1 Software defined networking

1.1 Networking devices

A typical network device (e.g routers, firewalls...) is usually composed by:

1. A data plane
2. A control plane

The data plane is used to handle individual data packets locally. For example the data plane in a router is used to determine to which interface a packet should be forwarded to. The data plane usually applies a large amount of small, fast operations.
The control plane is used to handle control messages from other devices in the network, and allows the network device to work from a network wide perspective.
The messages from other devices may be used to configure policies which are then applied by the data plane (e.g the control plane receives a policy that determines which packets should be filtered, while the data place does the actual filtering).
Compared to the data plane, the control plane is used for more complex operations that usually require coordination along network devices.

1.2 Definition

The idea behind software defined networking (SDN) is to de-couple the control plane from the data plane, not only on a logical level but also physically.

The control plane is no longer part of the network devices and is instead moved to a centralized software component called a *controller*.

The basic architecture of a SDN is composed by the controller and by a series of network devices which in this context are called *switches*.

A SDN controller

1. Is able to communicate with the switches
2. Can be programmed: it is now possible to write software that defines how the whole network behaves.
   The switches keep executing their data plane locally, according to the policies received by the controller.
3. Is able to acquire information about the network it is managing, e.g it can discover the topology of the network to know where the switches are places and how they are connected.

The goal of the controller is to communicate with all the switches to inject into them the desired network behavior

The controller is logically placed between the network infrastructure and the application layer.

Using a series of high level programmable interfaces (also called *APIs* A*pplication P*rogrammable *I*interfaces) the controller is able to communicate with its neighboring layers.

Using the APIs connected with to the application layer (the *northbound*)interfaces various network devices can be implemented, e.g a network balancer, a firewall.

The communication between the controller and the switches is made possibile by a series of *southbound* interfaces.

Using these interfaces the controller can effectively apply the policies to the network, e.g it can send traffic rules to the switches connected to the controller.

Inside each switch there is a set of *generic tables* (also called *flow tables*) that are used to store the configuration sent by the southbound interfaces.

The southbound interfaces can also be used in "reverse": the switches are able to communicate with the controller using a series of predefined messages.

Estendere con paradigma match + action?

1.3 Openflow

Despite being relatively new, software defined networking became a popular approach for organizing new networks (e.g in 5G networks) Compared to other widespread network technologies such as the TCP/IP stack, software defined networking is a relatively new approach: the first paper defining protocol for SDN (the *Openflow Protocol*) was published in 2008.

The OpenFlow protocol defines a sets of API for remote communication with network plane devices, i.e. a southbound interface.

SDN switches inside the context of OpenFlow are also called *generic* switches; This is to avoid confusion with the usual meaning of the term switch, which usually refers to a device used to manage level 2 functionalities (i.e. the data link level, used to handle the physical addressing with MAC addresses). Openflow switches are able to interact up to the fourth layer (i.e. the transport layer, which may include protocols such as TCP or UDP).

A large portion of an OpenFlow Switch is composed by functionalities related to the data plane, the only portion dedicated to the control plane is delegated to the *control channel*.

The control channel includes one or more OpenFlow channel that allows the switches to communicate with one or more controllers (multiple controllers may be used to avoid a single point of failure in the infrastructure). dd OpenFlow switches include three kind of tables, used for handling data plane functionalities:

- *Flow tables*: the most important kind of table, they can be chained in a pipeline.
  They contain the rules that are applied to the incoming and outgoing traffic.
- *Group tables*: Users of a SDN can be assigned to different group; group tables allow traffic to be sent in multicast to users of a specific group (e.g. group tables can be used to send outgoing traffic to multiple ports if necessary)
- *Meter table*: table used to collects statistic about the current network.
  The gathered data can then be used to handle QoS (e.g rate limiting for traffic directed to a specific port)

1.3.1 Configuring the tables

To work properly, the flow tables inside the switches have to be configured by the controllers in the network. The solution is usually an hybrid of two types of configuration:

1. *Proactive configuration*: When the switch is first inserted into the network, the controller proactively fills the tables.
   The entries in the tables are static and do not change across the lifespan of the switch in the network.
   This kind of approach is obviously unfit for rapid changing scenarios such as those encountered in managing networks.
2. *Reactive configuration*: Initially there no entries in the flow tables; the tables are fully configured at runtime according to the network state.
   If a switch does not known where a package should be send to, it forwards the packet to the controller.
   The controller is aware of the full topology of the network, and is then able to edit the received package to include the routing information, before sending the package back to the switch..
   This leads to increased round trip time for the first packets sent across the network.
   However, communication between the switches and controller usually represent a bottleneck for the network's performance; a large amount of packets sent to and from the controller may cause congestion in the network and limit the available bandwidth.

Usually the preferred solution is based on an hybrid approach.
A starting set of rules is used at the start up of the network, to cover the expected data flows in the network.
For any unknown data flow that may arise during the lifespan of the network new routing rules will be issued by the controller.

