ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SDN Controller Design for Dynamic Chaining of Virtual Network Functions.

Franco Callegati, Walter Cerroni,
Chiara Contoli, Giuliano Santandrea
Dept. of Electrical, Electronic and Information Engineering
University of Bologna – Italy

EWSDN 2015 – September 30, 2015 – Bilbao, Spain

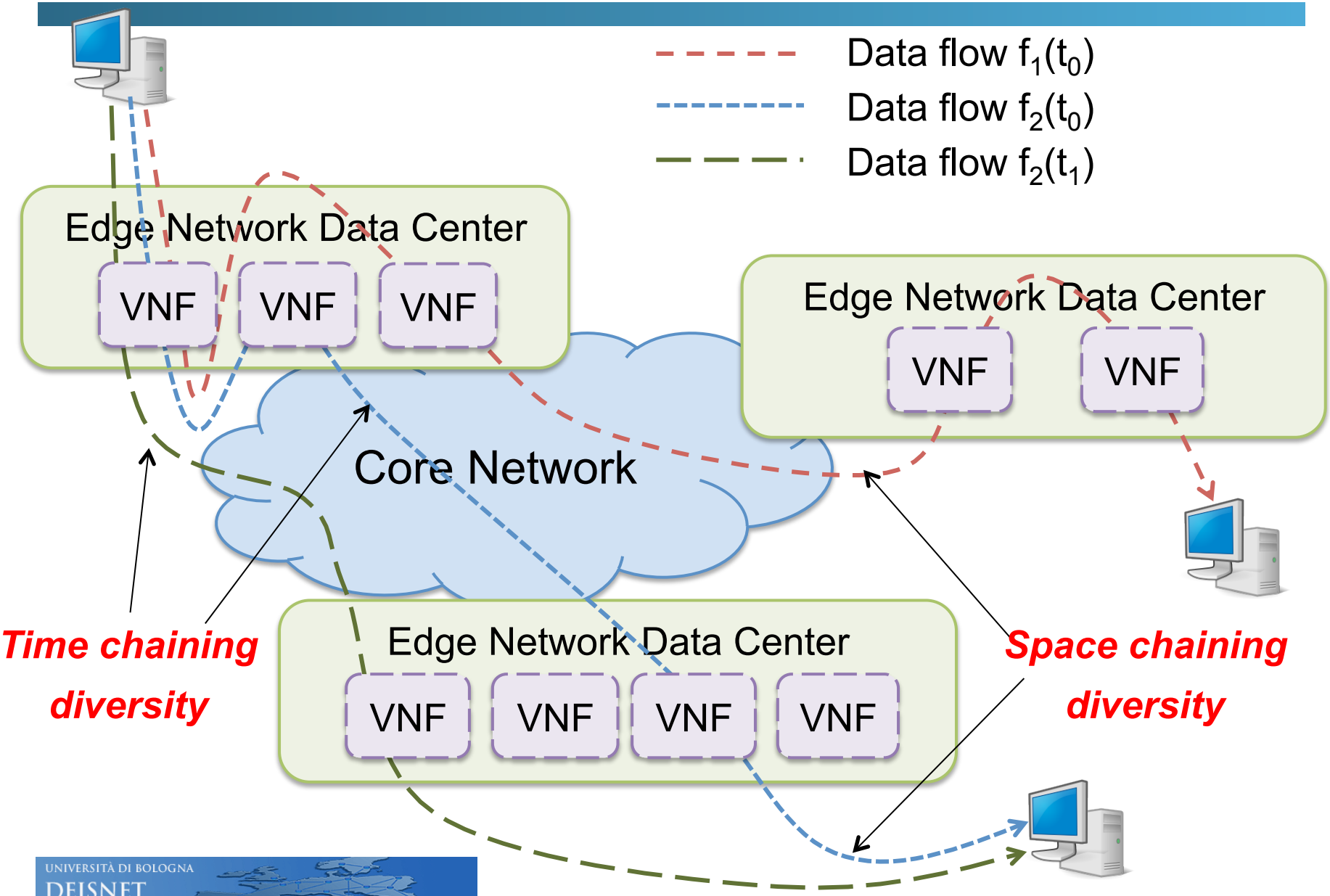
Context

- Network programmability is becoming feasible owing to recently developed key technologies
  - SDN
  - NFV
  - hardware capable of supporting them

- NFV as a flexible solution for replacing vendor dependent middle-boxes

- Paradigm shift will take place mainly at the **network edges**

NFV Chaining & dynamic traffic steering



Goal

- Suggest a design methodology for implementing SDN control plane capable of
  - Dynamically steering traffic towards required VNF
  - Achieving fully adaptive service chaining
- Case study:
  - Layer 2 (L2) Edge network
  - OpenStack cloud platform

- What about SDN controller design?
  - Mealy Machine abstraction as general approach to service chain reconfiguration
  - example: dynamic enforcement of QoS in a multi-tenant scenario

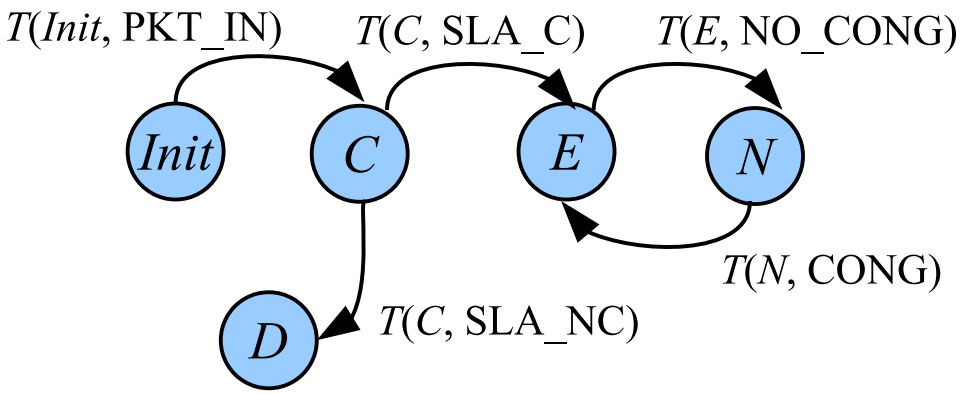
SDN controller abstraction

- Finite State Machine (FSM)
- Formal definitions:
  - *f*: traffic stream
  - *s*: set of state

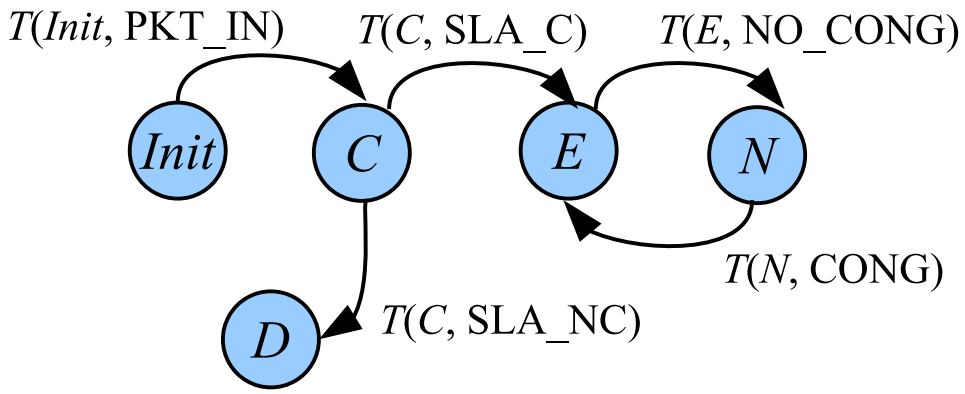
    $s \in \{Init, C, E, N, D\}$
  - *i*: set of input

