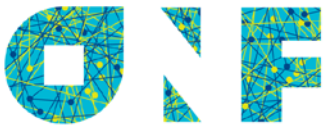

OPEN NETWORKING FOUNDATION

SDN Architecture

Issue 1.1
2016
ONF TR-521

Abstract

This document specifies the architecture of software defined networking (SDN). Issue 1.1 extends SDN architecture issue 1 in light of further work in the industry and in the ONF architecture working group. It also clarifies a number of topics in light of experience with issue 1.

ONF Document Type: TR (Technical Reference), non-normative, type 2
ONF Document Name: SDN Architecture 1.1

Disclaimer

Open Networking Foundation
2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303
www.opennetworking.org

List of Figures

1 Introduction

This SDN architecture issue 1.1 is a stand-alone document that clarifies and extends issue 1 [1], but does not render it obsolete. Appendix C describes and explains the differences.

A great deal of work on SDN has already been done, and continues to be done, in a number of ONF working groups, standards development organizations (SDOs), and open-source communities. Even if it were possible, the architecture would not attempt to invalidate existing work. However, the architecture does aspire to unify the discussion across these groups.

Except for illustrative examples, this architecture intentionally remains abstract and avoids details of target technologies. Additional documents, existing and potential, expand the architecture into focused areas. These include:

- TR-522, SDN Architecture for Transport Networks [4]
- TR-518, Relationship of SDN and NFV [2]
- TR-523, Intent NBI – Definition and Principles [5]

Document structure

Clause 2 is an executive summary, an abbreviated statement of the essentials. Subsequent clauses build up the terminology and concepts in detail, then use them to support the overall architecture.

Clause 3 describes the scope and purpose of the architecture.

Clause 4 concisely defines a number of key terms. Further discussion of the definitions appears in Appendix A.

Clause 5 identifies some of the key concepts of SDN and expands on their interpretation.

Clause 6 discusses the SDN controller in some detail.

Clause 7 describes applications, in less detail. Maximum freedom is allowed by intentionally underspecifying their internals and functions.

Clause 8 is an integrated view of the major SDN components and their interfaces.

Clause 9 views the architecture from a number of particular angles.

Having intentionally minimized the size of the main document text, it is recognized that a number of topics justify wider discussion. Some of them may be developed as stand-alone Architecture Notes documents over the course of time. The appendices contain informative material of sufficient interest to warrant inclusion in this document.

2 Executive summary

An architecture is a necessarily incomplete collection of perspectives over a set of underlying ideas. A consistent architecture reveals no contradictions when viewed from any of these perspectives. A useful architecture facilitates productive development of concepts into realities. An

open architecture minimizes the difficulty of extension in previously unforeseen directions. The architecture described in this document aspires to be consistent, useful, and open.

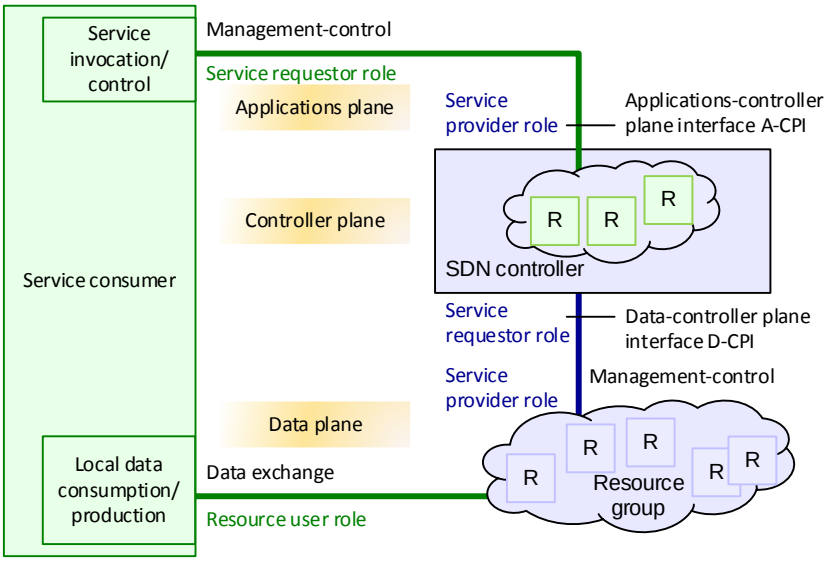

Figure 1 – Basic model

Figure 1 illustrates the basic model of SDN, that of a service consumer (client, user, customer) Green, who exchanges both data and management-control operations with some SDN server or provider, Blue. Although user data is ultimately forwarded or processed by some set of resources (R) that are owned by Blue, Green controls its service via a session contained in a management-control association. It does so by invoking actions on a set of virtual resources (R) that it perceives to be its own. Among other responsibilities, the SDN controller virtualizes and orchestrates the Green resource and service view onto its own underlying Blue resources and services. The concepts of resources and services are intentionally unbounded.

The SDN architecture extends the basic model and clarifies its implications. Key extensions include sharing resources a) among multiple clients, b) dynamically, c) in an optimum way. Other essentials of a complete architecture include management in the classical sense, both of network resources and of services.

This architecture usually portrays client and server as existing in separate business domains, illustrated with separate colors. The reason for this is to emphasize the need for traffic isolation, information hiding, security and policy enforcement at interface points. Depending on the contractual relationship between client and server, visibility and security criteria may be strict or relaxed in any particular deployment situation, including full transparency when appropriate.

SDN controller relationships

The central entity in an SDN is the SDN controller. Figure 2 illustrates some of its key functions and interfaces.

SDN is modelled as a set of client-server relationships between SDN controllers and other entities that may themselves be SDN controllers. In its role as a server, an SDN controller may offer services to any number of clients, while an SDN controller acting as client may invoke services

from any number of servers. As long as they exhibit appropriate interface behavior, the internal details of entities that are not SDN controllers are beyond the scope of the architecture.

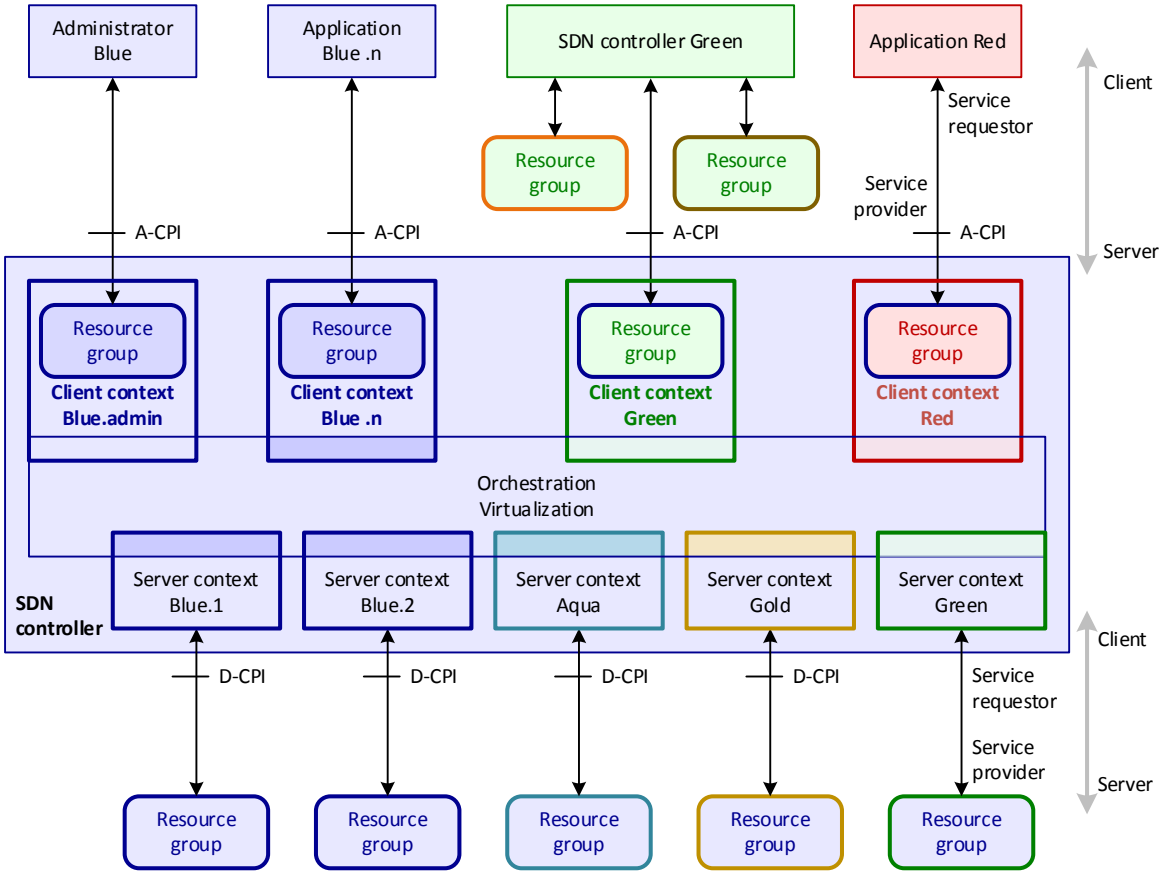

Figure 2 – Core of the SDN architecture

The architecture recognizes dual perspectives on the nature of client-server interfaces. The services perspective is particularly appropriate from a top-down or customer-provider viewpoint. The resources perspective is particularly appropriate from the bottom-up viewpoint of a resource owner, especially an internal administrator. The perspectives are complementary, but emphasize different things. The construction of views and mappings on a common underlying information model helps tie these perspectives together.

The SDN controller satisfies client requests by virtualizing and orchestrating its underlying resources. As the network environment changes and client demands change, the SDN controller is responsible for continuously updating network and service state toward a policy-based optimum configuration.

An SDN controller exposes services and resources to clients via applications-controller plane interfaces (A-CPIs), and consumes underlying services and resources via data-controller plane interfaces (D-CPIs). Each of these interfaces is a reference point for information hiding, traffic and namespace isolation, and policy enforcement.

Management and control are viewed as a continuum, in which an administrator role differs from that of ordinary applications only by having greater scope and privilege. The administrator has authority to configure the SDN controller itself, along with client and server contexts.

Recursion may be deduced from figure 1 by recognizing the repeated service requestor and service provider roles. SDN controllers may associate with other SDN controllers and non-SDN management-control entities in hierarchical or peer arrangements, within or across administrative domains. This permits SDN to span the range from end-user service negotiation to world-wide service provisioning to fine-grained control of individual network elements.

3 Scope

Software-defined networking (SDN) applies the flexibility of software to the entirety of the networking space. It includes unlimited numbers of business relationships, geography spanning the world, and everything from end-user service negotiation and delivery to the planning, installation, provisioning and maintenance of network infrastructure. As well as forwarding traffic, an SDN may process traffic, either as part of added-value services or for service and network maintenance purposes.

The overall architecture is intended to span the entirety of the space. Nevertheless, it is structured for easy subsetting, for example into carrier, cloud, campus or enterprise environments that purchase much of their network service from third parties, across opaque or semi-opaque business domain boundaries.

SDN is based on three principles, which are further explored in clause 5.1.

- Decoupling of traffic forwarding and processing from control
- Logically centralized control
- Programmability of network services

The major components of an SDN are resources and controllers. SDN controllers mediate between clients and resources to deliver services. Most resources are related to traffic in some way, but support resources are also recognized, for example security credentials and notification subscriptions.

The primary roles in an SDN are those of administrator, service requestor and provider, and resource user. Additional roles are not precluded.

This architecture document expands the meaning and implications of the principles, the required and optional functions of the components, and the interfaces, rights and responsibilities of the roles.

Architecture goals

The overall goal of the architecture is to assist providers in better serving their customers – in any number of dimensions – while reducing their own cost to deliver those services. SDN addresses all aspects of this goal, a few examples of which are:

- An environment that reduces the time and cost of developing new services
- Flexible definition and availability of resources, including virtual network functions (VNFs)

- On demand assembly of resources into services
- More efficient resource loading, with continuing real-time optimization
- Facilitating global semantic agreement with a common information model
- Merging traditional business and operations support system (BSS/OSS) functions with control

Because of its wide scope, not all aspects of SDN can be addressed in depth immediately. The architecture is intended to create a common understanding that facilitates parallel development as cost-benefit opportunities arise across the broadly defined problem space. While alignment on principles is important, the single most important underpinning of consistency is arguably a common information model.

The architecture describes principles, components and roles in abstract ways. Any number of implementations can therefore claim to comply with the architecture. Rather than as a vehicle for compliance statements, the architecture may better be used as a reference to which an implementation can be compared. Many open-source projects implement aspects of SDN; the architecture helps understand the gaps. Gaps are not necessarily deficiencies; partial solutions may be completely adequate for their target markets.

One of the long-standing problems of the industry has been silos, separate areas with separate expertise, separate staffing, separate business, service, and investment considerations. While the classic voice-data silo partition is finally gone, or nearly so, other silos persist. A few examples:

- Transport vs packet
- Wireline vs wireless
- Local vs long-haul (access vs core)

And new silos:

- Data center, cloud vs dedicated and dispersed physical elements
- Network functions virtualisation (NFV) vs SDN

There will continue to be important differences along these, and other, dimensions. However, SDN ought never provide a technical justification for the perpetuation or creation of silos. The goal of the architecture is to open possibilities, and especially to expose common ground, such that silos can be collapsed whenever and however it makes sense.

4 Definitions

A good definition is concise and correct, avoids direct or indirect self-reference, and aligns well with at least one of the understandings in common usage. If multiple understandings exist in common usage, a definition must take a position, rather than sanctioning confusion.

Further, a good definition should be narrow enough and crisp enough to exclude invalid candidates. Fuzzy outer bounds are a major flaw in many definitions in the industry.

The implications of a definition are often important, but not obvious. A definition may therefore require informative material that is not essential to the definition itself, but that assists in its understanding. Informal descriptions of the category and qualifiers may be helpful. Examples are often useful, both of candidates that do, and also of candidates that do not, satisfy the criteria. For brevity in the main text, these discussions appear in Appendix A.

4.1 Abstraction

Definition: The representation of an entity or group of entities according to some set of criteria, while ignoring aspects that do not match the criteria.

4.2 Association

Definition: The information needed by two contracting entities as a precondition to establishing management-control sessions between each other.

4.3 Client

Definition: An entity that receives services from a server.

4.4 Client context

Definition: The conceptual component of a server that represents all information about a given client and is responsible for participation in active server-client management-control operations.

4.5 Domain

Definition: A grouping of entities according to some set of criteria.

4.6 Management-control continuum (MCC)

Definition: The principle that the functions of management and of control are largely, if not entirely, the same.

> Note – In keeping with SDN convention, this document refers to one of the key entities as an SDN *controller*. The sessions it supports are usually referred to as *management-control*.

4.7 Orchestration

Definition: The ongoing selection and use of resources by a server to satisfy client demands according to optimization criteria.

4.8 Policy

Definition: An administrative rule or set of rules that specifies the action(s) to be taken when specified condition(s) occur.

4.9 Recursion

Definition: The repeated application of a pattern in which the input to each iteration is derived from the output of the previous iteration.

4.10 Resource

Definition: Anything that can be used to deliver a service.

4.11 Server

Definition: An entity that provides services to a client.

4.12 Server context

Definition: The conceptual component of a client that represents all information about a given server and is responsible for participation in active server-client management-control operations.

4.13 Service

Definition: The delivery of value for some time interval by a server to a client.

4.14 Service context

Definition: The conceptual component of a client context that represents all information about a given service.

4.15 Virtualization

Definition: The abstraction of particular underlying resources, whose selection criterion is the allocation of those resources to a particular client, application, or service.

5 Concepts

5.1 Principles of SDN

SDN is based on three architectural principles and the definition of open interfaces, as described below. All require interpretation.

