**18ECE320T - Software Defined Networks**

**UNIT 5**

**Forking:**

Forking is to take the [source code](https://www.techtarget.com/searchapparchitecture/definition/source-code) from an open source software program and develop an entirely new program. Forking is often the result of a deadlock in an open source project that is so insurmountable that all work stops. When a new code base is forked from an existing release, it cuts the connection back to the trunk. This means that the continuing innovation on that fork will not be contributed back to the original code base.

**Open Source Licensing Issues**

open source means it is source code provided without charge and that you may use and modify for a defined set of purposes. There is open source available that only allows non-commercial use.

Even when commercial use is allowed, however, there are significant differences in the obligations implicit in the use of such software.

Some major categories of open source licensing in widespread commercial use today are the GNU General Public License (GPL), BSD style, and Apache.

There are different organizations that opine on whether a given license is truly open source.

These groups include the Free Software Foundation (FSF) and the Open Source Initiative. They do not always agree.

**GPL**

* GPL is an **extremely popular** licensing form. The **Linux operating system** is distributed under the GPL license. It allows users to copy and modify the software for virtually any purpose.
* For many companies, though, GPL is too restrictive in one major facet. It incorporates the notion of **copyleft**, whereby if any derivative works are distributed, they must be distributed under the same license.
* If a commercial company extends a GPL open source code base to provide some new function that it wants to offer to the market, the modifications that are made must be made **freely available to the open source community** under GPL terms.
* If the function provided by the GPL open source can be used with little added intellectual property, GPL code may be suitable in **commercial products**. Ensuring that this distinction is kept clear can have massive consequences on the valuation that a startup receives from investors. For this reason, GPL open source is **used with caution** in industry.
* Nonetheless, GPL remains **the most widely used** free software license. There is a **GPL V.2** and **GPL V.3** that are somewhat more restrictive than the initial version.

**BSD**

* The BSD-family licensing model is more permissive. This style of license was first used for the Berkeley Software Distribution (BSD), a Unix-like operating system.
* There have been a number of variants of this original license since its first use; hence we use here the term BSD family of licenses.
* Unlike GPL code, the user is under no obligation to contribute derivative works back to the open source community.
* This makes it much less risky to incorporate BSD-licensed open source into commercial products.
* In some versions of this style of license, the advertising clause originally required to acknowledge the use of U.C. Berkeley code in any advertisement of the product.
* Although this requirement has been eliminated in the latest versions of the BSD license, there are still many free software products licensed with a similar restriction.
* An example is the widely used OpenSSL encryption library, which requires acknowledgment of the author in product advertising.

**Apache**

* The Apache license is another permissive license form for open source.
* The license was developed by the Apache Software Foundation (ASF) and is used for software produced by the ASF as well as some non-ASF entities.
* Software provided under the Apache license is unrestricted in use except for the requirement that the copyright notice and disclaimer be maintained in the source code itself.
* Because this restriction has minimal impact on potential commercialization of such software, many commercial entities build products freely incorporating Apache-licensed source code today.

**EPL**

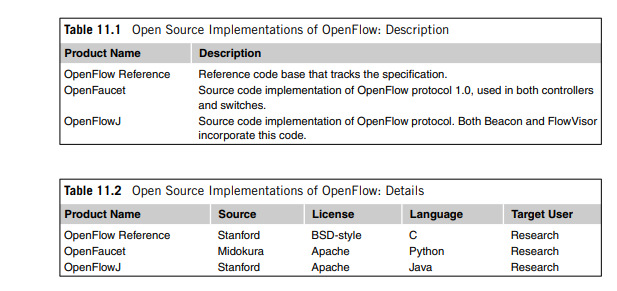
* The Eclipse Public License (EPL) can be viewed as a compromise between the strong copyleft aspects of the GPL and the commercially friendly Apache license.
* Under certain circumstances, using EPL-licensed source code can result in an obligation to contribute one’s own modifications back to the open source community.
* This is not a universal requirement, though, so developers can often incorporate this software into commercial products without risk of being forced to expose their own intellectual property to the entire open source community.
* The requirement comes into effect when the modifications are not a separate software module but contain a significant part of the EPL-licensed original and are thus a derivative work.
* Even when the requirement to release extensions to the original is not in effect, the user is required to grant a license to anyone receiving their modification to any patent the user may hold over those modifications.