    $i \in \{PKT\_IN, SLA\_C, SLA\_NC, CONG, NO\_CONG\}$
  - *A(f,s,i)*: actions (SDN technology dependent)
  - *T*: state transition function

    $(s,i) \rightarrow (s', A(f,s,i))$



States definition



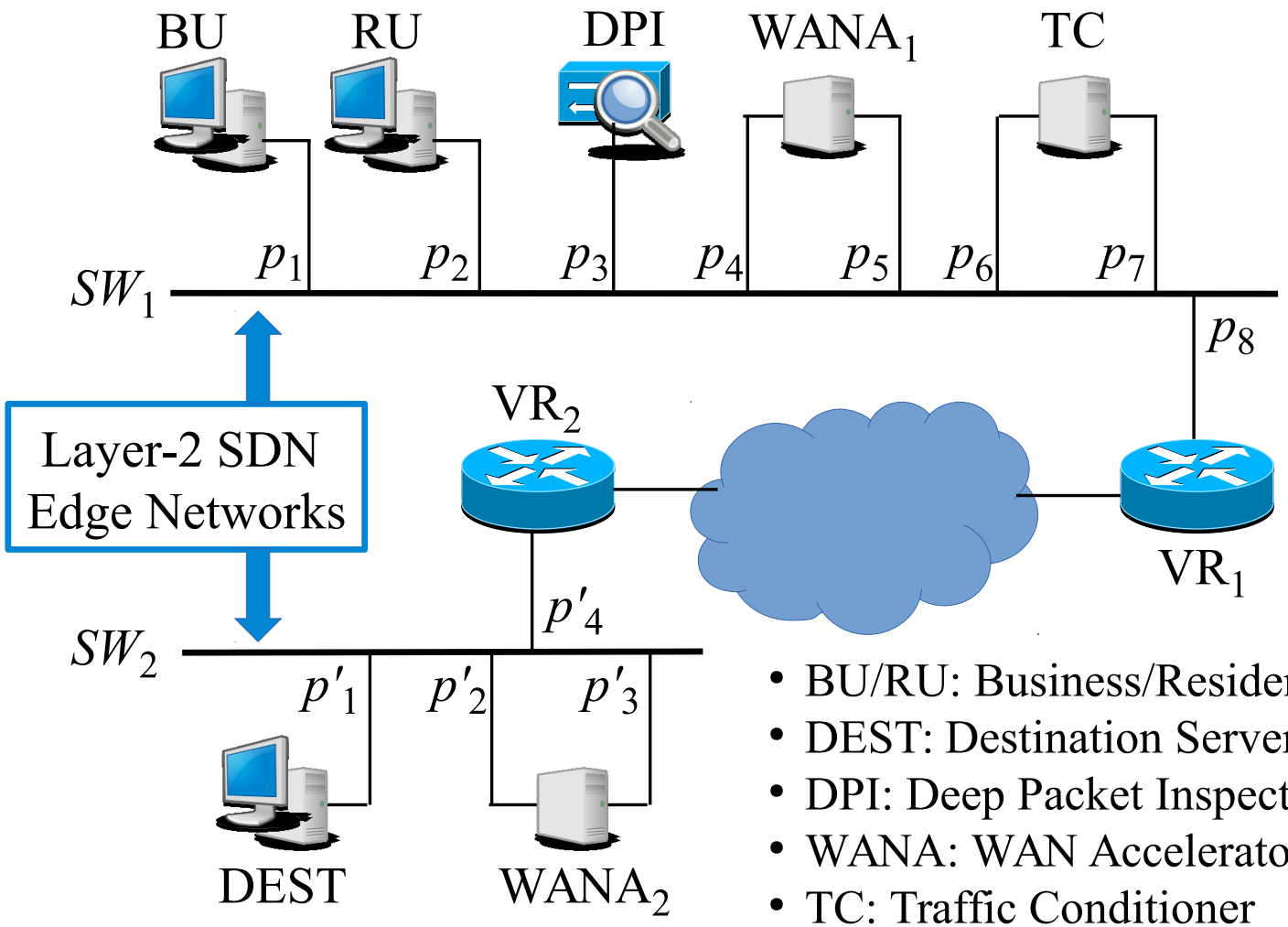
- State definition:
  - *Init*: flow independent rules are installed in the network nodes
  - *C*: flow *f* is analyzed and classified
  - *E*: flow *f* is strictly subject to QoS enforcement
  - *N*: flow *f* is not strictly subject to QoS enforcement
  - *D*: flow *f* is subject to policing actions

- Additional parameters can be defined

Genaral parameters

- $NT = \{SW_1, SW_2, \ldots, SW_{NSW}\}$ set of switch
- $SW_j = \{p_1, p_2, \ldots, p_{Np,j}\}$ set of ports
- $U = \{u_1, u_2, \ldots, u_{Nu}\}$ set of users
- $NF = \{F_1, F_2, \ldots, F_{NF}\}$ set of VNFs
- $Ch(f, s) = \{F_{l1}, F_{l2}, \ldots, F_{ln(f,s)}\}$ service chain of $n(f,s)$ VNFs applied to f in state s
- Getting topology information:
  - $(SW_j, p_m) = get\_port(u_k)$
  - $(SW_j, p_m) = get\_in\_port(F_l, d)$
  - $(SW_j, p_m) = get\_out\_port(F_l, d)$
- $SW_j \in NT$, $p_m \in SW_j$, $u_k \in U$, $F_l \in NF$ and $d \in \{inbound, outbound\}$
- $flow\_mod(SW_j, cmd, opts, match, fwdlist)$

Case study topology: OpenStack platform



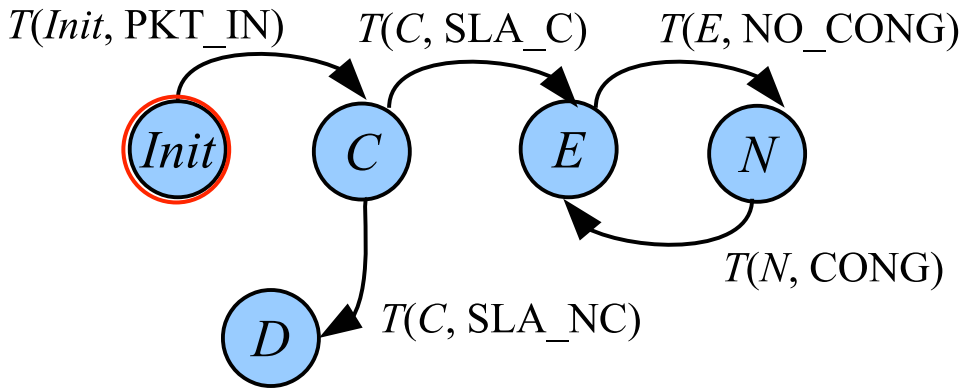
- BU/RU: Business/Residential User
- DEST: Destination Server
- DPI: Deep Packet Inspection
- WANA: WAN Accelerator
- TC: Traffic Conditioner
- VR: Virtual Router

