SPARC

ICT-258457

Deliverable 3.1

Initial Split Architecture and Open Flow Protocol study

Editor:	*Sonny Thorelli, Ericsson (EA)*
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M4
Actual delivery date:	M4
Version:	1.0
Total number of pages:	49
Keywords:	Split Architecture, ForCES, GMPLS, PCE, OpenFlow, FlowVisor, Network Virtualization, resiliency, Carrier Grade, Muli-layer, OAM, Virtual Port, MPLS, hierarchy, federation, topology, scalability.

Abstract

This SPARC deliverable addresses the architectural aspects of separating control plane from data plane in switching and routing. Moreover the data-plane can be separated in forwarding and processing entities. Similar concept was successfully deployed in other telecom applications. The objective of this work is to describe the state of the art and identify the outstanding issues to be solved when extending OpenFlow to meet carrier grade requirements for three identified use cases (WP2). Several aspects and characteristics that are missing currently are discussed and open questions identified. There are many open questions: how to achieve scalability with centralized controller, what processing is essential at data plane, how is the lack of OAM support solved, how do we support resilience, can there be a generic approach to extend data plane functionality, how is an circuit switched optical layer integrated, how do we coordinate muli-layer resiliency, how do we deal with network virtualization, and more.

Imprint

[Project title]	***Split Architecture***
[short title]	***SPARC***
[Number and title of work package]	***W3 – Architecture***
[Document title]	***Initial Architectural Split and Open Flow protocol study***
[Editor]	***Sonny Thorelli, Ericsson***
[Work package leader]	***Sonny Thorelli, Ericsson***
[Task leader]	***Sonny Thorelli, Ericsson***

Executive summary

This is the SPARC project first technical deliverable, addressing the architectural aspects of separating control plane from forwarding plane. This objective in this report is to form a baseline and document what is known, while identifying issues that need to be studied in subsequent deliverables.

Split Architecture, separation of data plane devices and control logics, has already been successfully deployed in operator domains (e.g. MSC-S/MGW split), cutting cost, simplifying complexity of distributed network devices, while centralizing control logic to primary sites. Currently we are in the prospect of extending the use of packet technologies throughout the operator networks, thus, to obtain a more packet efficient delivery of new services, e.g. the ubiquitous Ethernet. Therefore, this transformation would also benefit from a split architectural approach for a more efficient and flexible distribution of network functions.

There are three types of architectural split approach discussed in this report. 1) Open Flow that generalizes flow table forwarding and associates with virtual machines, data center and cloud computing. 2) ForCES that provide a comprehensive and generic framework for separation of forwarding and control functions within the IP layer. 3) GMPLS/PCE separates control and transport plane.

OpenFlow is the scheme that this project is targeting. Despite OpenFlow provides for several interesting characteristics it lacks many of the Carrier Class characteristics. One step in this direction has been to extend OpenFlow to handle also widely used MPLS. However, MPLS actions manipulate the header differently than other OpenFlow 1.0 actions; therefore, MPLS actions were isolated using an abstraction called a Virtual Port. This generic approach seems to be useful also when extending OpenFlow further to accommodate new flow definitions and actions without having to redefine the basic flow structure in an incompatible way.

Network virtualization is a technique exploited in many areas of networking both to increase the network utilization and simplify network management. Network virtualization allows several network instances to co-exist within a common physical infrastructure. It’s important though, the different network instances provide for a high integrity without influencing each others.

Currently there are two systems for virtualizing networks using OpenFlow: the policy based FlowVisor proxy and Onix, a more object oriented approach. Based on policy FlowVisor separates of the network into partitions called *slices* using the different match fields available in the OpenFlow FlowTable. Onix, on the other hand, allows for an explicit definition of a Virtual Network that is instantiated through OpenFlow. The Onix model could allow more transparent creation of Virtual Networks where problems such as coexistence of colliding address-spaces could be handled transparently.

Scalability becomes an important characteristic when centralizing control logic. Hierarchy and federation are two basic mechanisms to improve scalability. By adding an additional layer of controllers would result in an aggregate of underlying devices and networks. Federation is another tool that can improve scaling by partitioning the network into domains, where controllers exchange only aggregate information among themselves. However, this opens up new questions around synchronizing the controllers between different domains.

Carrier network are also characteristic by reliability and service management, which traditional networks are supports with the use of OAM tools. OAM is providing a large range of tools, ranging from connectivity check/validation to various forms of performances monitoring, where the latter validates Service Layer Agreements. OpenFlow currently do not at all define such OAM mechanisms and remains as one of the important open question to be answered, e.g. OAM flows?

OpenFlow basically collapses several network layers, L2-L4 (ETH MAC to TCP/UDP port), into one. However, attempt has been made to also include the optical layer (e.g. OTN/WDM), but these differ in characteristics by being circuit switched network layers. In order to solve this, introduction of a Virtual Port is discussed. When considering multi-layer survivability means more than coordinating restoration between multiple layers; however, maybe even would gain by being collocated at common controller that OpenFlow would provide.

The conclusion of this deliverable collects a number of open questions, which are suggested to be treated in sub-sequence studies within this project.

List of authors

Organisation/Company	Author
EA	Sonny Thorelli (Editor)
EA	Zhemin Ding
EA (SV)	James Kempf
ETH	Attila Takacs
IBBT	Dimitri Staessens
IBBT	Bart Puype
IBBT	Didier Colle
ACREO	Pontus Sköldström
ACREO	Anders Gavler
EICT	Hagen Woesner
DTAG	Mario Kind
DTAG	F.-Joachim Westphal
DTAG	Andreas Gladisch

Table of Contents

1 Introduction

1.1 Project Context

The project SPARC “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 with multi-million customers, 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 that exist in current components. It will build on OpenFlow and GMPLS technology as starting points and investigate if and how the combination of the two can be extend and migrate IP capabilities into operator networks with simpler and standardized technologies emerging from the data center.

1.2 Relation with other work packages

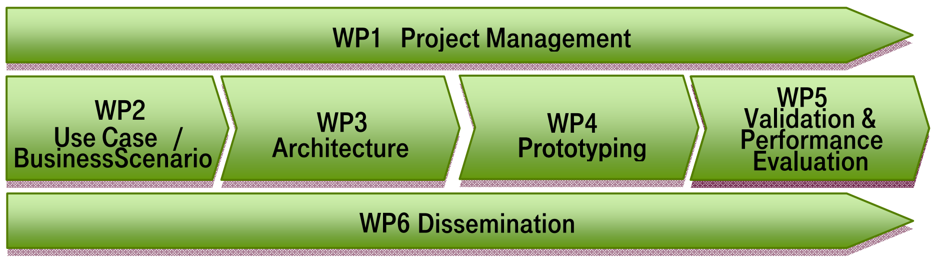

WP1 Project Management
WP2 Use Case / BusinessScenario
WP3 Architecture
WP4 Prototyping
WP5 Validation & Performance Evaluation
WP6 Dissemination

Figure 1: SPARC work package relation

In the “workflow” of the work packages, WP3 is embedded in-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 sub-set of resulting architecture, which feasibility will be validated in WP5. WP6 disseminates the result at international conferences and publications (Figure 1).

1.3 Scope of the deliverable

The Split Architecture consists of a system picture defining nodes, interfaces, procedures & protocols, APIs, and other potential networking and traffic aspects such as dimensioning, hierarchical relations etc. References to and experiences from implemented Split Architectures, specifically OpenFlow, form input to this task, as well as findings and results from work on use cases and requirements in WP2 ; the outcome is a generalization of a Split Architecture.

The intent with this “Initial Split Architectural and Open Flow Study”, deliverable D3.1 is to document the collective project understanding and state of the art on different aspects related to split architecture and to identify issues to be studied in sub-sequent deliverables.

2 Generals of split architecture

The use of packed based services has successfully been growing for more than 20 years; the growth in demand has so far been met by introducing ever larger monolithic routers. However, this technique is approaching its technologic and economic limits. It is more and more difficult to fulfill the increasing performance requirements with traditional router designs; and with the emergence of low cost data center hardware router vendors have also difficulties to validate the higher costs for the same performance compared to the price of commodity equipment. At the same time, the demands on the routing and switching control plane in access and aggregation networks are becoming more complex. Operators want the ability to customize routing to handle specific kinds of traffic flows near the edge, configure customized services that span aggregation networks, and achieve multi-layer integration, without the detailed low-level configuration typical of today's networks.

These trends suggest a different approach to routing architecture, in which the control plane is handled by a centralized server and the data plane consists of simplified switch/router elements "programmed" by the centralized controller.

This new routing architecture will focus on the split of control, forwarding and data processing elements. This evolution is similar to what happened in the mobile core network where the user plane and control plane processing has been decoupled and are supported today by different gateways.

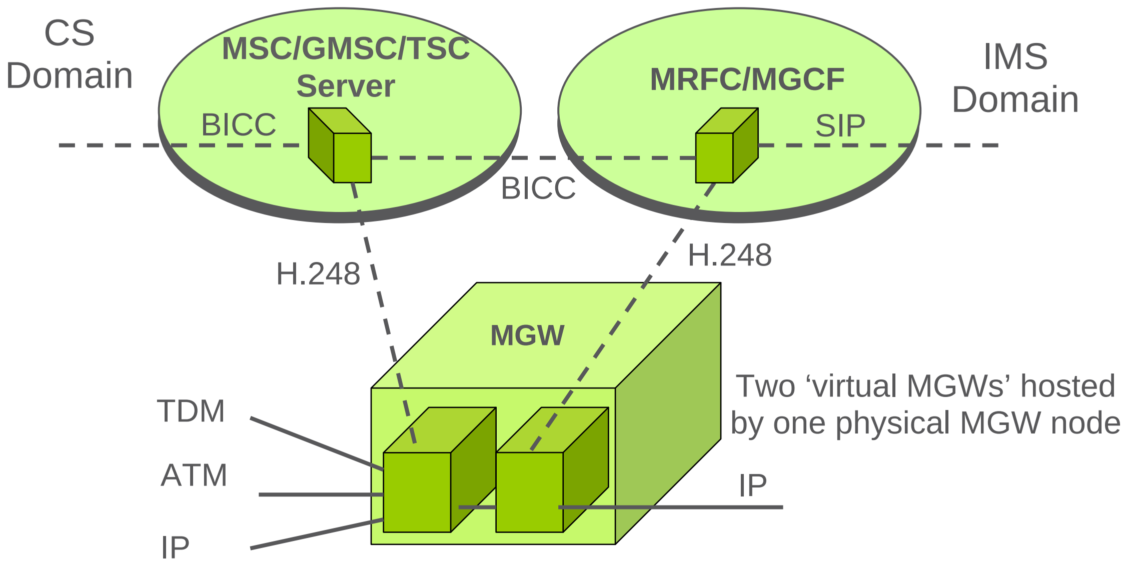


Figure 2: The mobile switch MSC was split into a MSC-Server and Media Gateway (MGW)

An earlier successful example is the split of the mature MSC mobile switch into MSC–server and Media-Gateway (Figure 2), deployed in real network 2004. Similar motivation for split architecture from this time also holds today. Here follows some key benefits motivating split architecture of the MSC node:

- Transport layer flexibility allowing easy of migration
  - use whatever transport technology
- Harmonization of transport technologies
  - a common based packet transport backbone
- Improved network resilience
  - overall higher network availability
- Centralized control and remote switching
  - simplified secondary sites and few primary sites

- ➢ Central device pools providing specific features centralized
  - limits need for upgrade to fewer sites
- ➢ Layer independency
  - evolve Service, Control, and Transport layer independently
  - allow rapid introduction of new network wide services

The split architecture enables a separation between functionalities which can be logically or physically grouped together. The key aspects are the following (see Figure 3):

- ➢ Split / Separation of a common control entity and a network-application, e.g. GMPLS, BGP, IPSec
- ➢ Split / Separation of control and forwarding/processing, i.e. separation of central control and network devices.
- ➢ Split / Separation of a data forwarding functionality and a data processing functionality, e.g. DPI, Ciphering, OAM.

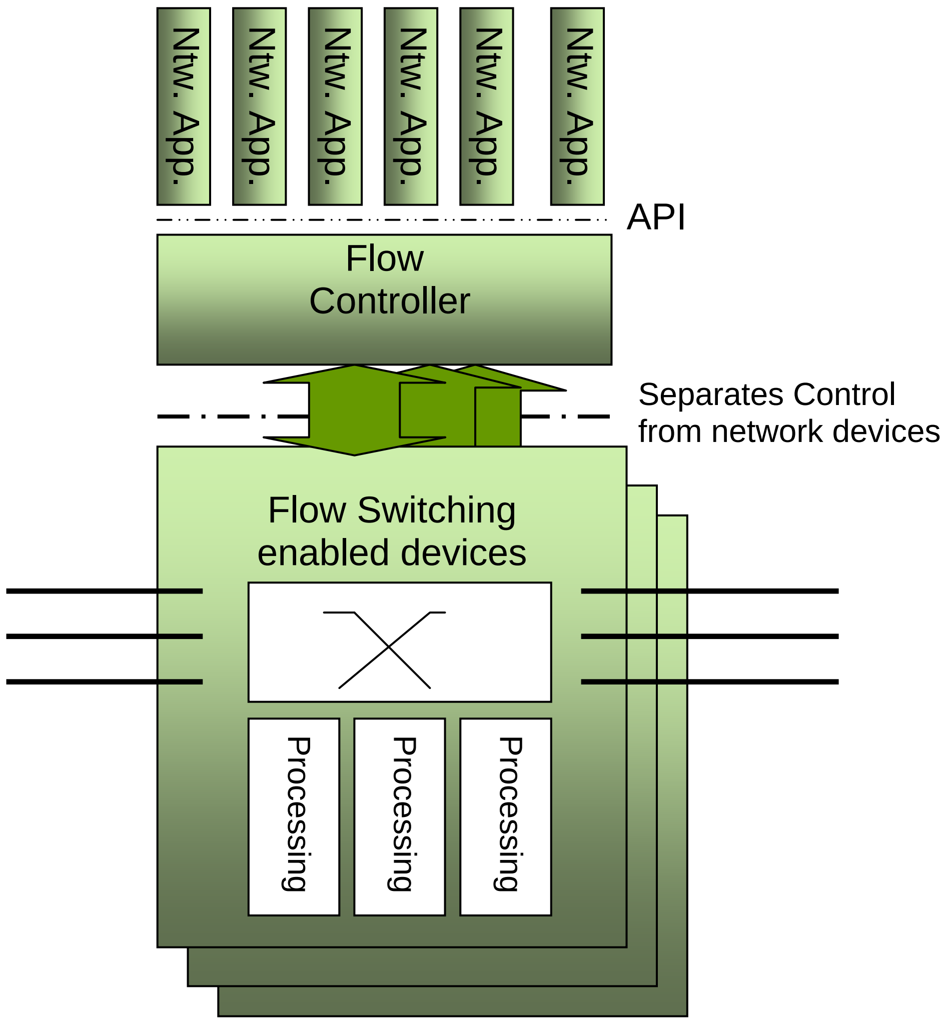


Figure 3: Aspects of the split architecture

The advantage of the split architecture are that customized control plane applications can be more easily and cheaply implemented on a server than on a monolithic router and switch control software that implements a distributed control plane. This centralized network control entity (a software component) balances traditional management plane and control plane functions, while the reliability and scalability of the controller infrastructure is becoming increasingly important. The centralized control approach solves issues involving long convergence times, unnecessary interactions between the different control mechanisms, and sub-optimal recovery and OAM at each layer that occur with the interaction between distributed control plane mechanisms such as BGP, IGPs, LDP, and STP/MAC learning.

Beside other influences, the recent discussion about Split Architecture was triggered by different US based activities dealing with OpenFlow. On the most generic level, three areas of applications can be identified for OpenFlow:

- Basic technology for experimental network facilities, providing the infrastructure to do experiments in the realm of networking research
- Application in and in-between data-center
- Application in the area of traditional, public telecommunication networks.

Concentrating on telecommunication networks, the Split Architecture principle can be applied at various parts of the network. Below the typical networking use-cases are illustrated.