**FOSS**

* The Stanford University Free and Open Source Software (FOSS) license is noteworthy in the manner in which it attempts to address the common problem with commercial use of GPL-licensed source code due to its copyleft provision.
* Stanford’s motivation in developing this concept was to avoid the situation where developers would hesitate to develop applications for the Beacon controller.
* Because of the way in which applications interface with the Beacon controller, if it were licensed under GPL without the FOSS exception, such developers would be obliged to contribute their work as GPL open source.
* This license is specific to the Beacon controller, though the concept can be applied more generally.
* In essence, Stanford wants to release the Beacon controller code under the terms of GPL.
* FOSS specifies that a user of the code must continue to treat the Beacon code itself under the terms of GPL, but the code they write as a Beacon application can be released under any of a number of approved FOSS licenses.

**open source implementations of OpenFlow**

* There are multiple open source implementations of OpenFlow
* Table 1 provide a basic description of available implementations;
* Table 2 list detailed information regarding the provider of the code, the type of license, the source code in which it is written (where relevant), and the likely class(es) of users.
* We have assigned these target classes of users using a number of criteria, including: (1) the nature of the license, (2) the maturity of the code, and (3) actual users, when known.
* OpenFlowJ is notable in that the Beacon controller, itself the fount of a number of other controller implementations, uses the OpenFlowJ code.
* In some cases the license or language may not be known. We indicate this by placing a hyphen in the corresponding table cell.
* We also mark the table cell with a hyphen for languages in the event that the project uses multiple languages

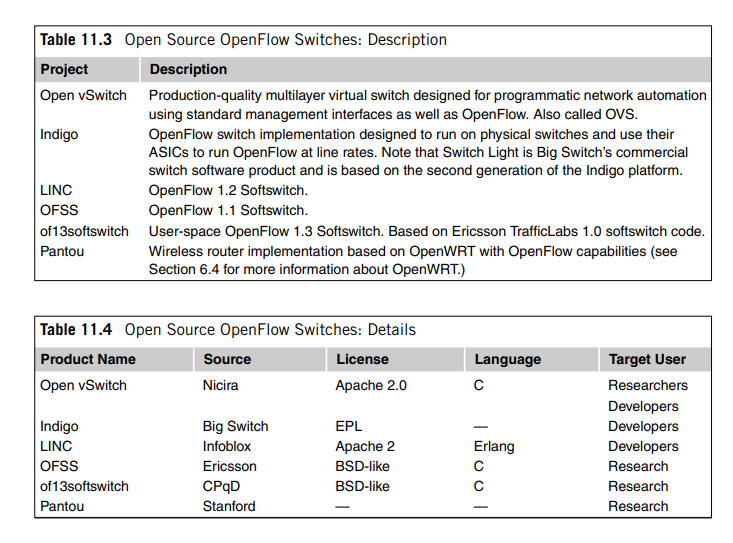


**Open Source Switch Implementations**

The open source switch implementations can potentially apply both to OpenFlow hardware device implementations as well as software-based virtual switches. Two most widely discussed switch implementations, OVS and Indigo, as well as the only OpenFlow wireless access point code base listed, Pantou.

**OVS**

OVS is the most mature and feature-complete open source implementation of the switch side of an OpenFlow implementation. OVS has been used as the control logic inside genuine hardware switches as well as a virtual switch running within a hypervisor to implement network virtualization. OVS uses OpenFlow for control of flows and OVSDB for configuration. OVS uses the sFlow protocol for packet sampling and monitoring. The code has been ported to multiple switching chipsets and has thus found a home in several hardware switches as well.



