Tailflow
An OpenFlow Controller Framework

Torbjörn Törnkvist
22 March 2013

Tail-f Systems

- Founded 2005
  - HQ in Stockholm Sweden, with US sales

- Software Products:
  - ConfD – On-device Management Agent
  - NCS – Network Control System

- Customers, + 75 world-wide including:

What the talk will cover

- **Part 1:** SDN - what is it?
  - ...and where does Openflow fit in?
- **Part 2:** Tailflow
  - An Openflow controller (and architecture)

Network management (according to Wikipedia)

Refers to the:

- activities
- methods
- procedures
- tools

that pertain to the:

- operation
- administration
- maintenance
- provisioning

of networked systems.

- A network management system (NMS) is used to monitor and administer a computer network or networks.
- Common access methods include:
    - SNMP
    - Command-Line Interface (CLIs)
    - NETCONF

Network management (according to Wikipedia)

Refers to the:
- activities
- methods
- procedures
- tools

that pertain to the:
- operation
- administration
- maintenance
- provisioning

of networked systems.

TRADITIONAL VIEW

- A network management system (NMS) is used to monitor and administer a computer network or networks.
- Common access methods include:
  - SNMP
  - Command-Line Interface (CLIs)
  - NETCONF

Network management (according to Wikipedia)

Refers to the:

- activities
- methods
- procedures
- tools

that pertain to the:

- operation
- administration

- A netw…
  monit…
  netwo…
- Comm…
  - SNMP
  - Command-Line Interface (CLI)
  - NETCONF

- Lots of tedious manual work
- Error prone

- Very costly!

Software Defined Networking (SDN)

- An approach to building computer networks that **separates** and **abstracts** elements of these systems.
- This makes it possible to apply modern software engineering techniques and practices.

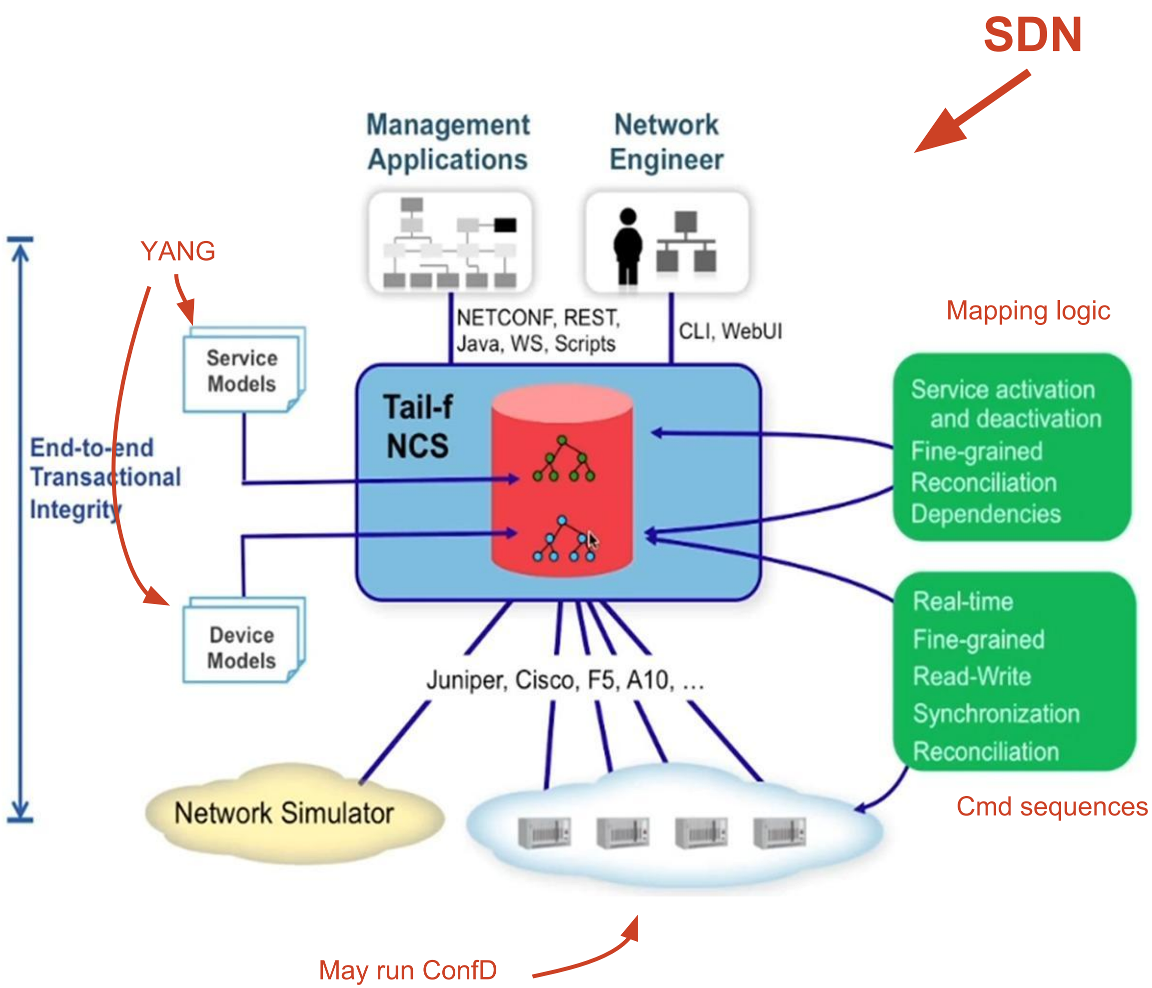
- Reduces time to deployment
- **Reduces costs!**