Three use cases are identified (Figure 4), in WP2, for a closer study on architectural requirement of split routing architecture:

- Aggregation/access domain of public telecommunication networks including mobile-backhaul, and virtualization in order to isolate different service groups or establish a shared use of infrastructure in between different operators
- Application of Split Architecture in Data center domains could be manifold. Two of the most interesting aspects are load balancing and the support of session / server migration for coordination of switch configuration and assigned server resources in a more dynamic way
- Today's public telecommunication networks design is based on multi layers. While OSI introduces a layering of functions required for networking, each layer in a telecommunication network is corresponding to a different technology as in IP/MPLS/Ethernet/SDH/WDM networks. However, very often this design principle is not followed. Since OpenFlow is inherently flattening the network layering structure, it provides options to simplify optimizations and signaling for multilayer traffic engineering purposes and also multi-layer recovery.

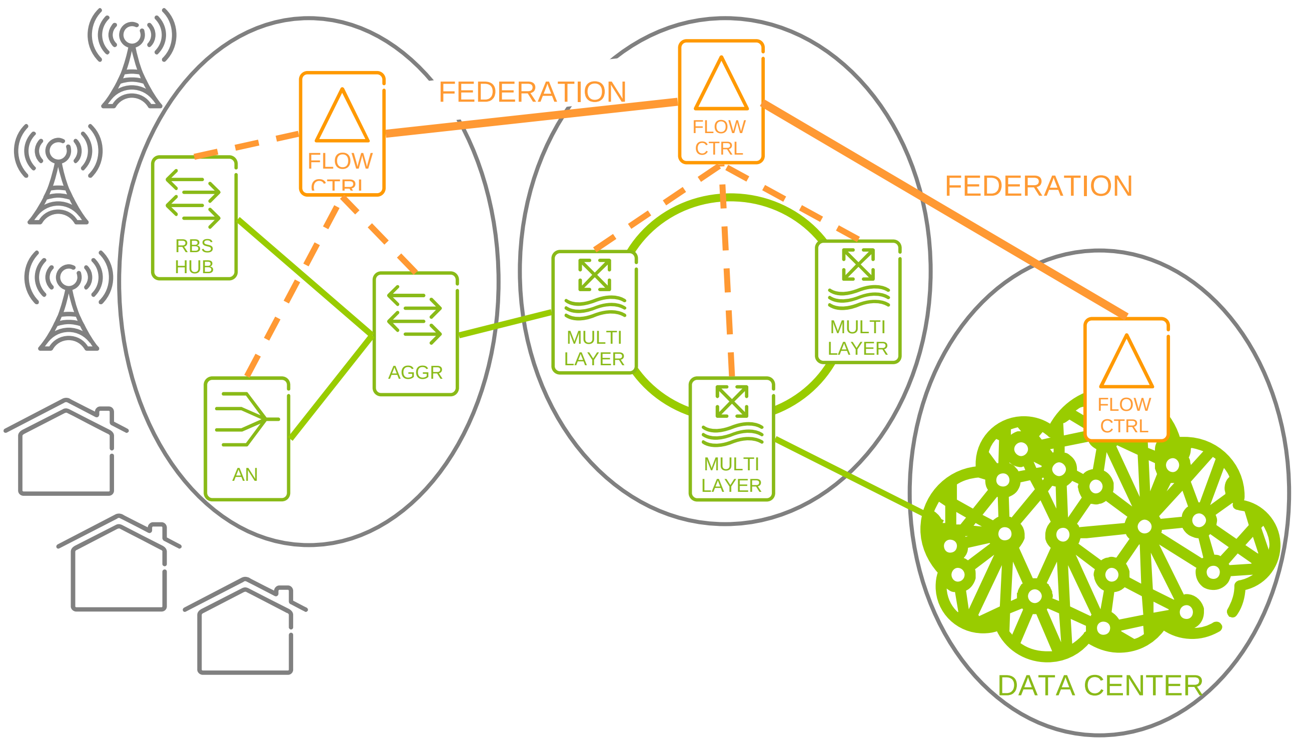


Figure 4: Three use cases: 1) access/aggregation incl. backhaul, 2) multi-layer incl. optical layer, and 3) data center.

3 Summarized requirements from WP2

Scope of this chapter is to summarize the first and preliminary results of the use case analysis and definition and requirements derived in WP2. The most important aspect in the context of SPARC is the extension of already existing split architecture approaches towards carrier grade characteristics.

3.1 Brief definition of Characteristics of Carrier Grade

The term “carrier grade” describes a set of functionalities and requirements that architectures should support in order to fulfill the operational part of network operators. There are five general attributes defined by Metro Ethernet Forum to define the term “carrier grade”:

- Standardized Services
- Scalability
- Reliability
- QoS
- Service Management

Standardized Services

For the “traditional” generic standardized services, there are three typical services which should be available:

- Virtual private line,
- Virtual private LAN and
- Virtual tree like services.

Those services are applicable on Layer 2 and 3 and with certain restriction on layer 1 circuit approaches as well. Each service requires (virtual) privacy and therefore a certain degree of isolation at data as well as control plane level. Isolation should provide the desired level of security as well.

In addition to those “traditional” services, services based on virtualization of the infrastructure (substrate network”) are an important topic of the project. A detailed definition in the context of the project is provided in section 7.

In brief a virtual network on top of a substrate network(s) using network slices, virtual nodes, and virtual links could be defined. Based on this definition new types of services possible:

- Virtualization on a substrate network level could be used to isolate service groups within the administrative domain of one operator (e.g. mobile backhauling and residential IP backhauling on a common aggregation infrastructure)
- Virtualization on a substrate network level could be as well used to isolate administrative domains of different operators. Based on this carriers-carrier service each operator will have a high degree of flexibility, e.g. by using different protocol stacks or different flavors of a control plane implementation.

Scalability

The scalability level is characterized by the future usage scenarios and should be aligned with the use case of WP2. There are two very distinct directions imaginable:

- The single operator, single segment, single customer and service case
- The multiple operator, multi segment (access, aggregation, metro, core as well as national and international), multiple customer (up to millions) and multiple service (business, information, communications and entertainment applications with voice, video and data) case

Reliability

On the reliability side, the primary target is to support the services running in the network with the demanded quality and availability requirements. Especially mobile networks are adding a new level of quality, see D2.1

for more details. In general, the network should be able to detect & recover from incidents without impacting users. In addition, the recovery time, when problems do occur, and configuration of OAM.

QoS

Quality of service (QoS) is an important matrix for the customer-operator and operator-operator interaction. First of all, a differentiation must be included and respected. Typically, QoS includes a defined level for the set of parameters, e.g. Committed Information Rate (CIR), bit error rate, etc.

Service Management

The service management aspect is of major importance in architectures and typically undervalued in development of new technologies. In general this includes the ability to monitor, diagnose and centrally manage the network. Anything should be based on standards and independent implementations. From current perspective, this should include:

- Configuration of OAM mechanisms
- Information exchange in-between network elements and network managements systems
- Specific requirements on autonomous forwarding decisions

3.2 Summary of requirements based on carrier grade characteristics

In the following sub-chapters more detailed requirements, clustered in the five defined areas are listed.

3.2.1 Standardized Services based requirements

The three “traditional” types of services and the two enhanced service groups, based on infrastructure virtualization should be available as defined in the paragraph above.

These services should fulfill the following requirements:

- Transparency towards the customer
- Interworking with customer networks as well as other operators (incl. control plane)
- Backwards compatibility / interoperability with existing services and protocols like TDM
- Openness with respect to granularity of bandwidth-, physical- and processing-resources and quality of service options
- Migration support in order to extend the architectures without disturbance of running services (cf. transparency). Here a certain degree of isolation is required, too.

3.2.2 Scalability

Some requirements on the architecture and system level:

- Architecture including set of Identifiers capable of representing up to millions of networks, network elements, customers and / or services
- Scalable mapping of customer domain and network domain address space
- Forwarding performance in terms of number connected customers and of flows to be supported in parallel as well as per second
- Scalable access and service parameter control lists (e.g. per customer / services for flows (which flow and number of parallel flows per customer / network))
- Scalable identifier in each flow for exact customer correlation

Similar requirements for data center, just slightly different vocabulary.

3.2.3 Reliability

In addition to the guarantee different classes of availability on a service level, the term reliability is also applicable on a network element level as well as on the scope of a whole network domain. Especially mobile networks are adding a new level of quality, more details will be provided in D2.1.

In general, the network should be able to detect & recover from incidents without impacting users.

3.2.4 Quality of Service

QoS includes defined levels for the set of parameters, like e.g.

- Committed Information Rate (CIR)
- bit error rate / frame loss,
- delay and delay variation

Those values should be defined for end-to-end as well as system level. It is difficult to define generic requirements for OpenFlow as requirements may vary from specific use case to case and be used in different modes of operation.

A detailed overview with respect to the access/aggregation use case could be found in deliverable D2.1 of the IST OASE project. It is best practice to define at least four different QoS classes in a public telco network application.

It is also a requirement that traffic carrying control information is of outstanding importance and should be handled within the highest QoS class.

Untagged traffic should be handled on a best effort basis.

3.2.5 Service Management

- Configuration of OAM mechanisms; control at link, system (incl. controller) and flow level
    - Identification of link failure
    - Loopback and connectivity check (e.g. ping and trace route)
    - Test signal
    - Performance measurement
    - The whole set of functions, described in IEEE 802.1ag (CFM) / ITU-T Y.1731
    - Analysis of configuration (with network management), e.g.
        - Interface up-/ down
        - Alarm signals
        - Performance measurement related information
- Information exchange in-between network elements and network managements systems
    - Generation of event based traps for all purposes and specific activities of protocols, limiters, timer, etc. indication of threshold violation
        - QoS like number of packets, bit error rate, packet loss, etc
        - MPLS forwarding data base
        - Number of routes to other network elements
    - Accounting information generation, e.g. bytes, start/end of session, etc.
- Autonomous forwarding decisions based on
    - At level of administrative decisions
    - At level of protocol based decisions; e.g. load balancing mechanisms like Equal Cost Multi Path (ECMP)
    - At level of policies; Bandwidth limitation in access node in up-/downstream according to product definition (differentiation between assured and best effort), maximal bandwidth
- Security

- Access control for services
- Unification of controlled (e.g. IP address) or uncontrolled identifier (MAC) to avoid spoofing
- Policy based requirements, e.g. broadcast disabled; split hoirizon; message reglementation for control protocols

3.3 Additional use case specific requirements

In addition to general requirements, first use case area specific requirements are already identified.

3.3.1 Access Aggregation

From an implementation point of view at least IPv6 and different MPLS options like MPLS-TP and Seamless MPLS should be supported. Especially IS-IS, BGP and LDP must be supported. For the tree-like services source specific multicast should be support with IGMPv3 for IPv4 and MLDv2 for IPv6 in access and aggregation networks as well as PIM-SSM for core network segments. While using MPLS, mLDP should be supported.

3.3.2 Data Center

No specific requirements are identified so far. The discussion in WP2 is still ongoing and an updated of the specific requirements will be included in a later phase of the project.

3.3.3 Multi-Layer

No specific requirements are identified so far. The discussion in WP2 is still ongoing and an updated of the specific requirements will be included in a later phase of the project.

4 Open flow architecture

4.1 OpenFlow switch specification overview

The OpenFlow control protocol provides a vendor agnostic interface for controlling network forwarding elements. The essence of OpenFlow is the separation of the control plane and data plane in routing and switching gear. The interface allows flexible deployment of the network control plane software, and simplifies the control plane on network forwarding hardware. The control plane can be deployed on a centralized controller that controls multiple forwarding elements which allows increased innovation in control plane software and simplified operations and management (O&M).

In the canonical OpenFlow architecture, the control plane in network switching and routing equipment is moved into a separate controller. The controller communicates over a secure channel with the switches through the OpenFlow protocol (see Figure 5, from Open Flow Switch Specification version 1.0). Software running on the controller "programs" the switches with flow specifications that control the routes of packets through the network. For routing purposes, the switches only need to run an OpenFlow control plane, considerably simplifying their implementation. The architecture allows the controller to run on a separate PC and control multiple switches, rather than having a distributed control plane with components that run on each switch.

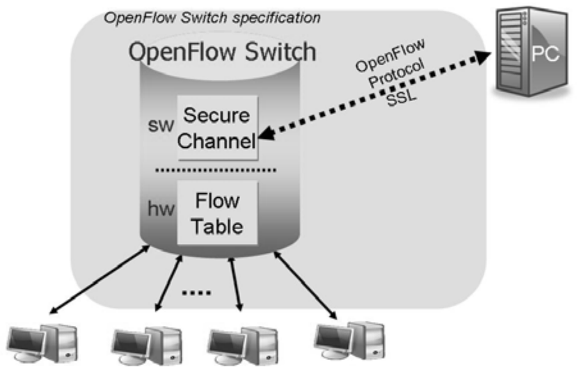


Figure 5: OpenFlow Routing Architecture

Switches are modeled as a flow table in which there are three columns: rules, actions, and counters. The rules column defines the flow. Rules are matched against the headers of incoming packets. If a rule matches, the actions from the action column are applied to the packet and the counters in the counter column are updated. If a packet matches multiple rules, the rule with the highest priority is applied. Each rule consists of elements from a twelve-tuple of header fields (see Figure 6, from OpenFlow Switch Specification version 1.0) or a wild card ANY. The set of possible actions are: forward as if OpenFlow were not present (usually utilizing the Ethernet spanning tree route), forward to the controller, forward out a specific port, and modify various header fields (e.g. rewrite MAC address, etc.).

In Port	Ethernet			VLAN		IP				TCP/UDP port	
	SA	DA	Type	id	priority	SA	DA	proto	ToS	src	dst

Figure 6: Twelve-tuple for Rule Matching

OpenFlow 1.0 is primarily designed for enterprise networks and lacks features, such as MPLS, necessary for deployment in operator networks. The goal for OpenFlow 1.1 is to support features that would allow OpenFlow to be deployed in wide area (i.e. operator) networks. The details of the specification are still in flux as this is being written, but the basic feature set has been determined and the plan is for the specification to be complete by the end of 2010.

The new features planned for OpenFlow 1.1 are the following:

- Port groups for multiple output ports that can support multicast, multi-path load balancing (e.g. ECMP), and other scenarios where a flow needs to be replicated or divided over multiple outputs.
- Multiple tables for reducing the complexity of supporting multiple sets of flow classification criteria, for example supporting QoS and forwarding decisions.
- Uniform support within table format, rule definition, and rule matching for tags. This includes both VLAN and MPLS. In OpenFlow 1.0, only VLAN tags are supported. Actions have been added to allow MPLS label processing as a consequence of rule matching.
- Additional support for QoS through differential queuing. OpenFlow 1.0 already has flow discrimination based on IP DSCP bits and VLAN PDCP.
- Miscellaneous cleanup items, such as properly handling IP TTL processing (necessary for properly implementing IP routing).

Note that the OpenFlow 1.1 specification only specifies features in the hardware abstraction layer (HAL) of the switch and in the protocol connecting the switch to the controller. Control plane support for these features, for example control protocols for MPLS label distribution, is still needed on the controller side.

4.2 Ericsson MPLS extensions

An important scalability feature for engineering operator networks is flow aggregation. Flow aggregation reduces the size of forwarding tables and allows more flexible traffic engineering. OpenFlow supports two methods of flow aggregation: wildcarding and VLANs. Wildcarding allows any of the packet header fields in the OpenFlow flow-matching rule to be specified as a wildcard, thereby aggregating the matching flows since all the flows receive the same forwarding treatment. VLAN tags are included in the set of matched header fields allowing flow aggregation to particular VLANs. Multiprotocol Label Switching (MPLS [2]) is widely used in operator networks, including in conjunction with optical circuit switching. MPLS allows coarse grained aggregation of flows near the network edge, simplifying (or eliminating in the case of packet to circuit transition) forwarding tables in the core.

MPLS forms flow aggregations by modifying the packet header to include a label. The label identifies the packet as a member of a forwarding equivalence class (FEC). A FEC is an aggregated group of flows that all receive the same forwarding treatment.

A data plane MPLS node implements three header modification operations:

- Push: Push a new label onto the MPLS label stack, or, if there is no stack currently, insert a label to form a new stack,
- Pop: Pop the top label off the MPLS label stack,
- Swap: Swap the top label on the stack for a new label.