Case study definitions

- $NT = \{SW_1, SW_2\}$
- $U = \{BU, RU, DEST\}$
- $NF = \{DPI, TC, WANA_1, WANA_2, VR_1, VR_2\}$
- Service chain definition:
  - $Ch(f_{BU}, Init) = Ch(f_{BU}, D) = nill$
  - $Ch(f_{BU}, C) = \{DPI \ \& \ VR_1\}$
  - $Ch(f_{BU}, E) = \{WANA_1, VR_1\}$
  - $Ch(f_{BU}, N) = \{VR_1\}$

- What about actions during state transition?

Init state actions



Initialization

1: **for all** F_l in $\{\text{TC}, \text{WANA}_1, \text{WANA}_2\}$ **do**
2: $(sw_{\text{in}}, p_{\text{in}}) = get\_in\_port(F_l, \text{outbound})$
3: $(sw_{\text{out}}, p_{\text{out}}) = get\_out\_port(F_l, \text{outbound})$
4: *fwdlist* = append("drop")
5: **for all** (sw, p) in $\{(sw_{\text{in}}, p_{\text{in}}), (sw_{\text{out}}, p_{\text{out}})\}$ **do**
6: *match* = "ofp_match(in_port = p, dl_type = ARP_TYPE, dl_dst = ETHER_BCAST)"
7: $flow\_mod(sw, \text{ADD}, \text{nil}, match, fwdlist)$
8: **end for**
9: **end for**

Init state actions

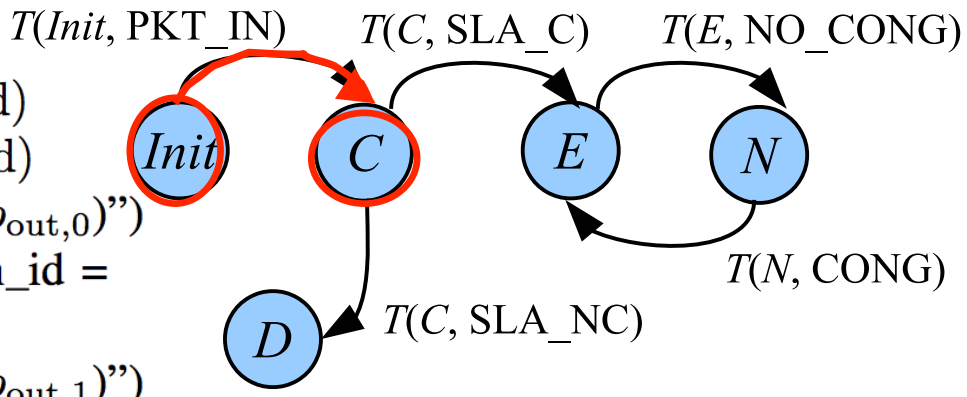
Initialization

1: **for all** F_l in $\{\text{TC}, \text{WANA}_1, \text{WANA}_2\}$ **do**
2: $(sw_{\text{in}}, p_{\text{in}}) = get\_in\_port(F_l, \text{outbound})$
3: $(sw_{\text{out}}, p_{\text{out}}) = get\_out\_port(F_l, \text{outbound})$
4: $fwdlist$ = append("drop")
5: **for all** (sw, p) in $\{(sw_{\text{in}}, p_{\text{in}}), (sw_{\text{out}}, p_{\text{out}})\}$ **do**
6: $match$ = "ofp_match(in_port = p, dl_type = ARP_TYPE, dl_dst = ETHER_BCAST)"
7: $flow\_mod(sw, \text{ADD}, \text{nil}, match, fwdlist)$
8: **end for**
9: **end for**

- Installing flow independent rules
  - ARP storm avoidance

T(Init, PKT_IN) = (C, A(f_{BU},Init,PKT_IN))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $(sw, p_{\text{out},0}) = get\_in\_port(\text{DPI}, \text{outbound})$
3: $(sw, p_{\text{out},1}) = get\_in\_port(\text{VR}_1, \text{outbound})$
4: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
5: *fwdlist* = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
6: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
7: *opts* = "hto=t_{out}, priority=h"
8: *match* = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
9: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
10: $(sw, p_{\text{in}}) = get\_out\_port(\text{VR}_1, \text{inbound})$
11: $(sw, p_{\text{out},0}) = get\_port(\text{BU})$
12: $(sw, p_{\text{out},1}) = get\_in\_port(\text{DPI}, \text{inbound})$
13: *fwdlist* = append("ofp_action_strip_vlan_vid()")
14: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
15: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
16: *match* = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
17: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$



T(Init, PKT_IN) = (C, A(f_{BU},Init,PKT_IN))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $(sw, p_{\text{out},0}) = get\_in\_port(\text{DPI}, \text{outbound})$
3: $(sw, p_{\text{out},1}) = get\_in\_port(\text{VR}_1, \text{outbound})$
4: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
5: *fwdlist* = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
6: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
7: *opts* = "hto=t_{out}, priority=h"
8: *match* = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
9: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
10: $(sw, p_{\text{in}}) = get\_out\_port(\text{VR}_1, \text{inbound})$
11: $(sw, p_{\text{out},0}) = get\_port(\text{BU})$
12: $(sw, p_{\text{out},1}) = get\_in\_port(\text{DPI}, \text{inbound})$
13: *fwdlist* = append("ofp_action_strip_vlan_vid()")
14: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
15: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
16: *match* = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
17: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$

Getting topology information

T(Init, PKT_IN) = (C, A(f_{BU},Init,PKT_IN))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $(sw, p_{\text{out},0}) = get\_in\_port(\text{DPI}, \text{outbound})$
3: $(sw, p_{\text{out},1}) = get\_in\_port(\text{VR}_1, \text{outbound})$
4: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
5: *fwdlist* = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
6: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
7: *opts* = "hto=t_{out}, priority=h"
8: *match* = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
9: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
10: $(sw, p_{\text{in}}) = get\_out\_port(\text{VR}_1, \text{inbound})$
11: $(sw, p_{\text{out},0}) = get\_port(\text{BU})$
12: $(sw, p_{\text{out},1}) = get\_in\_port(\text{DPI}, \text{inbound})$
13: *fwdlist* = append("ofp_action_strip_vlan_vid()")
14: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
15: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
16: *match* = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
17: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$