Tail-f products: **NCS** and **ConfD**

NCS can control and configure a heterogenus set of network elements.

- *Models*
- *Datastructures*
- *Mapping logic*
- *Auto rendered interfaces*
- *Transactions*

ConfD may run on the managed devices to provide CLI, NETCONF, SNMP... access.



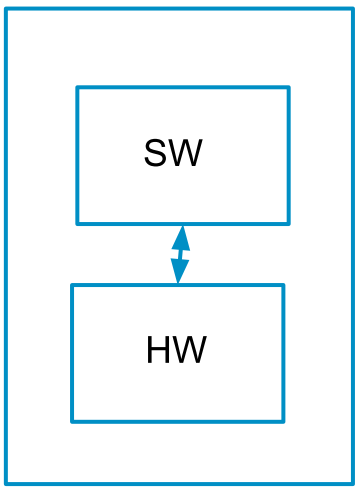
Software Defined Networking (SDN)

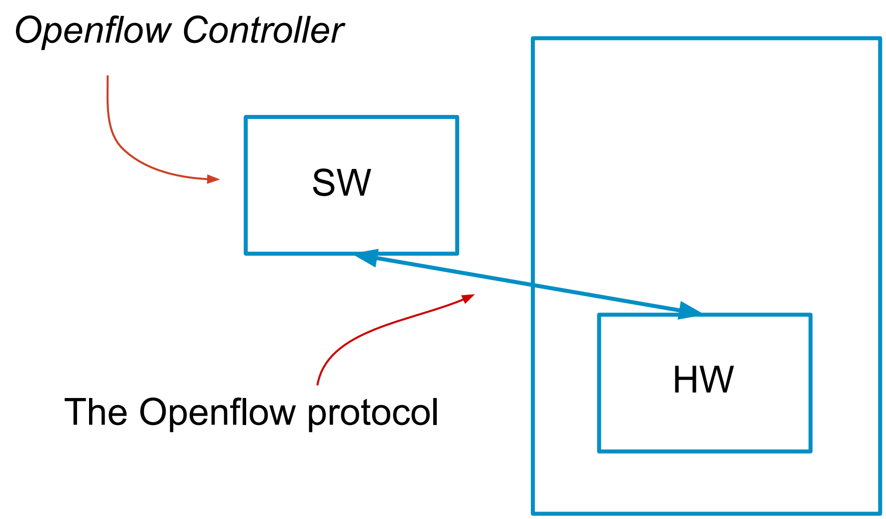
- It's not really about programming the network.

- **It's about programming network services!**

Openflow - what it is.

Traditional network element (according to my imagination)
(e.g a *switch/router/firewall*)

