SPARC
ICT-258457

Deliverable D3.2

Update on Split Architecture for Large Scale Wide Area Networks

Editors:	*Wolfgang John, Zhemin Ding; Ericsson (EA)*
Deliverable type:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M18
Actual delivery date:	M18
Version:	1.0
Total number of pages:	93
Keywords:	Split Architecture, OpenFlow, Carrier-Grade, Network Virtualization, Resiliency, OAM, QoS, Service Creation, Scalability, Energy-Efficient Networking, Multi-Layer Networking

Abstract

This deliverable aims to define a carrier-grade Split Architecture based on requirements proposed in WP2. It presents an updated Split Architecture for large scale wide area networks such as access/aggregation networks, and evaluates technical issues against certain architecture trade-offs. First, an overview of a carrier-grade Split Architecture is given, including a flexible control layer hierarchy and a related service architecture model. Next, extensions to OpenFlow to support a carrier-grade Split Architecture are discussed. Important topics included here are a) Openness & Extensibility: We present a solution of how to extend OpenFlow with more advanced processing functionalities on both data and control planes; b) Virtualization: We discuss how OpenFlow enables a flexible way of partitioning the network into virtual networks while providing full isolation between these partitions; c) OAM: We discuss the contradictory goals of legacy functionality vs. datapath element simplicity. We present a solution for both technology-specific OAM (MPLS BFD) and a novel technology agnostic flow OAM. d) Resiliency: We present approaches for survival from link failures or failures of controller or forwarding elements; e) Topology discovery: We discuss issues with current discovery of network devices and their interconnections; f) Service creation: We present solutions for integration of residential customer services in the form of PPP, and business customer services in form of Pseudo-Wires (PWE) into an OpenFlow controlled operator network; g) Energy-efficient networking: We present possible approaches for energy savings and discuss the requirements they pose on the OpenFlow protocol; h) QoS aspects: We discuss how traditional QoS tools such as packet classification, metering, coloring, policing, shaping and scheduling can be realized in an OpenFlow environment; i) Multilayer aspects: We discuss additional control requirements posed by optical network elements and outline three different stages of packet-optical integration in an OpenFlow environment. Finally, this report discusses selected deployment scenarios of a carrier-grade Split Architecture with OpenFlow. We show how OpenFlow-based Split Architectures could be adopted for relevant scenarios faced by modern operator networks such as service creation, general access/aggregation network scenarios and peering aspects, i.e., how to interconnect with legacy networks. Finally, we present a numerical scalability study indicating the feasibility of a Split Architecture approach in access/aggregation network scenarios in terms of scalability requirements.

Imprint

[Project title]	***Split Architecture for carrier grade networks***
[Short title]	***SPARC***
[Number and title of work package]	***WP3 – Architecture***
[Document title]	***Update on Split Architecture for Large Scale Wide Area Networks***
[Editors]	***Wolfgang John, Zhemin Ding***
[Work package leader]	***Wolfgang John***
[Task leader]	***Wolfgang John***

List of authors

Organization/Company	**Authors**
DT	**Mario Kind, F.-Joachim Westphal, Andreas Gladisch**
EICT	**Andreas Köpsel, Ahmad Rostami, Hagen Woesner**
EAB	**Wolfgang John, Zhemin Ding**
ACREO	**Pontus Sköldström, Viktor Nordell**
ETH	**András Kern, David Jocha, Attila Takacs**
IBBT	**Dimitri Staessens, Sachin Sharma**

Table of contents

List of figures and/or list of tables

1 Introduction

1.1 Project Context

The SPARC project ("Split Architecture for carrier-grade networks") is aimed at implementing a new split in the architecture of Internet components. In order to better support network design and operation in large-scale networks for millions of customers, with high automation and high reliability, the project will investigate splitting the traditionally monolithic IP router architecture into separable forwarding and control elements. The project will implement a prototype of this architecture based on the OpenFlow concept and demonstrate the functionality at selected international events with high industry awareness, e.g., the MPLS Congress.

The project, if successful, will open the field for new business opportunities by lowering the entry barriers present in current components. It will build on OpenFlow and GMPLS technology as starting points, investigating if and how the combination of the two can be extended, and study how to integrate IP capabilities into operator networks emerging from the data center with simpler and standardized technologies.

1.2 Relation to Other Work Packages

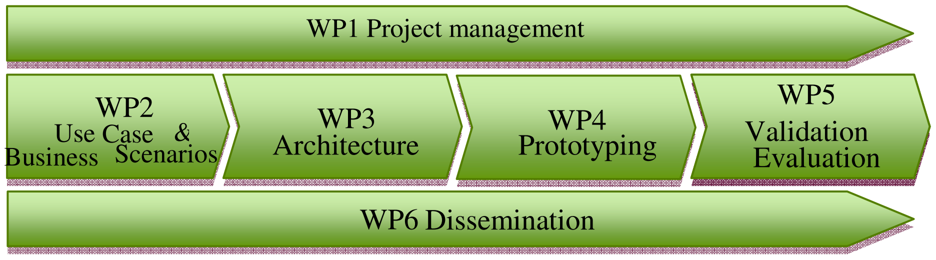


Figure 1: Relation of SPARC work packages

In the "workflow" of the work packages, WP3 is embedded between WP2 (Use Cases / Business Scenarios) and WP4 (Prototyping).

WP3 will define the Split Architecture taking use cases and requirements of WP2 into account and will analyze technical issues with the Split Architecture. Moreover, this architecture will be evaluated against certain architectural trade-offs. WP4 will implement a selected subset of the resulting architecture, and feasibility will be validated in WP5. WP6 disseminates the result at international conferences and publications.

1.3 Scope of the Deliverable

This deliverable tries to answer many questions raised in SPARC Deliverable D3.1. It also takes into account the requirements derived from different use cases in Deliverable D2.1. The major goal is to describe and compare different design approaches in OpenFlow (controller, forwarding element, interface protocol, scalability, resiliency, virtualization, etc.) to support and extend the Split Architecture for the access/aggregation use case in a carrier grade environment.

1.4 Report Outline

This report presents analyses and evaluations of technical issues regarding an updated carrier-grade Split Architecture, including further analysis and conclusions derived from architectural trade-offs. The report starts by giving an overview of the envisioned carrier-grade Split Architecture in Section 2. The section describes a generic, high-level overview of the Split Architecture, i.e., decoupling control from forwarding plane. It also discusses additional possibilities for a control architecture, like multiple parallel control planes and a layered hierarchy of stacked control entities. In Section 3 we then propose required extensions to OpenFlow in order to support the envisioned carrier-grade Split Architecture. This section proposes necessary extensions with respect to carrier-grade requirements based on missing features as identified in earlier SPARC deliverables (D2.1 and D3.1). Topics include openness and extensibility, virtualization and isolation, OAM, resiliency aspects, topology discovery, service creation, energy-efficient networking, QoS and multilayer aspects. Finally, in Section 4 we present selected deployment scenarios of a carrier-grade Split Architecture with OpenFlow. We show how OpenFlow-based Split Architectures could be adopted for relevant scenarios prevalent in modern operator networks, such as service creation, general access/aggregation, network scenarios and peering aspects. Finally, we present a numerical scalability study indicating the feasibility of a split-architecture approach in access/aggregation network scenarios in terms of scalability requirements.

2 Carrier-Grade Split Architecture Overview

2.1 Architecture Overview

Today the design of network elements (switches, routers, etc.) follows a monolithic design pattern, i.e., each networking element integrates functions for forwarding, control and processing. Usually it is not possible to access the interface in-between these functional blocks. Splitting this design into functional planes (forwarding, control, processing) and open the interfaces in-between the different planes, could be a successful way to relieve existing hardware and infrastructure from legacy architectures, and thus facilitate evolution and deployment of new protocols, technologies and architectures. In this deliverable we study extensions to the *split-architecture* design in order to fulfill carrier-grade requirements (e.g., OAM, resiliency, network virtualization, etc.). As a result, carrier networks could benefit from the advantages of Split Architectures and enable network operators to have a disjointed evolution of datapath and control mechanisms, which has the potential to pave the way toward more dynamic control of services and connectivity in carrier networks.

In order to realize a Split Architecture for carrier-grade operations, we decided to utilize the OpenFlow protocol in its current version 1.x [19], which provides a basic interface for controlling network forwarding elements. We have discussed the pros and cons of OpenFlow and the IETF's ForCES framework [21] in more detail in Deliverable D3.1 Section 5.2. To conclude the discussion started there, we initially had the impression in SPARC that the ForCES framework is in some aspects more mature when compared to OpenFlow. However, the strong industry support recently observed for OpenFlow, and the lack of support for ForCES in academia and industry, makes OpenFlow an interesting evolving technology today. Therefore, we will concentrate on OpenFlow for a carrier-grade Split Architecture in this and upcoming deliverables.

Split Architectures provide a significant degree of freedom to network planners and designers as they remove several constraints and boundaries found in legacy architectures. An operator network is typically structured in transport domains. Today the size and scope of such transport domains are typically fixed by manual network planning as routing and switching devices now provide different functions. However, in Split Architectures the differences among switching and routing devices start to disappear as datapath elements are capable of providing both functions in parallel. This enables network operators to adapt a fixed transport domain and its boundaries based on dynamic conditions (e.g., load situation, etc.), of course within the limits defined by the physical deployment and wiring of datapath elements. Several stages of integrating OpenFlow in carrier networks can be identified:

1. **Emulation of transport services**: As a first step, OpenFlow may be introduced in transport domains (e.g., Ethernet, MPLS, optics, etc.) by replacing legacy network devices with OpenFlow-compliant datapath elements and deploying a control plane that emulates behavior of the legacy transport technology in use, e.g., an Ethernet domain, an MPLS domain, etc. All nodes connected to such an OpenFlow enhanced transport domain still use legacy protocols for providing service and remain unaltered. OpenFlow in its versions 1.0 and 1.1 provide all the means to control Ethernet transport domains in such a scenario. However, support for enhanced Ethernet or MPLS services (e.g., those from the Metro Ethernet Forum), including OAM and reliability features, is beyond scope of OpenFlow 1.0/1.1.
2. **Enhanced emulation of transport services**: For a carrier-grade Split Architecture, a number of mandatory features and functions must be added to OpenFlow in order to fully comply with OAM requirements (among others), resiliency and scalability needs. OpenFlow lacks support for such advanced functions in versions 1.0 and 1.1 and must be extended accordingly to emulate carrier-grade transport services. Basic MPLS support was added to OpenFlow 1.1, but support (e.g., for MPLS-specific OAM schemes like BFD) is still lacking. We cover some of the necessary extensions to OpenFlow 1.0 and 1.1 in Section 3 of this deliverable, including OAM, advanced processing, interaction with legacy stacks, resiliency, and multilayer operation in OpenFlow. Again, all service nodes in this second integration scenario remain unaltered.
3. **Service node virtualization**: Thus far we have focused on ways of emulating legacy transport domains with OpenFlow. However, besides such basic transport services, carrier-grade operator networks provide a number of additional functional elements, e.g., for authentication and authorization, service creation, enforcing quality of service, etc. Most of these functions are today located on a limited set of network devices; the discussions in Deliverable D2.1 have documented the exposed position of the Broadband Router Access Service Gateway (BRAS) in carrier-grade operator networks according to the architecture defined by the Broadband Forum and deployed by most operators. A third integration level for a Split Architecture is virtualization of such service node functions inside OpenFlow. This involves control plane as well as datapath elements to cope with more advanced processing needs, as interfacing with more legacy protocol stacks must be supported. For the access/aggregation use case, we will showcase the virtualization of service nodes in OpenFlow in more detail in Section 3 based on an access/aggregation PPP/PPPoE UNI example.

4. **All-OpenFlow-Network**: Obviously, OpenFlow deployments may be pushed forward to other network domains as well, e.g., for controlling residential gateways (RGW) in customer premises networks or toward the operator's core domain. Controlling RGWs may simplify service slicing and service deployment in customer premises networks, but defines new constraints on an operator's control plane in a Split Architecture: Controlling customer-owned devices outside of the network operator's area of responsibility may impose additional security requirements. However, these security implications are beyond the scope of this deliverable.

Besides pure emulation of legacy transport services and service node functions, Split Architectures also pave the way toward network virtualization, e.g., for the sharing of a physical infrastructure among various operators. Our goal for a new split-operator network architecture is the ability to deploy various control planes comprising arbitrary control logics in parallel on the same physical infrastructure, and to assign flows dynamically based on operator and user policy to one of the available control planes while preserving the necessary degree of mutual flow-space isolation, while providing adequate scalability and performance.

We cannot anticipate the potentially wide variety of control planes that network operators may intend to deploy in the future, but a number of common principles should be available in a carrier-grade Split Architecture as proposed in this document:

Flowspace management: In the current OpenFlow specifications, a datapath element is controlled by a single control entity. For network virtualization this is a serious limitation, as this single controller is controlling the entire flowspace on the datapath element. Many available OpenFlow controllers address this issue by implementing a (de-)multiplexing entity and a publish/subscribe subsystem, to which multiple network applications can be attached. This virtualizes the flowspace as a network application can actually select the packets it is interested in. However, the existing controller implementations (e.g., NOX [23], Beacon [24], Trema [25]) all presume a cooperative publish/subscribe subsystem: The (de-)multiplexing entity does not authorize or control which parts of the flowspace a specific network application actually controls, so clashes among different network apps are likely. For network virtualization in carrier-grade environments, the system should prevent such collisions and provide some form of authorization of flowspace control for individual network apps (=control entities).

Therefore, we propose an extension of flowspace management to the OpenFlow base API so that a network application can actually specify which parts of the flowspace it claims control over, and the system can resolve potential control collisions among different control elements. And, contrary to existing software controllers like NOX, we relocate the (de-)multiplexing entity close to the northbound interface on the datapath element, thus enabling several control entities to control a single datapath element in parallel in a clear manner. Flowspace Management resembles the functionality provided by the experimental FlowVisor software [22], but contrary to FlowVisor, which is controlled via a dedicated management interface, we integrate all flowspace management messages in the OpenFlow base API. Flowspace Management is discussed further in Section 2.2.

Advanced processing: Interaction with legacy network domains and their associated protocol stacks is inevitable, at least during a migration phase while deploying a Split Architecture like OpenFlow in a carrier network. We have already pointed out (in the preceding Deliverables D2.1, D3.1) that the set of supported protocols is rather limited in OpenFlow. Furthermore, the action-based processing framework provides lightweight, stateless actions that do not maintain state among consecutive packets and cannot be used for more sophisticated state-maintaining protocols.

The virtual port idea has already been discussed in the OpenFlow community, i.e., encapsulating a specific processing functionality and hiding this from the OpenFlow datapath element. Virtual ports raise questions concerning a number of minor issues like the handling of processed packets (reinjection into the forwarding engine, direct send-out via another port, pairing of virtual ports for transparent encapsulation, processing, decapsulation, etc.). However, the black box principle of hiding a processing functionality behind a virtual port ignites another discussion about how to structure processing modules, how to interconnect and organize them and how to specify this configuration via the OpenFlow API (see Section 3.1). These questions are motivated by the fact that processing should be controlable via a common API across all different types of datapath elements, and should not be bound to proprietary interfaces and management APIs.

We propose a modularization of processing into atomic processing modules, whose mutual relationships are mapped to a layered control plane structure. Each control plane layer maps to an encapsulation (=a protocol) and traveling along the stack defines the links among these processing modules. We will cover this principle in more detail in Section 2.3.

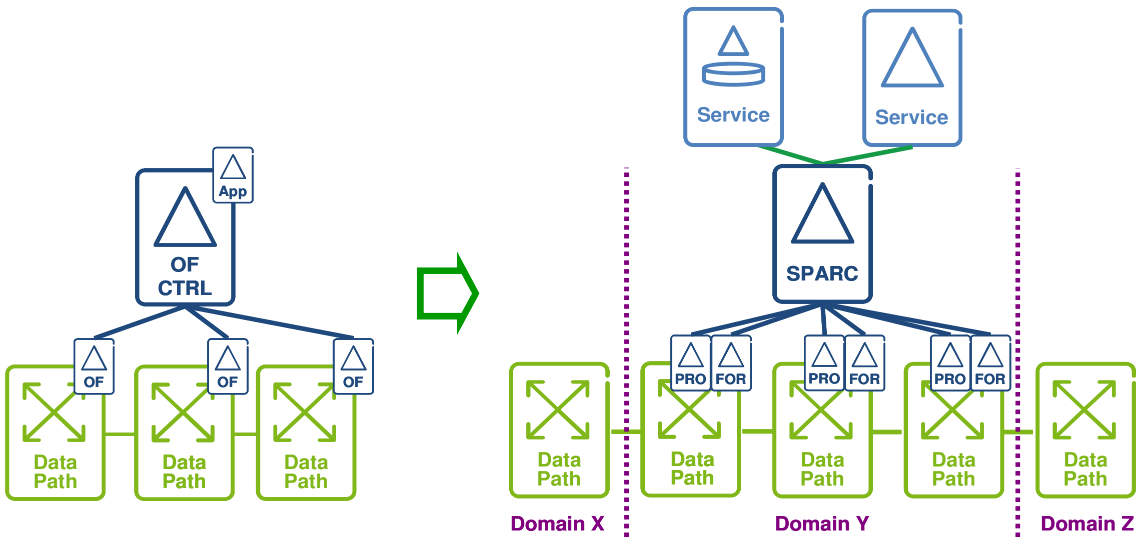


Figure 2: Current OpenFlow-based Split Architecture (left) vs. SPARC Split Architecture (right)

Recursive Stacking: Any Split Architecture needs some form of synchronization of a data model between data and control plane so that both planes share a common view of the controllable resources. OpenFlow defines a rather limited data model (pure Ethernet abstraction on ports, etc.), but the interesting aspect is that Split Architectures allow tight control over the details exposed by altering the data plane model visible to controlling entities. If we adopt the same API (like OpenFlow) on northbound and southbound interfaces, we can actually expose various data models with different levels of detail to different higher-layer control planes.

Such a recursive scheme as depicted in Figure 3 enables network operators to grant partial control over their network to third-party operators, e.g., content providers, while at the same time concealing any business-critical details from them. Further, concealing complexity from higher layers in a recursive stack of control entities may have a positive impact on overall performance and scalability. When adopting a recursive architecture, each layer acts similar to a proxy device and appears as a control entity to lower layers and as a data plane element to higher layers. To some extend the FlowVisor software already implements such a functionality for network virtualization. More details on the proxying functionalities of control entities follow in Section 2.4.2.

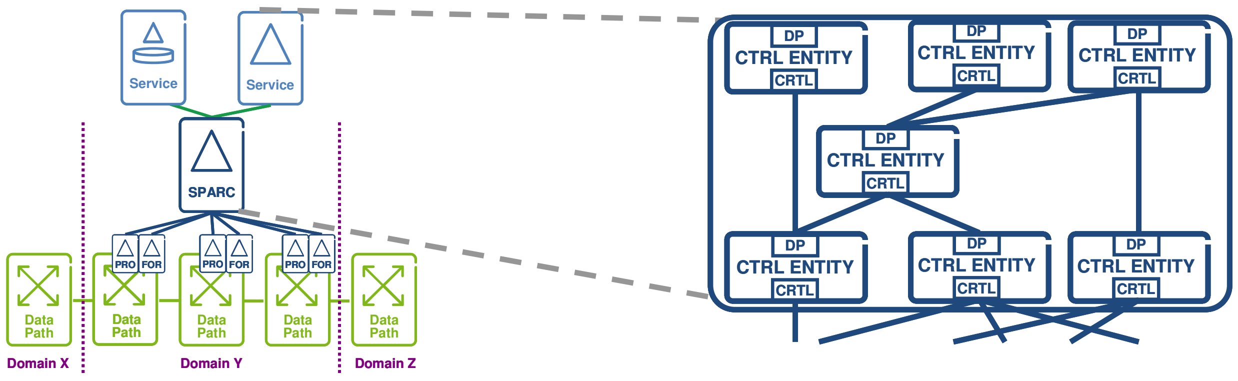


Figure 3: SPARC controller framework

Three remarks should be added here: 1) A layered scheme does not mean that we follow the ISO/OSI layering. Each layer is an OpenFlow layer and potentially covers the entire packet header. 2) Introducing a recursive scheme does not imply that any control plane must be organized in such a stack of layers. Rather, shaping the exposed data plane model offers an additional degree of freedom when creating service slices and granting control over these. We will cover the benefits in Sections 2.4.1 and 2.4.2 in more detail. However, creating a single layered control plane is still a valid option. 3) In addition, organizing the control plane in layers does not imply that a centralized control plane architecture is assumed. A layer may be organized with multiple distributed peers for scalability concerns so that it resembles a distributed IPC facility [26].

The proposed architectural extensions to a Split Architecture control plane based on OpenFlow do not dictate any constraints on control plane designers about how to organize the control plane. This provides network operators the freedom of adapting the control architecture according to their needs, and, at the same time, recursive stacking, flowspace management, and advanced processing offer additional freedom of designing service architectures and coping with advanced business models. In the following subsections we outline these principles of a general Split Architecture concept for carrier-grade networks as considered by SPARC.

2.2 Management of Flowspaces

2.2.1 The management of network applications in OpenFlow

Let us start with two observations from the existing control frameworks for OpenFlow (NOX is the most prominent open source controller at the time of this writing): Most control frameworks differentiate between a control framework and individual network applications. A publish/subscribe subsystem enables a network application to register for specific event types, and it is the network application's responsibility to filter only for those event notifications it is willing to control. However, this model of cooperation among multiple controllers relies on considerate behavior of all active network applications to prevent any intentional or unintentional collisions when controlling a specific flow. A race condition will occur if two network applications send FlowMod messages for the same packet buffered in the datapath, and this results in unpredictable behavior.

In the OF1.x series of specifications, OpenFlow defines a 1:n relationship between a controlling entity and a datapath element, i.e., a datapath is linked to exactly one controller device. As non-destructive cooperation among multiple network applications cannot be assured, the OpenFlow framework should be extended so that individual network applications register for the flowspaces they are willing to control. A datapath must verify all protocol messages received from a network application according to its previous flowspace registration and must drop control packets that control flows outside of this registered flowspace. As the final consequence, OpenFlow should be extended so that multiple network applications can control different parts of the overall flowspace and a (de)multiplexer function must be integrated to ensure this. The question arises as to where to deploy such a demultiplexing mechanism. Another observation from the existing NOX framework may help here: The interface between a network application and the publish/subscribe system is clearly defined – it is again the OpenFlow API, i.e., each network application is in fact an OpenFlow-compliant control entity by itself. Consequently, a network application could connect directly via a dedicated control connection to the underlying datapath element, and the datapath demultiplexes various control connections based on established flowspace registrations.

To summarize:

- Network applications are controllers by themselves.
- Multiple network applications may coexist in parallel, e.g., for network virtualization or due to splitting network functions into smaller control modules for easier implementation and deployment.
- In order to ensure proper coexistence of network applications, a (de-)multiplexing mechanism must be integrated in a datapath element.
- (De-)Multiplexing should be based on flowspace registrations, i.e., each controller must specify the subset of the flowspace it claims control of.
- With flowspace registrations and a multiplexing functionality, the 1:n relationship between control entities and datapath elements from the current OF1.x specifications is replaced by an m:n relationship where multiple controllers may control a single datapath element in parallel.

2.2.2 Slicing and flowspaces

Before we go further, let us briefly cover some properties of flowspaces and possible options for slicing flowspaces.

Packet header information is typically structured into fields as a MAC address (source and destination), VLAN tag, IP addresses, protocol type, port, MPLS tag #1,#2, … etc. OpenFlow defines 14 of these headers in version 1.1 of the specification. The different headers correspond to multiple layers and allow the scalability of communication by reducing the number of communicating entities per layer.

It is now possible to view all these headers as one large tag giving an identifier to an individual packet or flow. This *flattening of the name space* is an attractive feature of OpenFlow because it reduces the total numbers of layers (which typically translate into specific boxes in a carrier grade network). Still, however, the functions at the border of the network need to restructure the flat label into processable and meaningful headers.

The many-to-many relation of adjacent layers also enables the *parallel deployment of different control planes* and allows an individual assignment of resources to one of the deployed control planes. This requires an appropriate resource slicing or virtualization solution at any of the server control layers.

When multiple controllers share one underlying controller or datapath element, the slicing between them requires multiplexing/demultiplexing. Controllers of the control layer (n+1) need to be addressed from the underlying controller/datapath element. This demultiplexing of messages (e.g., the Packet_In message) needs to be encoded in the flow itself, as there is no additional information than the packet header. This means that the known endpoints of layer n are exposed to the layer (n+1) controller.

Each controlling entity of a client layer must be aware of which resource slices are allocated for it. This information can be obtained from management entities, as it is done in GENI's FlowVisor and Opt-In Manager solution [15]. The controlling entity may also poll the server layer to get the usable resources. As an alternative the controlling entity may request resources it is willing to control.

A layer (n) controller exposes the known endpoints (which are addresses from flowspace n). A layer (n+1) controller then requests the slice by specifying a subset of this list. Multiple parallel layer (n+1) controllers therefore would exclusively share the set of layer (n) endpoints. For example, an Ethernet controller would pick the Ethernet ports to be controlled by it, two Ethernet controllers on one switch would be possible, and the slicing would take place on physical ports.

As a second example, one layer up, a number of MAC addresses that is known to an Ethernet controller (through MAC learning, for instance) can be divided by multiple IPv4 controllers, corresponding to multiple routers on a single Ethernet switch. These routers would receive their own MAC addresses for the IP router ports from the layer (n), in this case the Ethernet controller.

A control layer receives two categories of configuration requests: downstream from a controlling entity (residing in a higher layer) designated to control the offered resources, and upstream from a controlled entity (residing in a lower layer) providing triggers. For example, a trigger can be a PDU from the data plane or a notification about the changes in the resources offered by the controlled entity.

To sum up: The service API between adjacent control layers shall provide a means of issuing configuration requests toward a lower control layer, receive configuration triggers from lower control layers along with enhanced management features, like flowspace management, virtualization, and control slice isolation. The service API will also be exposed to external content providers and thus needs authentication and security features.

2.3 Advanced processing framework

Deploying OpenFlow-enabled network domains is a challenge for network planners who must ensure interaction between legacy networks and protocol domains. A typical example of the interaction of an OpenFlow domain and legacy protocol environments is the access/aggregation use case as discussed in Deliverable D2.1. In this case, the access/aggregation domain attaches to the customer premises network via the RGW and the core network via the BRAS.

Interacting with legacy protocol stacks entails a number of changes to OpenFlow, especially the need for adding yet unsupported termination functions in the data plane. As we intend to avoid pollution of the base protocol by repeatedly patching in additional protocols, OpenFlow should be extended with a smart mechanism for easy addition of new protocols and processing capabilities. Moving termination functions into the control plane is another alternative, but this will presumably not fulfill performance requirements of operator networks. OpenFlow 1.x defines a framework for integrating lightweight processing actions. However, these actions share a common property as they are stateless in nature, i.e., they cannot keep state across several consecutive packets. Within the access/aggregation use case, we have already seen the need for stateful processing within the data plane (e.g., PPP-LCP and the joint PPP-PPPoE finite state machine and timeout events in ECHO request/reply exchanges with the inherent termination of the underlying PPPoE session).

The OpenFlow community has discussed the virtual port concept for some time now. Virtual ports may be used to attach arbitrary processing entities to a data-path-forwarding engine. Virtual ports are attractive candidates for advanced processing as they hide all processing logic inside a black box. Management of such virtual ports (the usual CRUD operations – Create, Read, Update, Destroy) may occur via some proprietary external management interface/API. However, the proprietary black-box principle endangers a split-architecture by making it non-portable among different datapath elements: Complex black boxes may force the control plane to adapt to this specific "magical virtual port" for providing its service. This contradicts the general concept of Split Architectures: A control plane should be independent from any changes in the data plane and vice versa. Therefore, what should an enhanced processing framework for OpenFlow look like?

Layering is a fundamental principle of organizing protocol stacks in today's packet switching networks. Each layer utilizes the transport services of lower layers for providing its own service that is exposed to higher layers via a service access point. OpenFlow flattens the flowspace and in principle dissolves the boundaries of packet headers, but when interacting with layered legacy stacks, following the layered stack for organizing the processing seems useful. Each layer uses transport services of its attached lower layer(s), e.g., in the access/aggregation defined PPPoE/PPP stack, PPP uses the transport services of PPP-over-Ethernet to convey its PPP packet data units to its peer. PPPoE utilizes Ethernet to convey PPPoE PDUs to its peer PPPoE entity, and so forth. A processing entity on an OpenFlow datapath element may follow this layering principle and may stack various “atomic” processing modules on top of each other, replicating the legacy protocol stack.

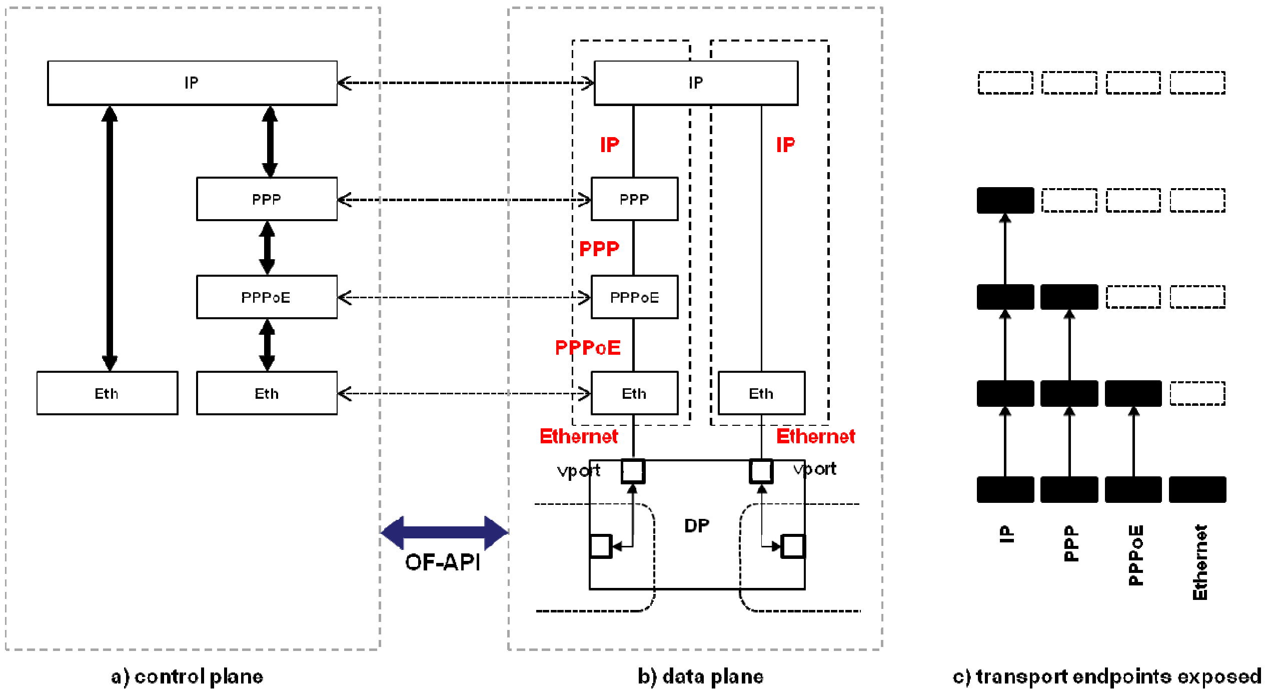


Figure 4: Emulating a legacy PPPoE/PPP stack in OpenFlow with atomic processing modules

A number of properties are conducive to such a layering of the control plane and its associated processing modules:

- A processing module should cover a single protocol entity (e.g., IP or PPP or VLAN or PBB), but not a combination of various protocol entities. This enhances reusability of existing processing modules as they can be inserted into a protocol stack at various places.
- A processing module should be easily describable via the OpenFlow API so that a control plane can make use of such a processing module across datapath elements from different manufacturers. A processing module description language is required.
- A processing module should be enabled to specify the service data unit (SDU) types it expects via its lower layer and higher layer service access points (SAP), e.g., a PPPoE module should accept Ethernet-only frames on its lower SAP and PPP frames on its higher layer SAP. A module description language should also cover the module’s constraints.

Processing modules define specific constraints on their higher- and lower-layer service access points, i.e., the SDU type expected on both interfaces. When linking two adjacent processing modules, the SDU types must match. A datapath element exposes all types of processing modules and their SAP constraints. Within the layered control plane, the lowest layer (presumably the Ethernet layer) sees all processing modules. While travelling upstream in the stack, some lower layer processing modules cannot be exposed further to higher layers, e.g., a PPPoE control entity provides a PPPoE transport service and uses a PPPoE decapsulation/encapsulation processing module in the data plane. However, such a PPPoE processing module will not be exposed by the PPPoE control entity toward higher layers because this type of processing module is not useful on higher layers. Figure 4 depicts the available processing modules (=transport endpoints) on the right hand side and shows how this set of modules decreases while moving upstream within the stack.

We cover this processing framework in more detail in Section 3.1.

2.4 Recursive Control Plane Architecture

2.4.1 Exposing dynamic data plane models

We have seen from the previous discussions that a Split Architecture could enable a network operator to deploy different control planes in parallel. In addition, a Split Architecture also enables a network operator to allow external third parties at least partial control over the forwarding mechanisms used for specific flows in the network. This allows network and content providers to jointly offer improved and optimized services. However, in such a scenario the network operator should have control over the level of detail exposed to a third party deploying its own control plane. This hides complexity from the external control plane and addresses the operator's security requirements as it keeps performance-relevant details confidential.

For example, let us take a typical data plane for an access/aggregation domain with a stack of transport technologies as shown in Figure 5: In this case, optical transport, Ethernet, MPLS, and IP define the overall data plane. Today the data plane operates as a black box, where the content provider has no insight about its structure, topology, load, etc. Users requesting access to the content provider's services may desire to be redirected to different data center locations based on several constraints, e.g., typically the quality of service parameters.

In a Split Architecture, a data plane exposes a data model that comprises all information about the data plane's internal structure. A network operator should be enabled to tailor the data plane model according to the desired level of detail to be exposed, which has been defined previously by the content provider and network operator in their contractual arrangements. Compare this to the existing peering and overlay models (see [14], Section 13.2 for a brief introduction and discussion of this topic): In a peering model, all details are actually available to an adjacent domain, while in the overlay model, the details are hidden behind the service API. A layered control plane in a Split Architecture applies the hybrid or augmented model, where the degree of information made available to a control layer is highly flexible.

In a Split Architecture, the data model can be easily adjusted to alter the level of abstraction, i.e., the details exposed to the control plane, by introducing a layered model. Here various control layers are stacked upon each other in a hierarchy and each layer acts as a controlling entity toward datapath elements in a lower layer and as a data plane (element or elements) toward higher layers.

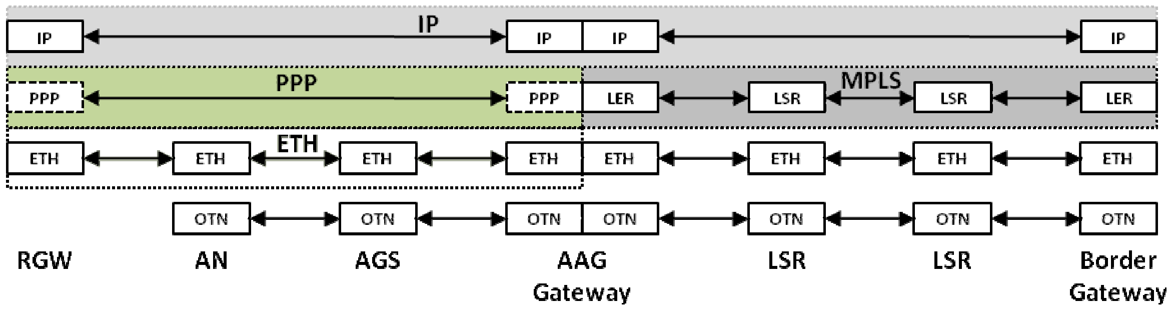


Figure 5: Full data plane model exposed

Such proxy behavior is shown in Figure 6, where a control layer in the control stack deployed by the network operator emulates a virtual topology: All nodes visible in the IP layer are exposed, all lower layer details remain hidden. This data plane model comprises all routers connecting core network and access/aggregation domains and the routers connecting the operator core with its adjacent operator networks (peering points) that lead toward the content provider's data centers. Any details within the access/aggregation domain (aggregation switches, access nodes, access technology, etc.) and within the core (MPLS domain, LSRs, LERs, etc.) are removed from the data plane model.

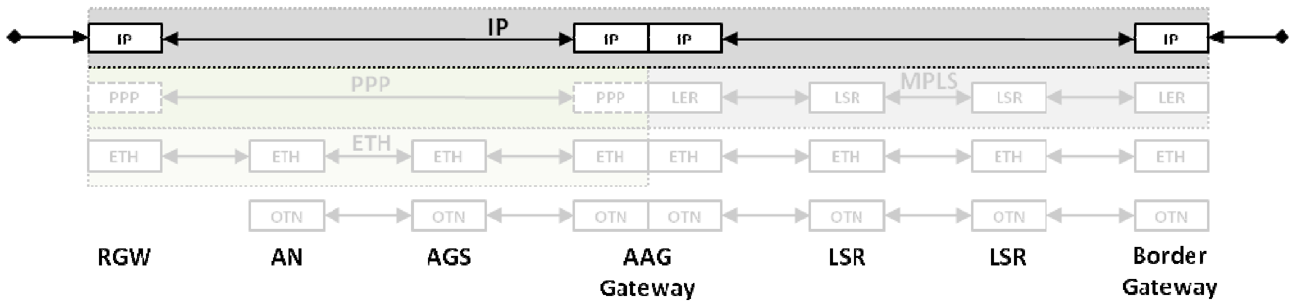


Figure 6: Only IP nodes exposed in data plane model

In principle, no restrictions on the structure of a virtualized data plane model exist, i.e., the control layer may create different emulated topologies depending on the needs of the higher control layers as shown in Figure 6 and Figure 7 respectively: The emulated data plane model depicted in Figure 6 consists of several datapath elements. The emulated data plane shown in Figure 7 provides a single datapath element only, where all users and peering points are emulated as being directly connected to one of the large switching element's ports.

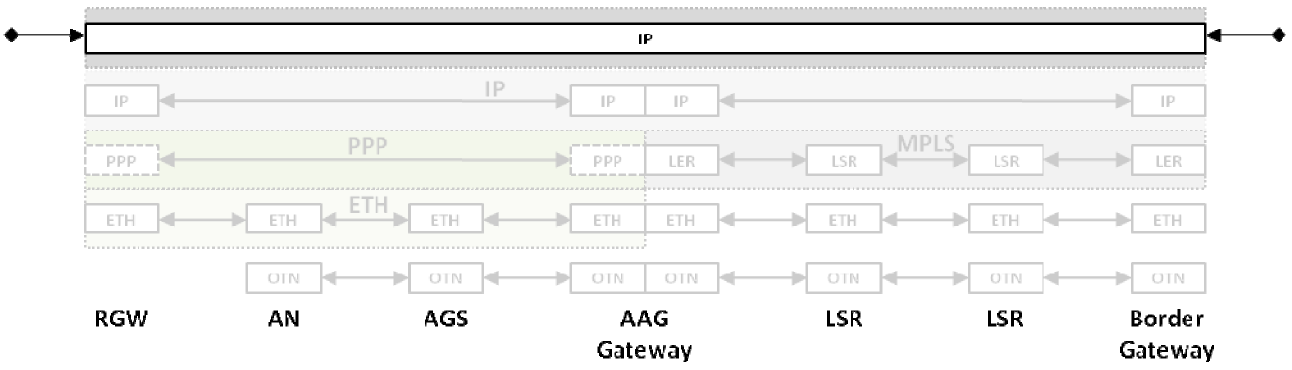


Figure 7: Virtual single IP node exposed in data plane model

In the simplified data plane model depicted in Figure 7, the actual flow forwarding between an ingress and egress port is done by the network operator's control plane, while in the enhanced data plane model from Figure 6 the content provider may make forwarding decisions for individual flows autonomously.

All emulated datapath elements hide details from higher layers: A forwarding request obtained from a higher layer must be mapped to appropriate control commands to handle the flow on the lower layers. These lower layers are represented by the backplane of the emulated datapath element. The exact manner of how to handle individual flows inside such a backplane is up to the owner of this control layer. In this example, the network operator may decide whether to use IP, MPLS, or yet another control mechanism for making such forwarding decisions on the backplane.

It is worth mentioning that a control layer may comprise various distributed instances – it is a distributed inter process communication (IPC) facility (DIF) [26]. Thus, some form of interworking among peer entities in a single "layer" may be established.

Deploying various control planes has an implication that we have not discussed yet: Each control plane controls only a specific set of flows in an operator network, not all flows, i.e., the data plane must forward flow control requests to the appropriate control plane based on some criteria. A carrier-grade Split Architecture should provide a means to negotiate the set of flows between data and control plane that are under the control of a specific control plane. Such a set of flows defines a flowspace. In the previous section we discussed an m:n relationship between multiple network applications and a datapath element, where a network application acts as a controller and defines the flowspace it claims control for. Flowspace registrations may be used for slicing between different control planes so that the data plane will forward all flow control requests that match the negotiated flowspace toward the appropriate control plane.

2.4.2 Virtual topology management

As discussed in the previous section, Split Architectures enable network operators to open their network toward content providers, while keeping control over the level of detail exposed in the hands of the network operator. This flexibility stems from the existence of a shared data model between the data plane and control plane. An intermediate proxy entity may alter such a data plane model. It controls the topology of an underlying data plane and exposes single or multiple topologies toward higher layers. Proxy entities may be organized in a stack of entities, thus various virtualized topologies may be created in a control plane. Please note that this stacking of proxy entities is not a classical layering as done in the ISO/OSI model or within PNA [26]. Rather, an intermediate proxy entity emulates virtual topologies in its exposed data model(s). Consequently, the northbound and southbound interfaces of such a proxy entity communicate according to the OpenFlow protocol.

When using flowspace registrations to give multiple control planes parallel access to a data plane, the question arises as to whether a single data plane model or individual models should be exposed to each of the attached control planes. This is a policy decision that depends on contracts negotiated between the network operator and any third operator responsible for attaching the control plane slice.

Creating a virtual topology conceals details of a lower-layer data plane model from higher control entities, e.g., an entire transport domain may be encapsulated within a single virtual datapath element. Such a scheme is shown in Figure 8: A proxy entity controls several datapath elements in the lower data plane and exposes a single datapath element toward the next higher stacked control entity.

- The lower data plane must be mapped in the exposed data model. A reasonable default may be to expose ingress/egress ports of the lower data plane as ports in the emulated single datapath element.
- Any control commands (FlowMod, PacketOut) received from the higher-layer control entities must be mapped onto appropriate lower layer commands.

- The transport network serves as logical backplane for the emulated datapath element. The technology used for realizing this logical backplane is invisible to the higher-layer control entities.