The MPLS label stack is inserted between the IP and MAC (Layer 3 and Layer 2) headers in the packet. MPLS label stack entries consist of 32 bits, 20 of which form the actual label used in forwarding. The other bits indicate QoS treatment, top of stack, and time to live.

The first modification required to OpenFlow is to increase the size of the tuple used for flow identification. In principle, the size of the MPLS label stack has no upper bound, but as a practical matter, most carrier transport networks use a maximum of two labels: one label defining a service (such as VPN) and one label defining a transport tunnel[1]. We therefore decided to extend the header tuple used for flow matching from 12 fields to 14. Only the actual 20 bit forwarding label is matched, the other bits are not included. Figure 7 shows the 14 tuple.

In Port	Ethernet			VLAN		MPLS		IP				TCP/UDP port	
	SA	DA	Type	id	priority	Label1	Label2	SA	DA	proto	ToS	src	dst

Figure 7: Fourteen-tuple for MPLS rules

With the exception of limited field rewriting, OpenFlow 1.0 actions perform simple forwarding, while the MPLS actions manipulate the header differently than other OpenFlow 1.0 action. Rather than inserting the MPLS protocol actions into the basic OpenFlow packet processing pipeline, the actions instead were isolated

[1] Future MPLS-TP carrier networks may use more labels.

using an abstraction called a virtual port. A virtual port can do arbitrary packet rewriting, or perform actions that require a state machine. Other protocols that manipulate headers in complex ways, such as tunnel protocols like GRE or IP-IP, can also be implemented using virtual ports. In addition, a few simple MPLS-specific actions were added directly to the flow table processing: drop a packet if the TTL is expired and increment a counter indicating number of expired TTL dropped packets. These actions, though not in the basic OpenFlow 1.0 set, are generally useful since they are also necessary for IP routing.

Virtual ports can be hierarchically stacked to form processing chains on either input or output. On output, virtual ports can be included in flow table actions just like physical ports. Virtual ports are grouped together with physical ports into a virtual port table. Figure 8 illustrates the virtual port table, together with a table row. Each virtual port table row contains entries for the port number, the parent port, the actions to be performed by the virtual port, and statistics.

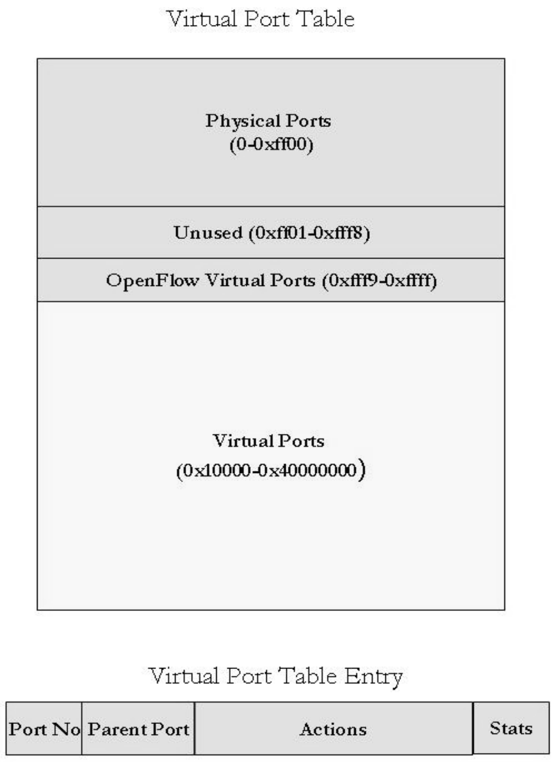

Virtual Port Table
Physical Ports (0-0xff00)
Unused (0xff01-0xfff8)
OpenFlow Virtual Ports (0xfff9-0xffff)
Virtual Ports (0x10000-0x40000000)
Virtual Port Table Entry
Port No
Parent Port
Actions
Stats

Figure 8: Virtual Port Table and Virtual Port Table Entry

The MPLS actions in the virtual port table consist of the following:

- push_mpls: Push a 32 bit label on the top of the MPLS label stack , decrement TTL and copy the TTL and QoS bits from the IP header or previous MPLS label,
- pop_mpls: Pop the top label on the MPLS stack, decrement TTL and copy the TTL and QoS bits to the IP header or previous MPLS label,
- swap_mpls: Swap the 20 bit forwarding label on top of the MPLS stack and decrement the TTL.

The OpenFlow protocol was extended with the following messages to allow the controller to program label switched paths (LSPs) into the switches:

- vport_mod: Add or remove a virtual port number. Parameters are the parent port number, the virtual port number, and an array of virtual port actions,
- vport_table_stats: Return statistics for the virtual port table. The statistics include maximum virtual ports supported by the switch, number of virtual ports in use, and the lookup count, port match count, and chain match count.

- port_stats: The OpenFlow port_stats message applies to virtual ports as well, but only the tx_bytes and tx_packets fields are used.

Finally, the OpenFlow switch_features_reply message was modified to include a bit indicating whether the switch supports virtual ports.

4.3 Limited Carrier Grade Support in OpenFlow

In this section, we discuss the possibilities of using OpenFlow to support carrier grade networks and the gap between the carrier grade network requirements and the OpenFlow carrier grade support. In addition to "standard" carrier grade requirements, additional challenges based on the notion of infrastructure virtualization have to be taken into account in the context of SPARC. Infrastructure virtualization is a very appealing concept for guarantying flexibility and high utilization of the network infrastructure in the context of infrastructure sharing (e.g. extended carriers-carrier services in an access-/aggregation-domain).

In general, OpenFlow does not support most of the Carrier Grade requirements defined in 3.2.

Some features discussed during the OpenFlow standardization that may be useful for carrier networks are, as of this writing, not anticipated to be part of the OpenFlow 1.1 standard. The reason is that the technical difficulty of incorporating multiple tables, and the amount of discussion necessary to agree on a standard, were so great that very little time was available for discussing these issues. They are:

- OpenFlow 1.1 distinguishes between different encapsulation types. Tags (MPLS and VLAN) are only valid over a single link while tunnels (IP-IP, GRE, GTP, etc.) are valid end to end. OpenFlow 1.1 will provide standardized support for tags but not for tunnels.
- Standardized support for virtual ports, which is necessary for implementing tunnels, will also not be provided. Virtual ports require the ability to define arbitrary packet rewriting within the actions and this requires that the code for such rewriting be on the switch (or, alternatively, that hardware for implementing the rewriting), and some way to configure the action, which is not available in OpenFlow 1.1.
- A generic approach such as Virtual Port that allows extension of flow definitions as well as actions without having to redefine the basic flow structure in an incompatible way. Virtual ports require some kind of hard state configuration mechanism, since the ports must be confirmed in place before any forwarding can be done, and there is also not anticipated to be a standard for this in OpenFlow 1.1. Another option for some cases would be to dynamically create Virtual Ports, or dynamically configure the Virtual Port behavior.
- OpenFlow 1.1 will not contain support for IPv6.
- OpenFlow 1.1 will not support pseudowire (RFC 4448). Processing of Ethernet frame headers stops when an MPLS header is detected.
- The "balance of system" protocol components – controller to switch transport layer, security, etc. – are likely to remain the same as in OpenFlow 1.0. Changes in these to support carrier class networks may be needed

It is anticipated that the extension mechanism will be utilized to prototype some of these features for inclusion in OpenFlow 1.2.

5 ForCES architectural framework

5.1 ForCES architecture overview

ForCES (Forwarding and Control Element Separation) defines an architectural framework on a network element level, specially, an IP network element level. A network element is composed of many logically separate entities but appear as an integrated network element to external entities.

There are two primary types of network element components: control-plane components and forwarding plane components. Control-plane components are usually based on general-purpose processor while forwarding plane components are usually based on specific hardware design. The purpose of ForCES is to provide a standard set of mechanisms for connecting these components, to enable increased scalability and to allow the control and forwarding planes to evolve independently.

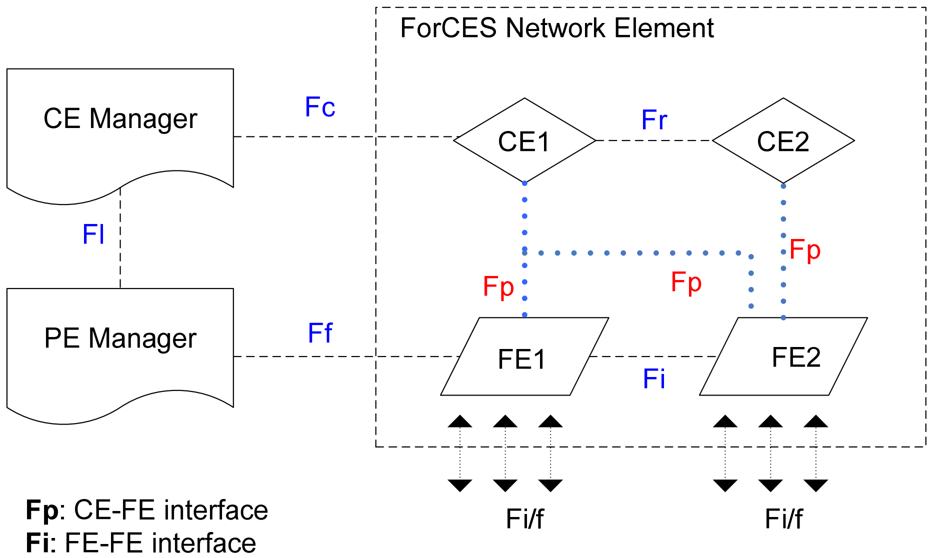
In ForCES, Forwarding Element (FE) is defined as a logical entity that uses the underlying hardware to provide per-packet processing and handling as directed/controlled by a Control Element via the ForCES protocol. Control Element (CE) is defined as a logical entity that uses ForCES protocol to instruct one or more Forwarding Elements (FEs) how to process packets. Control Elements (CEs) handle functionality such as the execution of control and signaling protocols.

The overview is organized in the following way: Firstly, the ForCES architecture framework (RFC 3746) is illustrated and ForCES protocol (RFC 5810) is introduced. We then define the ForCES requirements (RFC 3654), ForCES FE Model (RFC5812) and transport mapping layer (RFC5811).

ForCES Architecture Framework

ForCES architectural framework defines components of ForCES NEs which are composed of FEs and CEs, including the internal interfaces and external interfaces and several ancillary components. The logical components of ForCES and their relationships are shown in Figure 9.

This framework allows multiple instances of CEs and FEs inside one NE. Each FE contains one or more physical media interfaces for receiving and transmitting packets from/to the external world (The FE external interfaces are labeled as Fi/f).



Fp: CE-FE interface
Fi: FE-FE interface
Fr: CE-CE interface
Fc: Interface between the CE Manager and a CE
Ff: Interface between the FE Manager and an FE
Fl: Interface between the CE Manager and the FE Manager
Fi/f: FE external interface

Figure 9: ForCES architectural framework

For convenience, the logical interactions between these components are labeled by reference points Fp, Fc, Ff, Fr, Fl, and Fi.

All these reference points are important in understanding the ForCES architecture; however, the ForCES Protocol is only defined over one reference point -- Fp.

- CE and Fr Reference Point: the internal communication among CEs is out of scope in ForCES and considered an implementation issue; however, multiple CEs may be used for redundancy, load sharing, distributed control, or other purposes. ForCES will support mechanisms for CE redundancy or fail over.
- FE and Fi Reference Point: It is possible to partition one physical FE into multiple logical FEs. It is also possible for one FE to use multiple physical FEs. The partition and the internal communication of FEs are out of scope in ForCES and considered an implementation issue. FE is simple and dumb. Multiple FEs can be present in an NE and they can form different kinds of topologies such as mesh or ring.
- CE manager and FE manager: they provide the management function of NE, for example, which FEs a CE should control, the initialization sequence of FEs. The management functions are vendor specific and out of scope of ForCES.

ForCES Protocol

ForCES protocol only defines the communication between Control Elements and Forwarding Elements. This protocol does not apply to CE-to-CE communication, FE-to-FE communication, or how to manage FEs and CEs. The ForCES protocol is a master-slave protocol in which FEs are slaves and CEs are masters. This protocol includes both the management of the communication channel (e.g., connection establishment, heartbeats) and the control messages themselves.

In ForCES, the CE and FE configuration can be a chassis configuration: the control blades are the CEs while the router blades are the FEs, and the switch fabric backplane provides the physical interconnect for all the blades. The CE and EE configuration can also be extended to high speed LAN connection (Ethernet). In this case all CEs and FEs are physically separated boxes and communicate with each other by running ForCES. The goal of ForCES is to replace the proprietary interfaces between CEs and FEs with a standard protocol.

As ForCES is based on routers, the control functions and forwarding functions naturally are illustrated by router specific protocols. For example, the control functions in CE mainly include routing protocols (RIP, OSPF and BGP) and control/signaling protocols (RSVP and LDP); the forwarding functions in FE include LPM (longest prefix match) forwarder, classifiers, traffic shaper, meter, NAT (Network Address Translators), etc.

ForCES Architecture Requirements

The following is a highlighted list of ForCES architecture requirements.

- CE and FE can be connected by a variety of interconnect technologies (e.g Switch Fabric Backplane, Ethernet, ATM, etc)
- FE must support a minimal set of capabilities. ForCES can not restrict the capabilities that FEs may contain.
- A NE consisting of multiple CEs and FEs must support the appearance of a single functional device (e.g. TTL is only deducted by 1 if a packet transmits across this network element)
- A FE must support error monitoring and reporting.
- CEs must support CE redundancy or CE failover such as loss of connection between CE and FE. CE and FE must have ability to restore association and state resynchronization.
- FE must be able to redirect control packets to the CE.
- ForCES must support all of the router functions.
- CE must know the topology of FEs connected in the same NE.
- Scalability: NE must support at least hundreds of FEs and tens of thousands of Ports
- It must allow FE and CE to join and leave NEs dynamically
- It must support multiple CEs and FEs, although the coordination among FEs or CEs is out of scope.

ForCES FE Model

The ForCES FE model presents a formal way to define FE Logical Function Blocks (LFBs) using XML because XML has the advantage of being both human and machine readable with widely available tools support. An FE consists of a set of interconnected LFBs, where this interconnection of LFBs is described through an LFB Topology. LFB configuration components, capabilities, and associated events are defined when the LFB is formally created. Each LFB typically performs a single action such as classify packets or shape packets. The LFBs within the FE are accordingly controlled in a standardized way by the ForCES protocol.

An FE model is defined in Figure 10. All LFBs share the same ForCES protocol termination point. An LFB can have multiple components, inputs and outputs that are controlled by CE. In (P, M), P indicates a data packet while M indicates the metadata associated with a data packet.

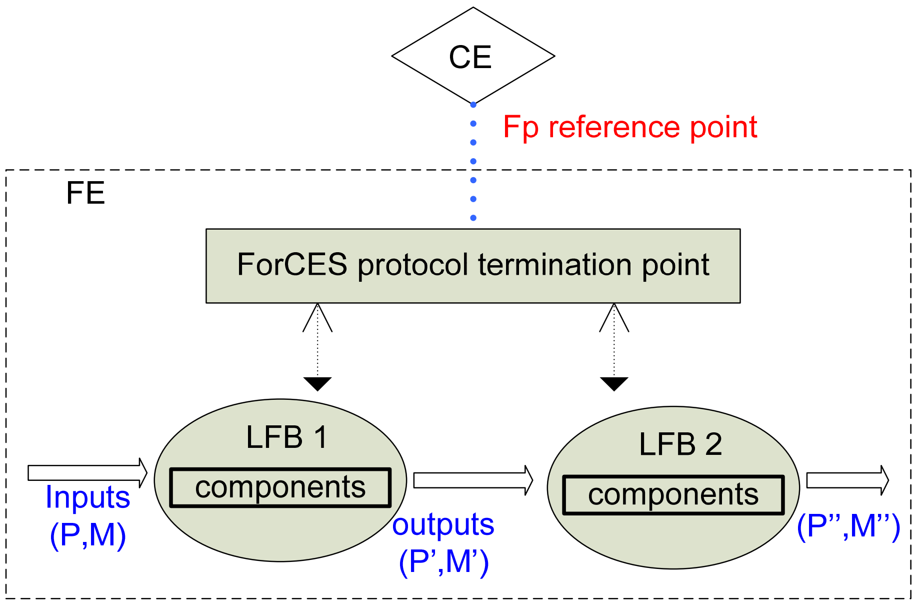


Figure 10: ForCES Forwarding Element model

Transport Mapping Layer

The TML transports the protocol messages and handles the issues such as reliability, congestion control, multicast, etc. All ForCES Protocol Layer implementations MUST be portable across all TMLs. The ForCES protocol layering constitutes two pieces, the PL (Protocol Layer) and TML (Transport Mapping Layer) as depicted in Figure 11. The PL is in charge of ForCES protocol semantics and message layout. The TML provides a hardware independent layer and enables ForCES protocol messages to be transmitted over any network technologies.