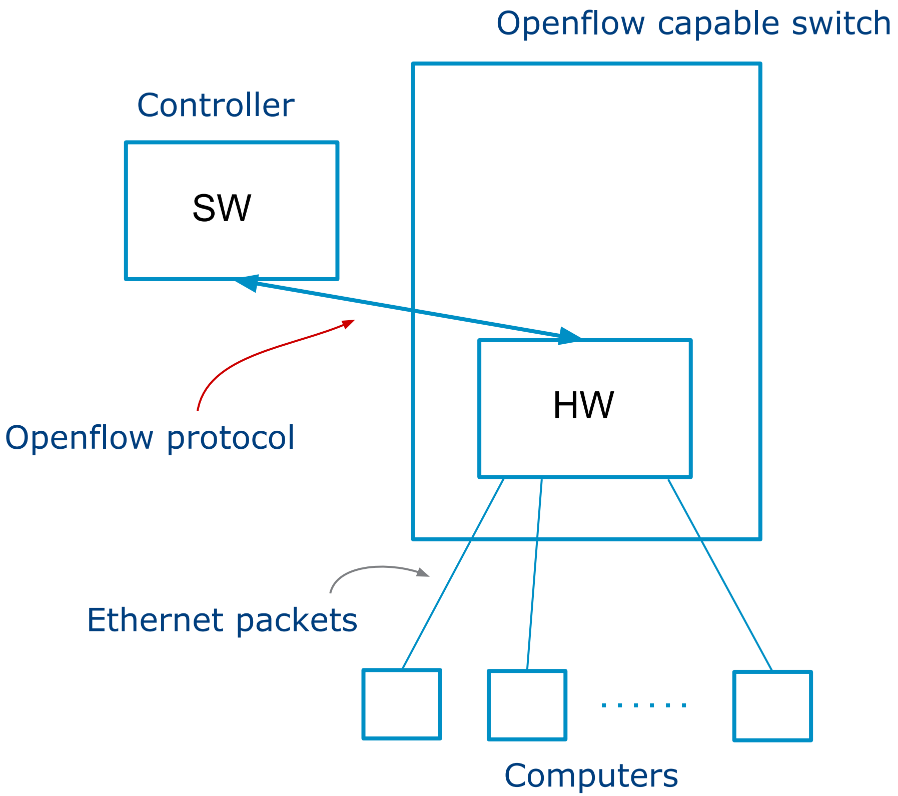




Openflow capable switch

Openflow - the essence of it

- When a packet enters the HW, it looks into its **flow table**, to see what to do with it.
- Packet header values are **matched** against the flow table entries.
- A matching entry renders corresponding **actions** to be applied to the packet.



Openflow - matching

- If no **matching** entry is found in the flow table, then send the packet to the **controller** (SW) for some decision making.

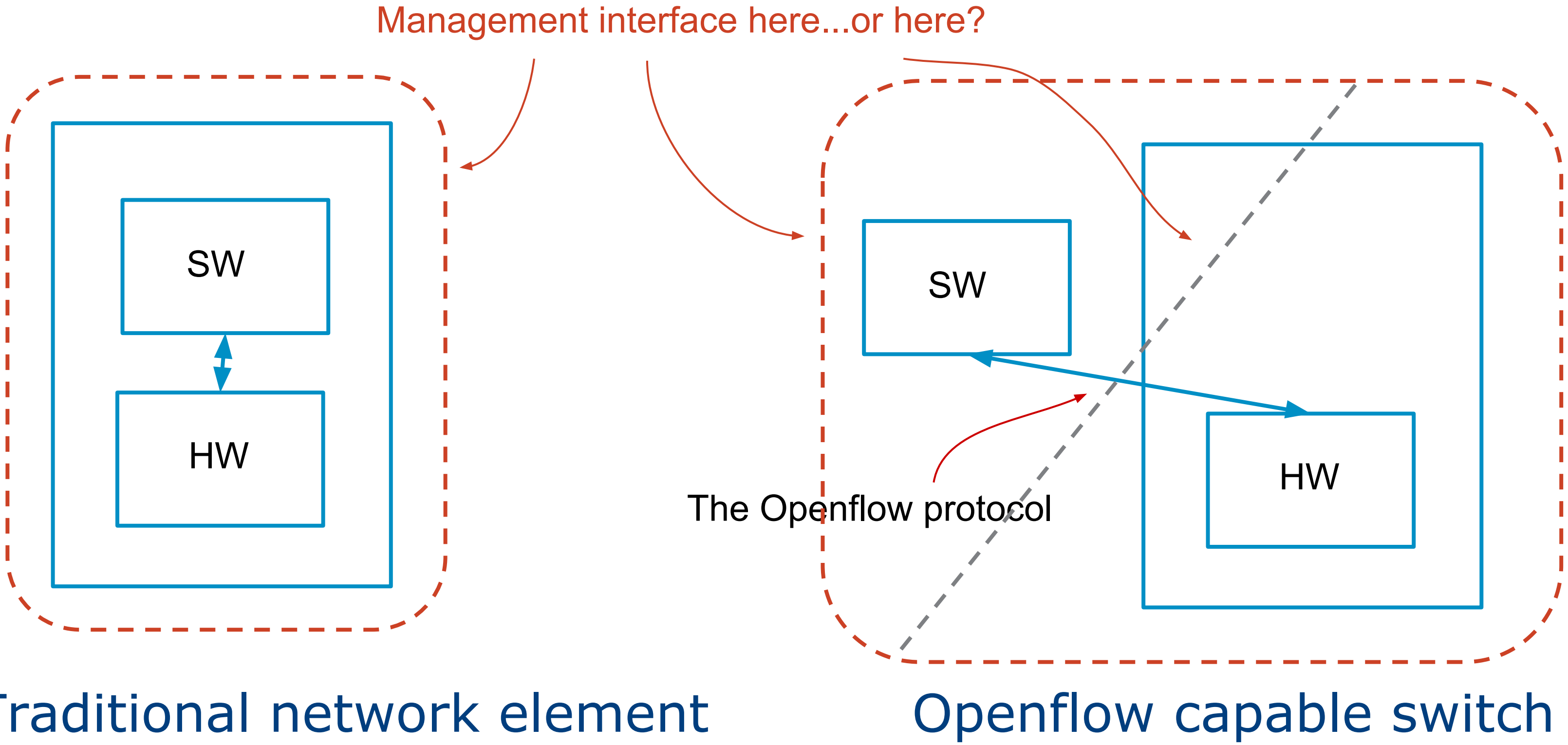
Ingress port	Ether source	Ether dest.	....	IP source	IP dest.	....	TCP/ UDP port

