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
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

Data flow $f_1(t_0)$
Data flow $f_2(t_0)$
Data flow $f_2(t_1)$

Time chaining diversity
Space chaining diversity
Goal

• Suggest a design methodology for implementing SDN control plane capable of
  o Dynamically steering traffic towards required VNF
  o Achieving fully adaptive service chaining
• Case study:
  o Layer 2 (L2) Edge network
  o OpenStack cloud platform
• What about SDN controller design?
  o Mealy Machine abstraction as general approach to service chain reconfiguration
  o 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 states
    \( s \in \{Init, C, E, N, D\} \)
  - \( i \): 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
General parameters

- \( NT = \{SW_1, SW_2, ..., SW_{NSW}\} \) set of switch
- \( SW_j = \{p_1, p_2, ..., p_{Np,j}\} \) set of ports
- \( U = \{u_1, u_2, ..., u_{Nu}\} \) set of users
- \( NF = \{F_1, F_2, ..., F_{NF}\} \) set of VNFs
- \( Ch(f, s) = \{F_{l1}, F_{l2}, ..., F_{ln(f,s)}\} \) service chain of \( n(f,s) \) VNFs applied to \( f \) in state \( s \)

- Getting topology information:
  - \( (SW_j, p_m) = \text{get\_port}(u_k) \)
  - \( (SW_j, p_m) = \text{get\_in\_port}(F_l, d) \)
  - \( (SW_j, p_m) = \text{get\_out\_port}(F_l, d) \)
- \( SW_j \in NT, p_m \in SW_j, u_k \in U, F_l \in NF \) and \( d \in \{\text{inbound, outbound}\} \)
- \( \text{flow\_mod}(SW_j, \text{cmd}, \text{opts}, \text{match}, \text{fwdlist}) \)
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:
  - \( Ch(f_{BU},\text{Init}) = Ch(f_{BU},D) = \text{nill} \)
  - \( Ch(f_{BU},C) = \{DPI \& VR_1\} \)
  - \( Ch(f_{BU},E) = \{WANA_1, VR_1\} \)
  - \( Ch(f_{BU},N) = \{VR_1\} \)

- What about actions during state transition?
Init state actions

**Initialization**

1. **for all** \( F_i \) in \{TC, WANA_1, WANA_2\} **do**
2. \((s_{win}, p_{in}) = get\_in\_port(F_i, outbound)\)
3. \((s_{out}, p_{out}) = get\_out\_port(F_i, outbound)\)
4. \(fwdlist = append(\text{"drop"})\)
5. **for all** \((s_w, p)\) in \{(s_{win}, p_{in}), (s_{out}, p_{out})\} **do**
6. \(match = \text{"ofp\_match(in\_port = p, dl\_type = ARP\_TYPE, dl\_dst = ETHER\_BCAST)"}\)
7. \(flow\_mod(s_w, \text{ADD, nil, match, fwdlist})\)
8. **end for**
9. **end for**
Init state actions

- Installing flow independent rules
  - ARP storm avoidance

Programming example:

```
Initialization
1: for all $F_i$ in \{TC, WANA1, WANA2\} do
2:   $(sw_{in}, p_{in}) = get\_in\_port(F_i, outbound)$
3:   $(sw_{out}, p_{out}) = get\_out\_port(F_i, outbound)$
4:   fwdlist = append("drop")
5:   for all $(sw, p)$ in \{(sw_{in}, p_{in}), (sw_{out}, p_{out})\} do
6:     match = "ofp\_match(in\_port = p, dl\_type = ARP\_TYPE, dl\_dst = ETHER\_BCAST)"
7:     flow\_mod(sw, ADD, nil, match, fwdlist)
8:   end for
9: end for
```
\[ T(\text{Init}, \text{PKT\_IN}) = (C, A(f_{BU}, \text{Init}, \text{PKT\_IN})) \]