5.1.1 Decoupling of traffic forwarding and processing from control

The purpose of this principle is to permit independent deployment of control and traffic forwarding and processing entities. This principle is not obviously a game changer per se. Transport equipment, for example, has long decoupled control from forwarding and traffic processing. However, decoupling is a necessary precondition of centralized control and of recursion, specifically hierarchical recursion. Decoupling also allows for separate optimization of platform technology and software life cycles.

The architecture reflects the decoupling principle in the form of an entity called the SDN controller, which has management-control responsibility for some set of resource groups. The resource groups are considered to be in a data plane, so-called because most of them are directly or indirectly associated with processing client traffic. Recursive decoupling is embodied in the idea that a client has management-control access to a virtual set of resources and services that is created on and exposed by an SDN controller.

Further reading: 5.1.4 Open interfaces, 5.4.2 Resources and resource groups, 6.1 SDN controller as feedback node, 6.5 Delegation, 6.6 Client context, A.9 Recursion

5.1.2 Logically centralized control

The term *logical* signifies that control behaves as a single entity, independent of its possible implementation in distributed form.

Decoupling of traffic processing from control is a precondition of centralized control. The centralized control principle asserts that resources can be used more efficiently when viewed from a wider perspective. A centralized SDN controller can orchestrate resources that span a number of subordinate entities, and thereby offer better abstractions to its clients than if it could only abstract subsets of individual subordinate entities. The best example of this is the exposure of a single monolithic forwarding domain that is built atop an arbitrarily large and complex underlying network.

However, a number of factors argue against a single monolithic control point for the entire global telecoms network.

- Scale is the obvious reason, along with related factors such as sheer propagation delay. The optimal choice among a large number of options may not justify the increased complexity, as compared with a slightly suboptimal choice among a smaller set of options.
- Management-control information exchange is highly constrained across trust boundaries. By separating SDN controllers along (at least) trust domain boundaries, the architecture exposes the necessary security and policy enforcement points.
- It is necessary that SDN co-exist with non-SDN technology in all possible ways: as controlling or controlled entities or as peers, within and across administrative, technology and other domains, responsible for greater or lesser functionality.
- It may be optimal to delegate the fine-grained operation of some functions (for example, protection switching) further down into network resources, with only the results summarized at the level of the more centralized controller.
- A more subtle reason is that management-control communication is distributed around the network on a network of its own, which itself requires management-control. Domain partitioning is likely to be a better solution than an all-inclusive network attempting to control itself.

The principle of centralized control is best understood as a recommendation to consider cost-benefit trade-offs of centralization.

Further reading: 6.1 SDN controller as feedback node, 9.1 Security, B.3 Realization considerations, B.5 Complexity

5.1.3 Programmability of network services

This principle permits a client to exchange information with an SDN controller, either by discovery or negotiation prior to the establishment of a service, or during the lifetime of a service according to changes in client needs or the state of the client's virtual resources. Agility is the payback, both in negotiating a satisfactory service and in turning it up. This is tied into the ability of SDN to dynamically tap a wide domain of existing resources, or to create new resources on demand, especially virtual network functions (VNFs).

The programmability principle is based on the premise that service-consuming applications and resources benefit from collaborating in detail in negotiating and delivering services, even on a continuing basis. This may be true of some applications. At the other end of a continuum, the intent interface model expects the client to express its desired outcome – possibly in considerable detail, possibly after considerable negotiation – but then leave the realization and real-time optimization to the SDN controller.

The merit of the intent model is that it disentangles resource operation from the client's purpose, allowing for simpler clients that may be more independent of the infrastructure. The client trades off simplicity for participation in fine tuning or continuing optimization, recognizing that, if the ongoing service cannot be delivered as committed, the feedback is more likely to be an abrupt failure indication.

Further reading: 5.3 Service and resource oriented models, 6.6 Client context

5.1.4 Open interfaces

This fourth aspect concerns SDN implementation and deployment, not the fundamentals of the architecture. It presupposes the well-defined partition of functions and interfaces, and specifies that the interfaces be public and open to community definition. The purpose of open interfaces is to encourage competition. The desired competition is most likely to emerge in the support of completely mainstream features on completely mainstream white box hardware.

It should be recognized, however, that proprietary features will continue to be offered, and that network owners will make their own trade-offs about the value of such features compared to the disadvantages of proprietary interfaces.

The open interfaces concept is a recommendation to perform standard functions in standard ways, while supporting flexibility in ad hoc extensions of interfaces for proprietary functions. Implications of open interfaces are for further study.

Whatever the nature of the functionality and interfaces, integral fit into an industry information model [3] is key.

5.2 Roles

The roles described in this clause are useful to the SDN architecture. Other roles are not precluded.

5.2.1 Administrator role

An administrator is characterized by having greater visibility and privilege than an ordinary client. Normally, an administrator would be a trusted employee of the same organization that operates the SDN controller (note). The administrator's responsibility is to create an environment that can offer services, to modify the environment from time to time, to monitor the environment for proper operation, and to act on exceptions beyond the ability of the environment to resolve internally.

> Note – Because the SDN controller is the center of the SDN architecture, this role is described in that context. A role with corresponding responsibility is also needed in non-SDN-controller applications and resource platforms.

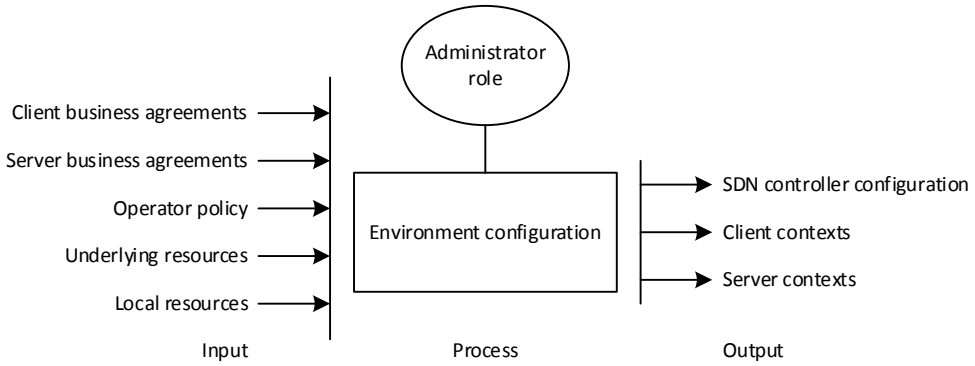

Figure 3 – Administrator role

Figure 3 shows that the administrator configures SDN controllers and the necessary contexts.

The administrator creates a working SDN controller from some substrate (creation of an SDN controller, possibly as a VNF, downloading its code, etc., is beyond the scope of the current discussion). The SDN controller is created by default with an administrator client context that has unrestricted visibility and authority to perform all other operations. The administrator configures the controller with server contexts to access underlying resources, and updates them from time to time as needed. The underlying resources are themselves configured by their own administrators at their own (underlying) levels.

The administrator then creates a client context for each of its clients, which includes allocation of underlying resources to the client in the process called virtualization, as well as supplementary configuration. The administrator configures each client context with policy that defines the actions and bounds permitted to the client. An administrator may modify a client context during its lifetime, and may destroy a client context if the client relationship terminates.

Because an SDN controller is responsible for continuously optimizing its use of resources according to a global optimization policy, the administrator installs and modifies the optimization policy as needed. An administrator may also be expected to create subscriptions and logs on its own behalf.

Further reading: 6.1 SDN controller as feedback node, 6.6 Client context, 6.9 Server context, B.3 Realization considerations

5.2.2 User and provider roles

Clients or service consumers satisfy their needs by requesting services from SDN controllers as providers, and they achieve their data transfer and data processing objectives as users of the corresponding resources (figure 1).

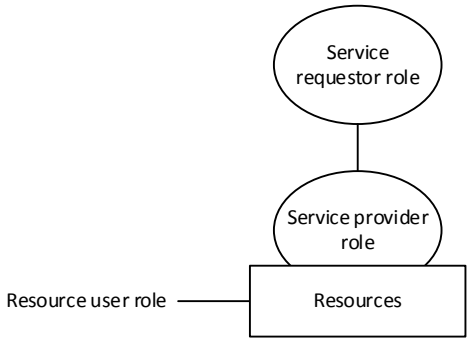

Figure 4 – User and provider roles

As illustrated in figure 4, the resources explicitly or implicitly available to a client play a service provider role, offering services to a service requestor for configuration. The service requestor represents the management-control aspect of the client in setting up the desired service; it may also subscribe to notifications from the provider. A given service requestor may invoke and manage-control any number of simultaneous services.

While the service requestor represents the setup of service by a client, the resource user role represents the client's use of the according resources to satisfy its service needs. Usually this is accomplished by exchanges in the data plane. The services offered by the provider are constrained only by the nature and functions of the available resources and the agreement between server and client.

Further reading: 6.6 Client context, 6.8 Service context, 6.9 Server context

5.3 Service and resource oriented models

As described in clause 2, a service-oriented model is especially appropriate from the top-down viewpoint of a customer, while a resource-oriented model is appropriate from the bottom-up viewpoint of an operator, and especially from the viewpoint of an operator's administrator role.

5.3.1 Service-oriented model

From the service-oriented perspective, the basic operation across an SDN interface is service invocation and management. According to this model, a client requests create, read, update or delete (CRUD) operations on all of, or a component of, a service context object. The server is an SDN controller, which is expected to validate the request against policy and available resources, and either satisfy the request or provide an appropriate exception response.

In most cases (note), the service context governs the behavior of data-plane resources that permit the client to exchange traffic with other network entities. Examples of such service contexts range from contracts for residential telephony to corporate virtual private networks (VPNs). At a very detailed level, a service context might represent a connection through a network.

Note – Examples of value received by a client that do not involve data-plane exchanges include information discovery and negotiation of various kinds, and directory lookups. In some such interactions, it may not be necessary to instantiate a service context object.

Service-related resources are released when the service context is deleted. A service context may persist indefinitely, i.e., until explicitly deleted, or it may terminate due to events such as signaling or the expiration of an explicit schedule. A management-control session between client and server may or may not exist continuously during the lifetime of a service context.

A service request (CRUD operation) is necessarily expressed in terms of entities, actions, and names/addresses known to the client. Internal functions of the client are not subject to architectural standardization, but it would be expected that the client retain a view of the service, as expressed in its own terms, possibly updated with state information derived by query or notification from the server.

The server necessarily retains the service context, which may include the service invocation as expressed by the client. The client's request is the desired outcome that guides the server in orchestrating and virtualizing underlying resources to satisfy the client's demand. This is a continuing process, not only as clients amend their existing services, but as network state and resource contention change over time.

The architecture does not constrain the entities and actions known to the client. In general, these client entities and actions must be mapped into entities and actions known to the SDN controller acting as server (note). In general, neither client nor server is in a position to supply complete mapping information, which must therefore be a shared responsibility. Shared semantics are established by prior agreement.

Note – Transparent mappings are not precluded. They may be especially appropriate for server administrators.

A number of ways may be appropriate to populate the mapping function. A few of them include:

- Offline business or technical level negotiation, e.g., of access points of presence (PoPs), with subsequent provisioning of explicit mapping equivalents into a mapping database to which both client and server have access.
- Auto-discovery, e.g., of the speed of an Ethernet PoP. Protocols such as LLDP [10] may operate across the client-server data-plane boundary.
- Publication by the server of a service catalog, or the equivalent, the schema of which may be used to express the desired service. Some such schemas may be known a priori from standards.
- Negotiation of agreement by protocol, for example through proposing and accepting inter-domain labels.

A given client may have any number of service contexts in place on a given server at any given time. The service context identifier allows for unambiguous CRUD operations on new or existing services.

A server may offer services to a number of clients. All of the server's information for a particular client is contained in a conceptual component called a client context. A client context may contain any number of service contexts.

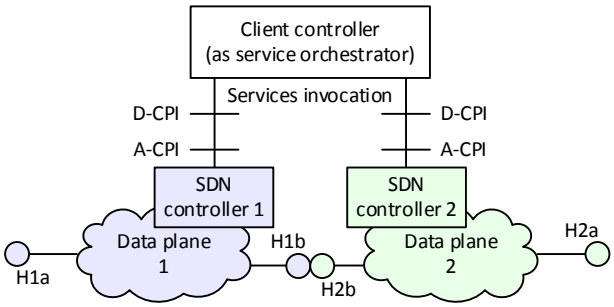

Figure 5 – Client as service orchestrator

As illustrated in figure 5, a client, itself acting as an SDN controller, may orchestrate services that span one or more servers. Orchestration of composite services requires that the client be responsible for choosing and provisioning intermediate interface points (H1b, H2b), as well as the service-specific behavior of each node. Successive partitioning is one expected way to realize a service, where a single SDN controller may perform the entire process for the subnetwork within its own domain (e.g., data plane node 1), or a hierarchical stack of controllers may successively process successively more detailed service invocations, or anything in between. Peer controllers may invoke neighbor recursion to achieve the same ultimate goal. In all cases, the edge points must be coordinated and therefore recursively visible, while the internal details of the service are optionally visible, but are often left for the immediate controller.

The selection, connection, and provisioning of network topology and nodes shades into the resource-based perspective on SDN.

Further reading: 6.1 SDN controller as feedback node, 6.2 Orchestration, 6.6 Client context, 6.8 Service context, 8.3 Peer controllers

5.3.2 Resource-oriented model

Resources are the things that can be used by SDN. A resource is represented at an interface by an instance (actual or potential) of some managed object class in an information model. An SDN controller exposes resources to a client, which is free to use them in its information queries and service requests.

Explicit client resources – i.e., those visible at an A-CPI – are typically established through prior business agreement and are provisioned into the SDN controller's virtualization function by the provider administrator role. These resources include a mapping from the client's name/address space to a pre-allocated equivalent resource in the underlying resources available to the SDN controller.

An example of a simple pre-allocated resource is a mutually agreed point of presence between a residential or enterprise network and a carrier. A more complex example is a complete, detailed subnetwork that might be leased by an enterprise from a carrier. The latter case implies correctly that client resources need not map to the same object classes as underlying server resources, and need not remain the same over the course of time. The SDN controller supports a stable view of

these resources to the client, but is free to vary the underlying resources as necessary, subject to client service level agreement (SLA) that may limit service disruptions.

It is also the case that client services often require the use of resources that have not been explicitly pre-allocated, for example subnetworks to interconnect client points of presence, or gateways to the E.164 numbering space or to the Internet. In this case, the SDN controller's orchestration function is responsible to select resources that will satisfy the client's service requests within the bounds of the client's SLA. Although these resources are dedicated to the client for the duration of the service, they are not exposed to the client. These resources therefore need not be mapped from the client's name/address space into the server's.

Again, there is no implication that the underlying dedicated resources remain the same for the duration of the service. The underlying resource inventory may change over time, due to failure, recovery, network build-out or administrative action. If NFV is in place, it may be possible to create entirely new resources or scale existing resources on demand and at convenient points in the network topology. Contention with other clients, especially those with more stringent SLAs, may cause resource rearrangement. And the client itself may request changes during the course of service, for example to improve its quality of experience or to reduce its cost.

Further reading: 5.4.1 Information model, 5.4.2 Resources and resource groups, 5.4.3 Resource data base, 6.2 Orchestration, 6.6 Client context, 6.8 Service context

5.4 Primitives

5.4.1 Information model

One of the SDN value propositions asserts that the communications environment of the future will include more vendors with a wide range of product offerings, on short and unsynchronized release schedules. Especially if a service invocation goes through several APIs from several vendors before reaching a data-plane device, semantic mismatches in the information conveyed will clearly result in chaos. A common information model (note) is key. As new fragments are developed, it is important that they be integrated into the common model.