flow table entry

- The SW tells the HW what to do now (and in the future by inserting a flow entry into the flow table of the HW).
- Coupled to a flow entry is a set of **actions**.
- Example of some actions:
  - Send out packet on <*port*>
  - Rewrite the <*ether source*> to some other address
  - Drop the packet (i.e no actions)

Openflow device management

All kind of devices need to be managed



SDN vs Openflow

Openflow is a component of SDN

Part2: How to write an OF application?

How can/should the SW be structured?

What about management?

The role of the controller

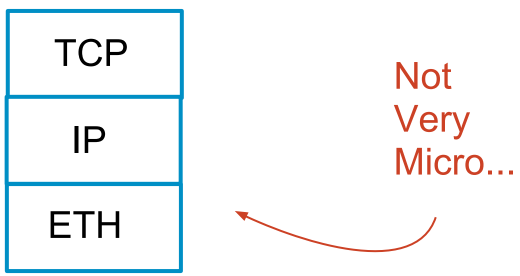
- From the Openflow controller point of view; an Openflow switch generates a number of events, for example:
    - *datapath-join* - when a switch connect to the controller
    - *packet-in* - when a packet is delivered from a switch
    - *flow-removed* - a flow (rule) was removed
      (e.g because of an expired timeout)

For each event we want to apply some logic!

Sources of inspiration

- Functional programming (of course...)
- The micro-protocol idea (what?)

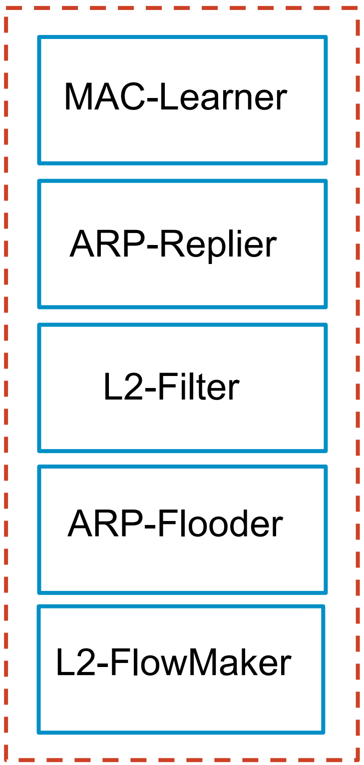
By partition complex protocols into simple micro protocols, each of which is implemented by a protocol layer. Protocol layers can be stacked on top of each other in a variety of ways.



Each layer encapsulates some minimal amount of logic in order to make it composable and easy to understand.

Somewhat more micro...

A stack of **flowlets**, forming a *virtual-L2-networks* application.



The *MAC-learner* snoop the incoming packets and store any new Mac+Port info in a DB

The *ARP-replier* can act as an ARP-proxy for a set of IP addresses.

The *L2-filter* will filter out which of the incoming packets that should be allowed further treatment of succeeding flowlets.

The *ARP-flooder* make sure that any incoming ARP request is flooded on all the switch ports.