ForCES have defined an SCTP-based transport mapping layer to standardize the communication between the CE and the FE across the Fp interface.

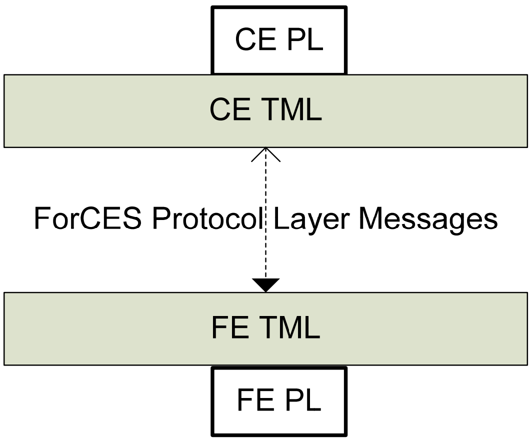


Figure 11: ForCES Transport Mapping Layer

5.2 Comparison ForCES and OF models

The motivation of ForCES and OpenFlow is very similar: to have standard mechanism allowing CEs and FEs to become physically separated standard components. Separate components would allow component vendors to specialize in one component without having to become experts in all components. Standard protocol also allows the CEs and FEs from different component vendors to interoperate with each other and hence it becomes possible for system vendors to integrate together the CEs and FEs from different component suppliers.

ForCES model is originated from routers. It focuses on separation of forwarding functions from controlling functions. It does not define a new generic data plane function. ForCES model employs a top-down approach, which provides a comprehensive and generic description of the architecture but fails to attract commercial application, e.g. no mainstream router vendors have any motivation to adopt this model. When ForCES model is conceived, the concept of virtual machines, data center or cloud computing is not well adapted so the model does not cover those areas.

OF model is originated from bridges. It not only focuses on separation of forwarding functions from controlling functions, but also creates a generalized flow table forwarding function. OF's data forwarding function does not see any layers and it treats all incoming frames as simply a flow to be matched against the flow table. On the other hand, OF's controlling function does see different layers. Although OF started as a academic project purely based on Ethernet, the bottom-up approach and its timely associate with virtual machines, data center and cloud computing attract many vendors and operators.

OpenFlow is based on the same principle of architecture split between control and forwarding plane as is the basis for the ForCES architectural framework. ForCES and OpenFlow protocols both standardize the communication over a secure channel between CEs and FEs. Both models are missing carrier grade capabilities such as OAM.

However, there are also many differences. In short, ForCES defines a model which is both more generic and more comprehensive than OpenFlow

- There is a notion of a Network Element, which comprises one or more CEs, and where each controls a set of FEs. This NE can interconnect to other NEs in the network, which can either be another ForCES-compliant NE, or some legacy NE (e.g. a router).
- ForCES is more generically defined, both regarding the protocol between CE and FE, as well as the model of the FE.
- ForCES also defines management interfaces towards the CE and the FE.

More specifically, the following tries to give a list of differences between ForCES and OF models

Bridge vs. Router

When OpenFlow is designed, the target hardware is based on Bridges; while when ForCES is designed the target hardware is based on Routers. OpenFlow defines Forwarding Element (OpenFlow Switch) and the OpenFlow Protocol, which describes communication between Controller and OpenFlow Switches. ForCES define Forwarding Element model and ForCES Protocol, which describes communication between CEs and FEs.

The FE in OpenFlow is based on Flow tables and processes Ethernet Frames and MPLS packets. The FE in ForCES is based on traditional Router function Model and processes IP packets. OpenFlow switch is a new switch architecture based on flow tables. ForCES FE reuses the existing hardware to provide per-packet processing.

The CE in OpenFlow has not specified which control protocols to be supported. The CE in ForCES requires supporting all common routing protocols (RIP, OSPF and BGP) and control/signaling protocols (RSVP and LDP);

Different architecture topology

For OpenFlow, controller controls multiple switches, replacing a distributed control plane with components that run on each switch. OpenFlow creates a split architecture targeting at network level.

For ForCES, Controlling Elements controls multiple Forwarding Elements, which consist of a Network Element in the network. Although there may be multiple FEs in a single NE, ForCES only creates a split architecture targeting at a network device level.

Multiple FEs with different topology can exist in one NE in ForCES. There is no corresponding concept in OpenFlow.

Forwarding Element

OpenFlow has generalized the concept of "Flow tables" with rules, actions and counts to replace traditional forwarding database and relay functions. The forwarding function in OpenFlow is very different from traditional Bridge forwarding functions.

ForCES defines Forward Element Model with XML. It does not create a new forwarding model but only describes the traditional router forwarding functions as an information model to facilitate the information exchange between CEs and FEs.

Failure and Recovery

OpenFlow has not defined CE/FE redundancy and failover yet. It defines an emergency mode or default mode that will only keep the configured flow entries when connection between FEs and CEs is lost. In the actual implementation, the FEs usually keep all the flow entries until the lost connection is re-established.

ForCES supports the dynamic join or leave of CEs and FEs. It defines 3 modes when the failure occurs between FEs and CEs:

- Cold Standby: when failure occurs between FEs and CEs, FEs will forget everything and restart initiating process when the connection is re-established
- Graceful Restart: FE holds on to the old state for a certain amount of time when failure occurs between FEs and CEs until the connection is re-established
- CE hot-standby: The backup CE is synchronized with the master CE. If the master CE can not control FEs due to node failure or connection failure, the back CE will take over the role of master CE and re-establish connection with FEs as soon as possible

Secure Channel

OpenFlow uses Transport Layer Security (TLC, RFC 5246) as a secure channel between CEs and FEs. ForCES uses Stream Control Transmission Protocol (SCTP, RFC 4960) as a secure channel between CEs and FEs.

QoS

OpenFlow has limited QoS by using a simple queuing mechanism and collects QoS statistics from counters in the flow table entries.

ForCES delegates QoS to the underlying TML (transport mapping layer). Specifically, the QoS (e.g. congestion control) is provided by SCTP, which can provide TCP like ordered, reliable, connection-oriented, flow-controlled, congestion-controlled data exchange or UDP like unreliable, unordered data exchange.

Operation Phase

ForCES defines 3 different operation phases:

1. Pre-association phase: it is a time period to decide the scope of NEs (CEs and FEs). This phase is out of scope of ForCES protocol;
2. Post-association phase: it is a time period to configure FEs and CEs with information necessary to contact each other. The phase includes association establishment and steady-state communication.
3. Association Re-establishment: it is a time period to re-establish the association if the previous association is lost due to administrative command or failure.

OpenFlow has not defined any operation phases yet. It mainly focuses on Post-association phase in the current protocol.

Scalability

OpenFlow has not defined any scalability requirement yet.

ForCES has the MUST requirement for the ForCES protocol to support “at least hundreds of FEs and tens of thousands of ports”. ForCES does not have a requirement related to the performance of a NE as the number of FEs or ports in the NE grows.

5.3 Borrowing selected ForCES concepts in OF

ForCES and OpenFlow take distinct but complementary approaches to separate control plane from forwarding plane. ForCES is centered on a conceptual model of the forwarding processes, and the parameters that control the forwarding model. To do this, it deconstructs the forwarding process into Logical Function Blocks (LFB). Each of these is described using an XML document which identifies the inputs and outputs, the processing logic, and most importantly the parameters of the operation of the LFB in a way that allows for protocol reference to this information. ForCES also uses this mechanism to represent the interconnection of the LFBs, allowing the CE to construct a processing path and define its parameters. The strength and weakness of this approach is that you also need clear function definitions for each and every processing block you re going to control.

In contrast, OpenFlow is build around a very specific processing model, and is concerned with the dynamic creation of flow controlling entries in a flow lookup table. This table uses multiple tuple (12 tuple in OpenFlow 1.0 and 14 tuple in OpenFlow 1.1) for lookups, and therefore can apply extremely versatile processing logic based on Ethernet or IP parameters. The processing logic assumes that packets which do not match the table are sent to the controller for analysis, forwarding, and to trigger the creation of new forwarding table entries. This allows for an extremely efficient protocol to manipulate this aspect of the processing logic.

The following is a list of concepts that OpenFlow can borrow from ForCES model.

Architecture Framework

ForCES defined a nice architecture framework. OpenFlow can borrow most of the abstract concept such as CEs and FEs. For example, the OpenFlow controller is a CE and the OpenFlow switch is an FE. The OpenFlow protocol is mapped onto the Fp interface between the CE and the FE.

In addition, these two approaches can be seen as complementary in an architectural sense. OpenFlow defines a specific processing space, and ForCES describes a way of assembling and configuring processing units. Thus, ForCES can be seen as a potential answer to the question about how OpenFlow based systems will allow the Control Element to configure the aspects of processing which can not be easily characterized as actions based on one flow, such as the parameters of queues, or the enabling of low level protocol operations such as OAM directly on the forwarding device. The use of the ForCES LFB concept could help describe the pieces of an OpenFlow system.

Model Definition

ForCES defines Forward Element Model with XML as an information model to facilitate the information exchange between CEs and FEs. By doing so, ForCES separates the actual implementation of FE from the ForCES protocol and provides a better way to accommodate future function extension.

OpenFlow may develop an information model in the similar way to describe the OpenFlow Switch function requirement. For example, we may extend the ForCES FE model “Logical Function Blocks” to model OpenFlow “FlowTable”.

In particular, the existing OpenFlow processing can be described using the ForCES XML language, and properties such as limits on the table size can then be accessed using the identifying information from the model.

Transport Layer Mapping

ForCES divides the ForCES protocol layer from the underlying Transport Layer by using TML. It enables ForCES protocol to be used with various networking/backplane technologies. The current OpenFlow only supports IP network. OpenFlow may adopt the similar approach to separate the protocol layer from the underlying transport layer and get rid of the limitation of only IP network.

Failure Recovery between CE and FE

ForCES supports the dynamic join or leave of CEs and FEs. It defines 3 modes when the failure occurs between FEs and CEs. OpenFlow may enhance its failure recovery mechanism by studying the proposals in ForCES.

Management interface/system

ForCES has defined managed objects for the ForCES Network Element in RFC 5813. OpenFlow has done little in this area and can propose its own managed objects based on ForCES.

6 GMPLS, PCE and split transport control plane

Today's networks follow a layered abstraction model. At the lowest layers, there are long-lived structures requiring civil works, way-leaves and negotiations with third parties, which need long term planning. At the highest layers, there is near-instantaneous creation and destruction of dynamic service instances for many users. In between is the transport layer, which provides a smoothed aggregate view of these demands. The layers and their respective properties are illustrated in Figure 12. The behavior of the IP network is contained within much slower-moving "aggregate tunnels" in the transport layer. The existence of a layer of aggregate virtual pipes allows the deployment of bulk recovery and restoration techniques, which would otherwise involve the destruction and recreation of much fine-grained flow state. This contributes to fast and predictable recovery of transport networks.

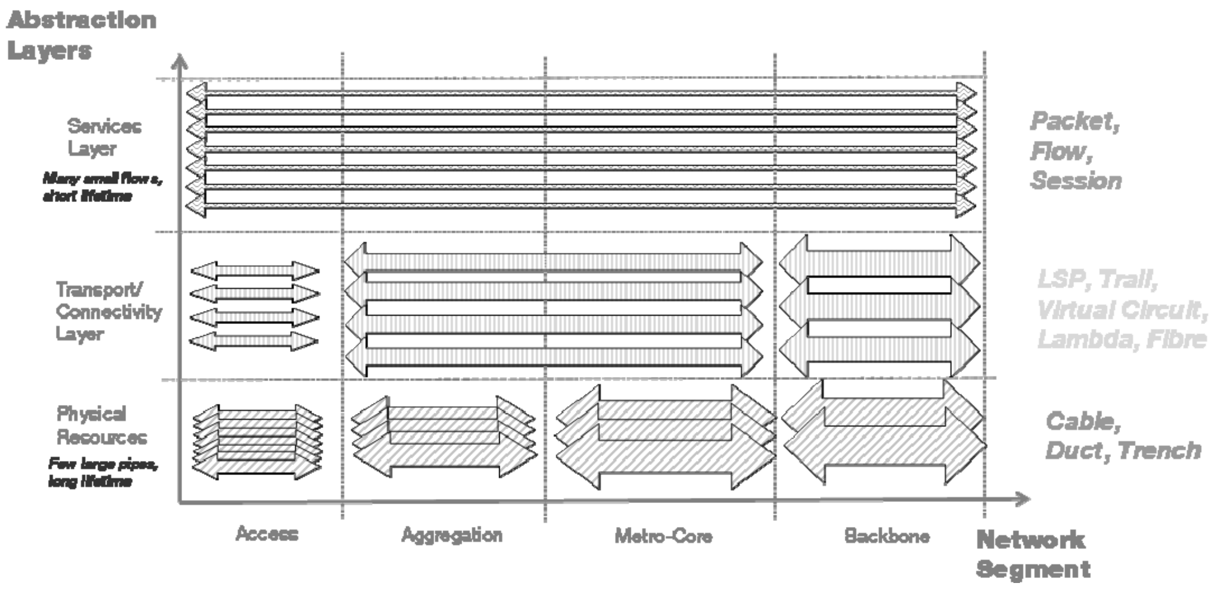


Figure 12: Network Abstraction Layers [GELS-COMMAG]

6.1 A control plane for the transport layer

Today, transport layers are mostly controlled by vendor-specific management systems. Such networks have achieved a high degree of stability and dependability. Nonetheless, there are disadvantages.

1. There are significant constraints on the mixing of vendor equipments, because of the consequent requirement to inter-work their management systems
2. Management systems cannot respond in real time. Hence management-based restoration of connectivity after failures involves significant downtime.
3. Manual database input is prone to error. Thus maximum use of auto-discovery is desirable.

There has long been discussion of the need for a standardized "transport control plane" to address these issues.

Generalized Multi-Protocol Label Switching (GMPLS) is evolving to fulfill this role. It extends the MPLS control plane to support many data plane technologies, from wavelengths in WDM networks to ATM virtual circuits. It is currently deployed in large SDH/OTN networks. The current justification of these deployments is to automate mesh network restoration, either stand-alone or together with configured protection.

6.2 The GMPLS control plane

In MPLS, control plane and data plane are tightly coupled. The structure, processing and location of the label in data packets are specific to the MPLS technology. Each IP packet is tagged with a 32 bit shim header which is used by MPLS routers to make forwarding decisions. There are two groups of MPLS signaling protocols; those concerned with Label Switched Path (LSP) signaling, and those delivering routing information. Two protocols have been defined to signal LSPs; Label Distribution Protocol (LDP), and the

Traffic Engineering extension of the Resource Reservation Protocol (RSVP-TE). Both the OSPF and the IS-IS routing protocols have been extended to advertise Traffic Engineering (TE) information.

GMPLS [IETF/CCAMP] extends the MPLS control plane to support various types of data plane: Packet Switching (PSC), Layer-2 Switching (L2SC), Time-Division Multiplexing (TDM), Wavelength (Lambda) Switching (LSC) and Fiber Switching (FSC) (Figure 13). It exploits the capability of the RSVP-TE and OSPF-TE or ISIS-TE protocols specified for MPLS. In order to generalize the label switching concept, the data plane and control plane need to be separated. The control plane is based on IP protocols over an IP control channel, and carries the information necessary to control LSPs in the underlying data plane. Transport technologies may not support the addition of explicit labels in the data plane. Nonetheless, they have characteristics similar to the label in MPLS forwarding. For instance, in optical cross-connects (OXCs) the logical identifiers of distinct fibers or lambdas within a fiber can be viewed as out-of-band labels. SONET/SDH has a multiplexing structure that is built on timeslots: here the level in the multiplexing tree can be interpreted as the label. In GMPLS these generalized labels are signaled in the control plane and used to configure the forwarding of data plane switching elements.