> Note – RFC 3444 [11] describes the difference between an information model and a data model. Compared to information model mismatch, data model incompatibility is far less difficult to discover, diagnose and resolve.

Purpose-specific data models and customized views can be derived from the common information model as necessary. These are comparatively easy to adapt pragmatically, as long as semantics are preserved. This allows purpose-specific APIs to be produced, while enforcing consistent semantics.

5.4.2 Resources and resource groups

Any SDN service is built upon some set of resources, whose functions and interfaces are configured to the particular need. Resources may be physical or virtual, active or passive, and in many cases, may be created, scaled, or destroyed, by or at the behest of the client or the server. Resources available to SDN include VNFs, as defined by the ETSI NFV initiative [6].

A resource is modeled at an interface as an instance of a managed object class in an information model. Resources can be repeatedly subdivided or combined into larger resources in any way that makes sense, either by definition of the underlying information model, by management-control configuration, or by real-time orchestration and virtualization. This includes so-called slices of resources, in which it may be desired to only abstract part of an underlying resource, for example some part of the capacity of a link.

A client sees a resource as defined by a view. The view is necessarily expressed using identifiers and concepts known to the client; mapping between client and server views may require joint input by client and server organizations.

A view may restrict the information available to the client, the actions that can be invoked, and the notifications that it publishes. If it is shared, the same underlying resource may be virtualized to present different views to different clients. A view may also represent information such as subdivided allocation, potential or actual existence, and potential scalability.

The resource or data plane is sometimes considered to be simply an undifferentiated cloud. However, resources naturally tend to fall into groups. Resource groups are not architecturally fundamental; they are convenient abstractions.

Resource groups may exhibit some or all of the following properties:

- Fate sharing; the likelihood that all resources in the group lose or regain connectivity to the SDN controller together, that all fail or recover together, that all are subject to backup, restoration and migration together.
- All of the resources in the group subject to the same management-control association.
- The ability to use relative distinguished identifiers within the context of a group, rather than more global identifiers.
- Tightly integrated functionality within the resources of the group. Delegated or autonomous functions exemplify this property: dynamic sharing of bandwidth, local fault recovery, fault correlation, the ability to inject test or other stimuli at one point and detect the consequences locally at some other point.
- A common notifications publication and subscription scope for all resources in the group.
- A knowable upper bound on the ability of resources to be augmented or scaled without having to migrate into other groups.
- A holding space for resources that are available for use, but are not actually in use at any given time.
- A topologically significant point in a network graph, useful in optimizing path or other computations.

Further reading: 5.3.2 Resource-oriented model, 5.4.1 Information model, 5.4.3 Resource data base, 6.3 Virtualization, 6.6 Client context, 6.8 Service context, 8.2 Notifications, 9.3 Relationship of SDN and NFV, B.2 Identifiers, B.6 Persistence

5.4.3 Resource data base

The resource data base (RDB) is the conceptual container for information that needs to be retained in an SDN controller beyond some small locality in process space and time (note). Among other things, the RDB is conceived as the local repository of all underlying resources. The RDB does not appear on the architectural drawings because it exists everywhere in an SDN controller, as needed to support the various active functions and interfaces.

> Note – This is to exclude cached information that remains within a single function, and calling parameters between functions.

An important criterion for retention of a data fragment in the RDB is the need for its availability after controller reinitialization or replacement. Large fractions of the information in client and server contexts are appropriate for the RDB, including views of underlying resources that are presented to clients, services and their resource mappings, topology and resource inventories, policies, profiles, and subscriptions. The identifier mapping between underlying resources and client resources must be retained here, as must the optimization criteria used by the orchestration function.

A second criterion for RDB storage recognizes that some information is created by one entity and may subsequently be used on a continuing basis by other entities. A prime example is the mirror of an underlying resource, conceptually contained in a server context.

Nothing is said about the implementation of the RDB, whether it is a formal database of some type, whether it is distributed or centralized, whether all data is represented in a mutually consistent format, etc. It is recognized that different backup and restoration plans will be appropriate for different data fragments.

An administrator role would have unrestricted access to the entire RDB. For other users, fine-grained access policy must be enforced on data fragments in the RDB, as they relate to provider, client or server contexts. Some policy may be implicit, e.g., no client can view information in the provider's or another client's context, but it may also be necessary to explicitly define additional access policy. Being persistent, such a policy would itself be contained in the conceptual RDB.

Further reading: 5.2.1 Administrator role, 6.1 SDN controller as feedback node, 6.6 Client context, 6.9 Server context, B.2 Identifiers, B.6 Persistence

5.5 Controllers and planes

The conventional view of SDN is structured into planes. A data plane processes user traffic, a control or controller plane hosts SDN controller instances, and instances of service-consuming applications reside in an applications plane. While this is a good introductory model, it is less useful when conceptualizing SDN in depth.

A given SDN controller governs some set of resource groups (in a data plane), whose state it controls in a feedback loop. It virtualizes these underlying resources and exposes a customized virtual resource environment (as a data plane) to each of its clients.

In much the same way that a controller is thus sandwiched between data planes, a network resource (note) is sandwiched between a client and a server. The resource is supported by an underlying virtualization (in a controller plane), and is used by a client or SDN controller (in a con-

troller or applications plane). Even a hardware network element exposes virtual resources to its SDN controller by way of a (possibly minimal) client context, which must have been configured by an administrator.

> Note – The term *network resource* distinguishes entities that have some relation to traffic processing from support resources such as client-server security credentials, notifications subscriptions, profiles, etc. These latter resources are typically not virtual.

As to applications, beyond the A-CPI, the SDN architecture says nothing about what may exist as a controller's superior entities, whether and how they serve end users directly or are themselves intermediaries of some kind.

The strongly recursive nature of the architecture reduces the conceptual value of the original SDN data, controller, and applications planes. Sometimes, management is also regarded as a plane, which does not fit well into pictorial representations of the architecture (note). As management and control are recognized as different aspects of the same thing, roles are a more useful way to model the relationships.

> Note – Parallel planes are meaningful and useful in terms of geometric analogy. Each plane is independent and sees only the planes above and below itself. The geometric implications of intersecting planes (i.e., lines) convey no useful analogies.

A final objection to planes as a fundamental concept is that everything of interest exists in instances. It can be argued that amorphous plane models have obscured important aspects of the problem space, such as controller-controller peer interactions and resource grouping.

For these reasons, this SDN architecture document uses the concept of planes occasionally, descriptively, informally, colloquially, but not rigorously.

Further reading: 5.2 Roles, 6.1 SDN controller as feedback node, 6.6 Client context, A.9 Recursion

6 SDN controller

The SDN controller is at the heart of the architecture. It is the intelligent entity that controls resources to deliver services. Its core function is the real-time multi-dimensional convergence of a changing resource environment and a changing service demand environment toward an optimum, where the optimization criteria may also change in time.

As exposed from the underlying infrastructure, a given resource is controlled by only one SDN controller (note). An SDN controller exercises management-control over some set of resource groups, conventionally portrayed in a resource or data plane to the south of the controller, and accessed through a data-controller plane interface (D-CPI). The aggregate of resource groups under a given SDN controller is referred to as an SDN control domain; the term may also include the SDN controller itself.

> Note – It is recognized that some aspects of an underlying resource may be controlled by other means. Coordination of these activities is necessary, but the details are beyond the scope of the architecture. An example would be an operations support system (OSS) in an

administrator role, disabling a resource via the SDN controller before performing a software upgrade.

An SDN controller offers services to its clients by exposing a resource group to each client. The interface to clients is conventionally drawn to the north of the controller, and called an applications-control plane interface (A-CPI). The interface is also often called a northbound interface (NBI), sometimes an intent interface. The mapping function at the interface permits an SDN controller to offer services to SDN entities, but also to applications that do not claim to conform to SDN principles.

An SDN controller may act as a service-consuming application by using resources and invoking services offered by subordinate or peer SDN controllers or non-SDN entities.

An SDN controller may be implemented via centralized or distributed computing technology, with or without redundancy. These are important considerations, but are implementation details of the SDN controller as a functional block, and lie beyond the scope of the SDN architecture.

Note – Architectures are recursive. It would be legitimate to develop an implementation architecture for SDN controllers, subject to alignment with the functional view of this top-level SDN architecture.

Further reading: 6.1 SDN controller as feedback node, 6.2 Orchestration, 6.3 Virtualization, 6.6 Client context, 8.1 Interfaces, 9.1 Security, B.1 Reliability and availability, B.6 Persistence

6.1 SDN controller as feedback node

As an architectural functional block, an SDN controller may appear in any number of environments. It is fair to ask what makes it an SDN controller, what aspect, should it be absent, would preclude a candidate body of software from being an SDN controller.

The core function of an SDN controller is to continually adapt the state of its resources to serve its clients according to an optimization policy. The concept of state is interpreted broadly. The very existence of a particular resource is itself a state, as are the values of all of its attributes and its relationship to other resources, both within and beyond the controller's domain. The essence of an SDN controller is its participation as the active entity in a feedback loop, as illustrated in Figure 6.

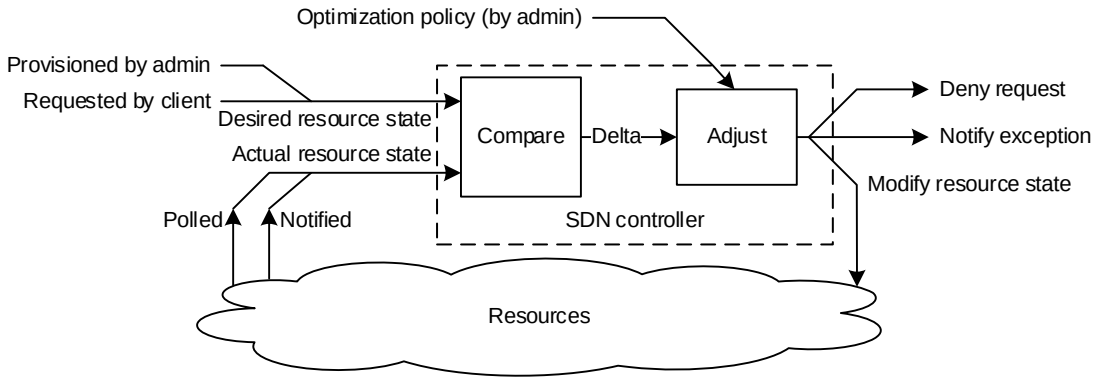

Figure 6 – Control as feedback

Control is the process of establishing and maintaining a desired state. Open-loop control is of little value in the SDN context; feedback is the essence of control.

Figure 6 illustrates that the administrator of the resources configures some desired state, which may change from time to time. The various clients invoke service requests, which translate into desired state. These also change from time to time, both as clients come and go, and also as each client modifies its service demands. The actual state of the resources includes measures of load, changes due to failure and repair, and administrative actions. Actual state may be polled on an ongoing basis or updated asynchronously by notification. The controller evaluates the difference between desired and actual state in light of the optimization policy (note), which may also change from time to time, and attempts to modify the state of the resources accordingly. The optimization policy may include the ability to create (or request the creation of) new resources, to scale or migrate existing resources, etc. State convergence may include negotiation with neighboring domains.

> Note – It is expected that the optimization policy would include compliance with all of the client SLAs. Nevertheless, there could be provider policies that governed how the effects of network overload or failure would be distributed among clients of various types.

If desired and actual states cannot be reconciled within the bounds of the policy, the SDN controller issues an exception, either immediately by rejecting a client request or as a notification during ongoing operation.

In addition to its core feedback control function, an SDN controller may invoke arbitrary supplementary functions, support arbitrary collections of additional features, any number of interfaces and protocols, any number of applications of arbitrary type and complexity, resources of any and all categories, etc., according to its particular deployed purpose.

Further reading: 5.4.2 Resources and resource groups, 6.2 Orchestration, 6.5 Delegation, 8.1.2 Additional interfaces, 8.2 Notifications, A.8 Policy, B.6 Persistence

6.2 Orchestration

In the sense of feedback control, orchestration is the defining characteristic of an SDN controller. Orchestration is the selection of resources to satisfy service demands in an optimal way, where the available resources, the service demands and the optimization criteria are all subject to change.

When the optimization function indicates that a better optimum can be achieved, the orchestration function adjusts the state of the resources under its control to move toward that optimum.

> Note – Policy may or may not prevent in-service reassignment of resources.

Depending on its environment and the optimization criteria, the orchestration function may be complex or relatively simple. Candidate orchestration algorithms may be evaluated on their ability to deal with complexity, as well as on their real-time responsiveness to change.

Orchestration includes at least the following functions:

- Validating service requests from clients against client-specific policy, denying requests that fall outside policy.

- Configuring resources and subordinate services to satisfy client service requests according to the provider's controller-wide policy and the specific policy and SLA associated with the client.
    - This includes configuring data plane entities to enforce policy themselves, for example by policing ingress traffic rates and priorities, by filtering ingress traffic address fields.
    - This includes creating and configuring data plane mechanisms for dynamic sharing, such as prioritized weighted fair queueing engines.
    - This includes the configuration of encapsulation, address translation, or other means to ensure mutual isolation of client traffic.
    - If permitted by client and provider policy, this includes requesting the instantiation, scaling, migration or deletion of resources, as may be appropriate.
    - This includes configuring and operating network support functions such as protection switch engines, connectivity fault management (CFM), spanning tree protocols (STP), performance monitoring, as specified by provider policy or as specified or implied by the client SLA.
    - These operations may need to be recursively invoked from one SDN controller downward to another before they reach actual traffic-processing engines.
- Coordinating service requests and service changes with neighbor domains, be they SDN domains or otherwise, and whether the other domains are peers or hierarchically subordinate.
- Publishing notifications of interest to particular client subscribers, in the terms of the pertinent client context, and notifications of global interest to global subscribers (administrators of the SDN controller).

Virtual resources in the client context are dedicated to the given client, and exist because the client needs to express its service demands in terms of some kind of entities, usually at least its points of presence on the provider's network. The satisfaction of a client service request may require the orchestration function to select additional resources from the underlying pool available to the server, i.e., resources that are not dedicated to the given client a priori, and invoke services against those resources. The selection of such shared resources is affected by policy and resource traffic load or congestion.

Because virtual resources, service requests, and notifications exist in specific client contexts, it is necessary that the orchestration function collaborate intimately with virtualization.

Further reading: 6.1 SDN controller as feedback node, 6.3 Virtualization, 6.6 Client context, 8.2 Notifications, 9.1 Security, A.8 Policy, B.2 Identifiers, B.5 Complexity, B.6 Persistence

6.3 Virtualization

As stated, orchestration is the core active component of an SDN controller. Virtualization is the complementary process that populates and maintains the resource and identifier environments

that link orchestration with clients. Orchestration and virtualization interoperate inextricably; for clarity, they are described as separate functions, without meaning to imply anything about separability of implementations.

Recall that virtualization is the abstraction of resources allocated to a particular client, application, or service. Because virtual resources are conceptually contained in client contexts, a client context is a precondition for a virtualization.

An SDN controller has a resource data base (RDB) that conceptually contains all of the underlying resources. Initially, none of them are allocated to any particular purpose (note). An administrator may create a custom virtualization for each client context, in the form of allocations from the underlying resource pool. By definition, a given virtual resource is wholly dedicated to a particular client (although still accessible to the administrator).

> Note – To avoid unintended consequences, it may be desirable that resources initially appear in administratively locked state.

Not all of a client's virtual resources are necessarily visible to the client. Of those resources that are exposed to the client, not all attributes are necessarily visible, and not all exposed administrator-writeable attributes are necessarily writeable by the client. Identifier translation is needed only for resources that are visible to the client. A virtualization may be created and populated in any way suitable for the circumstances.