OpenFlow actions

T(Init, PKT_IN) = (C, A(f_{BU},Init,PKT_IN))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $(sw, p_{\text{out},0}) = get\_in\_port(\text{DPI}, \text{outbound})$
3: $(sw, p_{\text{out},1}) = get\_in\_port(\text{VR}_1, \text{outbound})$
4: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
5: *fwdlist* = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
6: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
7: *opts* = "hto=t_{out}, priority=h"
8: *match* = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
9: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
10: $(sw, p_{\text{in}}) = get\_out\_port(\text{VR}_1, \text{inbound})$
11: $(sw, p_{\text{out},0}) = get\_port(\text{BU})$
12: $(sw, p_{\text{out},1}) = get\_in\_port(\text{DPI}, \text{inbound})$
13: *fwdlist* = append("ofp_action_strip_vlan_vid()")
14: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
15: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
16: *match* = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
17: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$

Installing rule on the switch

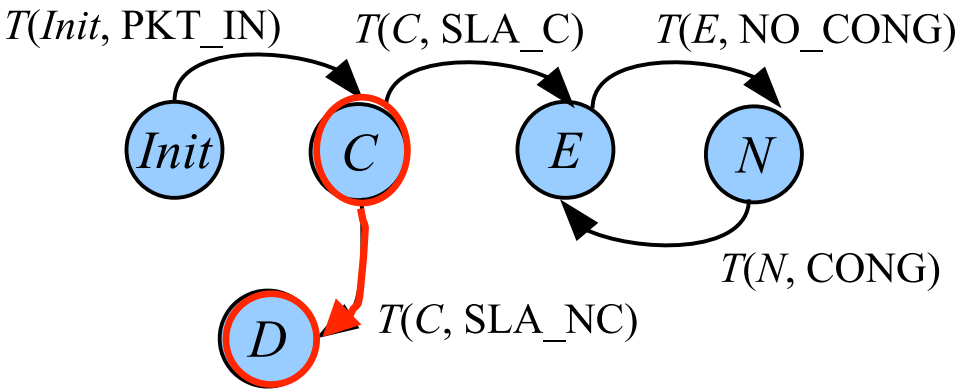
T(Init, PKT_IN) = (C, A(f_{BU},Init,PKT_IN))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $(sw, p_{\text{out},0}) = get\_in\_port(\text{DPI}, \text{outbound})$
3: $(sw, p_{\text{out},1}) = get\_in\_port(\text{VR}_1, \text{outbound})$
4: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
5: *fwdlist* = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
6: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
7: *opts* = "hto=t_{out}, priority=h"
8: *match* = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
9: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
10: $(sw, p_{\text{in}}) = get\_out\_port(\text{VR}_1, \text{inbound})$
11: $(sw, p_{\text{out},0}) = get\_port(\text{BU})$
12: $(sw, p_{\text{out},1}) = get\_in\_port(\text{DPI}, \text{inbound})$
13: *fwdlist* = append("ofp_action_strip_vlan_vid()")
14: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},0}$)")
15: *fwdlist* = append("ofp_action_output(port=$p_{\text{out},1}$)")
16: *match* = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
17: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$

Handling bidirectional flows

T(C, SLA_NC) = (D, A(f_{BU},C,SLA_NC))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $fwdlist$ = append("drop")
3: $opts$ = "priority=$h + 1$"
4: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
5: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$



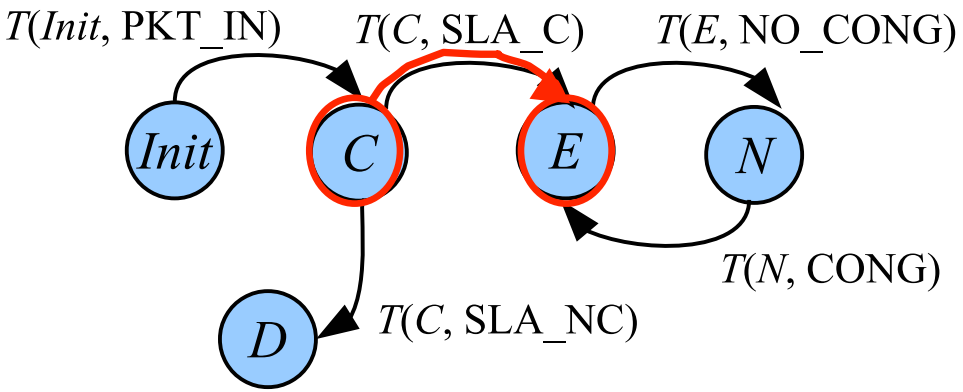
T(C, SLA_NC) = (D, A(f_{BU},C,SLA_NC))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $fwdlist$ = append("drop")
3: $opts$ = "priority=$h + 1$"
4: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
5: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$

Dropping non compliant traffic

T(C, SLA_C) = (E, A(f_{BU},C,SLA_C))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $(sw, p_{\text{out}}) = get\_in\_port(\text{WANA}_1, \text{outbound})$
3: $fwdlist$ = append("ofp_action_output(port=p_{out})")
4: $opts$ = "priority=$h + 1$"
5: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
6: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
7: $(sw, p_{\text{in}}) = get\_out\_port(\text{WANA}_1, \text{outbound})$
8: $(sw, p_{\text{out}}) = get\_in\_port(\text{VR}_1, \text{outbound})$
9: $fwdlist$ = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
10: $fwdlist$ = append("ofp_action_output(port=p_{out})")
11: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
12: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
13: $(sw, p_{\text{in}}) = get\_out\_port(\text{VR}_1, \text{inbound})$
14: $(sw, p_{\text{out}}) = get\_in\_port(\text{WANA}_1, \text{inbound})$
15: $fwdlist$ = append("ofp_action_strip_vlan_vid()")
16: $fwdlist$ = append("ofp_action_output(port=p_{out})")
17: $match$ = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
18: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
19: $(sw, p_{\text{in}}) = get\_out\_port(\text{WANA}_1, \text{inbound})$
20: $(sw, p_{\text{out}}) = get\_port(\text{BU})$
21: $fwdlist$ = append("ofp_action_output(port=p_{out})")
22: $match$ = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
23: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$



T(C, SLA_C) = (E, A(f_{BU},C,SLA_C))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $(sw, p_{\text{out}}) = get\_in\_port(\text{WANA}_1, \text{outbound})$
3: $fwdlist$ = append("ofp_action_output(port=p_{out})")
4: $opts$ = "priority=$h + 1$"
5: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
6: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
7: $(sw, p_{\text{in}}) = get\_out\_port(\text{WANA}_1, \text{outbound})$
8: $(sw, p_{\text{out}}) = get\_in\_port(\text{VR}_1, \text{outbound})$
9: $fwdlist$ = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
10: $fwdlist$ = append("ofp_action_output(port=p_{out})")
11: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
12: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
13: $(sw, p_{\text{in}}) = get\_out\_port(\text{VR}_1, \text{inbound})$
14: $(sw, p_{\text{out}}) = get\_in\_port(\text{WANA}_1, \text{inbound})$
15: $fwdlist$ = append("ofp_action_strip_vlan_vid()")
16: $fwdlist$ = append("ofp_action_output(port=p_{out})")
17: $match$ = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
18: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
19: $(sw, p_{\text{in}}) = get\_out\_port(\text{WANA}_1, \text{inbound})$
20: $(sw, p_{\text{out}}) = get\_port(\text{BU})$
21: $fwdlist$ = append("ofp_action_output(port=p_{out})")
22: $match$ = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
23: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$