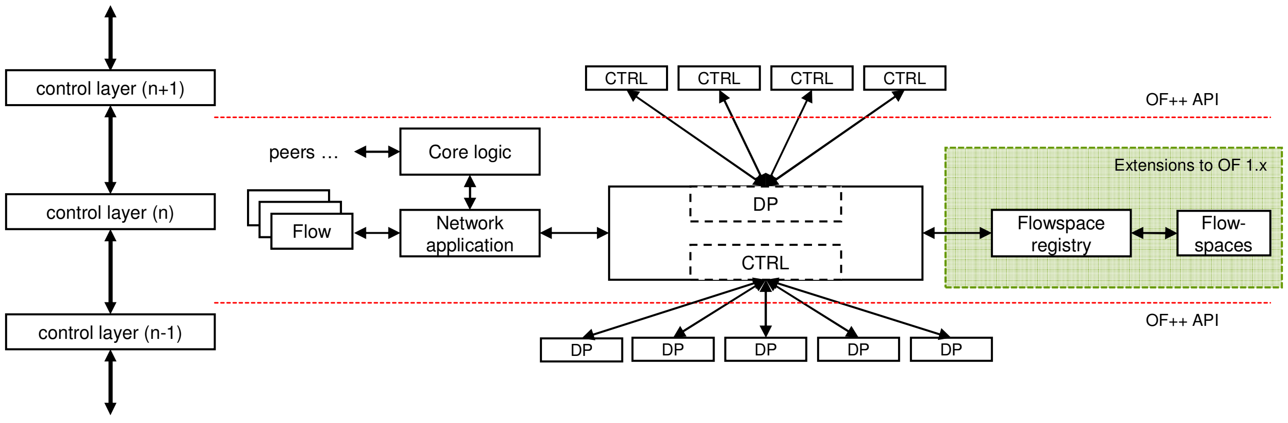


Figure 8: Virtual topology management with layered control entities

A proxy control layer may keep track of resource consumption in order to implement traffic-engineering mechanisms for providing adequate quality of service to the client layers. For example, this function covers the capacity of the emulated ports. The client layer may influence the shape of the emulated topology and may be able to set requirements on the capacities and capabilities of the emulated network. These requirements may change over time dynamically. Another benefit of the proxying functionality in the control entities is that it allows reusing the OpenFlow protocol to interface both directions, i.e., toward lower-layer and upper-layer entities.

Flow management may also provide a sufficient OAM toolset in order to determine if the configured flows are in line with the requested characteristics. In case of any deterioration, the flow manager should reconfigure the affected flows or notify the affected higher-layer controlling entity. Whenever the flow manager cannot reconfigure such flows, it will also notify the affected controlling entity. To monitor and validate the configured flow entries, the controller may collect statistics for monitoring traffic parameters for all active flows traversing the emulated data plane and consuming resources on egress and internal ports. These statistics can serve as a basis allowing the controlling entity to match a particular flow to its performance requirements set. Furthermore, the controller may use other means to verify the proper establishment and liveness of the configured flows.

A control layer may provide traffic shaping of individual flows based on local policy or some external mechanism on any outgoing datapath port.

Finally, at the end of this introductory section, a list of the core features of a hierarchical carrier-grade Split Architecture with the proposed functional properties can be summarized as follows:

- Several control planes can be deployed with minimized interference, i.e., control slice virtualization provides sufficient isolation between control slices.
- Flows can be assigned dynamically to different control planes based on the end user, the network operator and content provider policy.
- The hierarchical architecture allows a network operator to tightly control the emulated data plane exposed and hide any crucial details from a control plane.
- The hierarchical architecture allows stacking of an arbitrary number of control layers.
- The hierarchical architecture may enhance scalability of a carrier-grade Split Architecture control plane, as the hierarchy naturally enables smaller networking domains to appear as single datapath elements in higher layers concealing complexity.

3 OpenFlow Extensions for Carrier-Grade Split Architectures

From our earlier deliverables (D2.1 and D3.1) we concluded that current OpenFlow-based implementations do not fulfill carrier requirements, thus protocol extensions and standardization of certain functionalities are needed. In D2.1 and D3.1 we identified additional issues and open questions for carrier-grade Split Architectures. We will now summarize these previously identified requirements and identify missing features with respect to an OpenFlow-based Split Architecture. The identified missing features will be discussed in the present Deliverable D3.2 as separate study topics (Sections 3.1 to 3.9).

Requirements identified in D2.1

In Deliverable D2.1 we initially identified 67 particular carrier-grade requirements for Split Architectures based on our main use cases. These requirements have since been reduced in order to concentrate only on those requirements that are not already fulfilled by existing architecture concepts and available implementations, e.g., OpenFlow version 1.1. Specifically, the requirements have been prioritized with respect to overall importance, fulfillment in existing architecture concepts and/or existing implementations and their relevance for one or more use cases. As a concluding step in D2.1, the remaining list of particular requirements has been categorized into four groups of general, important requirements for Split Architecture carrier-grade networks:

(a) Modifications and extensions to the datapath elements

(b) Authentication, authorization and auto configuration

(c) OAM

(d) Network management, security and control

Group (a) covers modifications and extensions for the datapath element or the Split Architecture itself. The other four groups deal with needed extensions of carrier-grade operation of ICT networks. These aspects are (b) authentication, authorization and auto configuration, (c) OAM in the sense of facilitating network operation and troubleshooting, (d) network management, including the aspects for the use of policies in network environments, and security and control of the behavior of the network and protocols.

Open issue identified in D3.1

In Deliverable D3.1 we described existing split-architecture approaches, such as ForCES, GMPLS/PCE and most importantly OpenFlow. An assessment of these approaches regarding their carrier-grade readiness revealed a number of issues and open questions that need to be considered for future carrier-grade Split Architectures. In D3.1 the following topics have been identified that require special attention in the current architecture study:

(e) Network virtualization

(f) Recovery and redundancy

(g) Multilayer control

(h) OAM functionalities

(i) Scalability

(e) While OpenFlow basically supports network virtualization, a number of open questions remain, e.g., how to ensure strict virtual network isolation and integrity, and how overlapping of address spaces should be handled. (f) Recovery and redundancy includes open question regarding resiliency of the Split Architecture not only with respect to the data plane of the network, but also with respect to controller and control-plane failures. (g) Multilayer-control issues include integration of circuit switching, as is done on the optical layers in the Split Architecture (also potentially in OpenFlow), and multilayer coordination for optimization or resiliency purposes. (h) As a basic carrier-grade feature, service management requires OAM functionality, which has thus been identified as an important issue to be solved. (i) Finally, scalability also needs to be considered, not only concerning the data plane, but also when proposing controller architectures, as it is another key characteristic of carrier-grade networks.

Missing features identified for architecture study in the current Deliverable (D3.2):

With the overarching requirement groups for carrier-grade Split Architectures as a basis, we will now further detail the requirements in order to isolate the missing features and thus enable in-depth study and discussion. The technical features studied in this deliverable fulfill many generally important and use-case-independent carrier-grade requirements. However, readers should note that the main use case in mind for this document is use case 1 from D2.1, “Access / aggregation domain of carrier networks.” The resulting list of study topics will then help us in proposing improvements to Split Architecture networks within isolated focus areas (i.e., missing features).

D2.1 requirement group (a) “Modifications and extensions to the datapath elements” has been broken down into two detailed features identified as particularly relevant in a carrier-grade networking context:

- The current OpenFlow draft (OF 1.1) is rather limited in terms of supported protocols, a fact that has also been recognized by the wider OpenFlow community [3]. We also identified *Openness and Extensibility* as a crucial feature for carrier-grade networks. Here we will discuss ways of how to extend OpenFlow to support a more complete, stateful processing on datapath elements in order to enable OF support for further technologies with high relevance to carrier networks, such as PBB, VPLS, PPPoE, GRE, etc. [R28-33 in D2.1]
- *Multilayer aspects* have been identified as a very important study topic both in D2.1 (a) and D3.1 (g). Our focus here is on the extension of OpenFlow in order to control non-Ethernet-based layer 2 technologies, especially the integration of circuit switched optical layers into OpenFlow (packet-opto integration).

Requirement group (b) of D2.1, “Authentication, authorization and auto configuration,” is covered by the broader topic of *Service Creation*. In the SPARC project, we will specifically study the service creation aspect for residential customers, e.g., by integrating the BRAS functionality into the OpenFlow-based Split Architecture to thereby enhance the flexibility of the handling of control functions in this partition of the network [R6-7, R10, R23 in D2.1].

Requirement group (c) of D2.1 “OAM” has also been identified as important matter in D3.1 (h). OAM, with the purpose of facilitating network operation and troubleshooting, is generally recognized as an important carrier-grade feature, e.g., as part of service management in the MEF definition of carrier-grade requirements [2]. In this document, both a technology-dependent OAM solution (i.e., MPLS BFD) and a novel technology-agnostic *Flow OAM* solution will be discussed, the latter targeting a generalized OAM Split Architecture. [R22-27, R60, R67 in D2.1]

“Network management,” as included in D2.1 requirement group (d) is also mentioned as a crucial carrier-grade requirement in [2] as part of service management. In order to study specific extensions separately, we divided the wide scope of this requirement into several subtopics.

- We identified topics related to *Network Management* as a relevant separate study item. For the present deliverable, we do not have any relevant mature input at this time. For the final architecture study in D3.3, however, we plan to propose a general framework required for implementation of network management functions in a Split Architecture environment. Fault and Performance Management covered by OAM [R22-27, R60, R67 in D2.1]; Configuration Management [R17-19, R21, R34-36, R38-40, R51-56, R61-62 in D2.1]
- *QoS* support is generally considered as a key requirement for carrier-grade networks [2] and will be discussed as a separate study item. [R11-14, R48-49 in D2.1]
- Reliability is commonly seen as one key attribute of carrier-grade networks [2], i.e., the ability to detect and recover from incidents within a 50 ms interval without impacting users. In D3.1 (f) we also identified recovery and redundancy mechanisms as important issues regarding a (partly) centralized Split Architecture. We therefore identified the overarching study topic of *Resiliency* as a key feature that requires specific attention in this deliverable. [R42, R47 in D2.1]
- OpenFlow enables Split Architectures by providing an open management interface to the forwarding plane of datapath devices, which allows centralized control of multiple OpenFlow datapath elements by a single control element. In order to facilitate this centralized network management operation, we identified automatic *Topology Discovery* as an important feature for carrier-grade Split Architecture networks. [R27, R51 in D2.1]

The second part of requirement group (d) in D2.1 “Security and Control” has been identified as a very important carrier-grade aspect. However, the SPARC partners have decided not to discuss security as an isolated study topic. Many relevant security questions, however, will be brought up when discussing other features identified above, such as slice isolation in multiprovider scenarios.[R20 in D2.1]

As pointed out in D3.1 (e), one crucial feature of future carrier-grade networks is *Virtualization and Isolation*, enabling multiservice (within the responsibility of one operator) and multi-operator scenarios on a single set of a physical network infrastructure. These scenarios are gaining increasing relevance due to financial or regulatory constraints, creating the need to share infrastructures, especially in the access/aggregation domains. An aspect of major importance in this context is an efficient *isolation* of slices so that different service groups and control planes deployed by different operators are shielded effectively from each other and any unintended intervention between services or operator control planes is avoided. [R1-4, R41-45 in D2.1]

As identified in D3.1 (i) *Scalability* is another key characteristic of carrier-grade networks. In our architecture study, scalability is an inherent feature of the proposed SPARC controller framework (see Section 2.4). Furthermore, we provide results on scalability characteristics in a both numerical and experimental study (see Section 4.5).

As a final additional topic we also study *Energy-Efficient Networking* as a special group of issues related to network management (d). There is a rapid increase in networking equipment and data center facilities which is being fueled by the increasing popularity of many Web and OTT (over the top) services. It is therefore of great importance to reduce the carbon footprint of these facilities in all aspects, including network optimization. We believe that an OpenFlow-based Split Architecture can provide functionalities to increase the energy efficiency of modern and future networks. [R59-60 in D2.1]

To sum up, in this deliverable we focus on the study topics listed in Table 1. The topics are derived from the use-case-specific requirements in Deliverable D2.1 and our initial architecture study in Deliverable D3.1, where we identified missing features concerning carrier-grade Split Architecture networks.

Section	**Study topic**	**D2.1/D3.1 Req. group**
3.1	*Openness and Extensibility*	(a)
3.2	*Virtualization and Isolation*	(e)
3.3.3	*OAM: technology-specific MPLS OAM*	(c), (h)
3.3.4	*OAM: technology-agnostic Flow OAM*	(c), (h)
3.4	*Resiliency*	(d), (f)
3.5	*Topology Discovery*	(d)
3.4	*Service Creation*	(b)
3.7	*Energy-Efficient Networking*	(d)
3.8	*QoS*	(d)
3.9	*Multilayer Aspects*	(a), (g)
4.5	*Scalability*	(i)

Table 1: Summary of study topics derived by D2.1/D3.1

Note that we do not claim this list to be exhaustive. The above listed features have been chosen solely based on our own assessment of feature importance. In earlier SPARC deliverables (D2.1, D3.1) we have identified further relevant topics (e.g., security aspects, mobility aspects including mobile backhauling, multicast, etc.). However, many of these additional study items proved to be too extensive for the resources available within the SPARC project and have thus been considered out of scope for this deliverable. Note that a discussion of issues related to network management is planned as part of the final SPARC architecture study in Deliverable D3.3.

The following subsections will discuss each missing feature listed above. We discuss the separate study items in varying levels of detail (as indicated in the table). For each topic we provide an introduction and the motivation by outlining current state-of-the-art solutions. We then describe our proposed improvements to an OpenFlow-based Split Architecture in order to enable the respective feature. Specific technical details, such as extensions to OpenFlow configuration procedures or OpenFlow protocol messages, will be documented in SPARC Deliverable D4.2 “OpenFlow protocol suite extensions.”

3.1 Openness and Extensibility

3.1.1 Introduction: the processing framework in OpenFlow version 1.0/1.1

A closer look at the OpenFlow framework and its present capabilities immediately reveals one of its major deficiencies: its rather limited set of supported protocols. Initially invented in a campus networking environment, OpenFlow 1.0 supports a basic set of common protocols and frame formats like Ethernet, VLAN, and ARP/IPv4. With OpenFlow 1.1, support for Multi Protocol Label Switching (MPLS) has been added to the OF specification, making OpenFlow more useful for deployment in carrier networking environments. However, it is clear that beyond MPLS support for other protocols may also be desirable, e.g., PBB, VPLS, GRE or, revisiting the access/aggregation use case as discussed in SPARC, support for the Point-to-Point protocol including PPP-over-Ethernet. However, patching more and more new protocols and capabilities into OpenFlow is not an elegant approach as it pollutes the protocol and makes the framework much heavier than necessary.

Adding support for a new protocol entails changes to a number of functional elements in OpenFlow, namely the base OpenFlow API and its commands and the matching structure, as well as the packet classifier and the processing logic (e.g., the latter is required for providing the adaptation functionality for the protocol, i.e., striping or adding the entire or changing fields in the protocol header such as the IP TTL value). OpenFlow defines an abstract view for a datapath element, effectively decoupling the logical from the various physical architectures adopted by hardware manufacturers. For building high-performance hardware based switching/routing devices, this logical architecture must be mapped onto the available hardware elements. Software implementations can be easily extended and adapted with changes to the functional elements within OpenFlow's logical architecture. However, any such extension should take into account constraints by state-of-the-art hardware designs [17][18]. Otherwise, it may prove useless quite soon.

While capabilities of state-of-the-art switching chip sets used within datapath elements determine and limit the set of supported protocols, another option for opening and extending the OpenFlow architecture is the coupling of various switching chip sets within a single datapath element. FPGA-based chips or network processor entities (NPE) provide a (more) general programming environment for executing complex protocol manipulation functions, but typically lack the high throughput required for a vast number of flows that can be achieved by dedicated ASIC-based chip sets. Coupling various hardware chip sets as processing and forwarding stages in a single datapath element seems to be attractive for a carrier environment, and we will propose an extension to OpenFlow's logical architecture to support such coupling effectively.

3.1.2 Processing entities, actions and virtual ports

The need for more advanced processing capabilities is already under discussion in the OpenFlow community (see the current discussion regarding OpenFlow 1.2 [3]). While evolving from version 1.0 to 1.1, protocol support for MPLS was added to the OpenFlow specification, but there is still a need for additional protocols.

OpenFlow encapsulates processing functions within lightweight actions stored in a flow mod entry being applied to a packet while traversing the forwarding logic. An action may be either part of an ActionList as defined in OpenFlow 1.0 or part of an ActionSet obtained from a specific instruction as done in OF 1.1. One of the extended features under discussion for upcoming OpenFlow versions [3] is the concept of a virtual port that serves as a container for encapsulating an arbitrary processing functionality while attaching the processing entity via a port's interface directly to the forwarding logic. Its functionality seems to be quite clear, however, its semantics are not: If a virtual port is treated as a physical port, it should, e.g., also handle a flooded message, which might not be desirable in all cases. On the other hand, the proposed encapsulated processing entity is in principle a filtering device without port semantics that applies some processing logic onto traversing packets either from left to right or vice versa. Contrary to the stateless actions defined so far within the OpenFlow specifications, such a processing instance may host stateful processing logic, i.e., the state among consecutive packets may be stored permanently. In principle, a processing entity is a generalized action that encapsulates some processing logic, some state-based on the history of packets that were handled by this action and associated session state. In the context of the access/aggregation use case and PPP-over-Ethernet, the session state is uniquely identified by the 16-bit PPPoE session identifier, which is one example highlighting the need for permanent stage storage.

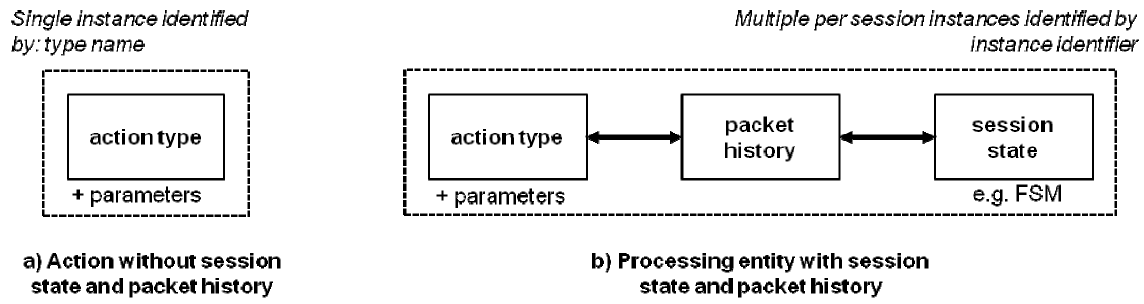

Single instance identified by: type name
action type
+ parameters
a) Action without session state and packet history
Multiple per session instances identified by instance identifier
action type
packet history
session state
+ parameters
e.g. FSM
b) Processing entity with session state and packet history

Figure 9: Actions vs. Processing Entities

Introducing processing entities implies a change to the OpenFlow processing framework. So far, an action has been referred to by its type name, as there was only a single instance deployed for handling all different flows and sessions. With processing entities, some session state is required for the processing logic, and this session state must be referred to by the action identifier. Thus, an action cannot be identified solely by its type name, rather the session identifier (=a specific processing identifier naming action type + session state) is required, as multiple instances of a processing entity may exist in parallel. This makes processing entities persistent entities in a datapath element that follow a typical CRUD life cycle (create, read, update, and destroy). A processing entity namespace must be defined, e.g., by using a unique 32-bit identifier, in order to identify individual entities. A processing entity may be managed dynamically by some counterpart in the control plane. The processing instance identifier links both processing instance and control entity to each other. The “update” message enables the control entity to control the processing instance (similar to a typical ioctl used under most Unix flavors of operating systems). In addition, a notification channel should exist in order to notify the control entity of special conditions, e.g., state changes in the processing entity’s state machine.

Obviously, some extensions to the OpenFlow base API for managing processing entities (CRUD + notifications) will be required for stateful processing instances. We will omit an exact definition of the necessary protocol extensions and their syntax here and leave this task to the OpenFlow standardization forum. However, in OpenFlow 1.1 a new functional element named “group table” has been introduced, consisting of a set of group entries. A group entry contains an ordered list of action buckets, each capable of carrying sets of actions. Group table entries are permanent entities and may be used as containers for hosting stateless actions and stateful processing entities.

Let us return to the virtual port concept introduced at the beginning of this section. Is there really a need for introducing another processing framework concept in OpenFlow in addition to an extended actions concept that covers stateless and stateful entities? Let us consider a simple example of an IP router device that is emulated by an OpenFlow datapath element. An IP router consists of transport endpoints (e.g., Ethernet transport endpoints, but other transport technologies may also be used), each identified by an IP address. Emulating a router’s behavior requires several processing steps as shown in Figure 10.

On reception, the IP header TTL field is decremented by one (let us assume it does not reach the value zero) and the router queries its routing information base (RIB) for the next IP hop. Once this next hop has been determined, the appropriate next transport domain and the next hop’s transport address in this domain are determined by querying the forwarding information base (FIB). The router queues the packet on the outgoing port and the Ethernet source address is replaced by the Ethernet address the router has been assigned in the next hop’s transport domain. OpenFlow mimics this behavior with the following actions: “decrement IP TTL” and “set DL source address” and “output packet on port.” Figure 10 also depicts the IP router’s operation with a hypothetical datapath element where all processing is done with virtual ports (the “output” action is the only exception to this rule). Both the virtual port and the action-based processing concept emulate the identical behavior for an IP router. We conclude from this simple example that in general any type of processing can be done within the actions framework and virtual ports can be, in principle, safely ignored.

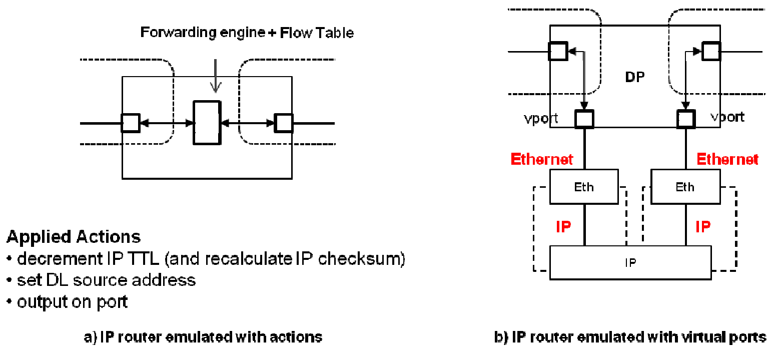

Forwarding engine + Flow Table
Applied Actions
• decrement IP TTL (and recalculate IP checksum)
• set DL source address
• output on port
a) IP router emulated with actions
DP
vport
vport
Ethernet
Ethernet
Eth
Eth
IP
IP
IP
b) IP router emulated with virtual ports

Figure 10: Emulation of an IP router with actions and virtual ports

Another aspect of virtual ports under discussion is packet generation. We conclude that virtual ports are not a mandatory prerequisite for injecting traffic into datapath elements. For packet generation a mechanism similar to a “Packet Out” message as sent by a control entity may be used by a processing entity to inject packets into a datapath element instead of originating them from a virtual port. For packet reception a flow mod entry may redirect the packet to the appropriate processing instance by using a new action “Process” as proposed below instead of redirecting packets into a virtual port.

Action “Process”: When executed this action redirects packets to a specific processing entity instance. A Action “Process” decouples a flow_mod entry from a processing entity. All OpenFlow 1.1 action-specific constraints also apply to the Action “Process” i.e., only a single action of any type may reside within an ActionSet. OpenFlow 1.1 defines an ordered list of actions, i.e., all actions within an ActionSet must be reordered according to this ordering policy before actually executing the ActionList. All “Set” related actions (lightweight processing) are executed after decrementing TTL values (position #5) and before any QoS-related actions are applied (position #7), therefore Action “Process” should occur before or after position #6. No restrictions apply to using Action “Process” inside a group table entry.

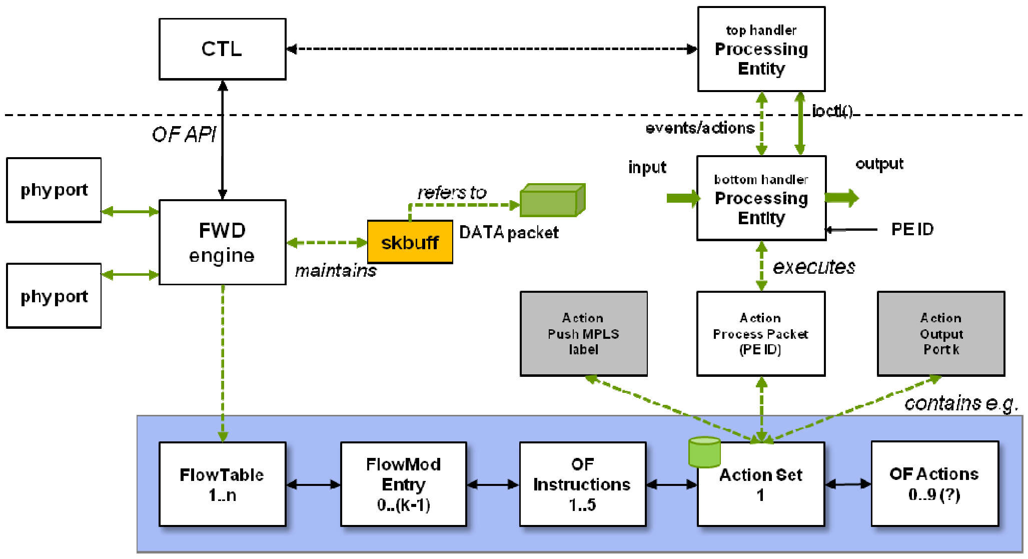


Figure 11: Proposed logical OpenFlow architecture

A processing entity is in principle logically split into two halves (see Figure 11): A top handler resides in the control plane while the bottom handler is located within the datapath element. For synchronizing state both handlers exchange events and actions among each other, effectively initiating state transitions. Note that both top and bottom handlers may be NULL handlers, i.e., all processing is done entirely either in the control plane (bottom handler is empty) or no controlling instance exists at all (top handler is empty). Control-plane-only processing moves all packets of a flow into the “slow path,” thus presumably degrading the flow’s forwarding performance significantly.

Each processing entity must be identified by a unique identifier (the “Processing Entity ID (PE ID)”). Upon execution of an Action “Process,” the packet is logically injected into the processing entity (in fact, it will obtain a reference to the buffer the packet is currently stored in). After completion, the next action will be executed.

3.1.3 Some (very brief) datapath design considerations

OpenFlow defines a generalized architecture for datapath elements. Hardware manufacturers map their physical hardware designs onto this generalized datapath architecture. In the previous section we discussed some options for extending OpenFlow’s processing framework. A hardware designer must decide how to integrate stateful processing entities in the OpenFlow architecture. Figure 12 depicts the common options for hardware designs for a datapath element.

Option 1: Most devices provide some general purpose execution environment as an embedded system running a common operating system. In this computing device a software-based OpenFlow-compliant datapath element may be deployed. Software environments are the most flexible execution environments; however, they may be limited in overall performance.

Option 2: Another common hardware design is the coupling of an embedded system with a specialized switching chip set. Various manufacturers offer specialized silicone dedicated to fast packet switching. Such ASIC-based designs also offer some processing capabilities like MAC rewriting, decrementing TTL values, recalculating header checksums, etc. Typically, these chip sets cannot be extended with additional processing capabilities. Advanced Processing entities may be hosted in the general computing environment, and the ASIC switching chip set may be used as some hardware accelerator for specific flows with limited processing requirements.

Option 3: A number of general purpose network processors are available from manufacturers today that offer the necessary degree of flexibility for conducting arbitrary processing tasks, but they are slower compared to specialized switching chipsets. However, a network processor may relieve the embedded system of some processing tasks.

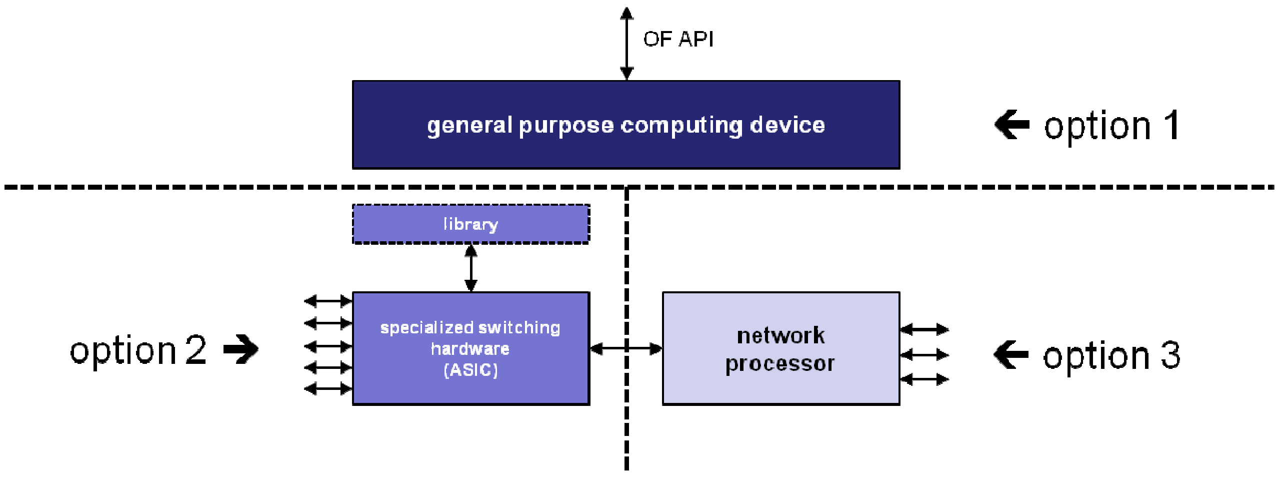

OF API
general purpose computing device
option 1
library
specialized switching hardware (ASIC)
network processor
option 2
option 3

Figure 12: General execution environment, hardware switching chip sets, and network processors

Diving into details of ASIC-based switching chip sets or discussing the best deployment strategy of network processors is beyond the scope of this deliverable, but from an architectural point of view, a number of data-path-element designs may be applied. We can expose each of the hardware chip sets toward the control plane and have a control entity deal with the complexity of moving flows between the various chip sets so that processing is done on the appropriate datapath. As an alternative, we can hide this complexity by integrating several chip sets and network processors and by emulating a more capable single datapath element. In the latter scenario, the interconnecting links between the switching chip sets act as the backplane of the emulated datapath element.

A recursive structure as proposed in the generalized carrier-grade SPARC architecture may be used here: The lowest-layer control entity provides the necessary logic to route flows between the special-purpose hardware devices and takes care of available resources. Flow mod commands received from a higher layer control entity must be mapped to a suitable processing device according to the contained actions and processing directives. The exact structure of such an adaptation layer is beyond the scope of this deliverable and may be discussed in a later document.

3.2 Virtualization and Isolation

3.2.1 What is network virtualization?

Network virtualization as such is not a new idea, in fact, many existing implementations exist on multiple layers in the network stack. These existing techniques are applicable in many situations, e.g., to improve network resource utilization through sharing among different tenants, to provide a logical separation of traffic between different entities, to simplify network management, and to provide connectivity over untrusted networks.

For example, to provide end-to-end, point-to-point or multipoint-to-multipoint connectivity, there are a number of different Virtual Private Network (VPN) techniques. A VPN can be created in many ways, for example, on top of layer 2, layer 3 or even layer 4 networks to provide the user with an interface that emulates a direct or routed/switched connection at the specific layer. For example, VPLS operates at layer 2 and creates MPLS pseudo-wire tunnels between a number of provider edge routers; these routers then provide a layer 2 interface to various customers and switches the layer 2 traffic over the tunnels. From the customer’s point of view, the endpoints act as if they are connected to a standard layer 2 switch. Similar techniques exist for providing connectivity on top of many types of networks, as well as providing different kinds of connectivity. Almost all possible combinations exist: Ethernet over IP, IP over ATM, ATM over MPLS, etc. One important aspect of VPN services is that the customer or user of the service has no control over how the service is implemented – he is only aware of the endpoint connections.

From the provider’s point of view the techniques used to implement VPNs allow him to better network utilization, since multiple customers can share the network resources, including both links and routers/switches. Using, for example, Virtual LANs (VLANs) on the Ethernet layer or Virtual Routing and Forwarding (VRF) on the IP layer, it is possible to create multiple forwarding tables inside a switch or router, forwarding traffic in different manners depending on the forwarding table assignment. This capability can be used either as a mechanism for providing services such as VPNs, or as a way of simplifying management of the network by splitting it into multiple separated domains. Assigning traffic to the different forwarding tables inside a router/switch can be done in several ways, for example by adding a tag to the traffic that identifies which forwarding table should be used – this is the case with VLANs. VRF does not use any explicit tag: With VRF, traffic is assigned to forwarding tables based on other criteria such as which link it came from. The link may be an actual physical link, or a virtual link implemented using a tunneling protocol such as GRE, IPSEC, etc.

While there are a lot of techniques for virtualizing parts of the network, the nodes, the links, and for creating virtual connectivity between end-points, they are typically applied as separate solutions and not as a complete solution. OpenFlow, on the other hand, gives us the possibility of combining the existing techniques / concepts and provide a more comprehensive solution, namely a virtual network. Virtual network service would provide not only end-to-end connectivity, hiding all the details of its implementation, but would also give the customer complete control of how traffic is forwarded and handled inside the network, allowing him to treat it as a part of his own network.

An environment such as a multi-tenant WAN or a fixed mobile-converged network, where multiple customers and competing service providers share a single physical network through network virtualization, imposes many requirements on the virtualization system. Not only must the system make sure that the traffic is separated between customers, ideally no information should be able to leak between the virtual "partitions" (unless previously agreed otherwise); it must also enforce any established SLAs. In addition, the system should be as flexible as possible in order to avoid costly interaction and coordination between the parties involved, thus reducing operational costs.

In the following sections we first investigate the derived requirements from Deliverable D2.1 and show how they are fulfilled by existing solutions for OpenFlow-based virtualization. We then investigate two aspects regarding virtualization – a) how to map or assign the customer's traffic into and out of the virtual networks and, b) what options are available for implementing the virtualization system itself and what changes are required to the protocol as well as the OpenFlow switch model in order to implement it in a carrier-grade manner.

3.2.2 Requirements for the virtualization system

In Deliverable 2.1 a number of requirements are derived and several of those apply directly to network virtualization, in particular these requirements:

R-2 The Split Architecture should support multiple providers.

R-3 The Split Architecture should allow sharing of a common infrastructure, to enable multiservice or multiprovider operation.

R-4 The Split Architecture should avoid interdependencies of administrative domains in a multiprovider scenario.

R-11 The Split Architecture should support best practices for QoS with four differentiated classes according to the definition documented by the Metro Ethernet Forum.

R-16 The Split Architecture must control the access to the network and specific services on an individual service provider basis

R-42 Data center virtualization must ensure high availability.

While most of these are quite general requirements, they highlight some issues with the existing implemented OpenFlow virtualization techniques: the FlowVisor and Multiple Switch Instances (a type of software-isolated virtualization). These existing solutions are described in more detail in Section 7.3-7.4 of Deliverable D3.1, but a short description is provided here.

The FlowVisor is a type of controller that acts as an OpenFlow protocol proxy, sitting between a number of controllers and switches, which forwards or denies OpenFlow commands from and to the controllers based on predefined policies. For example, the policies may restrict one controller to only send commands concerning a specific VLAN range and in that way virtualizes the network by restricting the view the different controllers have over the network, as well as what rules they may install, or put in another way, it restricts the connected controllers to only interact with a well defined part of the total flowspace.

Using Multiple Switch Instances, the switch runs multiple OpenFlow instances, allowing multiple controllers to connect to a single switch. Each of these OpenFlow instances is restricted by configuration to only use a subset of the ports on the switch, or a number of VLANs on a particular port.

More detail about the impact of the different derived requirements on the virtualization system:

R-2 – Multiple providers

Using a shared network, where multiple providers each have the ability of controlling part of the network via their own virtual network instance, increases the requirements for isolation in the virtualization system compared to the situation where a single provider uses virtual networks as a means to separate services. In the latter case, for example, information leakage between virtual networks or unfair bandwidth sharing between virtual networks is not a major problem since the single provider already knows the leaked information and is only "stealing" bandwidth from himself. In the first case, both of these problems are major issues and thus the virtual networks need to be strictly isolated from each other.

R-3 – Multiple services and multiple operators on a single link

Running multiple services, each in a virtual network, does not put very heavy requirements on isolation in terms of information leakage or fair use of the control channel. However, good QoS support may be important depending on what services are offered. This would include common QoS concepts like prioritization, traffic shaping, etc.

Sharing a single link for multiple operators requires some way of distinguishing packets belonging to the different operators, with more flexibility than reserving whole links per operator. This could involve operator-reserved VLAN tags, IP subnets, or MPLS label ranges, etc.

R-4 – Administrative interdependencies

When traffic enters the virtual network at the edge of the network, some mapping is necessary in order to map the incoming packets to the particular virtual network to which they should belong. This might be all traffic entering on a particular port, or traffic with some specific characteristics, such as certain VLAN tags, IP subnets, MPLS label ranges, etc. This is one area where flexibility is important in order to reduce the administrative interdependencies if, for example, two providers want to utilize or map the same VLAN range to their different virtual networks.

R-42 – High availability

A robust virtualization system should either be easy to duplicate or protect by other measures in order to maintain high availability, or it should be constructed in such a way that, from an availability point of view, it makes no difference if it is there or not.

Existing solutions have some problems meeting these requirements, as summarized in Table 2:

SPARC Requirement		**FlowVisor**	**Multiple Switch Instances**
Multiple providers	Level of support	**Poor**	**Implementation-dependent**
	Main reasons	OpenFlow lacks support for proper data plane isolation, good control plane isolation is possible	Data and control plane isolation may be good depending on the configuration and implementation
Multiple services	Level of support	**Poor**	**Poor**
	Main reasons	Lack of proper QoS support in OpenFlow	Lack of proper QoS support in OpenFlow
Multiple operators per link	Level of support	**OK**	**Poor**
	Main reasons	Only supports non-overlapping flowspaces	Typically separate ports per operator or non-overlapping VLAN tagging
Administrative interdependency	Level of support	**OK**	**OK**
	Main reasons	Flowspaces may not overlap, which requires coordination	Typically separate ports per provider or non-overlapping VLAN tagging
High availability		Additional single point of failure that has to be replicated	Same as without multiple switch instances

Table 2: Existing virtualization solutions with OpenFlow vs. SPARC requirements

As can be seen in this table, the current solutions do not provide adequate support for the requirements in many areas, therefore we have investigated what possibilities are available in order to create a system that fulfills the requirements to a higher degree.

3.2.3 Customer traffic mapping

Regardless of how the actual virtualization system is implemented, there is a problem at the edge between the customer and the provider of the virtualized network. How is traffic mapped from the customer network to the virtual network? At the edge nodes of the virtual network, the customers may each be connected to an interface of their own, or they may share an interface through some means, for example through some tunneling protocol, or simply by using different parts of the address space on some layer, like different VLANs or IP subnets. Additionally, this may not be consistent for all the customers' connections to the network, but may be different at various points of access. The virtualization system must be informed about how these connections are made in order to assign the traffic to the correct virtual network. The system may also require other virtualization related parameters such as the IP address, bandwidth requirements, priority of the traffic, and – depending on the virtualization system – what part of the flowspace should be reserved for the VN.

In the current FlowVisor implementation these parameters can be specified in a policy file on a per-switch basis, which may be a very time consuming and inflexible way of configuring the network. There is software to simplify the process, for example the Opt-In Manager, a web interface that allows users to define a network-wide flowspace they wish to control and to forward their request to an administrator for approval. While this simplifies the process, it still requires manual intervention for each slice.

One way of increasing the flexibility would be to allow automatic configuration either directly through the OpenFlow protocol itself or via an external protocol. Such a protocol could allow the network administrator to define some high-level SLA parameters such as maximum total bandwidth, as well as some limitations on the total flowspace the customer is allowed to use. Detailed configuration could then be left to the customer's controller which automatically (depending on the applications running) could define the VNs he needs, within his assigned flowspace, as well as how his traffic should be mapped to these VNs.

The same protocol could serve multiple purposes, but primarily it would be used to define how traffic should be handled at the edges of the network. However, it could also be used to specify the topology of the virtual network. For example, if the customer would prefer that the virtualization system abstracts the entire physical topology into an abstract node before presenting it to him, or if he would like a certain topology to be created using virtual links, that could be specified using the same protocol.

3.2.4 Network virtualization techniques for OpenFlow

A number of different models of how to perform virtualization in the context of OpenFlow can be imagined. All of these models contain three major parts that have to be virtualized: the control channel, the software part of the switch (the OpenFlow Instance, running on a general purpose CPU in the switch), and finally the hardware part of the switch. First we will examine some different virtualization models before we delve deeper into the options available for the three parts of the system.

Most essential to the virtualization process is some kind of translation / hypervisor unit that translates values between the real physical view and the different virtual views, a unit similar to a virtual memory management unit in computer architectures and the various kinds of hypervisors used to create virtual machines. As with computer virtualization hypervisors, there are many different options for how and where to implement it, but it has to be somewhere between the application logic and the physical fast-path hardware. In Figure 13 five different models are shown:

a) The FlowVisor approach, where the translation unit is placed outside of the switches and is shared across multiple switches and multiple controllers.

b) Each switch runs a translation unit that distinguishes between different connected controllers and performs the translation at the protocol level inside the OpenFlow instance on the switch.

c) Each switch runs multiple OpenFlow instances and the translation is performed between each OpenFlow instance and the fast-path forwarding hardware.

d) Each switch runs multiple OpenFlow instances and even the fast-path hardware has been split into several forwarding modules, one per OpenFlow instance. The "translation" is performed by restricting how ports are connected to the forwarding modules; the physical links may be split into multiple ports by using VLANs or MPLS labels, for example.

e) A model with multiple translation units, one responsible for virtualizing a single switch into multiple virtual ones, and another responsible for connecting multiple virtual switches and creating a virtual network representation. The first is responsible for virtualizing and isolating the switch resources, while the second connects to each virtual switch to create a virtual network topology, e.g., by presenting multiple switches as a single switch and managing tunnels that are presented as physical links similar to what is done in [29] and [30].

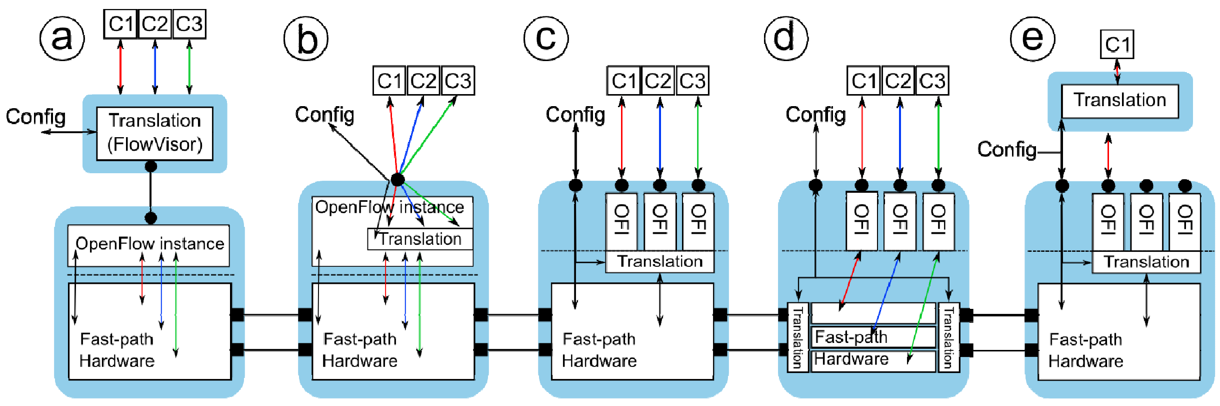


Figure 13: Different virtualization models for OpenFlow.