One method is implicit creation and population of resources, exemplified by a residential subscriber who supplies a street address to a portal as part of signing up for service, from which a fixed access point of presence can be derived as the necessary initial virtual resource. Additional client resources may also be populated from equipment and inventory records, such as digital subscriber line (DSL) modem or passive optical network (PON) optical network unit (ONU), and a physical or virtual residential gateway (RG). The services delivered to such a client are otherwise free to use whatever resources may be selected by the orchestration function.

Another method is explicit negotiation between customer and provider, for example to agree on the detail of a number of user-network interfaces (UNIs) of a corporate customer, or to specify an entire topology of a subnetwork that may be leased by the provider to a re-seller. Explicitly pre-negotiated resources permit the client to specify greater detail in its service requests, while correspondingly reducing the number of choices available to the orchestrator in satisfying service requests.

A third method is dynamic virtualization. In satisfying client service requests, the orchestration function may subdivide, combine or otherwise abstract underlying resources as necessary. The resulting resource would be recorded in the appropriate client and service context, though it would not be visible to the client.

Changes to resource environment, business agreements, or other factors, necessitate the ability to incrementally update virtualizations in place. Updates must be coordinated with the orchestration function.

Further reading: 5.2 Roles, 6.2 Orchestration, 6.6 Client context, A.15 Virtualization, B.2 Identifiers

6.4 Resource sharing

The principle of logically centralized control implies that no given resource is controlled by more than one external entity. If several clients contend for the resource, they must contend as clients of an SDN controller, which is responsible to arbitrate amongst their claims.

Resources may be permanently allocated to a given client or service in their entirety, in which case there is no possibility of contention. Resources may also be allocated as needed, either on demand or on a scheduled basis. Examples of such wholly allocated resources – at some level of virtualization – include physical points of presence (UNI, network-network interface (NNI) data plane ports), dedicated wavelengths, dedicated spectrum, dedicated time slots.

Resources may also be shared dynamically, e.g., packet by packet. This may require specialized features in the infrastructure, e.g., that the SDN controller support at least some aspects of prioritized weighted fair queueing, that the SDN controller (potentially in a hierarchy) recognize attributes such as priority, committed and extended information rate in service requests, and that it combine such requests in accordance with the provider's policy on oversubscription. Similar examples may be drawn from capacity allocation on wireless or passive optical networks (PONs).

Arbitrary dynamic sharing of all network resources is complicated by the need to identify specific underlying resources in a mix of arbitrary client topologies and at least a partial mesh in which arbitrary flows may be multiplexed on arbitrary links, changing the mix at every node. This complexity encourages the allocation of large flows to fixed capacity paths through the core network, with the core network engineered to achieve statistically satisfactory performance for smaller flows.

Especially for aggregate bundles of resources, e.g., to realize a particular end-to-end quality of service (QoS), it may be appropriate to use time-sensitive, usage-sensitive, or auction pricing as ways to prioritize the highest value demands. Such pricing could be based on pre-agreed policy or negotiated dynamically with clients. Store and forward network services may also assist in dealing with temporary resource limitations.

Another resource that can be shared is address space. Depending on business agreement, a client is often free to use all of the address space (e.g., IPv4, C-VID) that exists at its hand-off network layer; the client relies on the server to isolate traffic. Isolation may be achieved by physical separation, by encapsulation, or by network address translation (NAT). Alternatively, addresses or address blocks may be administratively allocated (and represented in policy form) when the client context is created, or may be requested as needed during operation.

Further reading: 5.4.2 Resources and resource groups, 6.2 Orchestration, 6.3 Virtualization, 6.5 Delegation, B.5 Complexity, B.6 Persistence

6.5 Delegation

The architecture allows an SDN controller to delegate functions into the underlying data plane for various reasons. Examples include PON dynamic bandwidth assignment (DBA), load balancers, protection switching engines, STP, LLDP, CFM.

The SDN controller acts as an administrator to install its policy that governs the behavior of these functions. Through certification, testing, specification, or otherwise, the responsible hu-

mans expect the function to conform to the range of possible policies, and expect the SDN controller to have no conflicts when it delegates the fine details to the intelligent function.

Further reading: 5.4.2 Resources and resource groups, 6.1 SDN controller as feedback node, 6.2 Orchestration, 6.4 Resource sharing

6.6 Client context

The relationship between client and server organizations (customer and provider, or even applications and infrastructure within a single company) is represented by an association, which defines the parameters of interoperation between client entities and server entities. This association must then be devolved onto specific instances of client and server platforms to allow and constrain management-control sessions between these instances. To represent the reciprocal relationship, clients are configured with server contexts, and servers (e.g., SDN controllers) are configured with client contexts.

A client context models everything that exists in an SDN controller (note) to support a given client. To a given SDN controller, a client exists only during the lifetime of its client context. If the relationship with the client terminates, all trace of the client may be removed by deleting its client context and performing suitable resource reclamation.

> Note – By the principles of recursion and abstraction, SDN access to an underlying network element requires that the underlying network element support a virtualization of its resources and a client context.

When a new client is to be recognized on a given SDN controller, the server administrator creates a client context that contains at least the mandatory association attributes, for example to establish identity and security that permit client-server management-control sessions (note). The administrator may also populate the client context with virtual resources and policy tailored to the client, or even service instances, depending on the agreement with the client. Additional initial information is included as may be appropriate.

> Note – If the client is also an SDN entity (controller or application), the client administrator performs symmetric actions.

After the creation of a client context, the client and server may establish one or more management-control sessions (for brevity, just called sessions). A session is the mechanism that supports management-control communication between a specific instance of a client and a specific SDN controller in the context of an association. During the session, the client may invoke services and modify the state of its resources.

Similar to user login, a management-control session normally begins with an exchange of identity and security credentials, followed by agreement on an initial state, much of which may be restored from prior sessions. (The initial state of the first session is that of the resource group established by the server administrator.) A session continues with the exchange of information. Each information exchange can be attributed to that session, for example in an audit log. A session may continue indefinitely, or end with an explicit logout, a failure, or a timeout. In many cases, the client will automatically restore a lost session, and it may be expected that the SDN controller also be able to restore a lost session.

A client's association and its services are required to persist even in the absence of management-control sessions between client and server. When the SDN controller and client establish subsequent management-control sessions, the client's policy, resource, and service views must be restored. Restored state may also include, for example, notification subscriptions and client-specific profiles. Logs may be needed, in part to permit clients to view events of interest that occurred during disconnection. Much of the client context must therefore be persistent. As the repository of client-sensitive information, backup and restoration of a client context must be secured from unauthorized access.

An association may allow any number of management-control sessions for a given client organization, from one or several client platforms. In a minimal client context that supports only one session, the session identification may be implicit.

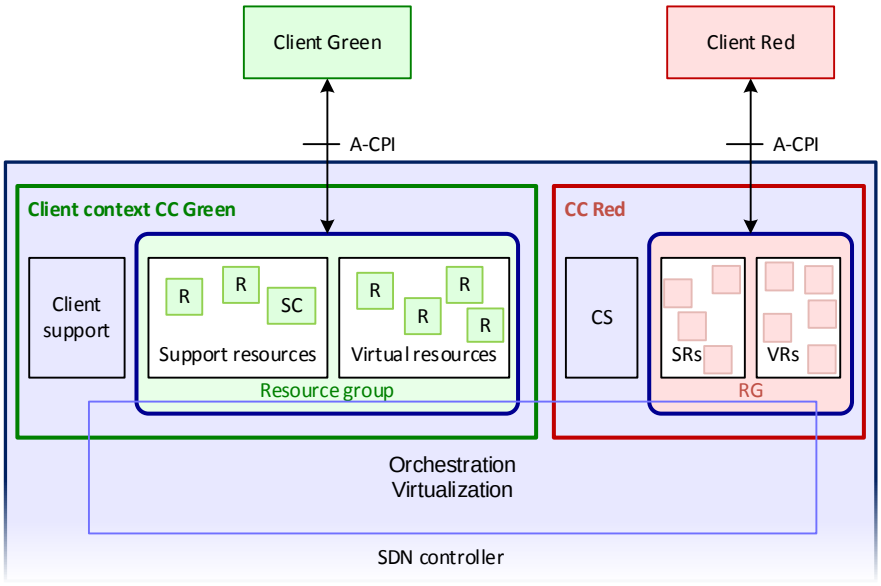

Figure 7 – Client context

Figure 7 illustrates that, logically, a client has visibility and control of the resources in its resource group. Semantically, the A-CPI between client and controller is at the boundary of the resource group. The client context also conceptually contains support information and functionality that are needed to support the client but that are not exposed to the client.

The functional contents of a client context are:

- One resource group. The resource group defines the semantic interfaces (A-CPI) exposed to the client. Two sub-groups are identified.
    - Virtual resources represent infrastructure resources that are created from the SDN controller's underlying resources through the process of virtualization, and that are exposed to the client by way of a mapping function. The most pertinent example is the client's point(s) of presence.
    - Support resources, which represent functions hosted in the SDN controller itself. Their purpose is to enable or facilitate interaction with the client. Examples of support resources include client login profiles, notification subscriptions, logs. As

suggested by the resource called SC, current service contexts are also retained here, including the client's SLA expectation.

- Client support, CS. This function supports the client in as many ways as may be necessary. It may include both code, ephemeral data, and persistent data. While the client's resource group contains resources that are at least visible to the client, though not necessarily writeable, the client support block may contain additional resources that are not exposed to the client at all, or private views of resources dedicated to the client. Content of the client support block includes information to map identifiers and actions between client and server, policies on what the client is allowed to see and do (client view definition), how client traffic is to be isolated (e.g., NAT, encapsulation, reserved or on-demand addresses or ranges), and provider logs or usage measurement for billing or other purposes. Part of the policy function deals with security, representing everything necessary to allow the client to connect to the SDN controller, for example certificates, keys, management-control channel encryption policy, client logins, policies on single or multiple login.

It will be apparent from the overlapping descriptions above that the blocks in figure 7 are not intended to be sharply bounded discrete components with defined interfaces, nor to imply an implementation. The decomposition is merely intended to clarify the functions of the client context. Additional functions may be included, subject to the definition that the client context represents all things and only things that support a given client, and that the client have visibility and freedom of action only within the bounds of policy enforced by the server.

When an SDN controller is created, a default client context is required, with unrestricted privileges and views. As well as configuring the SDN controller itself, and importing underlying resources, this permits the SDN administrator to create and populate client contexts for additional clients.

Further reading: 5.2.1 Administrator role, 5.4.2 Resources and resource groups, 6.3 Virtualization, 6.7 Service context, 6.9 Server context, A.2 Association, A.8 Policy, B.3 Realization considerations, B.6 Persistence

6.7 Multiple client management-control sessions

The client context of architecture issue 1.1 is a deep dive into the agent component of architecture 1.0. Among other things, it recognizes that the client may not have a continuous management-control session with the controller, but that the client environment and the services must persist nevertheless.

Having recognized that there can be zero active management-control sessions in a given client context at any given time, it is natural to enquire whether there can be more than one, either sequentially or simultaneously. The simultaneous case is more complex for various reasons, and is the basis for this clause.

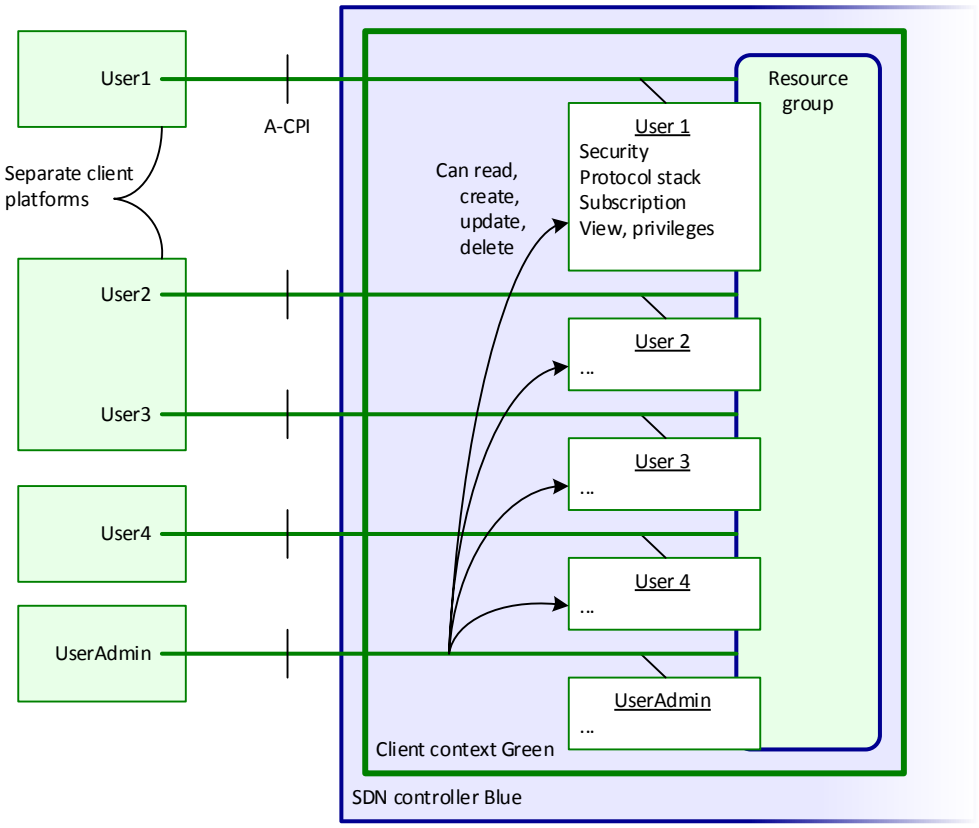

Figure 8 – Multiple users of a client context

Figure 8 serves as the basis for discussion. As to notation: Blue is a provider, the owner of the SDN controller; Green is a client who requires multiple user logins. The Green bound around the client context suggests that everything within is dedicated to Green, as distinct from other clients or general purpose use. Users are hosted on platforms shown as filled green rectangles, and are semantically connected to the resource group in the Green client context. All interactions between Green users and the Blue SDN controller occur within the context of some overarching business association, the concrete manifestation of which is the blue bound around the resource group, a policy enforcement barrier. A Green user has no visibility or control outside the resource group.

White rectangles with green borders indicate user-specific information and code, some of which may be internal to the resource group. These may be designated client sub-contexts.

It would be possible to support only a single client sub-context, allowing multiple sessions. In such a case, every user of the single login would have the same credentials, privileges, and profiles, including for example notifications subscription. If two sessions contended for the same resource, the resulting behavior would not be apparent.

If Green includes human users, however (or proxies thereof), each user may be expected to have different interests and different responsibilities, and therefore to require different views, different privileges, and different profiles. That is, beyond the global (Blue) policy applicable to all of Green's operations, each user login would be further constrained by an independent (Green) pol-

icy. Contention among Green users could also be determined as a matter of (Green) policy. These considerations are entirely up to Green.

Figure 8 therefore shows a user administrator login, which would be the default login created by Blue at the time of client context creation. The user administrator has unlimited visibility and privilege (unrestricted sub-context policy), but only within the Green resource group. Among its privileges is the ability to create and manage client sub-contexts for other Green users.

