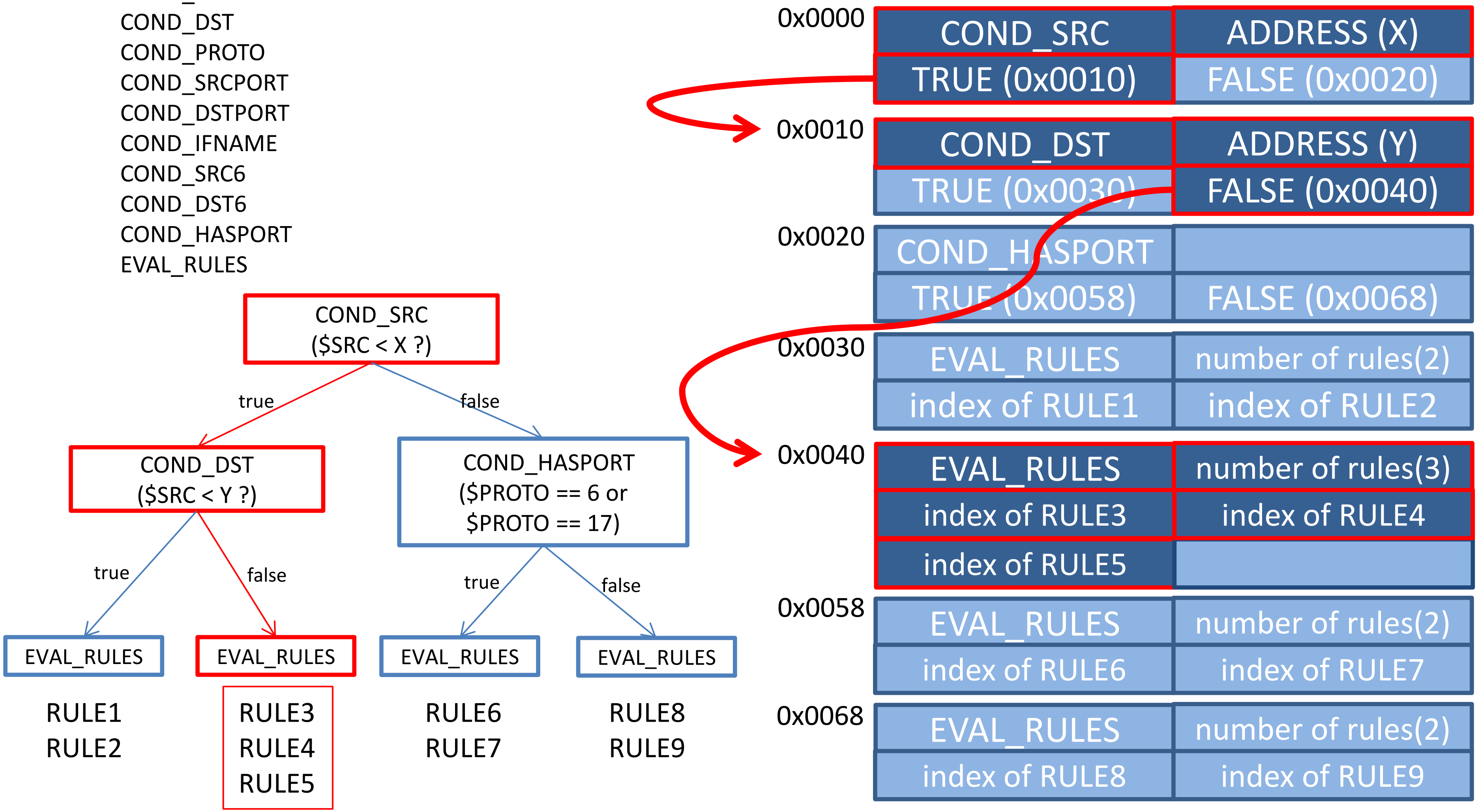


0x0000	COND_SRC	ADDRESS (X)
	TRUE (0x0010)	FALSE (0x0020)
0x0010	COND_DST	ADDRESS (Y)
	TRUE (0x0030)	FALSE (0x0040)
0x0020	COND_HASPORT	
	TRUE (0x0058)	FALSE (0x0068)
0x0030	EVAL_RULES	number of rules(2)
	index of RULE1	index of RULE2
0x0040	EVAL_RULES	number of rules(3)
	index of RULE3	index of RULE4
	index of RULE5	
0x0058	EVAL_RULES	number of rules(2)
	index of RULE6	index of RULE7
0x0068	EVAL_RULES	number of rules(2)
	index of RULE8	index of RULE9

INTERNAL CODE

Type of node

- COND_SRC
- COND_DST
- COND_PROTO
- COND_SRCPORT
- COND_DSTPORT
- COND_IFNAME
- COND_SRC6
- COND_DST6
- COND_HASPORT
- EVAL_RULES



COND_SRC ($SRC < X ?)

true

false

COND_DST ($SRC < Y ?)