3.2.4.1 Control channel isolation

To ensure isolation between the control channels, the network that connects the controller to the switches (be it out-of-band or in-band) has to support some kind of rate limiting to prevent one virtual network from using all the bandwidth on the control channel, e.g., by forwarding a large amount of packets to the controller. The same is true in the other direction; the different controllers should not be able to disturb each other by transmitting large amounts of data to the switch. In the simplest scenario, with only one switch and one controller, these two problems could be taken care of in the switch and the controller respectively by limiting the amount of data they are allowed to transmit. However, if one increases the amount of switches in the system, one will reach a point where the aggregate amount of (still limited per VN per switch) traffic from all switches is enough to cause disruption.

Additionally, in both out-of-band and in-band cases, the control traffic may be competing with other traffic for network resources, for example different control traffic using the same out-of-band network, or data traffic in the case of in-band control channels. This traffic may be using its own flow control, like TCP, but this is not always necessarily true. If the only traffic on the control network is the different TCP connection between controllers and switches, the built-in flow control will react to congestion and limit the bandwidth use of each connection so that the TCP connections get an approximately equal share each. However, depending on how the different virtual networks are operating, they may likely have different bandwidth requirements for the control channel: One may rely heavily on sending data traffic to the controller for further analysis, whereas another may only be sending OpenFlow commands over the control channel. In both the out-of-band and in-band cases, network-wide QoS reservations for the control traffic can solve the problem and provide fairness in the use of control channel resources. For example, a tenant that needs to analyze data traffic in the controller will probably consume more bandwidth than one that does not have that need – TCP flow control is not enough to satisfy such requirements.

Depending on the virtualization model used, local (per VN in the controllers and switches) rate limiting may still be necessary. When using a FlowVisor, the control channels for the different VNs are multiplexed over a single TCP/SSL connection, so it is impossible for intermediate network nodes to look inside to differentiate between different flows and enforce QoS for the various controllers. On the other hand, if the switches allow the different controllers to connect and control the virtual networks using, e.g., different source or destination IP addresses or port numbers, the control traffic network can easily distinguish between the different connections and thus enforce QoS policies.

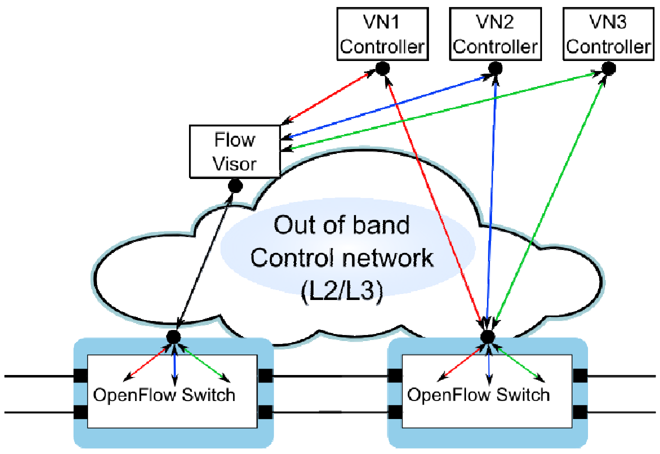


Figure 14: A single TCP/SSL session used by a FlowVisor (left) compared to a connection per controller (right).

3.2.4.2 Out-of-band vs. in-band control network

An out-of-band control network has many advantages compared to the in-band counterpart in the OpenFlow approach. It is simpler and easier to design in a reliable manner. However, out-of-band control networks might not be possible in some scenarios, for example in widely geographically distributed OLTs in service provider access networks. Even if an out-of-band network is possible, it would be more expensive than an in-band solution due to an entire extra network and extra ports on the hardware, in addition to increased configuration complexity. Given these considerations, in-band control channels are as important as out-of-band channels, if not more.

In an in-band scenario the control channel is vulnerable to misconfigured flow table entries since all packets, including control packets, traverse them. For example, the in-band solution implemented by Open vSwitch [31] requires the operator to preconfigured each switch with the neighboring switches' as well as its own MAC addresses and the controllers' IP addresses in order to install "invisible" FlowEntries that send control traffic to be processed by the local (non-OpenFlow) networking stack. This solution is quite fragile and configuration intensive, which could be operationally expensive in a large network. However, with a robust automated topology and controller discovery algorithm (see Section 3.5), even a quite complicated solution could be managed. It is important to note that such an automated discovery algorithm does not necessarily have to operate on all the virtual networks' control networks – these could be managed by the virtualization system itself.

In the in-band case, it is also very important that sufficient QoS support is available so that control traffic can be guaranteed priority over data traffic, in order not to lose control over switches in case of data traffic overload. This should be kept in mind when designing the QoS extensions discussed in Section 3.8. Additionally, QoS should not only be applied to control traffic entering the switch from the in-band channel, but also to outgoing traffic, without the traffic necessarily passing through the flow table when leaving the switch. This may require that an invisible / non-mutable high-priority queue be reserved for traffic from the local network stack.

3.2.4.3 OpenFlow state isolation

By *OpenFlow Instance* we refer to the software part of the OpenFlow implementation of a switch that is running on the normal switch CPU which is responsible for processing OpenFlow messages and performing anything else that is not handled by the hardware itself (e.g., implementing some actions, keeping statistics, etc). Depending on the virtualization model and the implementation, it may also contain a translation unit that has to keep the different connected controllers separated. However, running multiple OFIs as separate processes decreases the risk of them affecting each other, e.g., if misconfiguations that cause packets to go to the wrong VN. This could still occur in the datapath or in the translation unit, but the less space the different VNs share, the less likely it should be. Additionally, running them as separate processes reduces the risk of problems spreading from one VN to another through other software bugs that cause crashes. Depending on the implementation the OFI could for example, contain memory tables that could overflow, or it could perform processing using more than its fair share of switch CPU. If the operating system supports it, this can be limited through further isolation by a virtualization system, such as the lightweight Linux Containers [32] which enables strict management of switch memory and CPU usage by the OFIs.

Local virtual ports can be used to send flows to the switch's local networking stack, e.g., to implement an in-band control network. In order to support the multiple channels corresponding to each OFI, a predetermined range of Local port numbers could be used, one for each OFI. Alternatively, if the different OFI instances are multiplexed through a single Local virtual port using their respective IP addresses, it could lead to fairness and starvation issues.

The translation unit within the OFI is responsible for mapping the virtual identifiers to their physical counterparts as well as enforcing the separation between the different virtual "identifier spaces." The complexity of the translation unit depends on the choice of the virtualization model. For example, if per-VN encapsulation is used, similar to model (d) in figure [virt-models], the payload of the Packet_In and Packet_Out OpenFlow messages needs to be modified based on the port and VN they are associated with, and they should have the correct QoS markings applied for fair resource allocation. In contrast, correct translation and mapping of per-VN port numbers and port counters apply all the virtualization models discussed in Figure 13. For example, for statistics monitoring, per-VN port counters could be implemented through per-VN flow table entries [33].

One of the OFIs has a privileged role and is allowed to configure the translation unit as well as all the flow tables (without going through the translation unit). This process should be running at a higher priority than the others in order to increase the likelihood that the network owner can control the switch in case of issues with the other OFIs. Configuration of the flow tables can be performed through the existing OpenFlow protocol, which could be extended with, e.g., a vendor extension to also be able to define the VNs and configure the translation unit. Such an extension would contain, e.g., the VNID, the ports that should be included, which tables the VN should use and similar information.

3.2.4.4 Isolation in the forwarding plane

In the forwarding plane several things should be virtualized and isolated from each other, firstly the actual entries in flow table(s), the physical links (or parts of them) and any processing modules (e.g., OAM modules as presented in Section 3.3, or other processing resources reachable by the *Action Process* presented in Section 3.1). Exactly where the border between the OpenFlow instance and the hardware is not very clear and depends on each implementation.

There are two approaches to keeping the virtual networks separated on the link level: partitioning and encapsulation. With partitioning the link is split into multiple partitions by slicing the total flowspace, one slice per virtual network, for example by reserving a part of the total VLAN or IP address range per virtual network. To ensure that this separation is effective, all the virtual networks must be sliced based on the same part of the flowspace; one cannot slice some based on VLAN and some based on IP addresses since it can then be ambiguous as to which virtual network a packet with both a VLAN and IP header belongs to. However, these restrictions are strictly "link local," e.g., the same VLAN could be reused for a different virtual network on another port on the same switch. Additionally one could apply translation to the field used for distinguishing the different virtual networks, for example, if traffic belonging to two different virtual networks enters a switch from different ports, but with the same VLAN, one of them could be translated to a free VLAN when entering the virtual network and translated back to the original value when the packet leaves the virtual network.

With encapsulation some kind of encapsulation is used to separate the traffic when it traverses a shared link. Before transmitting a packet, each switch adds some kind of link-local encapsulation per virtual network; the encapsulation is used at the receiver to assign the packet to a certain virtual network and then removed before processing. This leaves each virtual network with a complete flowspace that has not been reduced to create separation, thus creating more flexibility at the cost of the extra processing needed to add and remove the encapsulation per hop. It also requires the switches to support some kind of encapsulation protocol such IPSEC, L2TP, GRE, MPLS, or PBB. Currently the only supported encapsulation method is MPLS, which is supported with the Ericsson extensions to OpenFlow Version 1.0 and by default in OpenFlow Version 1.1.

When it comes to the flow table(s) there are other approaches as well, basically either slicing or "splitting." With slicing the same restrictions are applied as in the case of the sliced link sharing above. Each virtual network is only allowed to insert FlowEntries that 1) a have a match that is within the specific flowspace assigned to the particular virtual network, and 2) have actions that cannot move the packet into a different virtual network, for example by changing the VLAN. Here again the approach has similar drawbacks as in the previous case – one needs to make sure that the sliced flowspaces do not overlap, which causes a loss of flexibility.

With splitting the flow table is cut into multiple pieces, either logically through some internal logic that is specific to the particular brand of switch, or physically. The logical split is of course more interesting since it is more flexible and cheaper (by not using multiple hardware components). With OpenFlow Version 1.1 a logic split can be constructed by restricting the use of the multiple tables, for example by assigning each virtual network access to ten tables instead of the full range. While the protocol cannot address more than 2^8 tables, this does not limit the potential number of VNs to 256. The flow table identifier has a local meaning in each OpenFlow protocol session. If the switch hardware can support more than 256 flow tables, they can all be used through translation, but **not** by a single OpenFlow session.

Processing units, like virtual ports and *Action Process actions* (described in Section 3.1), can be shared between virtual networks if they are able to distinguish between packets from different virtual networks. For example, in the case of the BFD OAM module discussed in Section 3.3.3, the module uses a Maintenance End Point Identifier (MEP-ID) to distinguish between different BFD sessions when transmitting and receiving packets. An identical MEP-ID could exist in several virtual networks and therefore it is necessary to combine the MEP-ID with some kind of virtual network identifier to create a unique session identifier. This most likely applies to all kinds of processing units, as well as other related things, such as packet counters, which also should be kept on a per-virtual-network basis and not (only) globally within each switch.

Thus far we have only discussed the ability to distinguish between different virtual networks and not mentioned enforcing fair use of resources. This could be handled through the standard QoS mechanisms (classification, metering and coloring, policing, marking, queuing and shaping) by applying them on a per-VN basis. However, support for QoS functionality is currently fairly limited in OpenFlow, so we do not go into detail here as this is discussed in Section 3.8. With the improvement suggested there it would be possible to create guaranteed per-VN bandwidth profiles.

3.2.5 Improvement proposal

By combining the ideas presented above, we can construct a complete virtualization system that provides a high degree if isolation, both in the datapath and on the control level, that is flexible and requires a low degree of interaction between the operators. The system presented here is shown in Figure 15 and is based on model e) in Figure 13.

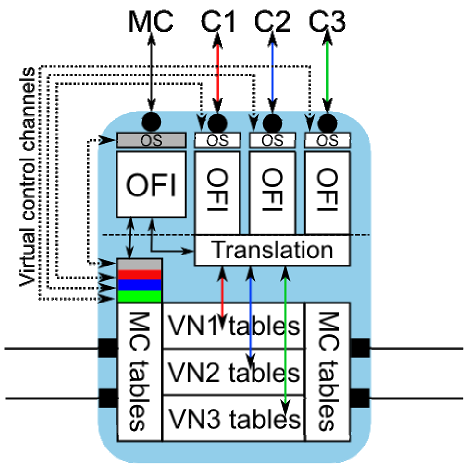


Figure 15: A combination of an encapsulated forwarding plane with flow table splitting, an in-band Ethernet-based control network and multiple isolated OpenFlow Instances. Translation is performed between the OpenFlow Instances and the fast-path hardware, and is configurable through a privileged OpenFlow Instance.

The datapath is built from model e) in Figure 13 to which we add encapsulation-based link separation combined with flow table partitioning. This results in a very flexible datapath virtualization system through the application of resource allocation and QoS mechanisms discussed in Section 3.8. This can be achieved by dedicating the ingress and egress flow tables on the datapath for this purpose. Link separation can be achieved, for example, through specific, predetermined MPLS labels, where at each link they are used to encode the VNID (MPLS replaces the existing EtherType with its own, therefore each VNID needs multiple MPLS labels assigned to it, one per EtherType used).

The OFIs could be separated through the virtualization mechanisms of the OS, and they connect to the datapath hardware through the translation unit. The OFI that handles the *Master Controller* (MC), usually operated by the network owner, configures the translation unit as well as all the ingress and egress flow tables (shown as MC tables in Figure 15). This OFI should run at a higher priority on the local switch CPU so that the MC communications with the switch are given higher priority in order to handle special events such as failure scenarios. Configuration of the MC tables can be performed through the existing OpenFlow protocol, with minor restrictions enforced for each VN controller. The translation unit also needs configuration in order define the different virtual networks, and this can be achieved through non-OpenFlow channels or by OpenFlow protocol extensions containing the definition of a particular VN on a switch (e.g., the VNID, MPLS labels belonging to it, the ports belonging to it, etc.).

The control network could be implemented as an in-band IP network and separated into two parts: the master control network and the virtual control channels. The virtual control channels are managed by the MC which is responsible for routing, establishing QoS, and reconfiguring the control network in case of failures, topology changes etc. The master control network, however, needs to be bootstrapped through manual configuration or via a dynamic discovery protocol, for example the one presented in Section 3.5.

3.3 Operations and Maintenance (OAM) Tools

OpenFlow, as an attractive element of the open split-architecture platform, aims to eliminate the weaknesses caused by proprietary hardware design and supply, i.e., the single sourced system software supply closely coupled with equipment vendors. However, this open but immature protocol still has significant drawbacks compared to the closed, traditional protocols: It only supports partial configuration of the switches. One important aspect which cannot be provisioned through the current OpenFlow solutions is OAM.

In earlier SPARC Deliverables (D3.1 and D2.1) OAM has been identified as an essential requirement for carrier-grade operator networks in order to facilitate network operation and troubleshooting. For instance, D2.1 proposes the following requirements:

"For operation and maintenance, a number of functions are already widely defined and implemented. This includes identification of link failures, connectivity checks and loopbacks in the data plane. In addition, it should be possible to send test signals and measure the overall performance. Standards to be supported are ITU-T Y.1731, IEEE 802.3ag/ah and respective IETF standards for IP and MPLS. Interfaces of switches, routers, etc., provide additional information for monitoring and operation like link down, etc. This includes monitoring of links between interfaces as well."

Specifically, three requirements regarding Operations and Maintenance have been derived in D2.1:

R-25 The Split Architecture should support OAM mechanisms according to the applied data plane technologies.

R-26 The Split Architecture should make use of OAM functions provided by the interface.

R-27 The Split Architecture shall support the monitoring of links between interfaces.

We see a few challenges with OAM in a Split Architecture:

- How to map traditional, technology-specific OAM elements to Split Architectures, i.e., how to integrate specific OAM tools (e.g., Ethernet OAM, MPLS OAM and IP OAM) into an OpenFlow-based Split Architecture.
- How to provide a generalized, technology-agnostic flow OAM for Split Architectures.
- Given that virtualization enables multi-operator scenarios, how can we support a multi-carrier service OAM and provide automatic OAM configuration in this environment?

In the following subsections, we will discuss the challenges listed above. We will start by giving an overview of the most prevailing existing packet OAM toolset, namely Ethernet Service OAM (IEEE 802.1ag, ITU-T Y.1731)) and MPLS-TP OAM (BFD, LSP-ping or ITU-T Y.1731 based). The we will discuss how the traditional OAM functions and roles map to our split-architecture design. As an example of how technology-specific OAM tools can be integrated, we will then outline the MPLS BFD solution as implemented in the SPARC prototype. Finally, we will go one step further and consider the case of a novel technology-agnostic flow OAM by providing an initial architectural solution.

3.3.1 Background: existing OAM toolset

According to the state-of-the-art design, OAM toolsets are technology-dependent, i.e., they are bound to the specific data plane technology they have been developed for (e.g., Ethernet or MPLS(-TP)). As we will outline in the following subsections, the existing toolsets have very similar purposes, such as the detection of a connectivity loss or a violation of a delay constraint. However, the methods and architectures used to reach those goals are different, which makes the various OAM solutions incompatible.

3.3.1.1 Ethernet Service OAM

Ethernet Service OAM is a key component of operation, administration and maintenance for carrier Ethernet based networks. It specifies protocols, procedures and managed objects for end-to-end fault detection, verification and isolation. Ethernet service OAM defines a hierarchy of up to eight OAM levels, allowing users, service providers and operators to run independent OAMs at their own level. It introduces the concept of a Maintenance Association (MA) that is used to monitor the integrity of a single service instance by exchanging CFM (Connectivity Fault Management) messages. The scope of a Maintenance Association is determined by the Management Domain (MD), which describes a network region where connectivity and performance is managed. Each MA associates two or more Maintenance Association Endpoints (MEP) and allows Maintenance Association Intermediate Points (MIP) to support fault detection and isolation.

The continuity check protocol is used for fault detection. Each MEP can periodically transmit connectivity check messages (CCM) and track CCMs received from other MEPs in the same maintenance association.

A unicast Loopback Message is used for fault verification. It is typically performed after fault detection. It can also confirm successful initiation or restoration of connectivity. Loopback Messages (LBMs) are transmitted by operator command. The receiving MP responds to the LBM with a unicast Loopback Reply (LBR).

A multicast Linktrace Message (LTM) is transmitted in order to perform path discovery and fault isolation. The LTM is transmitted by operator command. The intercepting MP sends a unicast Linktrace Reply (LTR) to the originator of the LTM. The originating MEP collects the LTRs and provides sufficient information to construct the sequence of MPs that would be traversed by a data frame sent to the target MAC address.

There are usually two types of Ethernet service performance management: Delay Measurement (DM) and Loss Measurement (LM).

DM is performed between a pair of MEPs. It can be used for on-demand measurement of frame delay and frame delay variation. An MEP maintains the timestamp at the transmission time of the ETH-DM frame.

LM is performed between a pair of MEPs, which maintain two local counters for each peer MEP and for each priority class – TxFCl: counter for data frames transmitted toward the peer MEP; RxFCI: counter for data frames received from the peer MEP. OAM frames are not counted.

3.3.1.2 MPLS(-TP) OAM

MPLS-TP OAM is still intensely discussed by the community, and is thus documented in IETF drafts only. Generally, there are two types of MPLS-TP OAM under discussion:

The first method is to enhance the available MPLS OAM toolset (LSP Ping and BFD) to meet the OAM requirements of MPLS-TP.

LSP Ping provides diagnostic tools for connectivity checks of MPLS tunnels, testing both data and control plane aspects. LSP Ping can be run periodically or in on-demand fashion. Essentially, LSP ping verifies the status of a complete LSP by inserting "echo" requests into the MPLS tunnel with a specific IP destination address. Usage of a dedicated address prevents the packet from being routed further at the egress LSR of the MPLS tunnel. On reception of an "echo" request, the designation LSR responds by sending an "echo" reply back to the originator of the request. For MPLS-TP, LSP Ping should be extended with tracing functionality (traceroute i.e., link-trace). Furthermore, LSP Ping should support point-to-multipoint LSPs and should be able to run without an underlying IP.

BFD (Bidirectional Forwarding Detection) is used for very fast proactive detection of data plane failures by connectivity checks. BFD is realized by "hello packets" exchanged between neighboring LSRs in regular, configurable intervals. If hello packets are not received as expected, a connectivity failure with the particular neighbor is detected. BFD packets can also be used to provide loopback functionality with replied received packets at the neighboring node. For working with MPLS-TP, BFD will be extended, e.g., for working without reliance on IP/UDP functionality.

The second method is to develop MPLS-TP OAM based on Y.1731 Ethernet service OAM. The basic idea is that Y.1731 specifies a set of OAM procedures and related packet data unit (PDU) formats that meet the transport network requirements for OAM. The actual PDU formats are technology agnostic and could be carried over different encapsulations, e.g., MPLS Generic Associated Channel.

3.3.2 Mapping OAM element roles to Split Architectures

While the OpenFlow protocol itself does not support any OAM solutions, it does not prohibit adopting technology-specific toolsets. For example, to support Ethernet OAM, Ethernet OAM modules could be configured in every node as in the following figure:

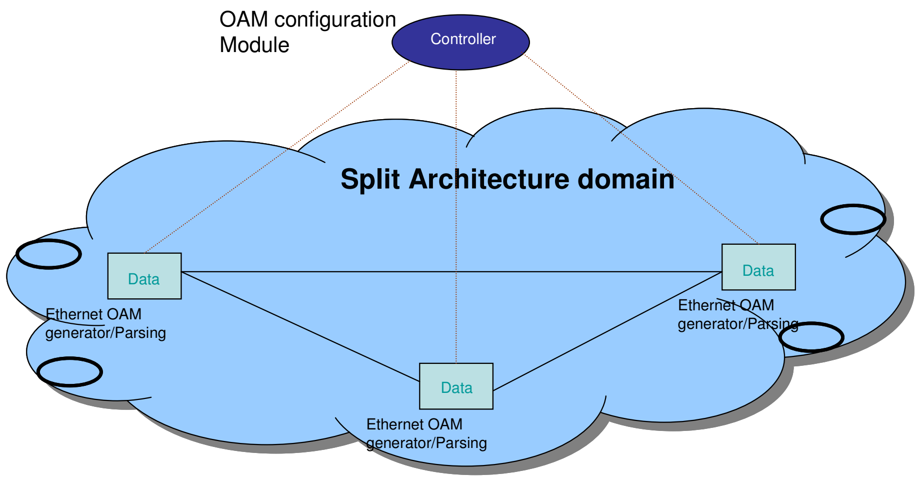

OAM configuration Module
Controller
Split Architecture domain
Data
Ethernet OAM generator/Parsing
Data
Ethernet OAM generator/Parsing
Data
Ethernet OAM generator/Parsing

Figure 16: Split Architecture OAM Configuration

This configuration works well with Ethernet traffic flows. However, a main feature of an OpenFlow-based Split Architecture (e.g., OpenFlow) is the support of various traffic flows such as Ethernet, MPLS or IP. Thus, in the case of MPLS traffic flows, the Ethernet OAM module would become ineffective and an additional MPLS OAM module would have to be added to the datapath elements. As a result, datapath elements would contain different OAM configuration modules and OAM generator/parsing modules for different traffic flows. This contradicts the original notion of Split Architectures having simple datapath elements, and this type of OAM integration would significantly increase the complexity of data forwarding devices. An additional problem with multiple OAM modules for different flow technologies is their different configuration models, which further complicates both datapath and control elements. Furthermore, OpenFlow allows routing and switching of packet traffic which does not strictly follow Ethernet switching, MPLS forwarding, or IP forwarding. The OpenFlow switch allows "flow switching," i.e., switching is done based on arbitrary combinations of bits in the flowspace, which currently mainly consists of the Ethernet, MPLS and IP header fields. Finally, a general problem for OAM integration into current OpenFlow scenarios is the lack of an explicit OAM indication which allows dispatching of OAM packets without interfering with OpenFlow match structures.

This section is devoted to proposing improvements to the OpenFlow switch model to implement OAM feature sets. During the design phase we were faced with two contradictory aspects:

- Compatibility with legacy OAM toolsets.
- Avoiding the need of developing and running dozens of toolsets at the same switch.

The first aspect is relevant when such logical connections are monitored that span OpenFlow and non-OpenFlow domains, as done in the integration of IEEE 802.1ag OAM to OpenFlow by SARA Computing & Networking Service [20]. In such cases the data plane switch must implement all technology-specific OAM features of the data layer. Considering the monitoring of OpenFlow domain internal connectivity only, this former aspect becomes irrelevant. As the datapath in OpenFlow domain is layering-agnostic, the goal is to provide a unified flow OAM using only a single OAM module in order to monitor all flows in this specific domain.

In the upcoming two subsections we will present improvement proposal methods for both, technology-dependent and technology-agnostic OAM toolsets. As an example of a technology-dependent solution, we present MPLS BFD for OpenFlow, which has also been implemented as part of the SPARC demonstrator. The second, more novel solution is technology-agnostic and provides a unified, generic OAM module for all flows. This solution is outlined only as an architectural concept, and many details relevant for implementation are still the subject of future work.

3.3.3 MPLS BFD-based Continuity Check for OpenFlow

OpenFlow MPLS BFD-based continuity check (CC) uses the virtual ports concept. A virtual port is identified with a port number within the port table exactly like a physical port, but additional actions not part of the standard OpenFlow ActionSet may be performed on the packet. Virtual ports can be chained. A virtual port chain on the input side resides between the input port where the packet enters and the first flow table, whereas a chain on the output side starts after the flow table and ends at a physical port where the packet exits. The output side chains can be addressed by entries of any flow tables. OpenFlow 1.0 included a few built-in virtual ports for actions such as forwarding a packet to the controller. But those virtual ports are reserved for specific purposes and cannot be configured through the OpenFlow interface. An extension is defined for OpenFlow 1.0 that allows the dynamic creation and configuration of virtual ports in order to implement MPLS-related packet manipulation.

To reduce the amount of resources (e.g., timers) needed to generate monitoring packets, the OpenFlow MPLS protection switching makes use of the group concept to perform multicast. The protection switching also uses this feature to replicate single monitoring packets and then send them, after being modified to identify a single BFD session, out through multiple ports. Since the standard OpenFlow 1.0 neither supports the group concept nor includes any other multicast method, the group concept has been added to the MPLS-enabled OpenFlow 1.0 switch implementation.

3.3.3.1 Ingress-side BFD control packet insertion into the MPLS G-ACh control channel

Figure 17 illustrates how BFD control packets are generated and inserted into LSPs on the ingress side. A virtual port represents a BFD packet generator. The BFD packet generator generates a BFD packet template with a configured interval, for example each second, each millisecond, etc (see ① in Figure 17). The BFD template packet does not contain any information related to a particular BFD session, however the packet generator fills in the timing fields of the packet. The BFD packet template also contains the required ACH TLVs (with empty values) and the GAL MPLS label on top. The switch input source port is set to the BFD packet generator representing the virtual port number upon input to the flow table.

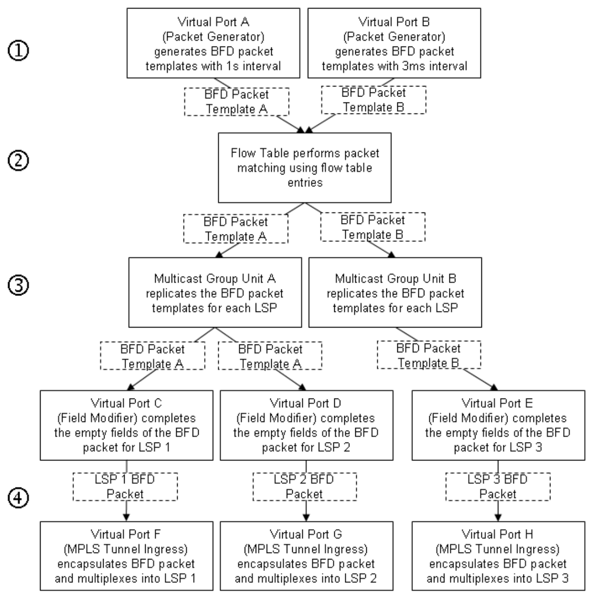

①
Virtual Port A (Packet Generator) generates BFD packet templates with 1s interval
Virtual Port B (Packet Generator) generates BFD packet templates with 3ms interval
BFD Packet Template A
BFD Packet Template B
②
Flow Table performs packet matching using flow table entries
BFD Packet Template A
BFD Packet Template B
③
Multicast Group Unit A replicates the BFD packet templates for each LSP
Multicast Group Unit B replicates the BFD packet templates for each LSP
BFD Packet Template A
BFD Packet Template A
BFD Packet Template B
Virtual Port C (Field Modifier) completes the empty fields of the BFD packet for LSP 1
Virtual Port D (Field Modifier) completes the empty fields of the BFD packet for LSP 2
Virtual Port E (Field Modifier) completes the empty fields of the BFD packet for LSP 3
④
LSP 1 BFD Packet
LSP 2 BFD Packet
LSP 3 BFD Packet
Virtual Port F (MPLS Tunnel Ingress) encapsulates BFD packet and multiplexes into LSP 1
Virtual Port G (MPLS Tunnel Ingress) encapsulates BFD packet and multiplexes into LSP 2
Virtual Port H (MPLS Tunnel Ingress) encapsulates BFD packet and multiplexes into LSP 3

Figure 17: Ingress-side OAM signaling generation

The template packet is sent to the flow table for matching (see ② in Figure 17). The flow table has been programmed with a flow rule that matches the incoming BFD packet template, specifically the “In Port” field matches the virtual port number of the BFD packet generator, and the MPLS field matches the GAL label number (13). The flow action forwards the packet to a multicast group that handles packets with the same generation interval as the BFD packet generator. If there are two packet generators with different intervals, the packets generated by them are forwarded to different BFD multicast groups (see ③ in Figure 17) based on the differing input port numbers.

When the multicast group receives a packet, it replicates it and forwards it to the output ports with which it has been configured. These virtual ports are field-modifier virtual ports (see ④ in Figure 17).

The field-modifier virtual port fills in the missing fields currently empty in the packet, i.e., the BFD packet fields for the particular BFD session: the timer values, the MEP-ID and the state variables, such as RDI indication. Thus, this virtual port represents the source functions of the MEP (source MEP). The MEP-ID is used to do a lookup in an associative data structure that contains all the BFD session information (state variables, descriptors, output virtual port number, etc.) for all the BFD sessions on the egress side of the LSP. Once the BFD packet has been completed, the packet is forwarded to an MPLS tunnel ingress virtual port, where the BFD control packet is treated in the same fashion as incoming user data traffic: The MPLS label for the LSP is pushed onto the label stack and the packet multiplexed into the MPLS LSP through the output physical port.

3.3.3.2 Egress-side BFD control packet processing

Figure 18 illustrates how BFD packets are processed on the egress side of the LSP. Incoming BFD packets enter the switch like any other packet on the same MPLS LSP (see ① in Figure 18). An incoming MPLS packet is matched in the flow table, and the packet is forwarded to an MPLS tunnel egress virtual port (see ② in Figure 18). This port is configured to pop the top label and examine any labels still on the stack. If the new top label is the GAL label, the packet is removed from the regular packet processing pipeline, but passed to the G-ACh module within the virtual port. The G-ACh module examines the channel type of the packet and the packet is sent to a Link Failure Detection module depending on the channel type (see ③ in Figure 18). The Link Failure Detection module performs a lookup in the BFD sessions’ associated data structure storage using the MEP-ID found in the BFD packet as the key (see ④ in Figure 18). If a session is found, the associated BFD session data and failure timer for the session are updated appropriately.

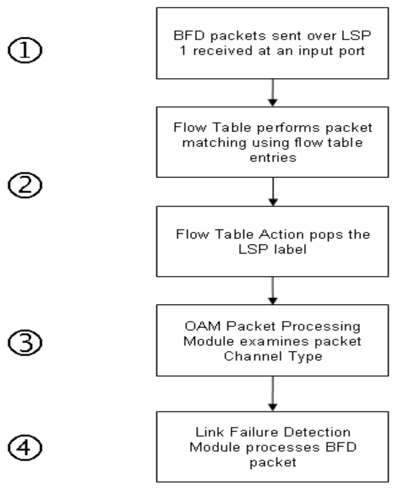

①
BFD packets sent over LSP 1 received at an input port
Flow Table performs packet matching using flow table entries
②
Flow Table Action pops the LSP label
③
OAM Packet Processing Module examines packet Channel Type
④
Link Failure Detection Module processes BFD packet

Figure 18 : Egress-side OAM signaling reception

3.3.3.3 BFD transmission timer updates

During its lifetime a BFD session uses at least two different transmission timer intervals, one before the session has been established (longer than 1 second) and another (depending on the failure detection resolution) once the BFD session has been established. This requires that a BFD session moves between different BFD packet generators, and that BFD group output ports are reconfigured when the transmission timer interval changes.

3.3.4 Technology-agnostic flow OAM

Technology-specific OAM solutions, as described in the MPLS BFD example above, have the advantage of straightforward integration with legacy domains in order to provide end-to-end monitoring of connections spanning multiple domains (including OpenFlow domains). However, technology-specific OAM solutions do not take the diversity of the OpenFlow flowspace into account, and target only specific flow types (e.g., MPLS flows in the BFD example). In order to support OAM functionalities for the entire flowspace, this approach would require a number of separate OAM tools (e.g., Ethernet OAM monitors Ethernet Traffic; MPLS OAM monitors MPLS traffic; IP OAM monitors IP traffic), which might lead to an unacceptable increase of complexity in the datapath elements.

Examining the different OAM technologies, we realized that they all have similar goals for the fault or failure in the network – detect, locate and report. In a Split Architecture domain, when the control plane is decoupled from the data forwarding plane, different traffic flows (Ethernet, MPLS or IP) are all treated equally by the data forwarding devices – and differences are only relevant to the controlling elements. Then the interesting question arises: In an OpenFlow-based Split Architecture environment, can we decouple the OAM monitoring traffic from the actual flow traffic (Ethernet, MPLS or IP)?

There are a few issues with decoupling OAM from regular flow traffic which need to be considered for a proposed solution. First, even if OAM traffic is decoupled, fate sharing needs to be ensured. Fate sharing means that OAM traffic needs to take exactly the same path as the actual flow traffic, i.e., it must go through the same link, the same node and use the same filtering table as the data traffic to be monitored. Secondly, there needs to be a way to explicitly indicate OAM packets in order to enable the datapath elements to handle OAM packets accordingly. And finally, some OAM may have technology-specific OAM requirements.

3.3.4.1 Independent OAM module

We propose an independent OAM module which is not associated with data traffic. This independent OAM module only generates or parses OAM PDUs (Protocol Data Units). To create a “fate sharing” path between the actual data traffic flow and OAM flow, we propose a Flow ID encapsulation/decapsulation module. This Flow ID module is associated with actual data traffic flow to be monitored so that the OAM traffic will have the same Flow ID and pass through the same link and the same node as the data traffic flow.

This general split-architecture OAM proposal has three advantages when compared to other solutions:

- Ubiquity: One OAM module supporting different traffic flows.
- Granularity: Supporting many services from a single carrier or many services from multiple carriers.
- Uniformity: Simplified and standard configuration and provisioning process.

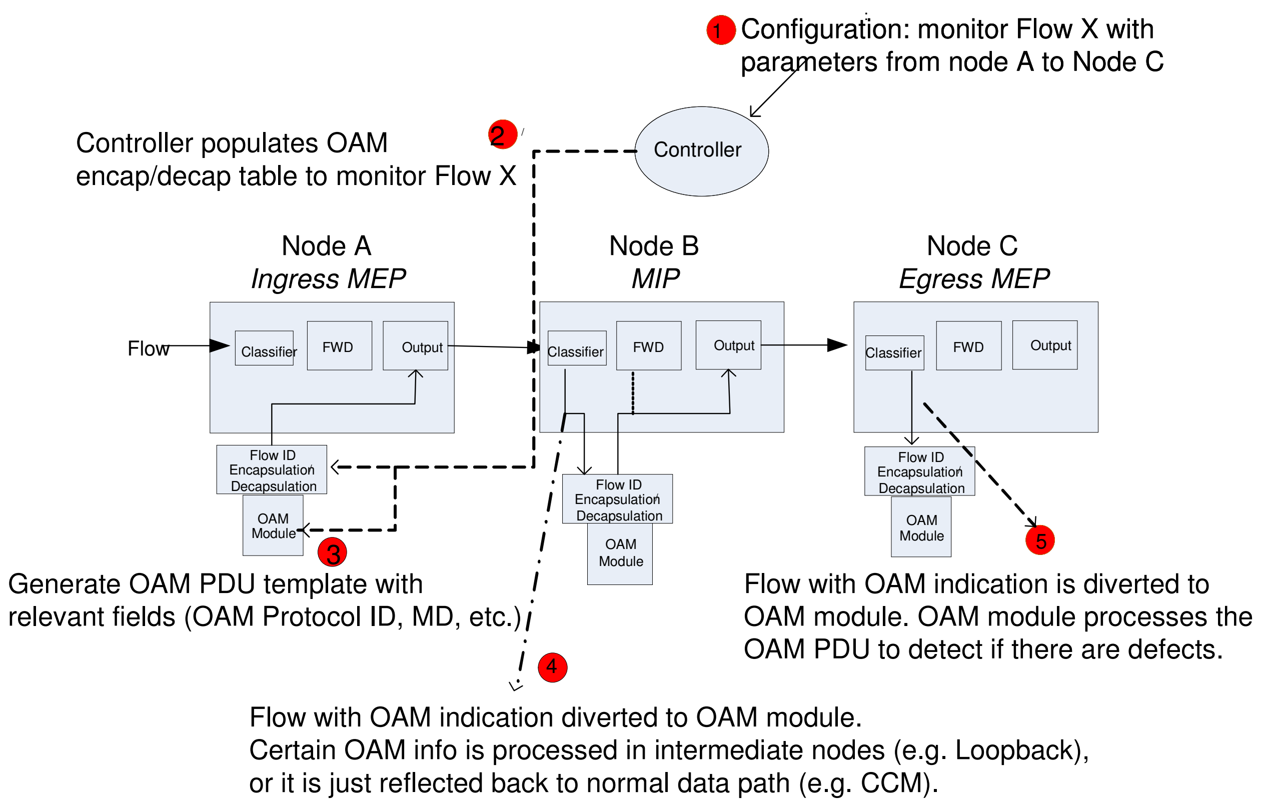

1 Configuration: monitor Flow X with parameters from node A to Node C
Controller
2 Controller populates OAM encap/decap table to monitor Flow X
Node A
Ingress MEP
Node B
MIP
Node C
Egress MEP
Flow
Classifier
FWD
Output
Flow ID Encapsulation Decapsulation
OAM Module
3 Generate OAM PDU template with relevant fields (OAM Protocol ID, MD, etc.)
4 Flow with OAM indication diverted to OAM module. Certain OAM info is processed in intermediate nodes (e.g. Loopback), or it is just reflected back to normal data path (e.g. CCM).
5 Flow with OAM indication is diverted to OAM module. OAM module processes the OAM PDU to detect if there are defects.

Figure 19: Flow OAM Architecture

Figure 19 contains a schematic overview of the proposed flow OAM, depicting the Flow ID module as well as the OAM module. To highlight the procedure of a typical OAM session including configuration, we list the necessary steps for an exemplary continuity check (CC) session:

1. The OAM configuration module in the controller receives a network management command to monitor traffic flow with certain parameters (Flow ID such as VLAN or MPLS label) from Node A to Node C. The OAM configuration module also manually or dynamically generates the corresponding MD (Maintenance Domain), MA (Maintenance Association), MEP and MIP.

2. The controller populates the Flow ID encapsulation/decapsulation tables of all nodes based on the traffic flow information which is to be monitored, for example, if the traffic flow is MPLS, the OAM PDU must be encapsulated with the correct Flow ID (MPLS label).

3. The OAM module in Node A generates an OAM packet template and fills the fields of the OAM PDU, such as OAM Protocol ID, MD, MEP, etc. In this case, the OAM module will create a CCM OAM PDU. This OAM PDU is fed into the Flow ID module which encapsulates this OAM PDU with correct Flow IDs (e.g., Ethernet MAC, VLAN or MPLS label) based on the Flow ID encapsulation/decapsulation table.

4. When the intermediate Node B receives traffic flow from Node A, the traffic flow with OAM indication (e.g., OAM EtherType) will be guided to the Flow ID module. The Flow ID module will strip the Flow ID and send the OAM PDU to the OAM Module for further processing. In this case, OAM PDU is a CCM OAM PDU, thus the intermediate Node B will not process this OAM PDU. The OAM PDU will be sent back to the normal traffic flow path.

5. When MEP Node C receives traffic flow from Node B, the traffic flow will be matched against a default flow table which has a special flow entry to guide traffic flow with OAM Type (e.g., OAM EtherType) to the Flow ID module. The Flow ID module will strip the Flow ID and send the OAM PDU to the OAM Module for further processing. In this case, the OAM module recognizes it is the destination of this CCM OAM PDU. The OAM module finally processes the OAM PDU to detect if there is a problem.

This is a general Flow OAM mechanism for Split Architectures. In practice, we can reuse all the existing implementation in IEEE 802.1ag and Y.1731. For example, the OAM PDU can be based on Ethernet OAM, whereas the actual traffic can be MPLS or IP or Ethernet. However, it is necessary to define an identifier for OpenFlow OAM indication. We suggest reusing the Ethernet OAM EtherType 0x8902, but a new special Flow OAM type or other fields in the OpenFlow match structure might also be used. Furthermore, since OAM is not part of current OpenFlow implementations [19], the selected OAM identifier thus needs to be considered by the OpenFlow parser (i.e., classifier) within each datapath element. Besides configuration routines, this modification of the OpenFlow classifier is also the major extension to the OpenFlow specifications required in order to implement the proposed flow OAM mechanisms.

In order to support multi-service and/or multi-operator scenarios, the Flow ID module can be implemented as a service multiplex entity as in Figure 20. The Flow ID service multiplex entity has one General SAP (Service Access Point), and a number of multiplexed SAPs, each of them assigned to a single VLAN identifier, or a MPLS label, or a generic Flow Identifier depending on the configuration of the controller. Upon receiving an OAM packet from the General SAP, the Flow ID multiplex entity uses the identifier (VLAN ID, MPLS label, etc.) or a generic flow identifier to select one of its corresponding multiplexed SAPs to present the OAM PDU to the specific OAM module instance. Similarly, upon receiving an OAM PDU from a specific OAM module instance from the multiplexed SAP, the identifier associated with this SAP is added to the OAM PDU before going to the General SAP.

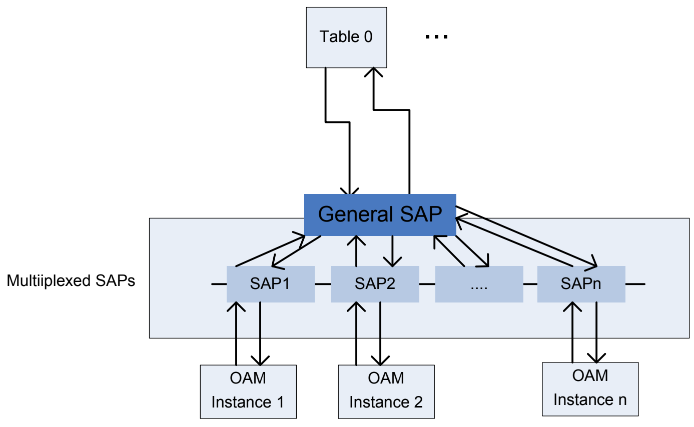


Figure 20: Flow ID Module

For technology-specific OAM requirements which an Ethernet OAM PDU cannot satisfy, we may define new PDU types to extend the functionality. Furthermore, it would also possible be possible to base OAM instances on MPLS BFD instead of Ethernet OAM.