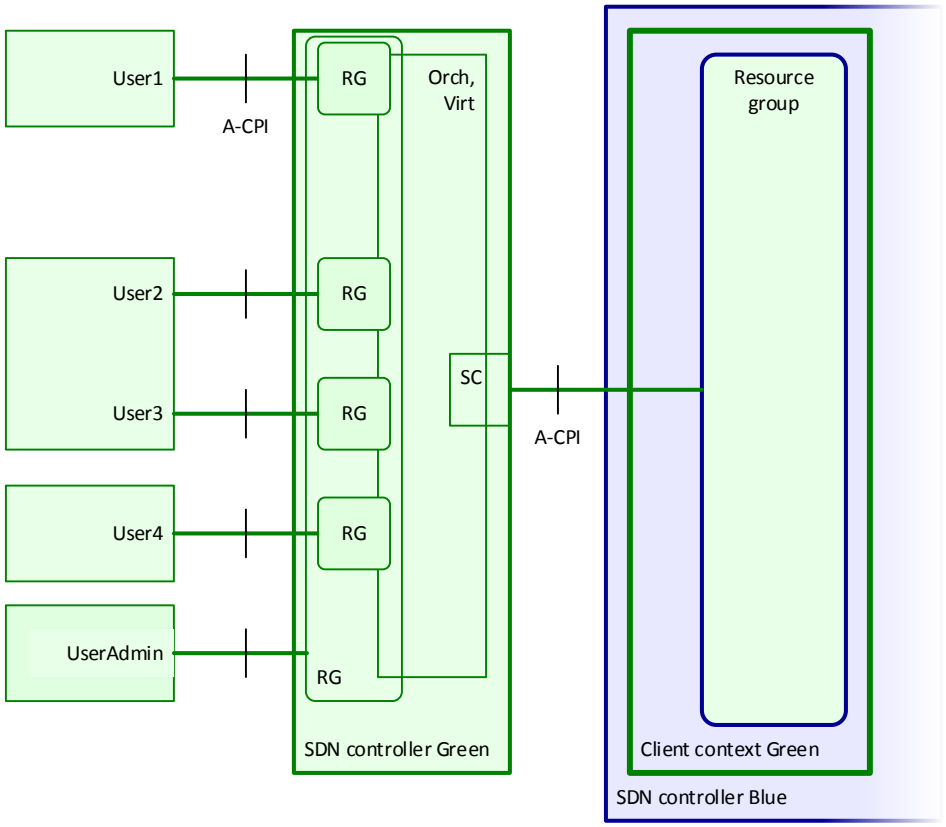

Figure 9 – Separated SDN controller Green

Code in the Green client context is responsible to orchestrate and virtualize Green resources and services, and in particular to resolve contention issues within Green's resource and service environment. At this point, Green's environment has become an SDN controller itself (figure 9).

> Note – The newly partitioned Green SDN controller may reside on same platform(s) that host the Blue SDN controller.

The separate Green SDN controller is now in a position to orchestrate resources across more than one SDN or legacy resource groups (see Green in figure 2). Further, the separate Green SDN controller can contain arbitrary code and use arbitrary external databases and supplementary functions, completely independent of Blue business or security concerns.

This arrangement may offer considerable benefits. Examples include:

- Enterprise client Green.com, with custom-tailored policies about which of its employees are allowed to see and do what, under the control of a Green administrator. These policies are of no concern to Blue in any event, and Green could well regard them as business

confidential. An example of responsibility subdivision would be one group of service administrators with minimal privilege, and a separate group of troubleshooters with full visibility of the Green context.

- Interpreting an intent interface. To say, “Bob is (or is not) allowed to connect to the Internet,” is to raise questions that cannot possibly be answered by Blue. Further, for security reasons, Green might not even want Blue to know how to interpret such a policy. The Green SDN controller may look at directories, policies, or anything else of interest to interpret the intent into terms known to Blue. The Green-Green A-CPI can be whatever is appropriate to the custom service; the Green-Blue A-CPI would be a standardized view of an information model.

- The Green SDN controller could be a web portal, fielding requests from various users for various Green services, validating and billing the users, and passing the requests into Blue for implementation. In this context, Green could be the services sales division of network operator Blue.

- As a variation on the previous point, Green client contexts could accept session requests from anonymous clients. Two options could be offered.

  Log in as an existing client. The Green SDN controller might need to refer to external directories or data bases to authenticate the user and discover enough information to redirect the user session to the proper client context on (in this case) the Blue SDN controller.

  Negotiate service and business offerings with a potential new customer, for example by exposing a catalog for user browsing and ordering. If an agreement was reached, a client context would need to be created on the spot on (in this case) the Blue SDN controller, and the client session redirected into that client context.

- The possibility of parallel protocols, for example OpenFlow-switch, OF-config and SNMP logins, having appropriately different views over the same underlying resource base.

- The possibility of an open roaming interface, available for anyone to connect. Upon connection, an open service could be established with a default SLA and policy for each device (e.g., free Wi-Fi). For premium service, it would also be possible to identify the device or user, for example via SIM card or web portal login, thereupon download and install a customized SLA and policy for that device, and collect usage and billing information.

- Customer self-install of residential access service, using a similar model, either modifying an existing service (e.g., when moving house) or newly subscribing. In this case, subscriber SLA and policy might need to be customized on the spot.

Figure 9 is an omniscient view that shows how the SDN architecture may be extended, but in practice, Blue neither knows nor cares whether the Green application is structured as an SDN controller.

As to the SDN architecture, this analysis suggests that a single instance of A-CPI per client context suffices, and that multiple logins need not be supported. That said, it might yet be worthwhile for Blue to offer a controller feature that permitted, for example, one writer and multiple readers with individual notification subscriptions. Another possibly worthwhile option might be for Blue to offer parallel logins for OpenFlow-switch, OF-config, OVSDB, and/or SNMP, with appropriately predefined profiles for each.

Further reading: 5.2.1 Administrator role, 5.4.2 Resources and resource groups, 6.6 Client context, 8.1.2 Additional interfaces, A.2 Association, A.8 Policy, Appendix C Evolution from issue 1 to issue 1.1

6.8 Service context

NB – Not to be confused with server context

Services usually involve data plane information exchanges between client and server domains (figure 1). These data plane exchanges typically continue during intervals when no active management-control session exists between the SDN controller and the client. The existence and characteristics of a service are represented by a persistent service context object in the client context of its owner.

> Note – A service is always delivered to some particular client, and therefore exists in some particular client context.

A service context conceptually contains at least all of the attributes of a service as requested by the client, and server-specific information necessary to map service attributes into the realization of the service. Notifications, logs, and profiles may or may not be associated with a service. There is no implication that a given service be owned by any particular user login, but also no prohibition.

A service context is populated by the server, driven by events that may include any or all of:

- Client context creation by server administrator, to provide prenegotiated or default service
- Service creation operation by client, specifying service type and attributes
- Service update operation by client, modifying existing service
- Changes in network state or policy on the part of the server that alter the static attributes of orchestration and virtualization necessary to realize the service.

When a service context is deleted, all resources used by that service are released and no further trace of the service remains in the server's environment.

The identifier of the service context allows for unambiguous exchange of service-related control, query and notification information between client and server. In the general case, a client can create, read, update and delete (CRUD operation) services at will. If policy restricts a client to a single service, the service context may be created by the server administrator, and may not be subject to create-delete operations by the client.

At any given time, zero or more services may exist for any given client.

Further reading: 6.6 Client context, 6.7 Multiple client management-control sessions, 6.9 Server context

6.9 Server context

NB – Not to be confused with service context

Recall that a client context contains everything needed by an SDN controller to support a client or application. By symmetry, a server context contains everything necessary and sufficient for an SDN controller (note) to manage-control a given underlying resource group. As with the client context, an administrator creates a server context for each expected underlying resource group.

> Note – The internal structure of an SDN application is out of scope of the architecture, unless the SDN application is itself an SDN controller. However, it would be expected that any application contain much of the information in a server context.

As with the client context, a server context is at least partly persistent. Figure 10 illustrates an SDN controller, with server contexts that relate it to underlying services and resource groups, two of which are within the same Blue trust domain as the controller itself.

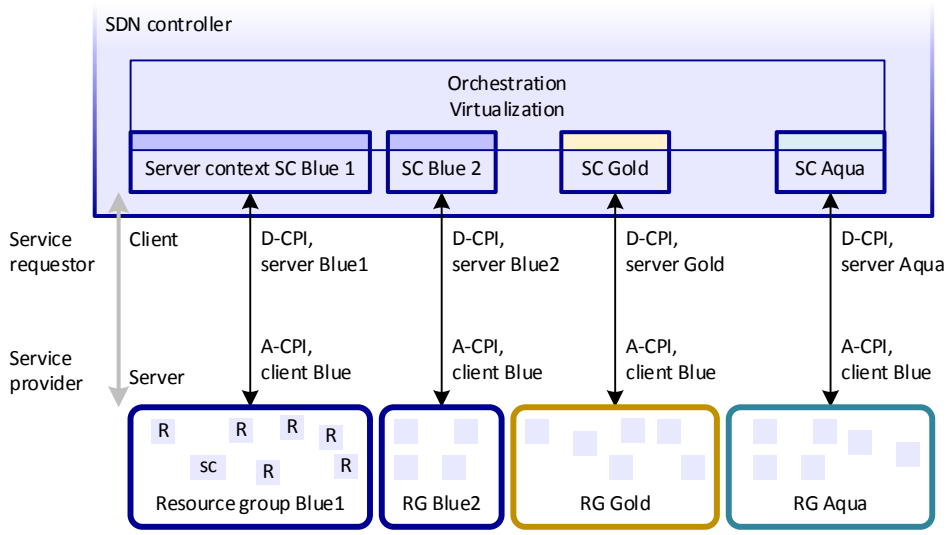

Figure 10 – Server context

An SDN controller must contain a server context for each resource group to which it may connect. This includes at least the association attributes, the login and security information that will be acceptable to the underlying resource group environment, and whatever stack parameters are needed to achieve semantic compatibility. If multiple logins from the same controller's server context are contemplated (e.g., parallel OpenFlow, OF-config, SNMP sessions), this information would be accordingly multiplied.

Customized software may be included, for example to adapt the SDN controller's internal APIs to the protocol and information model of the subordinate resource group. While the A-CPI references on figure 10 imply that the underlying resource groups are themselves aligned with the SDN architecture, this is not necessarily the case. They may well be non-SDN entities of any arbitrary nature, as long as the controller's server context is capable of importing their capabilities.

Note – OpenDaylight plug-ins may exemplify such customized software.

Dynamic information retained in the server context would include off-normal state information, such as alarms or notifications of reduced capacity.

The view provided by the underlying resource group is available to the SDN controller's server context. The server context may also contain information such as logs. Performance monitoring uploads may be retained here, at least temporarily until deleted or offloaded onto analytics systems. Other functions and other storage are not precluded.

It is for further study whether a client SDN controller requires policy enforcement to protect itself from its underlying resource servers.

Further reading: 5.4.2 Resources and resource groups, 5.4.3 Resource data base, 6.6 Client context, 6.7 Multiple client management-control sessions, A.2 Association

7 Applications

The classical SDN model describes an application plane, populated by instances of applications. An application is a client entity that may request services of some kind (usually involving data or resource plane exchanges) from an SDN controller server. The idea is recursive; an application can itself be an SDN controller. Sometimes applications are described in terms of end users, but from the perspective of a server SDN controller, there is no way to know. An SDN controller offers services; an application-plane entity on the other side of its A-CPI invokes the services.

Management-control interactions between applications and SDN controllers occur in sessions, which are subsumed by client-server associations. A compliant application thus contains a server context for each SDN controller with which it may need to establish a session.

Recognizing that an application may be an SDN controller, it may be useful to list some characteristics to clarify the terminology.

- From the viewpoint of an SDN controller, anything that invokes services at one of its A-CPIs is an application.
- If an entity orchestrates more than one resource group across its D-CPIs, it is an SDN controller (figure 5). Figure 10 illustrates an SDN controller as an application client of resource groups supported by four servers.
- As the intelligent entity in a feedback loop, an SDN controller is likely to engage in fine-grained state monitoring and to have a wide repertoire of nuanced responses to deviations from the desired state. A service-consuming application is more likely to express only its purpose (intent) against a single server, leaving the satisfaction of that purpose to underlying intelligence. Feedback to a non-SDN controller application is more likely to be coarse-grained: accept-deny or alarm on failure, with little expectation that the application can remedy the situation. A non-SDN controller application is more likely to have no ongoing management-control sessions and minimal or no notifications subscriptions.

- If an entity exposes its functionality to human users, it is probably an application. Because humans always communicate with software through mediation devices, the depth and sophistication of the mediation affects the interpretation of this criterion.

Further reading: 5.1 Principles of SDN, 5.3.1 Service-oriented model, 5.5 Controllers and planes, 6.1 SDN controller as feedback node, 6.2 Orchestration, 6.6 Client context, 6.7 Multiple client management-control sessions, 6.8 Service context, 6.9 Server context, A.2 Association

8 Putting it together: the integrated architecture

Figure 11 reiterates the earlier figure 2. It shows the controller as the central element in an SDN, and with its conceptual content of contexts and the orchestration-virtualization function. In this illustration, SDN controller Blue offers services to clients Green, Red and its own organization, Blue. Client context boundaries and resource groups are colored to indicate dedication to clients, while resource group border colors identify the underlying resource owner and the accompanying isolation and policy enforcement barrier. An administrator in the Blue organization has unrestricted view and privilege over the SDN controller itself, along with all client and server contexts.

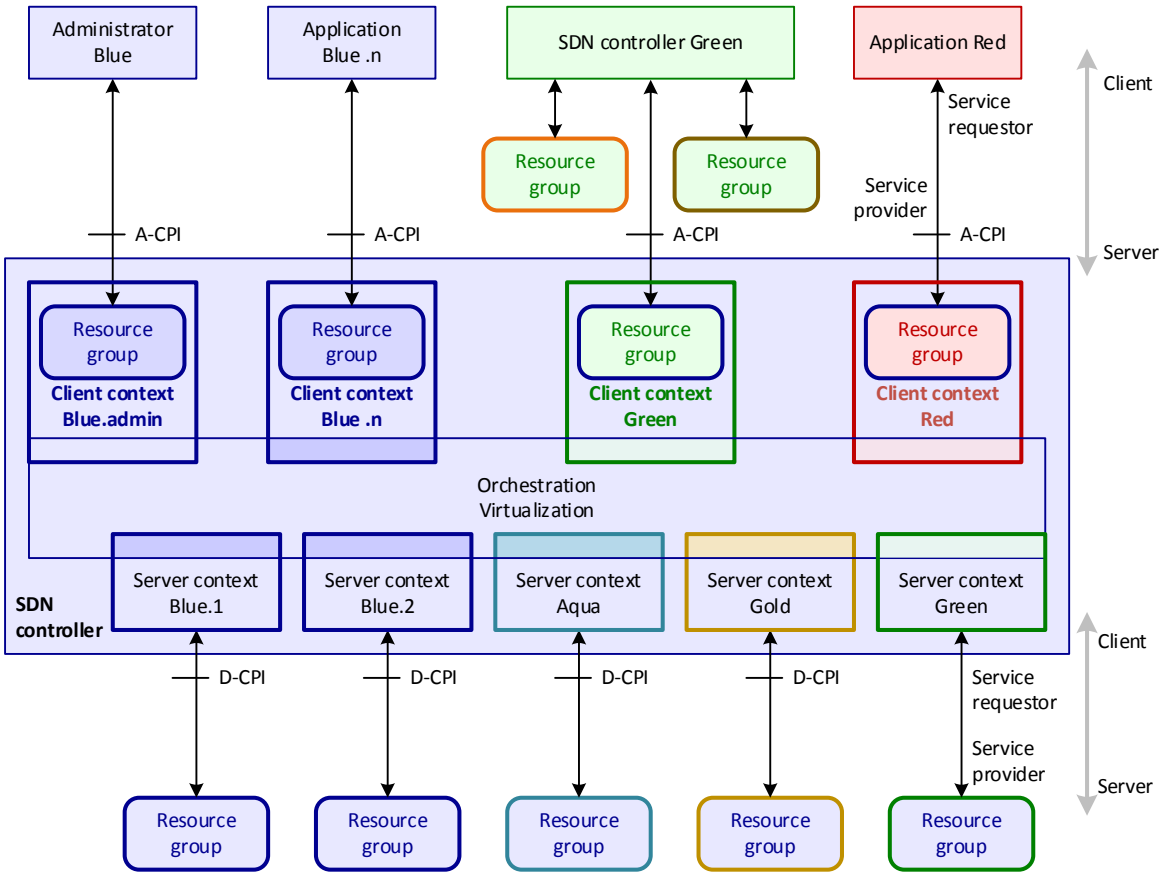

Figure 11 – SDN controller illustrating contexts

The SDN controller consumes underlying resources offered by Gold and Aqua, as well as two resource groups owned by itself, Blue. The Green client application is shown as an SDN control-

ler in its own right, orchestrating services and resources across several domains, one of which is Blue. At the D-CPI, the Blue SDN controller is shown in the same way, and in fact as a service requestor client to Green. Reciprocal requestor-provider roles allow Blue and Green to operate as peers in providing services that may have been requested by clients of either domain.