COND_HASPORT ($PROTO == 6 or $PROTO == 17)

true

false

true

false

EVAL_RULES

EVAL_RULES

EVAL_RULES

EVAL_RULES

RULE1
RULE2

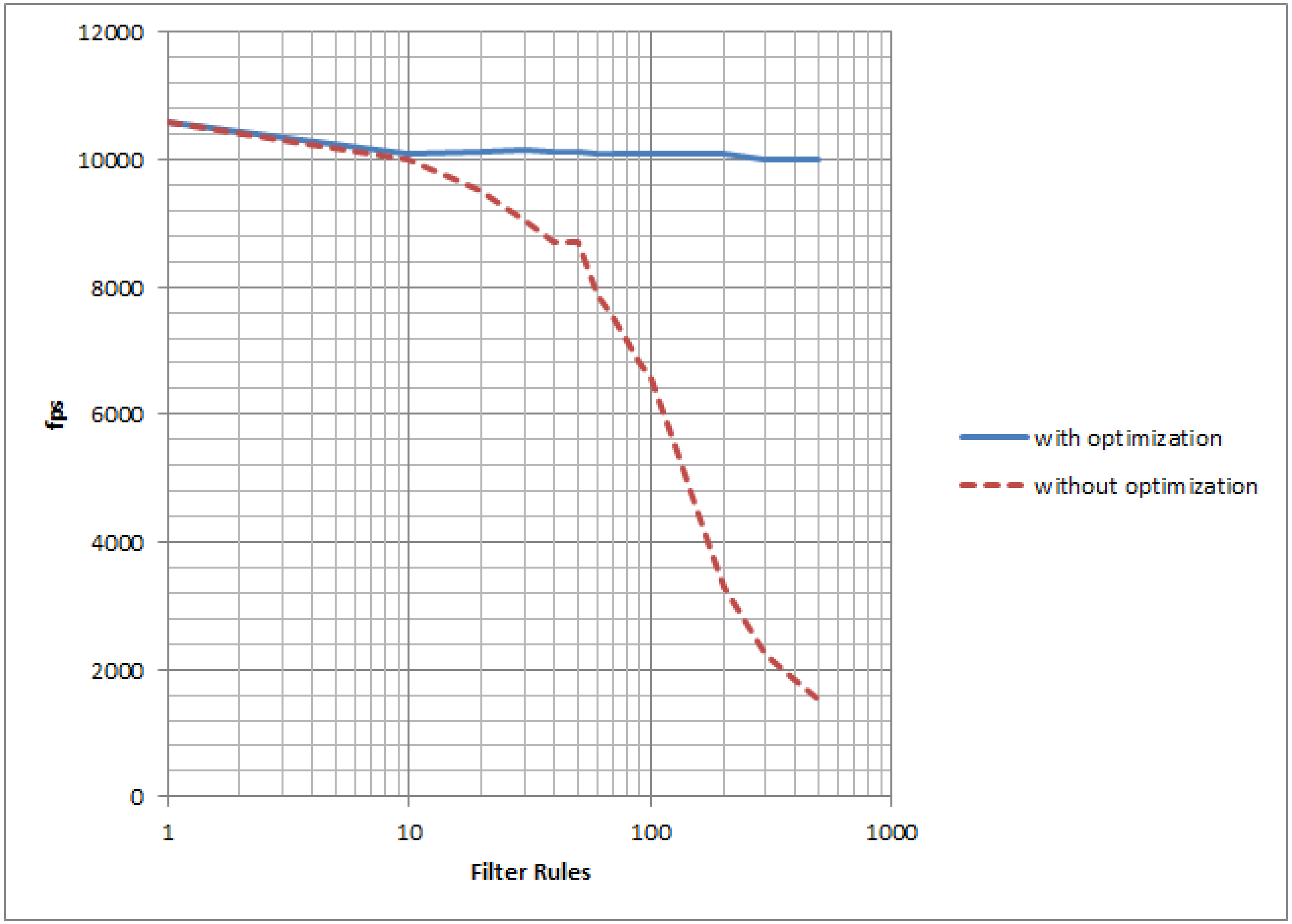
RULE3
RULE4
RULE5

RULE6
RULE7

RULE8
RULE9

0x0000	COND_SRC	ADDRESS (X)
	TRUE (0x0010)	FALSE (0x0020)
0x0010	COND_DST	ADDRESS (Y)
	TRUE (0x0030)	FALSE (0x0040)
0x0020	COND_HASPORT	
	TRUE (0x0058)	FALSE (0x0068)
0x0030	EVAL_RULES	number of rules(2)
	index of RULE1	index of RULE2
0x0040	EVAL_RULES	number of rules(3)
	index of RULE3	index of RULE4
	index of RULE5	
0x0058	EVAL_RULES	number of rules(2)
	index of RULE6	index of RULE7
0x0068	EVAL_RULES	number of rules(2)
	index of RULE8	index of RULE9