The Flow OAM architecture in 3.3.4.1 assumes that an OAM indication is considered by the OpenFlow classifier, so that the OAM flow can be detected and diverted to the corresponding OAM instance in the common OAM module. In other words, the architecture proposed actually decouples OAM traffic from regular data traffic. For this reason, fate sharing between the OAM flow and the data flow, which is to be monitored, needs to be enforced. However, we have not yet given a detailed description of how to ensure fate sharing. We will therefore present two proposals of how to achieve fate sharing. One is to create a virtual data packet in datapath elements in order to test the forwarding engine. The other approach is to use the META data header in the internal flow processing of each node.

3.3.4.2 Fate sharing with virtual data packets

The idea is to test the forwarding engine with a virtual data packet, which is created from the information carried in the payload of the actual OAM packet. The next hop for the OAM packet is selected based on the forwarding decision made for the virtual data packet. Before exiting at the output port, the virtual header will be stripped off and put back into the payload of the original OAM packet.

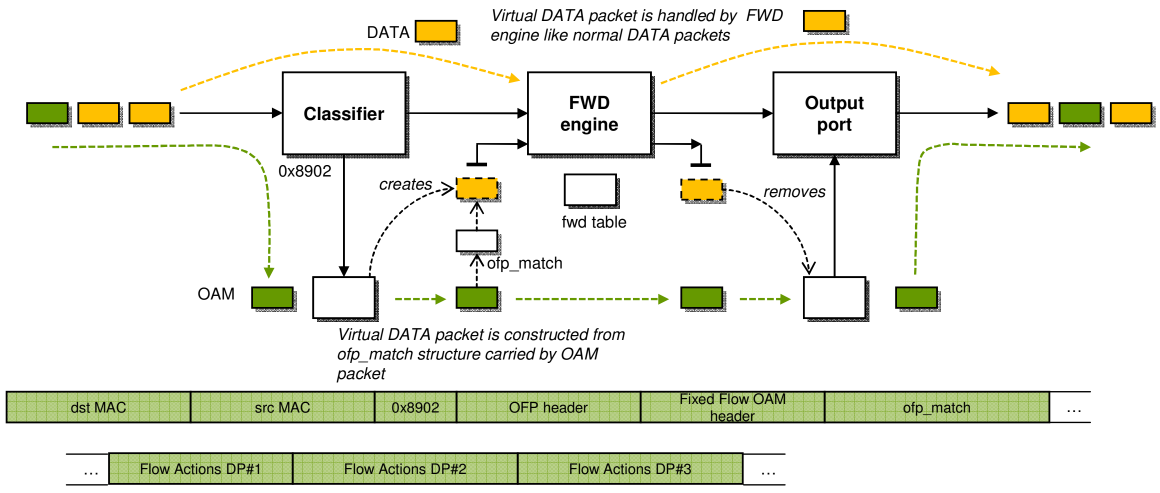


Figure 21: Virtual Data Packet

With the help of a virtual data packet, fate sharing is enforced since OAM packets travel through the same links, the same node and the same filtering tables as the corresponding data traffic. The procedure is described in the following steps explaining Figure 21:

- OAM packets (highlighted in green) arrive at datapath elements interleaved with data packets (yellow) for the corresponding flow on a certain interface.
- The classifier, which has been updated to recognize OAM packets based on some ID (e.g., EtherType 0x8902 as used by Ethernet OAM) diverts the OAM packet to the Flow ID encap/decap module and the corresponding OAM module instance (simplified as a single OAM module box in the figure).
- The OAM packet payload carries information about the packet headers used in the corresponding data packets (e.g. in the form of an OpenFlow match structure). The OAM module uses this information in order to create a virtual data packet whose header matches those of regular data packets of the specific flow.
- The newly created virtual data packet is inserted into the forwarding engine like any regular data packet. Since it has an identical header as regular data packets, it will also match the same rules and actions as corresponding data packets from the monitored flow.
- The OAM module collects the virtual data packet before it is sent out at an output port. The header of the virtual data packet is adjusted in accordance with the actions performed by the OpenFlow forwarding engine. The virtual packet header is stripped off and the header information is stored in the OAM packets payload.
- The OAM packet is forwarded at an output port as chosen by the forwarding engine or the controller. Thus, the OAM packet is again interleaved with the data traffic of the monitored flow.

A flow OAM solution implementing the virtual data packet approach as depicted above would result in a single, generic OAM module in each datapath element, covering the entire flowspace as offered by OpenFlow, and at the same time would ensure fate sharing of OAM and data traffic. While this approach presents an initial concept for an OpenFlow OAM solution, we acknowledge that it has some limitations and is thus still the subject of ongoing discussions and future work. First of all, the separate handling of OAM packets at each datapath element could have undesired effects on OAM performance measurements (e.g., delay) or cause packet reordering. And second, this type of generic flow OAM works only within an OpenFlow domain, but does not provide compatibility with legacy domains (i.e., traditional OAM tools). However, legacy OAM needs to be taken into account in scenarios where an OpenFlow domain is adjacent to non-OpenFlow domains, as in the access/aggregation use case described in Deliverable D2.1.

3.3.4.3 Fate sharing with metadata encapsulation

A different solution would be to add additional OpenFlow metadata of fixed length to each packet inside an OpenFlow domain. In order not to interfere with current standards, there are two options – either we add the additional metadata at the beginning of the packet or at the end of the packet. Since packets are of variable length, stripping bits off the packet may be easier at the beginning of the packet (i.e., before any protocol header). The OpenFlow metadata is always stripped by each OpenFlow datapath element for inspection by a module in the node, and added again to the packet on the line out. It is therefore not interfering with the switching in any way, but it enables additional management of flows without interfering with standardized protocols. OpenFlow switches which do not follow this OAM standard must perform the strip and add operation transparently (without processing the contents of the header).

Adding OpenFlow metadata to packets essentially corresponds to adding an extra encapsulation layer within OpenFlow domains, which would allow indication of special traffic (such as OAM or in-band controller traffic) without the need to interfere with the OpenFlow match structure. The disadvantages of this approach are that all switches must support the stripping operation. Furthermore, the OpenFlow domain would have a decreased effective MTU size. Advantages are simplicity and that all networked packets with the same headers (including OAM packets) follow the same path through the network, i.e., fate sharing is easy to ensure. Furthermore, all the processing of the metadata can be done in parallel to the switching operation.

For example, a 16-bit OpenFlow metadata field preceding packets in an OpenFlow domain could be specified as follows:

- Packet class (3 bits)
    - (000) Data packet
    - (001) OpenFlow control packet (e.g., for in-band OpenFlow)
    - (010) OAM packet
    - (011-111) currently undefined
- Protection class (3 bits)
    - (000) unprotected flow
    - (001) restored flow
    - (010) 1:1 protected flow
    - (011) 1:1 protected flow with restoration after dual failure
    - (100) 1+1 protected flow
    - (101) 1+1 protected flow with restoration after dual failure
    - (110-111) currently undefined
- Currently unused (9 bits)
- Parity check bit (1 bit)

3.4 Resiliency

Network resilience is the ability to provide and maintain an acceptable level of service in the presence of failures. Resilience is achieved in carrier-grade networks by first designing the network topology with failures in mind in order to provide alternate paths. The next step is adding the ability to detect failures and react to them using proper recovery mechanisms.

The resilience mechanisms that are currently used in carrier-grade networks are divided into two categories: reactive restoration and proactive protection. In the case of protection, redundant paths are preplanned and reserved before a failure occurs. Hence, when a failure occurs, no additional signaling is needed to establish the protected path and traffic can immediately be redirected. However, in the case of restoration, the recovery paths can be either preplanned or dynamically allocated, but the resources needed by the recovery paths are not reserved until a failure occurs. Thus, when a failure occurs, additional signaling is needed to establish the restoration path.

We divide this section into two parts: The first describes data plane failures managed by an out-of-band control plane; the second describes failure management when the failure is located in the control path.

3.4.1 Data plane resiliency

The currently implemented failure detection mechanism in OpenFlow networks is basically loss of signal (LOS) which is detected by the hardware of the switch (this causes an OpenFlow port to switch to down state from up). This mechanism only detects link-local failures – for example, it says nothing about failures in forwarding engines on the path. As link-local failures can be used to detect failures in restoration, the available mechanism in OpenFlow can be used in the restoration mechanism. However, since path protection requires end-to-end failure detection, we require an additional method to detect those failures. As explained in Section 3.3, BFD can be used to detect end-to-end failures on a path in a network. The same protocol can be used to trigger protection in OpenFlow networks.

Fast restoration in OpenFlow networks can be implemented in the controller. This demands immediate action from the controller after failure notification. Failure recovery can be performed by removing the existing flow entries affected by the failure and installing new entries in the affected switches as fast as possible following failure notification. The restoration mechanism can be seen in Figure 22A and consists of the OpenFlow switches A, B, C, D and E. Assuming the controller knows the network topology, we can calculate a path from a source node to the destination node. In Figure 22A, the controller first installs the path <ABC> by adding the flow entries in the switches A, B and C. Once a failure occurs on link BC and a failure notification message from OpenFlow switch B is received by the controller, it calculates the new path, <ADEC>. Since the flow in the flow entry for the working path <ABC> and restoration path <ADEC> is identical, but the action is different for OpenFlow switch A (i.e., to forward to the switch B or D), the controller must modify the flow entry at A. In addition, for the restoration path <ADEC>, there are no flow entries installed in nodes D, E, and C, so the controller must add these entries in the respective switches. The flow entry in C for the working path <ABC> and restoration path <ADEC> is likely different since the incoming port can be a part of matching header in the flow entry. Once all the affected flows have been updated/installed in all the switches, the network can be recovered. After the immediate action of restoration, the controller can clean up the other nodes by deleting the flow entries at B and C related to the older path <ABC>.

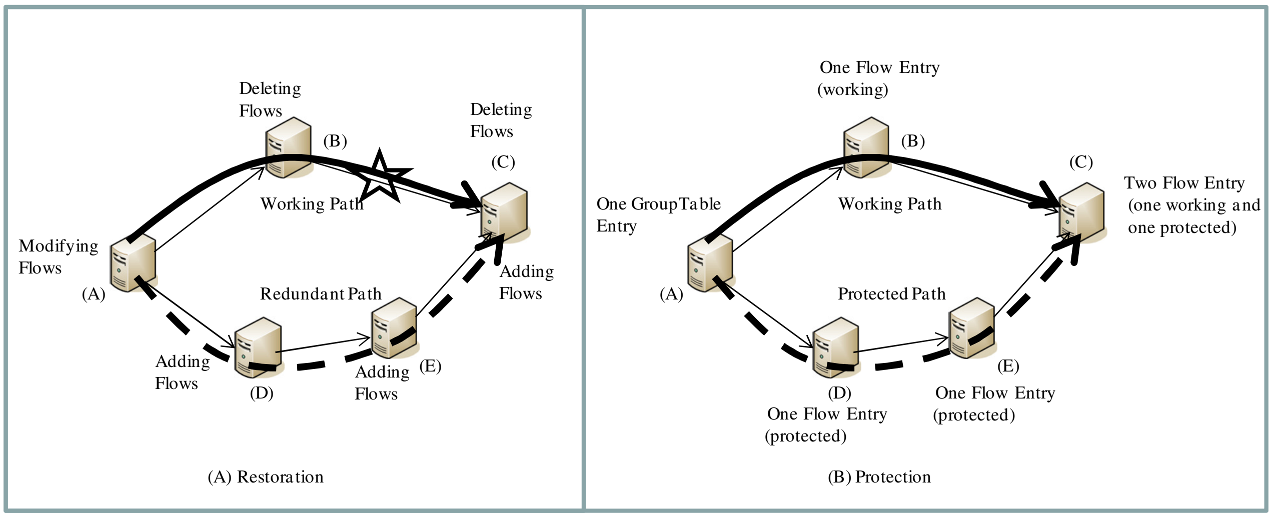


Figure 22: Recovery mechanism for OpenFlow networks

The total time to recover from failure using restoration depends on:

1. The amount of time it takes for a switch to detect a failure.
2. The propagation delay between the switch and the controller.
3. The time spent by the controller to calculate the new path.
4. The time spent by the controller to transmit the messages to modify/delete/add flow entries in the switches.
5. Time spent by the switches to modify/add the flow entries.

Packets may be lost during the time between failure detection and complete restoration. In order to reduce the packet loss resulting from a delay in executing the restoration action, we can switch over to the pre-established protection. Protection eliminates the need for the controller to update the datapath elements for modifying and adding new flow entries in order to establish an alternative path. This is accomplished by precomputing the protected path and provisioning it along with the working path. The advantage of this is fast recovery, but the disadvantage is the need for more entries in the flow tables.

To implement a protection scheme, we can use the group table concept (specified for OpenFlow v1.1). Unlike a flow table, the group table consists of group entries which in turn contain a number of actions. To execute any specific entry in the group table, a flow entry forwards the packet to a group entry with a specific group ID. Each group entry consists of the group ID (which must be unique), a group type and a number of action buckets. An action bucket consists of an alive status (e.g., watch port and watch group in OpenFlow v1.1) and a set of actions that are to be executed if the associated alive status has a certain value. OpenFlow introduces the fast failover group type in order to perform fast failover without needing to involve the controller. This group type is important for our protection mechanism. Any group entry of this type consists of two or more action buckets with a well-defined order. A bucket is considered alive if its associated alive status is within a specific range (i.e., watch port or watch group is not equal to 0xffffffff). The first action bucket describes what to do with the packet under the normal condition. On the other hand, if this action bucket is declared as unavailable, for example due to a change in status of a bucket (i.e., 0xffffffff), the packet is treated according to a “next” bucket, until an available bucket is found. The status of the bucket can be changed by the

OpenFlow switch by the monitored port going into "down state" or through other mechanisms, e.g., if a BFD session declared the bucket as unavailable. In our protection mechanism we used BFD to declare the bucket unavailable.

The protection mechanism for OpenFlow can be seen in Figure 22B. When a packet arrives at the OpenFlow switch (A), the controller installs two disjoint paths in the OpenFlow network: one in <ABC> (working path) and the other one in <ADEC> (protected path). The OpenFlow switch (A) is the node that actually needs to take the switching action on the failure condition, i.e., to send the packet to B on the normal condition and to send the packet to D on the failure condition. For this particular flow of the packet, the group table concept can be applied at OpenFlow switch (A), which may contain two action buckets: one for output port B and the other for output port D. Thus, one entry can be added in the flow table of the OpenFlow switch (A) which points the packet to the above entry in the group table. Since the group entries do not require any flow information, it can be added proactively in the OpenFlow switch (A). For the other switches, B and C for the working path, and D, E and C for the protection path, only a normal flow entry can be added. Thus in our case, the switch in C contains two flow entries, one for the working path <ABC> and other for the protecting path <ADEC>. Once a failure is detected by BFD, the OpenFlow switch (A) can change the alive status of the group entry to make the specific bucket unavailable for the action. Thus, action related to the second bucket, i.e., whose output port is D, can be taken when the failure is detected in the working path. As the flow entries in D, E and C related to the <ADEC> path are already present; there is no need to establish a new path in these switches once the failure is detected.

3.4.2 Control channel resiliency

This section discusses recovery from a control channel failure, i.e., when the connection between the controller and the OpenFlow switches fails. Earlier, we considered the failure in data plane links, i.e., the links between the OpenFlow switches. However, because OpenFlow is a centralized architecture (relying on the controller to take action when a new flow is introduced in the network), reliability of the control plane is also an important issue. There are multiple options for control plane resiliency. One can provide two controllers, each on a separate control network, and when a connection to one controller is lost, the switch can switch over to the backup network. This is a very expensive solution. Control plane resiliency can also be obtained by having a redundant connection to the controller, where restoration and protection mechanisms can be applied in the out-of-band network. However, in some cases, it is difficult to provide redundant path for an out-of-band control network. Another option is to try to restore the connection to the controller by routing the control traffic over the data network, i.e., an in-band solution. When a switch loses connection to the OpenFlow controller, it can send its control traffic to a neighboring switch which forwards the traffic to the controller via its own control channel. This requires that the controller detects such messages and establishes flow entries for routing the control traffic through the neighboring switch. This solution is an intermediate step toward full in-band control. An effective scheme for control plane resiliency in carrier grade networks may be to implement out-of-band control in the failure-free scenario, switching to in-band control for switches that lose the controller connection after a failure. In-band topology discovery is discussed in section 3.5.2. The same algorithm can be used for restoring the failure once the failure occurs in out-of band control plane path.

3.5 Topology Discovery

Topology discovery is the process of discovering the information about the network devices and their interconnections. This section is divided into two parts, the controller and In-band topology discovery. In the controller discovery mechanism, we discuss the topology discovery mechanism implemented for the OpenFlow networks in the NOX controller and then proposes an extension to this topology discovery mechanism.

3.5.1 Controller Topology Discovery Mechanism

The network devices in a network can be OpenFlow-controlled nodes and the non-OpenFlow-controlled nodes. The OpenFlow nodes are known to the OpenFlow controller at the initial stage when the controller establishes a control connection to the OpenFlow nodes. We need a mechanism to find the link between OpenFlow nodes and how these nodes are connected with the non-OpenFlow nodes. The current NOX controller utilizes LLDP (Link Layer Discovery Protocol) to recognize the link between OpenFlow nodes.

Two messages are important in understanding the topology discovery protocol in the NOX controller: (1) "packet-in" message, and (2) "packet-out" message. When a packet is received by the node and needs to be sent to the controller, it is sent via "packet-in" message. On the other hand, when the controller wants to send out a packet through any port of a node, it is sent via "packet-out" message.

The NOX controller uses a discovery module to recognize the topology of the network. The discovery module transmits a "packet-out" message to forward LLDP (Link Layer Discovery Protocol) packets among OpenFlow nodes. When the OpenFlow node receives this "packet-out" message, the LLDP packets are sent to the respective output port. When the corresponding OpenFlow switch receives this LLDP packet, it sends the "packet-in" message to the NOX controller. When the controller receives the "packet in" message and finds that it is a LLDP packet which was sent by a particular node, it has successfully discovered the link between the sender and receiver.

The link addition and failure detection time in the discovery mode depends on the send and timeout intervals of the LLDP packets. The send interval is the time after which the discovery module at the controller sends a LLDP "packet out" message to the connected nodes. The timeout interval is the time after which a failure is noticed if no LLDP message is received. Once the link is detected by the discovery mode, it is inserted into its topology database. On the other hand, when the controller receives a request for an action with respect to the unknown destination, it is broadcast to every port. In such situations, the destination is searched via MAC learning. However, the OpenFlow network may have loops in its topology. Hence, flooded or broadcasted packets may persist indefinitely, depending on the configuration of the OpenFlow domain. This might cause congestion, and furthermore discovery of all destinations by flooding (MAC learning) may not function correctly since nodes may receive packets from a source via multiple ports. The current solution in Ethernet networks to prevent loops in the topology is to draw a spanning tree and flood the packet around that spanning tree. In the OpenFlow case, the controller knows the topology, so the spanning tree can be drawn with the available discovery module. The controller can flood the data packet via the spanning tree to learn the path for the unknown destination, which can be an OpenFlow node or a non-OpenFlow node. The controller already knows the information about the OpenFlow nodes (during initial phase when it establishes the connection with the OpenFlow nodes) and topology of OpenFlow network (by LLDP protocol). Thus, the unknown destinations for packets will always be non-OpenFlow nodes. Therefore, the controller now only needs to learn about non-OpenFlow nodes connected with OpenFlow nodes. The controller also knows which links of the OpenFlow nodes are connected with the non-OpenFlow nodes, since links which do not receive the LLDP packet back again are links connected to non-OpenFlow nodes. On the other hand, there is also a case where a non-OpenFlow node also runs the LLDP protocol and sends a LLDP packet. In this case the controller cannot find the link connected to the non-OpenFlow node. To solve this problem,,the LLDP protocol of the controller can use the source MAC address of the packet to verify whether the LLDP message originated from a known OpenFlow node or not. For this purpose, invalid MAC addresses (00:00:00:00:00:00 or ff:ff:ff:ff:ff:ff) can be used in the source MAC address of the LLDP packet. If this is the case, non-OpenFlow nodes are discovered either by non-reception of replies or valid source MAC addresses. Now the controller can send the packet with unknown destination MAC address directly from the link to which it assumes that non-OpenFlow nodes are connected. In this case, each OpenFlow node has to transmit a full data packet to the controller when it does not find any entry in the flow table related to it. As a result, the entire OpenFlow network can behave like a single switch, where the controller discovers unknown links and transmits the packet only via these links to search unknown destinations.

3.5.2 In-band topology discovery

In-band control-channel restoration as introduced in Section 3.4.2 requires a specific topology discovery mechanism. In Figure 23, node (B) is connected to the controller directly and the other nodes (A, D, E and C) are not directly connected to the controller. In this case, switches A, D, E and C have to communicate with the controller in-band through the data plane link. A discovery mechanism is required from either the OpenFlow switches or the controller side in order to discover how to reach each other; this mechanism can be initiated either by the controller or by the switches. For the controller to start discovery, the discovery packets can be transmitted after a regular interval of time. This may increase the control traffic after some time as the controller needs to send the discovery packets even if no OpenFlow switch is newly connected. To reduce above control traffic, discovery can be initiated from the OpenFlow switches when not connected to the controller. Once the path to the controller is known, the connection with the controller can be established by installing flow entries in the switches..

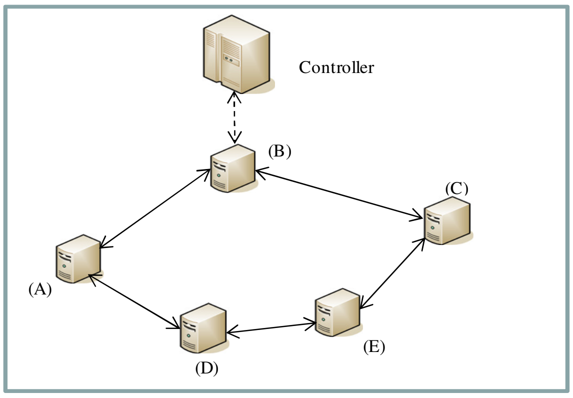


Figure 23: Control plane integration into data plane link

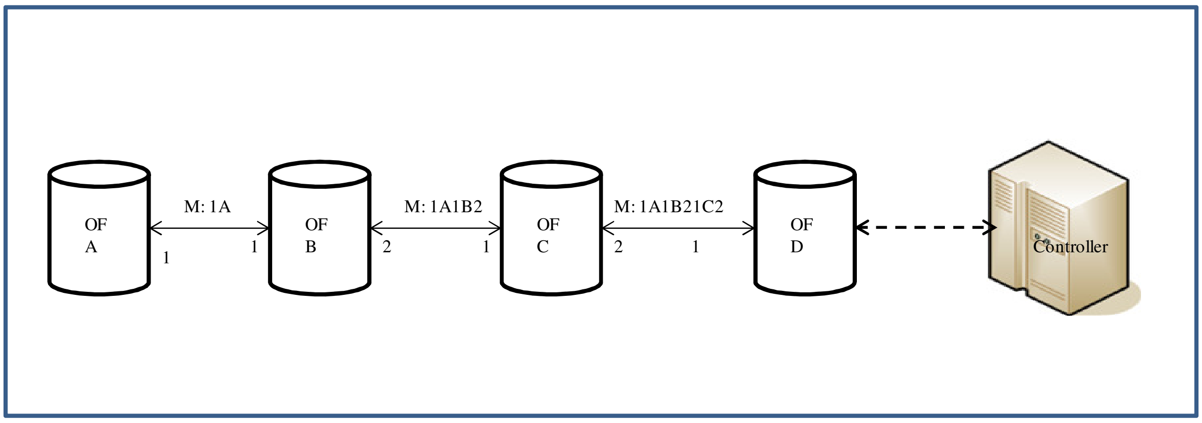


Figure 24: In-Band Discovery

The in-band discovery process is shown in Figure 24, where OpenFlow switches initiate the discovery process. To initiate path discovery to the controller, node A transmits a request as a single local broadcast packet (M:1A), which is received by all the nodes connected to node A, including node B. M:1A indicates that the message is generated by node A through output port 1. Each request identifies the initiator and the intermediate nodes (B, C and D in Figure 24). Each request records a list of <node,port> tuples along the traversed path. When another node receives this message and finds its own address in the list, the packet is discarded in order to avoid loops. If the node does not find its address in the list, but it knows the controller, it replies to the initiator of the request with a copy of the list. Each node that receives this reply message can establish the flow entries to create an in-band path to the controller. Otherwise, this node appends its own <node,port> tuple to the list (e.g., M:1A1B2 at node B) and propagates it by transmitting it as a local broadcast. This discovery process can be restarted by the source after some interval if the node did not receive any reply from the controller.

3.6 Service Creation

Service creation is a very general concept. In the telecommunication industry the concept describes service creation points (SCP) which are the points in the network where network functions with customer-specific or product-specific parameters needs to be configured. These are customer facing services, in network technologies often referred to as the service itself, and often referred to as a network service or a user session. For example, mobile networks provide adequate means for addressing various service slices via so called “Access Point Names” (APN) that tie a user session and its established PDP context (the tunnel actually connecting the user and the service creation gateway) to the service slice. Currently such a comprehensive architectural solution is lacking in fixed networks, e.g., in the architecture defined by the Broadband Forum for access aggregation domains. In this case a BRAS grants access to a user session which comprises only the default service slice (i.e., IP access) and does not allow differentiation between various service slices.

Besides the network service, there is also a transport function which typically aggregates multiple network services. One could say that network services are tunneled through transport services. For example, customers are using VLAN IDs to split between different services and in a certain forwarding engine; the VLAN IDs are mapped to MPLS labels in order to reduce configuration efforts since one only needs to configure a single transport service that can carry all the network services. Therefore, there might not be one-to-one mapping between a transport function, and the network services correlation with transport functions might not be given, or there may be a wish split transport and service (a service is a virtual connection between two points in this context, potentially stacked and transported within other virtual connections). Typical SCPs are located at the edge of the telecommunication network, for example the BRAS that is used to create residential services.
Service creation exists in various forms for different customer groups, e.g., an access service provided to residential customers is fundamentally different from the one provided to business customers. Typical examples for both cases are presented in the following section. Nonetheless, there are some requirements and process steps that are common to both cases and these will be explained first.
Another important aspect to be mentioned with regard to service creation is single point provisioning. Today, for certain protocols and their configurations, operators cannot cope with the growing complexity of the network. For example, the scalability of VLAN identifiers is limited to 4094 unique identifiers, but operators have many more users to organize in a typical network domain or segment. Technologies like Provider Bridge (Q-in-Q, IEEE 802.1ad) overcome this limitation, but require additional configuration efforts at the border of both technology variants. Therefore, the number of such provisioning points has to be minimized. This consequently requires good network design and/or qualified control plane protocols.

3.6.1 Service creation phases

Service creation denotes the process of connecting and granting access for a user and the creation of access to specific service slices, taking into account different limitations stemming from policy restrictions, which in turn are influenced by contractual constraints, etc. Service creation comprises several phases:

Attachment phase: A user must establish physical connectivity to the operator's network termination or authentication point. A network operator may adopt various options for establishing initial connectivity for the user, e.g., a legacy PPPoE/PPP session, a DHCP/IP-based solution, or an IPv6 link layer auto-configuration procedure. The network operator's policy will presumably demand prevention of user access to other service slices in this phase. Note that in this scenario there is a strict distinction between user identity and network attachment point. However, in existing solutions, this is not always the case, e.g., the line ID on a DSLAM access node identifies both the user ID and network attachment point. .

Default session phase: This phase is only applicable in cases where the user is not or could not be authenticated. e.g., the user does not have authentication information, but still requires network connectivity for emergency calls, resolving configuration/purchasing issues or to contact helpline services. Here special protocols or protocol configurations should be used such as PPPoE/PPP session establishment without authentication or DHCP-based IP configuration with limited access rights to a special service slice, e.g., a landing page or emergency VoIP system. If a user has sufficient authentication information, one should skip this phase and directly move to the authentication phase.

Authentication phase: Access to the user's service slices is granted based on proper user authentication, i.e., the network operator is able to identify the user and allocate a session for handling all management-related activities concerning service slice access. Again, a network operator should not be restricted in his selection of an authentication scheme and an authentication wrapper protocol for exchanging the necessary authentication PDUs – PPP, IEEE802.1x, PANA are some examples of authentication protocols that allow integration of various authentication schemes. Typically, this also includes some form of "binding" the user session between the network operator and user by deriving some keying material for message authentication and probably encryption.

Authorization (and signaling) phase: Once a user session has been established, various options for attaching the user to service slices exist: a) some form of dedicated signaling via a specific signaling protocol (resembling SIP-style signaling in mobile networks for accessing x-CSCF functions); b) some policy-driven automatic attachment based on user/operator contracts that correlates to the automatic attachment to the Internet service slice as done today in the existing access/aggregation architectures; c) some form of implicit signaling where the network performs some form of deep packet inspection in order to determine which is the appropriate service slice.

Session phase: User attachment is a complex process beyond the three phases discussed thus far. For attaching a user to a service slice, the management subsystem must ensure several constraints. Typically, a service slice consists of several service slice gateways and transport services between the user's network attachment point and the service slice gateway. It may be desirable to provide an adequate level of service quality as well. Furthermore, when IP connectivity is required for joining a service slice, compatibility of the addressing scheme adopted in the service slice and the addresses assigned during the initial attachment phase must be ensured. Today residential gateways (RGW) in fixed network environments are typically given a single IP address for accessing various service slices. A more advanced scheme may coordinate the assignment of different identifiers for different service slices on an RGW. The single IP address model also potentially stresses the service slice gateway with the need to do network address translation for providing services in a transparent manner, i.e., NAT helper applications may be required.

3.6.2 Relationship to requirements and OpenFlow 1.1

Overall, service creation relates to a multitude of the requirements detailed in D2.1. The list is as follows:

- R-1: A wide variety of services/service bundles should be supported.
- R-2: The Split Architecture should support multiple providers.
- R-3: The Split Architecture should allow sharing of a common infrastructure to enable multi-service or multi-provider operation.
- R-4: The Split Architecture should avoid interdependencies of administrative domains in a multi-provider scenario.
- R-8: The Split Architecture should support automatic transfer methods for distribution of customer profiles and policies in network devices.
- R-10: The Split Architecture should provide sufficient customer identification.
- R-21: The Split Architecture should monitor information required for management purposes.
- R-23: The Split Architecture should extract accounting information.

- R-29: It should be possible to define chains of processing functions to implement complex processing.
- R-30: The Split Architecture shall support deployment of legacy and future protocol/service-aware processing functions.

Obviously, R-1 – R-4 need to be covered by the service creation approach. Multiservice support is highly relevant both in current and future x-Play networks with business customer support. In multi-provider environments, it is important to distinguish between the various customers in relation to the appropriate provider. Here R-10 is demanded as well. In conjunction with customer identification, support for the configuration of customer-specific entities in edge devices (R-8) as service creation is required in telecommunication networks. In addition, R-21 and R-23 are relevant with the demand for management information in general and customer-specific terms. One of the most important requirements is the support of the deployment of legacy processing functions (like PPPoE or other tunneling protocols) as defined in R-30. In conjunction with PPPoE, for example, requirement R-29 is important as PPPoE is a mixture of different protocols and requires flexible decision logic (the complexity of PPP can be pointed out by 104 RFC's with "PPP" in the title).

Comparing the requirements in this section and the desired network functions from the previous section with the implemented OpenFlow switch specification version 1.1.0, it is rather difficult to outline specific missing features in the specification. However, the following existing features could be reused:

- Security mechanisms preventing unrestricted or authorized access could be implemented as a combination of default flow entries with appropriate counters and pointers to the authentication platform; counters must become more dynamic, allowing the submission of traps and notifications to network management systems.
- DDoS attacks could be prevented by appropriate counters for specific protocol requests like ARP requests; counters must become more dynamic allowing the submission of traps and notifications to network management systems.
- Spoofing could be prevented by appropriate flow entries after authentication.
- Accounting information collection mechanisms could be based on various counters.
- Management support is given to a limited extent with various "read state message" – here especially counters must become more dynamic allowing the submission of traps and notifications to network management systems.
- Authentication and related authorization of the datapath could be based on the setting of appropriate flow entries; however, the decision logic is currently beyond the scope of the OpenFlow specification.
- Forwarding to RADIUS entity, auto-configuration functions could be based on appropriate flow entries; however, the decision logic for these functions is currently beyond the scope of the OpenFlow specification.
- Configuration of profiles could be based on group tables; it is unclear how sophisticated the potential configuration options are.
- Limited support for IPv6 could be enabled by flow entries matching the IPv6 EtherType, but the specification lacks more advanced forwarding analytics.

Note that missing features do not have to be implemented in OpenFlow per se. However, it is necessary to point to functions and features which should be available to support service creation models. The essential missing features are:

- Modular decision logic for authentication/authorization and support for related protocols (depending on service creation models).
- Auto-configuration mechanism for customer services.
- Sophisticated management features like profile-based/policy-based network control and common protocol support like SNMP, potentially based on more dynamic counter mechanisms.
- Support for PPPoE matching and encapsulation/decapsulation.
- Support for IPv6 matching as well as modification actions.

3.6.3 Residential customer service creation (PPP and beyond)

For residential customers, service creation and single point provisioning requires a combination of different network functions like authentication, authorization and accounting, (auto-)configuration of layers 2 – 7, and security for the network infrastructure. In addition, it must be possible to trace and analyze problems and support today's current protocols. There are two typical protocol suites/models:

- The most common one is the Point-to-Point Protocol (PPP) in combination with RADIUS, in current deployments as a PPP over Ethernet (PPPoE) variant. PPP is a combination of a data plane encapsulation and protocol suite, which provides several functions like support for different data plane protocols, auto-configuration of layers 2 and 3, different authentication mechanisms and an interaction model with RADIUS. This interaction is implemented through a Broadband Remote Access Service (BRAS). The BRAS functionality is typically provided by a router and requires a high degree of processing and memory in the router compared to other operations. The BRAS is the mediation point between PPP and RADIUS. RADIUS provides AAA and configuration options as well as services between centralized user databases (like OSS and BSS systems) and the BRAS. The schematic diagram is illustrated in Figure 25.
- The second model is an adaptation of the Dynamic Host Configuration Protocol (DHCP) with several extensions, called DHCP++ in this deliverable. DHCP has its roots in LAN environments and provides auto-configuration support for layers 3-7. For carrier-grade support it requires additional protocols for AAA and security, and potentially auto-configuration for layer 2. Here a variety of options are discussed. Figure 25 only illustrates a model with AAA entry in the DSLAM.

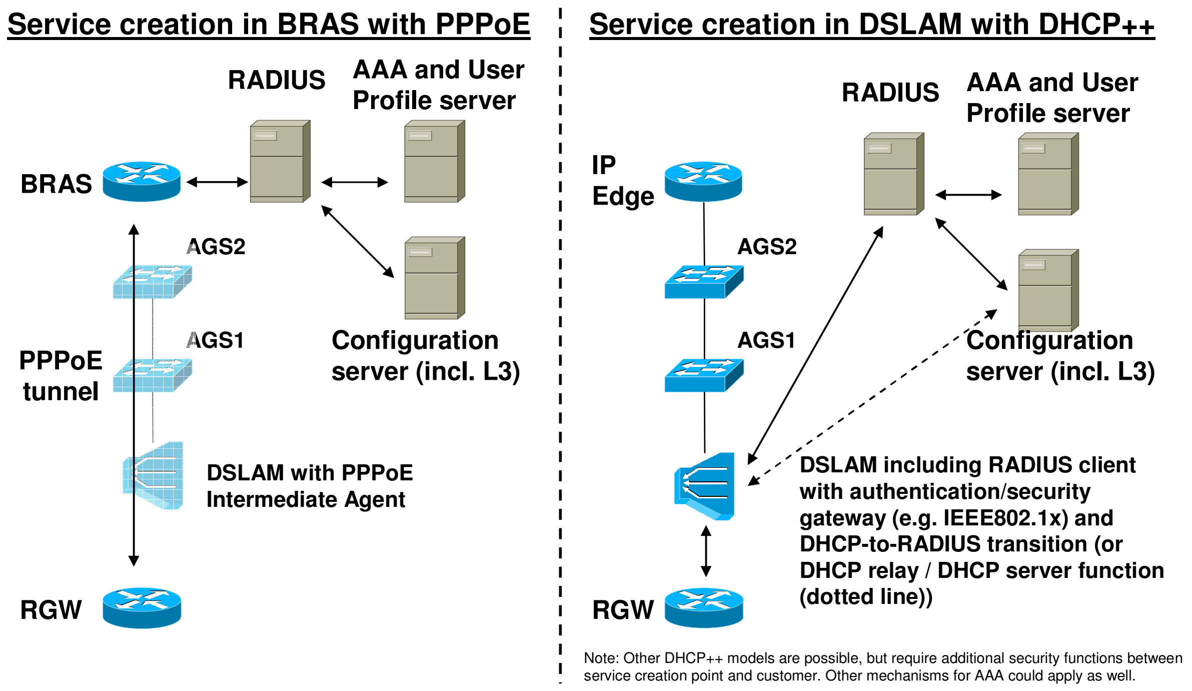

Service creation in BRAS with PPPoE
RADIUS
AAA and User Profile server
BRAS
AGS2
AGS1
PPPoE tunnel
Configuration server (incl. L3)
DSLAM with PPPoE Intermediate Agent
RGW
Service creation in DSLAM with DHCP++
RADIUS
AAA and User Profile server
IP Edge
AGS2
AGS1
Configuration server (incl. L3)
DSLAM including RADIUS client with authentication/security gateway (e.g. IEEE802.1x) and DHCP-to-RADIUS transition (or DHCP relay / DHCP server function (dotted line))
RGW
Note: Other DHCP++ models are possible, but require additional security functions between service creation point and customer. Other mechanisms for AAA could apply as well.

Figure 25: Service creation models BRAS with PPPoE and DHCP++ based on DSLAM

An important aspect is the transition from IPv4 to IPv6 in carrier networks. This aspect is not dedicated to service creation only, but it must be mentioned here explicitly. Several models are now available for the introduction of IPv6 in existing carrier environments like PPPv6, DHCPv6 or IPv6 route advertisements.

In summary, a service creation model has to cover the following aspects:

- Support for legacy protocols in general, PPPoE or DHCP-based IPv6 model as potential future production model.
- Support for common user databases like RADIUS.
- Security mechanisms
    - Authentication of users.
    - Authorization of users.
    - Security mechanisms preventing unrestricted or authorized access.
        - Discarding any data except authorization requests until successful authorization.
        - Restriction of access to resources associated to a user.
        - Protection of network infrastructure against any misbehavior like DDoS attacks (not necessarily with ill-effects) or spoofing.

- Collection of accounting information.
- Support for management.
- Auto-configuration of common L3-L7 functions like IP address, subnet mask, etc.

3.6.4 Business customer service creation based on MPLS pseudo-wires

Business customers typically require connectivity between different locations emulating a LAN environment over public telecommunication services. Service creation relies on the appropriate configuration of desired virtual networks. For example, a pseudo-wire is a generic point-to-point tunneling concept for emulating various services over packet-based networks. Several pseudo-wire-based services have been defined, e.g., TDM circuit emulation, ATM emulation, SONET/SDH emulation, and Ethernet emulation to provide business customers with sufficient options for interconnecting locations. Ethernet on MPLS pseudo-wires can be used to provide an emulated Ethernet connection also known as an E-Line service in MEF terminology. Typically, E-Line services are used to connect two branch offices at the Ethernet layer. Ethernet pseudo-wires are also often used to create Virtual Private LAN Services (VPLS) by establishing a mesh of MPLS LSPs between the provider edge routers and implementing an Ethernet to/from the LSP switch in these routers (similar to a normal Ethernet switch that handles the LSPs like ports) – in MEF terminology these are called E-LANs. MPLS-based VPLS services have become very popular, partly because of the possibility of performing extensive traffic engineering on MPLS LSPs, something that can be difficult with other packet-based networks.

There are several ways to create an E-Line/E-LAN service within an OpenFlow network. Since OpenFlow incorporates the Ethernet layer, one could imagine a network application running in the controller that simply installs Ethernet forwarding rules along a path in the OpenFlow network. However, since the MAC address space is flat, it would result in one flow table entry per OpenFlow switch for each MAC address that exists in the customers network, and that is something that does not scale very well. Additionally, there is a risk of MAC address collisions between E-Lines/E-LANs belonging to different customers (MAC addresses are not as unique as they should be, especially if they have been automatically generated and assigned to a virtual machine). Collisions between different E-Lines could be resolved through translation of the MAC addresses at the provider edge, however, this causes further complications in troubleshooting the network and might increase complexity if multiple controllers are involved, since this would have to be synchronized between the provider edge routers.

Pseudo-wire encapsulation resolves the scalability issues as well as the collision problem. Instead of requiring one flow entry per customer MAC address on all provider nodes, it requires two flow entries per E-Line service – one for each direction of the MPLS LSP carrying the pseudo-wire. Since the Ethernet frames are encapsulated, there is no risk of collisions – the customer MAC addresses are only “visible” at the provider edge nodes.

The overall pseudo-wire architecture is described in RFC3985, and the detailed specification for MPLS networks in RFC4385 and RFC4448. The first step when tunneling an Ethernet frame in MPLS PWE is to encapsulate the frame (source and destination MAC address, EtherType, and payload) with a PWE control word. The control word is used to provide various optional features, for example, it may contain a frame sequence number in order to enforce ordered delivery of the tunneled frames. This is optional when emulating Ethernet because Ethernet as such does not guarantee ordered delivery, therefore the control word can be left empty without violating the specification. Two MPLS labels are prepended on top of the control word, one for identifying the pseudo-wire tunnel and one for identifying the LSP. Finally, a new Ethernet header is added – this header does **not** contain any customer MAC addresses – instead it refers to the originating and next-hop provider router. Once the outer Ethernet header as been added, the frame is completed (see Figure 26).

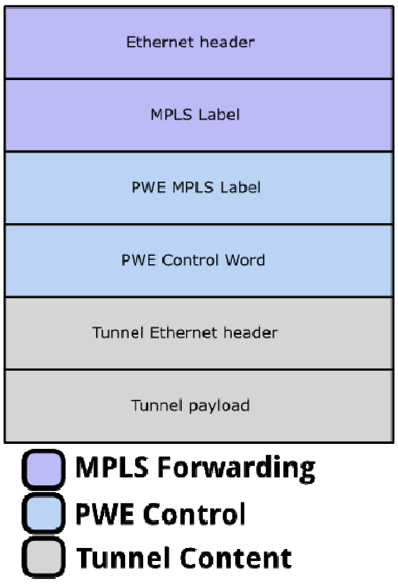


Figure 26: Typical pseudo-wire frame structure for Ethernet emulation

3.6.5 Overall conclusions for service creation

This section summarizes the general missing aspects in the current OpenFlow specification for supporting service creation. They are:

- Detailed, complete service creation model based on OpenFlow for carrier environments.
- Sophisticated push operation of flow entries during bootstrap from controller to switch (access node, e.g., DSLAM) for initial protection of aggregation network.
- Profile-based/policy-based configuration possibilities are required.
- Potentially fine granular configuration of flow entries for protocols or user-specific authorized access to services and/or providers (the latter in the case of multi-provider forwarding).
- Legacy support, e.g., for PPPoE, MPLS Pseudo-Wire or other tunneling protocols, especially from business customer demands.
- General support for IPv6.

