OPEN NETWORKING FOUNDATION

SDN architecture

Issue 1
June, 2014

Abstract

This document specifies the architecture of software defined networking (SDN). Based on an ONF introduction to SDN, it expands the principles of SDN and applies them to architectural components and interfaces. As a living document, it also identifies work for further study.

ONF Document Type: TR (Technical Reference), non-normative, type 2
ONF Document Name: SDN ARCH 1.0 06062014

Disclaimer

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

List of Figures

1 Scope

This document describes the SDN architecture. Its purpose is to guide further detailed activity in the various ONF working groups, while also serving as a reference for external communications from the ONF. The companion ONF Framework document (not yet published) describes what is desired. This document describes how this is to be achieved, at a high level.

The SDN architecture specifies, at a high level, the reference points and interfaces to the controller. The architecture describes a number of functions internal to the SDN controller and NE. Specific blocks that perform these functions are illustrated to aid the description, but are not per se required in an implementation. The interfaces to these internal functional blocks are not specified.

The specified behavior of the SDN controller or NE is confined to those aspects that are required to allow interoperable implementations to be deployed. The architecture is agnostic to the protocols across the interfaces (note).

> Note – Candidate protocols for various interfaces include OpenFlow switch (OFS) [2] and OF-Config (OFC) [3].

The SDN architecture allows an SDN controller to manage a wide range of data plane resources. A number of different data planes exist; SDN offers the potential to unify and simplify the configuration of this diverse set of resources.

The architecture also recognizes the reality that if SDN is to be successful, it must be deployable within the context of largely pre-existing multi-player environments, comprising many organizations or businesses, with the consequent need for policy and security boundaries of information sharing and trust. Real-world constraints include the need to co-exist with existing business and operations support systems, and other administrative or control technology domains. In less complex environments, such as limited scale enterprise networks, suitable functional subsets may be profiled from the architecture.

The SDN architecture recommends that common models and mechanisms be employed wherever possible to reduce standardization, integration and validation efforts. This also implies utilizing existing standards or accepted best practices where feasible.

A systems architecture partitions a complex system into modular parts, typically used to manage complexity, to allow for independent implementation and component reuse, or to meet other technical or business goals. However, there is no such thing as value-neutral design. The choice of component partitioning, which interfaces are defined, which protocols are open or proprietary, can have a profound influence on the types of services ultimately delivered to the end user [14]. Thus, an architecture necessarily makes choices; the choices and their rationale are presented in this document. This architecture contents itself with principles, rather than detail, expecting that clearly enunciated principles facilitate the myriad decisions required by working groups and implementers. At the same time, the architecture recognizes that SDN addresses environments sufficiently complex to require future extensions and clarifications. Implementation considerations are described, along with topics for further study.

Specific goals of this document include

a) Define an architecture for SDN.

b) Provide a foundation for information model development.
c) Describe entities in sufficient detail to permit the derivation of functions and interface definitions.
d) Provide high-level guidance and a framework for activities in the various ONF working groups.
e) Serve as a reference against which to discuss extensions, errors, omissions, and other changes that may be appropriate.
f) Aid in evaluating and comparing various approaches and solutions that claim to conform to an SDN architecture.
g) Facilitate SDN technical orientation for engineers, architects, and solutions specialists.
h) Offer sufficient value to be utilized across the broader SDN community.

2 Definitions, abbreviations, and conventions

2.1 Definitions

2.1.1 Terms defined elsewhere

This document uses the following terms defined elsewhere:

None

2.1.2 Terms defined in this document

This document defines the following terms:

Layer: a stratum in a framework that is used to describe recursion within the data plane. Adjacent layers have a client-server relationship.

Discussion: Forwarding in the data plane is described in terms of a stack of layer networks. The layer networks are related by adaptation (which may include multiplexing) and termination functions. The format of the data that is carried by each layer network is called its **characteristic information**. The definition of characteristic information includes the adapted user information, plus the overhead necessary to operate the layer network (for example, detection of errors or misconnections) and in the case of a physical layer, may include aspects such as wavelength and symbol coding. See *level*.

Level: a stratum of hierarchical SDN abstraction.

Discussion: This architecture uses *level* to sharpen the distinction between hierarchical abstraction and traffic signal adaptation. See *layer*.

Information model: a set of entities, together with their attributes and the operations that can be performed on the entities. An instance of an information model is visible at an interface.

Discussion: This architecture uses object-oriented terms to describe information models.

SDN controller: a software entity that has exclusive control over an abstract set of data plane resources. An SDN controller may also offer an abstracted information model instance to at least one client.

Discussion: The underlying resources are presented to the controller as an information model instance. The resources are manipulated by exercising the methods of the managed object (MO) instances.