**Big Switch Networks** offers the **Indigo switch** code base under the Eclipse public license.

Like OVS, Indigo is targeted for use both on physical switches as well as an OpenFlow hypervisor switch for network virtualization environments. In particular, this code can be used to convert a legacy layer two or three switch into an OpenFlow switch. By making this code base available to switch manufacturers, especially smaller manufacturers that may lack the means to develop their own OpenFlow switch-side control code, Big Switch hopes to expand the market for its controller.Because Indigo is integrated with Ethernet switch ASICs, it is possible to switch flows under the OpenFlow paradigm at line rates.

**Distinction between OVS and Indigo**

* Indigo is implemented specifically to support OpenFlow, whereas OVS can support other control mechanisms such as OVSDB.
* In addition, OVS has broader support for affiliated network virtualization features than Indigo and includes richer contributions from the open source community.
* Nevertheless, after OVS, Indigo remains a more likely candidate for an open source starting point for a commercial OpenFlow physical switch than the alternatives.
* From a feature-set point of view, OVS seems to be a superset of Indigo’s features

**Pantou** is actually designed to turn commodity APs into an OpenFlow-enabled AP. This is accomplished by integrating the NOX OpenFlow implementation with OpenWRT. OpenWRT is an open source project that applies the SDN principle of opening up the device to a wide selection of low-cost consumer AP hardware. This union of OpenFlow with basic wireless AP functionality available on low-cost hardware is a boon to researchers who want to experiment with the largely virgin field of SDN applied to 802.11 networks.

**Controller Implementations**

**Beacon**

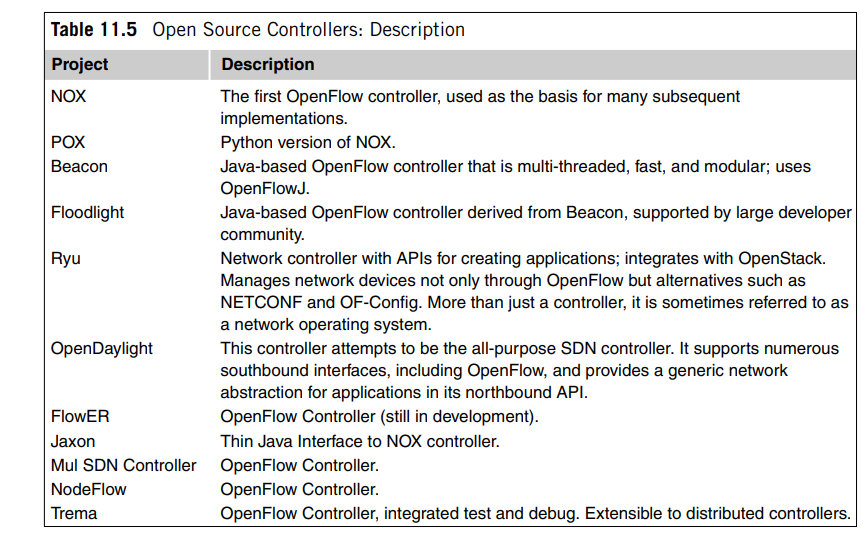
* The Beacon controller was based on OpenFlowJ, an early open source implementation of OpenFlow in Java.
* The OpenFlowJ project is hosted at Stanford University.
* Beacon is a highly influential controller, both for the large amount of early OpenFlow research and development that was done on that controller as well as being the code base from which the Floodlight controller source code was forked.
* Beacon is a cross-platform, modular OpenFlow controller.
* Beacon has been successfully deployed in an experimental data center consisting of 20 physical switches and 100 virtual switches.
* It runs on many different platforms, including high-end Linux servers.
* Though the core controller code is protected with GPL-like terms, the FOSS license exception allows developers to extend the controller with applications that may be licensed under more commercially favorable terms.
* Beacon’s stability distinguishes it from other controllers used primarily for research purposes.
* Modules of Beacon can be started, stopped, and even installed while the other components of Beacon continue operating.
* Evidence of the impact that this code base has had on the OpenFlow controller industry is found in the names of the two most influential open source controllers being used by industry today, Floodlight and Open Daylight, both echoing the illumination connotation of the name Beacon.