comparison graph for packet forwarding with/without optimization

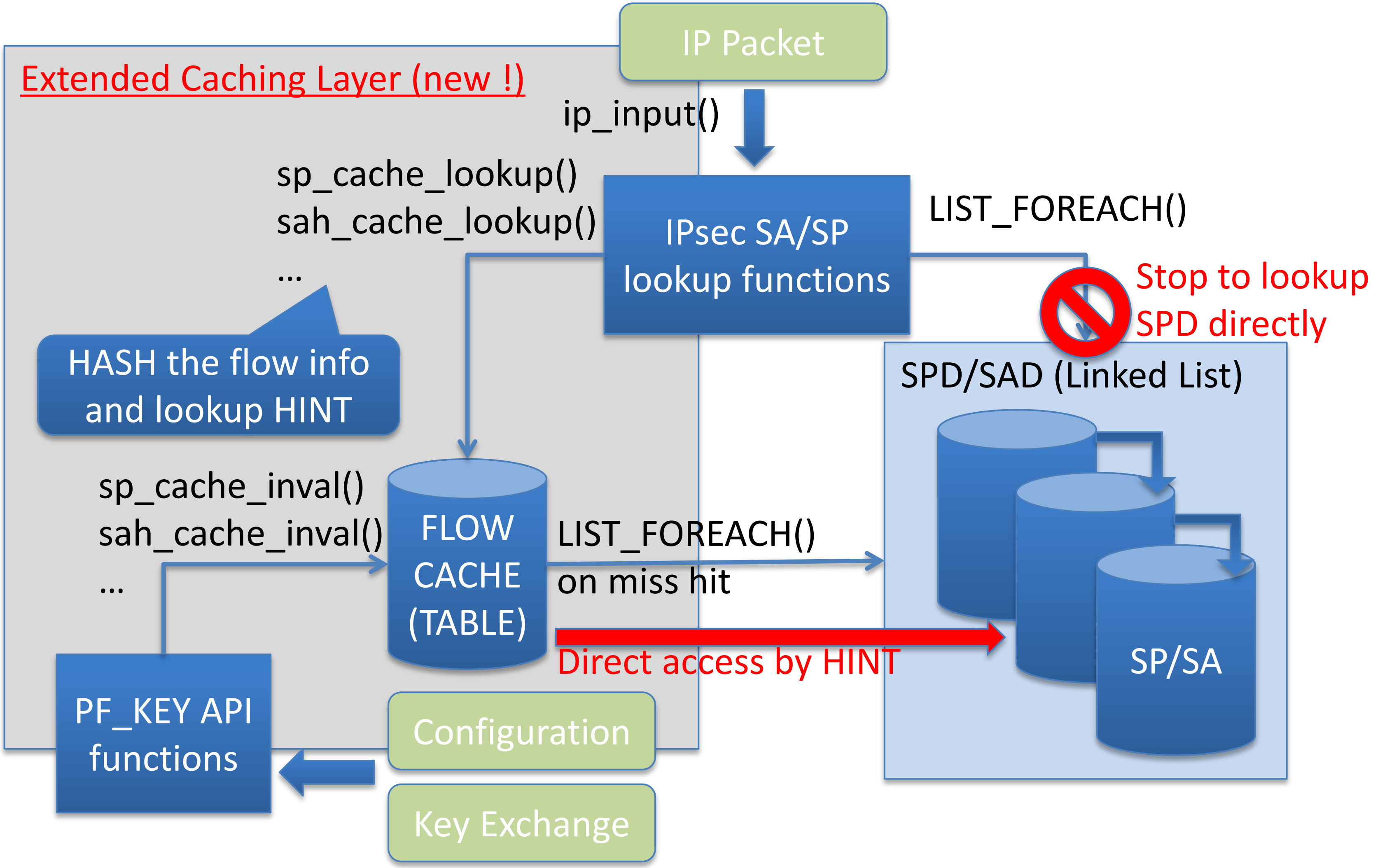


Summary

- To realize high-speed by optimizing filter rule scanning
- Add conditional branches to reduce testing rules
- Complex rules group under 3 patterns;
  - Likely match (true)
  - Never match (false)
  - In balance (both of true and false)
- Selecting conditional value by all exploration