An SDN controller may be implemented as any number of software components, which reside on any number of physical platforms. The distributed components are required to maintain a synchronized and self-consistent view of information and state. This requirement bounds the concept of a single SDN controller; software components that do not share this characteristic are necessarily external to the controller. (Initialization, resilience, synchronization delay considerations are internal to the SDN controller and are not part of this definition.)

Controller plane interface (CPI): the generic interface to an SDN information model instance. A CPI instance may be further specialized with a prefix character.

Discussion: An SDN controller supports three functional interface types:

- a **D-CPI** between data and controller planes, across which the SDN controller controls data plane resources,
- an **A-CPI** between application and SDN controller, across which an application receives services from the SDN controller,
- and a management interface, across which resources and policy may be established, as well as other more traditional management functions.

The same CPI may have different designations depending on the perspective of the viewer.

Any number of D-CPI instances may be supported by an SDN controller that associates with multiple data plane entities. Any number of A-CPI instances may be supported by an SDN controller in the service of multiple applications. There is one management interface. The specification of the D-CPI and A-CPI is independent of the characteristics of SDN controller distribution.

Additional interfaces are not precluded.

Network element: A group of data plane resources that is managed as a single entity.

Discussion: A network element (NE) provides a common name space used by the SDN controller to access resources that forward, manipulate or store user data.

The data plane perimeter of a NE is bounded by a set of external interfaces. These interfaces may be on physical or logical ports.

The NE provides at least one logical data-controller plane interface (D-CPI) that allows its functions to be managed and controlled by the SDN controller. The logical CPI may contain any number of communications channels or protocols.

This definition does not specify the geographical distribution of the resources that comprise an NE. A localized NE might be a self-contained shelf equipped with circuit packs, a small server, or a top-of-rack (TOR) switch. A distributed NE may be exemplified by a passive optical network (PON) access system. A virtual NE (a VM for

example) may be defined on some particular physical component, or may span a number of them.

An optical interface plug-in would normally not be considered to be an NE.

2.2 Abbreviations and acronyms

This document uses the following abbreviations and acronyms:

3GPP	Third Generation Partnership Project
ACID	Atomicity, consistency, isolation, durability
ACL	Access control list
A-CPI	Application-controller plane interface
AIS	Alarm indication signal
API	Applications programming interface
BFD	Bidirectional forwarding detection
BGP	Border gateway protocol
BIP	Bit interleaved parity
BSS	Business support system
C2C	Controller to controller
CCM	Continuity check message
CFM	Connectivity fault management
CPI	Controller plane interface
CRUD	Create, read, update, delete
DC	Data center
D-CPI	Data-controller plane interface
DDOS	Distributed denial of service
DNS	Domain name system
DOS	Denial of service
DPCF	Data plane control function
EMS	Element management system
ETSI	European Telecommunications Standards Institute
GMPLS	Generalized multi-protocol label switching
GRE	Generic routing encapsulation
HAL	Hardware abstraction layer
ICMP	Internet control messaging protocol
I-CPI	Intermediate-controller plane interface
IETF	Internet Engineering Task Force
IP	Internet protocol
IRTF	Internet Research Task Force
ISO	International Standards Organization
ITU-T	International Telecommunications Union – Telecommunication Standardization Sector
LAN	Local area network
LLDP	Link layer discovery protocol
MAC	Media access control
MEF	Metro Ethernet Forum
MEP	Maintenance association end point, Maintenance entity group end point
MO	Managed object
MPLS-TP	Multi-protocol label switching, transport profile
NAT	Network address translation
NBI	Northbound interface
NCD	Network control domain
NE	Network element
NFV	Network Functions Virtualization
NGMN	Next-generation mobile networks

NMS	Network management system
OAM	Operations, administration, maintenance
OFC	OpenFlow-Config protocol
OFS	OpenFlow-switch protocol
OIF	Optical Interworking Forum
ONF	Open Networking Foundation
OSS	Operations support system
OTN	Optical transport network
OVSDB	Open vSwitch data base
PCE	Path computation element
PCEP	Path computation element communication protocol
PEP	Policy enforcement point
PM	Performance monitoring
PON	Passive optical network
QoS	Quality of service
RDB	Resource data base
SDH	Synchronous digital hierarchy
SDN	Software-defined networking
SDNC	SDN controller
SDO	Standards development organization
SLA	Service level agreement
SNMP	Simple network management protocol
STP	Spanning tree protocol
TCA	Threshold crossing alert
TL1	Transaction language 1
TMF	TM Forum
TOR	Top of rack
VID	VLAN identifier
VLAN	Virtual local area network
VM	Virtual machine
VN	Virtual network
VNE	Virtual network element
WAN	Wide area network
WG	[ONF] working group

2.3 Conventions

An *abstraction* is a representation of an entity in terms of selected characteristics, while hiding or summarizing characteristics irrelevant to the selection criteria.