Since control and data plane are separated, GMPLS adds a new protocol: the Link Management Protocol (LMP). LMP is responsible for neighbor discovery, maintaining control channel connectivity, verifying data link connectivity, correlating link property information, suppressing downstream alarms and localizing link failures.

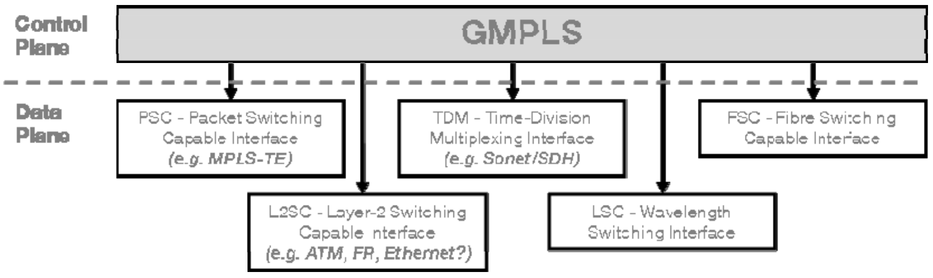

Control Plane
GMPLS
Data Plane
PSC - Packet Switching Capable Interface (e.g. MPLS-TE)
TDM - Time-Division Multiplexing Interface (e.g. Sonet/SDH)
FSC - Fibre Switching Capable Interface
L2SC - Layer-2 Switching Capable Interface (e.g. ATM, FR, Ethernet?)
LSC - Wavelength Switching Interface

Figure 13: The common GMPLS control plane [GELS-COMMAG]

Recently the GMPLS control plane has been extended with support for Path Computation Elements (PCEs) [IETF/PCE]. Constraint-based path computation is a fundamental building block for traffic engineering in transport networks. Path computation in large, multi-domain, multi-region, or multi-layer networks is complex and may require special computational components and cooperation between the path computation modules of different network domains.

A Path Computation Element (PCE) is an entity that is capable of computing the path of a TE LSP by operating on the traffic engineering database and of applying computational constraints during the computation. The PCE entity is an application that can be located within a network node or component, or on an out-of-network server. In the PCE architecture two basic entities are distinguished; PCEs that are responsible for the path computation service and Path Computation Clients (PCCs) that utilize the service of PCEs. The standardization work focuses on specifying the communication interface: the Path Computation Element Communication Protocol (PCEP) between these entities.

With GMPLS Control Plane and Data Plane are decoupled and this way the same control protocols and mechanisms can be applied for any transport technology. The PCE architecture takes this separation one step further by decoupling the “intelligence” from the control plane. Namely, path computation algorithms, policies and constraints can now be applied by a separate software component that can be accessed via a standardized interface, the PCEP protocol [3].

7 Network virtualization

7.1 Introduction

Virtualization is a well-known concept in engineering and information science. Network virtualization 49is an approach:

> ... whereby several network instances can co-exist on a common physical network infrastructure. The type of network virtualization needed is not to be confused with current technologies such as Virtual Private Networks (VPNs), which merely provide traffic isolation: full administrative control as well as potentially full customization of the virtual networks is also required. This also means that non-IP networks could run alongside the current Internet realized as one future virtual network. The point is that each of these virtual networks can be built according to different design criteria and operated as service-tailored networks.

The virtualization of an inter-network architecture like the Internet includes a variety of different individual resources. In general the resource could be split in three different groups:

- Network nodes
- Network links
- Other resources in the network

In the area of network nodes, the commonly included nodes are routers, last mile access and end user nodes. Even here different kinds of virtualization are possible e.g. whole systems, interfaces or forwarding data bases.

Network links have always been a scarce resource, especially in the wide area network domain. To share these resources and/or strictly isolate traffic on them multiple techniques have been implemented, for example VLANs in the IEEE802 family.

Networks are increasingly extended and utilized by other resources, which also can be virtualized. Most relevant are servers in data centers that are providing application services to end customers and optical layer switches such as optical cross-connects. In addition, research is investigating other kinds of resource usage like content centric networking.

For virtualization, a set of three important requirements is commonly defined [4]. The first on is isolation, where concurrently existing virtual networks must be prevented from influencing each other adversely. This includes virtual nodes running on the same host as well as virtual links sharing the same substrate link. For a virtual link, the given SLA – whether statistical multiplexing is applied or not – has to be fulfilled in the presence of concurrent virtual links. The second on is efficiency meaning that the overhead of network virtualization has to be low and should not be visible inside the virtual network, the difference between a virtual network and a physical network should be as close to zero as possible. The third one is the need for identical behavior. Seen from the inside, a virtual network consisting of virtual nodes and links should behave exactly as a physical network built from corresponding real nodes and links.

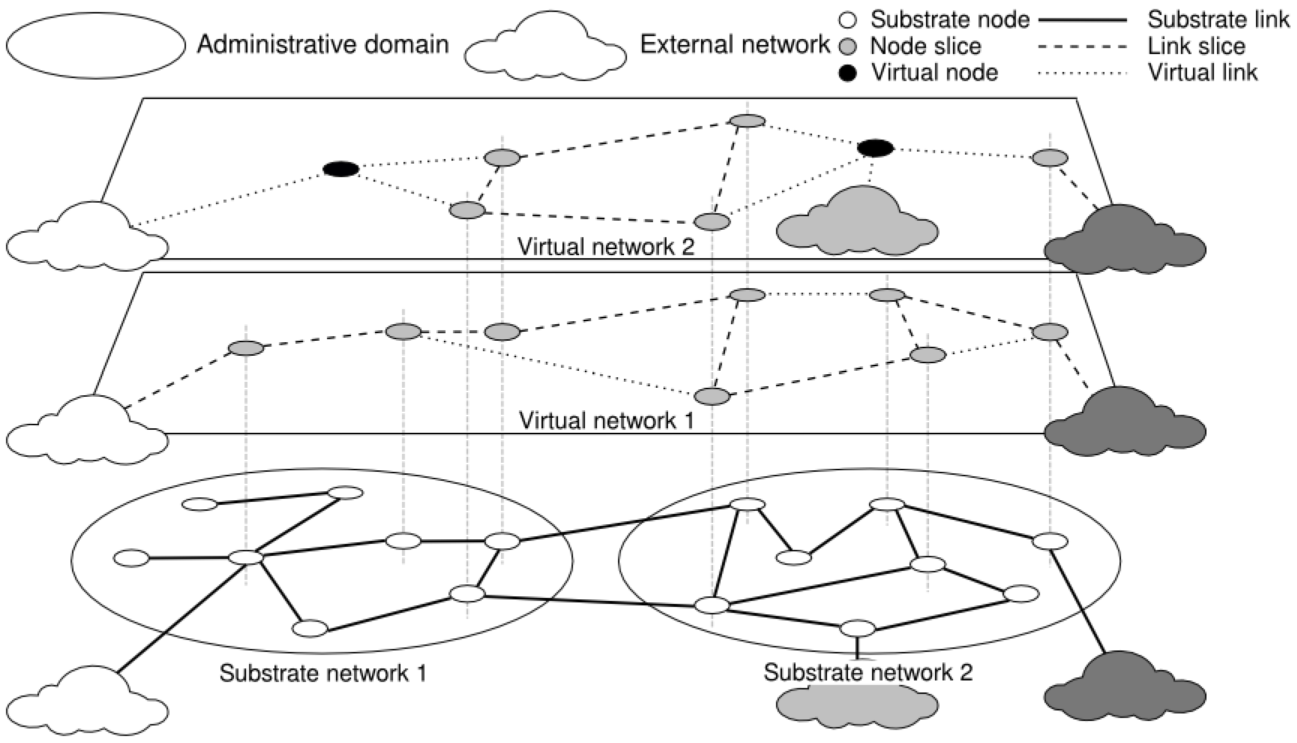


Figure 14: Two Virtual Networks on top of two administratively separate substrate networks using network slices, virtual nodes, and virtual links.

When defining the resources to be virtualized one should distinguish between the *physical* resources (also known as the substrate), *sliced* resources, and *virtual* resources. The substrate consists of the physical nodes and the physical links, for example physical switches, routers, or physical Ethernet links. By subdividing (or *slicing*) the substrate resources one creates *node slices* and *link slices*. The combination of node and link slices results in a *network slice*. It is important to note that the topology of a network slice is strictly limited to a subset of the substrate network topology.

Virtual resources however are not limited by the substrate topology; they are links and nodes that instantiated on top of the substrate and can be used to create topologies that e.g. can be handled more efficiently by routing protocols. A virtual link can be implemented by using a tunneling technology to create a link between two nodes that are not connected by a substrate link. Virtual nodes can be created using one of the well known computer virtualization techniques (see section 7.3). A *virtual network* may contain a number of virtual nodes and/or links as well as network slices (as illustrated in Figure 14).

Slicing the substrate, instantiating virtual resources, and managing the results requires a control system called *virtual node control*. Using the control system an operator can manage the creation, modification, and destruction of virtual network. The virtual node control system may provide other services as well, for example a configuration interface to a running node via a console or other management systems.

For the successful implementation of multiple virtual networks in a substrate certain aspects have to be managed like address identifiers or different markings in protocol headers, which are not used consistently among different operators. Another important feature is programmability, the ability to change or introduce new software to support e.g. new protocols without replacing the substrate.

7.2 Design goals for Network Virtualization systems

In the paper [[5]] the authors' summaries the current state of research in Network Virtualization and present a number of collected architectural principles and design goals that should be fulfilled by the Network Virtualization system. Some have already been mentioned but here is a complete summary: These are summarized here:

- **Coexistence** Multiple VNs should be able to coexist and span multiple domains.
- **Recursion** A guest VN should be able to act as a host VN.

- **Inheritance** A VN should be able to inherit attributes from its parents, for example bandwidth constraints.
- **Revisitation** A physical or virtual node should be able to host multiple guest nodes, both in different VNs and within one VN.
- **Flexibility** A VN should not be constrained in its topology, its forwarding mechanism or in its control protocols.
- **Manageability** It should be possible to manage a VN without the need for coordination across administrative bounds.
- **Scalability** There should not be any fundamental limits on the scaling of the amount of guest VNs.
- **Isolation** VNs should be completely isolated from each other in all aspects. Misconfiguration in one VN should not in any way affect coexisting VNs.
- **Stability and Convergence** VNs should not only be isolated from errors in other VNs but also from errors in the host network. If an error in the host network still affects VNs they should converge back into a working state automatically.
- **Programmability** The virtual nodes should be programmable in order to support flexibility and manageability. Programmability should be secure, fast, and possible in all planes (data-, control-, etc).
- **Heterogeneity** The solution should not be tied to any particular networking technology either in the host network or outside the virtual network.
- **Legacy support** The solution should support interactions with legacy networks.

While these are very high goals they are useful as an “gold standard” that actual solutions or proposals can be compared with. For actual solutions many of these goals may be unnecessarily high, for example Recursion is perhaps not realistically achievable or even useful for more than a few sub-VNs.

Programmability

Network programmability means that multi-purpose bare hardware can be programmed at any time into a specific product addressing specific (single or combination of) purpose(s) that may change over time. For example, today very few products can act as PBB-TE or (G)ELS switch [6], although they are both a kind of label switch (functionally similar as an MPLS switch) where the labels are encoded using only Ethernet frames: network programmability avoids these product specific limitations.

A split architecture where external controllers can control/configure functional elements or bare hardware is already a huge step in the direction of network programmability. However, network programmability at its full potential should also be capable of enabling the programming of functionalities into general purpose hardware. For example, a network element based on the CLICK modular router would require a protocol interface that allows external controllers to load the network element with any CLICK element and to specify how these CLICK elements should be linked together. Although CLICK modular routers are software based and thus not capable of delivering the highest performance, approaches as in the NetFPGA initiative have that potential as the network ports connect to a large FPGA that can be reprogrammed with any bitware as long as it fits in the FPGA footprint.

As discussed above, both the network virtualization and network programmability concepts are important tools in reducing costs. A mix of both results in an even higher cost saving potential.

Figure 15 illustrates how this can be achieved. At the left, the programmable network element (e.g., an OpenFlow switch) is shown. In the middle, a controller proxy (in case of OpenFlow this is called a FlowVisor) guarantees that controllers belonging to one network slice cannot control/affect/access the virtual node from another slice. At the right side, a controller per slice is shown that controls the part of the hardware/network element belonging to that slice through the controller proxy.

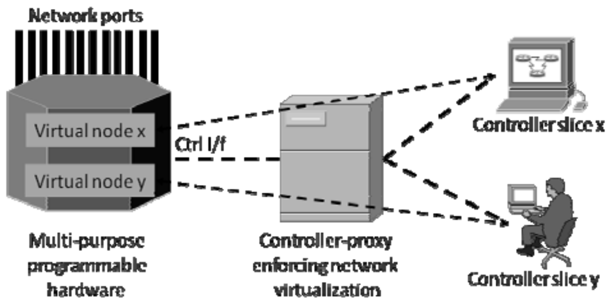


Figure 15: Programmable network slices

7.3 Existing Virtualization Methods

Since the concept of virtualization is such a broad one a more detailed look at the different established technologies for performing link- and node virtualization and what they provide is useful in order to understand the difference between them and the OpenFlow approach.

Link virtualization

There are several ways of virtualizing links on practically all network layers. The different techniques can be classified into two groups, those that allow Recursion and those that do not (i.e. those capable of creating Virtual Links and those that only slice links). Examples of techniques that do not allow Recursion is IEEE 802.1Q C-VLANs and WDM, these can be used to separate traffic over one physical link but do not allow further separation within the sliced link. It is not possible to sub-divide a lambda without resorting to other technologies. MPLS[2]-, GRE[7]-, and PBB-TE[8] tunnels are technologies that in theory allow infinite sub-division, in practice the recursion depth is limited by e.g. the underlying Maximum Transmission Unit (e.g. in the case of PBB-TE the standard MTU limits recursion depth to 22 stacks).

Comparing available technologies with the gold standard described in section 6.2 one may conclude that the technologies that allow recursive tunnels and supports advanced QoS (i.e. per level of recursion) for isolation are the closest match to the ideal, for example MPLS tunnels.

Node virtualization

There are numerous existing techniques for either slicing or virtualizing nodes, they usually involve either:

- Modifying the control- and forwarding mechanism and keeping multiple instances of Forwarding Information Base (FIBs) and Routing Information Bases (RIBs) as well as other necessary data structures.
- Virtualizing control- and forwarding by running multiple instances of the relevant software such as the operating system, routing protocol processes etc.

An example of modification of the control- and forwarding protocols is VLANs in Ethernet [9] (compared to a non-VLAN enabled switch). When using VLANs to partition the physical network a modified version of the control protocol, the Spanning Tree Protocol (such as MSTP or PVST+) calculates an individual spanning tree group of VLANs or per VLAN. The Ethernet forwarding table, the MAC table, is modified to exist in multiple instances, each associated to a VLAN Identifier that represent the Virtual LANs.

A similar approach in Layer 3 is Virtual Routing and Forwarding (VRF). It is achieved by keeping multiple RIBs and FIBs, a set of interfaces associated with these as well as rules and routing protocols that are allowed to edit the tables. By using multiple tables packets arriving on for example different ports get their next hop from different tables and thus are routed differently. For example, VRFs are typically instantiated on a per-VPN basis in the Provider Edge (PE) routers in BGP/MPLS IP VPN provider networks [10]. An advantage of the independent VRF instances is that overlapping address ranges in the different VPNs don’t lead to conflicts.

In the optical layer, using optical cross-connects, the same approach would be to define virtual cross-connects as separate entities that each are only allow to cross-connect a subset of the physical ports or wavelengths.

