but I want U, V, W, Z. So how can I force IP control plane to find a path as the shortest one?
We can play with the links weights, for instance we can increase a lot cost ~~[illegible]~~ UX and UW

How can I go X, W, Y, Z? there are a set of algorithms that do this job. They are centralized algorithms. ~~[illegible]~~ (WEIGHT OPTIMIZATION ALGORITHM) It takes as input a graph $G(N,L)$, $\Delta(f_1, f_2 \dots f_n)$, Link capacity
graph demand
and it returns as output the weight W that optimize this function which is that that minimize link congestion
Now we can configure in the router this set of weights W ~~[illegible]~~ and get the routing that we want.
HOWEVER: this algorithm is heavy, we need to run it offline using a simple server, and it configures the weights

So once I do that the IP control plane will take the decision in a distributed manner and it will end up finding the exact path that I want, but in the meanwhile, the traffic demand could change, so we could to it once more.
So it is quite hard to do that using the distributed control plane.
On the other hand if I have the central controll we can simply monitor the infrastructure using the openflow protocol and always optimize the routing according to the current demand.

---

2) Another problem is if we want to split the traffic from U to V and X.
This is not something that can be done with IP control plane.

---

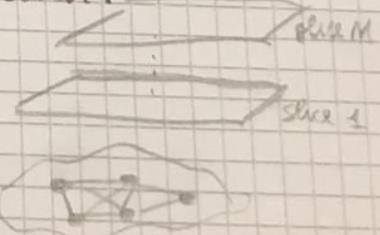
3) Another example is NETWORK SLICING (it is the way to differentiate how the traffic is handled) (It is one of the basis of 5G architecture).

5G has a set of requirements:

- Massive machine type communication MMTC
(and this intended to ~~[illegible]~~ provide connectivity for IOT networks and to provide machine-machine communications. It is used to realize smart environments)
- Ultra Reliable and Low Latency communication URLLC
(this is hard to realize because the connection is wireless, so there could be obstacles along the path).
It is used for example for remote control of drones or robots
- Enanced Mobile BroadBand .. MBB
(provides a lot of data rate speed to inbound and outbound traffic)

These three requirements represent 3 different objective functions and they are almost impossible to get all together.

So we have the physical infrastructure and a set of virtual layers called slices (as many as we want).



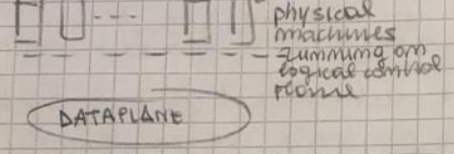
i need to serve different applications in different ways

SDN

control plane and data plane are PHYSICALLY SEPARATED.
In order to to keep control peane and data peane ~~separated~~ communicate I need to connect them another network infrastructure
(It can also by directly the data peane infrastructure through a link connecting one of the switches to the controller OR it can be a dedicated infrastructure) parallel to the data peane infrastructure

The remote controller represents the network operating sistem
It is a simple logical entity that can be actually mapped on top of different physical machines.

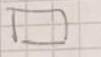
FIRST POSSIBILITY

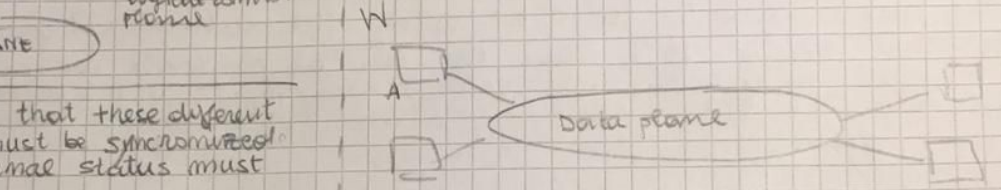


This means that these different machines must be synchronized (their internal status must be equal)
I can use them all together or I can have an active one and a backup one

OTHER

If the data plane is huge (example is a network that goes across USA) and we have SDN switches on the west cost and on east cost