In this document, a *virtualization* is an abstraction whose selection criterion is dedication of resources to a particular client or application. When the context is general, for example when speaking of virtual network elements (VNEs), the term *virtual* may be used even when *abstract* might suffice. *Virtual* is also sometimes used colloquially to mean non-physical.

The architecture relies heavily on the *client-server* model, in which a higher, more general, or more abstract entity, the client, receives something of value from a lower, less general or less abstract entity, the server. In software terms, the client sends commands to the server, and the server sends responses and may send notifications to the client. In a business context, the client is the one who pays the bill. Both kinds of client-server relationship can be recursive through any number of middlemen.

At every client-server boundary, there is the possibility of an administrative client-server relationship, i.e., a trust domain boundary. Sometimes the term *provider* is used as the equivalent of *server*, and the terms *customer*, *tenant*, or *user* may be used as the equivalent of *client*.

Clients and servers may also be mentioned in discussing layer networks, especially to emphasize the common case when a layer characteristic information adaptation (e.g. adaptation between TDM and packets) coincides with an administrative or trust client-server boundary.

The architecture uses the term *layer network* (always qualified as server layer network, client layer network) to reflect recursion in the data plane ([5], [6]). Data, controller and application spaces are distinguished by the term *plane*, and recursive strata across these planes are called *levels*.

A *tunnel* is a server layer network connection that is visible to a client layer network as a link. The term *tunnel* is often used when traffic from a client layer network is handed off across a business boundary to a server layer network that transparently conveys the traffic to some distant set of client-visible termination points. The controller may also set up tunnels in its own control domain, for example by encapsulating Ethernet client layer network traffic into pseudowires. A group of tunnels is often referred to as an *overlay* network or an *underlay* network, depending on whether the perspective is from a client or server respectively. As with any other connection, a tunnel may include protection capability.

The architecture also employs the concepts of *controllers* and *agents*, in which the agent is responsible for carrying out the commands of the controller and notifying the controller of events that are specified by the controller. For clarity, and because its functional responsibilities are quite different, a component called a *coordinator* is responsible for acting on behalf of an SDN manager; the term *agent* is not used in this context.

The concept of *management* encompasses operations to support the infrastructure, for example operations between SDN manager and SDN controller or NE. This includes such classic functions as equipment installation and maintenance, and software upgrade. Management functions special to SDN include the allocation of resources and policy to particular SDN clients or applications, and the provisioning of information necessary to permit separated functional entities to communicate, for example NEs and SDN controllers.

Management functions may be performed by any number of entities, the details of which are out of scope of the SDN architecture. This document abstracts all management functions into a block called *OSS* (operations support system).

The concept of *control* encompasses operations performed by a client or performed by a server on request by a client, for example operations between SDN controllers and NEs or applications.

The SDN architecture specifies functional interfaces between software components, without constraints on physical location. In general, an applications programming interface (API) can be tunneled through a network protocol to support separation, while a network protocol interface can be omitted in case of co-residency, to expose an API. References to either APIs or protocols are understood to be typical or expected implementations, but not mandatory.

One such interface class exists between applications and SDN controllers. It is often referred to as a northbound interface (NBI) or northbound API, and conventionally carries the implication that service invocation flows south across the interface. The boundary between SDN controller and application is largely a matter of perspective (note). The SDN architecture uses the term application-controller plane interface (A-CPI) to designate this boundary, and avoids the northbound terminology.

Note – For example, a path computation engine (PCE) might exist

a) as a component of the controller, used by the controller itself, but externally invisible,

b) as a component of the controller, exposed as a service to be invoked by external applications, or
c) as an application, an external service to be invoked by a controller.

Throughout this document, colors are used to denote administrative (e.g., trust) domains. To fully clarify the security implications, each color may be regarded as a separate company. Each domain is named to match its color: Blue (usually shown as the network provider), Green, Red, etc.

Figure 2.1 illustrates the simple abstract network example used throughout the document. It could represent any kind of network, e.g., a transport network, a data center network. The network is owned by a provider designated and shown as Blue. Rectangles indicate network elements (NEs), whose identifiers imply nothing more than a drawing convenience. Lines between NEs designate links; open endpoints indicate data plane handoff points that are suitable for connection to network equipment outside the SDN controller's data plane domain. In a number of examples, the external ports are used by two clients, Green and Red, according to the endpoint color and labels.