In Section 4.3 we will follow up on the issues raised here and present a detailed model for implementation of service creation for both residential and business customers in an OpenFlow-based Split Architecture environment.

3.7 Energy-Efficient Networking

The global concern about climate change on our planet is also influencing the ICT sector. Currently ICT accounts for 2 percent to 4 percent of carbon emissions worldwide. About 40 percent to 60 percent of these emissions can be attributed to energy consumption in the user phase, whereas the remainder originates in other life cycle phases (material extraction, production, transport, and end-of-life). By 2020 the share of ICT in worldwide carbon emissions is estimated to double in a “business as usual” scenario. Thus, an important goal for future networking is the reduction of its carbon footprint.

One way to gain higher energy efficiency in networking is the wider use of optical technologies, since optical signals consume less energy than electrical signals. Additionally, there has been an increase in research worldwide related to sustainable networking in recent years, with initiatives such as the current EU-funded Network of Excellence TREND [5], COST Action 804 [6], the GreenTouch consortium [7] and the CANARIE-funded GreenStar Network [8] which also has European members, including the SPARC members IBBT and Ericsson.

3.7.1 Current approaches to reducing network power consumption

3.7.1.1 Network topology optimization

Different strategies to save power in networks are possible. At the highest level one can investigate whether *optimizations* are possible *in the network topology*. Currently networks are designed to handle peak loads. This means that when the loads are lower, there is overcapacity in the network. At night the traffic load can be only 25 percent to 50 percent of the load during the day. This lower load could allow for a more simplified network topology at night which in turn enables switching off certain links. Additionally, switching off these links allows for line cards to be switched off and thus leads to reduced node power consumption. A concept that implements this principle is MLTE (multilayer traffic engineering). The MLTE approach can lead to power savings of 50 percent during low load periods. However, many access networks are organized in tree structures, so shutting down links is not a feasible option. This means that dynamic topology optimization cannot be applied in all network scenarios, and is not feasible in access networks.

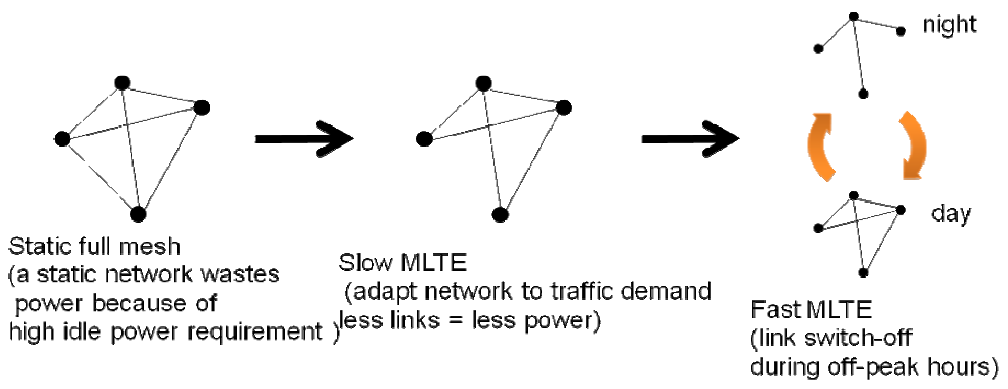


Figure 27: Multilayer Traffic Engineering (MLTE)

3.7.1.2 Burst mode operation

Given a certain static network topology, further optimizations can be achieved by *burst mode* operation. In burst mode operation, packets are buffered in a network node and then sent over the link at the maximum rate. In between the bursts the line can be powered down. This strategy can be useful mainly in access networks due to the "burstiness" of the traffic. However, the difference in power consumption between different link rates is mainly manifested at the higher bit rates. Furthermore, burst mode operation works with very small time scales, so the number of components which can be switched off is limited. Finally, burst mode operation requires larger packet buffers which also need power. Hence it is yet unclear whether this strategy can lead to significant power optimization in reality.

3.7.1.3 Adaptive link rate

Another approach for static network topologies is to use *adaptive link rates*. Adaptive link rate also exploits the "burstiness" of the traffic. The approach is based on the principle that lower link rates lead to lower power consumption in the network equipment. Saving energy is possible by estimating the average link rate required on a line and adapting the link rate to this level. However, similar to burst mode operation, adaptive link rate requires larger packet buffers, which might reduce some of the actual power savings.

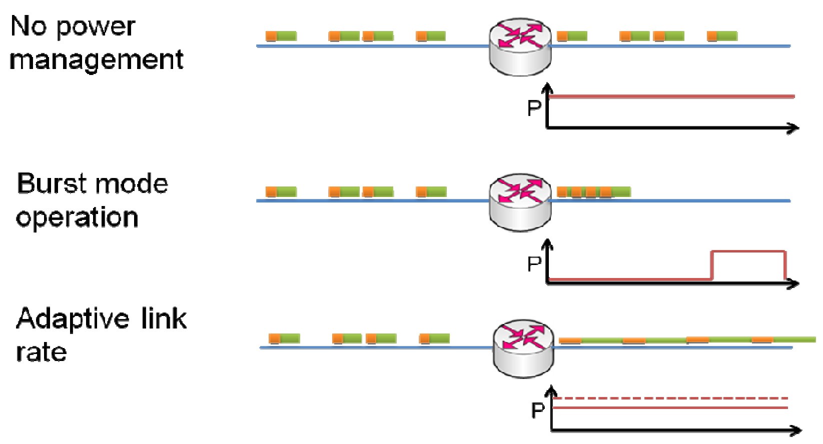


Figure 28: Power Management

3.7.2 Sustainable networking with OpenFlow

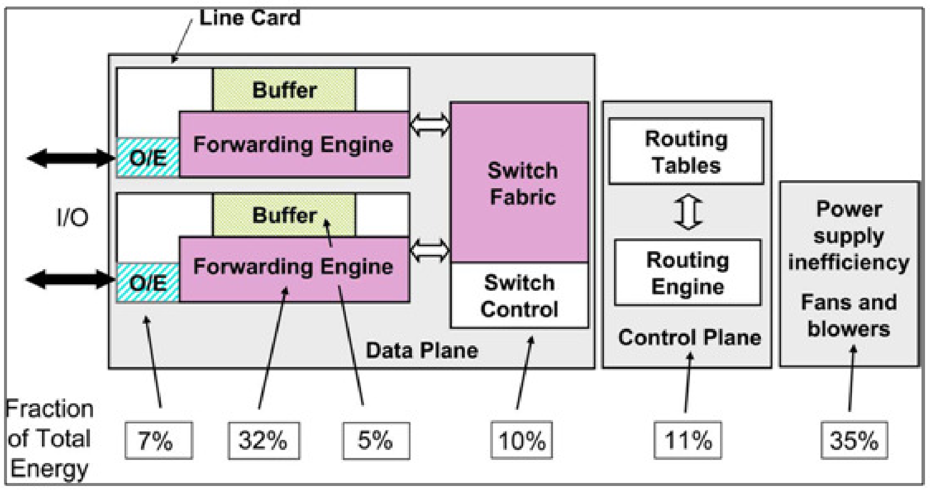


Figure 29: Energy consumption across the functional blocks of a high-end core router [9]

Centralizing the control software often means an automatic reduction in switch power consumption offset by consumption of a new element. As we can see from [9], most of the power consumption stems from the Forwarding Engine (32%) and the Cooling/Power Supply (35%). However, a rather small, but still significant amount (11%) of the power is consumed by the control plane. Split Architectures with OpenFlow enables us to reduce this part by moving the routing tables (RIB)/routing engine and control plane functionality to the controller and keeping only the forwarding engine (FIB) in the switch with a smaller OpenFlow software component and extra hardware for handling communication with the controller. However, the controller will consume more power due to additional computational loads. As a result, it is unclear if an OpenFlow architecture per se will actually reduce power consumption compared to conventional network architectures.

3.7.2.1 OpenFlow extensions for energy-efficient networking

The OpenFlow architecture enables us to implement *optimization of network topology*, e.g., MLTE operations as an application in the OpenFlow controller. In order to reap the maximum benefit, the controller should be able to power up/power down parts of the switch on demand as a function of the energy-efficient algorithms in the application. To enable power management for static network topologies aw well, we need to add the *burst* and *adaptive link modes* in the switches and advertise them to the controller. On the controller side, it should be extended to allow control of such energy-efficient features.

To enable energy-efficient networking applications to run on the OpenFlow controller, some extra messages should be added to the OpenFlow specification which not only indicate the status of a port on the switch, but also allows us to control the individual ports. In D4.2 we will present details of our proposed extensions for dissemination of power management capabilities, for monitoring the related switch parameters and for controlling these capabilities.

3.8 Quality of Service

To understand the QOS requirements, the different logical blocks of a 10 gigabit Ethernet switch model with line cards and a backplane are shown with the potential stress points in Figure 30. This model is taken from an IXIA white paper [34]. It addresses six different typical stress points, typically designed to operate at line speed under certain loads within which they may become congested and cause increased latency, packet drops and other problems:

1. Ingress packet buffer

   The ingress packet buffer stores received packets that are waiting to be processed. These buffers may overflow if upstream nodes are transmitting packets faster than the switch can processes them.

2. Packet classification

   The packet classifier uses the header information that has been parsed from the incoming packets in order to classify them into different service classes. Depending on the design and the types of packets received, packet classification may not be able to operate fast enough. For example, complicated packets with multiple levels of headers (e.g., packets that are being tunneled) may require multiple table lookups which increase the required amount of processing time per packet in order to classify them.

3. Traffic management

   The traffic management modules are responsible for applying QoS through the mechanisms discussed below. During high loads these modules may become heavily stressed as queue management algorithms such as random early detection are activated in an attempt to reduce the load.

4. Crossbar switch and backplane interconnect

   The crossbar switch and backplane interconnect are responsible for transferring packets between different connected line cards. Depending on the architecture of the particular switch, the backplane may cause blocking under certain traffic patterns. Advanced queuing and scheduling algorithms, combined with fast interconnects, can limit this problem or even completely resolve it.

5. Multicast replication and queues

   Multicast replication is usually performed in two stages, one at the ingress line card in order to multicast the packet to different egress line cards, and another at the egress line card(s) in order to multicast the packet to multiple ports. Multicast packets compete for the same resources as the unicast packets and may be the cause of congestion both at the egress ports and when going through the crossbar switch and the backplane.

6. Control plane

   The rate at which the control plane is able to update the tables used by the switch, for example in forwarding tables, may cause problems in error conditions when a high number of changes are performed per second.

As we can see, congestion can be caused by traffic consisting of complicated packets or multicast. However, even simple unicast traffic may be the most obvious cause of congestion when multiple ports are trying to forward traffic through the same outgoing port and the combined packet rate/bit rate is higher than the line rate of the outgoing port. This may cause congestion first in the egress packet buffer, which in turn may cause blocking in the crossbar switch, which in turn may cause congestion in the incoming packet buffer.

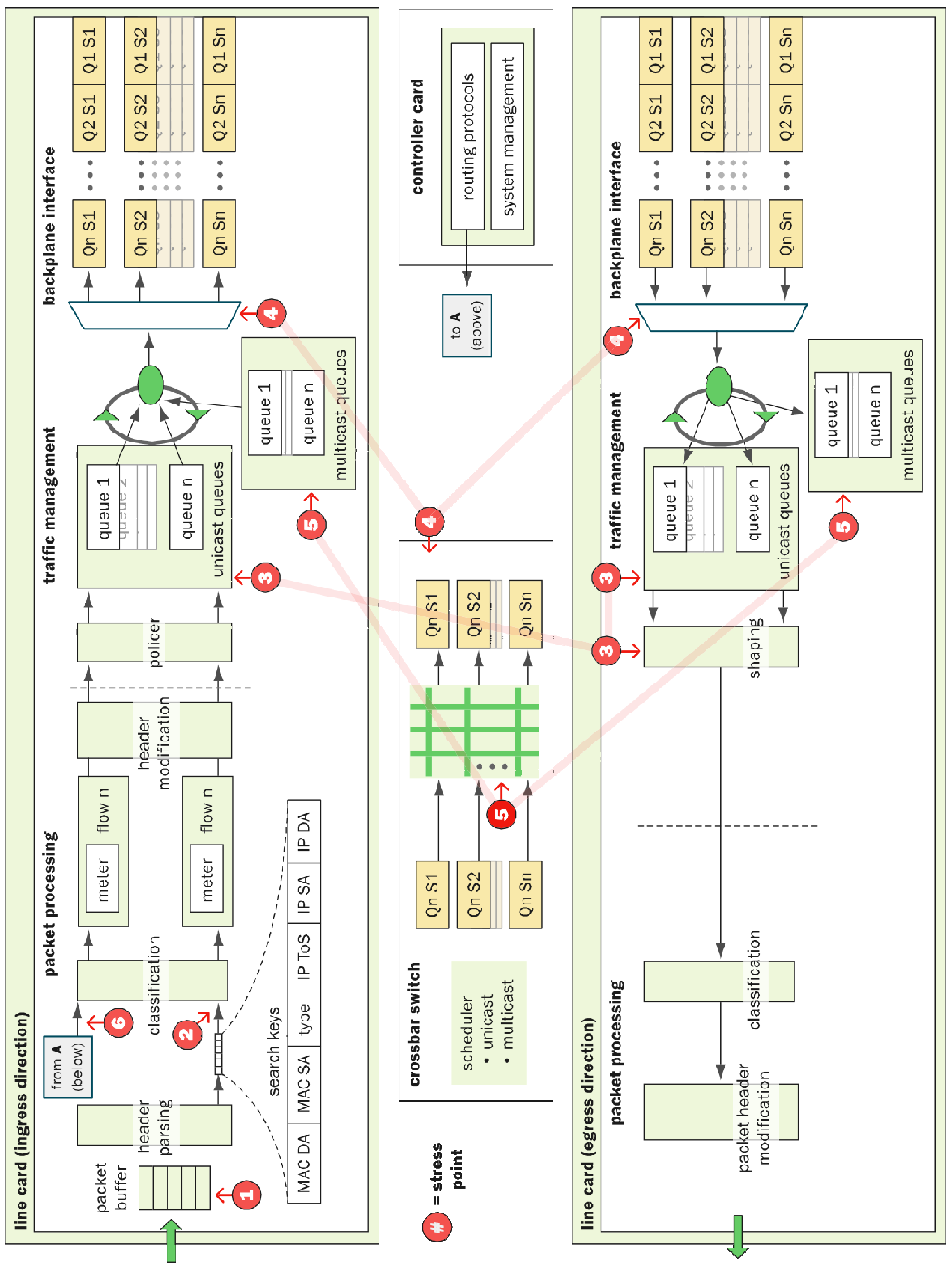


Figure 30: Logical operation of a 10 gigiabit Ethernet switch with line cards, from [34].

3.8.1 Queuing management, scheduling, traffic shaping

Typical quality of service (QoS) implementations in most routers and switches are constructed from a number of tools that are able to affect incoming packets in order to prioritize certain traffic, ensure proper bandwidth usage, smoothen traffic flows and reduce congestion in the network. Normally five methods are used for this: classification, traffic policing, traffic scheduling, traffic shaping, and packet rewriting. Figure 31 illustrates an example of QoS processing chaining; the order of the tools (or the tools used) in the figure are in no way canonical – different QoS goals may require different tools in a different order. Exactly which capabilities are needed by the different tools again depends on the requirements in a particular instance, as well as where they should be placed in the packet processing chain from incoming port to outgoing port. As shown in the figure, traditional devices may allow some QoS actions to take place before the packet enters the actual switching/routing stage, for example policing, or allow queue management actions on the incoming packet buffers. For a more detailed discussion of these subjects see [35]and [36].

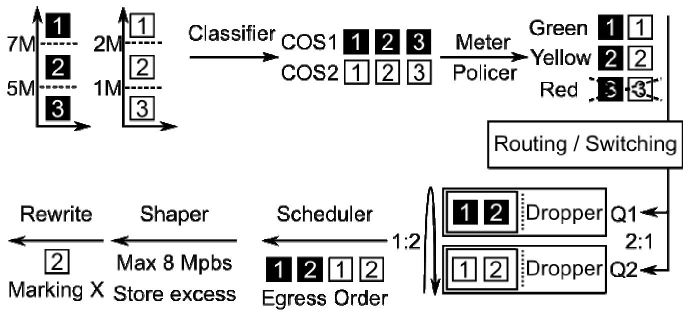


Figure 31: QoS processing pipeline where packets from different flows are classified and metered before going through the routing/switching stage. The packets that were not dropped are then scheduled, shaped, and rewritten.

Classifier

The classifier is responsible for recognizing different traffic flows based on a number of criteria, for example L2/L3 addresses, TCP/UDP port number, explicit packet marking (DSCP/802.1p), incoming physical port, etc. Classified packets are placed in a service class that shares a common QoS treatment in the switch.

Metering and coloring

Traffic metering and coloring measure the packet rate and assigns colors to the incoming packets based on their arrival rate. Typically a dual-rate mechanism is employed: Packets not exceeding the first rate are colored green, packets exceeding the first rate but not the second are colored yellow, and finally packets that exceed both are colored red. Typical implementations may use token bucket-based algorithms such as the “two-rate three-color marker” [rfc2698] or “virtual scheduling” as used in ATM networks [37].

Policing

Traffic policing is used to enforce a maximum transmission rate based on the colors assigned to the packets. For example, red packets may be dropped immediately while yellow packets are candidates for dropping, whereas green packets should be allowed through without any restrictions.

Shaping

Traffic shaping is used to not only enforce a maximum transmission rate, but also to smoothen the traffic flow. This may be based on the same coloring used for policing, or new measurements may be made. Instead of dropping the offending packets, they are placed in a buffer until they may be transmitted without violating the defined packet rate or bit rate. However, if the shaping buffer fills up, packets may even be dropped there.

Scheduling

Packet scheduling is responsible for selecting packets from queues representing different service classes and placing them in the output queue in a particular order. Many different scheduling/queue management algorithms exist, from a basic “round-robin” scheme to more complicated schemes such as “priority-based deficit-weighted round-robin” and congestion avoidance algorithms such as flow-based random early detection.

Rewriting

If one wishes to carry explicit service class or priority markers in the packets themselves (for use by other routers/switches), these are written to the packets by the packet rewriter which modifies the packets before transmission.

3.8.2 Improvement proposal

OpenFlow has basic support for some of the methods described above. For example, **classification** can be performed by matching packets in the flow table(s). Multiple matching per packet is not included in the specification OpenFlow 1.0, Thus it is not possible to decouple service class classification from flow classification (i.e., using one match for determining service class, and a different one for determining how to forward the packets). If one wishes to have a generic rule for a flow to perform forwarding but apply different QoS schemes, one needs to combine the forwarding and QoS rules, which leads to a linear increase of flow table entries (e.g., with four different QoS service classes the number of entries becomes four times larger). In OpenFlow 1.1 multiple matches are possible, so there is no need to combine the rules, and the number of entries may only increase with a constant number at the cost of an additional match. The metadata field available in OpenFlow 1.1 – which is associated with a packet while it traverses the processing pipeline – could be used to temporarily store the assigned service class for use in later matching stages if necessary.

Metering and coloring is not available in any OpenFlow versions as they have packet and byte per flow entry. However, these are used by the controller to gather statistics and cannot affect how packets are processed within the switch. Metering and coloring could easily be implemented in OpenFlow 1.1 as a processing action that measures the packet rate or bit rate and writes the determined color to the metadata field. Following tables could then use the metadata to allow the packet to continue in the processing pipeline or to drop the packet. Since shaping uses similar mechanisms, it seems it should be as easy to implement as a processing action. However, since shaping requires buffering of packets that exceed the packet rate or bit rate, it becomes more complicated. This would require that the processing pipeline be capable of temporarily storing some packets somewhere in the pipeline, continue processing incoming packets, and then pick up processing of the stored packets mid-pipeline when the shaping mechanism allows it.

Packet rewriting is supported by both OpenFlow Versions 1.0 and 1.1 with processing actions; in Version 1.0 by Ethernet 802.1p priorities and IP ToS bits; in Version 1.1 by MPLS traffic class bits. These fields may also be used as matching targets in the flow table(s). Matching and writing these fields makes it possible to interact with legacy systems on the QoS level, for example to utilize the QoS classification performed and written by legacy systems.

Scheduling has limited support (in both Version 1.0 and Version 1.1) with queuing schemes that can be attached to an outgoing port. Currently only a single type of queue is supported, namely the minimum guaranteed rate queue, which acts as a traffic shaper attached to an outgoing port. This queue allows one to reserve a percentage of the outgoing bandwidth of a port, but it is not defined whether it should take priorities into account, e.g., via the different packet marking mechanisms. This queuing concept could be extended with more advanced queuing mechanisms, supporting different kinds of queuing algorithms such as random early detection (RED). Additionally they could take into account not only the explicit packet markings (i.e., 802.1p, IP ToS or MPLS TC), but also use parts of the metadata field (only available in OpenFlow Version 1.1) to allow packet prioritization without explicit marking. For example, the packet metering functions could modify bits 0-3 of the metadata, or these bits could be modified by explicit metadata modification actions.

One major issue with the current queuing mechanism is that the queues can only be attached to outgoing ports – it is not possible to chain queues to other queues. This makes it difficult to construct hierarchical queuing structures, something which highly simplifies construction of typical QoS schemes. Using hierarchical schemes one can easily design complex QoS setups like the one depicted on the left in Figure 32, where two different organizations share the bandwidth of a single link. Within their shares they have further subdivided the bandwidth between a number of services, some with low-latency requirements and some with less stringent latency requirements. The bandwidth values are the guaranteed values for these classes. However, if there is leftover bandwidth one level higher up in the hierarchy, this bandwidth may be used without violating any guarantees. For example, organization B may be using 75 Mbps of the total capacity if organization A is not currently utilizing all of its guaranteed bandwidth. Packets that do not match any of the defined QoS service classes may also utilize any leftover bandwidth, provided that no guarantees are broken. This type of sharing of the leftover bandwidth is difficult to manage in a fair way with a flat organization (on the right in Figure 32), whereas with a hierarchical organization it is easy to determine how the share of the leftover bandwidth should be distributed among the different service classes.

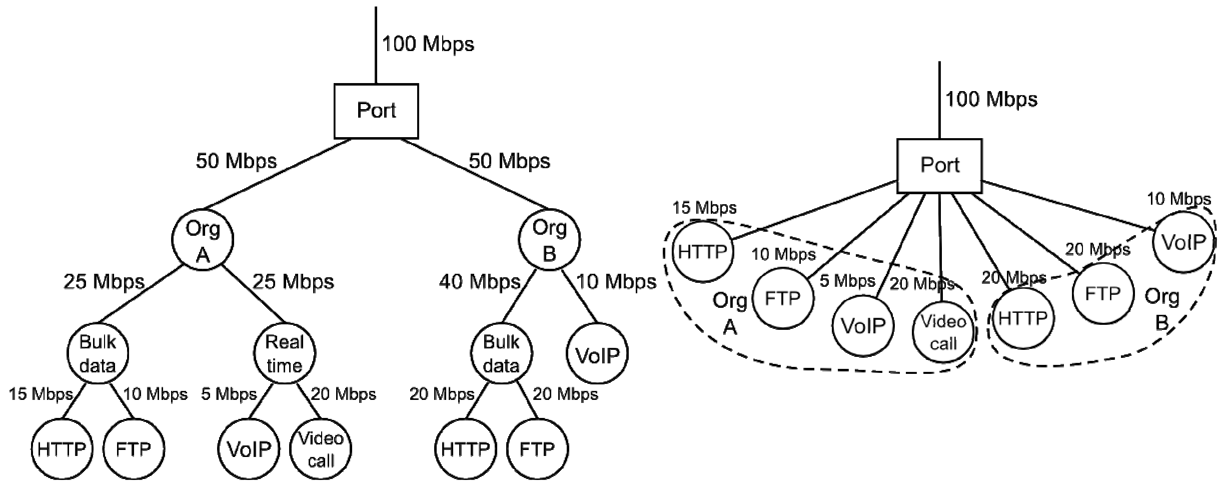


Figure 32: The hierarchical and the flat QoS model

The hierarchical QoS model also fits very nicely with the virtualization concepts discussed in Section 3.2. In this case the network operator can create and attach queues to the ports per virtual network. These queues are then presented as physical ports to their respective virtual networks, which can attach their QoS classes directly to this queue (see Figure 34). This not only insures isolation between the different virtual networks and allows complex queuing setups within the virtual networks, but can also allow the creation of additional virtual networks within previously defined virtual networks (i.e., recursion).

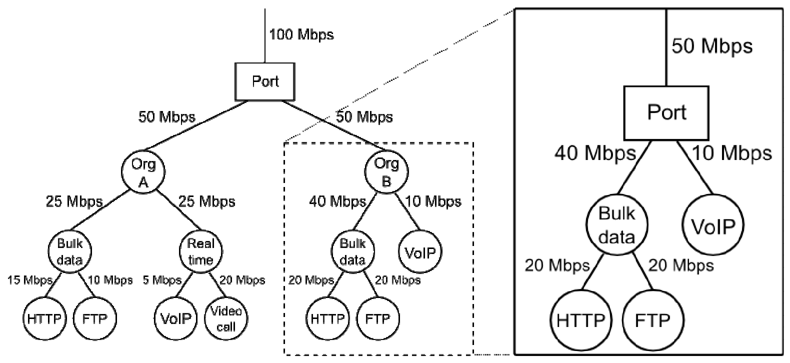


Figure 33: Representing a leaf of one QoS hierarchy as the root of another, virtualized one.

3.9 Multilayer Aspects: Packet-Opto Integration

Multilayer nodes are often also called multi-region nodes (RFC5212), exhibiting at least two different switching capabilities (ISC). This section deals with the integration of optical networks and OpenFlow, but the main problem descriptions and solution proposals apply to wireless interfaces as well.

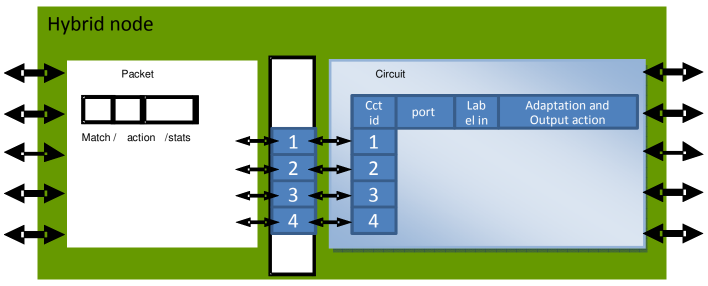


Figure 34: Hybrid node, composed of a packet OpenFlow switch and a circuit (TDM or WDM) switch.

Introduction of current generation optical networks into the domain controlled by OpenFlow is a tedious task because of the complexity of optical signal transmission:

- While an Ethernet **port** in a conventional OpenFlow switch will have little more status than being "UP" or "DOWN", WDM ports are characterized by the number of channels available, the nature of the transceivers (e.g., short/long range), available bit rate and in the near future even the size of the available spectrum. The transmission via Ethernet cable is assumed error-free (because the signal is terminated and regenerated at each switch). However, the **quality of the received signal** in an optical path varies as function of the length of the optical path, the number of hops without full regeneration, the number of active wavelengths, the modulation scheme, etc. This has the side effect that signal quality is attributed to an optical path (a flow) instead of to port.
- While **actions** in OpenFlow are typically coded as packet header modifications, and output actions, which can be arbitrarily defined, optical crossconnects differ in the available switching capabilities. As an example, ROADMs can be fixed or colorless, directive or directionless, contention-prone or contention-less. These switching constraints are typically non-existent in Ethernet, and consequentially in OpenFlow switches. In addition, the number of transponders in a so-called "hybrid switch" (switch with non-homogeneous ports, see Section 4.2.1 of RFC5212) limits the throughput.
- While a **label** in OpenFlow can be directly matched on the packet headers, the label information for wavelength or OTN channels is indirect. Encoding for labels beyond packet transmission has been introduced along with GMPLS (RFC3471) and has been extended. All these labels, however, are only visible in the control plane and need to be mapped to a certain configuration of the switching matrix in an optical crossconnect.

3.9.1 GMPLS and OpenFlow

Most of the problems described above were encountered some five years ago in the IETF GMPLS working groups (ccamp) when defining GMPLS extensions for wavelength switched networks (WSON). There is no obvious reason to invent new encodings for ports, labels, adjustment capabilities and optical impairments. This was recently determined in an internal project of one of the SPARC partners [38].

Encodings	IETF GMPLS/WSON	OpenFlow 1.1.0 spec
Port	RFC4202 (PSC,L2SC, TDM, LSC, FSC) RFC4206, RFC5212, RFC 6002 (DCSC)	Section A.2.1: `enum ofp_port_features`
Match (Flow label)	RFC 3471, RFC4606 (TDM), RFC6205(WDM), draft-li-ccamp-flexible-grid-label-00 (FlexGrid)	Section A.2.3: `struct ofp_match`
Action (Adjustment capacity)	RFC 5212, RFC5339, RFC 6001 (IACD)	Section A3.6: Supported action bit field in `struct ofp_table_stats`
Flow stats (Impairment encoding)	draft-bernstein-wson-impairment-encode-01.txt	Section A3.6: `struct ofp_flow_stats`

Table 3: Options for the transport of encoded information in OpenFlow

The question here is rather how and where these encodings can be introduced in OpenFlow. It turns out (see Table 3) that potentially all of the peculiarities of optical transmission can be architecturally covered by the existing OpenFlow standard 1.1.0. However, some of the mappings require a different understanding of the nature of a *flow* in optical networks. This is especially the case for the use of `ofp_flow_stats` for the encoding of optical impairments. In fact, it is the different nature of the optical flow that prohibits flow statistics relying on packet counts.

3.9.2 Virtual ports vs. adaptation actions

As discussed in Section 3.1.2, there would be a functional equivalence between virtual ports and potentially complex "processing actions" if the latter would be able to keep state information. For optical networks relying on circuit switching, one could create virtual ports that identify an established circuit, thereby hiding the optical details of this circuit and making the port appear again as a regular OpenFlow port.

All configuration of this virtual port would go through a config protocol, creating/updating/deleting the port, but also managing its internal behavior. The virtual port concept would split the hybrid node into two nodes (see Figure 34), a conventional (packet-switched) OpenFlow controlled part and another part that may or may not be under the control of OpenFlow.

Adaptation actions have the potential to pull the encapsulation/decapsulation into the domain controlled by OpenFlow, making the use of a separate config interface for virtual ports superfluous/obsolete.

Capabilities are bound to tables, starting from OpenFlow 1.1, which means that a specific encapsulation/decapsulation action like, e.g., the Generic Framing Procedure (GFP), could be called as an action on a flow. However, output actions are not yet port-specific, which means that a GFP-mapped TDM signal could end up in an Ethernet port by misconfiguration. It would be helpful to associate ports to tables for a clean configuration of hybrid nodes.

3.9.3 Three phases of introduction

OpenFlow can be gradually introduced to control optical network equipment, leveraging on existing control plane implementations. This may be advantageous as it saves development cost and may as well protect IPR used in the Path Computation Elements (see RFC4655) of vendors. The following three phases follow the implementation time line in the project OFELIA as presented in [39].

3.9.3.1 Interworking

In this first step of packet-opto integration, the optical domain appears as the backplane of a single Ethernet switch (Figure 35). All optical features (ports, labels, actions) are hidden from the OpenFlow part and handled entirely by the optical control plane (e.g., GMPLS).

Operation is such that a `packet_in` message from one of the hybrid switches triggers the setup of a light path via the GMPLS UNI. This means that the `flow_mod` entries that are generated by the OF controller for the hybrid switches are following the establishment of a light path between the transponders (appearing as virtual ports between the packet and the circuit switch part of the hybrid switches).

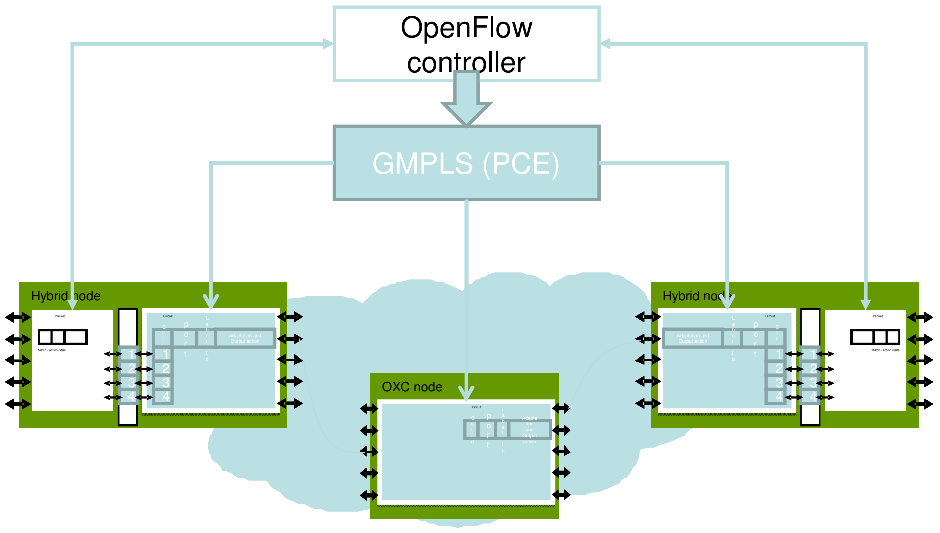


Figure 35: Interworking of OpenFlow and GMPLS

3.9.3.2 Abstracted optical layer

A second step of integration is the OpenFlow control of optical nodes (Figure 36). This means that each network element has an agent translating the received OpenFlow messages into local (typically SNMP) control commands. A `flow_mod` from the controller is then used to configure an entry in the switching matrix.

Path computation will still be done in a GMPLS PCE, but all configuration of the network elements would now go through the OF controller.

On the OpenFlow side this step will require encoding of label types and adaptation actions. Ports can still be considered abstract, as they only appear after a feasibility check

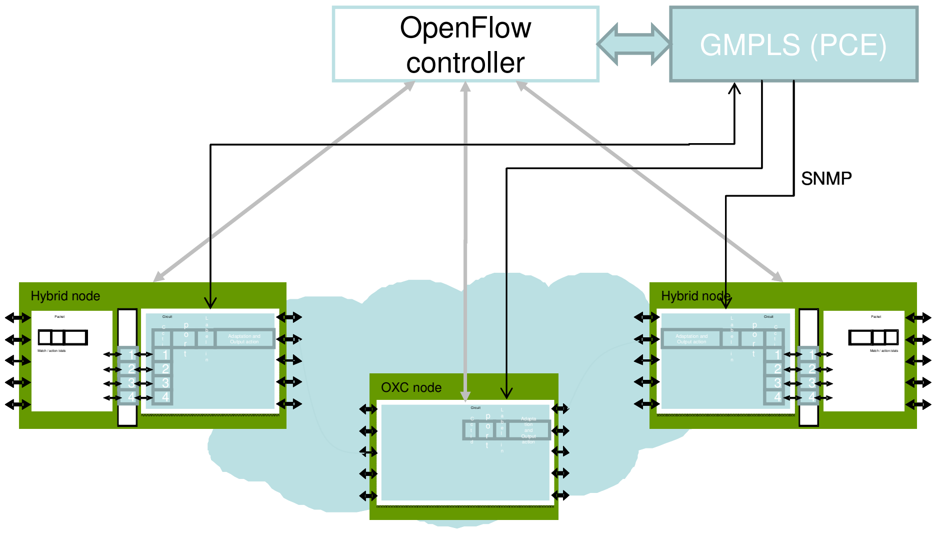


Figure 36: Direct control of optical network elements by the OF controller. Path computation is still being done in a vendor's control plane.

3.9.3.3 Impairment-aware OpenFlow

The ultimate step integrating packet and optical transmission will be the mapping of the attached transport technologies (i.e., ISC) to flow tables. This will require association of ports to flow tables, the definition of impairment-annotated flow_stats and the path computation as an integral part of the multilayer OpenFlow controller, as indicated in Figure 37.

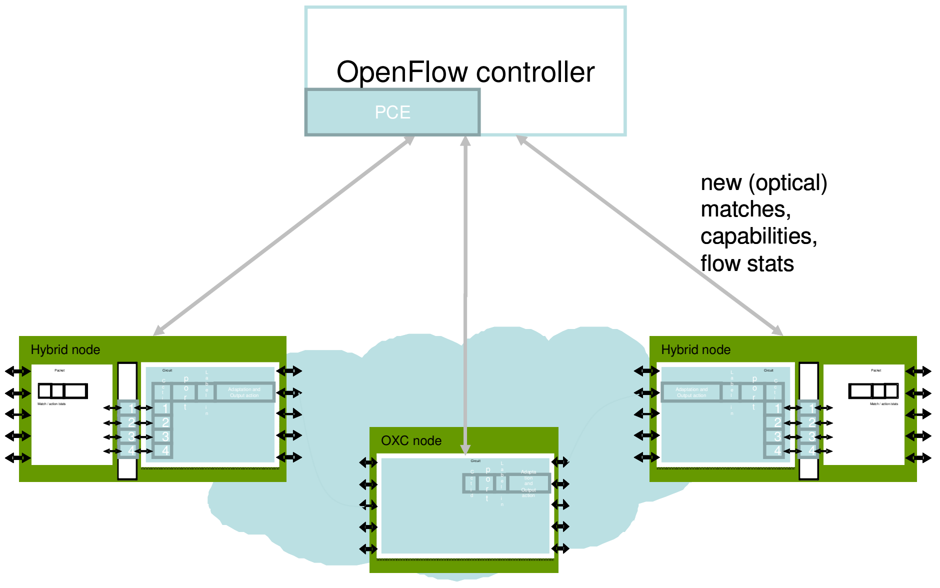


Figure 37: Integration of a PCE into a multilayer OpenFlow controller

4 Implementing Carrier-Grade Split Architecture in an Operator Network

4.1 Current Operator Networks

This section presents details of today's typical hierarchies in operators' networks. The targeted general data plane architecture for the main use case considered, namely access/aggregation networks, is described in Figure 38 (Section 2.2.1.1 of D2.1).

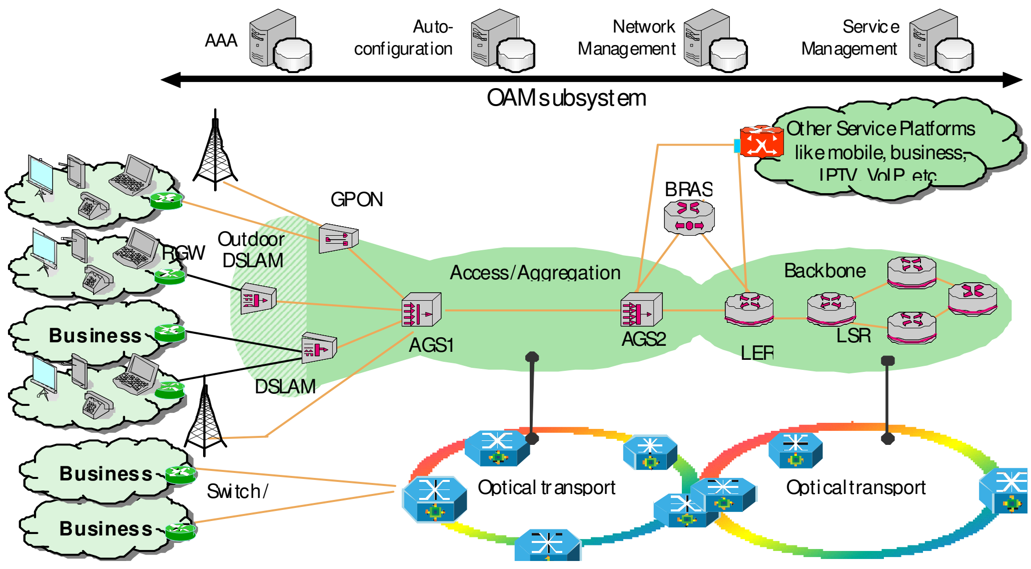


Figure 38: Access/Aggregation Data Plane Architecture

The hierarchical structure is like a snowflake: one meshed core and multiple tree/ring access/aggregation areas. There are three layers present: the optical transport layer, the packet layer (in green) and the application services layer.

The optical transport layer implements all functionalities to transmit the traffic from the customer edges toward the service edge devices and back. It can be either statically deployed or dynamically configured. Some consideration about multilayer aspects (including optical integration) in a Split Architecture scenario can be found in Section 3.9.

The packet layer is typically split in between service and transport, the former represents the logical link between customers and service creation platforms (corresponding to the network service and user session in Section 3.6); the latter represents the aggregated customer transport services (corresponding to the transport function in Section 3.6). Packet transport services are established between customer edges (typically a Residential Gateway, RGW) and edge devices like BRAS or LER.

The application services layer (corresponding to the service slice in the user session in Section 3.6) provides the "real" services for customers between end customer devices (e.g., PCs or set-top boxes) and devices in service platforms, e.g., servers for voice services or serving web pages.

There are no direct connections between different aggregation domains at the packet layer – they are connected by the core network. For the deployment scenarios described in this section, we focus on the packet layer only and we mainly consider MPLS as end-to-end packet layer technology.

Currently parallel developments are taking place in the access/aggregation network domain. These result from different, yet complementary motivations, outlined in the following subsections.

4.1.1 Convergence of transport networks

There is a trend toward an integration or convergence of separated networks with similar packet transport requirements. Typically one transport network based on optical and/or packet technologies is used between end customers (or mobile base stations) and the core network border, but different core network edge devices terminate the transport services and forwards them to separate core networks depending on the type of service. In principle, residential customer edges for Internet services, IPTV or voice, business customer edges and mobile network edges do not differ in terms of transport technology or to a large extent in terms of the control plane for the forwarding because of the convergence to IP/MPLS based networks. Therefore, the integration of the different edges into one edge seems to be a logical step as depicted in Figure 39.

Today a typical operator has at least three different edges: mobile, business and residential IP (typically BRAS-based), and potentially some more for different service platforms. These different edge devices are converging into one edge device, typically referred to as the Broadband Network Gateway (BNG), which integrates all functions related to transport and network service (to a large extent similar functions), e.g., BRAS for residential customers, IP/MPLS edge for business, mobile edges (depending on the different evolutions of 3GPP and LTE, different names are used – not shown in Figure 39) and potentially service gateways for IPTV (including multicast and access control) and voice services (not shown in Figure 39). Further integration could be achieved with appropriate virtualization and isolation techniques, so that edge devices of 3third parties could be integrated in the same device as well (see Section 3.2).