1. \((sw, p_{in}) = \text{get\_port}(BU)\)
2. \((sw, p_{out,0}) = \text{get\_in\_port}(\text{DPI}, \text{outbound})\)
3. \((sw, p_{out,1}) = \text{get\_in\_port}(\text{VR}_1, \text{outbound})\)
4. \(\text{fwdlist} = \text{append}(\text{ofp\_action\_output(port}=p_{out,0})\)"
5. \(\text{fwdlist} = \text{append}(\text{ofp\_action\_vlan\_vid(vlan\_id = }\
\text{internal\_vid})\)"
6. \(\text{fwdlist} = \text{append}(\text{ofp\_action\_output(port}=p_{out,1})\)"
7. \(\text{opts} = \text{hto=tout, priority=h}"
8. \(\text{match} = \text{ofp\_match(in\_port}=p_{in}, \text{get\_match}(f_{BU}))\)"
9. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
10. \((sw, p_{in}) = \text{get\_out\_port}(\text{VR}_1, \text{inbound})\)
11. \((sw, p_{out,0}) = \text{get\_port}(BU)\)
12. \((sw, p_{out,1}) = \text{get\_in\_port}(\text{DPI, inbound})\)
13. \(\text{fwdlist} = \text{append}(\text{ofp\_action\_strip\_vlan\_vid()}\)"
14. \(\text{fwdlist} = \text{append}(\text{ofp\_action\_output(port}=p_{out,0})\)"
15. \(\text{fwdlist} = \text{append}(\text{ofp\_action\_output(port}=p_{out,1})\)"
16. \(\text{match} = \text{ofp\_match(in\_port}=p_{in}, \text{get\_match}(f'_{BU}))\)"
17. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
\[ T(\text{Init}, \text{PKT\_IN}) = (C, A(f_{BU}, \text{Init}, \text{PKT\_IN})) \]

1: \((sw, p_{in}) = \text{get\_port}(BU)\)
2: \((sw, p_{out, 0}) = \text{get\_in\_port}(DPI, \text{outbound})\)
3: \((sw, p_{out, 1}) = \text{get\_in\_port}(VR_1, \text{outbound})\)
4: \(\text{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=p_{out, 0})”})\)
5: \(\text{fwdlist} = \text{append}(\text{“ofp\_action\_vlan\_vid(vlan\_id = internal\_vid)”})\)
6: \(\text{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=p_{out, 1})”})\)
7: \(\text{opts} = \text{“hto=t_{out, priority=h”}}\)
8: \(\text{match} = \text{“ofp\_match(in\_port=p_{in}, get\_match(f_{BU})”})\)
9: \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
10: \((sw, p_{in}) = \text{get\_out\_port}(VR_1, \text{inbound})\)
11: \((sw, p_{out, 0}) = \text{get\_port}(BU)\)
12: \((sw, p_{out, 1}) = \text{get\_in\_port}(DPI, \text{inbound})\)
13: \(\text{fwdlist} = \text{append}(\text{“ofp\_action\_strip\_vlan\_vid()”})\)
14: \(\text{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=p_{out, 0})”})\)
15: \(\text{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=p_{out, 1})”})\)
16: \(\text{match} = \text{“ofp\_match(in\_port=p_{in}, get\_match(f’_{BU})”})\)
17: \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)

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

1: \((sw, p_{in}) = \text{get\_port}(BU)\)
2: \((sw, p_{out, 0}) = \text{get\_in\_port}(DPI, \text{outbound})\)
3: \((sw, p_{out, 1}) = \text{get\_in\_port}(VR_1, \text{outbound})\)
4: \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out, 0})"})\)
5: \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_vlan\_vid(vlan\_id = internal\_vid)"})\)
6: \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out, 1})"})\)
7: \(\text{opts} = \text{"hto=t_{out}, priority=h"} \)
8: \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})"})\)
9: \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
10: \((sw, p_{in}) = \text{get\_out\_port}(VR_1, \text{inbound})\)
11: \((sw, p_{out, 0}) = \text{get\_port}(BU)\)
12: \((sw, p_{out, 1}) = \text{get\_in\_port}(DPI, \text{inbound})\)
13: \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_strip\_vlan\_vid()"})\)
14: \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out, 0})"})\)
15: \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out, 1})"})\)
16: \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f'_{BU})"})\)
17: \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
\[ T(\text{Init}, \text{PKT\_IN}) = (C, A(f_{BU}, \text{Init}, \text{PKT\_IN})) \]