- Traffic steering:
  - First sent to WANA_1
  - Then to VR_1
  - Reverse order for inbound traffic case

T(C, SLA_C) = (E, A(f_{BU},C,SLA_C))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $(sw, p_{\text{out}}) = get\_in\_port(\text{WANA}_1, \text{outbound})$
3: $fwdlist$ = append("ofp_action_output(port=p_{out})")
4: $opts$ = "priority=$h + 1$"
5: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
6: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
7: $(sw, p_{\text{in}}) = get\_out\_port(\text{WANA}_1, \text{outbound})$
8: $(sw, p_{\text{out}}) = get\_in\_port(\text{VR}_1, \text{outbound})$
9: $fwdlist$ = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
10: $fwdlist$ = append("ofp_action_output(port=p_{out})")
11: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
12: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
13: $(sw, p_{\text{in}}) = get\_out\_port(\text{VR}_1, \text{inbound})$
14: $(sw, p_{\text{out}}) = get\_in\_port(\text{WANA}_1, \text{inbound})$
15: $fwdlist$ = append("ofp_action_strip_vlan_vid()")
16: $fwdlist$ = append("ofp_action_output(port=p_{out})")
17: $match$ = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
18: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
19: $(sw, p_{\text{in}}) = get\_out\_port(\text{WANA}_1, \text{inbound})$
20: $(sw, p_{\text{out}}) = get\_port(\text{BU})$
21: $fwdlist$ = append("ofp_action_output(port=p_{out})")
22: $match$ = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
23: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$

- Traffic steering:
  - First sent to WANA_1
  - Then to VR_1
  - Reverse order for inbound traffic case

T(C, SLA_C) = (E, A(f_{BU},C,SLA_C))

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $(sw, p_{\text{out}}) = get\_in\_port(\text{WANA}_1, \text{outbound})$
3: $fwdlist$ = append("ofp_action_output(port=p_{out})")
4: $opts$ = "priority=$h + 1$"
5: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
6: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
7: $(sw, p_{\text{in}}) = get\_out\_port(\text{WANA}_1, \text{outbound})$
8: $(sw, p_{\text{out}}) = get\_in\_port(\text{VR}_1, \text{outbound})$
9: $fwdlist$ = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
10: $fwdlist$ = append("ofp_action_output(port=p_{out})")
11: $match$ = "ofp_match(in_port=p_{in},$get\_match(f_{\text{BU}})$)"
12: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
13: $(sw, p_{\text{in}}) = get\_out\_port(\text{VR}_1, \text{inbound})$
14: $(sw, p_{\text{out}}) = get\_in\_port(\text{WANA}_1, \text{inbound})$
15: $fwdlist$ = append("ofp_action_strip_vlan_vid()")
16: $fwdlist$ = append("ofp_action_output(port=p_{out})")
17: $match$ = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
18: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
19: $(sw, p_{\text{in}}) = get\_out\_port(\text{WANA}_1, \text{inbound})$
20: $(sw, p_{\text{out}}) = get\_port(\text{BU})$
21: $fwdlist$ = append("ofp_action_output(port=p_{out})")
22: $match$ = "ofp_match(in_port=p_{in},$get\_match(f'_{\text{BU}})$)"
23: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$

- Traffic steering:
  - First sent to WANA_1
  - Then to VR_1
  - Reverse order for inbound traffic case

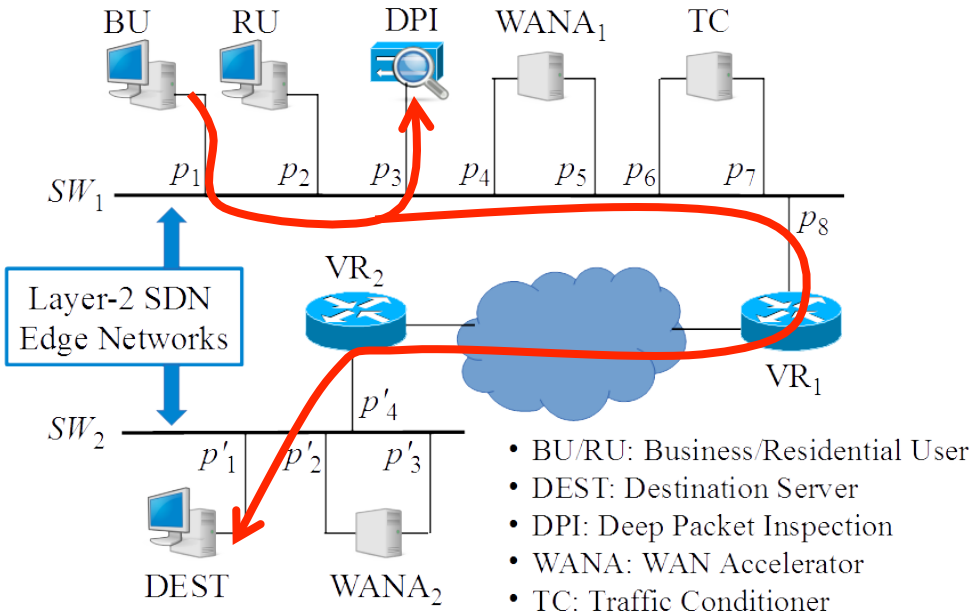
State transitions: observations

- Others state transitions can be easily derived from previous cases:
  - *T(E, NO_CONG) = (N, A(f_{BU},E, NO_CONG))*: steps similar to those of state transition from *Init* to *C*
  - *T(N, CONG) = (E, A(f_{BU},N, CONG))*: steps similar to those of state transition from *E* to *N* (*flow_mod* command changed to DELETE)
- VNF chaining is driven by current network conditions
- Steering actions can also be replicated for RU flows

Testbed setup

- Network topology created with OpenStack platform
  - POX SDN Controller
  - BU, RU, DPI (nDPIReader), $WANA_1$ (TrafficSqueezer) and TC implemented as VMs
  - Destination edge network outside OpenStack cluster
- Throughput measured at each VNF ports
  - DPI
  - $WANA_1$
  - TC
  - VR_1
- Iperf traffic generator: 100 Mbit/s

Proof of concept: SDN controller design

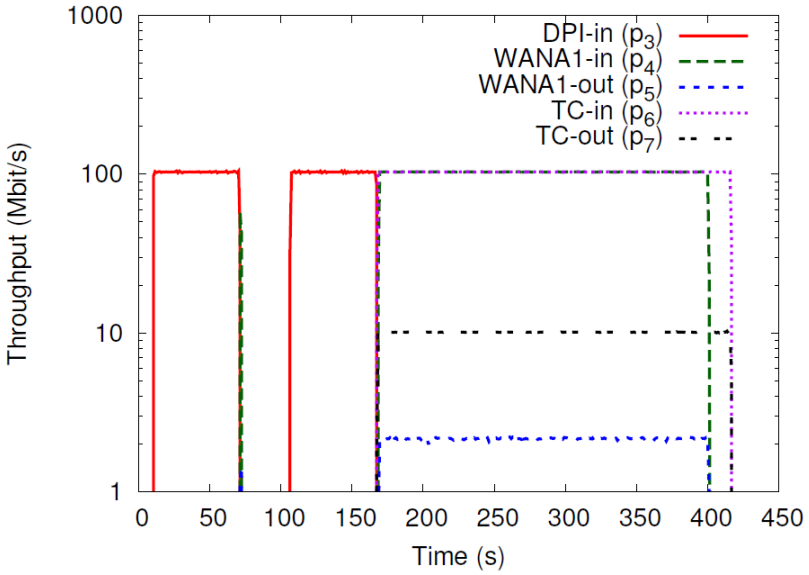


- BU/RU: Business/Residential User
- DEST: Destination Server
- DPI: Deep Packet Inspection
- WANA: WAN Accelerator
- TC: Traffic Conditioner
- VR: Virtual Router