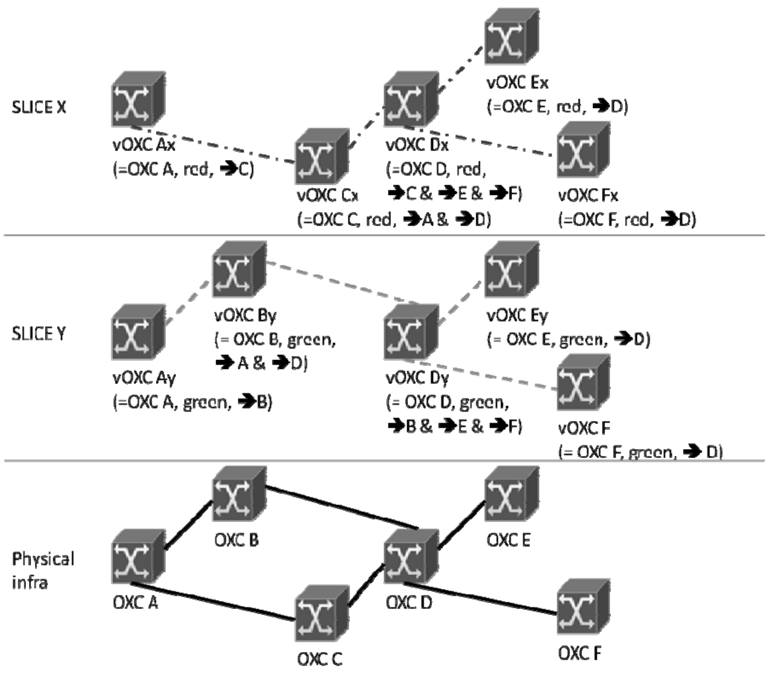

SLICE X
vOXC Ax (=OXC A, red, ➔C)
vOXC Cx (=OXC C, red, ➔A & ➔D)
vOXC Dx (=OXC D, red, ➔C & ➔E & ➔F)
vOXC Ex (=OXC E, red, ➔D)
vOXC Fx (=OXC F, red, ➔D)
SLICE Y
vOXC Ay (=OXC A, green, ➔B)
vOXC By (= OXC B, green, ➔A & ➔D)
vOXC Dy (= OXC D, green, ➔B & ➔E & ➔F)
vOXC Ey (= OXC E, green, ➔D)
vOXC F (= OXC F, green, ➔ D)
Physical infra
OXC A
OXC B
OXC C
OXC D
OXC E
OXC F

Figure 16: Creating slices in OTN

The other virtualization approach, actually running multiple instances of software, has a large range of implementations. Depending on the degree of virtualization support one would like to provide the functionality may range from the minimum necessary to give the impression of a virtual node to all of the functionality available in the physical node. The virtualization system may be implemented in software only or software with hardware assistance or completely in hardware. Cisco refers to these implementations as Software-Isolated Virtualization and Hardware-Isolated Virtualization [11].

The techniques for implementing Software-Isolated Virtualized Routers are analogous to the different kinds of virtualization available in the realm of computer virtualization where several modes of virtualization exist:

- **Full virtualization:** Virtualization of the whole computer, devices such as network cards etc are emulated by the host system. The host system instantiates guest systems that are not "aware" of the fact that they are running in an emulated environment (e.g. VMWare, KVM).
- **Para-virtualization:** Virtualization of the operating system. The guest and host operating system are both "aware" of the virtualization and take advantage of this by not fully emulating all hardware, instead the systems use specialized drivers that gives higher performance than the full virtualization approach. The tradeoff here is less flexibility since the guest system has to be modified (e.g. XEN).
- **Contextualization Virtualization of processes**: Normally processes are isolated into separated virtual memory segments but can still interact with each other on many levels, for example through the file system. With contextualization a process, or a group of processes, can be prevented from interacting with other process groups, making them "unaware" of other processes running on the system. This method provides the best performance (essentially zero performance penalties [12]) but is also the least flexible (e.g. Linux-VServer, LXC).

All these approaches to computer virtualization can be applied in network elements as well, from virtualizing the entire networking element to virtualizing the operating system to running groups of processes in different contexts.

Hardware-Isolated Virtualized Routers typically involves a complete separation of line and control cards, more or less equal to having two physical routers in a single chassis. In some sense this is virtualization since the power supply and chassis is shared.

Again comparing these techniques with the ideal [section 7.2] the conclusion is that some kind of SVR technique is the closest match. In theory they allow infinite recursion (although not possible in practice) and decent isolation. With software based virtualization techniques there are still many isolation issues which could has serious security implications (for example cross-VM information leakage), however this area is likely to improve due to the growing use of computer virtualization in the industry.

Some of the generic CPU virtualization techniques are probably already in use in some available equipment (e.g. to implement logical routers in Juniper routers), however how to virtualized and strictly isolate access to other shared resources in a router such as hardware implemented FIBs remain, to our knowledge, open questions. It is also a question of balance between strictness of the isolation and system performance.

7.4 Network Virtualization using OpenFlow

FlowVisor – OpenFlow proxying

One solution for network virtualization on top of OpenFlow is FlowVisor [13]. A FlowVisor acts as a proxy between Switches and Controllers, i.e. it connects to OpenFlow-enabled switches using the OpenFlow protocol and is itself in turn connected to by Controllers using the same protocol. FlowVisor does not allow the creation of ideal Virtual Networks, for example programmable virtual nodes and virtual links cannot be created. Rather it allows a separation of the substrate network into partitions called slices based on the different match fields available in the OpenFlow FlowTable. For example it can isolate all control of an IP subnet on some specific switch ports to a certain Controller. Since FlowVisors are transparent proxys they may be cascaded as in Figure 17, where one FlowVisor could be slicing the control of the whole VLAN range into two and another FlowVisor further sub-dividing it. In this case Controller 1 would be in charge of all traffic in half the total VLAN range whereas Controller 2 and 3 would control one fourth each.

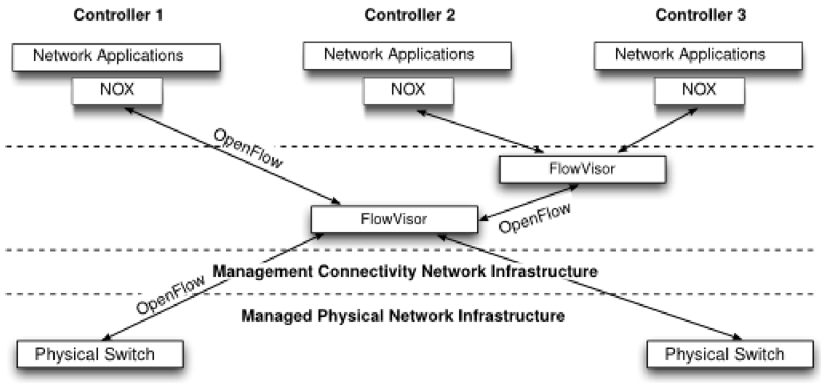


Figure 17: Network slicing using FlowVisor

How slices are partitioned is controlled by a policy database in each FlowVisor, essentially a text file that specifies which resources belong to a slice and which Controller the slice belongs to. The slices are not allowed to overlap (unless specified), these policies are enforced during the proxying process. When receiving a message from a Controller the message is inspected, rewritten, and verified against the policies and then sent to the appropriate Switch. If a Controller message violates the policies an error message is sent in reply. Similarly the messages sent from switches are inspected and sent to the appropriate Controller(s).

While not providing “true” network virtualization FlowVisor is a powerful tool that likely can be applied for the use-cases we consider. Some of the major functionality missing in is:

Virtual Links Due to the lack of tunneling functionality in OpenFlow 1.0 it is not possible to create virtual links; slices can only be defined as a subset of the physical topology. This problem can likely be solved by through tunneling, for example using the Ericsson MPLS extensions or other features that may be available in OpenFlow 1.1.

Isolation issues While traffic can be kept separated, isolation in the sense of slices not being able to interfere with each other is lacking. Slices may interfere with each other in several places, for example on links by using to much bandwidth or in switches where they compete for CPU time, memory, and other resources such as OAM functions, packet encryption units etc. Ideally one would like to be able to allocate a certain amount of resources per slice; these resources should then be further sub-dividable and allocated to sub-slices. Temporary solutions for both bandwidth and CPU usage are suggested [13]. They suggest limiting bandwidth usage by using existing prioritization fields in VLAN tags (IEEE 802.1p) or in the IP header. They also suggest limiting CPU usage in switches and FlowVisors by rate-limiting the OpenFlow protocol message rate, however this solution does not handle isolation of other switch resources usage such as packet encryption. The temporary bandwidth isolation supports needs improvement so that Hierarchical QoS can be enforced, this problem might be solved by the more advanced QoS features in OpenFlow 1.1. How to isolate switch resources is still an open question. Another isolation issue is how to cleanly define the different VNs as there is no OpenFlow specific way to tag traffic with a “VN identifier”. Instead one has to use existing protocols, for example VLAN identifiers, to identify to which VN a packet belongs. However, if one uses existing protocols to provide the VN identifier there is a risk of collisions since the incoming traffic to a VN might already use the particular field. This could be temporarily solved by translating the colliding identifiers at the border nodes at the cost of using FlowTable entries and complicating management. However, a better solution should be found.

Programmability Limited programmability is supported by OpenFlow, primarily through the ability of redirect data packets to the Controller, use a Network Application to process them, and either changing the FlowTables or transmitting new packets through the switch. These operations may be very resource intensive depending on what type of processing is being done and in some cases simple not fast enough. Some processing simply is not possible to perform in this way and probably has to be done on the switch, either in dedicated hardware or using the switch CPU. However there is no way to dynamically add packet processing/originating functions to a Switch. This problem can be mitigated by the existing support for vendor extensions which allows vendors to statically add functionality to the protocol and Switches.

Other issues are more specific to the FlowVisor approach to virtualization. By acting as a transparent OpenFlow proxy that enforces policies on the message exchange it allows simple slicing of the network and compatibility with existing Controllers can use. However, since all communication is through the OpenFlow protocol there is no way to provide a higher abstraction layer that can be more flexible for defining and controlling the Virtual Network.

Onix – Distributed OpenFlow Controller

An alternative approach to FlowVisor is to present a different API that allows for an explicit definition of a Virtual Network that is instantiated through OpenFlow. Such a model could allow more transparent creation of VNs where problems such as coexistence of colliding address-spaces could be handled transparently through for example address translation. The OpenFlow Controller Onix that is in development primarily by Nicira and Google has gone that route [14]. As of writing not very much is known about Onix, what follows is our tentative understanding based on a single paper. The Onix controller consists of three major modules, a switch import/export module that communicates with the physical infrastructure through OpenFlow, a distribution module that distributes the network abstraction among multiple Onix controllers and finally the network abstraction itself, the Network Information Base (NIB) (see Figure 18).

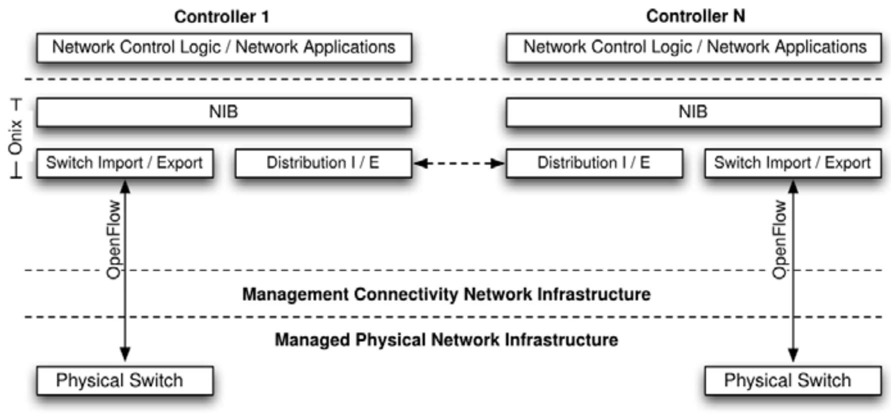

Controller 1
Controller N
Network Control Logic / Network Applications
Network Control Logic / Network Applications
Onix
NIB
NIB
Switch Import / Export
Distribution I / E
Distribution I / E
Switch Import / Export
OpenFlow
OpenFlow
Management Connectivity Network Infrastructure
Managed Physical Network Infrastructure
Physical Switch
Physical Switch

Figure 18: The Onix Model

The Network Information Base is central to the design; it represents the underlying physical topology and the events ongoing in it and gives the Network Applications a view of what they are controlling. The Network Applications interact with the physical topology through an API that allows them to listen to events and modify the NIB; these modifications are propagated to the actual network by the switch import/export module. A simplified overview of what entities the NIB contains can be seen in figure X which does not show all the attributes that can be manipulated.

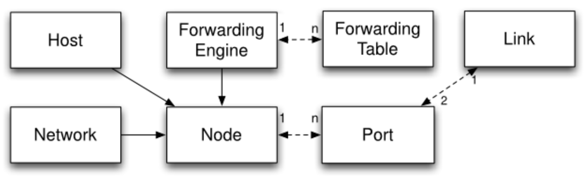


Figure 19: Entities in the Network Information Base

With multiple Onix controllers in the network the NIB has to be distributed to them, by default Onix has two methods of doing this, a Distributed Hash Table (DHT) and a transactional persistent database. The DHT does not guarantee consistency and is suited for volatile information with rapid changes whereas the traditional database does provide consistency but is slower. Through the NIB API a Network Application can configure how certain information is distributed, by for example configuring certain attributes to be distributed via the DHT.

In order to be able to scale there are two mechanism that can be used to limit the NIB size in an Onix controller, NIB partitioning and aggregation. When partitioning is used an Onix instance only keeps a subset of the NIB up-to-date and in memory, this not only reduces the amount of memory used but also the traffic to keep the NIB synchronized, it also hides any events occurring in the part of the NIB that is not kept. Aggregation allows one Onix instance to distribute a subset of its NIB as a single aggregate node which allows it to hide all internal state and events while distributing relevant information to other controllers. In a similar fashion the NIB can also contain the representation of Virtual Nodes and Links.

Onix shares most of the virtualization issues with FlowVisor since they both use OpenFlow which is where the problems stem from. However with Onix for example Virtual Links would be easier to solve if OpenFlow supports it since one has decoupled the switch control protocol from the network representation, making it much more flexible. With Onix a Virtual Link would be created through the NIB and represented as any other link in the network, with FlowVisor there is no link representation, rather one would police who is allowed to create certain Virtual Links.

However since FlowVisor and Onix operate on different levels one could imagine situations where both approaches could be applied, for example when different administrative domains are involved. In that case the operators of the substrate networks could use the FlowVisor approach to cut coarse network slices that are accessible through the simple OpenFlow interface. Multiple substrate network slices could in turn be concatenated by Onix to produce a coherent Virtual Network (similar to what is illustrated in Figure 14).

8 Hierarchy of controller and federation options

Hierarchy and federation are two basic mechanisms to improve scalability. By adding an additional layer of controllers, lower layer controllers will be responsible only for the control of a fraction of the network while they will also provide aggregate information to higher layer controllers which in turn will not need to handle all details of the entire network. This is similar to virtualization in that controllers provide an aggregate, virtual view of the underlying components and network.

Federation is another tool that can improve scaling by partitioning the network into domains which controllers exchange only aggregate information among themselves. Various protocols can be used for federation, e.g., IGPs, EGPs to distribute information, to setup connectivity RSVP-TE signaling and PCEP.

These two approaches can also be combined to fit different network scenarios and scalability requirements.

When it comes to controller functionality, we need to differentiate two scenarios: 1) homogeneous and 2) heterogeneous controllers. In the first case controllers have the same functionality, e.g., responsibility for transport connectivity provisioning and maintenance, here hierarchy and federation is primarily used for scaling. In the latter case, there are different controllers in the network with different responsibilities, e.g., one module for subscriber identification, another for configuring services, yet another to provision the required connections and set aside resources. In this case controllers rely on each other's functionality and besides scaling the emphasis is more on the functional distribution between controllers. In this case protocols between the controllers are selected based on the functional needs, e.g., Diameter, PCEP, etc.

The details of controller hierarchy and federation will be defined in subsequent documents for a selected use-case and following the specification of the single controller case.

9 Multi-layer control

9.1 Introduction

Today's networks feature many more layers than the tradition OSI/ISO model foresees. In fact, while OSI introduces a layering of *functions* required for networking, each layer in a telecommunication network is corresponding to a different *technology* as in IP/MPLS/SDH/WDM networks. We observe that each technology layer can be separated into a control and data plane, but that there is not necessarily a 1:1 map between the two.