2. SAD/SPD cache and IPsec Interface

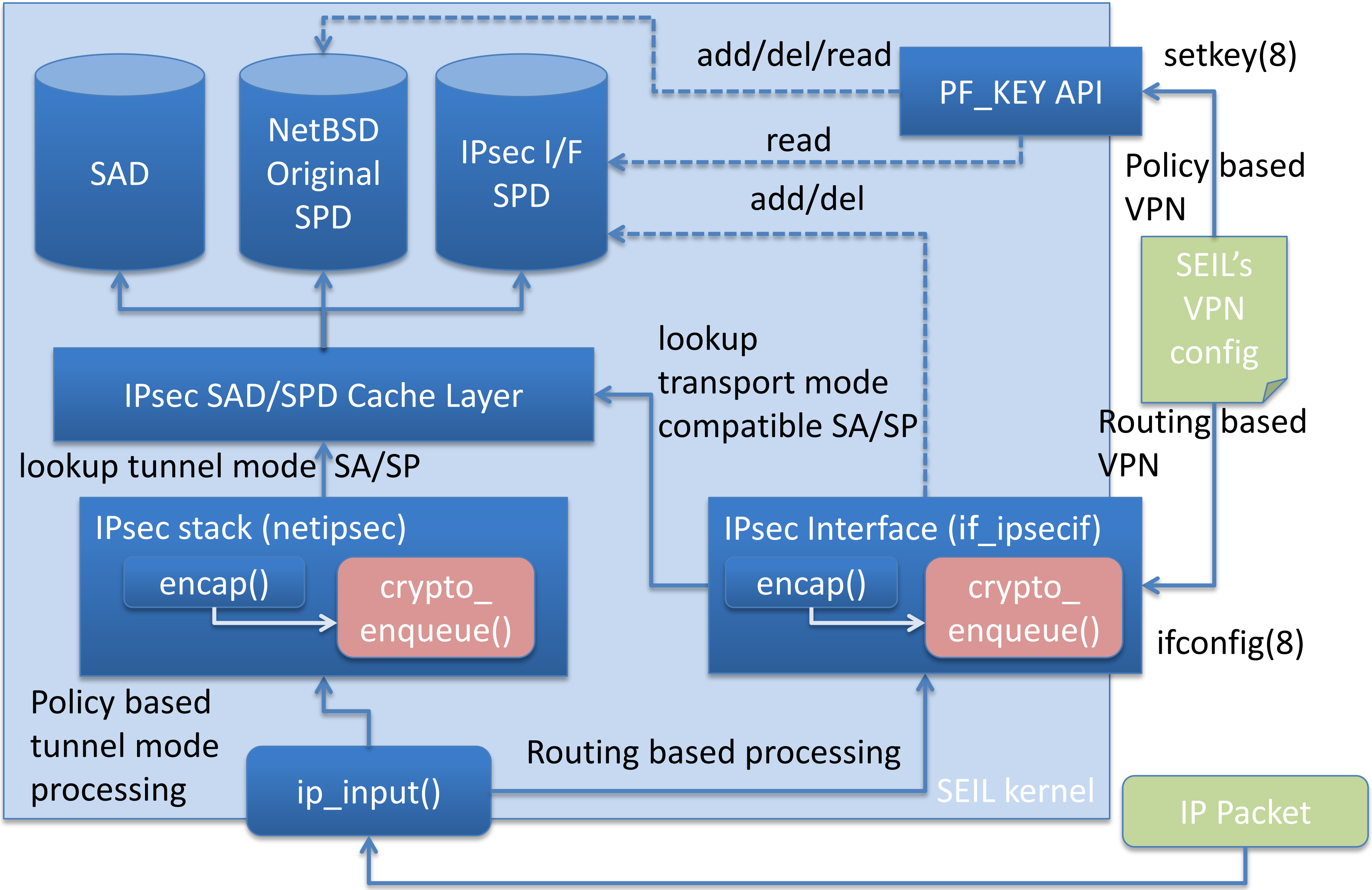
Add Caching layer to IPsec key management subsystem(PF_KEY)



IPsec flow cache

- Calculate simple hash value from:
  - source address
  - destination address
  - source port (for UDP/TCP)
  - destination port (for UDP/TCP)
- Store the hash value to open hash table
  - The table has 512 entry to a list of flow info
  - The list has 4 entry
  - we need to tune those values for each of products.
- There are 2 hash tables, positive caching and negative caching

IPsec tunneling device(if_ipsecif)



Route packet into IPsec tunnel

Policy based processing (Managed by static configuration like IP filter)

Source	Destination	Protocol	Port	Action
192.168.1.0 /24	10.1.0.0/24	TCP	80	IPsec
192.168.1.0 /24	10.2.0.0/24	TCP	80	IPsec
…				

Routing based processing (Managed by static configure or OSPF, RIP, etc..)

Destination	Gateway Interface
10.1.0.0/24	ipsec0
10.2.0.0/24	ipsec1
…	

Why routing?

- Existing redundancy techniques using widely deployed routing protocols
- Seamless integration with existing routings.
- To gather filtering rules in IP filter sub system.
- There is few requirements for complicated policy using source, protocol, and so on, especially in site-to-site VPN connection.

Configuring IPsec tunneling device

1. A user configures ipsec tunneling device like gif interface.
2. Then our kernel automatically create SPD for the tunneling device. The SPD is fully compatible with existing IPsec stack(netipsec) and IKE servers and is separated from NetBSD's original SPD.
3. The IKE server generates IPsec-SAs for the tunneling device. Our IKE server has an option of Phase2 ID selection for interoperability (tunnel endpoint address or network 0.0.0.0/0)

```
# ifconfig ipsec0 tunnel 203.0.113.1 203.0.113.2
# ifconfig ipsec0 inet 192.0.2.1
# ifconfig ipsec0
ipsec0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST>
        tunnel inet 203.0.113.1 --> 203.0.113.2
        inet 192.0.2.1 ->  netmask 0xffffff00
        inet6 fe80::2e0:4dff:fe30:28%ipsec0
          ->  prefixlen 64 scopeid 0xf
# setkey -DP
203.0.113.2[any] 203.0.113.1[any] 41(ipv6)
        in discard
        spid=36 seq=3 pid=1807
        refcnt=1
…..
```

Considerations

- There are multiple packet classifiers in kernel….
  - IP filter
    - rich rule
    - fast caching
    - state control
    - optimized internal representations (iipf, npf)
  - ALTQ
    - fast classify
    - separated point of probe/enforce
  - IPsec
    - support cryptographic parameter
  - BPF
    - highly programmable VM
  - vSwitch?
  - Multi queue capable NIC?

