If an SDN switch loses the connection to its controller, what are the possible options for how the switch behaves?

An SDN switch can be programmed with Fail Secure or/and Fail Standalone modes.

Fail Secure mode continues to forward packets but drops all messages for the controller and Fail Standalone mode continues to operate in native switch or router mode and OpenFlow pipeline processing stops.

What are the advantages and disadvantages of a RESTful interface to an SDN controller?

REST is very popular and widely known. It is easy to program in any major language. Data is not tied to resources or methods like RMI. Can transfer data in various formats like JSON or XML. REST on HTTP cannot pre-emptively send data and is limited to one session between client and server. This changes with SPDY and HTTP2.0.

What is ‘Intent-Based Networking’?

Intent-based networking is a concept that aims to replace the manual configuration of networks with a system where administrators provide the controller with the outcome they want, and the network achieves that state by itself through artificial intelligence and machine learning.

>>> Also: Intent-based Networking is an attempt to standardise the northbound controller interface.

In the context of SDN, what are the differences between a software switch and a hardware switch?

A software switch runs on general-purpose hardware while a hardware switch runs on purpose-built hardware. Hardware switches are meant for high port-density for end-users/phones/servers that a virtual switch cannot match unless everything is already virtual.

>>> Good points above. More could be added, e.g. (some of this overlaps with what you said, but is more detailed):

>>> - a hardware switch has TCAM memory, so it can perform rule matches very quickly; a software switch must keep all its rules in RAM, and search through data structures to find matches, making it much slower.

>>> - On the other hand, a hardware switch can be slow at updating its flow tables because of a bottleneck in changing TCAM entries; a software switches doesn't have this bottleneck.

>>> - A hardware switch has a fixed number of physical ports between which it can forward packets; a software switch has no physical ports, but virtual ports can be added dynamically, e.g. to connect virtual machines.

>>> - A hardware switch usually has a low-powered CPU, and so is limited in the amount of processing that it can do in addition to its core packet forwarding function; a software switch normally runs on a server, so CPU power and memory are plentiful, allowing the switch to do more advanced packet processing – e.g. firewall, IDS, NAT, load balancing

‘A network administrator should use either all software switches or all hardware switches in their organisation’s network, but not both’. Do you agree? Explain your answer.

No idea.

>>> I won't give a full answer here. The answer isn't on the slides and probably not on Google. But it's something you should be able to think about. Does it seem like a good idea to have all software switches, given the trade-off in forwarding speed? On the other hand, if you have VMs running on a server, you have to have a software switch to allow them to share the use of the physical NIC(s) on the server, so why not make that SDN controlled? I'll leave you to think it through from a practical point of view to formalise an answer.

What functions must an SDN controller provide?

An SDN controller manages flow control to the switches/routers “below” (via southbound APIs) and the applications and business logic “above” (via northbound APIs) to deploy intelligent networks. They consolidate and mediate between different controller domains using common application interfaces.

>>> You could find a better answer in the slides - have a look at slides 32 and 33 (there's repetition between them) of the 'How SDN Works' slide-deck. The points just need to be expanded into sentences.

Why does an SDN controller need to keep a flow-rule cache?

A flow-rule cache defines the action that should be taken in the event of a message arriving at a port from an address. It is instrumental in defining all the operations of a network.

>>> A flow-rule defines the action that should be taken in the event of a packet arriving at a port on a switch from another device. And flow-rules on a switch are contained in flow-tables. A flow-rule cache, on the other hand, is the controller's copy of the rules that it thinks are in the switches' flow-tables. In the slides, it's described as mirroring the flow-tables in the switches. The purpose is that if there is a mis-configuration of a switch discovered, the flow-rule cache can be used to re-build the switches flow table. Or if a switch fails, the flow-rule cache can be used to identify what flows are affected, and rules adjusted on neighbouring switches to re-route flows around the failed switch. The flow-rule cache can be used for keeping track of flow-rule histories, which might be required later for auditing purposes. The flow-rule cache can be useful for verification – e.g. checking a sequence of rules to make sure a packet can reach its destination. The cache can be analysed to identify more efficient combinations of rules.

*A network manager you know is planning to use SDN applications running on a controller to replace some expensive hardware appliances. Give your advice on how she should proceed.*

If the hardware is already in place, then maybe it’s still faster. Does the network change much? Is there a need for a central controller?

>>> You could write a longer answer. What about how mature SDN implementations are? For the hardware appliances, are there direct equivalent SDN applications? Do they have the same functionality, the same load-handling capabilities? Is it acceptable that the SDN applications will be located centrally (at the controller), or will something be lost by not being able to position them as can be done with the hardware appliances? Maybe there's scope for a gradual migration to SDN applications - and it might turn out that it makes sense to retain some hardware appliances, so as not to lose functionality, or for load-sharing with the SDN applications.

*Compare two SDN controllers.*

ONOS vs ODL.

Both are written in Java.

In the event of a fault, ONOS does eventual consistency and ODL does high availability.

Scaling of both eventually leads to diminishing returns as the number of activities and syncing increases. ONOS guarantees all nodes have the same data.

Both in the Linux Foundation umbrella, more support for ODL.

>>> All relevant points. Also, they're both open-source. ODL traditionally had a vendor focus, ONOS was service-provider focussed. Both provide Java and RESTful northbound APIs.

*There are several SDN controllers currently available. What led to there being more than one? How do you expect the situation to develop in the next few years, and why?*

No idea.

>>> The fact that OpenFlow is an open standard protocol meant that anybody could implement it. The developers who implemented it early on did so as open-source code, which anybody could use or modify for their own purposes. And that's what happened - Floodlight and ODL were based on Beacon. ONOS seems to have developed independently, but the developers would have been informed by the efforts of the other controller teams. There were other controllers developed as well - Pox, Ryu, lots of proprietary controllers, and projects that didn't make it to full fruition. This is how things work in the open-source community, especially with a new technology, and from seeing what happened with Linux, for example, it would be expected that the market would coalesce around a small number of controllers (ODL and ONOS), with room for niche controllers (Pox for teaching and research), or mavericks that have very specialised functionality.