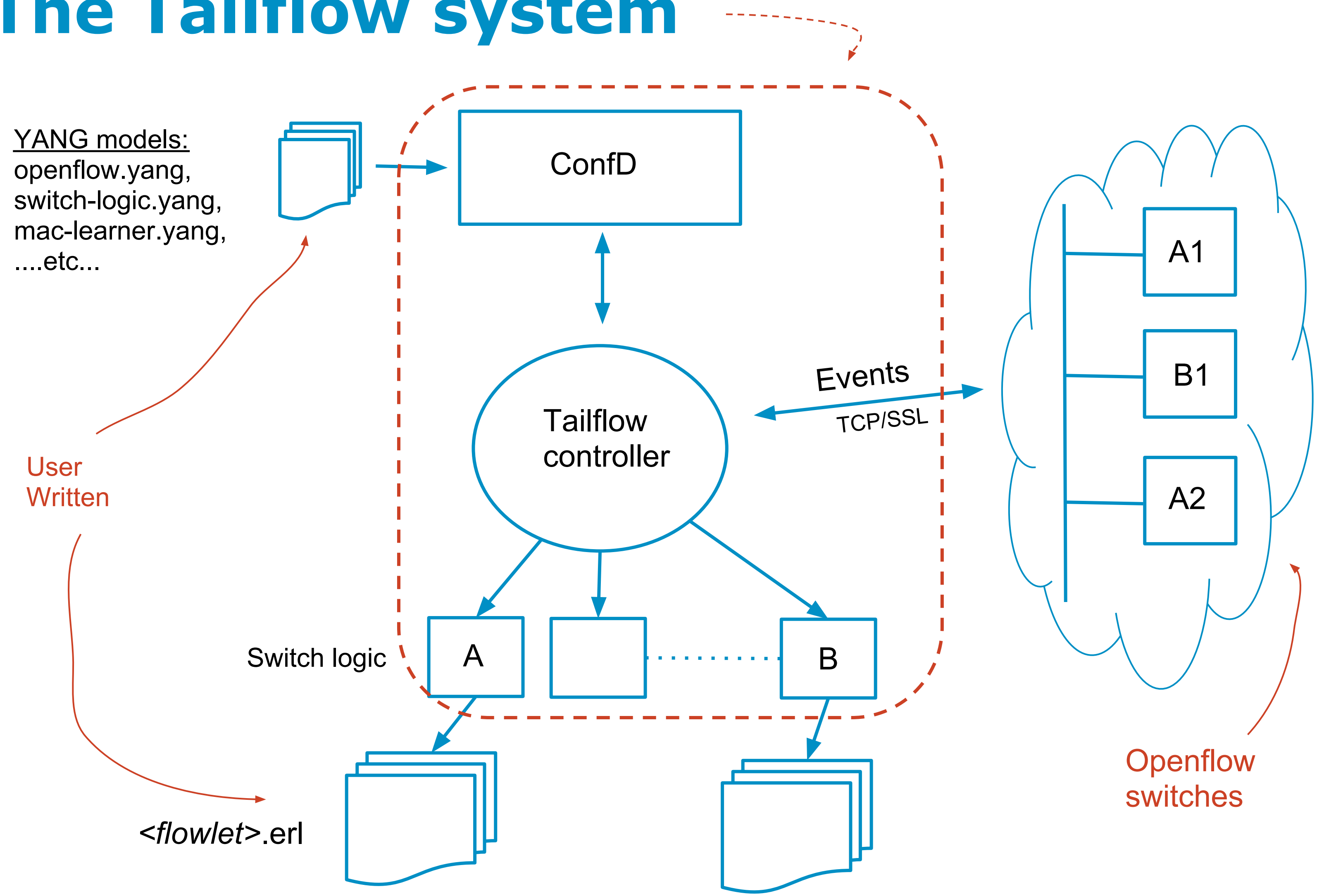
The *L2-flow-maker* creates an L2 flow for any incoming packet it sees.

Tailflow key components

- **Flowlet ::= Erlang module + Yang module**
  - The configuration is described by Yang
  - Erlang modules are ordered in execution stacks
  - An Erlang module can return either of:
    - {break, LocalState}
    - {continue, LocalState, EventState}
    - {jump, LocalState, EventState, NewStack}

- **Switch-logic ::= Flowlet configs + Flowlet stacks**
  - Can be applied to a particular switch (*datapath_id*)
  - Or to any switch

The Tailflow system



Yang model of the controller (simplified!)

```
container controller {

 leaf listen-port {type uint16;}

 list switch-logic {

   leaf name {type string;}
   leaf datapath-id {type datapath-id_t; description "Datapath identifier or Any";}

   container flowlets {description "Augmentation point for flowlets.";}

   list flowlet-execution-stack {

     leaf name {type string;}
     list flowlet {
       leaf id {type identityref {base "of:flowlet-type";} description "Flowlet identifier";}
       leaf erlang-module {type string; description "Erlang module implementing the flowlet";}
     }
   }
  }
}
```