1.4 SD-WAN

1.4.1 WANs

WANs (*Wide Area Networks*) are typically used by businesses who need to interconnect different sites of an organization (e.g branches of a bank or point of sales of a retail store).

Businesses may decide to invest in a WAN to reach for a higher level of quality than what may be offered to a typical consumer by an ISP.

WAN connectivity, also known as *generalized connectivity* is by far more expensive than generalized connectivity (i.e the "normal" consumer internet).

The extra cost may be justifies by the guarantees a WAN can provide in terms of quality of service (e.g availabilirt, end-to end latency...).

In a WAN the local networks of the branches are connected to the core of the network via border routers; there are multiple ways of achieving this design:

1. *Dedicated physical WAN* : the business buys the full physical infrastructure that the networks is built on (the cables, the switches etc.).
   While the company is the owner of the whole WAN, the cost is extremely high and network management (e.g enforcing security) is the the company's responsibility.

2. *Leased Lines* : the business relies on an ISP to establish private circuits between the branches. (e.g branch 1 must be connected to branch 2, which must be connected to branches 3 and 4)
   The private circuit may be created by assigning a dedicated wavelength to a particular customer.
   This solution offers only partial control, but is cheaper and offsets some of the network's management issues to the ISP.

3. *MPLS networks*: The business still relies on an ISP to provide connectivity, but unlike leased lines there are no established circuits between the branches. Instead, a mesh connectivity is provided , with a certain QoS guarantee.
   For each branch a *customer edge router* is installed, allowing the branch to access the MPLS network via a *provider edge router.*
   The *provider edge router* (PE) is in between the MPLS network and the companies, one PE may manage connections for multiple customers.

4. *SD-WAN*: this kind of WANs will be discussed in detail in the next section.

1.4.2 SD-WAN

The principles of software defined networking can be applied when building a WAN; While SD-WAN were introduced recently (in 2014) their popularity increased due to the possible decrease in costs when compared to traditional WAN solutions.

SD-WAN works by stipulating multiple contracts of generalized internet connections (e.g 4G, POM . . .) and then combining the different networks to guarantee a certain quality of service level.

Like the MPLS approach, a CPE (*Customer Premises Equipment*) is installed in each branch of the business.

The CPEs is connected to the branch's LAN and to multiple generalized connectivity networks.

The traffic generated by the LAN may then travel in different networks, potentially even from different ISPs.

An SDN controller, in this specific context called an *SD-WAN Controller* is responsible to communicate to each CPE how the outgoing traffic should be managed (e.g two branches may be connected using a fiber cable, if the cable connection fails the controller will instruct the branches to communicate using a 4G network).

MPLS may still be used in a SD-WAN context, a company could decide to use other (usually cheaper) kind of connections for most use cases and reserve a MPLS connection only exceptionally.

2 Load balancing in SD-WANs

As we previously discussed, in SD-WANs a controller's task is to instruct CPEs on how they should distribute outgoing traffic among the available links.
To controller receives data from the strategy CPEs (e.g bandwidth usage) and the tries to build an optimal strategy.
While this holds true in most cases, some strategies may avoid using a controller
Some of the challenges in load balancing include:

1. Formalizing the problem: how can we define the load balancing problem in a structured away, to then be able to solve it?

2. Evaluating a strategy: what metrics should be considered when trying to find the best strategy for traffic handling? (e.g. end to end latency, packet loss . . .)

2.0.1 Definitions

This definition of the problem consider a load balancing strategy that avoids the use of a controller.

Each edge device (i.e a switch) is equipped with a load balancing agent that along with all the other agents tries to minimize the MLU (Maximum Link Utilisation).
Each load balancing agent manages a set of OD (Origin-Destination) that begins from the edge device itself. The flows will be called *tunnels* for the rest of the paper. Tunnels can be split over multiple paths (and therefore multiple links); the amount of traffic going through to each tunnel is continuously updated by agents. The measure of how the traffic is divided on the tunnels is called a *target split ratio*

2.0.2 Formalizing the problem

We define the problem on a network, which we encode with a graph:

$$G = (V, A)$$

Where:

- V is the set of nodes
- A is the set of arcs

We then define:

- $C : A \to \mathbb{R}_+$: the function that gives the capacity of each arc
- : K: the set of tunnels
- $d : K \to \mathbb{R}_+$: the function that indicates the traffic demand of each tunnel
- $P^k = \{p_0^k, \ldots, p_{|P^k|}^k\}$: the set of available paths for tunnel $k \in K$

We assume that for all $k \in K$, $|P^k| \geq 2$. If there is just one path for any given tunnel, the associated tunnel can be removed from the association problem.
The core of the problem is minimizing the maximum link utilization defined for a set A of links as:

$$MLU(A) = \max_{a \in A}\{\frac{V_a}{C_a}\}$$

Where

- V_a is the amount of traffic of link a
- C_a is the capacity of link a.

With $LU_a = \frac{V_a}{C_a}$ we will the denote the link utilization of $a \in A$.