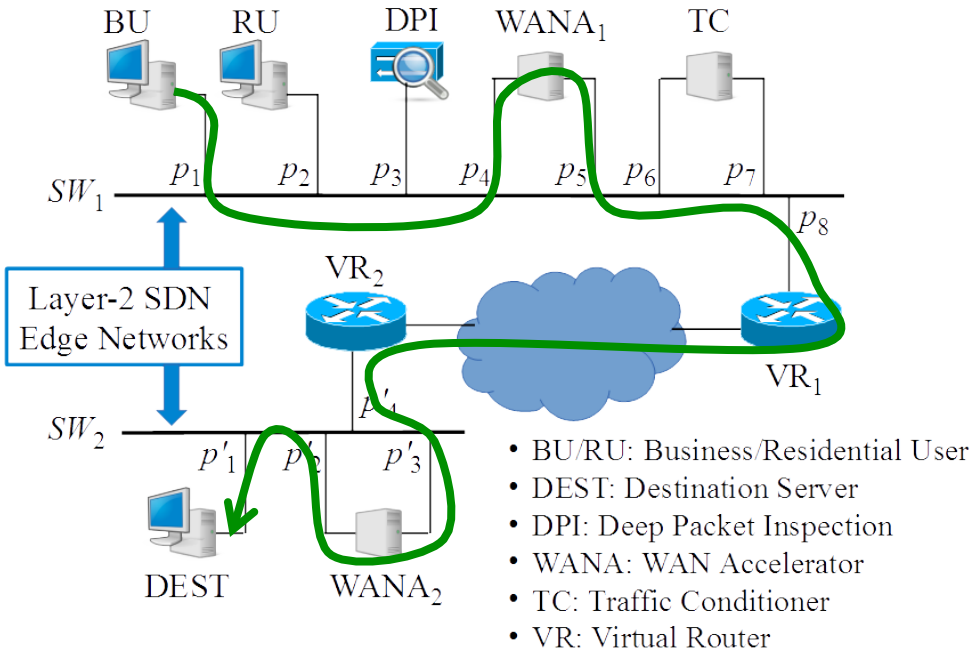
Phase 1: classification

Rules installed in SW1:

- Traffic from to DEST is forwarded both to VR1 and DPI
- Similarly for inbound packets



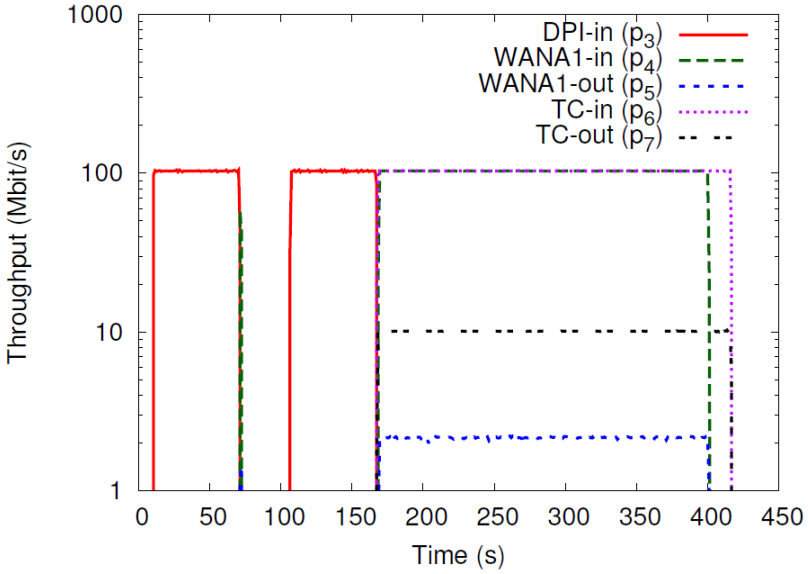
Proof of concept: SDN controller design



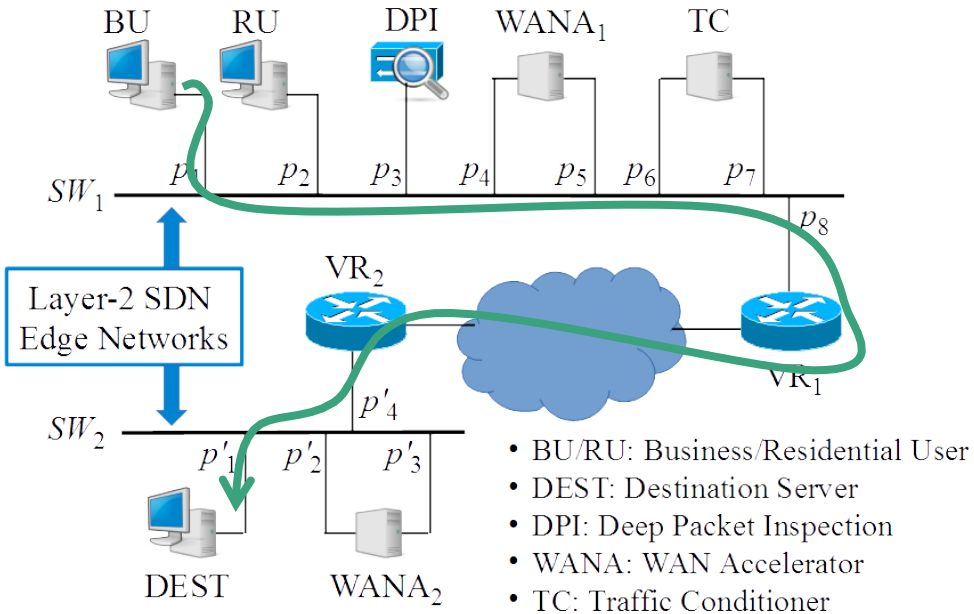
Phase 2: SLA compliance

Rules installed in SW1:

- Traffic from to DEST is forwarded to VR1 via WANA1
- Similarly for inbound packets