The resource group in the administrator client context includes the default unrestricted policy, and the information necessary to manage (note) the SDN controller and the various client and server contexts. In addition, it contains support objects (resources) for use by the administrator login, for example logs, subscriptions, and profiles.

> Note – Because the idea of client or server contexts is that they contain all of the information related to a client or server, the administrator's resource group is modeled as containing only the information needed to manage such client and server contexts, but not their contents. Whether this distinction actually matters is an open question.

Further reading: 5.2.1 Administrator role, 5.4.2 Resources and resource groups, 6.6 Client context, 6.9 Server context, 8.3 Peer controllers, A.8 Policy

8.1 Interfaces

8.1.1 Controller plane interfaces

The data-controller plane interface (D-CPI) is the interface between an SDN controller and the resource groups under its direct control. The application-controller plane interface (A-CPI, also called NBI) is the interface between application and SDN controller, across which an application invokes services from the SDN controller.

By recursion, A-CPI and D-CPI are instances of a common interface, as viewed from the server and client sides, respectively. These reference points are not intended to constrain the choice of protocols, models, or functions. The semantic content that may be conveyed across the interface is unbounded. A client should be able to query or discover the underlying environment and subscribe to notifications from a server. A client commands; a server responds.

From semantics down through the various levels of the stack, any given session must support at least a subset of some common information view and should deal with misalignment gracefully. Exact or partial compatibility across an interface may be known a priori, discovered or negotiated when the client-server session is established, custom-configured, or any combination of those.

The idea of a priori knowledge deserves explanation. In real-world deployments, functionality, information model compatibility and stack compatibility will be known and specified from the initial engineering of the network through vendor and product selection, through product qualification, and into coordinated deployment, along with integration into related systems including business processes such as product catalog development, advertising and billing. Client and server associations and contexts will have to be created and configured, and mapping databases will have to be established. The configuration process may be simplified by dynamic mutual discovery of capabilities, but random run-time matches of arbitrary clients with arbitrary SDN controllers is unlikely.

Further reading: 5.3.1 Service-oriented model, 5.4.1 Information model, 5.4.2 Resources and resource groups, 6.6 Client context, 6.9 Server context, 7 Applications

8.1.2 Additional interfaces

An SDN controller may be implemented as a collection of distributed components, the interfaces between which are internal, and out of scope. This clause describes external interfaces to the controller that are not explicitly shown in the architecture. These fall into three broad categories, which overlap widely.

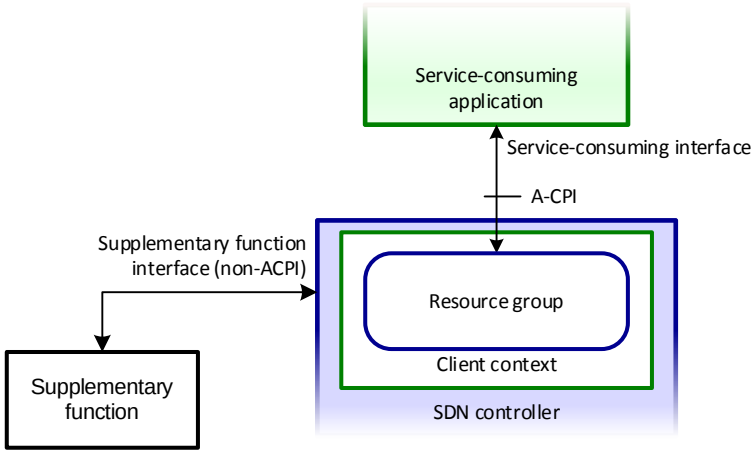

Figure 12 – Supplementary function example

Supplementary functions

An SDN controller may use other interfaces and other functions, and in particular may invoke supplementary functions as exemplified in figure 12. If a supplementary function modifies the state of network resources or services, the modified state must be flagged to the SDN controller that invoked the function, for example via notification or function return parameters.

Supplementary functions may also be internal to the SDN controller. The architecture makes no distinction between internal and external functions; it assumes that the SDN controller can achieve the necessary outcome.

An SDN controller may perform a given function by executing its own code, including library code, or by invoking supplementary functions that reside externally. For example, scope and clean partitioning suggest that path computation will almost certainly be external to an SDN controller, as will services such as authentication, authorization, accounting (AAA), dynamic host configuration protocol (DHCP), domain name system (DNS).

An SDN controller is architecturally allowed to instantiate, destroy and scale resources, some types of which may be subsumed under the concepts and tools of the ETSI NFV discipline. Because of the current prominence of NFV, it is worth emphasizing a strong preference that the mechanism be an invocation by the SDN controller of NFV-related tools, either from a library or across a protocol interface, but not by way of reinvention.

> Note – If an NFV-related entity invokes SDN functions, it should align with the usual SDN server-client model at an A-CPI.

Directories and databases

An SDN controller requires considerable information for its normal operation, which is architecturally modelled as being available in a conceptual resource data base. It may be desirable that some of this information reside elsewhere, accessible to the SDN controller when needed.

Such information might include widely shared subsets of a common information base, such as inventory, common service profiles, client associations, and client contexts. The SDN controller may update such common information, and may require transactional interfaces to them to assure consistency. The SDN controller may also need to subscribe to change notifications from such external information.

Information to support mappings between client and server views of resources and services may reside externally to the SDN controller. Reasons for external hosting may include security aspects of read-write access shared between the server (provider) administration and various clients. Support for roaming or multiple access services may also require that parts of associations, client contexts and service contexts reside centrally, from where they can be dynamically downloaded onto SDN controllers as needed.

Backup and restoration datasets are additional examples of pertinent external information stores, as are logs and performance monitoring archives, and software management repositories.

Other FCAPS functions

An SDN controller is responsible for establishing and maintaining services for its clients. A number of additional functions fall in the FCAPS definition (fault, configuration, accounting, performance, security), some of which may be partly or entirely outside the scope of a given SDN controller implementation.

A prime example is the exception thrown by an SDN controller when it is unable to converge desired state to actual state. This may trigger diagnostic, fault isolation, and repair activity that is not part of the SDN controller's normal functions. Similarly, functions such as equipment installation may be beyond the normal scope of an SDN controller's duties and capabilities.

The reason for a possible non-CPI interface to the SDN controller is that such activities typically involve the SDN controller, at least to the extent of overriding its normal operation in relation to certain resources, and often as proxy for external activities such as testing or diagnosis.

> Note – Most of these examples imply the possibility of human involvement in greater or lesser detail. Depending on circumstances and the SDN controller's capabilities, an ordinary administrator role at an A-CPI may suffice.

Restoration from backup and software upgrade are related functions that may need to be performed on SDN controllers that are at least partially operational during these processes.

Further reading: 5.4.3 Resource data base, B.3 Realization considerations, B.4 Initialization, B.6 Persistence

8.2 Notifications

Notifications refer to autonomous messages that provide information about events, for example, alarms, performance monitoring (PM) threshold crossings, object creation/deletion, attribute value changes (AVC), state changes, etc. Notifications are an important link in the feedback mechanism that is the core of an SDN controller. All SDN entities are expected to follow a publish-subscribe notifications model. Subscriptions and profiles are associated with particular client logins and are persistent.

A client context includes a definition of the notifications available to that client, and is thus part of the virtualization of underlying resources configured by the server administrator.

The notifications available to any given client must be discoverable by that client, and notifications management capabilities must be supported. These may include subscription management and, if alarms are subscribed, severity assignment profiles, active alarms lists, and alarm reporting control (ARC).

Notifications from an underlying layer may be of interest to clients, but must be virtualized and mapped into the client context. This may require more than identifier translation. For example, an attribute value change notification from the underlying resources may become an object creation notification to a client. Notifications delivered to a client should also be available to the administrator or a log for troubleshooting, correlated with the triggering notification from the infrastructure.

Particularly because a client need not maintain a continuous management-control session with the SDN controller, client-visible notification logs may also be required.

The administrator role can control notification subscriptions on behalf of itself or any of its clients. This is recursive to a client that plays the administrator role in its own client context.

ITU-T M.3702 [12] is the recommended baseline for notifications management. Further detail of the SDN notification framework appears in a Notifications Framework document, in progress.

Further reading: 6.1 SDN controller as feedback node, 6.6 Client context, 6.7 Multiple client management-control sessions, B.2 Identifiers

8.3 Peer controllers

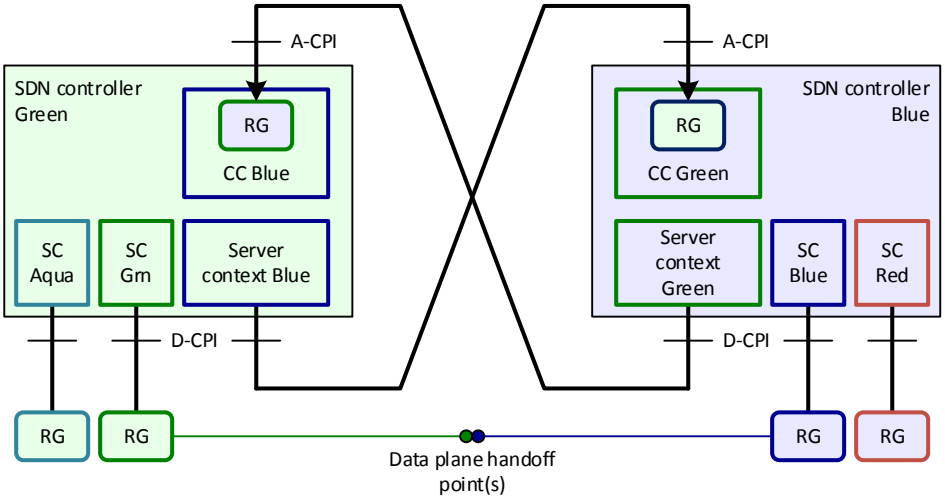

Figure 13 – Peers as symmetric requestors and providers

Figure 13 illustrates two SDN controllers in a peer relationship, in which either may act as client to invoke services from the other as server. Each requires a client context and a server context for the other. SLA, enforced policy and information hiding are applicable between peers just as between hierarchical provider and customer. The client context and server context for a peer may be separate, as shown, but they could combine certain aspects such as association credentials and security policy.

Further reading: 6.6 Client context, 6.9 Server context, 9.1 Security, A.8 Policy

9 Specific perspectives on the architecture

9.1 Security

Independent of SDN or other management-control paradigms, the data plane faces the same security issues. Many other operational concerns are also the same as in current networks, for example the procurement and deployment of executable software, the security of the management communications network, the integrity of directories and network services such as DNS, and the need to safeguard against insider errors and attacks.

What is different in SDN is the widespread exposure of management-control interfaces to third parties, and the wide range of services that may be offered by way of these interfaces. With more parties involved across more interfaces that offer greater functionality, the difficulty of assuring proper behavior across each interface increases manifold, whether misbehavior is due to error or malice. Interfaces traceable to a common information model are an important way to ensure API integrity from the beginning. Refer to [13] for additional discussion of some of these aspects.

The SDN architecture specifies that a management-control session between a client application and an SDN controller be properly authenticated and secured. Not least because a client application may not be fully trustworthy, the architecture provides for limited information exposure by

way of the virtual resources offered to the client, and policy enforcement on all of the client's management-control operations. Access security, tailored virtual resources and policy enforcement are essential parts of the client context. Security in the context of intent NBI is further discussed in [5].

Flaws in the client context may result from software or configuration error, including errors in mapping intent from client to server models, from insider tampering, or from network environment shifts that expose new possibilities beyond those covered by policy rules. Probing and auditing of client contexts may be valuable in discovering such flaws, with audit logs available for post facto analysis.

It may also be permitted that a client download executable code onto an SDN controller. Even if such code is certified and secured, it behooves the SDN controller to bound its execution environment within at least the same client context, information hiding, and policy enforcement constraints.

Possibly as a new, or exaggerated, feature of SDN, backups from SDN controllers and applications may contain commercially sensitive information, for example about customers and partners, and must be protected from leakage, not only during storage but also upon re-installation or subsequent analysis. Historically, much of this information would have been localized in semi-isolated BSS/OSS environments.

Further reading: 5.4.1 Information model, 6.6 Client context, A.8 Policy, B.6 Persistence

9.2 Migration and coexistence

When new resources are developed and installed, for example virtual network functions or network services from concepts developed by ETSI NFV ISG, it is natural to deploy them in an SDN domain and use them in alignment with this SDN architecture. However, the broad scope of the SDN architecture and its abstract nature are also intended to facilitate incremental adoption of SDN, and coexistence with current networks.

An SDN controller may be instantiated above a non-SDN domain, abstracting the views presented by non-SDN network elements, element management systems (EMSs), or network management systems (NMSs) into a common information model that can be virtualized and used to deliver services to clients.

A subnetwork, which may be as small as a subset of a single network element, may be an SDN island domain, in which the SDN controller adapts the given subnetwork resources to control and management by non-SDN entities, be they peer or superordinate entities. An SDN controller may act as a peer in multi-domain information exchanges such as via border gateway protocol (BGP).

An SDN controller may have responsibility for certain aspects of resource control, but not others. In the near term, much existing FCAPS and OSS/BSS functionality will remain with current OSS/BSS/NMS entities. When distinct entities control different aspects of the same resources, it will be important to ensure that they do not disrupt each other.

In short, SDN principles can be applied incrementally to peer, subordinate, and superordinate entities, as opportunity and business justification arises, and need not be viewed as all or nothing, or as a strong push into asset replacement.

Further reading: 5.3.1 Service-oriented model, 5.3.2 Resource-oriented model, 5.4.1 Information model, 8.1.2 Additional interfaces

9.3 Relationship of SDN and NFV

The ETSI NFV ISG focuses primarily on the creation and management of VNFs. SDN can help organize VNFs into NFV network services (NS). VNFs and NSs then become resources for use in constructing and dynamically optimizing services for clients, which is the focus of SDN. To a significant extent, these disciplines thus complement each other. The SDN architecture fully recognizes the value in the work done in ETSI NFV ISG, and encourages its use.

ONF TR-518 [2] is a more detailed comparison of SDN and NFV perspectives. ETSI NFV ISG has also published a less abstract view of the topic [9].

It may be expected that all OSS/BSS/NMS/EMS and SDN controllers will be or become VNFs, along with utility services such as DNS, DHCP, etc. These are already software entities, and may already be able to run on x86 cores, so the migration should be straightforward. But much or all of the "data plane" exposed by these entities is network management-control traffic, so the levels of abstraction will need to be carefully considered.

Further reading: 5.3.2 Resource-oriented model

Appendix A. Discussion of definitions

This appendix contains informative material of sufficient interest to warrant inclusion in this document.

A definition should be concise and precise. However, informative material is often helpful in interpreting the definition, drawing out its implications, and relating it to other concepts. Clause 4 contains the definitions used in this document; this appendix contains their discussions.

A.1. Abstraction

Definition: The representation of an entity or group of entities according to some set of criteria, while ignoring aspects that do not match the criteria.