1: \( (sw, p_{in}) = \text{get\_port}(BU) \)
2: \( (sw, p_{out,0}) = \text{get\_in\_port}(\text{DPI, outbound}) \)
3: \( (sw, p_{out,1}) = \text{get\_in\_port}(\text{VR}_1, \text{outbound}) \)
4: \( \text{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=p_{out,0})”}) \)
5: \( \text{fwdlist} = \text{append}(\text{“ofp\_action\_vlan\_vid(vlan\_id = internal\_vid)”}) \)
6: \( \text{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=p_{out,1})”}) \)
7: \( \text{opts} = \text{“hto=t_{out, priority=h}”} \)
8: \( \text{match} = \text{“ofp\_match(in\_port=p_{in, get\_match(f_{BU})})”} \)
9: \( \text{flow\_mod(sw, ADD, opts, match, fwdlist)} \)
10: \( (sw, p_{in}) = \text{get\_out\_port}(\text{VR}_1, \text{inbound}) \)
11: \( (sw, p_{out,0}) = \text{get\_port}(BU) \)
12: \( (sw, p_{out,1}) = \text{get\_in\_port}(\text{DPI, inbound}) \)
13: \( \text{fwdlist} = \text{append}(\text{“ofp\_action\_strip\_vlan\_vid()”}) \)
14: \( \text{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=p_{out,0})”}) \)
15: \( \text{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=p_{out,1})”}) \)
16: \( \text{match} = \text{“ofp\_match(in\_port=p_{in, get\_match(f'_{BU})})”} \)
17: \( \text{flow\_mod(sw, ADD, opts, match, fwdlist)} \)

Installing rule on the switch
\[ T(\text{Init}, \text{PKT\_IN}) = (C, A(f_{BU}, \text{Init}, \text{PKT\_IN})) \]

1: \((sw, p_{in}) = \text{get\_port}(BU)\)
2: \((sw, p_{out,0}) = \text{get\_in\_port}(\text{DPI, outbound})\)
3: \((sw, p_{out,1}) = \text{get\_in\_port}(\text{VR}_1, \text{outbound})\)
4: fwdlist = append(“ofp\_action\_output(port=p_{out,0})”)
5: fwdlist = append(“ofp\_action\_vlan\_vid(vlan\_id = \text{internal\_vid})”)
6: fwdlist = append(“ofp\_action\_output(port=p_{out,1})”)
7: opts = “hto=t_{out}, priority=h”
8: match = “ofp\_match(in\_port=p_{in}, get\_match(f_{BU})”
9: flow\_mod(sw, ADD, opts, match, fwdlist)
10: \((sw, p_{in}) = \text{get\_out\_port}(\text{VR}_1, \text{inbound})\)
11: \((sw, p_{out,0}) = \text{get\_port}(BU)\)
12: \((sw, p_{out,1}) = \text{get\_in\_port}(\text{DPI, inbound})\)
13: fwdlist = append(“ofp\_action\_strip\_vlan\_vid()”) 
14: fwdlist = append(“ofp\_action\_output(port=p_{out,0})”) 
15: fwdlist = append(“ofp\_action\_output(port=p_{out,1})”) 
16: match = “ofp\_match(in\_port=p_{in}, get\_match(f'_{BU})”
17: flow\_mod(sw, ADD, opts, match, fwdlist)

Handling bidirectional flows
\[ T(C, \text{SLA_NC}) = (D, A(f_{BU}, C, \text{SLA_NC})) \]