Ethernet switch framework

Why Ethernet switch?(1)

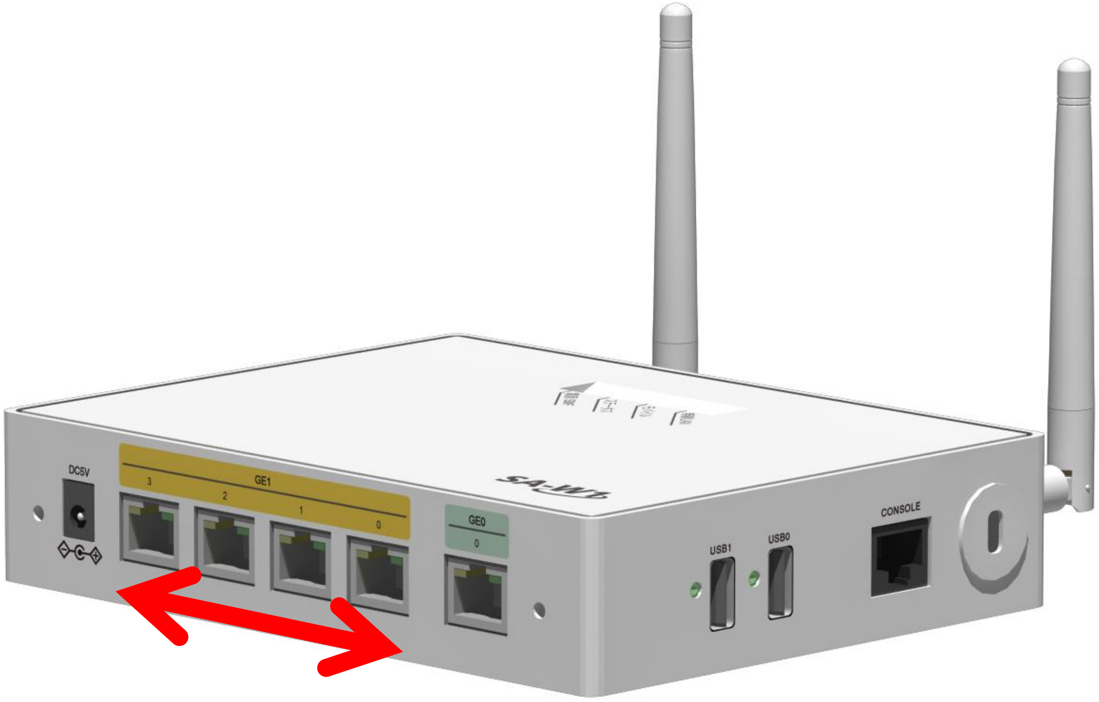
- SOHO router
- Home Gateway
- (big L2 switch)

Why Ethernet switch?(2)

- For business use
  - VLAN
  - Port mirroring
  - Check the forwarding database
  - Check port status
  - Control each port’s media setting.

Ethernet switch framework

- Designed and implemeted by Hikaru Abe.
- To support SA-W1’s Ethernet switch port.
  - Marvell 88E6171R



Design Concept

- Separate functions into:
  - – Ethernet switch common function part
  - – Hardware specific part
- comparison

	Common function part	Hardware specific part
Ethernet interface	if_ethersubr.c	if_bge.c
Ethernet switch	if_etherswsubr.c	mvls.c

Design Concept (2)

- Control/check Ethernet switch function using with swconfig(8)
- Control/check media setting using with ifconfig(8)