Discussion: Even when the underlying entity is completely physical, it is always viewed through a filter of some sort. The filter necessarily loses some aspects of the original and thus necessarily provides an abstract view. Some filters are natural, for example human visual or tactile perceptions, but those of interest in SDN are software representations chosen to expose aspects of interest. Any number of filters (abstractions) can exist for a given underlying entity. Abstractions can be aggregated and/or further filtered to create additional abstractions.

It is sometimes considered that, at the bottom of a recursive hierarchy, the lowest level SDN controller controls a bare metal switch, for example through OpenFlow. It must be understood that, even in this case, the hypothetical bare metal switch necessarily exposes abstractions, namely one or more OpenFlow logical switches, upon which OpenFlow commands act.

A similar case can be made about any other management-control interface, and about even further descent into the internal structure of managed entities.

The inclusion of groups of entities in the definition is recognition that entities may be merged or combined in an abstraction.

A.2. Association

Definition: The information needed by two contracting entities as a precondition to establishing management-control sessions between each other.

Discussion: A typical association would include mutually agreed security credentials and corresponding local policy regarding acceptable session encryption characteristics.

Such an association may be established by offline negotiation and installed manually by administrators of the entities to be associated. An association may also be bootstrapped from almost nothing. An example is the willingness of a web portal to accept an anonymous session from any other entity, during the course of which the necessary identity, security and policy information can be negotiated. Upon satisfactory completion, the resulting association would allow subsequent sessions to perform a far wider scope of operations, but only between entities included by the association agreement.

Any entity that is willing to support sessions would normally offer an “almost nothing” association, sufficient to recognize and respond to session proposals from other entities. Such proposed sessions would then be mapped into existing associations, or into the bootstrap association, if supported.

A typical association, as described above, may also be augmented by the definition of roles and logins, with different profiles for different logins (profiles being understood as open to any interpretation).

An association allows for management-control sessions between any pair of devices that satisfy the security criteria. The security criteria may be restricted to specific devices, e.g., by including MAC address or SIM card credentials.

The difference between an association and a management-control session is that an association represents the potential for communication, while a session represents an open communications channel between two entities subsumed by the association.

A.3. Client

Definition: An entity that receives services from a server.

Discussion: Refer to clauses 5.3.2 Resource-oriented model, 5.4.2 Resources and resource groups, A.11 Server

A.4. Client context

Definition: The conceptual component of a server that represents all information about a given client and is responsible for participation in active server-client management-control operations.

Discussion: Refer to clauses 6.6 Client context and 6.7 Multiple client management-control sessions

A.5. Domain

Definition: A grouping of entities according to some set of criteria.

Discussion: Common domains of interest to SDN include

- Administrative domains, the complete set of resources owned or controlled by a given business entity.
- Geographic domains, partitioned within an administration for scaling, or across administrations for other reasons, for example the North American [telephony] Numbering Plan.
- SDN control domain, the set of resources directly controlled by a given SDN controller.
- Technology or deployment domains, for example transport, access, wireless, cloud. NFV may be a separate domain. Non-SDN domains would typically be distinguished from SDN domains.

A.6. Management-control continuum (MCC)

Definition: The principle that the functions of management and of control are largely, if not entirely, the same.

Discussion: Ultimately, the existence and behavior of networks can be traced to decisions and actions by humans. Activities performed by humans are often thought of as management. The further away from human involvement, the more likely an activity is thought of as control, perhaps exemplified at the extreme by an automatic gain control built into a hardware feedback loop. The idea of human involvement implies that feedback in a management context involves human action, which is to be avoided as much as possible. Automated feedback is preferably described as control.

In the continuum, a great many activities may be performed at a greater or lesser remove from specific human involvement. In addition, to a managed or controlled target entity, it makes no difference why it is being asked to perform some action, or by whom. A practical consequence is that management-control functional components may be re-used in products or deployments that are thought of as managers, as controllers, or anything in between.

In SDN, both terms may be used where their functions align with common usage, but *control* is the preferred default term, as it better matches the idea of feedback in real time.

> Note – It is observed that the resource life cycle maintenance perspective of the ETSI NFV ISG causes it to use the term *management* by default.

Further reading: A.8 Policy

A.7. Orchestration

Definition: The ongoing selection and use of resources by a server to satisfy client demands according to optimization criteria.

Discussion: Refer to clause 6.2 Orchestration

A.8. Policy

Definition: An administrative rule or set of rules that specifies the action(s) to be taken when specified condition(s) occur.

Discussion: The term *policy* is widely used but rarely defined. In some contexts, any rule is a policy, including for example, a match-action entry in a switch. In this architecture, a policy is understood to focus on management (human) purpose.

It is convenient to describe two categories of policies, either governing expected behavior or responding to exceptions. An administrator may choose how extensively to populate either category, depending on the perceived adequacy of default behavior and (for exceptions) the assessed probability and consequences of exceptions.

> Expected behavior policies may be exemplified by rules that govern traffic isolation between clients, for example by L0, L1, L2, L3 separation, and with or without management pre-assigned segmentation of the spaces involved.
>
> Exception behavior policies may be exemplified by rules to respond to client requests that cannot be served within an existing resource commitment. Choices might include suitable combinations of: silent discard of such requests, deny responses, with or without notification to the server administrator, logging, claiming additional resources or creating new resources to satisfy the requests, along with an accounting entry for billing, etc. Back-pressure on traffic sources could be a policy response to actual or potential resource congestion or overload.

Clearly, the scope of a policy can be as global or as granular as may be needed. Any given platform (SDN controller) must have sufficient scope to observe the policy criteria and execute the policy outcome. This may require policy evaluation across a number of coordinated subsystems in a separate policy decision point (PDP). In that case, the individual nodes might have policies directing them to generate notifications or queries when various observable circumstances arose locally, and then wait for a decision from the PDP.

Further reading: A.6 Management-control continuum (MCC)

A.9. Recursion

Definition: The repeated application of a pattern in which the input to each iteration is derived from the output of the previous iteration.

Discussion: As used in the SDN architecture, recursion refers to the management-control relationship between clients and servers, in which a client consumes resources and services offered by a server, which may itself be a client of one or more additional servers.

Although neighboring entities must support a mutually agreeable interface, it is not implied that the same interface need be supported by other recursively related pairs of neighboring entities, nor that the detailed functionality of one entity be the same as that of another.

Because the A-CPI and D-CPI expose only the immediate neighbors of a given entity, an entity cannot determine whether it is in a recursive relationship, where it is, or how deep the recursion may be. Indeed, an entity with multiple A-CPIs and D-CPIs may fit at different levels into a number of recursions of differing depth.

> Note – Special characteristics of an interface may provide clues, for example visibility of bits in registers that may be understood to be very close to hardware.

Mathematical or computational recursion is recognized, but differs from SDN management-control recursion inasmuch as iteration normally occurs within a single process on a single platform, the data type is strictly constrained (e.g., floating point real), and the result is often produced by the final iterative step.

In the context of layered transport networks, the principle of recursion has been key in assuring that network patterns and structure are not obscured in a cloud of complex relationships. Layering and partitioning represent the functional and topological dimensions of recursion. The hierarchy of transport network layers is consistently described via the recursive client/server paradigm, employing the recursive adaptation of characteristic information. Partitioning (within a layer network) creates a hierarchy of abstractions via recursive tailoring of the amount of topological detail that may be viewed.

The SDN architecture identifies two kinds of management-control recursion that are significant.

- Hierarchical recursion is a pattern in which higher-level SDN controllers orchestrate a broad scope of resources and services across one or more lower-level SDN controllers with narrower scopes and less abstract resources. In the inverse direction, resources exposed at lower levels of abstraction are recursively partitioned and recombined into increasingly abstract resources and services at higher levels.
- Neighbor recursion is a pattern in which SDN controllers peer to deliver services across SDN control domains. All participants would be expected to expose comparable levels of abstraction and service, and any SDN controller could act as either client or server to its neighbors, ad hoc. Services invoked by neighbor recursion are more likely to fit a call model – service invocation, service usage, service release – rather than persisting indefinitely.

A.10. Resource

Definition: Anything that can be used to deliver a service.

Discussion: A resource is modelled as an instance of a managed object class. Resources may be subdivided and combined with other resources in any way supported by the information model. When underlying resources are virtualized by an SDN controller, the result is a new resource that may be offered to a client. When exposed resources are manipulated by a client, possibly with additional resources as marshalled by the server, the result is a service.

Resources are intentionally defined very broadly, in keeping with the broad scope of SDN. The following examples illustrate the range of possible resources:

- A dark fiber
- An equipment plug-in
- A termination point
- A virtual machine
- A virtual network function (VNF)
- A switch
- A firewall
- A subnetwork
- A catalog of service offerings

Refer to clauses 5.3.2 Resource-oriented model, 5.4.1 Information model, 5.4.2 Resources and resource groups, 6.3 Virtualization

A.11. Server

Definition: An entity that provides services to a client.

Discussion: The SDN architecture is extensively based on client-server relationships in management and control. A server, which may be an SDN controller, exposes resources to zero or more clients, which may themselves be SDN controllers or service-consuming applications. The clients invoke various operations on these resources to realize services.

Especially when client and server are in separate administrative domains, the server may be called a provider, and the client may be called a user, customer, or consumer (note). A client may also be called an application.

> Note – The industry sometimes uses the term *tenant* in this context. However, the counter-party to a tenant is a landlord, which is usually inappropriate for an SDN context, so the term is deprecated in this architecture.

Refer to clause A.3 Client

A.12. Server context

Definition: The conceptual component of a client that represents all information about a given server and is responsible for participation in active server-client management-control operations.

Discussion: Refer to clause 6.9 Server context.

A.13. Service

Definition: The delivery of value for some time interval by a server to a client.

Discussion: It is understood that a relationship usually involves a mutual exchange of value, for example, but not necessarily, financial compensation. Prepaid services notwithstanding, a service invocation is generally the acceptance of an offer, with compensation to be delivered later. A given set of entities may offer and invoke services from each other ad hoc. There need not be a fixed server-client relationship.

A service continues to exist during intervals when there is no management-control session between client and server entities. Services such as VoIP may be quiescent during such intervals; other services such as transport may include continuing data-plane activity. In the case of roaming and mobile services, the physical point of attachment may vary over time, even across administrative domains.

In many cases, the actual delivery and use of the service happens across a data plane interface (see figure 1). In such cases, the management-control interface to the SDN controller (A-CPI) is used to invoke, control, modify, and terminate the service, but not to receive the value delivered by the service.

A.14. Service context

Definition: The conceptual component of a client context that represents all information about a given service.

Discussion: Refer to clause 6.8 Service context.

A.15. Virtualization

Definition: The abstraction of particular underlying resources, whose selection criterion is the allocation of those resources to a particular client, application, or service.

Discussion: Refer also to clause 6.3 Virtualization.

It must be understood that virtual has a number of meanings in the industry, sometimes implying only a software implementation on general-purpose computing hardware. This approximates its meaning in ETSI NFV [2], [8].

The SDN meaning is independent of underlying implementation. Further, an SDN virtualization may be several steps of abstraction away from any implementation. These steps of abstraction may subdivide and combine underlying implementations in arbitrarily complex ways. As an example, a VPN may abstract “real” layered networks from a number of administrative domains, with some topology visible and tunnels to conceal further underlying connectivity, and with reserved or on-demand capacity on its various links.

Appendix B. Operational considerations

This appendix contains informative material of sufficient interest to warrant inclusion in this document.

B.1. Reliability and availability

In the data plane, reliability and availability considerations are largely unchanged when management-control is logically separated from traffic processing infrastructure. Protection switching and load sharing concepts remain applicable, and the ability of an SDN controller to delegate functions into the infrastructure helps to avoid unacceptably slow responses to events.

Although SDN does not mandate a move of existing hardware functions into software, such a move is facilitated by SDN, and in particular NFV, and is under way today. From the reliability and availability viewpoint, a network implemented in software, as distinguished from a network controlled by software, requires re-examination.

- Software data plane functions may be more finely decomposed and therefore require more network assets, and more dynamic network assets, to chain them together, than was the case before.
- The dynamics of failure discovery may differ. For example, loss of signal on a physical link can be signaled immediately to concerned parties, while loss of heartbeat may be recognized only after several heartbeat intervals elapse. This will affect the agility of response, the proper kinds of response, and the commitments that can be made to customers.
- ETSI NFV virtual network functions reside on general purpose substrates. New instances may be readily created as an alternative to repair of a failed instance, or hot spares may be available for immediate use, subject to correspondingly agile updates to connectivity.
- Because failures may lose state, restoring state from checkpoints or continuously synchronizing spare instances will be important.

As to control, the possible separation of an SDN controller from traffic processing resources implies a reduction in availability. This may be addressed in several ways.

- Traffic processing engines may need to continue their function unchanged in the absence of an active controller connection, or may need to shut themselves down gracefully. This will affect partitioning and delegation of functionality from the SDN controller into data plane elements.
- SDN controllers may be implemented in load-sharing or redundant configurations, with near real-time synchronization. Subordinate systems may need to support multiple controller sessions or fast establishment of new controller sessions.

The notion of recursion implies that a service request may need to propagate through a number of hierarchical or neighbor SDN controllers before it is satisfied. If a controller fails or is otherwise unable to perform its expected functions, it will be necessary to unwind intermediate resource commitments, either for a new attempt or for a failure indication to the originating client.

It may be helpful to re-use concepts such as resource reservation in the forward direction and commitment in the reverse, or acknowledgement, direction.

Further reading: 6.5 Delegation, 6.7 Multiple client management-control sessions, B.6 Persistence

B.2. Identifiers

The foundation module of the core model fragment of the ONF common information model [3] includes material on identity, names, labels, and addresses. This architecture document refers to them generically as identifiers. Identifiers are essential to determine what is under consideration and where it is to be found, both of these within some space that may be implicit.

In the SDN architecture, identifiers are significant because clients view resources and services in their own terms, which must be mapped into the server's identifier space, as well as possibly translated into different object classes.

It is not specified how identifiers in either space are initially established.

- They may be administratively configured and known by manual provisioning, discovery, directory query or other means. Recognition of a wireless client by SIM card is one example; another is translation from a customer's service address to a provider's equipment or cross-connect panel appearance.
- They may be proposed by one side and accepted by the other during negotiation.
- Other mechanisms are not precluded.

When a client's virtual resource is derived as a one to one view of some underlying resource, a simple identifier mapping suffices. A virtual resource may also be based on a complex view of several underlying resources. As an example, a single subnetwork identifier exposed to a client may map onto a set of underlying resources that represents arbitrary functions, technologies, administrations, geographic locations, etc. Creating and traversing such mappings are major functions of virtualization and orchestration. Because the orchestration function generally has a choice of underlying resources, the mapping may change over time.

Further reading: 5.4.1 Information model, 6.2 Orchestration, 6.3 Virtualization, 6.6 Client context

B.3. Realization considerations

The administrator role creates and modifies client and server contexts and the SDN controller's optimization criteria. Any or all of these could be done through offline tools, with the result installed in the SDN controller as a subsequent step.

An SDN controller requires a significant amount of persistent information, particularly in client and server contexts. Backup and restoration may be appropriate for such information. It will be important to understand the primary source of each fragment of persistent information and to ensure timely consistency in mirrors, copies or derivations. These issues exist in current networks; whether there are new aspects in an SDN is a matter for further study.

Given suitable definition of which instances of client application are permitted to connect with which instances of SDN controller, at least part of the client context may be amenable to common external storage, accessible through a directory service, such that it can be downloaded to any of several possible SDN controllers included in the association with the given client. As an example of this flexibility, if a roaming user were to connect to a foreign SDN, a partial client context for that user could be dynamically downloaded and installed on the SDN controller from the user's home location, whereupon the SDN controller could dynamically instantiate services as specified by the included service contexts.