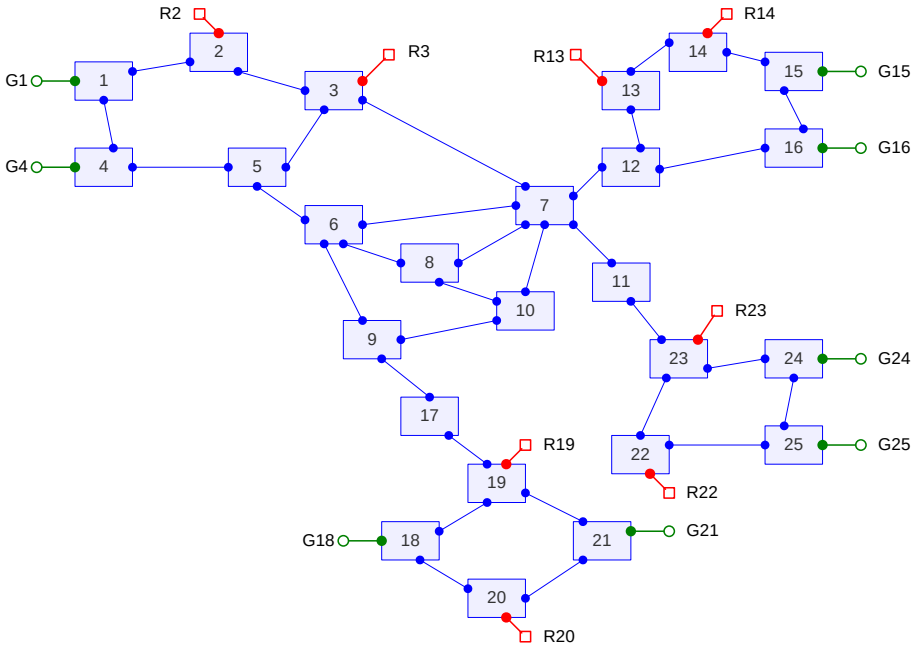

Figure 2.1 – An example abstract network

3 SDN overview

This clause describes the architecture in two ways. Clause 3.1 is a high-level descriptive overview, while clause 3.2 describes the essentials of the architecture as concisely as possible. The remainder of the document derives and explains the architecture, and expands on some of its implications.

3.1 Descriptive overview

The aim of SDN is to provide open interfaces that enable the development of software that can control the connectivity provided by a set of network resources and the flow of network traffic though them, along with possible inspection and modification of traffic that may be performed in the network. These primitive functions may be abstracted into arbitrary network services, some of which may not be presently apparent.

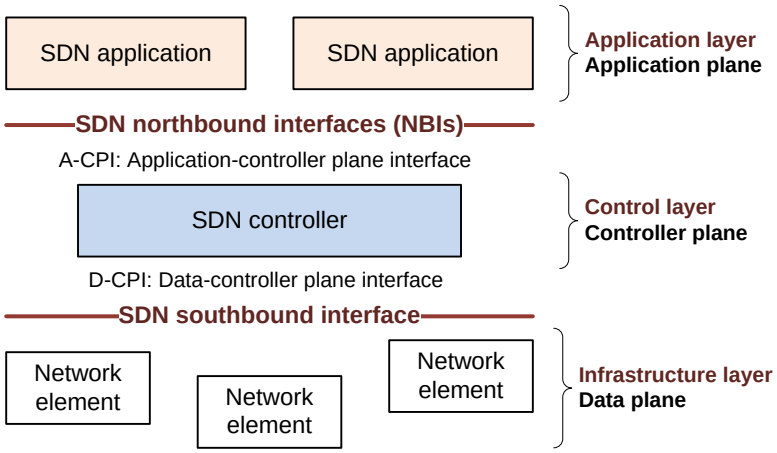

Figure 3.1 – Basic SDN components

Figure 3.1 introduces the basic SDN components, with terminology similar to that from the original ONF white paper, "Software-Defined Networking: The New Norm for Networks" [1]. The initial view comprised infrastructure, control and application layers (red text), which are designated in this architecture document as data, controller, and application planes (black text). The infrastructure layer (data plane, note) comprises network elements, which expose their capabilities toward the control layer (controller plane) via interfaces southbound from the controller. (In [1], this is called a control-data plane interface.) The SDN applications exist in the application layer (plane), and communicate their network requirements toward the controller plane via northbound interfaces, often called NBIs. In the middle, the SDN controller translates the applications' requirements and exerts low-level control over the network elements, while providing relevant information up to the SDN applications. An SDN controller may orchestrate competing application demands for limited network resources according to policy.

> Note – The concept of a data plane in the context of the SDN architecture includes traffic forwarding and processing functions. A data plane may include the necessary minimum subset of control and management functions.

This view requires further development and precision if it is to provide a rigorous technical SDN architecture that can inform technically versed network architects inside and outside of ONF. This architecture document therefore defines functions, interfaces and components, explains their relations and guides the development of information models, while not over-specifying.

Terminology modifications reflect the fact that some aspects of control inevitably reside in all layers, but the interface of interest is that between an SDN controller and its adjacent entities. The major horizontal groupings are called *planes* to avoid confusion with the term *layer*, which is used in the sense of layer networks, for example when packets are mapped to MPLS, further into Ethernet, and further into wavelengths.