9.2 Unified control plane

The *functions* that are executed within the *technology layers* are often repeating. This way, routing for example is done multiple times in the stack, starting from the dimensioning of a physical topology to the RWA (Routing and Wavelength Assignment) in optical networks to Spanning trees to Interior Gateway protocols up to application layer routing as it is done between data centers. It is understood that the task of harmonizing multiple routings gets more and more complex with the number of layers involved.

The necessity to reduce the number of layers in modern networks has led to the idea of a unified control plane. Next-generation networks will however still consist of at least two network layers: i.e., a packet-switched (e.g., IP/MPLS) network layer over a circuit-switched optical transport network layer (OTN and WDM switched sub-layers), as depicted in **Figure 20**.

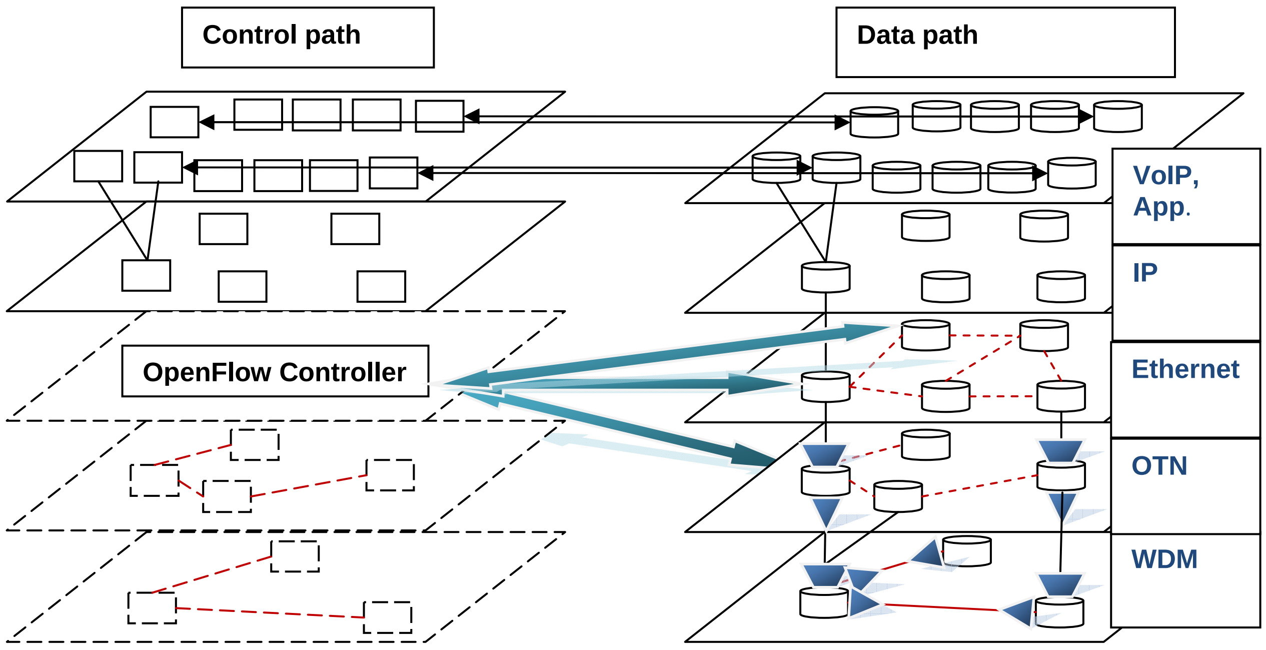


Figure 20: Layered network architecture. In the lower layers, adaptation and termination functions according to ITU G.805 are depicted. If lower layers in the data path shall be controlled by the OpenFlow controller, then the respective termination and adaption functions have to, as well.

9.3 Multilayer Traffic Engineering

Multilayer traffic engineering (MLTE) uses cross-layer optimization techniques to extend classic single layer TE (routing) with dynamic logical topology reconfiguration. Circuits provided over wavelengths in the OTN are set up and torn down dynamically in order to provide capacity in the packet switched layer (typically according to traffic demand). The optical layer could be composed by 1 or 2 switching sub-layers (OTN and WDM), thus, multi-layer TE has to deal with 2 or 3 layers according the specific network scenarios.

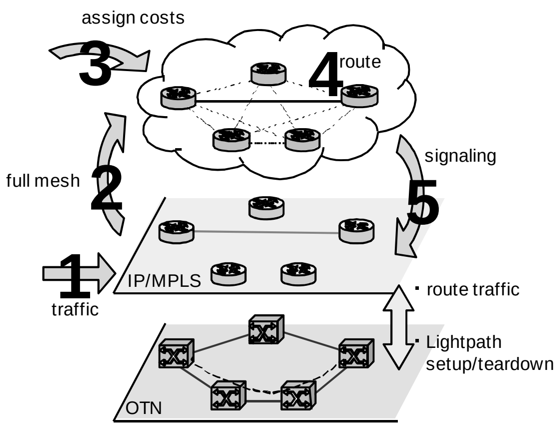


Figure 21: Multilayer traffic engineering example

The example in Figure 21 shows a MLTE strategy which routes traffic on a virtual full mesh first using a cost function based on current (measured) end-to-end traffic demand volume [15]. The cost function distributes flows on a minimum of links in the virtual full mesh; afterwards the rerouting (layer 3) and circuit setup/teardown is signaled using IP/MPLS and OTN layer control.

9.4 Benefits of OpenFlow for Multilayer Networks

Since OpenFlow is inherently flattening the network layering structure, it provides options to simplify optimizations and signaling for multilayer traffic engineering purposes and also multilayer recovery (See section 10.4). Multilayer recovery between layers typically works from the bottom up, where the server layer waits for the client layer to restore as much traffic as possible before starting its own recovery mechanisms (escalation strategies). In OpenFlow networks this could be simplified significantly and true multilayer recovery mechanisms may be easier to implement, as OpenFlow will not try to recover individual Ethernet or IP links but restore each affected flow in its entirety.

9.5 Layering and OpenFlow

OpenFlow flattens the layered architecture in that it is based on a single switching format – Ethernet. This however does not mean that control would necessarily be flat (i.e., a unified control plane) as well. There are different options of positioning functionality in the stack of both control and data path.

We split functionality first into control and data "plane" or "path":

Which function goes into the control stack?	Which functions are performed in the data path?
Routing	Forwarding
Name resolution	Header en/decapsulation (i.e., source/sink of traffic)
Signaling	Processing
	Error and flow control
	Creation of monitoring, recovery, etc frames (e.g., BFD)

We observe that many of these functions are present multiple times in the network stack, e.g., routing information is exchanged by IGP protocols like OSPF and by spanning tree protocols, plus potentially other routing protocols dedicated to the optical layer. Some authors extend this concept to recursive layering models (RNA/J.Touch, NIPCA/J.Day), but for the purpose here we consider OpenFlow, and that is, Ethernet, as the central layer, and the case that there are optical or TDM layers below it.

In general, wherever layers are stacked there are so-called adaptation and termination functions. ITU-T G.805 defines the adaptation function and the termination function. The adaptation function defines how data belonging to a client layer network is embedded into data of a server layer network. The termination function adds monitoring information to the server layer network connection, taking care of a reliable data transmission [19].

Transport networks below Ethernet have been demonstrated to be controllable by the OpenFlow controller NOX [16] as well [17]. It is interesting to see how the OpenFlow extensions correspond to MPLS extensions when evolving to GMPLS (see section 6) as it introduces interface specs like Lambda_switch_capable

This sense of Multi-Layer is however limited to a direct control of the lower layers through the upper layers. There is no traffic upwards from lower layers to the OpenFlow controller, because typically there is no visibility of the contents of a flow within an optical or SDH cross-connect (OXC or ROADM) or Digital Cross-Connect (DXC).

The more interesting aspect is the control of hybrid, i.e., edge nodes through OpenFlow. These nodes consist of conventional OF switches (left side) and a cross-connect. In between, Virtual Ports are introduced that implement a termination and adaptation function. The control of these Virtual Ports is an open question in the OpenFlow architecture. Some form of "configuration interface" is needed that assigns VCG groups, wavelengths, or SDH VC to these ports. Typically this information is *hard state*.

Extending the concept of virtual ports to a dynamic configuration would require the modification of the OpenFlow OF interface in a very technology-specific way. One may therefore advocate the introduction of a second – processing - interface that would take over the configuration of the termination and adaptation function and leave the internal forwarding to the OpenFlow interface (Figure 22).

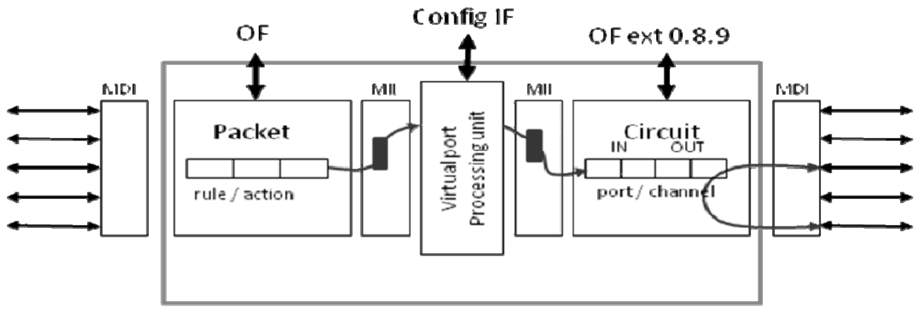


Figure 22: Multilayer OpenFlow switch. Virtual ports are assigned between the forwarding elements and a processing unit that implements the layer change, i.e. a termination and adaptation function.

10 Recovery and redundancy

10.1 Introduction

Communication networks are subject to a wide variety of unintentional failures, caused by natural disasters, wear out and overload, software bugs, human errors, and so on, as well as intentional interruptions due to maintenance [18]. Disruption of communications can have wide-scale effects, which is why recovery mechanisms are provisioned in these networks.

10.2 Failure detection in packet networks

In order to recover from a network failure one must first know that a failure has occurred, where it has occurred, which resource(s) it has effected, and if that resource is up for recovery. Failure detection can take place on many levels, for example through the physical loss of light or carrier. However some failures do not cause these errors, for example an error in the routing or forwarding process, these have to be detected on a higher layer. On the link- and network layer failure detection is usually performed through pro-active transmission of ”Hello packets”. In for example the routing protocol OSPF Hello packets play two roles, they are used both for neighbor discovery and failure detection and are periodically transmitted on all active links [20]. In the OSPF case the typical frequency for transmitting Hello packets is 10 seconds, with a dead-interval of 4 times that value. If a Hello packet is not received within the dead interval the neighbor is considered to be down, this in turn triggers a recalculation of the forwarding table. Depending on the values that have been configured for a number of timers in the routers and the topology of the network, total recovery (i.e. convergence from the fault may be in the order of minutes.

In enterprise networks a recovery time measured in minutes may be acceptable, TCP connections may even stay alive during the recovery process (depending on OS configuration) thus letting the failure and recovery pass without any major disruption of the network. In carrier networks such a long recovery time is likely unacceptable as there are SLAs that has to be met. In order to decrease the failure detection time more aggressive methods has to be used, both for the failure detection itself and the recovery process. A requirement for the maximum total recovery time is defined in ITU-T as well as for MPLS-TP [21] [22] to be ”sub-50ms”, meaning that a failure has to be detected and recovered from in less than 50 milliseconds.

The recovery process is typically replaced with pre-calculated protection paths that can be used immediately to circumvent the error, without requiring any recalculation. The pre-calculated paths may range from end-to-end to local, in the end-to-end case the path uses a completely disjoint path through the network thus avoiding any link or node that may have failed. In the local case (e.g. MPLS Fast Reroute) the recovery path may avoid just a single link, a single node or several nodes and links. The key is that regardless of end-to-end or local protection the paths are already created and can be used almost instantaneously (forwarding tables has to be updated).

The failure detection mechanism remains similar but with the packet transmission frequency increased and the dead interval lowered to give a failure detection time appropriate for the time it takes to perform the recovery process. There are minor variations on how failure detection is performed in different technologies, for example Bi-directional Forwarding Detection (BFD) [23] varies slightly from Ethernet Connectivity Fault Management (CFM, IEEE 802.1AG) [24].

Bi-directional Forwarding Detection is strictly end-to-end, the BFD version of ”Hello packets” are transmitted and received only by the endpoints in the BFD session, any intermediate node forwards the packets as any other data packet. In BFD the transmission interval is variable within a wide range, the dead-interval is a multiple of this timer, and the transmission interval may also be different in the different directions of the session. Another BFD feature is that these values may be renegotiated during the session, for example if one end-point is experiencing heavy load.

In Ethernet CFM the Hello packets, Continuity Check Messages, are multi-casted by all Maintenance End Points (MEPs) in an Maintenance Association (MA), and are processed by all Maintenance Intermediate Points (MIPs) and MEPs in that MA. The MIPs and MEPs keeps a record of all reachable nodes within a particular MA, if any of them times out an error has been detected. The transmission frequency and dead-interval is discrete in Ethernet CFM, there are 8 predefined transmission intervals from 3.3 ms to 10 minutes.

10.3 Failure detection and recovery using OpenFlow

There are several problems with implementing advanced, sub-50ms, failure detection and recovery using OpenFlow-enabled switches, both with the failure detection mechanism and the recovery mechanism.

The failure detection mechanisms require transmission of “Hello packets” on data links at a relativity high frequency and currently the only way of doing this is from the Controller through the Secure Channel. This not only “clogs” the Secure Channel with potentially thousands of Hello packets per second but also interferes with the detection mechanism as packets are not only traversing the link(s) (and possibly node(s)) that should be monitored but also the Secure Channel itself. Any jitter in the Secure Channel may cause packets to arrive too late and cause a false positive in the failure detection. Jitter in the Secure Channel is expected since it may be running in-band with the data traffic and, even worse, it may not even be functional due to the failure we want to detect.

Similarly the Secure Channel also interferes with the recovery mechanism as currently the only way to change the FlowTable is through the Secure Channel. To detect a failure and recover from it would require traversing the Secure Channel multiple times, at least once for detecting the fault and at least once for recovering from it through changing the paths by updating the FlowTable. This is assuming that the Controller still can reach the affected nodes through the Secure Channel.

Clearly the current OpenFlow mechanisms can not be to implement reliable sub-50 ms recovery. The obvious solution is moving the both the failure detection and recovery functionality into the Switches themselves. This solution highlights a few of the limitations of the current OpenFlow specification, on a high level the limitations in Network Programmability. Specifically the need for a mechanism to both originate and process packets locally on a Switch as well as mechanism for the Switch to modify its FlowTable without instructions from a Controller

10.4 Multi-layer recovery and restoration

10.4.1 Introduction

Typically, there is a client-server relationship between the adjacent layers of a multilayer network. Each of these layers may have its own (single-layer) recovery schemes. In a multilayer network scenario, a single failure in the server layer will typically induce a cascade of failures in the client layers. The failure in the server layer is referred to as the root failure; the corresponding failures in the higher client layer(s) are called secondary failures. This is illustrated in Figure 23 Figure 24, where the failure of link E-D in the server layer (e.g. a fiber cut in the OTN) gives rise to the simultaneous failures of three links (*a*-*d*, *e*-*c* and *e*-*d*) in the client layer (e.g. disruption of IP connections). These three client links are part of a Shared Risk Link Group (SRLG) [25].

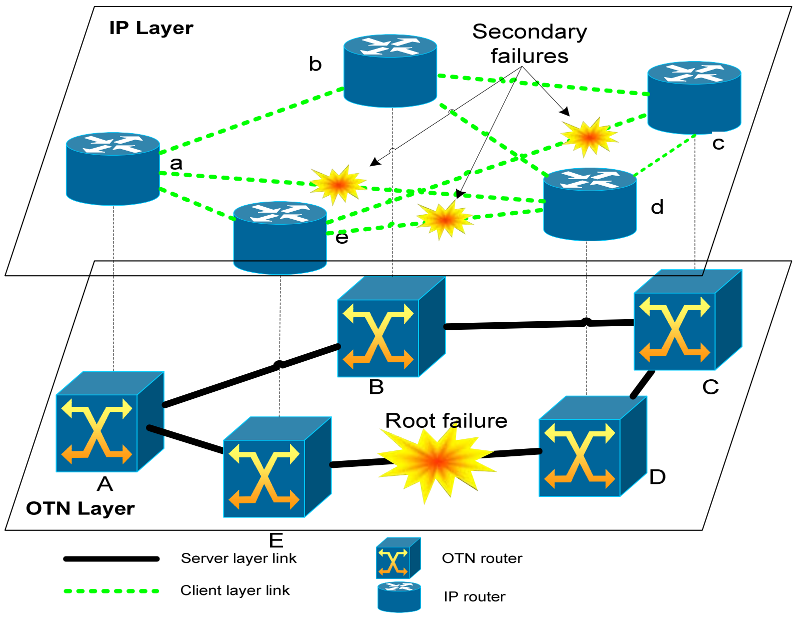


Figure 23: Secondary failures

Recovery at the bottom OTN layer has the benefit that only a simple root failure has to be treated, and that the number of required coarse granularity recovery actions is minimal. Such a strategy will not recover higher-layer failures and may isolate an IP node if its corresponding OXC node fails. Still, exclusive higher layer recovery leads to a lot of finer granularity recovery actions, and will not recover traffic that is not terminated in the IP layer (e.g., leased wavelengths).