The problem can be formulated with the following variables:

- $x_p^k \in [0, 1]$: split ratio on path $p \in P^k$ for tunnel $k \in K$
- $\theta \in \mathbb{R}_+$: maximum link utilization of A

$$\min \theta$$

$$\sum_{p \in P^k} x_p^k = 1 \qquad \forall k \in K \qquad (1)$$

$$\sum_{k \in K} d_k \sum_{p \in P^k, a \in p} x_p^k \leq \theta C_a \qquad a \in A \qquad (2)$$

$$0 \leq x_p^k \qquad \forall k \in K, \forall p \in P^k \qquad (3)$$

Constraints (1) guarantee that all traffic is split on the paths, while constraints (2) permit to compute the MLU.

2.0.3 Alternative formulation

While the previous definition allows us to define the problem in terms of link utilization, it does not rely on the use of controllers.

This paper [referenza] provides an approach to the load balancing problem in a SD-WAN context, trying to formalize the problem using linear programming.

In an OpenFlow network, an incoming flow f experiences five delays before being served.

When the first packet of a flow reaches the ingress switch s, it is queues within a local queue Q_s.
After waiting a time of T_W^S the flow is then served for a duration T_S^S. If the switch does not know the destination of the flow (i.e no there are no matching ***flow entries*** in the local flow table) a ***PacketIn*** message is sent to the controller c through a path for a duration of $R(s,c)$. The controller then receives the ***PacketIn*** message and queues the message for a duration of T_W^C, before handling it within a duration of T_S^C.
Finally, the response is sent back to the switch s using a ***PacketOut*** message or a ***FlowMod*** message through a path within a duration of $R(c,s)$.
All subsequent packets of the flow f will be then forwarded based on the flow entry already installed in the flow table of the switch s.

Let $T_f^{(s,c)}$ be the whole delay experienced by a flow f at the switch s before being served by the controller c.
The delay can be expressed as follows:

$$T_f^{(s,c)} = T_W^S + T_S^S + R(s,c) + T_W^C + T_S^C + R(c,s)$$

Our objective is to minimize the above delay.
$T_f^{(s,c)}$ consists of three parameters:

1. $T_{sj}(s) = T_W^S + T_S^S$: the switch sojourn time (i.e. how much time the packets spends in the switch s)
2. $T_{RTT}(s,c) = R(s,c) + R(c,s)$: the round trip time between the switch and the controller
3. $T_{sj}(c) = T_W^c + T_S^c$: the controller sojourn time (i.e. how much time the packets spends in the controller c)

We can then rewrite the $T_f^{(s,c)}$ delay as:

$$T_f^{(s,c)} = T_{sj}(s) + T_{RTT}(s,c) + T_{sj}(c)$$

Now that we have a cost function, we can then take a linear programming approach to the problem resolution. Let S_{c_j} be the set of switches managed by the controller c_j, where $c_j \in C$ and $S_{c_j} \subseteq S$. The main goal is to find an assignment between the switches (S) and the controllers (C) that minimizes the cost function $T_f^{(s,c)}$.

$$\begin{cases} \text{min} & \sum_{c_j \in C} \sum_{s_i \in S} T^{(s_i,c_j)} \\ \text{subject to:} & \bigcup_{c_j \in C} S_{c_j} = S \end{cases} \tag{4}$$

The constraint in (4) states that each ingress switch must be assigned to one controller. However, a controller may serve more than one ingress switch, but it also may not support any new ingress switch when it is overloaded.
The assignment of switches to controller should be dynamic and should consider at any given time:

1. The current load on a controller
2. The current load on a switch
3. The round trip time between them

The set of controllers involved in the assignment process should be dynamically updated to exclude the overloaded controllers.
The mapping problem between controllers and switches can be resolved using linear programming, as it will be discussed in greater detail in the following section.

2.1 Linear Minimum Cost Bipartite Assignment Programming

Given the two sets S and C and the cost function $T_f^{(s,c)}$, we formulate the minimum cost bipartite assignment problem as follows:

$$\begin{cases} \text{min} & \sum_{c_j \in C} \sum_{s_i \in S} T^{(s_i,c_j)} \\ \text{subject to:} & \sum_{c_j \in C} x_{ij} = 1 \ \forall s_i \in S \\ & x_{ij} \geq 0 \end{cases} \tag{5}$$

Where x_{ij} represents the assignment of the switch s_i to the controller c_j , taking value 1 if the assignment is done and 0 otherwise.

The paper [https://link.springer.com/article/10.1007/s10922-020-09523-2] goes into details on how both the switch sojourn time and the controller sojourn do does not depend on x_{ij} (the fact that a particular switch is assigned or not to a controller does not have an impact on the switch nor the controller sojourn time).

Therefore we can rewrite the formulation of the problem including only the round trip time.

$$\begin{cases} \text{min} & \sum_{c_j \in C} \sum_{s_i \in S} T_{RTT}^{(s_i,c_j)} x_{ij} \\ \text{subject to:} & \sum_{c_j \in C} x_{ij} = 1 \ \forall s_i \in S \\ & x_{ij} \geq 0 \end{cases} \tag{6}$$