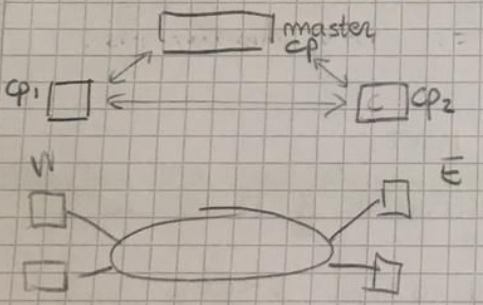




Now controller that A receives a packet but doesn't know what to do, so it will send it to control peane. But control plane could be in the middle of USA, so we have huge LATENCY
So a solution could be to create different regions inside the dataplane, and I can create two different control plane entities and a master control peane (CP)
These entities need to communicate (because for example if I have a packet that goes from W → E then we will use the control of different control entities.

In this way we reduce LATENCY

N.B. these control data peane are logical so we can implement them using multiple physical machines.



Above remote controller we have applications.

Red arrows in figure 22 that connect remote controller to switches in data plane are dataplane - control peane interactions that will be realized through the openflow protocol. And in ~~order~~ order to enable openflow protocol in the devices they must have the control agent interface (they can speak the open-flow language).

There is a north-bound interface allowing controller to communicate with applications.

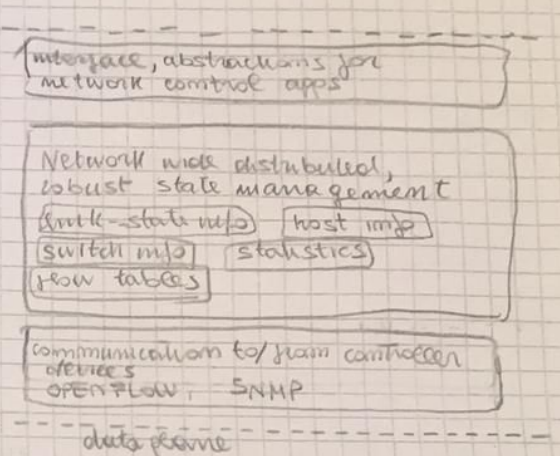
COMPONENTS of SDN controller

Internal architecture of an SDN controller (This is a logical view)

SDN controller is a software running inside the general purpose machine (it could be a very powerful one such as server machines, or simple hosts such as a laptop)

Internal every controller is organized in a different way, but the general scheme is this one →

All controller are divided into 3 layers: south bound interface, north bound interface and network stuff.



South interface: the most common protocol is OpenFlow.
(In IP-networks SNMP protocol that can also be used in SDN context, in IP is used by a central monitoring system that checks the health status of a network. But it can be used in SDN to interact with the data plane.)

central part: it is where the main status informations of the network are stored (link-state info, host info, switch info, flow tables, statistics)
It does know nothing about the situation on the data plane, so the way it fulfil those tables is by continuously asking to the nodes in data plane for informations.

north interface: controller builds an abstraction to interconnect the applications with the data plane. It provides a set of interfaces to talk with the controller itself (most common one is RESTful API, Intent, network graph (used for routing applications))

OPENFLOW PROTOCOL

It is a protocol that ~~controls~~ implements control - data plane interaction
OPENFLOW IS NOT SDN! it is just the south-bound interface

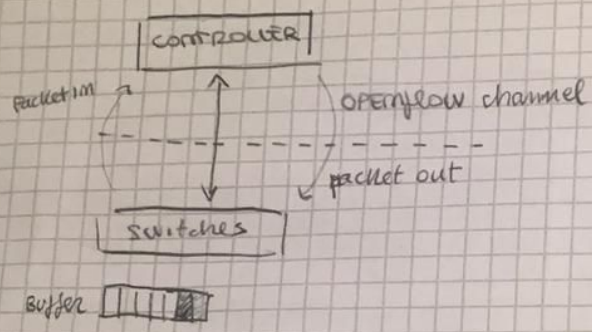
Control msg that go from control to switches and vice versa are ~~very~~ important because are control msgs. If one of these is lost, it is a problem. In order to ~~not~~ ensure the correct delivery of all msgs openflow is run inside a TCP connection. So there is a TCP connection between every switch in the data plane to the controller. We can also ~~encrypt~~ encrypt the information, authentication and so on.