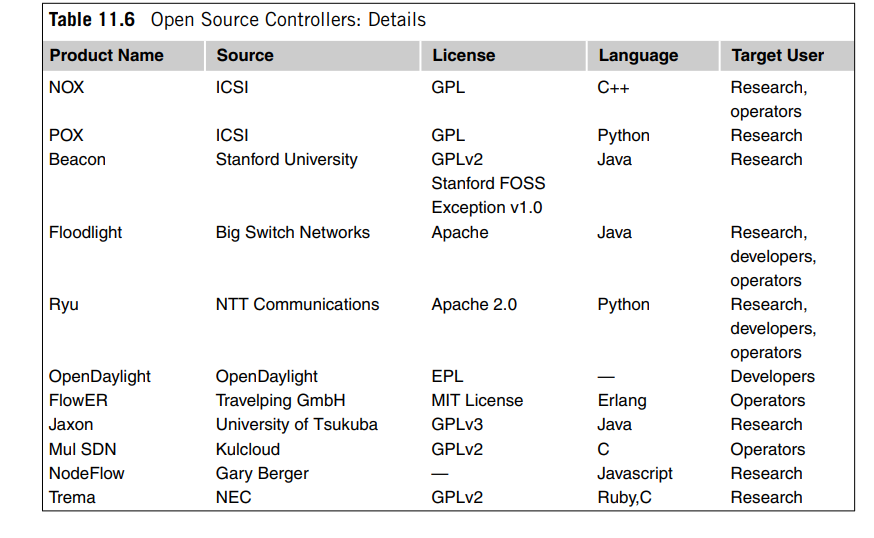
**Floodlight**

* Floodlight was forked from Beacon prior to Beacon being offered under the current GPL/FOSS licensing model .
* Because Floodlight itself is licensed under the Apache 2.0 license, it is not subject to the copyleft provisions of the Beacon license, and the source is thus more likely to be used by recipients of the code who need to protect the intellectual property of their derivative works on the code base.
* Floodlight is also integrated with OpenStack.
* A number of commercial OpenFlow controllers are being developed using Floodlight as the starting point.
* This includes Big Switch’s own commercial controller, Big Network Controller.
* Big Switch’s business strategy surrounding the open source controller is complex. The company offers a family of applications that work with the Floodlight controller.
* Though its commercial controller will not be open source, the company promises to maintain compatibility at the interface level between the commercial and open source versions.
* Big Switch joined the Open Daylight project, donating its Floodlight controller to the effort in hopes that this action would further propagate Floodlight.

**Open Daylght**

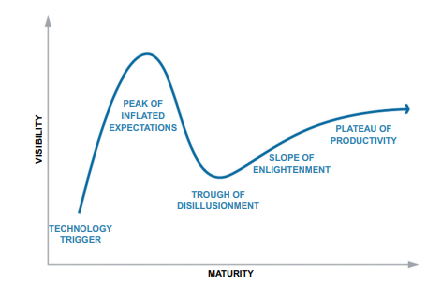
* The OpenDaylight project was formed in early 2013 to provide an open source SDN framework.
* The platinum-level sponsors included Cisco, Brocade, Ericsson, Citrix, Microsoft, and Big Switch Networks.
* Unlike the ONF, OpenDaylight only considers OpenFlow to be one of many alternatives to provide software control of networking gear.
* Big Switch initially joined and donated its Floodlight controller to OpenDaylight, anticipating that Floodlight would become the key OpenFlow component of the project.
* This expectation was quickly shattered as OpenDaylight decided to make the Cisco Open Networking Environment (ONE) controller the centerpiece and to merely add Floodlight technology to that core. (Cisco now markets a commercial version of the ONE controller as the Extensible Network Controller, or XNC.)
* Subsequent to this, Big Switch reduced its involvement in the OpenDaylight project and ultimately withdrew completely.
* The XNC controller may use a variety of methods to control network devices.
* It is thus noteworthy that both SDN via APIs and Open SDN are present in the same XNC controller.
* The facts that Big Switch stepped back from its commitment to OpenDaylight, there is a total lack of coordination between the ONF and OpenDaylight, and the platinum sponsors are almost all incumbent network equipment manufacturers have all led to a growing skepticism about the compatibility of OpenDaylight with the ONF.