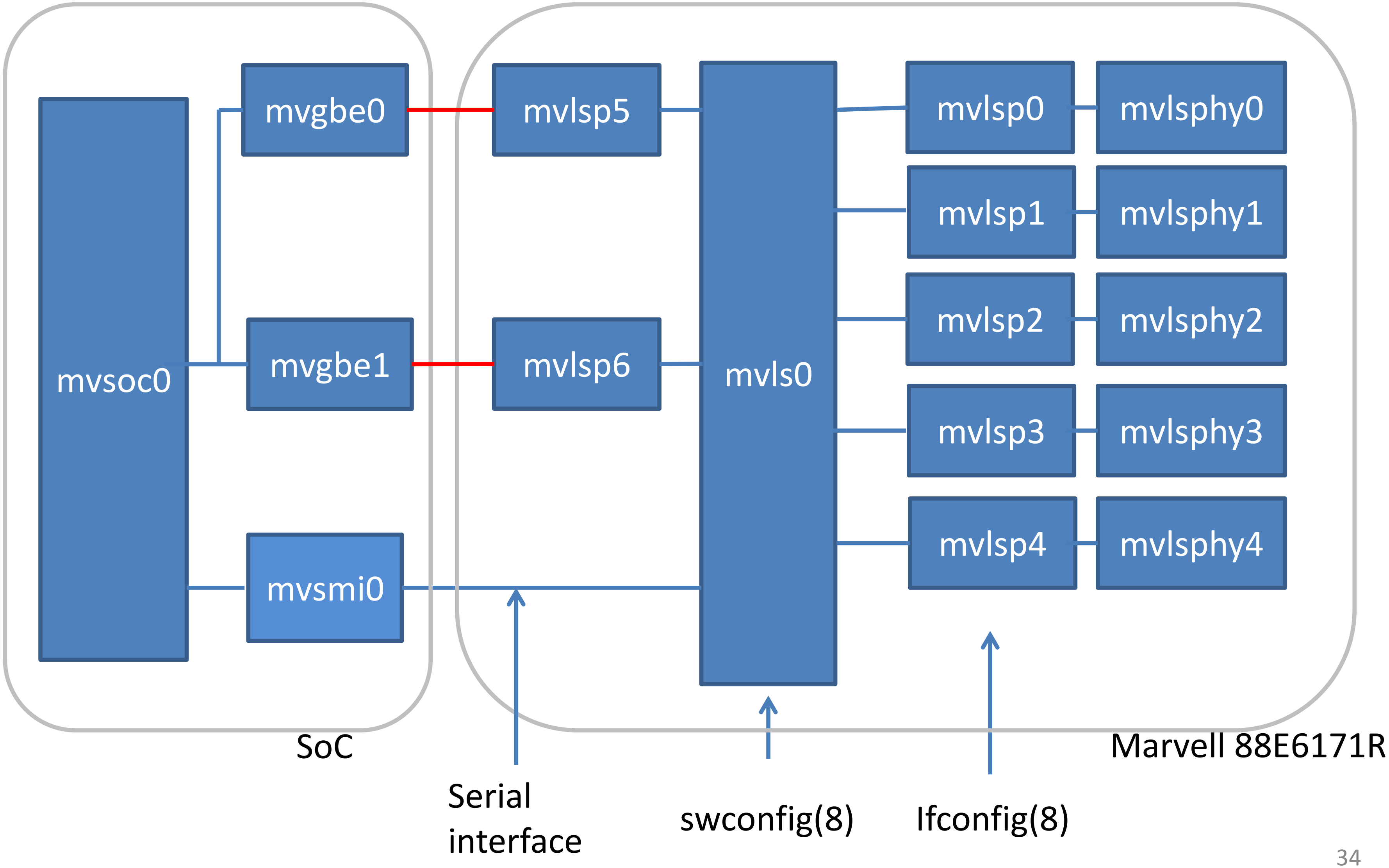
- VLAN
- Port mirroring
- Check the forwarding database

swconfig(8)

- Check port status, counters
- Control each port’s media setting

ifconfig(8)

Block diagram



Implementation of mvls(4) and mvlsp(4)

- Use ifnet structure for both drivers.
- mvlsp(4) connects each phy using with mii(4)

ifconfig, netstat, snmp can be used
without any modification

Ethernet switch drivers on SA-W1(dmesg)

```
mvsmi0 at mvsoc0 unit 0 offset 0x72004-0x72007: Serial Management Interface
mvls0 at mvsmi0 addr 0-31 gpio 11 irq 107 single-chip rev 2: Marvell Gigabit Ethernet Switch
mvlsp0 at mvls0 port 0: Marvell Gigabit Ethernet Switch External Port
mvlsphy0 at mvlsp0 phy 0: Marvell 88E6171 Gigabit Switch PHY, rev. 0
mvlsphy0: 10baseT, 10baseT-FDX, 100baseTX, 100baseTX-FDX, 1000baseT-FDX, auto
                                   (snip)
mvlsp4 at mvls0 port 4: Marvell Gigabit Ethernet Switch External Port
mvlsphy4 at mvlsp4 phy 4: Marvell 88E6171 Gigabit Switch PHY, rev. 0
mvlsphy4: 10baseT, 10baseT-FDX, 100baseTX, 100baseTX-FDX, 1000baseT-FDX, auto
mvlsp5 at mvls0 port 5: Marvell Gigabit Ethernet Switch Internal Port
mvlsp6 at mvls0 port 6: Marvell Gigabit Ethernet Switch Internal Port
mvgbec0 at mvsoc0 unit 0 offset 0x70000-0x73fff: Marvell Gigabit Ethernet Controller
mvgbe0 at mvgbec0 port 0 irq 11
mvgbe0: Ethernet address 00:e0:4d:30:00:38
mvgbe0: connected to mvlsp5 with rgmii
mvgbec1 at mvsoc0 unit 1 offset 0x74000-0x77fff: Marvell Gigabit Ethernet Controller
mvgbe1 at mvgbec1 port 0 irq 15
mvgbe1: Ethernet address 00:e0:4d:30:00:39
mvgbe1: connected to mvlsp6 with rgmii
```

Implementation of Ethernet Switch common func.