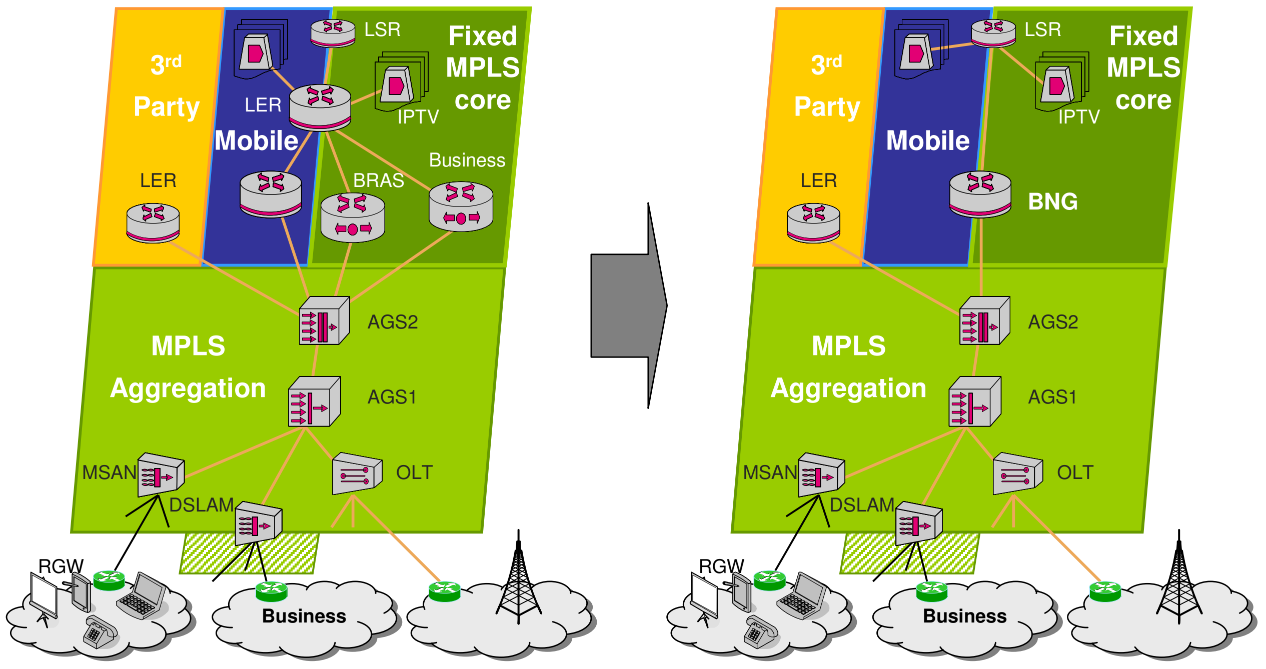


Figure 39: Convergence of transport network edge devices

4.1.2 Distribution of service functions

At the same time, there is a shift of functions from a more centralized model to a more distributed one. For example, BRAS functions or IPTV service functions are placed closer to the end user, e.g., already at the first aggregation switch (AGS1), and not in the edge devices of the core network. Section 2.2.1.4 of D2.1 explains that mobile backhauling requires additional functions closer to the mobile base stations as well. An example is shown below in Figure 40 describing three different models for distributing the functionality of the current IP edge in a residential network scenario.

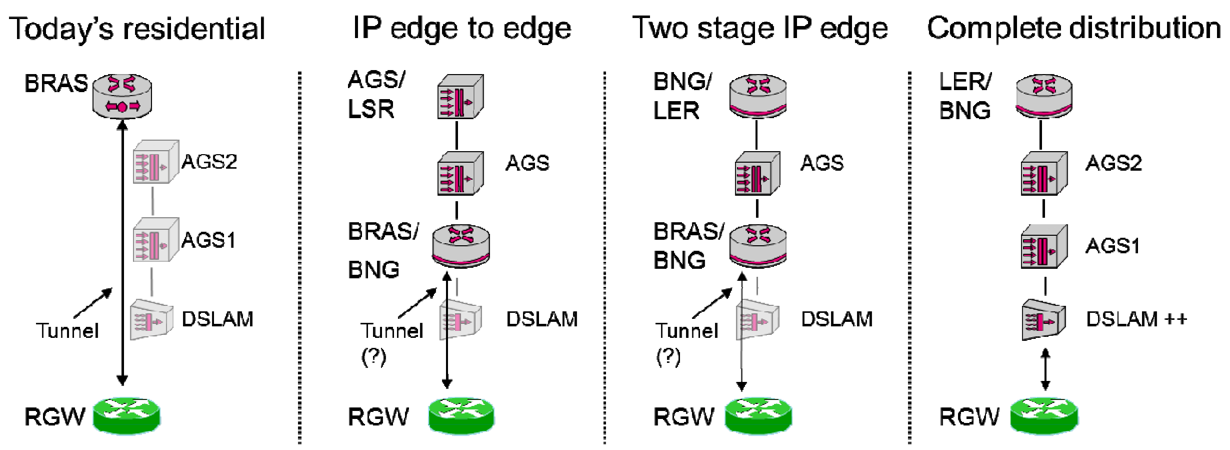


Figure 40: Evolution of degree of centralization w.r.t. placement of the IP edge node in a DSL-based network

Today's model is typically based on a BRAS as service creation element using the PPP protocol to tunnel between the customer and the BRAS. Thereby the traffic is transport transparently from the customer over the access/aggregation network to the BRAS. In the "IP edge-to-edge" model, the edge network functions are integrated in the BRAS or BNG (former AGS1) and the service creation is exactly the same from a customer's point of view. The former IP edge either becomes a switch (LSR or AGS) or is potentially removed completely. In any case, the IP/MPLS core domain is now terminated closer to the customer. In the second evolution model, called "two-stage IP edge", is similar to the previous one, but the IP edge at the border to the core network is not removed. Either a Label Edge Router (LER) or a BNG, depending on the demand of the network operator, will provide the second edge function for transport services (see Section 4.1.1). Again, from a customer's point of view everything remains the same. The IP edge at the border (BNG or LER) will be simplified because every customer related function (e.g., AAA) is moved to the AGS1. Overall the model provides some more scalability and flexibility for further steps. In the "complete distribution" model any customer related, intelligent, function is configured in the DSLAM, including MPLS tunneling as well as BRAS functionality, etc. Here simple transport mechanisms could be applied to the access/aggregation networks but, depending on the size of the network, another IP edge (LER or BNG) will be required for sufficient scalability because of control plane issues. In all of these models another service creation mechanism could be used between the RGW and the IP edge/DSLAM++, indicated by the "?" in Figure 40.

The important consequence is that the design of access/aggregation networks demands flexibility in the future in terms of location of network functions like AAA or policy distribution and associated scalability in order to enable migration paths or multiple providers in varying network scales or sizes.

4.2 OpenFlow in Access/Aggregation Networks

One essential question in the design of the Split Architecture is how to integrate OpenFlow into the existing network architecture. There are three aspects to consider: The first aspect is dealing with the *level of the hierarchy* at which we introduce OpenFlow. The second aspect relates to the *number of hierarchy levels* controlled by OpenFlow. Finally, the third aspect focuses on *which functionalities* are configured through OpenFlow. Considering these aspects, we will present three evolutionary approaches for today's residential service creation as in Section 3.6.3.

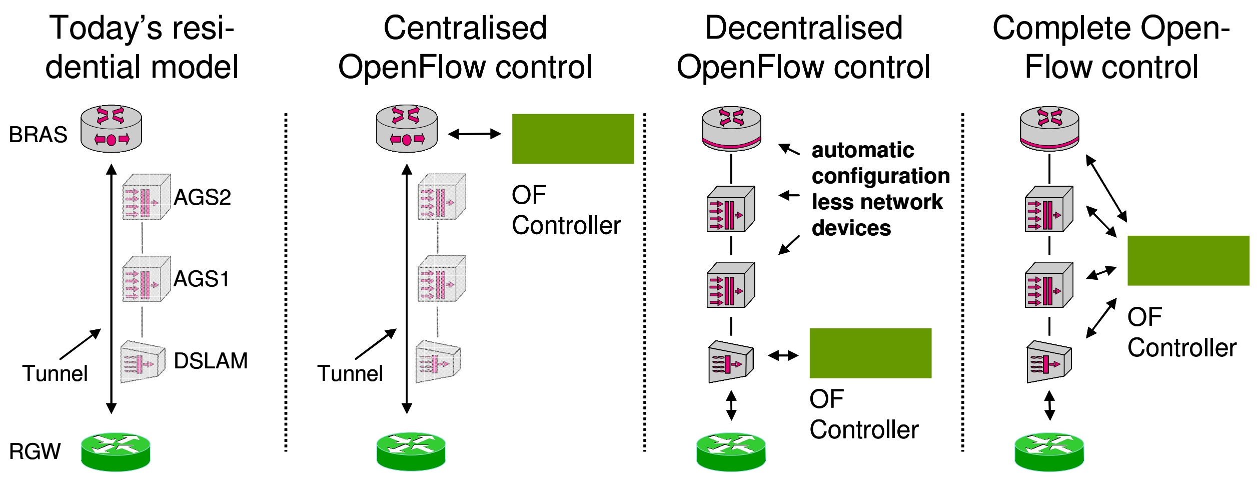

Today's resi-dential model
Centralised OpenFlow control
Decentralised OpenFlow control
Complete Open-Flow control
BRAS
AGS2
AGS1
DSLAM
Tunnel
RGW
OF Controller
Tunnel
automatic configuration less network devices
OF Controller
OF Controller

Figure 41: Three models for attachment of OpenFlow in access/aggregation networks

The "Centralized OpenFlow Control" model would be very similar to today's model with the exception that the OpenFlow controller would manage the central IP edge. This would result in a centralized network element in which potentially all customer traffic must be managed through OpenFlow with fine granular control – requiring powerful hardware. In addition, this model requires, similar to today's model, a mechanism to maintain routes to forward all packets from the RGW to the central element – the IP edge.

In the "Decentralized OpenFlow Control" model an OpenFlow controller would configure the DSLAMs. The difference to the previous model is that the DSLAMs control a significantly smaller amount of flows and bandwidth, but the OpenFlow controller needs to handle more connections to different DSLAMs than in the centralized model. The connection between the device and the OpenFlow controller, or between the OpenFlow controller and the network management system, requires additional network connectivity, which is implemented either with an out-of-band dedicated control plane network or multiplexed with the data links resulting in an in-band control network. Again, similar to the centralized model, the connection between DSLAM and the IP edge needs to be more automatic and the OpenFlow controller/subsystem needs to trigger the configuration of required transport connections.

In the "Complete OpenFlow Control" approach all devices, including the DSLAM, are controlled by an OpenFlow controller. Besides managing service configuration at the DSLAM, the controller is also responsible for provisioning transport functions of all forwarding devices along the network path to the IP edge. Note that, for performance and compliance reasons, mechanisms other than OpenFlow may be used for managing these transport functions, but how they are implemented depends on the capabilities of the OpenFlow controller

The decision favoring or rejecting a particular model is not part of the discussion in this section. Most likely there will be no clear general preference for a particular model in real network deployments, as this depends on several other aspects such as the size of the network, the availability of a hybrid OpenFlow switch model, which functionality needs be controlled, etc. The discussion of these models continues in the following Section 4.3.

4.3 Implementing OpenFlow for Residential and Business Services in Carrier Environments

In Section 3.4 service creation was introduced and detailed for residential and business services. Aspects of OpenFlow and related implementations were presented. From a technical perspective, the level of detail was not sufficient. Thus a detailed proposal for residential and business customers is presented here.

4.3.1 Residential customer service with OpenFlow

As described in Section 3.4, there are several solutions available with varying implementation options, see Figure 42 below for a general overview.

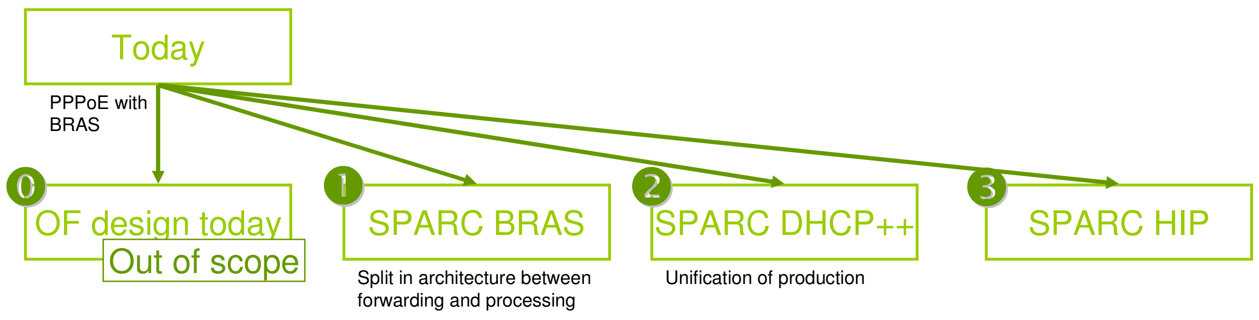


Figure 42: General implementation options for service creation

Option 0, "OF design today," follows the design rules and principles of the OpenFlow development done up to Version 1.1 – add any new protocol structure to the base protocol. As already discussed, this could result in an "explosion" of required protocol options, and hardware support might not be implemented in any case. In addition, the split between forwarding and processing could not be integrated. Therefore, this solution is considered out of scope.

Option 1, "SPARC BRAS," transforms current service creation design into the SPARC design principle of the split between forwarding and processing. The essential aspect in this solution is the required support for PPPoE processing, which could be hardware supported or emulated in various ways. Overall, several options are discussed in the following subsections.

Option 2, "SPARC DHCP++," uses the split of forwarding and processing, but essentially introduces a new set of protocols which needs to be supported in the OpenFlow environment. From a carrier's perspective, the model provides an integrative approach for the various service creation models used today. The target of this solution is to not introduce new required protocol support to the base specification, but to propose additions for required features in OpenFlow. At the current stage of the project, this model is considered out of scope and will be detailed in a future deliverable

Option 3, "SPARC HIP," is a very forward-looking approach enabling different new architecture features such as some kind of mobility support. At the current stage of the project, this model is considered out of scope.

Beside the need for legacy support for PPPoE, this section details two important requirements on OpenFlow improvements as seen from SPARC: The first one is decision logic for authentication and authorization. The authentication aspect is discussed a separate subsection in order to outline the requirements and the potential options in more detail. The second aspect is IPv6 and its integration in the architecture, and more specifically its impact on service creation. Work on IPv6 is continuing and will be detailed in a future deliverable.

4.3.1.1 Authentication

In Section 3.4, authentication has been identified as one of the requirements and important phases of service creation. Some more detailed information on the specific models of authentication for residential customers in the models based on PPP as used in the model SPARC BRAS and DHCP as used in the model SPARC DHCP++ is presented here.

The level of detail of authentication can differ depending on the desired level of information, ability of fine granular management and demands from legal aspects (which are omitted here). In general, there are two different levels: the authentication of a single user and the authentication of a connection per port. In the latter case, it could not be assumed that only one customer is authenticated. For example, fixed Internet dial-in environments (e.g., POTS, ISDN) were authenticating a single user only (and demanded the function for multiple users per dial-in connection) until the wide-spread deployment of residential gateways in xDSL environments, which typically uses only one set of user name/password for the whole group of users connected to them. Therefore, the most common model today is the authentication of a connection per port. Similar models exist in mobile environments as well, where the definition of connection/port differs, but uses the same principle. The authentication of a single customer could be required in different situations/cases:

- Each customer needs to be authenticated in the case of multiple customers per connection per port.
- Customer authentication in nomadism and a number of mobility cases.

Another important reason for authenticating on the port level is the configuration of a customer / service specific profile (parameters such as QoS profiles, nomadism, access to service platforms, etc.). In general the same reasoning applies for the authentication of a single user and therefore this model should be combined with the authentication of a port.

Port-level authentication is typically handled through the help of an (virtual) identifier (defined as Line ID) and a mechanism to include this identifier in the connection setup process. In principle, the Line ID could be anything that uniquely identifies the port (the process of creation/provision of the LineID is not yet standardized). Since the authentication is performed centrally at the BRAS node, a mechanism to pass the Line ID (however defined) to the BRAS is essential. The mechanism to include the Line ID in the connection setup process depends on the protocol used in the service creation. For PPPoE, this is done by the PPPoE Intermediate Agent. In DHCP a similar function exists with option 82 (DHCP Relay Agent Information Option) function, integrated into the DSLAM.

With OpenFlow authentication can be a rather simple or extremely complex processes depending on the two outlined authentication targets (single user or connection/port). Both mechanisms in today’s carrier environments (PPPoE Intermediate Agent and DHCP option 82) could be supported easily with the right filtering mechanisms in the DSLAM and forwarding to appropriate processing, or through the use of the hybrid mode and ignoring the authentication messages from the OpenFlow-configured part of the switch. On the other hand, the port information is integrated in any OpenFlow packet-in message sent to the controller. Therefore, the information about the port could be used in any controller application that in turn could, e.g., fetch it from a database attached to the controller. This approach would require OpenFlow support in the DSLAM or any other network device terminating the customer’s access line. Otherwise appropriate port information would not be available for the controller or would have to be added to the request (packet-in message) with appropriate mechanisms like adding a virtual identifier (e.g., VLAN ID) at the ingress of the DSLAM (the port) and the correlation of VLAN ID with the respective DSLAM port. Figure 43 depicts the principle of the PPPoE model and the potential application of the split control functions. Resource configuration is an essential part of service creation, but will be documented in an upcoming SPARC deliverable – it is included in the illustration in order to show the difference between the two models. Today service creation is performed centrally in the BRAS and the configuration information is transmitted via RADIUS. In an OpenFlow environment, an AAA application has to take over the task, or at least needs a link to the resource configuration system/function via the OpenFlow controller.

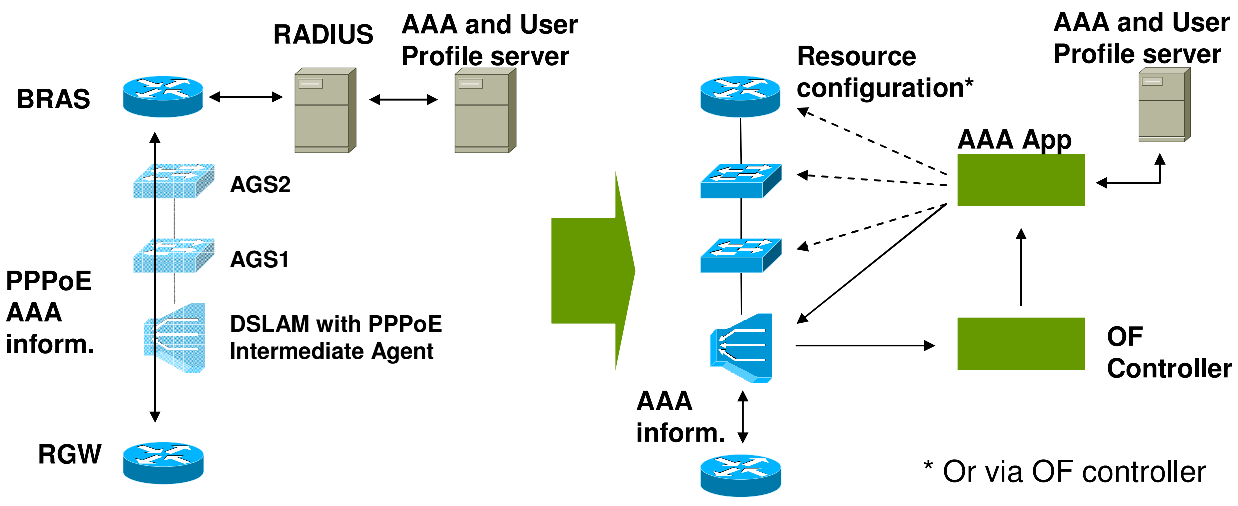


Figure 43: AAA integration in PPPoE and OpenFlow

Authorization is the step after authentication and could be handled in a simple fashion with OpenFlow too. The controller (or rather, an authorization application running on top of it) can simple send appropriate replies in "Packet-out" messages and configure the forwarding and processing policies at the DSLAM using e.g. flow modification messages.

4.3.1.2 Example: SPARC BRAS

This example is the simplest step forward for service creation in OpenFlow. It changes as little as possible compared to the base model, but leverages some advantages of the Split Architecture. The goal is to do no modification at all with respect to the customer's device in order to maintain complete support for legacy devices.

This option implements the split between control and datapath as well as forwarding and processing in the common service creation model *BRAS with PPPoE*. PPPoE is not a complex protocol as such, but the implementation requires high CPU and memory resources compared to what is typically available in conventional routers. This is due to the session state machine and its related configuration as a routing interface per customer (each customer is separated in an IP subnet). The break in control and datapath, an independent processing interface capable of PPPoE processing, could be implemented with the following two options:

1. The BRAS becomes a switch, the control part of the BRAS (representing PPPoE configuration messages as well as interfaces to databases like RADIUS) becomes an OF controller application and is moved to a computing platform (OpenFlow is used for configuration of forwarding entries and the processing interface).
2. Similar to option 1, but processing is done "somewhere" between customer and core;
    a. Processing in hardware.
    b. Processing in software.

The options 1 and 2 are illustrated in comparison with today's model in Figure 44. The two suboptions of option 2 are not indicated in detail.

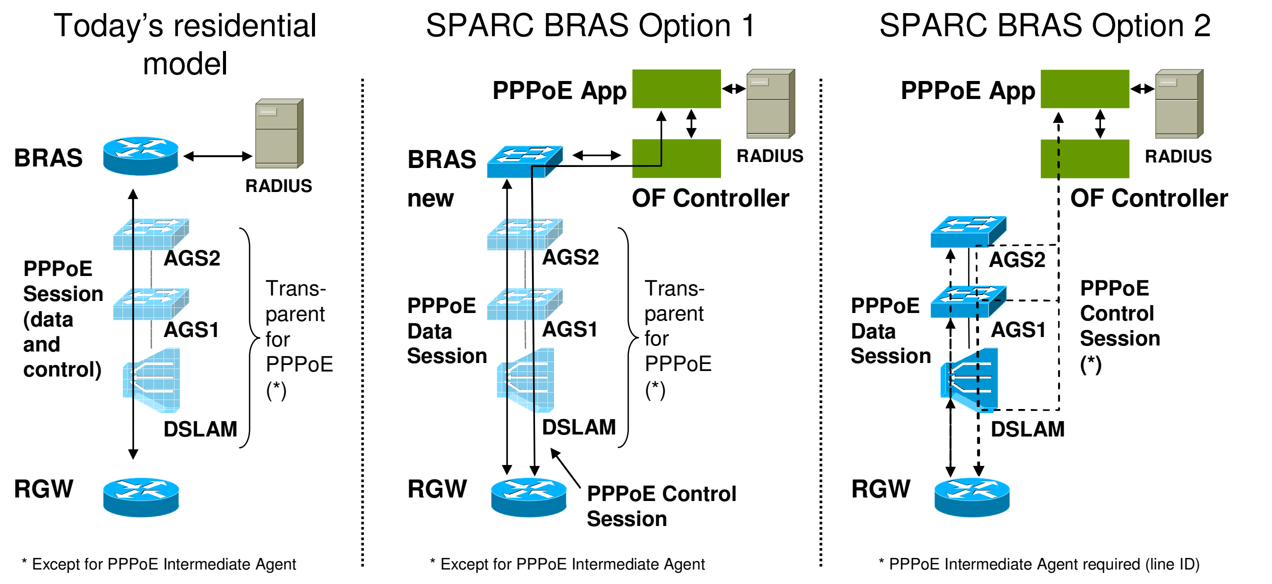

Today's residential model
BRAS
RADIUS
PPPoE Session (data and control)
AGS2
AGS1
DSLAM
Trans-parent for PPPoE (*)
RGW
* Except for PPPoE Intermediate Agent
SPARC BRAS Option 1
PPPoE App
RADIUS
BRAS new
OF Controller
PPPoE Data Session
AGS2
AGS1
DSLAM
Trans-parent for PPPoE (*)
RGW
PPPoE Control Session
* Except for PPPoE Intermediate Agent
SPARC BRAS Option 2
PPPoE App
RADIUS
OF Controller
AGS2
PPPoE Data Session
AGS1
PPPoE Control Session (*)
DSLAM
RGW
* PPPoE Intermediate Agent required (line ID)

Figure 44: SPARC BRAS options in contrast to today's residential model

The current model is already explained in detail in Section 3.6.3. The important aspect is that the data and control part of the PPPoE session is encapsulated and sent within the same session. Details about the general PPPoE protocol and its session state machine are included in IETF RFC 1661[4]

In SPARC BRAS option 1, a PPPoE application is introduced on top of an OpenFlow controller, which is connected to the new BRAS device, a more simplified BRAS compared to the current residential model. Simplification means that the control part no longer resides on the BRAS device itself, but access and understanding of PPPoE with related protocols is given, so a PPPoE session is still established. Between the new BRAS and the Residential Gateway (RGW) the session is logically separated into data and control, but in reality treated as it is in today's model. The difference resides in the handling of the control part of the PPPoE session in the new BRAS and the related introduced elements (the OpenFlow controller and the PPPoE application). The OpenFlow controller handles incoming requests without configured forwarding rules from the new BRAS, e.g., PPPoE requests. The parsing result is forwarded to the PPPoE application based on knowledge about the PPPoE protocol. The PPPoE application handles the PPPoE control session part of the protocol similarly to a current BRAS implementation. In response to a request, the PPPoE application triggers the new BRAS device (via the OpenFlow controller) to install the correct forwarding rules and the correct processing function for adding and dropping the PPPoE related headers (which are encapsulating the IP packets). These rules include a split between data and control session-related message flows and are the same as in SPARC BRAS option 2.

In SPARC BRAS option 2, the idea of option 1 is taken one step further. In option 1, the processing interface is based in the new BRAS, so the device is operated not in a routing mode, but in a switch mode and makes use of advanced processing capabilities like PPPoE data termination. This simplifies the control plane and associated resources in the device (e.g., routing interface per customer) to a large extent. In option 2, this processing interface is placed somewhere in the packet transport platform between RGW and core, either in AGS2, AGS1 or the DSLAM. Other aspects are similar to option 1. There are two options for the processing interface. Either a support of the processing directly in hardware (e.g., in the switching chip set of the DSL line card) or an alternative, realize the processing in software running on a generic compute hardware. The second option could be a processor somewhere in the device or attached to the device, e.g., the main processor in an aggregation switch. The flow table entries in the TCAM must be set/assigned to appropriate output ports like a virtual port for the PPPoE processing.

In Annex A, SPARC BRAS option 2 will be described in more detail, including a discussion of PPPoE integration in a DSLAM.

4.3.2 Business customer services based on MPLS pseudo-wires with OpenFlow

Business customers use a number of different technologies to interconnect locations, and carriers provide different options for service creation of these services. In order to analyze the impact of OpenFlow on the service creation, and to have a meaningful relation to the work done in the demonstrator development in SPARC WP4, it was restricted to Ethernet services and MPLS first.

In Figure 45 a provider edge router providing pseudo-wires is illustrated; while this example deals with Ethernet over MPLS, the same general architecture is used for multiple types of pseudo-wires. Frames are received from a customer edge switch/router and are first processed in the "Native Service Processing" (NSP) module – this refers to Ethernet-specific processing such as modifying VLAN tags, priority marking, bridging between different pseudo-wires, etc. Once through the NSP module, the "PW Termination" module is responsible for maintaining the pseudo-wire by, e.g., encapsulating/decapsulating frames and performing any necessary processing such as buffering in case of ordered delivery. Finally the packets are delivered to the "MPLS Tunnel" for MPLS encapsulation and transmission across the network.

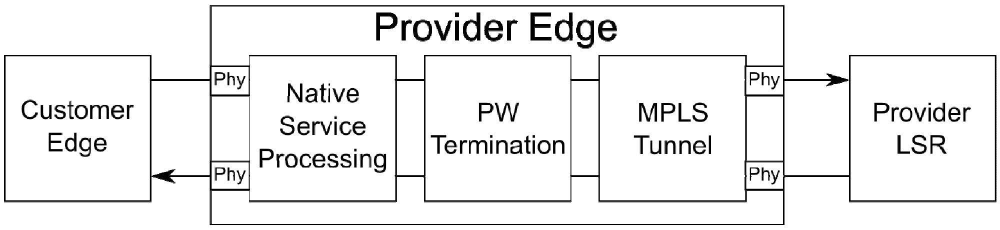


Figure 45: Pseudo-wire processing in Ethernet over MPLS pseudo-wires (based on Fig.3 of RFC4448)

In OpenFlow Version 1.1 this model can be followed, for example, by having one flow table per module, with the encapsulation/decapsulation actions implemented as vendor specific actions, and any additional processing managed either through process actions or virtual ports (as discussed in Section 3.1). If we focus on MPLS pseudo-wires with Ethernet emulation (and for the moment ignoring other types of pseudo-wires), we need a number of new actions corresponding to the different steps described above. First, at the ingress of the pseudo-wire, we need an action that takes an incoming Ethernet frame and turns it into the payload of a new frame. Once the new frame has been created, we need to add the control word. From this point on we can use existing actions to push MPLS labels, set the correct EtherType, and add correct MAC source and destination addresses. At the egress of the tunnel this should be mirrored by the inverse actions, in particular one to strip the outer headers and convert the payload back into an Ethernet frame.

4.4 Split Control of Transport Networks

4.4.1 Cooperation with legacy control planes

As discussed in Section 2.1, there are four stages when introducing OpenFlow in an access/aggregation network. In the first two stages, OpenFlow-based control solutions configure the transport nodes, while in the third state the service functionalities are implemented through an OpenFlow-based centralized control scheme. In these three stages, some nodes and/or some functions of those nodes are configured with protocols other than OpenFlow; only the fourth stage considers OpenFlow to be the sole forwarding node configuration protocol.

There are several OpenFlow integration options (as discussed in Section 4.1) in the third stage: the centralized and the distributed OpenFlow control. In both options, the intermediate transport nodes are still not configured via OpenFlow, but provisioned through the management plane or by making use of legacy control protocols. For example, an Ethernet-based aggregation switch can be configured using SNMP. Therefore, the OpenFlow controller should be able to interact with a management/control entity responsible for provisioning the transport connections. For this purpose, an interface between the legacy management/control entity and the OpenFlow controller is essential. This raises the demand for cooperation with legacy management or control planes. In our above example, the OpenFlow controller must be able to communicate with an entity responsible for managing Ethernet transport and thus for provisioning connectivity between the service nodes (e.g., DSLAM, BRAS).

The fourth stage, in which all aspects of all aggregation nodes are configured via OpenFlow, does not require such "vertical" cooperation with non-OpenFlow control entities. Even in this latter case, the whole network, as shown in Figure 38, is still not under the control of OpenFlow due to these aspects:

- Mature control planes are deployed in some network segments and any gain of substituting them with OpenFlow is not clear, and
- Covering a whole network of thousands of nodes with a single controller entity raises scalability issues.

In any of the discussed implementation cases, the OpenFlow controller must be able to cooperate with the other domains of the operators network. It is important to emphasize that any kind of control solution (centralized, distributed, static, dynamic, etc.) can be deployed in those domains. Therefore, the controller must be able to cooperate with the control function of those domains; this is referred to as horizontal interworking or peering and raises two major issues.

The various control functions use different resource description models. For instance, the MPLS control plane was designed as a protocol running between physical nodes. Hence, the internal structure of routers is less relevant and information about the internal details of the nodes is not disclosed as a simplified view only. This view encodes the router with its interfaces and capacity information, assigned only to these interfaces. GMPLS control follows similar abstractions even in multilayer cases: All information is tied/bound to the interfaces of the nodes. In the case of WSON, this abstraction level is changed by adding further details of the node's internal capabilities, but it still uses a generic model. On the other hand, an OpenFlow-based transport controller has much more detailed information about the managed domain. For horizontal interworking with legacy domains controllers via MPLS, GMPLS, etc., that information will be essentially filtered: A virtualized view of the managed domain is derived and provided toward the peering control entities.

Furthermore, the control plane entities may allow for different control plane network implementations: For example, MPLS supports an in-band control plane, where the protocol messages travel together with the regular data traffic. GMPLS is also able to operate with in-band control channels, but it also supports use of the out-of-band control plane. While the Split Architecture inherently supports the out-of-band control network, it can provide in-band options as well: The controller is able to instruct the data plane nodes to inject the protocol messages into the data stream toward the peer control node and to demultiplex the protocol messages received from the peer node.

4.4.2 Semi-centralized control plane for MPLS access/aggregation/core networks

Considering typical network structures (as shown on Figure 38 and Figure 39), the network consists of two parts: The access/aggregation using various forwarding technologies (e.g., Ethernet or MPLS), whereas IP/MPLS is the predominant technology in the core network segments. This also implies the control plane used in the core. This means that the controller, which manages the transport connections in the access/aggregation network segment, must be able to exchange IP/MPLS control protocol messages with the distributed IP/MPLS control plane of the core. The IP/MPLS control plane has both of the major issues discussed above.

As IP/MPLS uses link state routing protocols (either OSPF or ISIS) to keep the topology databases synchronized at the protocol speakers, a very simple network model is used. According to this model each protocol speaker advertises only its identifiers to the attached network and the detected adjacent routers, and by default does not report any information about its internal capabilities or structure. It additionally uses signaling protocols, e.g., LDP, to provision end-to-end MPLS label switched paths. However, the label distribution mechanism allows the adjacent nodes to agree on the used label value, but it does not instruct any node about how to configure its internal elements. This means that the controller must implement an IP/MPLS control-compliant view of the managed domain and a mapping mechanism between the physical network and the logical representation. As discussed, the IP/MPLS control plane is an in-band control plane, so the OpenFlow controller must be aware of that.

An extension to the link state routing protocols allows the assignment of further opaque attributes to the link. These additional attributes are also disseminated to other protocol speakers, although they do not carry any relevant information for the routing protocol. Assigning link characteristics such as available bandwidth, delay, etc., as opaque attributes supports implementation of traffic engineering in IP/MPLS networks. However, to provision traffic-engineered LSPs, an additional protocol is used: RSVP-TE. The IGP protocol must implement additional structures to advertise the opaque attributes as well, and such extended protocols are referred to as IGP-TE (e.g., OSPF-TE). These extensions convert the link database to a Traffic Engineering Database (TEDB).

4.4.2.1 Options for connecting the controller to the IP/MPLS control plane

The dissemination areas of the link state IGPs (OSPF/ISIS) define a structure for the IP/MPLS control plane. Adding all protocol speakers to a common dissemination area will result in an accurate view of the network at all speakers, allowing proper calculation of the LSPs. The drawback of such a solution is the scalability because the number of nodes of the same dissemination area increases. Several documents (see [40]) report that the while the core network can be covered by a single dissemination area, the whole network cannot. Therefore, in this section we discuss two alternatives of how a controller managing an access/aggregation domain could be attached to the distributed IP/MPLS control plane of the core.

A possible implementation is when the controllers act as simple IP/MPLS protocol speakers and they are attached directly to the core network's control plane, just like the simple core routers. Then the controllers and the core routers share the same dissemination area (OSPF area) as shown in Figure 46.

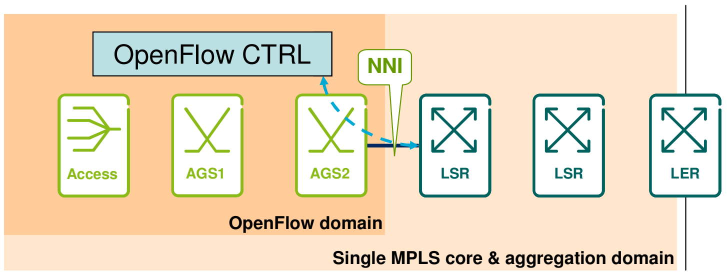


Figure 46: Single dissemination area option

Consequently, each controller has the same topology database as all the other core routers and controllers. Based on the shared database, every controller or router can initiate an LSP configuration to all other routers or controllers. The signaling protocols, LDP and the RSVP-TE are used to manage the LSPs.

In some cases, the IP/MPLS network splits into multiple dissemination areas. Area Border Routers (ABRs) reside at the border of the dissemination areas. Thus the controller can be part of either the backbone dissemination area or any of the stub/attached areas.

In the former option, the ABR can be considered part of the OpenFlow controller domain (as shown on Figure 47). This option is roughly similar to the single dissemination area option because the controller communicates with all nodes in the backbone using OSPF-TE, LDP or RSVP-TE for dissemination. Relying only on these protocols, it is possible to create LSPs in the core area only. Spanning LSPs covering multiple dissemination areas require additional protocols:

- One possibility is to introduce MP-BGP as described in the seamless MPLS concept [40]. This means that the controller must support the MP-BGP protocol as well as extended router and LSP redistribution mechanisms.
- Another option is to introduce some parts of the GMPLS' multi-domain extensions based on signaling extensions of RSVP-TE. Since such an LSP spans multiple dissemination areas, the source node has an accurate view of the local area only, and it has connectivity information only for the remote one. This affects the path calculation mechanisms used. It is possible to calculate the path domain-by-domain, but it will not be optimal. A better solution is to adapt the Backward Recursive Path Computation (BRPC) algorithm and use the PCEP to synchronize the calculated path fragments. To support this option the controller must implement the extended version of RSVP-TE as well as PCEP.

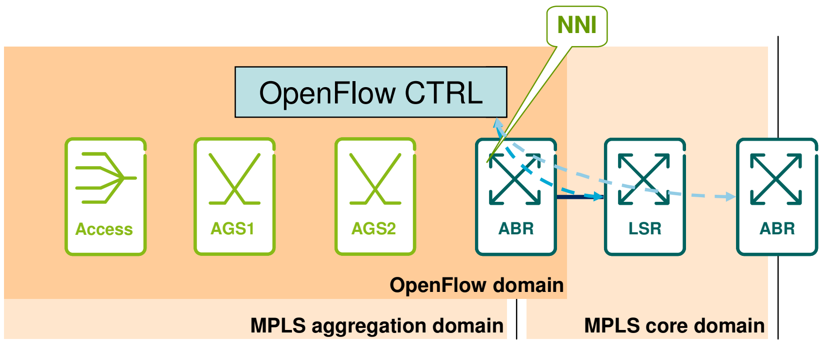


Figure 47: ABR is under OpenFlow control

In the latter option (see Figure 48) the controller is part of an attached dissemination area. Just like the other multi-area option, it implements the basic protocols to configure the LSPs within the area and applies BGP or multi-domain RSVP-TE for configuring the end-to-end path. The significant difference is that the controllers are not directly involved in the core network configuration, and they have a limited view. The pros and cons of the two multi-area options are discussed as part of the scalability evaluation in Section 4.5.

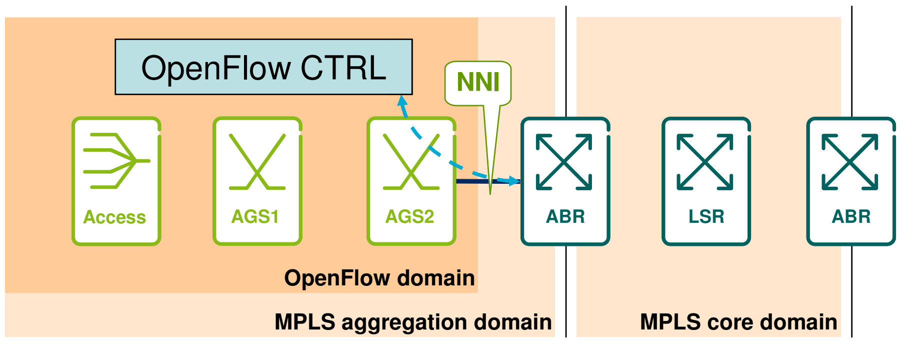


Figure 48: ABR is not under OpenFlow control

Based on the implementation cases discussed, the NNI interface running between the controller and the IP/MPLS domain shall implement the following protocols:

- A link state IGP protocol (either OSPF or ISIS) to share connectivity information.
- TE extensions of the above IGP protocols to share TE attributes and/or provide detailed information about the internal structure of the OpenFlow domain.
- To signal intra-domain MPLS connections, LDP can be used for best effort, RSVP-TE for TE-enabled connections.
- To signal inter-domain MPLS connections, MP-BGP can be used for best effort, RSVP-TE (RFC5151) with optional PCEP support for TE-enabled connections.
- To support multicast, mLDP or RSVP-TE (RFC4875) can be used.

4.4.2.2 Domain representation model

The IP/MPLS control plane design assumes that the protocol speakers are actually routers, and they could consider all other speakers as routers. As a result, that information model focuses on the links running between these routers, and very limited information is disclosed about the internals of the routers. This model was kept when TE capabilities were added: The TE attributes were tied/bound to the link descriptors (see Traffic Engineering Link, TE Link concept). One alternative is to keep the IP/MPLS model as is, and the controller is then developed with functions to support the existing models. As an alternative the IP/MPLS information model may be extended, similar to the WSON extensions for GMPLS [27]. However, it will violate the assumption of not making any updates to the core network. Therefore, this latter alternative is not discussed here.

A trivial consequence is that the OpenFlow controller must implement an appropriate mapping function between the controller’s internal model and the IP/MPLS information model because the IP/MPLS model is not sufficient to describe all aspects of the OpenFlow domain. This mapping function can be implemented in many ways.

A possible realization is the emulation of the control plane (see for instance QuagFlow [28]). The OpenFlow domain switches are replicated as emulated routers running the IP/MPLS control plane. This creates a logical view of the OpenFlow domain topology and all control plane actions are emulated. If more routers run in the emulated environment, they will synchronize their state even though they are running in the same controller.

Instead of replicating the whole OpenFlow domain with emulated IP/MPLS routers, we propose representing the whole OpenFlow domain as a single IP/MPLS router. This single virtual router is considered during the interaction with the legacy IP/MPLS control plane. Its identifiers and virtual interfaces and are advertised in OSPF, and its content determines the sent signaling messages (LDP or RSVP-TE) as well. Upon reception of any signaling messages its content will be uploaded. Besides eliminating the unnecessary domain internal state synchronizations, this approach has scalability advantages as well: The rest of the IP/MPLS does not need to take care of the internal structure of the OpenFlow domain.

4.4.2.3 Updated controller architecture

The general control plane architecture described in Section 2.4 does not cope with all the requirements dictated by the considered MPLS-based access/aggregation scenario. That hierarchical organization of controller layers does not consider peering control plane entities at the same control layer. To support such peering control plane entities, the control plane model must be extended as described below.

According to the above examples, the peering control planes may use different approaches to manage their supervised network segments. Therefore, direct exchange of the internal data models of the network segments is impossible without any agreed translation functions. Even if such translation functions were defined, implementing would not be recommended due to scalability and privacy issues. For example, one operator may not want to disclose all internal information to other operators. Furthermore, sharing the information describing databases may place unacceptable processing burdens on the interoperating control planes: One controller would process each and every change in the databases of other controllers.

Router virtualization is an obvious choice to alleviate both problems. In this case, the peering controller plane must support a common information model and associated procedure set. The virtualization models used here are roughly the same as those used in hierarchical interworking, i.e., the managed network domain can be represented as a switch, or a set of switches with a physical/emulated topology. The associated procedures sit on the top of this virtualization model and are implemented by protocols. In order to enhance flexibility, the virtualization model and the associated procedures are detached, i.e., the different sets of protocols may use the same virtualization model, and the protocols actually used may be configured. The resulting updated control architecture is depicted in Figure 49.