1: \((sw, p_{in}) = \text{get\_port}(BU)\)
2: \(\text{fwdlist} = \text{append}(\text{“drop”})\)
3: \(\text{opts} = \text{“priority}=h + 1”\)
4: \(\text{match} = \text{“ofp\_match}(\text{in\_port}=p_{in}, \text{get\_match}(f_{BU}))”\)
5: \(\text{flow\_mod}(sw, \text{ADD}, \text{opts}, \text{match}, \text{fwdlist})\)
$$T(C, \text{SLA\_NC}) = (D, A(f_{BU}, C, \text{SLA\_NC}))$$

1: \((sw, p_{in}) = get\_port(BU)\)

2: \(fwdlist = append(\text{"drop"})\)

3: \(opts = \text{"priority=\(h + 1\)"} \)

4: \(match = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})\)"} \)

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. \((sw, p_in) = get\_port(BU)\)
2. \((sw, p_out) = get\_in\_port(WANA_1, outbound)\)
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_{BU}))\"
6. \(flow\_mod(sw, ADD, opts, match, fwdlist)\)
7. \((sw, p_in) = get\_out\_port(WANA_1, outbound)\)
8. \((sw, p_out) = get\_in\_port(VR_1, outbound)\)
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_{BU}))\"
12. \(flow\_mod(sw, ADD, opts, match, fwdlist)\)
13. \((sw, p_in) = get\_out\_port(VR_1, inbound)\)
14. \((sw, p_out) = get\_in\_port(WANA_1, inbound)\)
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_{BU}))\"
18. \(flow\_mod(sw, ADD, opts, match, fwdlist)\)
19. \((sw, p_in) = get\_out\_port(WANA_1, inbound)\)
20. \((sw, p_out) = get\_port(BU)\)
21. \(fwdlist = append("ofp\_action\_output(port=p_out)")\)
22. \(match = \"ofp\_match(in\_port=p_in, get\_match(f_{BU}))\"
23. \(flow\_mod(sw, ADD, opts, match, fwdlist)\)
\[ T(C, \text{SLA}_C) = (E, A(f_{BU}, C, \text{SLA}_C)) \]

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. \((sw, p_{in}) = \text{get\_port}(BU)\)
2. \((sw, p_{out}) = \text{get\_in\_port}(WANA_1, \text{outbound})\)
3. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out})"})\)
4. \(\text{opts} = \text{"priority=h+1"}\)
5. \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})")}\)
6. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
7. \((sw, p_{in}) = \text{get\_out\_port}(WANA_1, \text{outbound})\)
8. \((sw, p_{out}) = \text{get\_in\_port}(VR_1, \text{outbound})\)
9. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_vlan\_vid(vlan\_id = internal\_vid")})\)
10. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out})"})\)
11. \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})")}\)
12. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
13. \((sw, p_{in}) = \text{get\_out\_port}(VR_1, \text{inbound})\)
14. \((sw, p_{out}) = \text{get\_in\_port}(WANA_1, \text{inbound})\)
15. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_strip\_vlan\_vid()"})\)
16. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out})"})\)
17. \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})")}\)
18. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
19. \((sw, p_{in}) = \text{get\_out\_port}(WANA_1, \text{inbound})\)
20. \((sw, p_{out}) = \text{get\_port}(BU)\)
21. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out})"})\)
22. \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})")}\)
23. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)

- Traffic steering:
  - First sent to \(WANA_1\)
  - Then to \(VR_1\)
  - Reverse order for inbound traffic case
\[ T(C, \text{SLA}_C) = (E, A(f_{BU,C,\text{SLA}_C})) \]