Example config: l2-filter-flowlet

```
switch-logic switch {

    datapath-id        Any;

    flowlets {
        ...
        l2-filter {
            ...see other slide...
        }
    }

    flowlet-execution-stack switch-stack {
        flowlet mac-learner    {erlang-module tailflowlet_mac_learner;}
        flowlet arp-replier    {erlang-module tailflowlet_arp_replier;}
        flowlet l2-filter      {erlang-module tailflowlet_l2_filter;}
        flowlet arp-flooder    {erlang-module tailflowlet_arp_flooder;}
        flowlet l2-flow-maker  {erlang-module tailflowlet_l2_flow_maker;}
    }

    start-flowlet-stack switch-stack;

}
```

Yang model of: *l2-filter flowlet*

```
identity l2-filter {base of:flowlet-type;}

augment "/of:openflow/ofc:controller/ofc:switch-logic/ofc:flowlets" {

  container l2-filter {

    list rule {ordered-by user; description "Define filter rules to be applied.";
      leaf name {type string; description "Name of filter rule.";}

      container condition {
        leaf is-arp-request {type boolean; description "Check if incoming packet is an ARP request.";}
        leaf is-on-the-same-l2-network {type boolean;
              description "Check that the L2 src and dest. are on the same virtual L2 network";}
         ...more conditions here...

      container action {description "Specify what should be done..."}
        choice action_type {
          case pass {leaf pass { type empty; description "Continue with the next flowlet";}}
          case drop {leaf drop { type empty; description "Stop any further flowlet execution";}}
          case goto {
            leaf goto {
              type leafref {
                path "/of:openflow/ofc:controller/ofc:switch-logic/ofc:flowlet-execution-stack/ofc:name";
              }
            }
```

Example config: l2-filter-flowlet

```
l2-filter {

        rule allow-arp-request {
            condition { is-arp-request true;}
            action    { goto arp-stack;}
         }

        rule allow-same-l2-network {
            condition { is-on-the-same-l2-network true;}
            action    { pass;}
         }

        rule deny-all {
            action { drop;}
         }
 }
```

Example: tailflowlet_l2_filter.erl

```
-behaviour(tailflowlet).

init(#flowlet_init{ip=Ip, port=Port} = X) ->

    {ok, CDB} = tailflow_cdb:connect(Ip, Port),
     Rules = get_the_rules(X, CDB),
     #s{rules = Rules}.



packet_in(#s{rules=Rules} = State,
          #event_data{msg = Msg}) ->

    Flow = flower_flow:flow_extract(0, Msg#ofp_packet_in.in_port,
                                       Msg#ofp_packet_in.data),

    Result = exec_rules(Rules, State, Msg, Flow),
    ....etc...
```

Proof of concept demo:

Using **Tailflow** and **OpenVswitch** we implemented a control application for a simulated Data Center with virtual L2 networks.