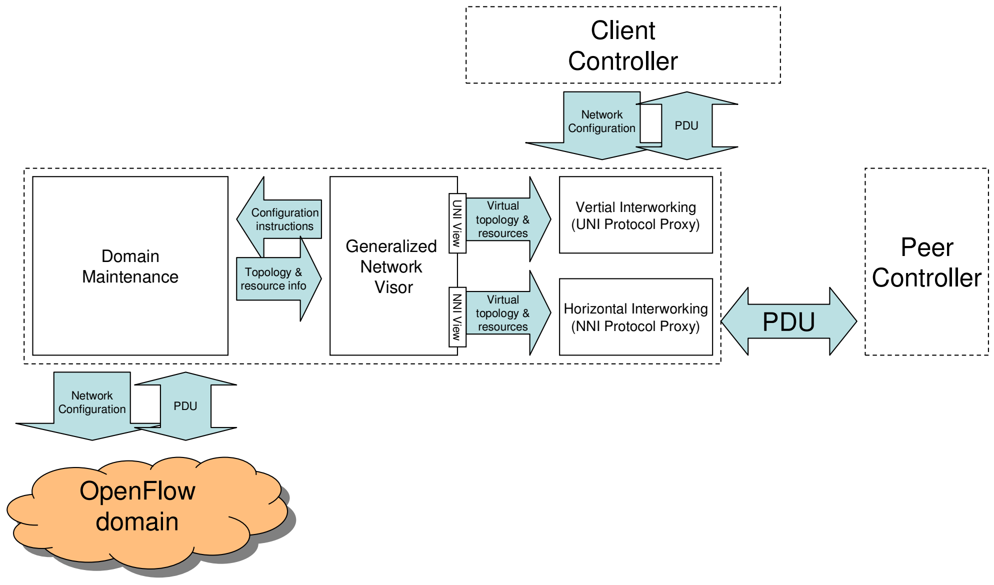

Client Controller
Network Configuration
PDU
Domain Maintenance
Configuration instructions
Topology & resource info
Generalized Network Visor
UNI View
NNI View
Virtual topology & resources
Virtual topology & resources
Vertial Interworking (UNI Protocol Proxy)
Horizontal Interworking (NNI Protocol Proxy)
PDU
Peer Controller
Network Configuration
PDU
OpenFlow domain

Figure 49: Revised controller architecture

Here the controller is comprised of three major elements:

- The Domain Maintenance module manages the OpenFlow domain, deploys flows, reacts to topology changes etc.
- A Generalized Network Visor realizes the virtualization feature by managing virtual (emulated) topologies. The namespace and resource management is also implemented here.
- Interworking modules implement the protocol and functional modules to communicate with other control plane entities. These modules operate on top of the virtualized topologies provided by the generalized visor, which controls the access of the interworking modules to the OpenFlow switches.

With these extensions a controller implementing the split transport control plane will comprise several key modules.

The Domain Maintenance module is responsible for managing the OpenFlow domain. It authenticates the connected data plane switches, and it maintains an appropriate view of the topology of the available resources of the managed network domain. It also provisions all desired aspects of flows within the OpenFlow domain: calculating paths fulfilling the traffic engineering objectives, deploying monitoring endpoints and the protection infrastructure. It also provides an interface to other modules. Through this interface the other modules can post configuration requests to the Domain Maintenance module and they can receive reports of events in the managed domain.

The Generalized Network Visor supports the various virtualization models and implements a virtual router model. The essential transformation functions between the virtual model and the managed topology are realized by the visor as well. For example, a configuration request in the virtual router may trigger a request for a flow establishment from the Domain Maintenance module. The result of a successful deployment of such a flow may result in installation of a new forwarding entry in the virtual router.

The virtual router is used by the NNI protocol proxy that steers the communication with the IP/MPLS protocol control plane. For example, it can integrate all relevant legacy protocols (OSPF-TE, LDP, BGP, etc.) that run as part of the controller. Another option is to make use of external protocol implementations. In this case the controller hosts a proxy or kernel part of the protocol stack and uses stack-specific protocols (e.g., the zebra protocol) to communicate with the protocol implementation. As a third option, external protocol stacks are used, but the virtual router model is exported and the controller acts as a switch by providing standard switch forwarding configuration interfaces (e.g., SNMP).

4.5 Scalability Characteristics of Access/Aggregation Networks

This section presents our scalability investigations regarding the access/aggregation use case. First we provide a high level introduction and then we present the results of our numerical analysis before finally showing the simulation results.

4.5.1 Introduction to the scalability study

There are many aspects of scalability, e.g., the scalability of an architecture, of a protocol, of a given scenario. We are focusing on the generic Split Architecture concept and the main use-case of the project.

4.5.1.1 Deployment – network topology

Our deployment scenario is based on D2.1 and serves as a basis for the scalability study. Figure 50 shows the common view of the project partners in the schematic view of the access/aggregation network area.

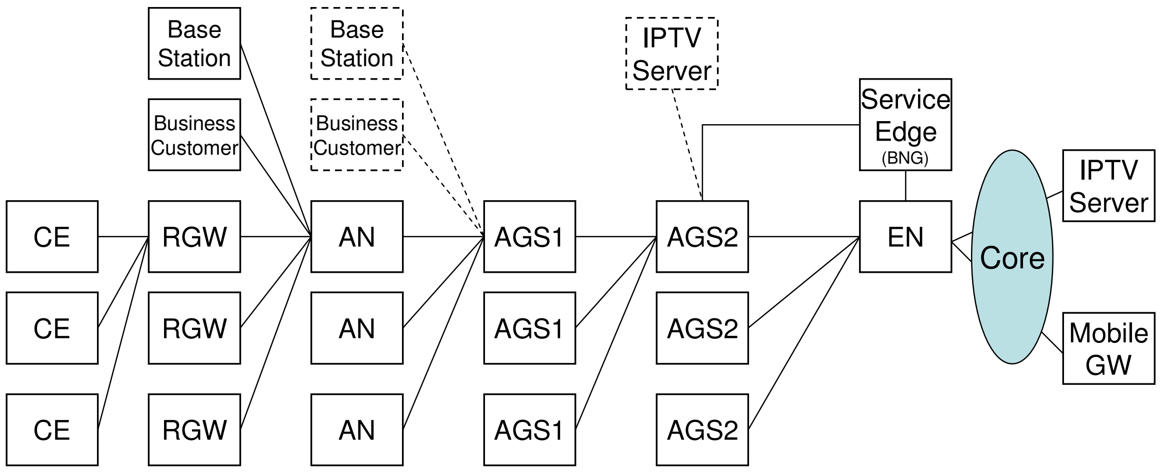


Figure 50: Deployment scenario for scalability studies

The OpenFlow domain consists of Access Nodes (AN), first-level aggregation switches (AGS1), second-level aggregation switches (AGS2) and edge nodes (EN). All other entities, both on the client side (Customer Equipment, Residential Gateway, Business Customer [BC], mobile Base Station [BS]) and the server side (Service Edge, IPTV server, Mobile GW) are not part of the OpenFlow domain. Note that multiple access/aggregation domains can be connected to the core network and they can be OpenFlow-based, but they have no direct (OpenFlow) interaction.

Based on D2.1 we have three scenarios that cover current and future deployment numbers.

- In the “Today” scenario there is 1 PoP location (corresponding to our EN) for 500,000 households (which are represented by the RGWs). This covers about 1 million inhabitants.
- In the “Future” scenario 1 PoP location will cover around 2,000,000 customers.
- In the “Long-term” scenario 1 PoP location will serve 4,000,000 customers.

In all scenarios the number of devices will relate to each other as: customers devices (CE) >> customer edge (RGW) >> access nodes (AN) >> edge nodes (EN).

Additionally, the sum of access nodes (AN) relates to the sum of aggregation nodes (AGS1+AGS2) and edge node (EN) as 10:1.

Based on these assumptions, network topologies can be drawn for the different scenarios.

4.5.1.2 Deployment - services

Regarding the services provided by using this network, we have made the following assumptions based on D2.1.

There are residential services, namely the simple service (e.g., PPPoE) for Internet access and IPTV. The simple service provides bidirectional connectivity between the RGW and the service edge. The IPTV service provides bidirectional connectivity between the RGW and the IPTV server, where the direction from the IPTV server dominates, and typically the same data is simultaneously sent to multiple RGWs.

There are business services, namely the simple service (as for residential customers), the Point-to-Point (PtP) and the Multipoint to Multipoint (MPtMP). The PtP service connects two business customers, while the MPtMP connects multiple business customers. All of these services are bidirectional.

The mobile backhaul service connects a base station to a mobile GW in a bidirectional way.

4.5.1.3 Scalability concerns

There are theoretical constraints due to protocol limitations, e.g., the maximum number of fields or objects, maximum length or value of fields or objects, maximum size of packets, etc. The OpenFlow protocol is relevant in our case. Other protocols that may be affected are the distributed control plane protocols, e.g., MPLS (OSPF, LDP, RSVP). Our focus is not on the analysis of a given protocol, but rather on a more generic approach to the split control plane. Protocol-specific issues are beyond our scope.

Computational resource limits may be a bottleneck at the central controller (in such a case these calculations may be made in a decentralized way, e.g., single central logical view vs. distributed physical view). The complexity of the algorithms used can be checked here, e.g., NP hardness. However, the time budgeted for running an algorithm is important – from this point of view the P complexity class could be still too difficult in some practical cases.

The control network's capacity limits the overall control traffic, thus introducing upper limitations on the frequency and size of node configuration commands. This can have an effect on the possible network size (e.g., the number of nodes/ports/links) or on the reaction time of the overall network.

The control network introduces non-negligible delays due to the geographical distance between switches and the controller. This results in a lower limit on reaction time even if the controller had infinite computational power.

4.5.2 Numerical model

This section describes our numerical model and the results of the of the study based on this model.

4.5.2.1 High-level description of the model

The model is based on Section 4.5.1 and it includes the network nodes, services and scenarios identified there, with the following assumptions. Clients (RGW, BC, BS) are connected only to the AN, not to higher level aggregation nodes. SE is not directly interconnected to an AGS2. The MPtMP business service is not modeled. The UNI interface is on the left side of the AN, while the NNI is on the right side of the EN, see Figure 52.

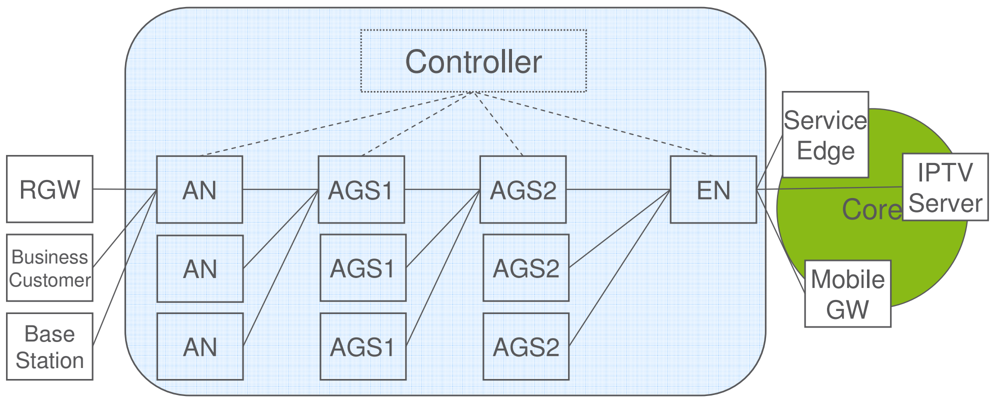


Figure 51: Simplified domain view

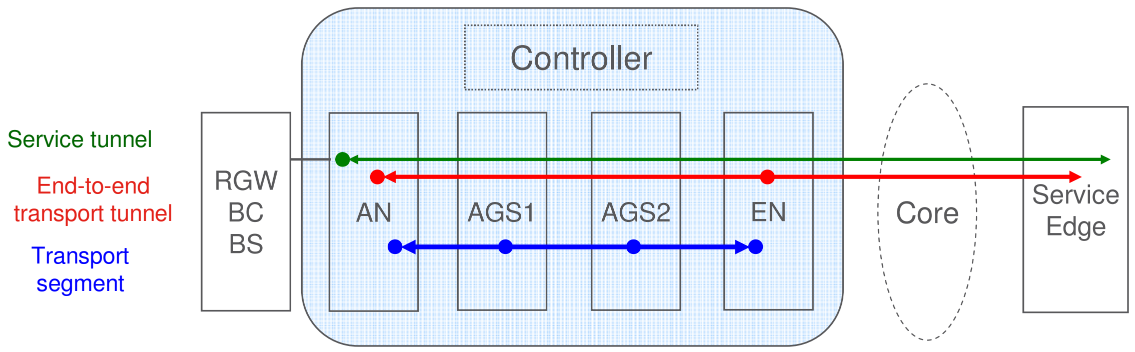


Figure 52: Tunnels considered|

To transport the above services appropriately, the transport connection architecture is assumed as depicted in Figure 52. The blue connections are OpenFlow domain internal transport segment tunnels. There are multiple options for organizing these tunnels. There can be a segment tunnel connecting each AN to the EN as shown in the figure.

End-to-end tunnels (shown as a red line) are defined between nodes, e.g., the AN and the SE. Such tunnels overlap OpenFlow and non-OpenFlow domains.

Service tunnels (shown as a green line) within the E2E tunnels identify the service, e.g., the user in the case of an AN where multiple RGWs are connected. The exception is IPTV, which is treated as a special E2E tunnel in the model.

This triple-layer connectivity set provides compatibility with the IP/MPLS core, flexibility within the OpenFlow domain, and it also supports scalability (by aggregating connections).

4.5.2.2 Static requirements from the model

The figure below (Figure 53) shows the amount of equipment units to be managed by the controller according to the main scenarios based on our model. Two alternatives are offered for each scenario, both fulfilling the requirements. Note the we will later concentrate on one alternative only because the results of the alternative topologies have the same order of magnitude results in each case.

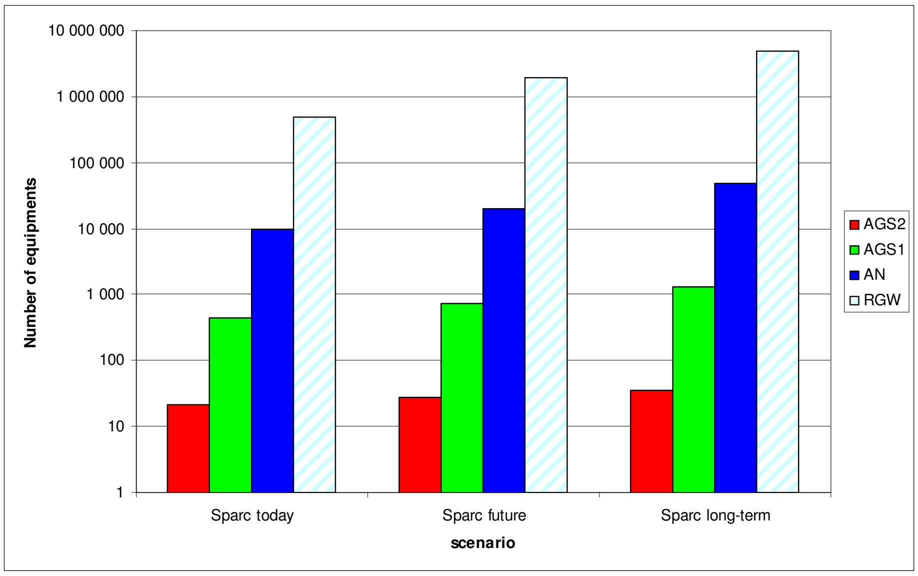


Figure 53: Number of equipment units

Figure 53 shows that the number of OpenFlow switches in an OpenFlow domain is around 10,000 / 20,000 / 50,000 for the specific scenarios.

Figure 54 shows the number of tunnels terminating or crossing a given type of node.

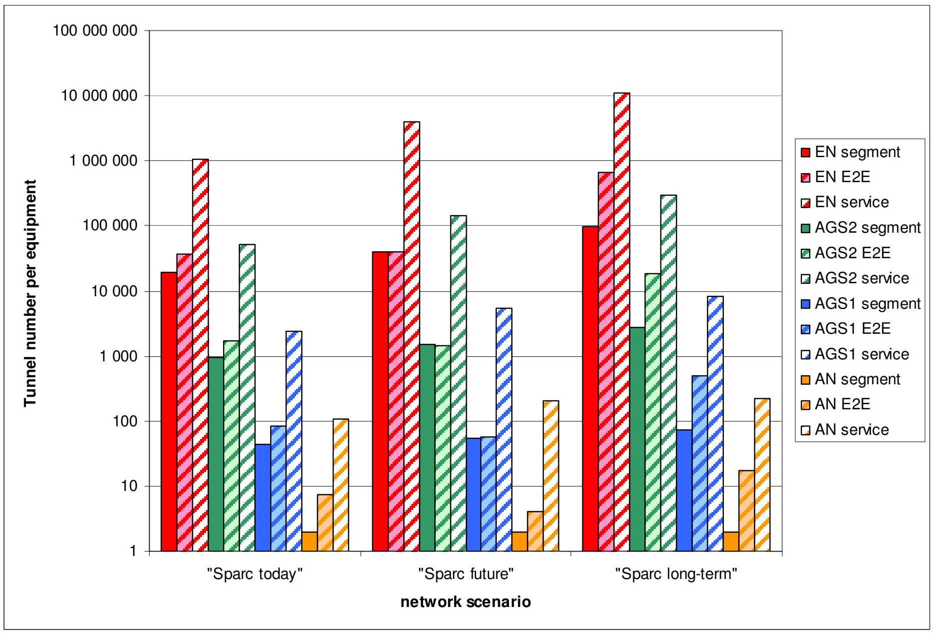


Figure 54: Number of tunnels

Note that a flow entry isn't needed for each type of tunnel in each type of node – for example, an aggregation switch is unaware of the service tunnels passing by because they are encapsulated in E2E tunnels which are encapsulated in transport segment tunnels. This figure illustrates why encapsulation is needed here and why automatic advertising of all interfaces (e.g., via LDP) is not recommended for use here.

Figure 55 below shows the number of flow entries to be supported by the network entities in our access/aggregation use case for the different scenarios.

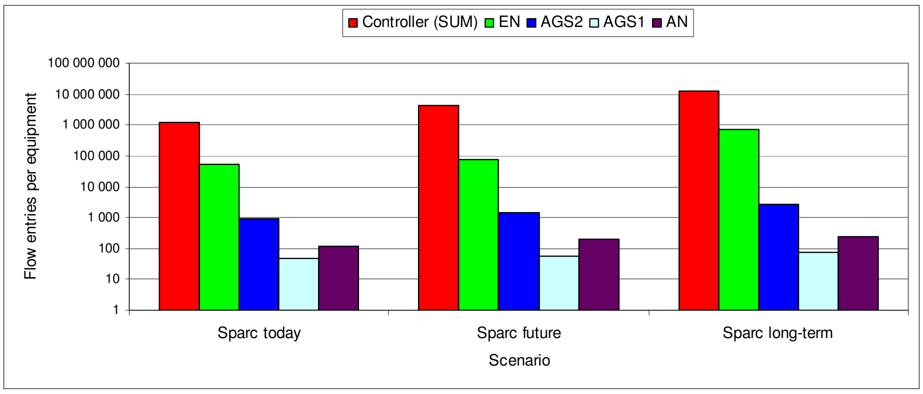


Figure 55: Number of flow entries

We can see that for the lower part of the aggregation about one hundred, while for the upper part of the aggregation thousands of flow entries have to be supported by the switches. The controller has to manage from one to ten million flow entries overall in the whole domain.

Based on the static analysis the following KPIs have been calculated for the scenarios (today/future/long-term) for a given OpenFlow access/aggregation domain:

- Number of OF switches in an OF domain: 10,000 / 20,000 / 50,000
- Number of UNI interfaces: 500,000 / 2,000,000 / 5,000,000
- Number of NNI interfaces: 1
- Number of flow entries in a switch, depending on aggregation level: hundreds – thousands – tens of thousands
- Number of actions of a flow entry: 1 – 2

4.5.2.3 Dynamic results from the model

To address the dynamic behavior of the OpenFlow domain, we considered the effects of two type of events.

The first one was a link going down an AN-AGS1, an AGS1-AGS2 or an AGS2-EN link, to be handled by the controller. We assume redundancy in the topology, so all traffic can be rerouted to a backup path. Regarding the reconfiguration of the flow entries, we calculated two values for each link-down:

- The best case, where there is a direct alternative link between the same two nodes, resulting in limiting the modifications only to those two nodes.
- The worst case, where the alternative link is connected to a different (but same aggregation level) node, with the most disjoint path to the EN compared to the original failed one. In this case the flow reconfigurations are not limited to the nodes directly affected, and higher aggregation level nodes will also be reconfigured.

Figure 56 shows how a link-down affects the various tunnels according to the model.

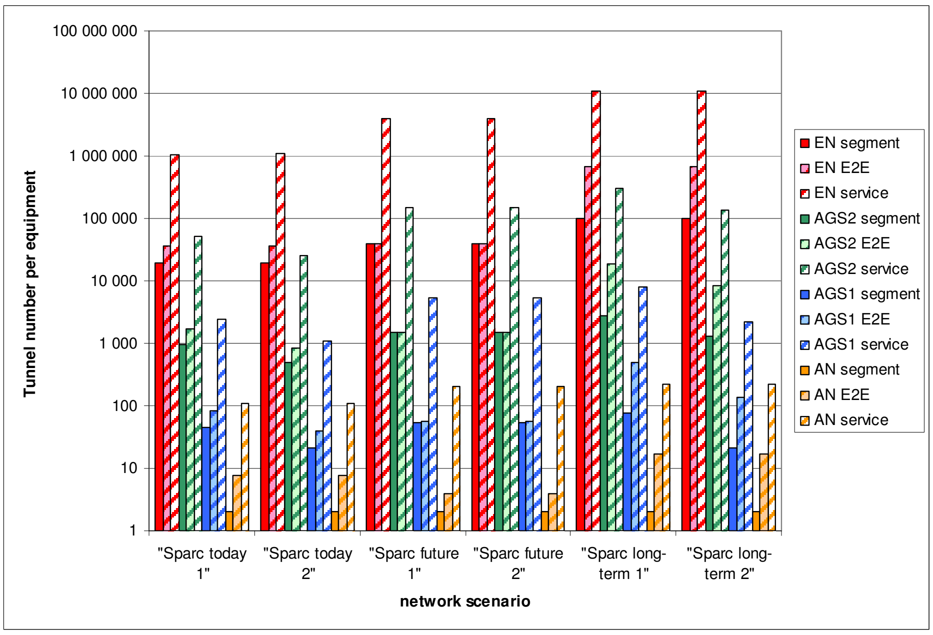


Figure 56: The effect of a link down: tunnels

From the best case we can derive requirements for the switches, while from the worst case we can derive requirements for the controller.

Figure 57 shows the required flow modification commands for restoring the tunnels via the controller.

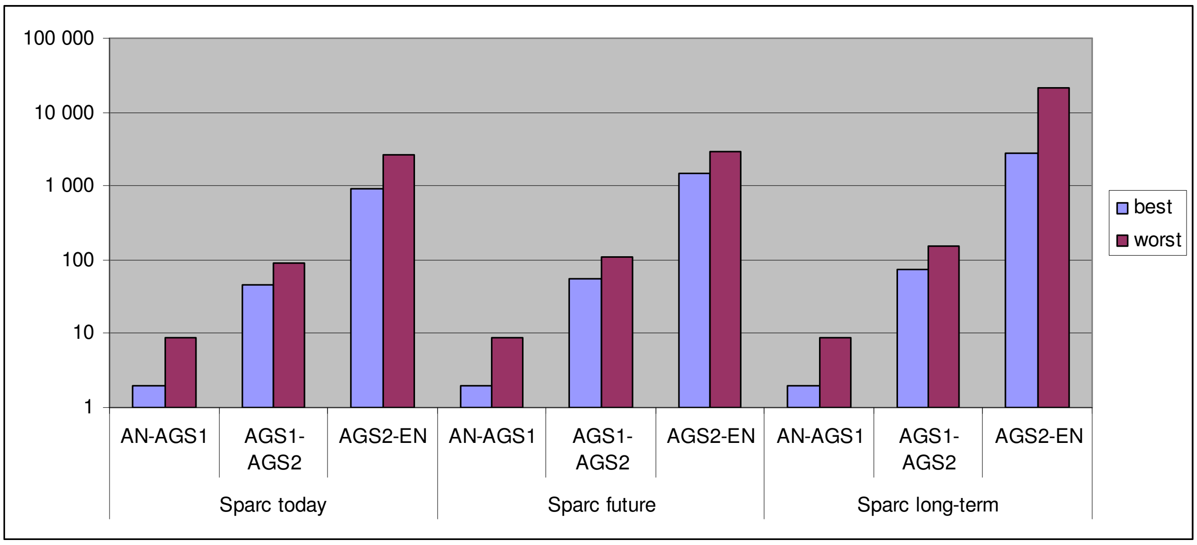
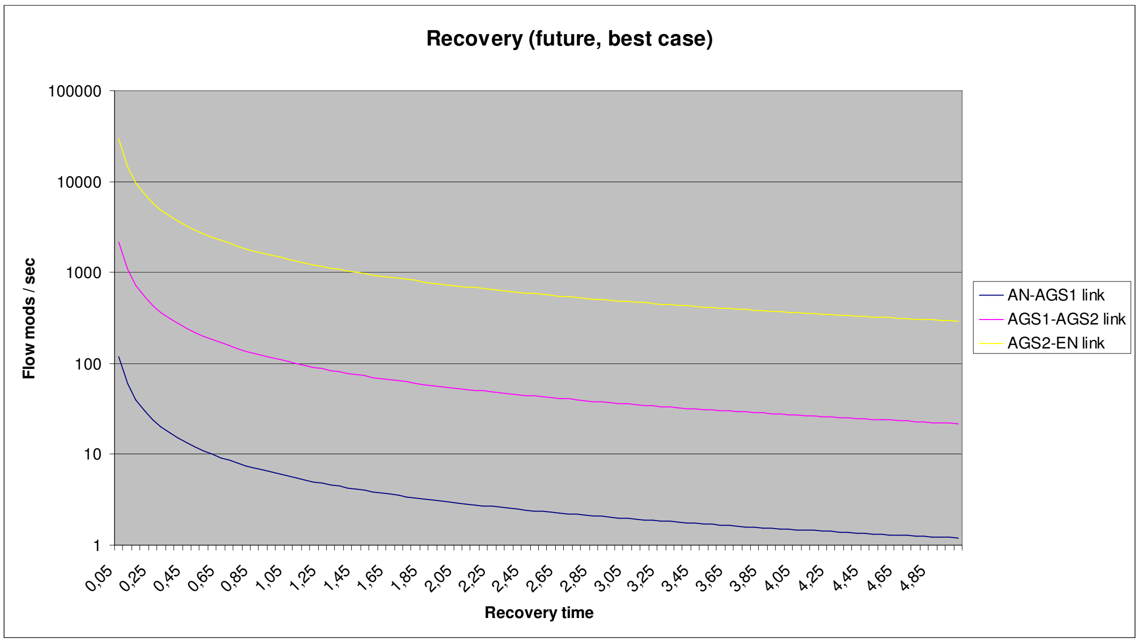


Figure 57: The effect of a link-down: flow mods

Flow_mods to be handled by a switch in the case of a non-AGS2-EN link failure are in the magnitude range of tens and a few hundreds. For the controller, the magnitude range is in the hundreds of flow entries.

In the case of an AGS2-EN link failure, the switch has to be able to handle a few thousand of flow_mods, while the controller magnitude range is a thousand or a few ten thousands.

Taking into account the time available for sending these changes results in Figure 58. These diagrams can be used to derive requirements, e.g., if 1 sec. can be used for the controller to send the configuration messages (see horizontal axis), then this results in a given flow mod/sec controller requirement (vertical axis).



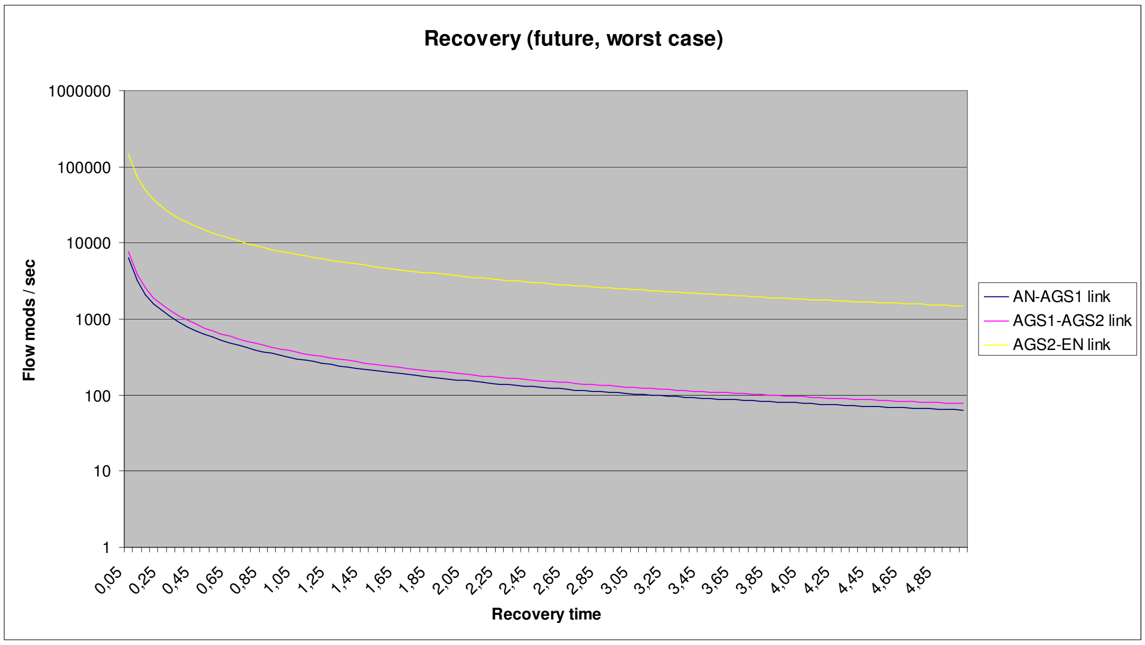


Figure 58: Recovery times

Note that the time refers only to sending the flow_mod messages – other aspects are not shown here.

The second type of event we considered is a burst of IPTV channel changes. This could happen if a significant percent of the users change channels, for example in the case of the beginning of a sport event or the evening news. The figure below shows the number of flow_mod messages the controller has to send. The best case refers to the situation where the video stream is already at the AN, whereas in the worst case the path for the stream has to be configured from the core for each request (which is slightly above the realistic scenario, but definitely an upper limit). The result is shown in Figure 59.

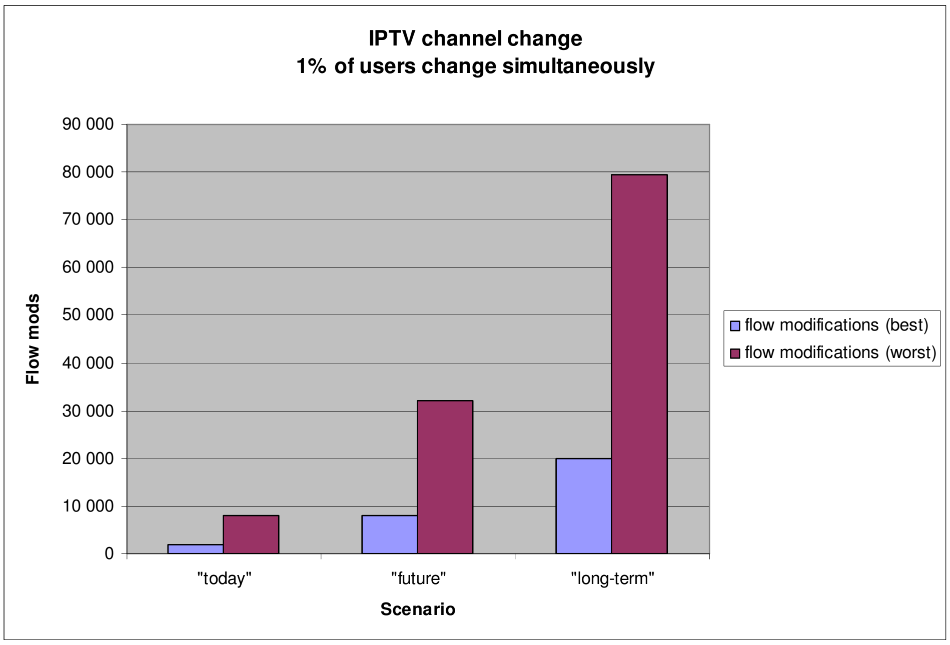


Figure 59: IPTV channel change

While the best case is a requirement for the AN and a lower limit for the controller, the worst case is a pessimistic upper limit requirement for the controller. Note: 1 percent of IPTV users change channels simultaneously in this example, but since it is scaled linearly, it is easy make calculations for other numbers.

Taking into account the time available for sending these changes results in Figure 60. Note that the time refers only to sending the flow_mod messages – other aspects are not shown here.

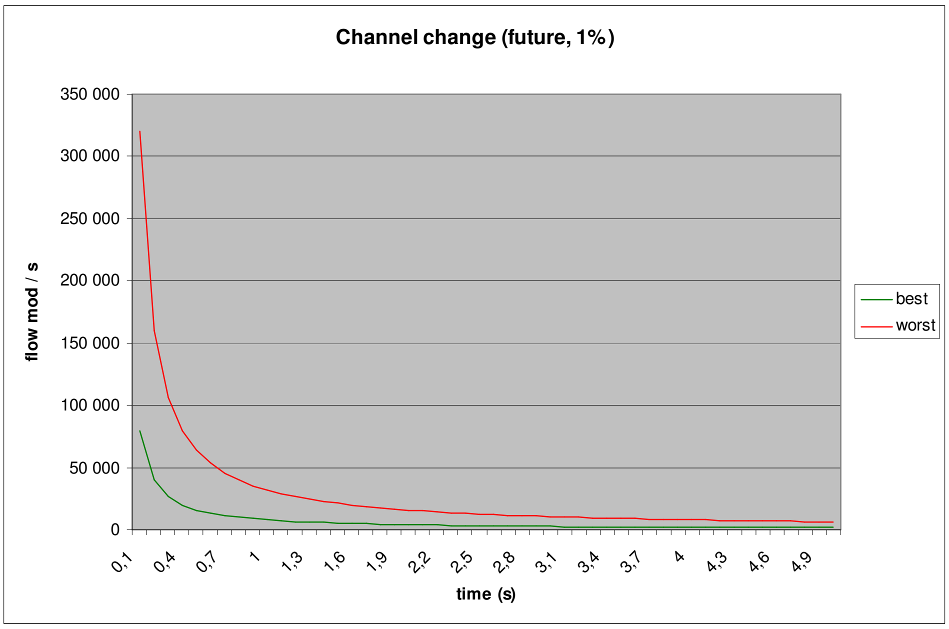


Figure 60: IPTV channel change time

If we assume 1 second response time for the channel change request and 1 percent of the users change, then the flow mod rate per second to be issued by the controller is in the best case 2,000 / 8,.000 / 20,000, while in the worst case 8,000 / 32,000 / 80,000.

4.5.2.4 Comparison – existing switch and controller performance

We found the following related numbers in the DevoFlow [10] paper, which discusses a different use case and setup, however it provides numbers for expectable switch and controller performance, regardless of the actual setup:

> "We found that the switch completes roughly 275 flow setups per second. This number is in line with what others have reported [11]."

This performance is far below that needed to provide carrier-grade controller-based resiliency during a link-down event, however, it may be suitable for best effort type of service. Regarding the IPTV channel change event, this switch performance seems to suit the lower parts of the access/aggregation network.

> "[12] report that one NOX controller can handle 'at least 30K new flow installs per second while maintaining a sub-10 ms flow install time ... The controller's CPU is the bottleneck.' "
>
> "Maestro [13] is a multi-threaded controller that can install about twice as many flows per second as NOX, without additional latency."

This reported controller performance seems to be in the order of magnitude suitable for handling link-down effects in the network in a carrier-grade sense. For the IPTV channel change, it depends on the specific requirements.

4.5.2.5 Updates to the initial topology and tunnel structure

The high flow_mod values in the case of link failure restoration close to the EN are the result of the many tunnels between the ANs and the EN. To decrease this high number of flow_mod messages, those tunnels could be aggregated, thus having only a single transport segment tunnel between neighboring nodes. In the case of restoration, an alternative segment tunnel can be built between the same two nodes over other multiple nodes. This increases the static number of flow entries per forwarding device and the traffic volume. However, it drastically decreases the message number in the restoration case. Figure 63 shows the comparison of a tree (snowflake) topology (Figure 51), a tree + ring combination topology (Figure 61), and the same topology with the single hop transport segments (Figure 62). Order of magnitude differences are evident.

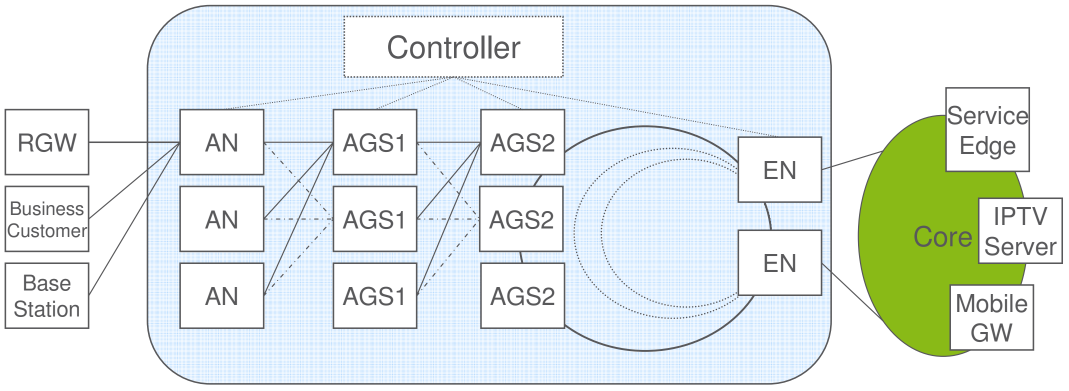


Figure 61: Ring topology

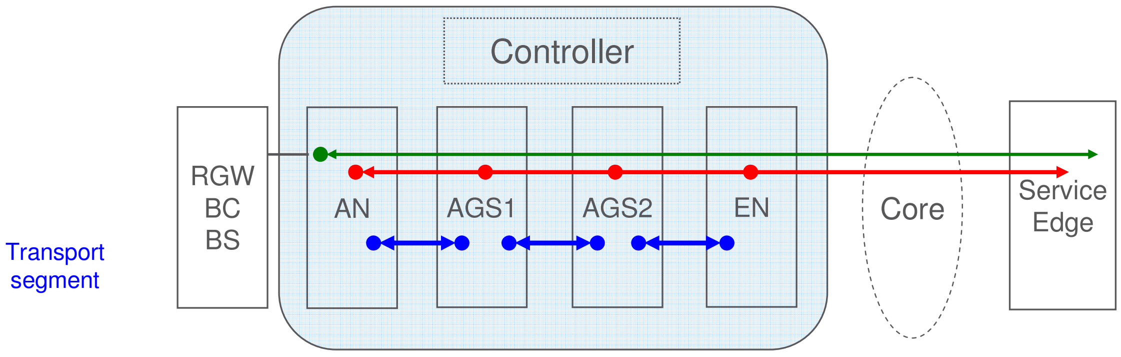


Figure 62: Alternative OpenFlow domain internal tunnel structure

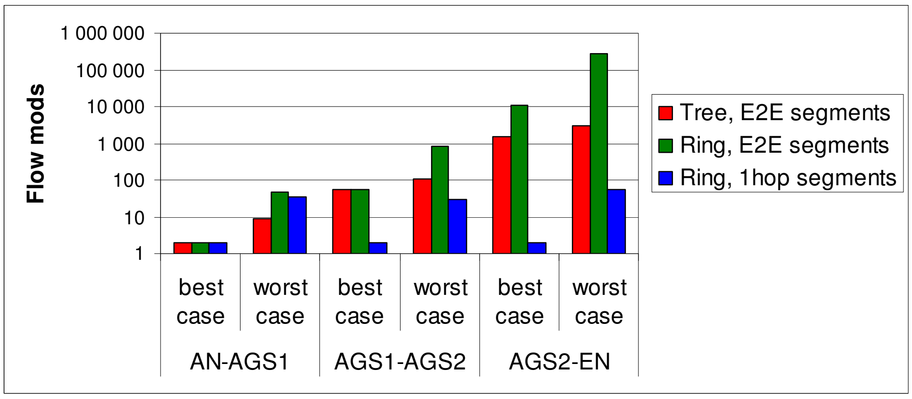


Figure 63: Topology and domain internal tunnel structure effects

However, reducing the required flow_mod messages has specific disadvantages as well, such as more static flow entries or increased traffic volume in the case of restoration. A detailed analysis is beyond the scope of this deliverable.

4.5.2.6 Conclusions from the scalability study

Looking at the static behavior, requirements from the model seem to be in the order of magnitude or below the capabilities of existing controllers and switches.

Regarding the dynamic behavior, there are scalability concerns, especially when strict time constraints are given. We showed that by changing the connection structures scalability can be significantly increased.

5 Summary and Conclusions

In this deliverable we presented the first iteration of an updated Split Architecture for large-scale wide area networks, such as carrier-grade operator networks. Based on our conclusions from D3.1, we focused on OpenFlow as an enabling technology for the Split Architecture. OpenFlow recently gained strong momentum in both industry and academia, and it appears to be an interesting evolving technology, reasonably well-suited for realizing split-architecture operations in networks. However, earlier SPARC Deliverables (D2.1 and D3.1) made it clear that current OpenFlow implementations do not fulfill carrier requirements. In this deliverable, we therefore discussed a suitable framework for controlling carrier-grade operator networks and investigating missing features in OpenFlow as identified during the SPARC project in the context of access/aggregation network scenarios.

In this final section, we will summarize and conclude our discussion of the Split Architecture for large-scale wide area networks. We will recap our considerations regarding the SPARC control framework, the proposed extensions to OpenFlow and the different deployment scenarios for OpenFlow-based Split Architectures in typical access/aggregation network settings. A final, refined Split Architecture proposal will be presented at the end of the SPARC project in Deliverable D3.3, which will be an updated version of the current Deliverable D3.2, including further insights derived from the implementation and validation of the SPARC prototype in WP4 and WP5.

5.1 SPARC Control Framework

We propose a control framework for modern operator networks offering flexible and extensible deployment options. This is especially important in the face of network virtualization, which allows several network instances to coexist within a common physical infrastructure. Our goal for the split-carrier network architecture is therefore the ability to deploy various control planes in parallel on the same physical infrastructure and to assign flows based on operator and user policies to one of the available control planes. Since we cannot anticipate the potentially wide variety of control planes network operators may intend to deploy in the future, we propose adopting three common principles providing flexibility and scalability to a carrier- grade Split Architecture: Flowspace Management, Advanced Processing, and Recursive Stacking.

The proposed *Flowspace Management* functionality extends the OpenFlow API with a (de-)multiplexing entity at the north bound interface on the datapath elements, allowing assignment of slices of the complete flowspace offered by OpenFlow to separate control entities. This allows multiple control entities or network applications to control single datapath elements by specifying which the parts of the flowspace they want to take control of. The functionality is quite similar to FlowVisor, but instead of controlling flowspace slicing via a dedicated management interface, we propose integrating all flowspace management messages directly into the OpenFlow API and relocating the visor functionality onto the datapath elements.