**Gartner Hype Cycle**

* The turbulence that has surrounded the SDN movement during these early years has closely tracked the well-known **Gartner Hype Cycle**. Hype Cycle predicts that disillusionment is followed by the slope of enlightenment and then by the plateau of productivity. SDN to transition from the slope of enlightenment to the plateau of productivity, an entirely new generation of switching ASICs that have been purposely built with SDN and OpenFlow in mind.



**Potential Novel Applications of Open SDN**

**Managing Nontraditional Physical Layer Links:**

There is growing interest in using OpenFlow to control devices that have flows but are not traditional packet switches. The two most promising areas involve flows over optical and wireless links.A use case in which SDN was used for offloading elephant flows onto optical devices. This represents one example of the general area of using OpenFlow to manage flows over physical layers that are outside the domain of classical LANs and WANs.Two startups, Plexxi and Vello, that have brought products to market that marry optical switching with OpenFlow.OpenFlow is proposed as a mechanism to segregrate traffic from different providers and different types into separate flows that are then transmitted over a single shared wireless backhaul medium.

**Applying Programming Techniques to Networks**

The greater network programmability inherent in OpenFlow provides benefits in the richness of policies and in the fine-grained control it offers the network programmer.Thus, part of the ultimate success of OpenFlow is the development of tools that will aid in detecting such programming flaws.Open SDN provides a proper abstraction of the network that allows us to address networking challenges more efficiently.Network abstraction allows other advanced programming methodologies to be applied, including debuggers, analysis tools, network simulation, verification tools, and others.

**Security Applications**

**Hiding IP Addresses :** Many network attacks are based on identifying active IP addresses in a targeted domain. Protecting hosts by making it impossible for an attacker to identify a host’s IP address would be an effective countermeasure.Each protected host be assigned a virtual IP address that is the one exposed to the outside world by DNS lookups. In this method, the OpenFlow controller randomly and at high frequency assigns the virtual IP address to the protected hosts, maintaining the temporary mapping of virtual to physical IP addresses. The translation from the virtual IP address to the physical IP address happens at an OpenFlow switch immediately adjacent to the protected hosts. The unpredictability and speed of the virtual IP address mutation is key to thwarting the attacker’s knowledge of the network and the planning of the attacks.

**Roaming in Mobile Networks**

Mobile Traffic Offload

Mobile offload means moving a client mobile node (MN) from one RAN to another. This might make sense for a number of reasons, but the one most often offered is to shunt the traffic to a RAN where the spectrum is more available or less expensive than the one currently used by the MN. Such offloading has been contemplated for some time by mobile operators, but existing approaches have not provided the flexible, fine-grained control offered by Open SDN.For example, if a user is currently connected to a heavily loaded WiFi hotspot and located within a lightly loaded LTE cell, it may make sense to do reverse offload and move the user from WiFi back to LTE. Such a decision hinges on observing rapidly changing conditions and could not be put into effect based on static policies.

**Traffic Engineering in Mobile Networks**

Dynamic Assignment of Flows to Fluctuating Backhaul Links

A downside of wireless backhaul is that the bandwidth of the wireless backhaul is both more limited and, more important, less stable than in its wired counterparts. Current resource management practices are static and do not redirect load dynamically based on short-term fluctuations in wireless capacity.