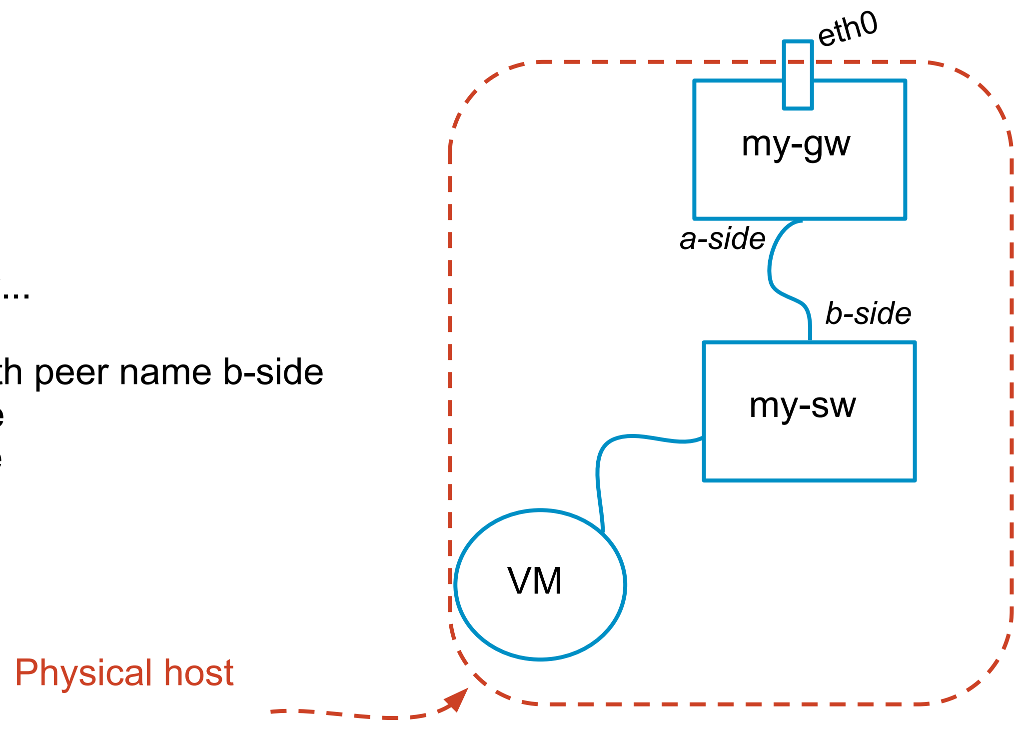
OpenVswitch primer

- A virtual switch that supports OpenFlow.
- Create bridges (switches), connect VM's.
- Create virtual networks.

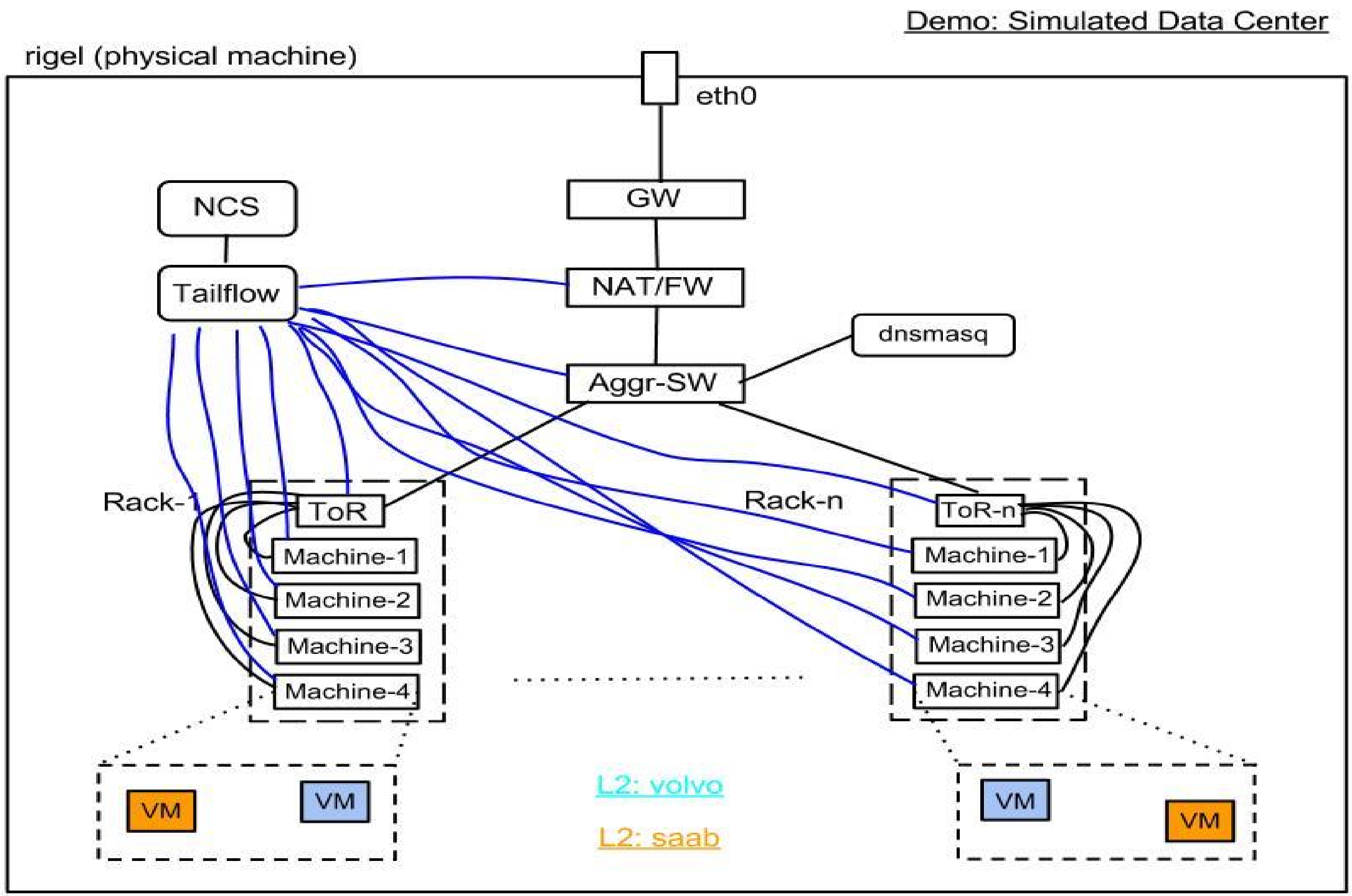
```
ovs-vsctl add-br my-gw
ovs-vsctl add-port my-gw eth0

ovs-vsctl add-br my-sw
...start VM, connected to my-sw...

ip link add name a-side type veth peer name b-side
ovs-vsctl add-port my-gw a-side
ovs-vsctl add-port my-sw b-side
```