Today, OpenFlow 1.x focuses on the interface between forwarding and the control plane. *Advanced Processing* is supposed to be carried out by applications on top of the controller, while the set of available processing actions on datapath elements is restricted to lightweight, i.e., stateless actions, e.g., simple header field modifications. In an operator network, however, there are requirements for more advanced processing functions in the datapath elements as well. Such requirements include, e.g., OAM functions as well as legacy tunneling and encapsulation functions such as PWE, PPP and GRE, among others. We propose a modularization of processing into atomic processing modules, whose mutual relationships are mapped to a layered control plane structure.

In order for the control plane to offer the flexibility and extensibility required for modern operator networks, we propose a control plane framework realized by *Recursive Stacking* of control entities. In the SPARC controller framework each control entity performs a different set of functions on information provided by control entities below the present layer in the controller hierarchy, and also provides processed information to control entities above. Each control entity can act as a proxy and thus present a virtualized topology to higher layer entities. The interface between controller entities is also defined by the OpenFlow protocol. Such a hierarchical architecture helps break up the complex task of controlling huge and heterogeneous operator networks into smaller manageable pieces. Furthermore, it enables the operator to fine-tune the level of abstraction provided to higher layers, i.e., the granularity of the implementation details presented. In other words, the goal of recursive stacking of entities in the control layer is to take advantage of the modularity and abstraction offered by a layered model, while doing away with the strict and rigid layering structure as offered by traditional network layering such as the ISO/OSI model.

To sum up, the proposed hierarchical carrier-grade Split Architecture enables operators to deploy several control planes with minimal interference and assign flows dynamically based on given policies. Furthermore, it provides a network operator with tight control of the level of detail when data plane details are exposed to higher control planes or third parties.

5.2 SPARC OpenFlow Extensions

In this deliverable we began by identifying four stages of how OpenFlow can be integrated into carrier grade-networks:

1. *Basic emulation of transport service*: In this stage an OpenFlow data and control plane emulates and replaces legacy transport technologies (e.g., Ethernet, MPLS).
2. *Enhanced emulation of transport services:* OpenFlow is again used to provide transport services. However, a number of features and functions are added to both the data and control plane in order to comply with carrier-grade requirements such as OAM and resiliency aspects.
3. *Service node virtualization*: In this stage, besides transport services, OpenFlow also takes control of (distributed) service node functionalities, including service creation, authentication and authorization.
4. *All-OpenFlow-network*: In this integration stage OpenFlow also controls other network domains, e.g., customer premises equipment such as RGWs and the operator's core domain.

Considering that current OpenFlow 1.x is sufficient to provide integration step 1, the main focus of this deliverable is on integration step 2, i.e., studying possible extensions to OpenFlow in order to fulfill carrier-grade requirements. In some cases, we are also starting to touch upon integration step 3, e.g., by discussing service creation in Section 3.4.

Section	**Study topic**	**Level of the proposed improvements**	**Specification details in D4.2**
3.1	*Openness and Extensibility*	Detailed	Yes
3.2	*Virtualization and Isolation*	Detailed	Yes
3.3.3	*OAM: technology-specific MPLS OAM*	Detailed (BFD)	Yes
3.3.4	*OAM: technology-agnostic Flow OAM*	Conceptual	No
3.4	*Resiliency*	Conceptual	No
3.5	*Topology Discovery*	Conceptual	No
3.4	*Service Creation*	Detailed (PWE, PPP)	Yes
3.7	*Energy-Efficient Networking*	Conceptual	Yes
3.8	*QoS*	Conceptual	No
3.9	*Multilayer Aspects*	Conceptual	No

Table 4: Summary of OpenFlow extensions studied in Section 3

The possible extensions studied have been derived from the requirements and open questions identified in earlier SPARC Deliverables D2.1 and D3.1. In

Table 4 we summarize the topics discussed in Section 3 of this document. In the table we also indicate the level of detail of the resulting study. In cases where we provide firmly defined extensions to OpenFlow, the detailed specification can be found in SPARC Deliverable D4.2 "Description of OpenFlow protocol suite extensions," as also pointed out in the table.

Openness and Extensibility: As pointed out earlier, extended and more advanced processing functionalities at the data plane are desirable in many situations, e.g., to cope with legacy encapsulation protocols as required in the access/aggregation use case (PPP, PWE, etc.). We discussed the concept of virtual ports as a means of providing advanced processing functionalities in datapath elements, as currently under discussion in the "Open Networking Foundation" (ONF). Instead of virtual ports, which overload the port semantics, we propose similar processing entities which allow keeping state on datapath elements. A processing entity in this sense is a generalized OpenFlow action that encapsulates some processing logic. Processing entities are then accessed by an action type "process" from within the forwarding pipeline with the help of a unique identifier, the Processing Entity ID.

Virtualization: A crucial feature of future carrier-grade networks is network virtualization, enabling multi-service and multi-operator scenarios on a single set of physical network infrastructures. In this deliverable we propose improvements to the isolation and automation aspects of an OpenFlow-based network virtualization system. Regarding isolation, we identified the current lack of QoS primitives in OpenFlow as a major barrier. We also identified extensions to enable automation virtual network setup and management.

OAM: One important requirement for carrier-grade networks is the availability of proper OAM solutions. Regarding OAM integration in OpenFlow, we identified two contradictory aspects: On the one hand, integrating existing OAM tools (e.g., Ethernet or MPLS OAM) provides desired compatibility with legacy OAM toolsets and offers well-known standard functionalities. On the other hand, integrating existing OAM tools in an OpenFlow environment means eventually integrating several technology-specific toolsets, substantially increasing the complexity of datapath elements. In this deliverable we discussed both aspects, a technology-dependent MPLS OAM solution and a technology-agnostic generic flow OAM solution. A more advanced solution, combining the advantages of both approaches, requires further study and is still subject of future work.

As an example of a *technology-specific OAM*, we propose the integration of MPLS BFD in OpenFlow, which has also been implemented as part of the SPARC demonstrator. In this implementation we utilize extended processing functionalities in the datapath elements by means of virtual ports, thus creating a scalable carrier-grade OAM tool.

We also presented an initial architectural concept for a *technology-agnostic, generic flow OAM* solution for OpenFlow. We propose an independent OAM module, which may reuse existing OAM toolsets, e.g., IEEE 802.1ag-based Ethernet OAM. The OpenFlow classifier on the datapath elements is then modified in order to divert incoming OAM packets to the OAM module based on an OAM indication, e.g., a specific EtherType value. Since the OAM module is now decoupled from data traffic, we need to enforce fate sharing of OAM and data traffic. One proposed method of ensuring fate sharing is based on so-called virtual data packets. These virtual packets are created from the information carried within actual OAM packets and only exist locally within a datapath element in order to test the OpenFlow forwarding engine.

Resiliency: Resiliency and reliability are important requirements for carrier-grade networks. In this deliverable we first studied mechanisms to ensure data plane resiliency in an OpenFlow scenario. Data plane resiliency can be realized in terms of rerouting, restoration and protection. We also discussed control plane resiliency, including scenarios of both out-of-band and in-band control networks.

Topology Discovery: We discussed issues with current topology discovery modules and propose a modification based on a spanning-tree approach. As a result, the bandwidth usage on the controller links is decreased and at the same time loop prevention is provided in the OpenFlow network. We also propose a mechanism for in-band control channel establishment.

Service Creation: In our context, we define service creation as the configuration process of network functions at service creation points (SCP) within (or at the boarder of) the access/aggregation network in order to provide services to various types of customers. Besides connectivity, the configured functions include, among others, authentication, authorization and accounting (AAA) aspects. We discussed integration of residential customer services in the form of PPP and business customer services in the form of pseudo-wires (PWE) in an OpenFlow-controlled operator network. Possible deployment scenarios of the discussed solutions are outlined in Section 4.3.

Energy-Efficient Networking: Centralized control software like OpenFlow offers additional option for reducing network energy consumption. We discussed possible energy saving approaches (e.g., network topology optimization, burst mode operation, adaptive link rate) and the requirements they place on OpenFlow.

QoS: Support for QoS mechanisms is generally considered a key requirement for carrier-grade networks. Furthermore, QoS plays an even more important role in a virtualized, multi-provider, multi-service operator network enabled by Split Architecture as discussed in this deliverable. We discussed how traditional QoS tools such as packet classification, metering, coloring, policing, shaping and scheduling can be realized in an OpenFlow environment.

Multilayer Aspects: We discussed the extension of OpenFlow toward control of non-Ethernet-based layer 1/2 technologies by using the example of circuit switched optical layers, i.e., packet-opto integration. We first discussed the additional requirements placed on a control framework by optical network equipment. We then outlined three possible phases for realizing packet-opto integration in an OpenFlow environment.

5.3 Implementation Scenarios

In Section 4 we presented implementation scenarios of the proposed carrier-grade Split Architecture in an operator network, specifically in access/aggregation networks. Besides OpenFlow-controlled transport connectivity, we proposed three evolutionary approaches for OpenFlow integration of service creation functions in access/aggregation networks, depending on which elements and functionalities should be controlled by OpenFlow. These approaches include centralized OpenFlow control of a single service creation point at the IP edge only (e.g., BRAS), a decentralized model expanding OpenFlow control to aggregation devices (e.g., DSLAM), and finally a complete OpenFlow controlled approach including all devices in the access/aggregation domain.

Regarding a decentralized OpenFlow control model, we further discussed general implementation options for service creation. As an example of residential customer services we outlined the implementation of a “SPARC BRAS” by splitting forwarding and processing functions and following the Split Architecture principle. For realization of SPARC BRAS, the integration of PPP and PPPoE functionality into the OpenFlow data and control plane is required and can be realized by means of advanced processing entities as proposed in Sections 2.3 and 3.1. A detailed discussion of PPPoE integration in a DSLAM is included in Annex A of this document.

Earlier, we defined four stages showing how OpenFlow can be integrated into carrier-grade networks. OpenFlow typically does not control all parts of the network structure, except in integration stage four. As an example, access/aggregation networks might be realized in a split-architecture design, whereas the core network segment to which these networks connect still has legacy IP/MPLS as the predominate technology. This leads to an additional requirement: The split-architecture domain must cooperate with legacy control planes in a suitable peering or horizontal interworking model. In Section 4.4 we discussed different options for connecting domains controlled by the SPARC control framework to legacy IP/MPLS control planes. As a result, we will propose an updated controller architecture, also taking horizontal internetworking aspects into account.

Finally, we examined the feasibility of a Split Architecture with a numerical scalability study based on an idealized deployment model of an access/aggregation network. The results show that there are no stability concerns for static scenarios. The resulting requirements from the numerical model are in the order of magnitude or even below the capabilities of existing control and datapath devices. However, dynamic behavior (e.g., reconfiguration due to link failures) might raise scalability concerns, especially when strict time constraints are given. However, we could show that changing the connection structures by more careful network planning can increase scalability significantly even in the face of dynamic behavior.

6 Annex A: SPARC BRAS Implemented in the DSLAM

In the following section we detail the SPARC BRAS option 2 as introduced in Section 4.3.1.2. The split BRAS is implemented in the DSLAM with the virtual port concept and processing in software (represented by the virtual port). It should be noted that this model represents a relatively concrete implementation concept that is not necessarily desired for standardization purposes as other implementations might exist as well.

First a brief look at the architecture and the exchanged messages as well as data and control packet flows as presented in Figure 64.

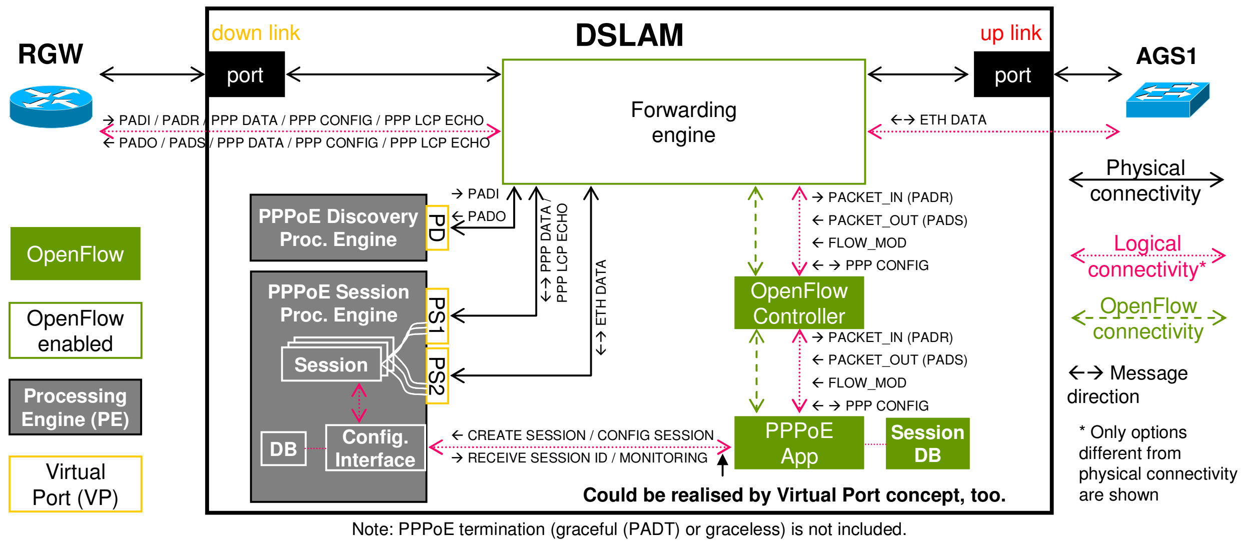

RGW
down link
DSLAM
up link
AGS1
port
port
Forwarding engine
→ PADI / PADR / PPP DATA / PPP CONFIG / PPP LCP ECHO
← PADO / PADS / PPP DATA / PPP CONFIG / PPP LCP ECHO
←→ ETH DATA
Physical connectivity
Logical connectivity*
OpenFlow connectivity
←→ Message direction
* Only options different from physical connectivity are shown
OpenFlow
OpenFlow enabled
Processing Engine (PE)
Virtual Port (VP)
PPPoE Discovery Proc. Engine
PD
→ PADI
← PADO
PPPoE Session Proc. Engine
Session
PS1
PS2
←→ PPP DATA
PPP LCP ECHO
←→ ETH DATA
DB
Config. Interface
→ PACKET_IN (PADR)
← PACKET_OUT (PADS)
← FLOW_MOD
← → PPP CONFIG
OpenFlow Controller
PPPoE App
Session DB
← CREATE SESSION / CONFIG SESSION
→ RECEIVE SESSION ID / MONITORING
Could be realised by Virtual Port concept, too.
Note: PPPoE termination (graceful (PADT) or graceless) is not included.

Figure 64: Schematic diagram of PPPoE integration in a DSLAM with components and message flow in between

The customer is connected via the Residential Gateway through a DSL-port at the DSLAM. The DSLAM is connected with an Ethernet-based port to AGS1. Internally in the DSLAM the ports are connected to the forwarding engine, potentially via a separate line card, which is not included in the figure. The forwarding engine is OpenFlow enabled. There are some additional software components providing the BRAS control, plus forwarding and processing functions. Two options are possible for these software components. Either these software components (the OpenFlow controller) are located within the DSLAM (as shown in the figure) or they are located “somewhere” in network. In the latter case, a communication channel is needed between the DSLAM and the node hosting the software components. To keep the model simple, we assume here that the software components reside in the DSLAM and no additional communication or hosting entities are required. The first software component is the OpenFlow controller, which provides the base functions for OpenFlow-enabled devices (e.g., the forwarding engine) and forwards special packets to desired applications capable of providing correct control functions to the flows. One of these applications is the PPPoE application, the second software component, which handles the main part of the control of typical BRAS-like creation of user sessions or configuration of session parameters. Related to the PPPoE application is the session data base (Session DB), which contains all information about ongoing PPPoE user sessions. In practice this could be a connection to the RADIUS platform (as in typical operator platforms) or a locally configured database. The other two software components are the PPPoE discovery processing engine (PDPE) and the PPPoE session processing engine (PSPE). Both are not necessarily software components only, but might include a hardware part for the specific processing as well. Today the DSLAM can (already) typically process certain kinds of PPPoE messages and, depending on the capabilities, this could be used for general PPPoE processing as well. The processing engines are connected to the forwarding engine by virtual ports, which in turn act like “normal” physical ports. For the PSPE, there are two ports for the PPPoE and for Ethernet-tagged packets each, as well as a configuration interface. These two components are responsible for acting in the session initiation phase (PDPE) and for the continuous PPPoE operation (PSPE) once customers try to establish, configure, transmit and terminate a PPPoE session. The PDPE is independent of any other components and is integrated by setting TCAM entries in the forwarding engine directing any session initiation message (PADI and PADO) to this component. By contrast, the PSPE is dependent on the OpenFlow controller and the respective PPPoE application. The initiation messages for the concrete session (PADR and PADS) are forwarded to the PPPoE application. This PPPoE application handles all the necessary details for the session establishment and triggers the PSPE for setting up a session interface and context for the specific user request. Additionally, monitoring information (like termination acknowledgement) is transmitted (like termination of session) between the PPPoE application and PSPE. More details on the message flow are detailed in the following paragraphs.

Figure 64 shows different kinds of connectivity. Physical connectivity refers to a link between ports and the forwarding engine transmitting packets in the fashion a switch would do in a normal, non-OpenFlow mode of operation. It should be noted that some physical connections are not shown in the figure, for example between the PPPoE application and the PSPE. In addition, there is OpenFlow connectivity, which requires a physical connectivity that is based on the OpenFlow protocol only. There is also logical connectivity between different entities. They represent transmission of certain control message types as well as connections between certain components. In addition, the messages transmitted between certain entities are included in order to provide an initial overview of the distribution of these messages in the system as such. A more detailed flowchart is shown in the Figure 65.

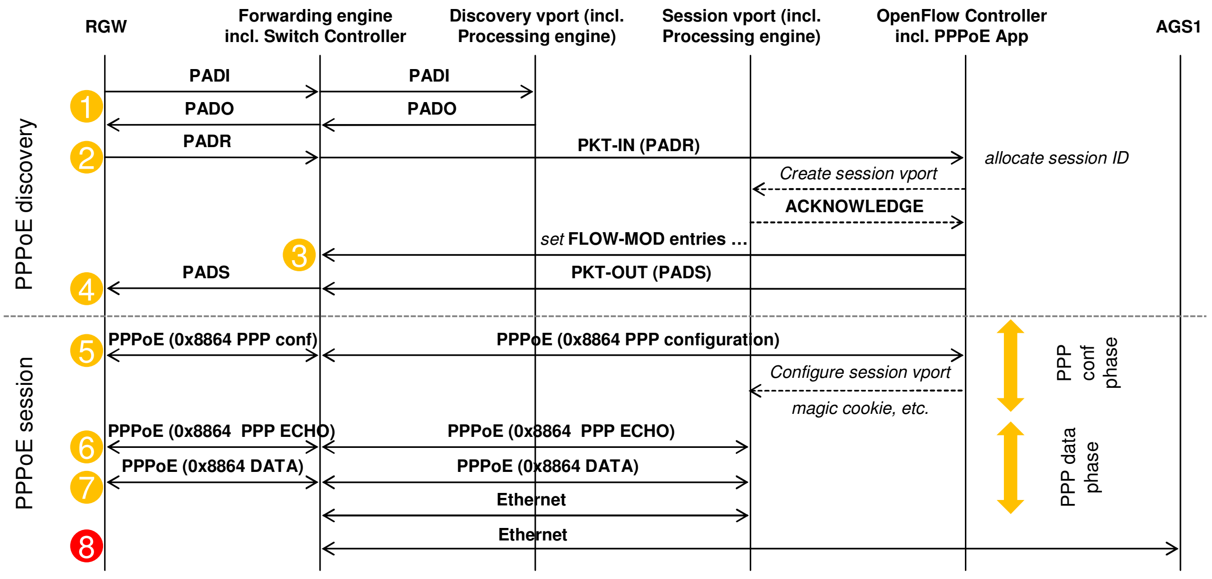


Figure 65: Flowchart for PPPoE connection in DSLAM scenario

The PPPoE connection is split into two phases, discovery and session, which can be described in even more detail. The flowchart depicts eight steps and their relationship with the components at the x-axis and the different phases on the y-axis. It illustrates a successful connection setup, but does not include PPPoE session termination and potential unsuccessful steps during connection setup and session configuration. The details of the eight steps are as follows:

1. The user wants to set up Internet connectivity and the RGW sends a PADI message (simplified: who is a BRAS and what is your address?) to the DSLAM. Based on a preconfigured entry in the forwarding engine, the message is forwarded to the discovery virtual port belonging to the PDPE, which replies with an appropriate PADO message (simplified: I am and this is my address.) to the RGW. The desired BRAS address included in the message could be implemented in two ways: It could point to the OpenFlow controller or the PPPoE application (this requires a MAC address for one of the elements), or it employs a used, uncommon address (is not already present or known in the MAC table of the switch) while the next PPPoE packet in the connection setup is forwarded to the OpenFlow controller and the PPPoE application anyway. With the reception of the PADO message, the RGW knows the address of the BRAS and starts the "real" PPPoE connection setup.

2. In step two the RGW sends the PADR message (simplified: I want a connection with this BRAS.) to the DSLAM. Here the message type is unknown to the forwarding engine and therefore is forwarded to the OpenFlow controller, which is able to identify it as a PADR message and forwards it to the PPPoE application. The PPPoE application acts similarly to the BRAS and creates a session with a respective ID and forwards this information to the PSPE. This could be done either by using the PSPE session virtual port, a separate virtual port or another kind of link between these two entities. The PSPE has to create an appropriate session context which allows the PSPE to properly react to data sent from the user and the Internet (represented by the AGS1) by encapsulating or decapsulating the messages with a PPoE header. When the PSPE has been successfully configured, this is acknowledged to the PPPoE application.

3. After configuration of the processing module, the forwarding engine requires some more information. Based on the session ID and other information (like MAC and IP addresses), flow_mod entries are calculated and forwarded via the OpenFlow controller to the forwarding engine. Here the PPPoE application needs to take information already anticipated from step 5 into account, e.g., the IP address. Now packets from the RGW and AGS1 are forwarded to the PSPE and then forwarded to the respective output port.

4. In this step the connection setup is finalized by sending the PADS message to the RGW. This is done by the PPPoE application which encapsulates the answer for step 2 in a packet-out message and forwards it via the

OpenFlow controller to the forwarding engine, which transforms it into a PPPoE PADS message (simplified: this is your session information) and sends it via the physical output port to the RGW.

5. After the discovery stage the session stage starts. In the first step, necessary protocols are configured or information for network configuration exchanged. This includes MTU size, IP address, network mask, etc. which is transported in special PPPoE messages like LCP or IPCP. These messages are sent from the RGW to the DSLAM which forwards them to the PPPoE application and vice versa. After successful agreement on the parameters, the session context in the PSPE needs to be updated properly. Depending on the implementation, forwarding entries (flow table entries) need to be updated. In step 3 it was assumed that the IP address was already anticipated by the PPPoE application, but the configuration of the flow table entries could be done in this step as well. In addition, if something has changed (for whatever reason), modifications of the flow table entries might be necessary in this step. But after this step, data could be exchanged as represented in step 7.
6. This step is important for operators in order to monitor the connectivity. The BRAS, represented here by the PSPE, sends echo messages to the RGW and in the case of an active session, receives replies. A reply is sent by the PSPE to the forwarding engine and then forwarded to the appropriate xDSL port to the customer. Accordingly, reply messages are transmitted in the reverse way.
7. The data transmission consists of two parts. The first one, described in this step, consists of the packet transmission between RGW and PSPE; the second consists of the transmission of packets between PSPE and AGS1. Packets are encapsulated by the RGW with PPPoE and a marking that these packets contain data (and not configuration or monitoring information). This is recognized by the forwarding engine with the help of appropriate forwarding entries (step 3 or 5) and forwarded to the PSPE, which then decapsulates the packets.
8. In this step, the decapsulated packets from step 7 are forwarded by the PSPE to the forwarding engine, which in turn forwards them to the output port connected with the AGS1. Transmission of packets from AGS1 to RGW follows step 7 and 8 in reverse.

One very important aspect in this model is the configuration of appropriate flow table or forwarding entries in the forwarding engine in order to connect the different components. Figure 66 is a list of potential TCAM entries for the model detailed in this section. While changing some aspects, the flow table entries have to be revised accordingly.

	Nr.	inport	Source Address	Destin. Address	Ether Type	Output	OF packet	Remarks
PPPoE discovery	1	downlink port	RGW MAC	Broadcast MAC	0x8863	Discovery VP (PD)		PADI
	2	Discovery VP (PD)	VAC MAC	RGW MAC	0x8863	downlink port		PADO (downlink port as metadata required!)
	3	downlink port	*	VAC MAC	0x8863	OFPP_CONTROLLER		PADR
	4	OFPP_CONTROLLER	VAC MAC	RGW MAC	0x8863	downlink port	PKT-OUT	PADS
PPPoE session	5	downlink port	RGW MAC	VAC MAC	0x8863	Session VP (PS1)		one PPPoE Session VP (e.g. PS1) per user session
	6	downlink port	RGW MAC	VAC MAC	0x8864	Session VP (PS1)		
	7	Session VP (PS1)	RGW MAC	VAC MAC	0x8863	OFPP_CONTROLLER		e.g. PADT
	8	Session VP (PS1)	RGW MAC	VAC MAC	0x8864	OFPP_CONTROLLER		all PPP packets, except DATA and ECHO (e.g. PPP Config)
	9	Session VP (PS2)	VAC MAC	AGS1 MAC	0x0800	uplink port		
	10	uplink port	AGS1 MAC	VAC MAC	0x0800	Session VP (PS2)		
	11	OFPP_CONTROLLER	VAC MAC	RGW MAC	0x8863	downlink port	PKT-OUT	PADT
	12	Session VP (PS1)	VAC MAC	RGW MAC	0x8864	downlink port		PPP DATA and ECHO PDU

VAC MAC = Virtual Access Concentrator MAC (= BRAS / PPPoE App) indicates Routing mode, otherwise routing gateway MAC must be used

Figure 66: Forwarding entries for PPPoE-based service creation in DSLAM

There are twelve entries required, four represent the discovery phase and eight the session phase. Two entries (1 and 2) are required for step 1 of Figure 66. The first one could be even more generic with a completely "wild carded" MAC address. Entries three and four are required for the forwarding of the connection setup to the PPPoE, message four is a packet-out message which needs to be handled in the forwarding engine. The other messages depend on the existence of a virtual access concentrator (VAC) MAC address (PPPoE standards refer to the term as access concentrator, but this is not precise enough for our model). In normal PPPoE operation, packets are intercepted by the BRAS and therefore the source MAC address is exchanged or RGW uses the BRAS as the destination MAC address. In our concept two models could apply. If the BRAS model is completely transferred to the DSLAM, a routing interface per customer needs to be created which has its own MAC address. This model is called routing model hereafter. In the second option, the routing interface remains at a more concentrated location (e.g., AGS1 or LER) and no routing interface per customer needs to be created. But the MAC address of the routing interface needs to be used in the packets and therefore learned by the PSPE and/or PPPoE app. The forwarding entries in Figure 66 assume the routing model and use the address of a virtual access concentrator (virtual means that this device is not dedicated to be BRAS and the implementation exists in software only). The data transmission is handled with the flow tables entries 6, 9, 10 and 12. The entries 5, 7, 8 and 11 are needed for forwarding control information like monitoring requests.

Abbreviations

3GPP	Third-generation partnership program
ADSL	Asymmetric Digital Subscriber Line
AES	Advanced Encryption Standard
AGS	Aggregation Switch
ANDSF	Access Network Discovery Selection Function
AP	Access Point
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
ATM	Asynchronous Transfer Mode
AWG	Arrayed-waveguide Grating
BB	Broadband
BBA	Broadband Access
BBF	Broadband Forum
BB-RAR	Broadband Remote-Access-Router (SCP for Fast Internet)
BE	Best-Effort
BFD	Bidirectional Forwarding Detection
BG	Broadband Aggregation Gateway
BGP	Border Gateway Protocol; Distance Vector Routing protocol of IETF (EGP)
BRAS	Broadband Remote Access Server / Service
BRPC	Backward Recursive Path Computation
BSS	Basic Service Set
CAR	Committed Access Rate
CBO	Class-Based Queuing
CBWFQ	Class-Based Weighted Fair Queuing
CCM	Continuity Check Message
CDMA	Code Division Multiple Access
CE	Control Element
CHG	Customer HG; HG in customer site
CIDR	Classless Inter-Domain Routing
CIPM	Cisco IP Manager
CIR	Committed Information Rate
CLI	Command Line Interface
CLNC	Customer LNS, LNS in customer site
CORBA	Common Object Request Broker Architecture
CoS	Class of Service
CP	Connectivity Provider
CPE	Customer Premise Equipment
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CR-LDP	Constraint-based LDP
CSCF	Call Session Control Function
DCF	Distributed Coordination Function
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
DiffServ	Differentiated Services, IETF
DNS	Domain Name Server
DOCSIS	Data Over Cable Service Interface Specification
DPI	Deep Packet Inspection
DRR	Deficit Round Robin
DS	Differentiated Services
DSCP	Diff Serve Code Point
DSL	Digital Subscriber Line
DSLAM	Digital Subscriber Line Access Multiplexer (network side of ADSL line)
DWDM	(Dense) Wave-Division-Multiplex
dWRED	Distributed WRED
DXC	Digital Cross-Connect
ECMP	Equal Cost Multi-Path
ECN	Explicit Congestion Notification
ECR	Egress Committed Rate
EGP	Exterior Gateway Protocol
EIGRP	Enhanced IGRP
EN	Edge Node
ePDG	Evolved Packet Data Network Gateway
ESS	Extended Service Set
FE	Forwarding Element
FEC	Forwarding Equivalence Class
FEC	Forward Error Correction
FIB	Forwarding Information Base
FMC	Fixed Mobile Convergence
ForCES	Forwarding and Control Element Separation
FPGA	Field Programmable Gate Array
FSC	Fiber Switching
FTTCab	Fiber to the Cabinet
FTTH	Fiber to the Home
FTTLEx	Fiber to the Local Exchange

FW	Firewall
GbE	Gigabit Ethernet
GFP	Generic Framing Procedure
GLONASS	Globalnaja Nawigazionnaja Sputnikowaja Sistema (Russian satellite system)
GMPLS	Generalized Multi-Protocol Label Switching
GNSS	Global Navigation Satellite System
GPON	Gigabit Passive Optical Network
GPS	Global Positioning System
GRE	Generic Route Encapsulation
GTS	Generic Traffic Shaping
GUI	Graphical User Interface
HCF	Hybrid Coordination Function
HDLC	High-level Data Link Control
HG	Home Gateway
HIP	Host Identify Protocol
IACD	Interface Adjustment Capability Descriptor
ICR	Ingress Committed Rate
ICT	Information and Communication Technology
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force (www.ietf.org)
IF	Interface
ISC	Interface Switching Capabilities
IGMP	Internet Group Management Protocol
IGP	Interior Gateway Protocol
IGRP	Interior Gateway Routing Protocol
IntServ	Integrated Services, IETF
IP	Internet Protocol
IPTV	IP television
ISCD	Interface Switching Capability Descriptor
ISDN	Integrated Services Digital Network
IS-IS	Intermediate System - Intermediate System; Link State Routing Protocol from OSI (IGP)
ISO	International Organization for Standardization
ISP	Internet Service Provider
ITIL	IT Infrastructure Library
ITU	International Telecommunication Union
L2F	Layer 2 Forwarding
L2TP	Layer 2 Tunnel Protocol
LAC	L2TP Access Concentrator
LAN	Local Area Network
LDP	Label Distribution Protocol
LER	Label Edge Router; MPLS-based router with MPLS, IP-VPN and QoS edge support
LER-BB	Broadband LER; LER for DS and higher
L-GW	Local Gateway
LLDP	Link Layer Discovery Protocol
L-LSP	Label-inferred LSP
LMP	Link Management Protocol
LNS	L2TP Network Server
LSP	Label Switch Path
LSR	Label Switch Router; MPLS-based router in the inner IP network. Only IGP knowledge.
LTE	Long Term Evolution
MAC	Media Access Control
MAN	Metropolitan Area Network
MEF	Metro Ethernet Forum
MGW	Media Gateway
MIB	Management Information Base
MLD	Multicast Listener Discovery
MLTE	Multilayer Traffic Engineering
MME	Mobility Management Entity
MPLS	Multi-Protocol Label Switching
MPLS-TP	MPLS Transport Profile
MSC	Mobile Switch Controller
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NGN	Next-Generation Network
NIC	Network Interface Controller
NMS	Network Management System
NNI	Network-to-Network Interface
NP	Network Provider
NSP	Native Service Processing
NTP	Network Time Protocol
OAM	“Operation, Administration and Maintenance” or “Operations and Maintenance”
ODU	Optical Data Unit
OF	OpenFlow
OFDM	Orthogonal Frequency Division Multiplexing
OLT	Optical Line Termination
OSI	Open Systems Interconnection
OSNR	Optical Signal-to-Noise Ratio

OSPF	Open Shortest Path First; Link State Routing Protocol from IETF (IGP)
OTN	Optical Transport Network
OTU	Optical Transport Unit
OXC	Optical Cross-Connect
PANA	Protocol for carrying Authentication for Network Access
PBB-TE	Provider Backbone Bridge Traffic Engineering
PCEP	Path Computation Element Communication Protocol
PDU	Protocol Data Unit
PE	Provider Edge; Service Creation Point for IP-VPN
PER	Provider Edge Router
PGW	Packet Data Network Gateway
PIM	Protocol Independent Multicast
PIP	Physical Infrastructure Provider
PMIP	Proxy Mobile IP
PoP	Point of Presence
POTS	Plain Old Telephony Service
PPP	Point-to-Point Protocol
PPPoE	PPP over Ethernet
PSTN	Public Switched Telephone Network
PVC	Permanent Virtual Circuit (permanent L2 connection, e.g., Frame Relay, ATM)
PWE	PseudoWire Emulation
QoE	Quality of Experience
QoS	Quality of Service; general for differentiated quality of services or absolute quality of services.
QPSK	Quadrature Phase-Shift Keying
RADIUS	Remote Authentication Dial-In User Service
RAR	Remote Access Router (SCP for OCN)
RARP	Reverse ARP
RFC	Request for Comment (in IETF)
RGW	Residential Gateway
RIB	Routing Information Bases
RIP	Routing Information Protocol; Distance Vector Routing Protocol from IETF (EGP)
ROADM	Reconfigurable Optical Add-Drop Multiplexer
RR	Route Reflector for BGP/MP-BGP
RTP	Real Time Protocol
RTT	Round Trip Time
SAE	System Architecture Evolution
SAP	Service Access Point
SBC	Session Border Controller
SCP	Service Creation Point
SDH	Synchronous Digital Hierarchy
RSVP (-TE)	ReSource reserVation Protocol (-Traffic Engineering)
SDN	Software Defined Networking
SDU	Service Data Unit
SE	Service Edge
SGW	Serving Gateway
SHG	Separate HG, separate HG device with virtual HG
SIP	Session Initiation Protocol
SIPTO	Selective IP Traffic Offload
SLA	Service Level Agreement
SLNS	Separate LNS, separate LNS device with virtual LNS
SMS	Service Management System
SNMP	Simple Network Management Protocol
SONET	Synchronous Optical Network
SP	Service Provider
SPARC	Split Architecture for carrier-grade networks
SSID	Service Set Identifier
SSM	Source Specific Multicast
STA	Station
STM	Synchronous Transfer Module (STM-1: 155 Mbit/s, STM-4: 622 Mbit/s, STM-16: 2.5 Gbit/s; STM-64: 10 Gbit/s)
TCAM	Ternary Content Addressable Memory
TCM	Tandem Connection Monitoring
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TE	Traffic Engineering
TKIP	Temporal Key Integrity Protocol
ToR	Top of the Rack
ToS	Type of Service
TR	Technical Report (from BBF)
TTL	Time to live
UDP	User Datagram Protocol
UNI	User Network Interface
VEB	Virtual Ethernet Bridges

VEPA	Virtual Ethernet Port Aggregator	WAN	Wide Area Network
VLAN	Virtual LAN	WDM	Wavelength Division Multiplexing
VM	Virtual Machine	WEP	Wired Equivalent Privacy
vNIC	Virtual NIC	WFQ	Weighted Fair Queuing
VoIP	Voice over IP	WP	Work Package
VPLS	Virtual Private LAN Service	WSON	Wavelength Switched Optical Network
VPN	Virtual Private Network	WT	Working Text (from BBF)
VSI	Virtual Station Interface		

References

[1] K. Kompella, Y. Rekther, "Virtual Private LAN service (VPLS): Using BGP for Auto-Discovery and Signaling," IETF RFC 4761, 2007 (accessed: 2011-09-05).

[2] Metro Ethernet Forum (MEF), online: http://metroethernetforum.org/index.php (accessed: 2011-09-05)

[3] OpenFlow 1.2 Proposals, wiki online: http://www.openflow.org/wk/index.php/ OpenFlow_1_2_proposal (accessed: 2011-09-05)

[4] W. Simpson, "The Point-to-Point Protocol (PPP)," IETF RFC 1661, 1994 (accessed: 2011-09-05)

[5] Toward Real Energy-efficient Network Design (TREND), online: www.fp7-trend.eu (accessed: 2011-09-05)

[6] Energy efficiency in large scale distributed systems (IC804), online http://www.irit.fr/cost804/ (accessed: 2011-09-05)

[7] GreenTouch, online: http://www.greentouch.org (accessed: 2011-09-05)

[8] GreenStar Network, online: http://www.greenstarnetwork.com/ (accessed: 2011-09-05)

[9] R.S. Tucker, R. Parthiban, J. Baliga, K. Hinton, R.W.A. Ayre, W.V. Sorin, "Evolution of WDM optical IP networks: A cost and energy perspective," IEEE Journal of Lightwave Technology 27(3), 243-252, 2009

[10] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, Sujata Banerjee: "DevoFlow: Scaling Flow Management for High-Performance Networks." SIGCOMM'11.

[11] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar: "Can the production network be the testbed?" In OSDI, 2010.

[12] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker: "Applying NOX to the Datacenter." In HotNets, 2009.

[13] Z. Cai, A. L. Cox, and T. S. E. Ng. "Maestro: A System for Scalable OpenFlow Control". Tech. Rep. TR10-08, Rice University, 2010.

[14] A.Farrel, I.Bryskin. "GMPLS – Architecture and Applications." Morgan Kaufman, Elsevier 2006.

[15] Expedient. Control framework for OpenFlow test beds. Available online: http://yuba.stanford.edu/~jnaous/expedient/ and http://groups.geni.net/geni/wiki/OpenFlow/Expedient

[16] ICT-258457 SPARC Deliverable D2.1 "Initial Definition of Use Cases and Carrier Requirements." Available online: http://www.fp7-sparc.eu/project/deliverables/

[17] M.Casado, T.Koponen, D.Moon, S.Shenker. "Rethinking Packet Forwarding Hardware." Seventh ACM Workshop on Hot Topics in Networks (HotNets-VII), Calgary, Alberta, Canada, October 6-7, 2008

[18] J.C.Mogul, P.Yalagandula, J.Tourrilhes, R.McGeer, S.Banerjee, T.Connors, P.Sharma. "API Design Challenges for Open Router Platforms on Proprietary Hardware." Seventh ACM Workshop on Hot Topics in Networks (HotNets-VII), Calgary, Alberta, Canada, October 6-7, 2008

[19] OpenFlow 1.1 Specification, online: www.openflow.org/documents/openflow-spec-v1.1.0.pdf (accessed: 2011-11-29)

[20] SARA Computing & Networking Services, "dot1ag-utils." Online: https://noc.sara.nl/nrg/dot1ag-utils/index.html (accessed: 2011-12-01)

[21] L. Yang, R. Dantu, T. Anderson, R. Gopal. "Forwarding and Control Element Separation (ForCES) Framework." IETF RFC 3746, 2004 (accessed: 2011-12-20).

[22] Rob Sherwood, Glen Gibby, Kok-Kiong Yapy, Guido Appenzellery, Martin Casado, Nick McKeowny, Guru Parulkary. "FlowVisor: A Network Virtualization Layer," http://www.openflowswitch.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf (accessed: 2011-12-20)

[23] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. 2008. NOX: toward an operating system for networks. SIGCOMM Comput. Commun. Rev. 38, 3 (Jul. 2008), 105-110. http://doi.acm.org/10.1145/1384609.1384625

[24] Beacon Openflow controller, https://openflow.stanford.edu/display/Beacon/Home (accessed 2011-12-20)

[25] Trema Openflow framework, http://trema.github.com/trema/ (accessed 2011-12-20)

[26] John Day, Ibrahim Matta, Karim Mattar."Networking is IPC: a guiding principle to a better internet." CoNEXT, 2008

[27] Y. Lee, G. Bernstein, Wataru Imajuku.“ Framework for GMPLS and PCE Control of Wavelength Switched Optical Networks (WSON),” IETF Internet draft (informational), 2011, https://tools.ietf.org/html/draft-ietf-ccamp-rwa-wson-framework-12 (accessed 2011-12-20)

[28] M.R. Nascimento, C.E. Rothenberg, M.R. Salvador, M.F. Magalhães,"QuagFlow: partnering Quagga with OpenFlow," SIGCOMM, 2010

[29] Casado, M. and Koponen, T. and Ramanathan, R. and Shenker, S., "Virtualizing the network forwarding plane," Workshop on Programmable Routers for Extensible Services of Tomorrow, 2010

[30] Salvadori, E. and Corin, R.D. and Gerola, M. and Broglio, A. and De Pellegrini, F., "Demonstrating generalized virtual topologies in an openflow network," ACM SIGCOMM, 2011

[31] OpenvSwitch, "Open vSwitch: An Open Virtual Switch," http://openvswitch.org (accessed 2012-12-22)

[32] lxc Linux Containers, http://lxc.sourceforge.net (accessed 2012-12-22)

[33] Foster, N. and Harrison, R. and Freedman, M.J. and Monsanto, C. and Rexford, J. and Story, A. and Walker, D., "Frenetic: A network programming language," ACM SIGPLAN, 2011

[34] IXIA, "10-Gigabit Ethernet Switch Performance Testing," http://www.ixiacom.com/pdfs/library/white_papers/10ge.pdf (accessed 2012-12-22)

[35] Barreiros, M. and Lundqvist, P., "QOS-enabled Networks: Tools and Foundations," Wiley, 2011

[36] Farrel, A. et al., "Network Quality of Service Know It All," Morgan Kaufmann, 2009

[37] Perros, H.G., "An introduction to ATM networks," Wiley, 2002

[38] Shirazipour M., Tatipamula M., “Design Considerations for OpenFlow Extensions Toward Multi-Domain, Multi-Layer, and Optical Networks”, ECOC OFELIA Workshop, 2011

[39] Authenrieth, A.: “Extending OpenFlow to Optical Wavelength Switching – Challenges, Requirements, and Integration Model”, ECOC OFELIA Workshop, 2011, slides to be found at http://www.fp7-ofelia.eu/news-and-events/workshops/ecoc-2011-ofelia-workshop/ (also for [38])

[40] N. Leymann, B. Decraene, C. Filsfils, M. Konstantynowicz, D. Steinberg, "Seamless MPLS Architecture," IETF Internet draft (informational), 2011