With that in mind, figure 3.2 adopts the revised terminology and adds the management function, which is often omitted from simplified SDN representations. Although many traditional management functions may be bypassed by the direct application-controller plane interface (A-CPI), certain management functions are still essential. In the data plane, management is at least required for initially setting up the network elements, assigning the SDN-controlled parts and configuring their SDN controller. In the controller plane, management needs to configure the policies defining the scope of control given to the SDN application and to monitor the performance of the system. In the application plane, management typically configures the contracts and service level agreements (SLAs). In all planes, management configures the security associations that allow distributed functions to safely intercommunicate.

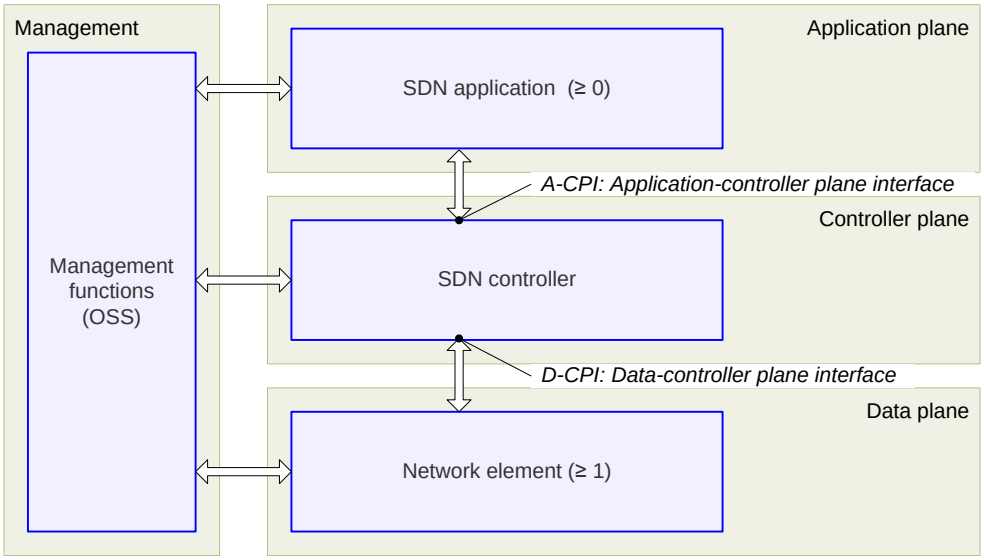

Figure 3.2 – SDN components with management

Figure 3.2 summarizes the SDN architecture, with the terminology and reference points used throughout the sequel. It shows distinct application, controller and data planes, with controller plane interfaces (CPIs) designated as reference points between the SDN controller and the application plane (A-CPI) and between the SDN controller and the data plane (D-CPI). The information exchanged across these interfaces should be modeled as an instance of a protocol-neutral information model.

While customer systems have historically interfaced the network indirectly, by way of the provider's business or operations support systems (BSS/OSS), SDN envisions that customer

applications may have dynamic and granular control of network resources through direct access to an SDN controller. Recognizing the likelihood of a business boundary between provider and customer, it is therefore essential that the architecture recognize a business or organizational boundary between the SDN controller plane and the applications that use it. Provider and customer exist in different trust domains.

This architecture document uses colors as a visual aid to emphasize trust domains. Blue is the default, and may be thought of as a network provider, while other colors, such as green and red, indicate customers, tenants, or even distinct organizational or application entities within the overall Blue trust domain.

Figure 3.2 thus shows only a single trust domain. Figure 3.3 extends the idea to show multiple trust domains. Each trust domain is understood to have its own management functionality. Trust domains may logically extend into components of other trust domains, as exemplified by the green and red agents in the blue SDN controller.

> Note – It is important to understand that code that executes in the red and green agent boxes in the controller plane would be installed and managed by the blue administration. This is the meaning of the phrase *logically extend*.