Proof of concept: SDN controller design

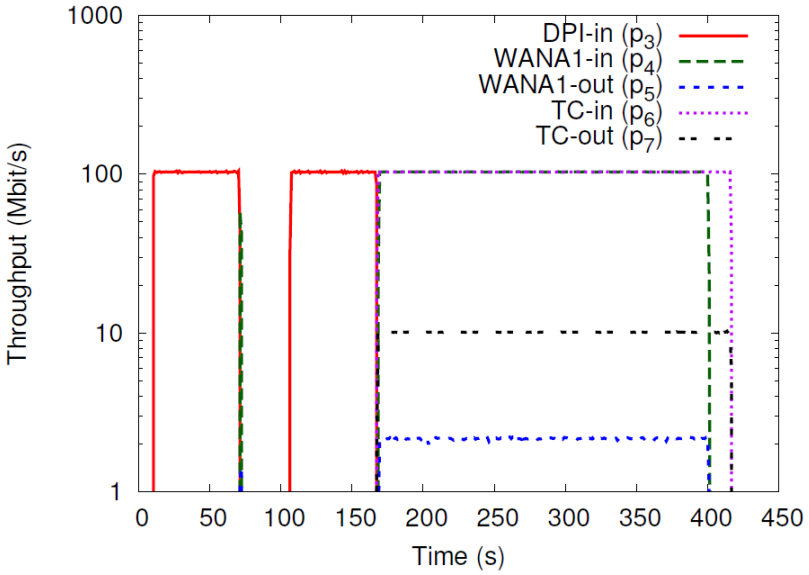


- BU/RU: Business/Residential User
- DEST: Destination Server
- DPI: Deep Packet Inspection
- WANA: WAN Accelerator
- TC: Traffic Conditioner
- VR: Virtual Router

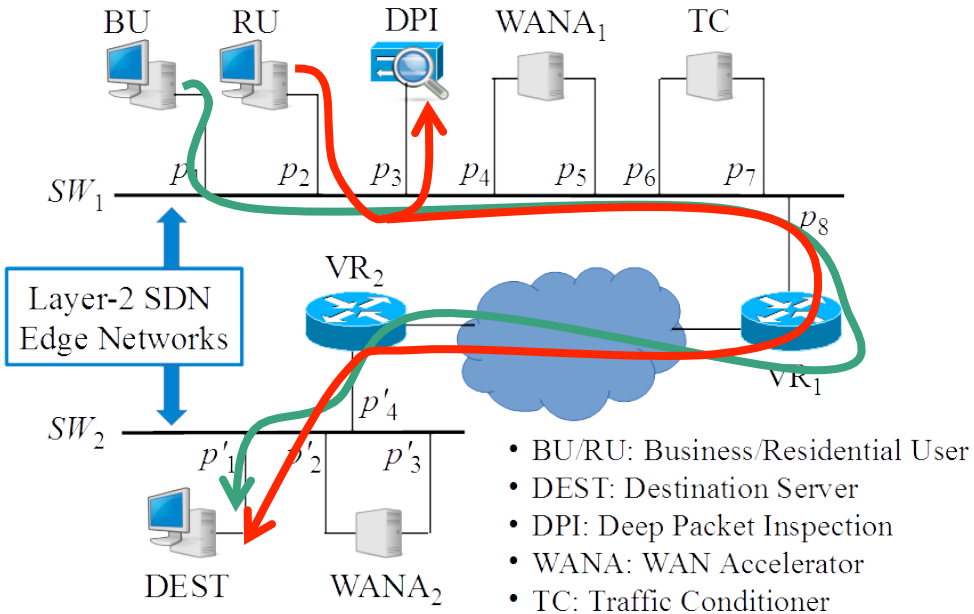
Phase 3: no congestion → SLA not enforced

Rules installed in SW1:

- Traffic from to DEST is forwarded directly to VR1
- Similarly for inbound packets



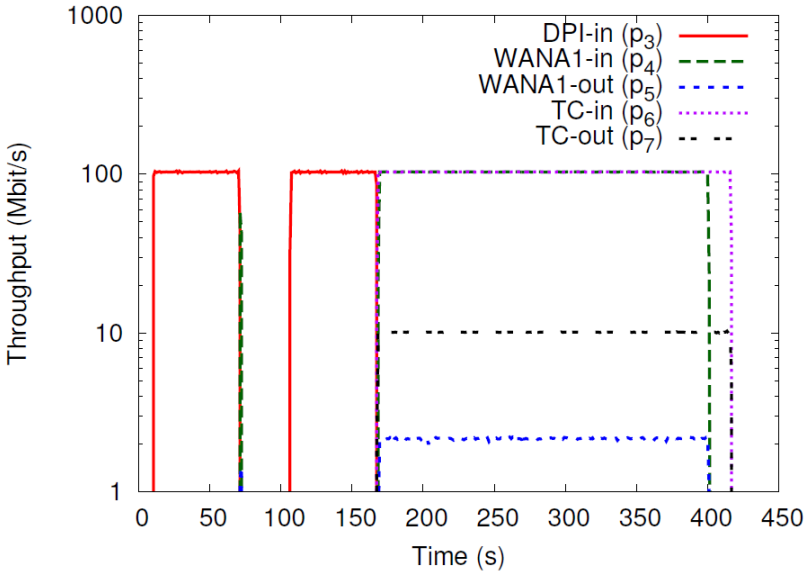
Proof of concept: SDN controller design



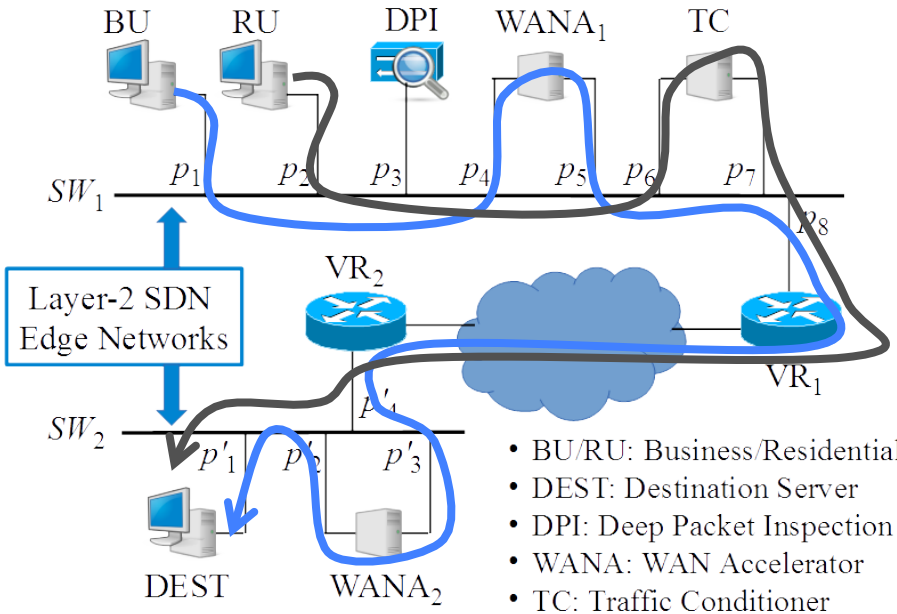
Phase 4: classification

Rules installed in SW1:

- Traffic from BU to DEST is forwarded directly to VR1
- Traffic from RU to DEST is forwarded both to VR1 and DPI
- Similarly for inbound packets