1. \((sw, p_{in}) = \text{get\_port}(BU)\)
2. \((sw, p_{out}) = \text{get\_in\_port}(WANA_1, \text{outbound})\)
3. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out})"})\)
4. \(\text{opts} = \text{"priority=h+1"}\)
5. \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})"})\)
6. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
7. \((sw, p_{in}) = \text{get\_out\_port}(WANA_1, \text{outbound})\)
8. \((sw, p_{out}) = \text{get\_in\_port}(VR_1, \text{outbound})\)
9. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_vlan\_vid(vlan\_id = internal\_vid)"})\)
10. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out})"})\)
11. \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})"})\)
12. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
13. \((sw, p_{in}) = \text{get\_out\_port}(VR_1, \text{inbound})\)
14. \((sw, p_{out}) = \text{get\_in\_port}(WANA_1, \text{inbound})\)
15. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_strip\_vlan\_vid()"})\)
16. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out})"})\)
17. \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})"})\)
18. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)
19. \((sw, p_{in}) = \text{get\_out\_port}(WANA_1, \text{inbound})\)
20. \((sw, p_{out}) = \text{get\_port}(BU)\)
21. \(\text{fwdlist} = \text{append}(\text{"ofp\_action\_output(port=p_{out})"})\)
22. \(\text{match} = \text{"ofp\_match(in\_port=p_{in}, get\_match(f_{BU})"})\)
23. \(\text{flow\_mod(sw, ADD, opts, match, fwdlist)}\)

- 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 (\textit{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), WANA₁ (TrafficSqueezer) and TC implemented as VMs
  - Destination edge network outside OpenStack cluster
- Throughput measured at each VNF ports
  - DPI
  - WANA₁
  - TC
  - VR₁
- Iperf traffic generator: 100 Mbit/s
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

BU/RU: Business/Residential User
DEST: Destination Server
DPI: Deep Packet Inspection
WANA: WAN Accelerator
TC: Traffic Conditioner
VR: Virtual Router
Proof of concept: SDN controller design

Phase 3: no congestion \( \rightarrow \) 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

Phase 5: SLA compliance & congestion → SLA enforced

Rules installed in SW1:
- Traffic from RU 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

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

<table>
<thead>
<tr>
<th>Flow</th>
<th>State transition</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{BU}$</td>
<td>$Init \rightarrow C$</td>
<td>10.62</td>
</tr>
<tr>
<td></td>
<td>$C \rightarrow E$</td>
<td>71.36</td>
</tr>
<tr>
<td></td>
<td>$E \rightarrow N$</td>
<td>73.46</td>
</tr>
<tr>
<td></td>
<td>$N \rightarrow E$</td>
<td>168.43</td>
</tr>
<tr>
<td></td>
<td>flow terminated</td>
<td>404.11</td>
</tr>
<tr>
<td>$f_{RU}$</td>
<td>$Init \rightarrow C$</td>
<td>106.45</td>
</tr>
<tr>
<td></td>
<td>$C \rightarrow E$</td>
<td>167.36</td>
</tr>
<tr>
<td></td>
<td>flow terminated</td>
<td>416.74</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>State</th>
<th>Max</th>
<th>Min</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTT (ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>14,840</td>
<td>1,525</td>
<td>3,972</td>
</tr>
<tr>
<td>E</td>
<td>10,382</td>
<td>2,006</td>
<td>3,313</td>
</tr>
<tr>
<td>Jitter (ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0,322</td>
<td>0,080</td>
<td>0,125</td>
</tr>
<tr>
<td>E</td>
<td>0,491</td>
<td>0,067</td>
<td>0,152</td>
</tr>
</tbody>
</table>
Conclusion

• Design methodology for a SDN Controller capable of steering traffic flows in a dynamic NFV environment
  o QoS enforcement in a multi-tenant cloud scenario
  o Proof-of-concept on the OpenStack platform

• General approach adopted
  o FSM able to capture sequence of operations that need to be execute on flows, regardless underlying network infrastructure
  o It can be further extended

• Towards a possible orchestration approach
  o Mutual dependence of different flows
  o Network resource contention
Functional architecture
THANKS FOR YOUR ATTENTION!

QUESTIONS?