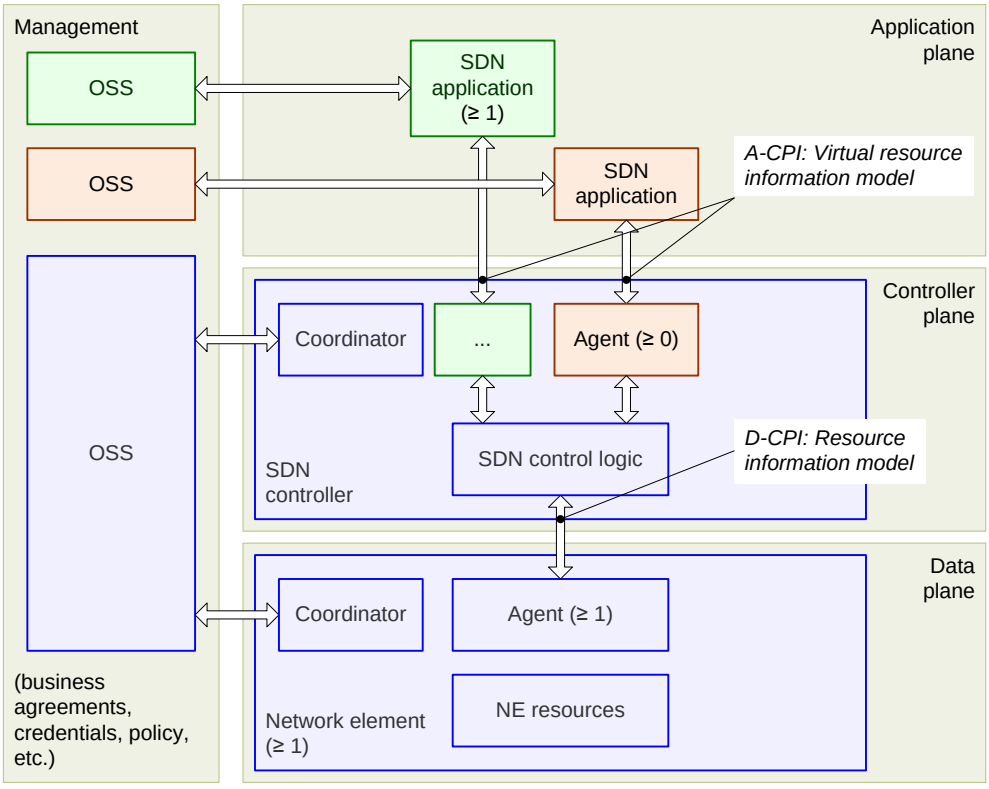

Figure 3.3 – SDN overview, with physical data plane

Figure 3.3 also shows agents and coordinators in the SDN controller and the network elements. The agents support the concept of sharing or virtualizing the underlying resources, for example, which network element ports are SDN-controlled (as opposed to hybrid or legacy ports), or the details of the virtual network that are exposed to the SDN applications, while isolating one

customer's service from another's. In the SDN controller, different agents may expose control over the network at different levels of abstraction (latitudes) or function sets (longitudes). It is the SDN control logic's task to map and arbitrate between the networking requirements from all SDN applications and translate them into instructions for the network element (NE) resources exposed through the NE agents. The coordinators in both the network element and the SDN controller install customer-specific resources and policies received from management.

Multiple agents may exist at the same time in any one network element and SDN controller, but there is only one logical management interface, and therefore only one coordinator per network element or SDN controller.

Clause 4 considers the meaning and implication of the SDN principles in further depth, and introduces the major entities whose functions and interactions comprise the architecture. Because the SDN controller is at the heart of the architecture, clause 5 further expands controller plane functions and interactions, while clause 6 describes implementation considerations.

The ONF SDN architecture is also summarized in a document entitled *SDN architecture overview* [4]. In the event of discrepancy between this document and the architecture overview, this document shall prevail.

3.2 Concise statement of architectural essentials

Figure 3.4 shows the major components and interfaces of the SDN architecture. The architecture makes no statement about the physical realization of the components.

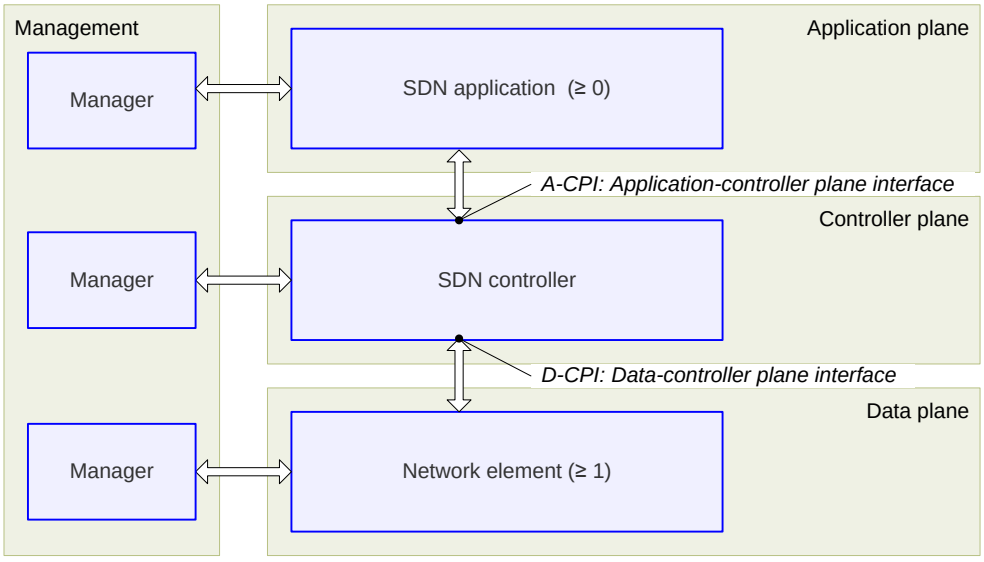

Figure 3.4 – SDN overview, with physical data plane

Data plane

The data plane comprises a set of one or more network elements, each of which contains a set of traffic forwarding or traffic processing resources.