- Use ifnet structure (as descrived before)
- Add new ioctls.

```
/* ioctl commands */
#define ETHSWPGRADD        0    /* add port group (ifeswreq) */
#define ETHSWPGRDEL        1    /* delete port group (ifeswreq) */
#define ETHSWSPGRMEM       2    /* set port group member (ifeswmereq) */
#define ETHSWPFDBADD       3    /* add port fdb (ifeswreq) */
#define ETHSWPFDBDEL       4    /* delete port fdb (ifeswreq) */
#define ETHSWSPFDBMEM      5    /* set port fdb member (ifeswmereq) */
#define ETHSWVLADD         6    /* add vlan entry (ifeswreq) */
#define ETHSWVLDEL         7    /* delete vlan (ifeswreq) */
#define ETHSWSVLMEM        8    /* set vlan member (ifeswmereq) */
#define ETHSWSPDFLTVL      9    /* set port default vlan (ifeswreq) */
#define ETHSWIMISET       10    /* start ingress mirroring (ifeswmireq) */
#define ETHSWIMIUNSET     11    /* stop ingress mirroring (ifeswmireq) */
#define ETHSWOMISET       12    /* start egress mirroring (ifeswmireq) */
#define ETHSWOMIUNSET     13    /* stop egress mirroring (ifeswmireq) */
#define ETHSWGPFLAGS      14    /* get port flags (ifeswreq) */
#define ETHSWSPFLAGS      15    /* set port flags (ifeswreq) */
#define ETHSWFLSHFDB      16    /* flush address table (ifeswreq) */
#define ETHSWGFDB         17    /* get address table (XXX) */
```

Usage of swconfig(8)

```
# swconfig
usage:  swconfig <dev> group <groupid> [member '<port>...']
        swconfig <dev> -group <groupid>
```

Broadcast domain

```
        swconfig <dev> portfdb <fdbid> [member '<port>...']
        swconfig <dev> -portfdb <fdbid>
```

Forwarding DB

```
        swconfig <dev> vlan <vlanid> [member '<port>[<(u)ntag,(t)ag>]...']
        swconfig <dev> -vlan <vlanid>
        swconfig <dev> defaultvlan <port> <vlanid>
```

vlan

```
        swconfig <dev> mirror-rx <dstport> '<srcport>...'
        swconfig <dev> -mirror-rx
        swconfig <dev> mirror-tx <dstport> '<srcport>...'
        swconfig <dev> -mirror-tx
```

mirroring

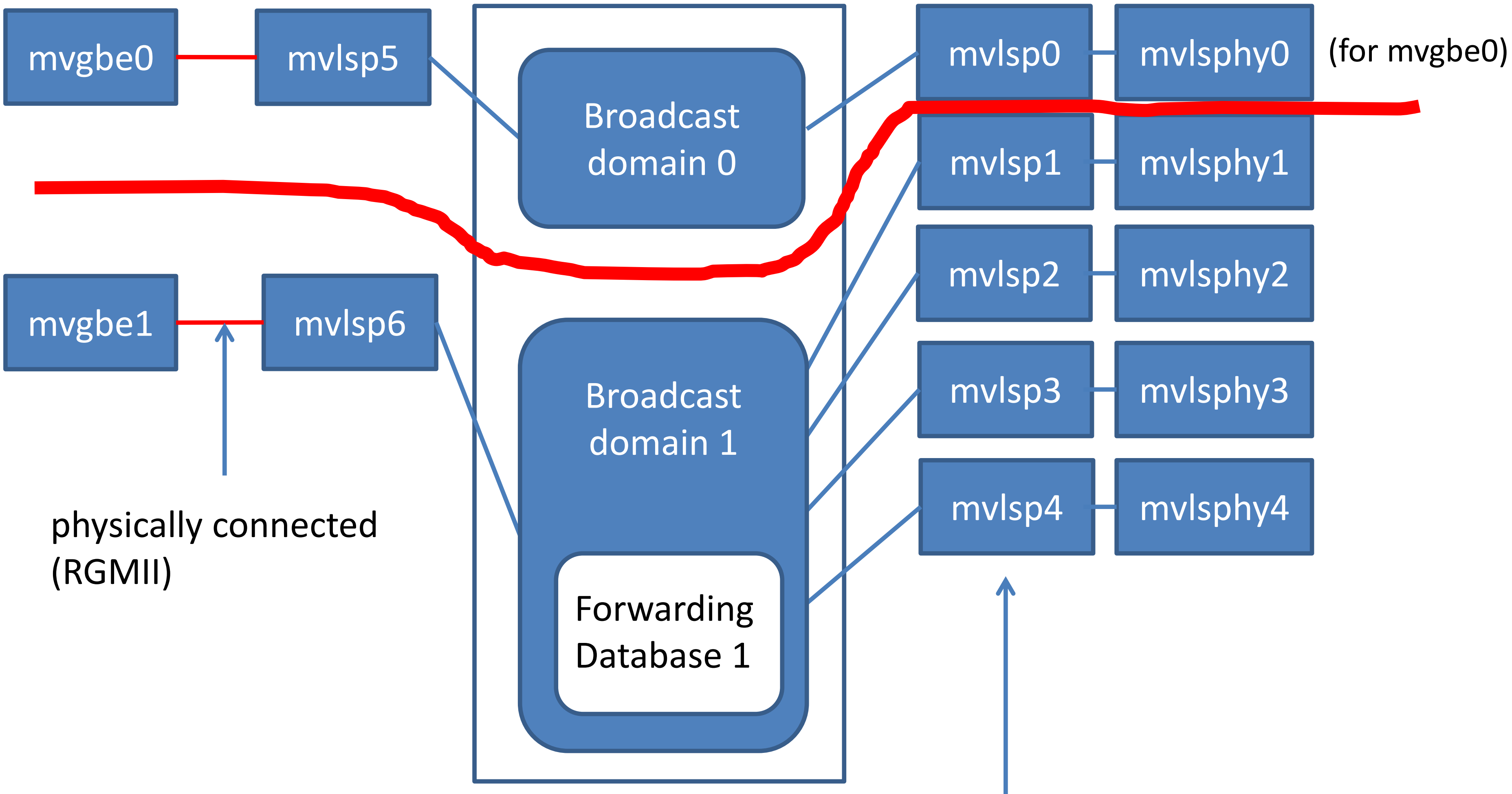
```
        swconfig <dev> nolearning|-nolearning <port>
        swconfig <dev> notagged|-notagged <port>
        swconfig <dev> nountagged|-nountagged <port>
```