OpenFlow controller be enabled to be aware of the current available bandwidth on the set of wireless links it is managing. It may be managing a hybrid set of wired and wireless backhaul links.

If OpenFlow is responsible for assigning user flows to that set of backhaul links, that assignment can be made as a function of the SLAs specific to each flow. High SLA (higher guarantees) traffic can be steered over wired links, and low SLA traffic can be steered over a wireless link.If one wireless link is experiencing temporary spectrum perturbation, OpenFlow can shunt the traffic over a currently stable wireless link.

**Energy Savings**

**ElasticTree**

One approach to applying OpenFlow to energy savings in the data center, called ElasticTree.

Data centers’ networking infrastructure of links and switches is designed to handle an anticipated peak load and is consuming more power than necessary during normal operation.

If a means could be found to only power the minimum necessary subset of switches at any moment, there is an opportunity for significant energy savings.

During periods of less than peak load, OpenFlow can be used to shunt traffic around switches that are candidates for being powered off.

Such a system assumes an out-of-band mechanism whereby switches may be powered on and off via the OpenFlow application.

Such systems exist and are readily integrated with an OpenFlow application.

By varying the number of powered-on switches, their system can provide the ability to fine-tune between energy efficiency, performance, and fault tolerance.

**SDN and NFV Mobile Network Architectures**

In the next generation 5G mobile core network, there are two realization options, either SDN based or NFV based.

**SDN Based Deployment**

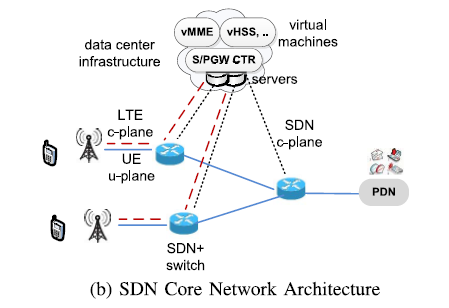
Here the control plane mobile core functions run as VNFs while the gateway functions, i.e., SGW and PGW, are decoupled into SDN controllers and special purpose SDN+ switches, as shown in Fig. The SDN controllers, deployed at the data center infrastructure, configure the SDN+ switches which handle the data plane traffic. The controllers implement the control plane of the core network gateway functions.

SDN controllers are required to handle the LTE control plane signaling procedures which are defined by the 3GPP standard, i.e., exchange of signaling messages with the radio access network in order to support the user’s attachment to the mobile network or user’s mobility

According to the signaling procedures, the controllers are responsible to configure the data plane, i.e., SDN+ switches, via the SDN API used by the operator. Additionally, the controllers need to collect the data usage of each user from the data plane switches for the purpose of charging and accounting.

On the other hand, the SDN+ switches implement the gateway data plane functions. One important data plane function needed at the SDN+ switches is GTP tunneling which is used to identify data plane traffic of users.

The SDN+ switches monitor the data plane statistics for charging and accounting. Additionally, the SDN+ switches need to support the configuration of quality of service classes that can be assigned to users.



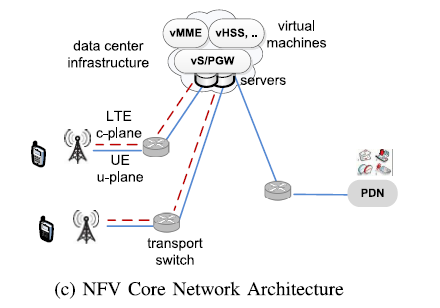
**NFV based deployment**

The control plane mobile core functions as well as the gateway functions, i.e., SGW and PGW, run as VNFs (vS/PGW) on commodity hardware at data centers. This means that the gateway’s control plane as well as the data plane processing is running on commodity servers in the cloud. The data plane processing on commodity servers can be accelerated by solutions such as Intel DPDKthe legacy core network hardware would be replaced by simple forwarding switches, i.e., transport switches, that forward both the data plane and control plane traffic between the radio access network, the data centers and the external network, as illustrated in Fig.