Resources are always abstractions of underlying physical capabilities or entities.

Controller plane

The controller plane comprises a set of SDN controllers, each of which has exclusive control over a set of resources exposed by one or more network elements in the data plane (its span of control).

Clause 4.3.5 explains how resources can be shared on a best-efforts or first-come-first-served basis.

Additional interfaces to SDN controllers are not precluded.

The minimum functionality of the SDN controller is to faithfully execute the requests of the applications it supports, while isolating each application from all others.

To perform this function, an SDN controller may communicate with peer SDN controllers, subordinate SDN controllers, or non-SDN environments, as necessary.

A common but non-essential function of an SDN controller is to act as the control element in a feedback loop, responding to network events to recover from failure, reoptimize resource allocations, or otherwise.

Application plane

The application plane comprises one or more applications, each of which has exclusive control of a set of resources exposed by one or more SDN controllers.

Additional interfaces to applications are not precluded.

An application may invoke or collaborate with other applications. An application may act as an SDN controller in its own right.

Management

Each application, SDN controller and network element has a functional interface to a manager.

The minimum functionality of the manager is to allocate resources from a resource pool in the lower plane to a particular client entity in the higher plane, and to establish reachability information that permits the lower and higher plane entities to mutually communicate.

Additional management functionality is not precluded, subject to the constraint that the application, SDN controller, or NE have exclusive control over any given resource.

Administration

Each entity in a north-south progression through the planes may belong to a different administrative domain. The manager is understood to reside in the same administrative domain as the entity it manages.

ONF protocols

The OF-config protocol is positioned to perform some of the functions that are needed at the management interface.

The OF-switch protocol is positioned to perform some of the functions that are needed at the D-CPI and possibly at the A-CPI.

4 Principles and architectural components

This clause introduces the principles of SDN, and the functional entities and relationships that form the SDN architecture. Subsequent clauses expand on the introductory material to derive additional constituent entities and relationships.

4.1 Principles

The ONF high-level view of SDN is described in [1]. From this and other sources, several basic principles of SDN may be adduced. Their implications are briefly summarized here, and are expanded in detail in subsequent clauses.

- Decoupling of controller and data planes

  This principle calls for separable controller and data planes. However, it is understood that control must necessarily be exercised within data plane systems. The D-CPI between SDN controller and network element is defined in such a way that the SDN controller can delegate significant functionality to the NE, while remaining aware of NE state. Clause 4.3 lists criteria for deciding what to delegate and what to retain in the SDN controller itself.

- Logically centralized control

  In comparison to local control, a centralized controller has a broader perspective of the resources under its control, and can potentially make better decisions about how to deploy them. Scalability is improved both by decoupling and centralizing control, allowing for increasingly global but less detailed views of network resources. SDN controllers may be recursively stacked for scaling or trust boundary reasons, a topic described in clause 5.

- Exposure of abstract network resources and state to external applications

  Applications may exist at any level of abstraction or granularity, attributes often described as differing latitudes, with the idea that further north suggests a greater degree of abstraction. Because an interface that exposes resources and state can be considered a controller interface, the distinction between application and control is not precise. The same functional interface may be viewed in different lights by different stakeholders. Just like controllers, applications may relate to other applications as peers, or as clients and servers.

The principle of abstracting network resources and state to applications via the A-CPI allows for programmability of the network. With information about resources and their states, applications are able to specify requirements and request changes to their network services via the SDN controller, and to programmatically react to network states.

Further, the concept of hierarchically recursive application/controller layers and trust domains also allows application programs to be created that may combine a number of component applications into a more comprehensive service.

The SDN architecture clarifies the meaning and implications of these principles by identifying the basic functional entities and the information and operations that need to be exchanged over various interfaces among them. The architecture further decomposes these functional entities into a not necessarily comprehensive set of functional components.

This architecture incorporates the concept of trust domain boundaries, which is vital to widespread commercialization. The architecture defines components entirely within particular trust domains, with well-defined reference points to other trust domains. Strong abstraction barriers help to protect the commercial and business interests of stakeholders, while recognizing and accommodating widely varying trust relationships. The uniformity of the architecture also facilitates the design and audit of security measures.