control

```
        swconfig <dev> flushfdb <fdbid>
        swconfig <dev> showfdb <fdbid>
```

admin

Relation between mvgbe and switch



(for mvgbe0)

physically connected (RGMII)

swconfig(8)

Ifconfig(8)

Settings

```
# cat /etc/ifconfig.mvls0
!swconfig $int group 0 member '0 5' nolearning 0 nolearning 5
!swconfig $int group 1 member '1 2 3 4 6'
!swconfig $int portfdb 1 member '1 2 3 4 6’
up
```

Ifconfig -a

```
$ ifconfig -a
mvls0: flags=41<UP,RUNNING> mtu 1500
mvlsp0: flags=41<UP,RUNNING> mtu 1500
        media: Ethernet autoselect (1000baseT
full-duplex)
        status: active
(snip)
mvlsp4: flags=41<UP,RUNNING> mtu 1500
        media: Ethernet autoselect (none)
        status: no carrier
mvlsp5: flags=41<UP,RUNNING> mtu 1500
        media: Ethernet manual (none)
mvlsp6: flags=41<UP,RUNNING> mtu 1500
        media: Ethernet manual (none)
```

```
mvgbe0:
flags=8843<UP,BROADCAST,RUNNING,SIMPLEX
,MULTICAST> mtu 1500

capabilities=3700<IP4CSUM_Rx,IP4CSUM_Tx,T
CP4CSUM_Rx,UDP4CSUM_Rx,UDP4CSUM_Tx>
        enabled=0
        ec_capabilities=1<VLAN_MTU>
        ec_enabled=0
        address: 00:e0:4d:ff:03:54
        media: Ethernet manual (none)
        status: active
        inet6 fe80::2e0:4dff:feff:354%mvgbe0
prefixlen 64 scopeid 0x9
```

Ifconfig -av

```
$ ifconfig -av
mvls0: flags=41<UP,RUNNING> mtu 1500
        input: 0 packets, 0 bytes
        output: 0 packets, 0 bytes
mvlsp0: flags=41<UP,RUNNING> mtu 1500
        media: Ethernet autoselect (1000baseT
full-duplex)
        status: active
        input: 1394324 packets, 13433429 bytes
        output: 373752 packets, 256165 bytes
(snip)
mvlsp4: flags=41<UP,RUNNING> mtu 1500
        media: Ethernet autoselect (none)
        status: no carrier
        input: 0 packets, 0 bytes
        output: 0 packets, 0 bytes
```

```
mvlsp5: flags=41<UP,RUNNING> mtu 1500
        media: Ethernet manual (none)
(snip)
mvlsp6: flags=41<UP,RUNNING> mtu 1500
        media: Ethernet manual (none)
(snip)
mvgbe0:
flags=8843<UP,BROADCAST,RUNNING,SIMPLEX
,MULTICAST> mtu 1500
capabilities=3700<IP4CSUM_Rx,IP4CSUM_Tx,T
CP4CSUM_Rx,UDP4CSUM_Rx,UDP4CSUM_Tx>
        enabled=0
        ec_capabilities=1<VLAN_MTU>
        ec_enabled=0
        address: 00:e0:4d:ff:03:54
        media: Ethernet manual (none)
        status: active
        inet6 fe80::2e0:4dff:feff:354%mvgbe0
prefixlen 64 scopeid 0x9
```

Considerations?(1)

- What is the best way to configure(8) Ethernet switch
  - Almost all ethernet drivers asuume that MII PHY is connected, so they call mii_attach().
  - Ethernet switch may be connected via
    - GMII or RGMII
    - I2C or MDIO.
  - It might be difficult to identify what device is connected to the MAC.

Considerations? (2)

- Relations between bridge(4) and l2sw(4)
  - Some functions are the same.
- Relations between l2sw(4)’s vlan function and vlan(4)’s vlan function.
  - Should be synchronized with each other?
- Spanning tree protocol
  - Not added into our implementation yet.

Work in progress

- “Improving bridge(4) or Toward a Unified L2 Framework”
  - Presented in NetBSD BoF in March 14th, 2014.
  - http://www.netbsd.org/~ozaki-r/pub/AsiaBSDCon2014-BoF-goda-ozaki.pdf

Conclusion

- We’d like to show other work, too.
  - in *BSD conference?

Thank you.