10.4.2 Interworking between layers

Recovery can be uncoordinated, resulting in parallel recovery actions at different layers. Though such a strategy is simple and straightforward, it has its consequences on efficiency, capacity requirements and even ability to restore the traffic. During recovery, OTN layer ‘extra traffic’ pre-emption may trigger further upper layer protection [3]. Protection on different layers has different convergence times. A sequential approach using e.g. a hold-off timer or recovery token ensures that slower upper layer recovery actions are delayed until the recovery process escalates upward from the OTN layer.

10.4.3 Multilayer Survivability Strategies

Multilayer survivability means more than coordinating the recovery between multiple layers. For static recovery techniques, spare resources for the same path can be allocated in both layers (e.g. as both an OTN backup path and an IP spare link), called duplicated protection. Despite the reduced complexity, this is a rather expensive solution. Improved multilayer strategies include *logical spare unprotected*, where the IP layer spare resources are not themselves protected, and the *common pool strategy*, where IP spare resources can be pre-empted for OTN layer protection [26].

Dynamic multilayer survivability strategies [27] differ from such static strategies in the sense that they actually use logical topology modification for recovery purposes. Similar to MLTE, they set up and tear down lower layer network connections that implement logical links in the higher network layer in real-time, to reconfigure the logical IP network in case of a network failure. This approach has the advantage that in the logical network, spare resources do not need to be established in advance. In the optical layer spare capacity still needs to be provisioned to handle lower layer failures. Enough capacity is also needed in the optical layer to support the reconfiguration of the logical IP network topology.

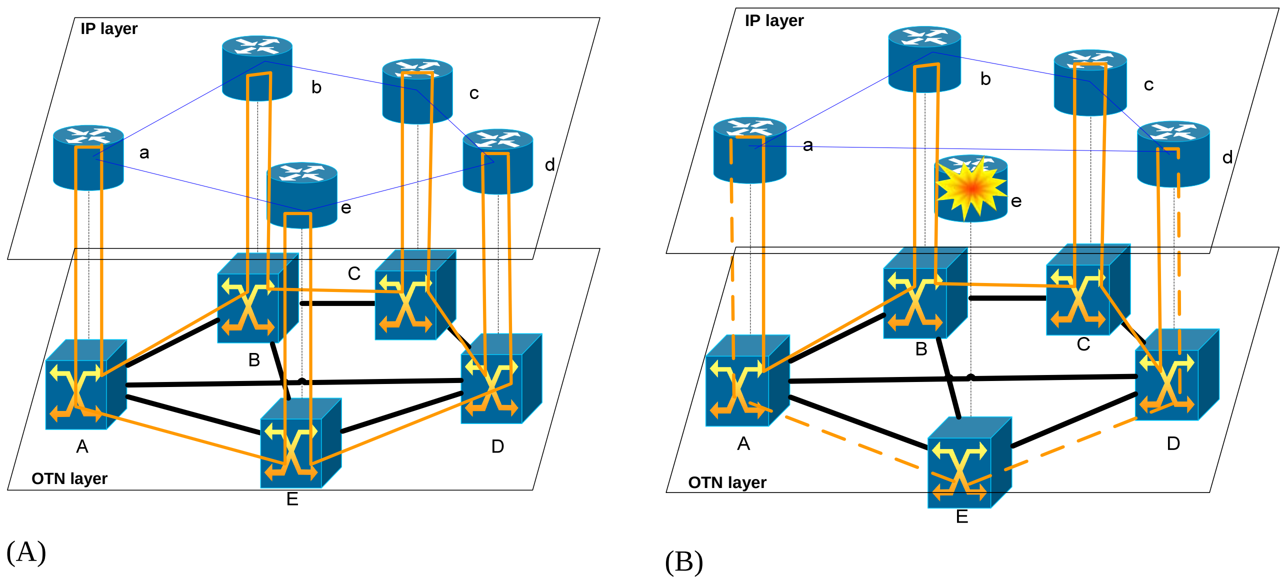


Figure 24: Dynamic reconfiguration

An illustration of dynamic reconfiguration of the logical topology in case of failures is given in Figure 24: Initially, the traffic from router *a* to router *d* is forwarded through router *e*. To this end the logical IP network contains the IP links *a*-*e* and *e*-*d*, implemented by the light paths A-E and E-D. When router *e* fails, nodes *a* and *d* will detect this failure, and request the OTN for a tear-down of the links *a*-*e* and *e*-*d*. The resulting free capacity in the optical layer can be used to set up a direct logical IP link from router *a* to router *d*, over a light path between OXCs A and D.

11 Conclusion

This report started from a general perspective of split architecture, referring back to previous successes of splitting the architecture. Three different model of splitting the control from packet forwarding was discussed, comparing OpenFlow with ForCES. Focusing into OpenFlow, while addressing the functionality carrier class required, primarily a number of open questions was identified. These open questions will form a base for the succeeding studies in SPARC project.

One could consider two different approaches to obtain carrier class flow switching, either to extend the OpenFlow protocol by patching in new network functionality based on evolving requirements and applications, or to identify a more general approach that allows for dynamic modification of the network functionality. One question is whether there is a Generic Forwarding Plane that can be programmed to meet new formats, e.g. operator specific formats, without adding too much of complexity and thereby cost? It is still open, however, the definition of such a Generic Forwarding Plane and if this can be standardized.

Virtual Port was introduced to accommodate MPLS; isolating the manipulation of the forwarding packet header, e.g. label swapping. A Virtual Port can be made generic to accommodate also other type of data formats on demand, i.e. network programmability, the ability of dynamically changing the behavior of both the data- and control plane, for example adding support for a new version of the IP protocol. Integration of the optical layer (e.g. OTN/WDM) is another example of data format that needs different processing than normal packet forwarding. A protocol for configuring the Virtual Port and other hard states data is neither defined.

Scalability is another key characteristic of Carrier Class networks. Two basic mechanism are consider to solve scalability, 1) controller hierarchy that aggregates underlying OpenFlow domains, or 2) partitioning the network in OpenFlow domains that interconnects in peer. Is classification of packet made once, or is there reclassification per domain. What protocols are needed to synchronize between the OpenFlow domains?

Carrier Class also requires Service Management, which is supported by means of OAM. Currently, OpenFlow lacks support for any type of OAM. OAM is essential to implement features such as: resiliency, observability, service validation, etc. Performance and accuracy do depend on implementation of OAM close to the data plane. How does these maps to OpenFlow? Is there Flow OAM and how would such be defined? Would this provide for a flattened OAM model within the OpenFlow domain? How should legacy OAM be treated when passing OF domain?

So, by adding Carrier Class feature to OpenFlow also puts more demand on the data plane processing capability. This leaves question on the split of feature, and how much of these are put to the OpenFlow interface. Time critical features needs support from the data plane, while those across-network coordinated action better be place centrally. How much of the data processing should be performed on the switch vs. what can be accomplished in the controller?

OpenFlow implements support of Network Virtualization. However, it remains here to define how strict virtual network isolation can be implemented to provide for integrity of carrier grade characteristic. Another open question is how virtual network slices can be defined without overlapping address spaces. The Network Virtualization architecture is heavily dependent on the use case defined.

Another open question, not elaborated in this document, is the resilience mechanisms of the controllers themselves. How is state preserved in the network, if a controller goes down? How does the controller get back the state when it comes up again, this is also a performance issue on how fast the controllers are synchronized.

List of figures and/or list of tables

Abbreviations

API	Application programming Interface
BER	Bit Error Rate
BFD	Bi-directional Forwarding Detection
BGP	Boarder Gateway Protocol
CE	Control Element
CIR	Committed Information Rate
DHT	Distributed Hash Table
DPCP	Differentiated Services Code Point
DPI	Deep Packet Inspection
DXC	Digital Cross-Connect
ECMP	Equal Cost Multi Path
FE	Forwarding Element
FEC	Forwarding Equivalence Class
FEC	Forward Error Correction
FIB	Forwarding Information Base
ForCES	Forwarding and Control Element Separation
FPGA	Field Programmable Gate Array
FSC	Fiber Switching
GMPLS	Generalized Multiprotocol Label Switching
HAL	Hardware Abstraction Layer
IGP	Interior Gateway Protocol
IP DSCP	IP Differentiated Services Code Point
L2SC	Layer-2 Switching Capable
LAN	Local Area Network
LDP	Label Distribution Protocol
LDP	Label Distribution Protocol
LFB	Logical Function Block
LFB	Logical Function Blocks
LMP	Link Management Protocol
LSC	Wavelength (Lambda) Switching Capable
LSP	Label Switched Path
MA	Maintenance Association
MAC	Media Access Control
MEP	Maintenance End Point
MGW	Media Gateway
MLTE	Multilayer traffic engineering

MPLS	Multiprotocol Label Switching
MPLS-TP	Multiprotocol Label Switching Transport Profile
MSC	Mobile Switch controller
NE	Network Element
NetFPGA	Network-Field Programmable Gate Array
NIB	Network Information Base
O&M	Operations and Management
OAM	Operation Administration and Maintenance
OF	Open Flow
OSPF	Open Shortest Path First (routing protocol)
OTN	Optical Transport Network
OXC	Optical Cross-Connect
PBB	Provider Backbone Bridges
PBT	Provider Backbone Transport
PCC	Path Computation Client
PCE	Path Computation Element
PCEP	Path Computation Element Communication Protocol
PE	Provider Edge
PL	Protocol Layer
PSC	Packet Switching Capable
QoS	Quality of service
RIB	Routing Information Bases
RSVP-TE	Resource Reservation Protocol
SCTP	Stream Control Transmission Protocol
SDH	Synchronous Digital Hierarchy
SPARC	Split Architecture
STP	Spanning Tree Protocol
TDM	Time Division Multiplexing
TDMC	Time-Division Multiplexing Capable
TE	Traffic Engineering
TML	Transport Mapping Layer
TTL	Time To Live
VLAN	Virtual LAN
VN	Virtual Networks
VPN	Virtual Private Network
VRF	Virtual Routing and Forwarding
WDM	Wavelength Division Multiplexing

References

[1] Information Technology Infrastructure Library (ITIL), available online http://www.itil-officialsite.com/home/home.asp

[2] E. Rosen, A. Viswanathan, R. Callon: "RFC3031 - Multiprotocol Label Switching Architecture" (2001).

[3] Attila Takács, Howard Green, and Benoit Tremblay: "GMPLS Controlled Ethernet: An Emerging Packet-Oriented Transport Technology", IEEE Communications Magazine, September (2008).
[IETF/CCAMP] http://tools.ietf.org/wg/ccamp/charters
[IETF/PCE] http://tools.ietf.org/wg/pce/charters

[4] G. J. Popek, R. P. Goldberg. "Formal requirements for virtualizable third-generation architectures". *Communications of the* ACM 17(7), pp. 412-421, July (1974).

[5] Chowdhury, N.M.M.K. and Boutaba, R., "Network virtualization: state of the art and research challenges," Communications Magazine, IEEE, jul (2009),Vol. 47, No. 7, pp. 2 -26, http://www.mosharaf.com/wp-content/uploads/nv-overview-commag09.pdf

[6] D. Colle, W. Tavernier, A. Gladisch, "Ethernet: beyond the LAN?", DRCN2007 tutorial, October 7th, (2007): http://ibcn.intec.ugent.be/downloads/Tutorial%20DRCN2007.pdf

[7] D. Farinacci, T. Li, S. Hanks, D. Meyer, P. Traina, RFC2784 – "Generic Routing Encapsulation (GRE)" (2000).

[8] IEEE 802.1Qay-2009 IEEE Standard for Local and metropolitan area networks-- Virtual Bridged Local Area Networks -- Amendment 10: "Provider Backbone Bridge Traffic Engineering" (2009).

[9] IEEE 802.1Q-2005 IEEE Standard for Local and Metropolitan Area Networks -- Virtual Bridged Local Area Networks, (2006).

[10] E. Rosen, Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC4364, February (2006).

[11] Cisco Wite Paper, "Router Virtualization in Service Providers", http://www.cisco.com/en/US/solutions/collateral/ns341/ns524/ns562/ns573/white_paper_c11-512753.pdf

[12] Camargos, F.L., "Virtualization of Linux servers," Linux Symposium, p. 63 http://www.linuxsymposium.org/archives/OLS/Reprints-2008/camargos-reprint.pdf

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

[14] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling_, Leon Poutievskiy, Min Zhu, Rajiv Ramanathany, Yuichiro Iwataz, Hiroaki Inoue, Takayuki Hamaz, Scott Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks," 9th USENIX Symposium, Vancouver, Canada, Oct 4-6 (2010), http://www.usenix.org/events/osdi10/tech/full_papers/Koponen.pdf

[15] Bart Puype, Didier Colle, Mario Pickavet, Piet Demeester, Multilayer traffic engineering for multiservice environments, Photonic Network Communications, Vol. 18, No. 2, Oct. 2009, pp. 150-159.

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

[17] Saurav Das, Guru Parulkar, Nick McKeown, Preeti Singh, Daniel Getachew, Lyndon Ong, "Packet and Circuit Network Convergence with OpenFlow," OFC 2010, San Diego, CA, Mar 23- (2010). http://tiny-tera.stanford.edu/~nickm/papers/Openflow-OFC10_invited.pdf

[18] Jean-Philippe Vasseur, Mario Pickavet and Piet Demeester, *Network Recovery*, San Francisco: Morgan Kaufmann, 2004.

[19] Freek Dijkstra, Bert Andree, Karst Koymans, and Jeroen van der Ham, "Introduction to ITU-T Recommendation G.805", Dec (2007). http://staff.science.uva.nl/~fdijkstr/publications/G805-introduction.pdf

[20] Jean-Philippe Vasseur, Mario Pickavet and Piet Demeester, *Network Recovery*, San Francisco: Morgan Kaufmann, (2004)

[21] B. Niven-Jenkins,D. Brungard,M. Betts, N. Sprecher, S. Ueno, „RFC5654 - Requirements of an MPLS Transport Profile”, Sep (2009).

[22] Elisa Bellagamba, Nurit Sprecher, “MR-258 Enabling Next Generation Transport and Services using Unified MPLS,” White Paper BBF, Oct (2010).

[23] D. Katz, D. Ward, “RFC5880 - Bidirectional Forwarding Detection (BFD)”, Jun (2010).

[24] IEEE 802.1ag - IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks Amendment 5: Connectivity Fault Management, Dec (2007).

[25] Dimitri Papadimitriou et al., *Shared risk link groups encoding and processing*, Internet draft, work in progress, June 2002, www.ietf.org

[26] Nico Wauters, Gzim Ocakoglu, Kris Struyve and Pedro Falcao Fonseca. *Survivability in a New Pan-European Carriers' Carrier Network Based on WDM and SDH Technology: Current Implementation and Future Requireme*nts, IEEE Comm. Mag., vol. 6, no. 8, pp. 63-69, Aug 1999.

[27] Bart Puype, Jean-Philippe Vasseur, Adelbert Groebbens, Sophie De Maesschalck, Didier Colle, Ilse Lievens, Mario Pickavet, Piet Demeester, Benefits of GMPLS for multilayer recovery, IEEE Communications Magazine, Vol. 43, No. 7, pp. 51-59, July 2005.

[28] “D-3.1.1 Virtualisation Approach: Concept”, 4WARD, Sep (2009).