An SDN controller may be implemented in a distributed computing form, with redundancy or functional separation or both. Maintaining synchronized state during operation and during exception recovery is important.

As a new network entity of arbitrary and varying complexity, it may be expected that an SDN controller will very often be implemented as a virtual network function (VNF). If the SDN controller has responsibility within the same NFV domain, it will be important to plan a controllability tree that avoids the SDN controller unintentionally disrupting its own operation.

Further reading: 5.1.2 Logically centralized control, 5.2 Roles, 6.6 Client context, 6.9 Server context, 8.1.2 Additional interfaces, A.2 Association, B.6 Persistence

B.4. Initialization

The preparation of network resources for service requires operations such as code installation and configuration of initial parameters. Ultimately, these may come down to craft terminals and faceplate connectors, factory-installed software that is at least capable of downloading and installing operational software, and the like. (Some equipment may be able to configure itself and appear autonomously to its neighbors as a new resource without manual intervention.) Certain levels of test and equipment management may also be required during initial turn-up. These functions are not new to SDN. It is for further study whether the SDN architecture has anything special to say about them.

The SDN architecture presupposes an initial SDN controller environment that includes working software with an administrator association and client context. The administrator client context permits the creation of additional client and server contexts. As the administrator creates server contexts and the controller and corresponding underlying servers establish management-control sessions, the resources exposed by those servers are made available in the RDB, from which they may be statically allocated by the administrator to particular client contexts or consumed dynamically by the orchestration function.

> Note – Architecturally, transparent pass-through of underlying resources is thought of as a degenerate case of virtualization. Just as with similar situations in information hiding, strong security, and policy enforcement, transparent virtualization may explain why many existing approaches to intra-administration SDN do not recognize the need for these functions.

Given the necessary physical or virtual raw material, an SDN controller may instantiate or cause the instantiation of, new resources from scratch, e.g., VNFs on virtual machines (VMs) or on bare metal compute servers.

Further reading: 5.4.3 Resource data base, 9.3 Relationship of SDN and NFV, B.6 Persistence

B.5. Complexity

Orchestration, the core function of an SDN controller, is a multi-dimensional real-time optimization problem over a complex and potentially large space. Further, it must simultaneously work across the underlying resources and all of their various virtualizations. The complexity of the controller is a legitimate concern. Several approaches are available, some of which are listed below. These and others are likely to be useful in various combinations and according to circumstances.

- Continuing improvements in computing technology assist in the ability of an SDN controller to handle larger workloads, especially when the controller is implemented as a VNF that can be distributed, scaled, or migrated.
- Subdividing the problem space is a common scaling technique, whether by defining a greater number of smaller SDN domains or by allocating separate functional areas to separate SDN controller components. Existing BSS/OSS/NMS functionality may be re-used en bloc or incrementally.
- An important opportunity for separation of concerns is the ETSI NFV focus on life cycle maintenance of certain resources, and the SDN focus on services and the use of resources for service delivery. To illustrate the point, a soft switch may be a virtual network function (VNF), created by NFV entities and made available as a network node to an SDN controller. The SDN controller would then dynamically configure layering and forwarding of particular flows or flow bundles through the switch.
- The optimization problem may be simplified if the SDN controller's feedback criteria define a range of results that is considered to be good enough, rather than seeking to converge on a precise optimum. It may both simplify the problem and improve customer quality of experience if resources are not reallocated while they are in use.
- Although its functionality is reduced, an SDN controller is simplified if it deals only with the allocation of pre-existing resources, and does not consider the possibility of scaling resources or creating new resources. In such a case, some other entity (e.g., related to ETSI NFV) might have responsibility for dynamic resource inventory, updates to which could be made known to the SDN controller.
- The architecture assumes that some part of the underlying resources can be efficiently virtualized into disjoint subsets that satisfy the needs of various clients, including QoS and availability commitments, while isolating each from the others, and instrumenting the lot for monitoring, fault management, billing, and network engineering. The virtualization is assumed to remain optimal, or near optimal, in the face of customer churn, network build-out, and other such factors.

  Some aspects of this problem are comparatively easy to address, for example isolating customers in a common address space, or the policing and prioritizing of traffic near the point of network ingress. Other aspects may be harder to automate.

- Whether intentional or otherwise, over-engineering reduces the complexity of resource management. This is particularly true in the core, where elephant services (few, large, long-lived) are worth managing explicitly, while mouse services are best treated statistically. The cost of under-utilized resources is visible, but when compared to the comparatively hidden cost of complexity, especially the development and maintenance of algorithms, a certain amount of over-engineering may well be cost-effective.
- As suggested, the greatest cost may be that of human effort in developing and – not to be underestimated – validating resource allocation and optimization algorithms. Machine learning or artificial intelligence may play an important role in this regard.

Further reading: 6.2 Orchestration, 6.3 Virtualization, 9.3 Relationship of SDN and NFV, A.6 Management-control continuum (MCC)

B.6. Persistence

Information has a useful scope and a useful lifetime. Information also exhibits primacy characteristics, in the sense that some information is primary and other information is derived.

> Note – Global business agreements (associations) exemplify primary information; the interpretation of a global business agreement into SLA or policy for a specific SDN controller or service exemplifies derived information. An accepted service request is primary, its implementation is derived. Except for planned resources that do not yet exist, it is widely accepted that the network itself is the primary source of resource state, from which additional information may be derived.

Policy for redundant and persistent information storage should trade off primacy and reconciliation against the cost of derivation. Factors to be considered include security, synchronization, storage, bandwidth, code, practices development, training, and administration. The choice of hosting site for redundant and persistent information must also consider the likelihood and consequences of various exception cases. For a given SDN entity, several policies and hosting sites may be appropriate for different classes of information.

It is expected that much of the content of client and server contexts and the provider's optimization policy will justify remote persistent storage and possibly local redundancy to reduce the effect of SDN controller failure. When several business entities are involved in a relationship, each bears primary responsibility for its own information storage, but nothing precludes one player from offering synchronization, backup and restoration services to other players.

Further reading: 5.4.3 Resource data base, 6.6 Client context, 6.7 Multiple client management-control sessions, 6.9 Server context, 8.2 Notifications, 9.1 Security, A.2 Association, B.1 Reliability and availability, B.3 Realization considerations, B.4 Initialization

Appendix C. Evolution from issue 1 to issue 1.1

This appendix contains informative material of sufficient interest to warrant inclusion in this document.

As described in clause 1, SDN architecture issue 1.1 clarifies and extends issue 1 [1]. Experience with issue 1 identified the need for clarification in a number of areas, including some of the terminology. A number of concepts evolved, and several new concepts were brought into the forefront of issue 1.1. The differences in terminology, presentation, and emphasis are sufficient to warrant an explicit comparison. Figure 14 is quoted from figure 3.3 in SDN architecture issue 1.

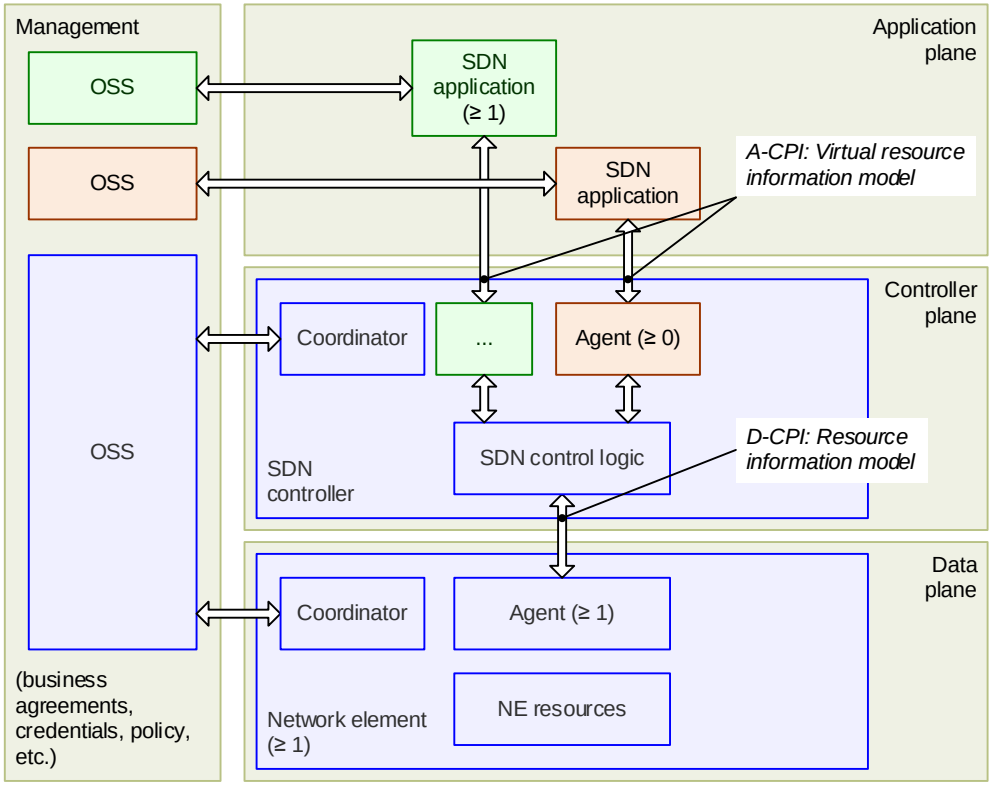

Figure 14 – Issue 1 architecture

Issue 1 shows a management block with a distinct interface and termination in the managed entities. Issue 1.1 incorporates the concept of management-control continuum (MCC), in which an administrator has greater scope and privilege than non-administrator roles, but is not fundamentally different. Both issues expect that the administrator would exist within the same business or trust domain as the entities being managed-controlled.

The data plane in issue 1 is described as a set of (virtual) network elements. The term *network element* has implications within the industry that may be misleading in some cases. In issue 1.1, the issue 1 network element is subsumed as one class of the *resource group* concept.

While issue 1 refers to resources in general, nothing more than a reference to (virtual) network elements is said about the way in which resources may be grouped. As a matter of clarification, resource grouping has been added to issue 1.1.

The agent in issue 1 is conceived as a window through which a remote SDN controller or application could manage its underlying resources. In issue 1.1, the term has changed to client context, and a great deal of its functionality has been explicated.

Suggested by arguments of symmetry, issue 1.1 introduces a server context concept, each instance of which is responsible for interaction with an underlying server instance.

Issue 1 identifies virtualization, orchestration and real-time responsiveness as attributes of an SDN controller, but the controller's role as the intelligence in a feedback loop is described as an option. Issue 1.1 recognizes that feedback intelligence is the essential characteristic of anything that purports to be an SDN controller.

Although it is not apparent from figure 14, issue 1 takes an omniscient view of recursion. Issue 1.1 recognizes the fact that individual entities see only their neighbors and have no visibility of possible further recursion. While the omniscient perspective is valid, the view from a single given entity is easier to relate to current open-source or vendor-specific product development.

Issue 1.1 introduces the idea that a client may have more than one service active simultaneously (hence the service context concept) and that a client may not have a continuous management-control session with the server during the lifetime of its services. Further, a client business may require server logins for a number of its employees. Sessions and persistence are needed for these and other reasons.

Issue 1 describes a resource-based SDN architecture. Work on intent interfaces has been synthesized into issue 1.1 as a complementary service-based view of the architecture. In the general case, the client and server are jointly responsible for developing mappings that translate between the client's frame of reference and the server's frame. The mapping allows for separation between standard information models and arbitrary client- or application-specific models.

Issue 1 implies that all of the resources used by a client are visible to the client for direct manipulation. Issue 1.1 recognizes that the client typically sees only a subset of its resources, often only its service endpoints, with all of the supporting resources allocated, either statically or dynamically, by the SDN controller.

Issue 1 refers to resources generically; issue 1.1 explicitly includes the contributions of the ETSI NFV ISG in offering virtual resources to SDN environments.

Further reading: 5.3 Service and resource oriented models, 5.4.2 Resources and resource groups, 6.1 SDN controller as feedback node, 6.2 Orchestration, 6.6 Client context, 6.7 Multiple client management-control sessions, 6.8 Service context, 6.9 Server context, A.6 Management-control continuum (MCC), B.6 Persistence

Appendix D. Back matter

This appendix contains informative material of sufficient interest to warrant inclusion in this document.

D.1. Acronyms

AAA	Authentication, authorization, accounting
A-CPI	Applications-controller plane interface
API	Applications programming interface
ARC	Alarm reporting control
AVC	Attribute value change (notification)
BGP	Border gateway protocol
BSS	Business support system
CFM	Connectivity fault management
CPI	Controller plane interface
CRUD	Create, read, update, delete
C-VID	Customer VLAN identifier
DBA	Dynamic bandwidth assignment
D-CPI	Data-controller plane interface
DHCP	Dynamic host configuration protocol
DNS	Domain name system
DSL	Digital subscriber line
EMS	Element management system
ETSI NFV ISG	European Telecommunications Standards Institute – Network functions virtualization – Industry specification group
FCAPS	Fault, configuration, accounting, performance, security
IP	Internet protocol
LLDP	Link layer discovery protocol
MAC	Medium access control
MCC	Management-control continuum
NAT	Network address translation
NBI	Northbound interface (A-CPI)
NFV	Network functions virtualization
NMS	Network management system
NNI	Network-network interface
NS	Network service
OF	OpenFlow
ONF	Open Networking Foundation
ONU	Optical network unit
OSS	Operations support system
PDP	Policy decision point
PM	Performance monitoring
PON	Passive optical network
PoP	Point of presence
QoS	Quality of service
RDB	Resource data base
RG	Residential gateway
SDN	Software defined networking
SDO	Standards development organization
SIM	Subscriber identity module
SLA	Service level agreement
SNMP	Simple network management protocol
STP	Spanning tree protocol
UNI	User-network interface
VM	Virtual machine
VNF	Virtual network function
VPN	Virtual private network

D.2. References

[1] ONF TR-502, SDN architecture, Issue 1, 2014
[2] ONF TR-518, Relationship of SDN and NFV, 2015
[3] ONF TR-512, Core information model (CoreModel), 2015
[4] ONF TR-522, SDN Architecture for Transport Networks, 2016
[5] ONF TR-523, Intent NBI – Definition and Principles

[6] ETSI NFV, Network Functions Virtualisation, An introduction, benefits, enablers, challenges and call for action, 2012
[7] ETSI NFV GS NFV-INF 001 V1.1.1, Network Functions Virtualisation (NFV); Infrastructure Overview, 2015
[8] ETSI NFV GS NFV 004 V1.1.1, Virtualisation Requirements, 2013
[9] ETSI NFV GS NFV-EVE 005 V1.1.1, Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework, 2015
[10] IEEE Std 802.1AB-2009, IEEE Standard for Local and Metropolitan Area Networks – Station and Media Access Control Connectivity Discovery, LLDP
[11] IETF RFC 3444, On the difference between information models and data models, 2003
[12] ITU-T Recommendation M.3702, Notification management – Protocol neutral requirement and analysis, 2010
[13] ONF TR-511, Principles and Practices for Securing Software-Defined Networks, 2015

D.3. Contributors

Malcolm Betts, ZTE
Chen Qiaogang, ZTE
Luis Miguel Contreras Murillo, Telefonica
Nigel Davis, Ciena
Paul Doolan, Coriant
Dave Hood, Ericsson
Chris Janz, Huawei
Kam Lam, Alcatel-Lucent
Li Fengkai, Huawei
Manuel Paul, DT
Lothar Reith, DT
Sibylle Schaller, NEC Labs
Fabian Schneider, NEC Labs
Stephen Shew, Ciena
Eve Varma, Alcatel-Lucent
Maarten Vissers, Huawei