Controlling a simulated Data Center

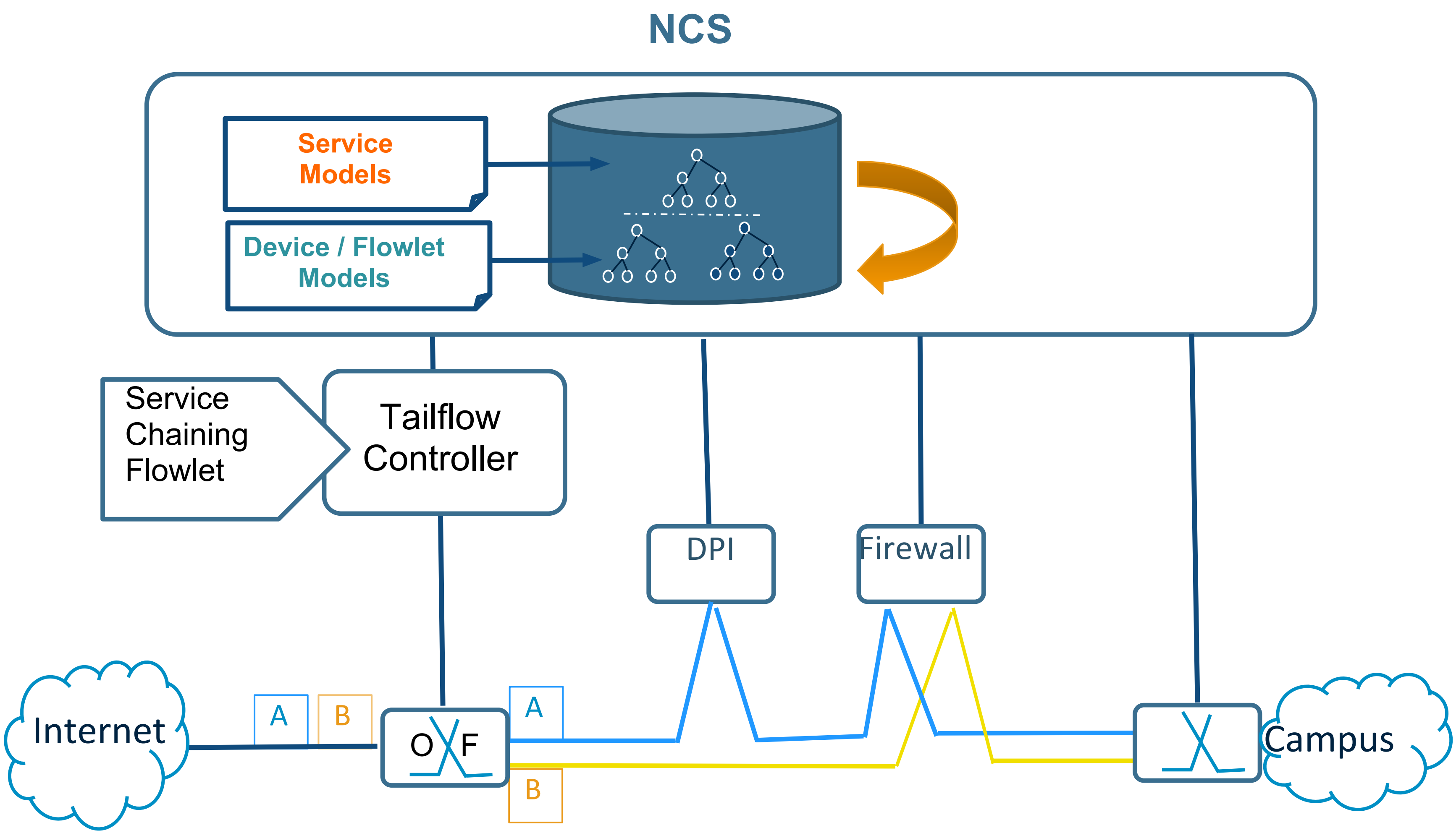


Realistic scenario?

The previous example, implementing a NAT/FW is not very realistic perhaps...

So let's finish with a more realistic use case scenario for Openflow: *Service Chaining*

SDN Use Case: Service Chaining



Thank you for listening!

Questions?