Proof of concept: SDN controller design

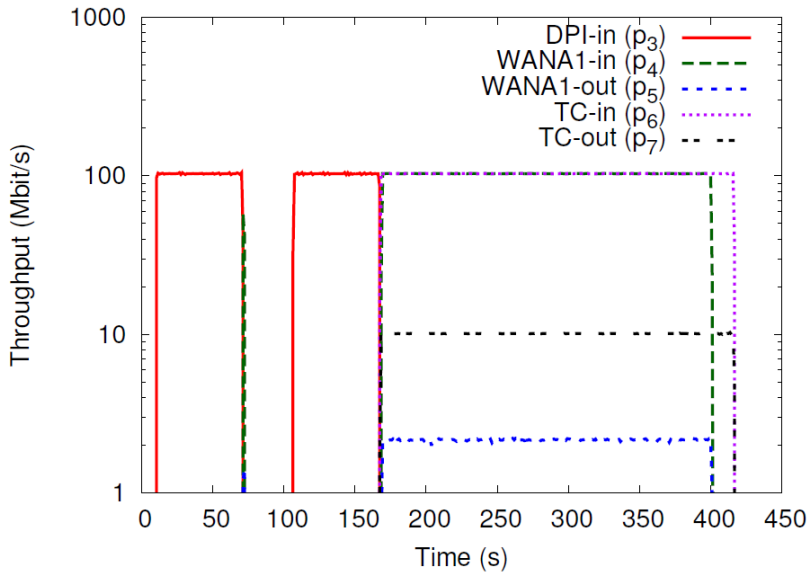


- BU/RU: Business/Residential User
- DEST: Destination Server
- DPI: Deep Packet Inspection
- WANA: WAN Accelerator
- TC: Traffic Conditioner
- VR: Virtual Router

Phase 5: SLA compliance & congestion → SLA enforced

Rules installed in SW1:

- Traffic from to DEST is forwarded to VR1 via $WANA_1$
- Traffic from RU to DEST is forwarded to VR_1 via TC
- Similarly for inbound packets



Measurements

Flow	State transition	Time (s)
f_{BU}	$Init \rightarrow C$	10.62
	$C \rightarrow E$	71.36
	$E \rightarrow N$	73.46
	$N \rightarrow E$	168.43
	flow terminated	404.11
f_{RU}	$Init \rightarrow C$	106.45
	$C \rightarrow E$	167.36
	flow terminated	416.74

Indexes:

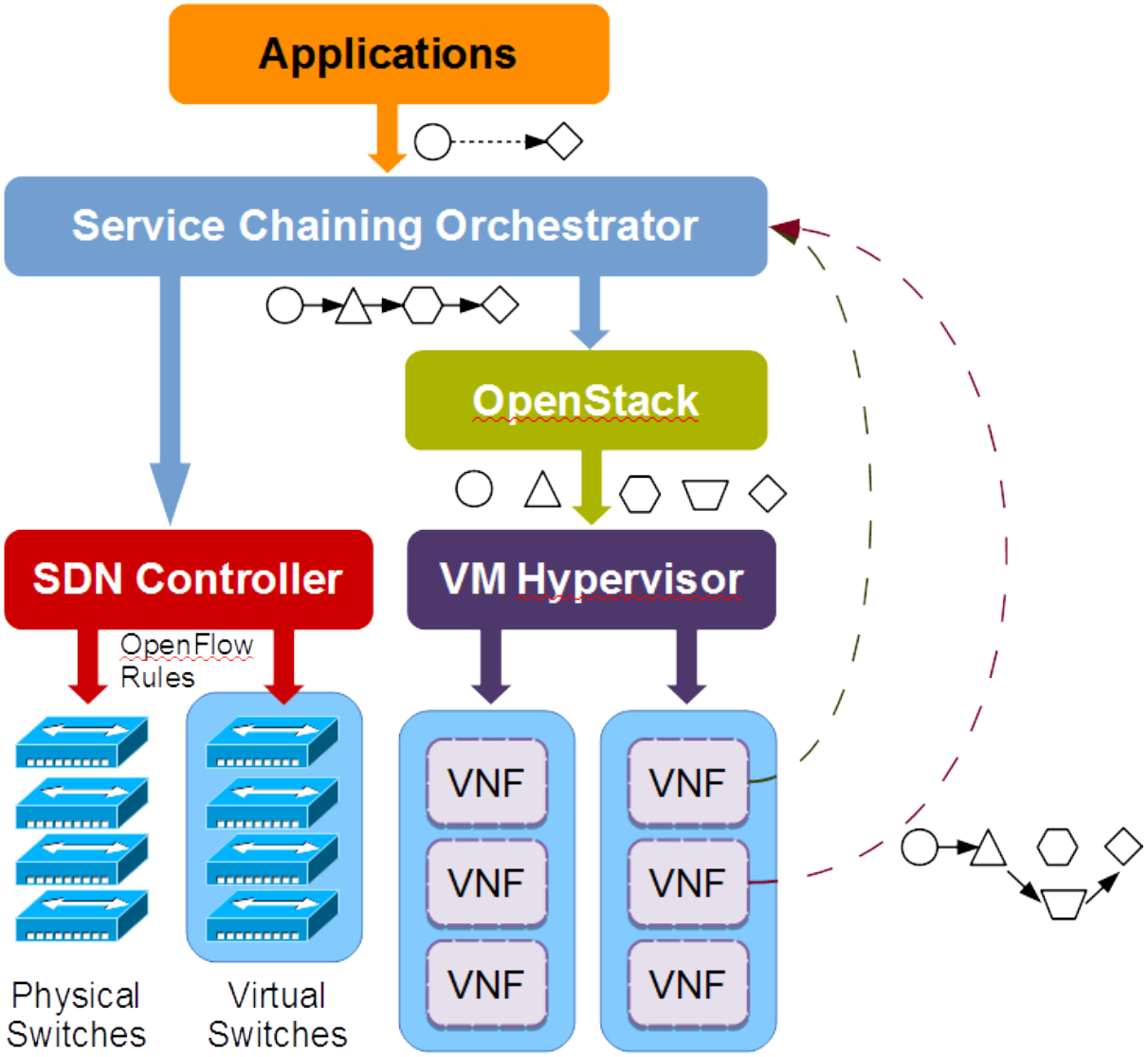
- RTT and Jitter experienced by UDP flows generated by RU
- Average obtained from 20 experiments
    - VNFs placed on the same server

	State	Max	Min	Average
RTT (ms)	C	14,840	1,525	3,972
	E	10,382	2,006	3,313
Jitter (ms)	C	0,322	0,080	0,125
	E	0,491	0,067	0,152

Conclusion

- Design methodology for a SDN Controller capable of steering traffic flows in a dynamic NFV environment
  - QoS enforcement in a multi-tenant cloud scenario
  - Proof-of-concept on the OpenStack platform
- General approach adopted
  - FSM able to capture sequence of operations that need to be execute on flows, regardless underlying network infrastructure
  - It can be further extended
- Towards a possible orchestration approach
  - Mutual dependence of different flows
  - Network resource contention

Functional architecture



THANKS FOR YOUR ATTENTION!

QUESTIONS?