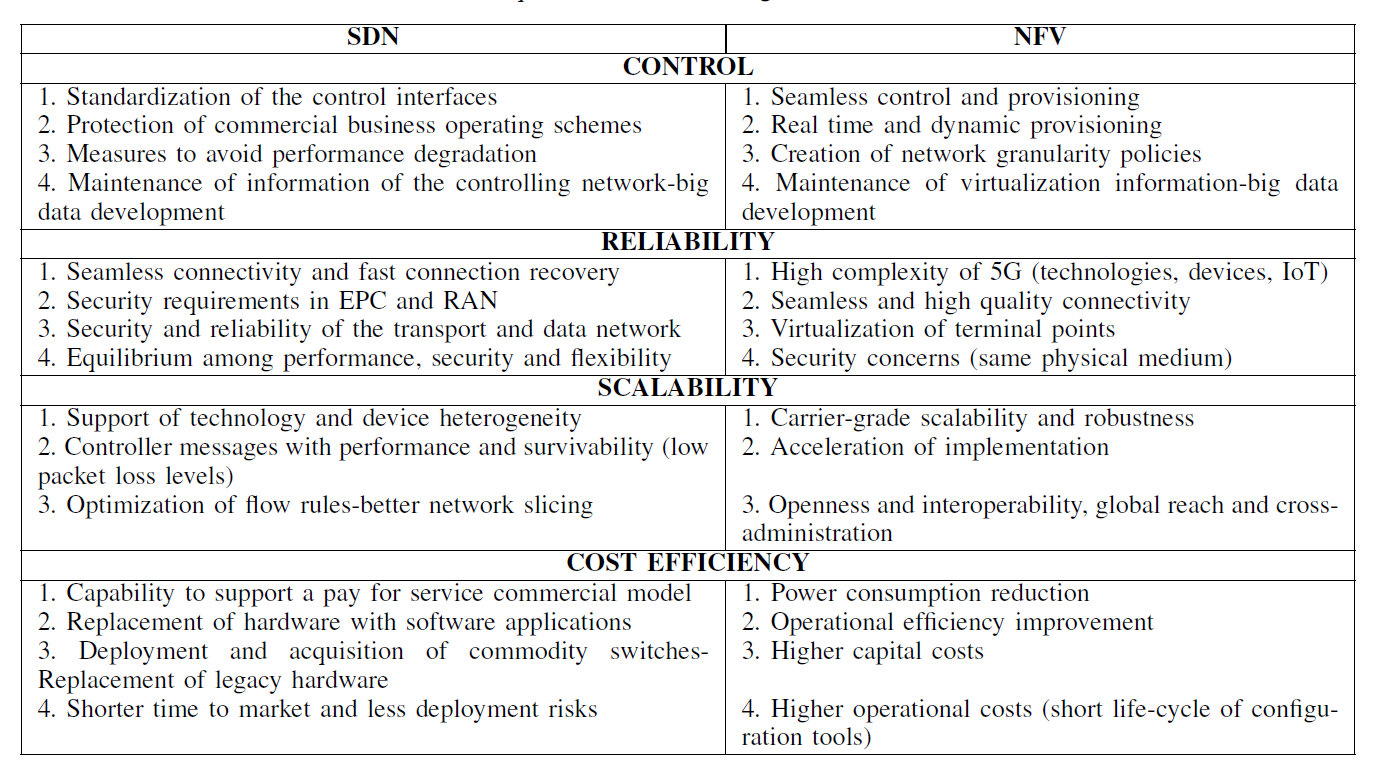
Note that in this architecture, all mobile core network functions are

migrated to software running on commodity servers and are fully independent from hardware, i.e., functions which handle control plane only, e.g., MME, and functions that handle both data as well control plane, e.g., SGW and PGW.

This implies that there is no processing, i.e., function, implemented on the forwarding switches of the mobile core network.

****

**The main requirements and challenges linked to the SDN and NFV**

****