There are 3 Types of msgs:

- CONTROLLER-TO-SWITCH msgs: all msgs that goes from the controller to SDN switches
- ASYNCHRONOUS msgs: all msgs from switches to controller ~~(they are asynchronous)~~
- SYMMETRIC msgs: keep-alive msg. They are sent periodically

Why asynchronous msg are called that way? →

Why are asynchronous msgs called this way?
SDN is designed to work in a reactive way



Consider that a new packet arrives ~~if the~~ the switches does not know what to do because the ~~entire~~ flow table is ~~empty~~ empty. This generates an EVENT in the data plane and the event is the FLOW RULE MISSING.
So the switch sends the packet to the controller using a specific openflow msg called PACKET-IN that contains the entire packet. (or to reduce latency we can send only a portion of the packet. The end switch has a buffer where incoming packets are stored that are waiting for a decision of the controller. The position inside the buffer is called buffer_ID. So I will send to the controller only buffer-id)
Then the controller sends its response through PACKET-OUT. In case only a part of the packet was sent, the controller specifies also the buffer-id.

However, if we do this thing for every new packet belonging to the same flow, we will have many flow rule missing and there will be a lot of control plane - data plane interactions that could create congestions to applications.
So to prevent that the controller uses FLOW-MODE that implies the creation of a flow rule inside the SDN switch. Each flow rule has ~~also~~ also a set of time out: IDLE_TIMEOUT and HARD_TIMEOUT.
If The IDLE expires, the flow rule must be removed from the flow table as well as HARD one.
The idle is the time passed since the last time I used the rule, on the other hand hand is the time passed since I have installed the rule.
So if I am not using so often the rule the idle can ~~expire~~ expire and I will remove that. If I have a too old rule (even though I am using it a lot) hand one will expire and I will delete the rule.
This mechanism is used to prevent the saturation of the flow table of the switch.

So, why msgs are called asynchronous?
While msgs from the controller are sent periodically, the msgs from the switch to the controller are generated by events. And no body knows when an event will happen.

CONTROLLER TO SWITCH MESSAGES

SDN control/data plane interaction EXAMPLE

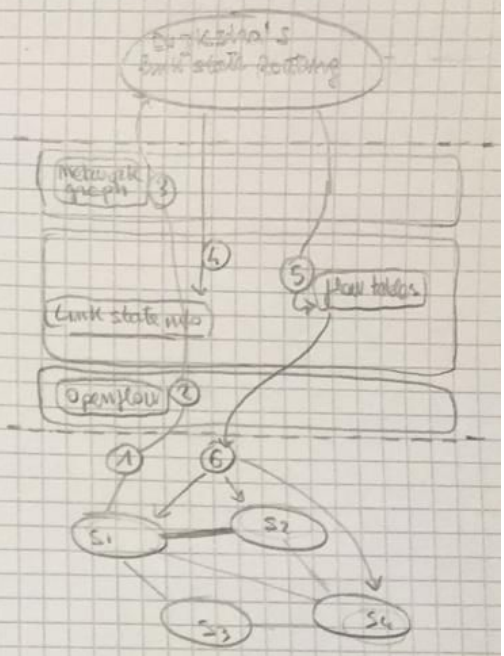
It represents classical link state routing of IP over an SDN network.
Let's assume that switches have been proactively configured with a set of flow rules.
Anyway, a new event happen.
So the link fails.

So we need to calculate a new set of routing path (because — is not working anymore so all flows that are routed on this link cannot correctly delivered to the destination. So we need to update the routing.

The first thing is that since the event occurs in the data plane, The switch one generates an asynchronous msg to the controller. It notifies to OpenFlow interface.

The information is updated in the link-state info.

Then the controller processes again the data and produces a new model for the network graph (that will be the graph minus the link —)



Then the application is asked and says to recompute the graph using Dijkstra's algorithm.
Once it gets the path, changes it updates the flow table. The controller writes the set of flow rules and then it notify the set of new flow rules to every single switch.