The high-level model of all vertical SDN architecture interfaces is the exposure of an information model instance by a server to a client, upon which the client can perform create-read-update-delete (CRUD) and class-specific operations. This emphasizes the importance of a common information model throughout. From this perspective, the management function is responsible for instantiating information models and policies that define the capabilities exposed across interfaces between planes, especially across trust domain boundaries. Figure 4.1 illustrates the notion that the client-server (or controller-agent) model is applicable at as many levels of SDN controller hierarchy as may exist.

Hierarchical levels serve two purposes.

1. Scaling and modularity: each successively higher level has the potential for greater abstraction and broader scope.
2. Security: each level may exist in a different trust domain. The level interface is a standard reference point for inter-domain security enforcement.

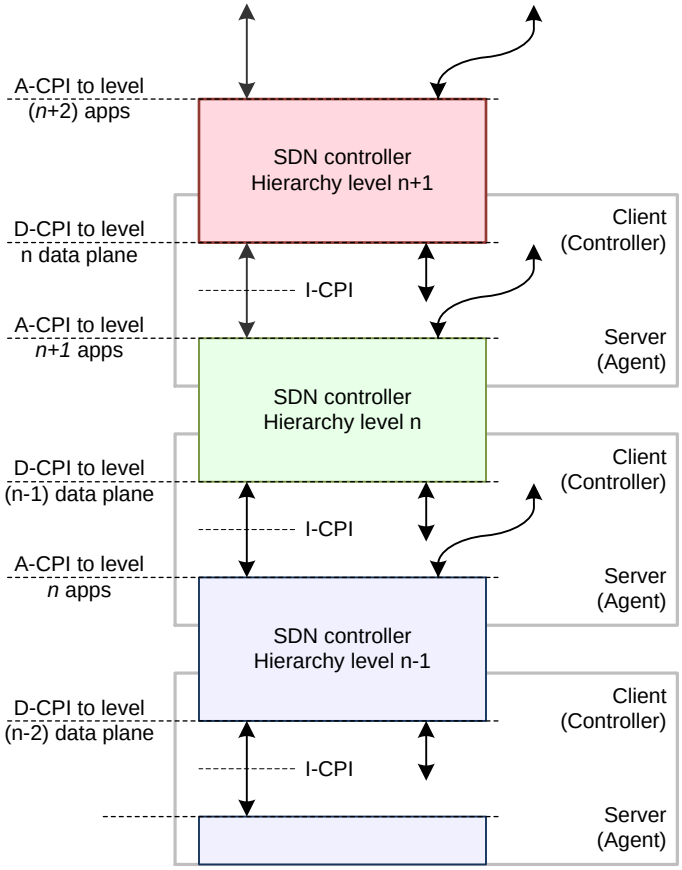

Figure 4.1 – Recursive hierarchical roles

Within the recursive hierarchy, an SDN controller or an application may consider itself as the direct controller of an information model instance that represents a suitably abstracted virtual network (VN). From this frame of reference, it supports a data-controller plane interface, D-CPI. Be it a (virtual) network element, another SDN controller, or even an application, the subordinate entity sees the superior entity as an application, supported by an A-CPI. From a global perspective, either or both of these interfaces may appear as intermediate CPIs, I-CPIs. It follows that, with the exception of physical NEs, a given entity may occupy any of the data, controller or application planes, depending on perspective.

All north-facing interfaces expose managed object instances for client use, but at different levels of abstraction.

At any level of the recursive hierarchy, a resource is understood to be subject to only one controlling entity. While an SDN controller may support any number of D-CPI instances, no resource on the subordinate plane is subject to more than one of them, nor is the resource subject to other SDN controllers (note). Allocation of resources to particular control domains is a management function.

Note – Resource sharing is described in clause 4.3.5.

Some applications may require all-or-nothing semantics, that is, transactional integrity (the ACID property: atomicity, consistency, isolation, durability) (note). Expansion of global and

abstract operations invoked by such an application implies transactional semantics at each lower level of abstraction, continuing all the way down into the hardware. Further, failed transactions must not leave behind stranded resources. Each level of hierarchy is recursively responsible for orchestrating the transactional semantics of its subordinate entities.

> Note – Distributed transactional integrity carries a heavy cost, and may not be required in all cases. If transactional integrity is not supported intrinsically, implementers should consider other means to recover from transaction failures, possibly including manual resource recovery. In some cases, inbuilt timeouts may suffice to recover stranded resources.

4.2 Data plane

The data plane incorporates the resources that deal directly with customer traffic, along with the necessary supporting resources to ensure proper virtualization, connectivity, security, availability, and quality. Figure 4.2 expands the NE resources view of figure 3.3 accordingly. The NE resources block comprises data sources, data sinks and forwarding and/or traffic processing engines, as well as a virtualizer whose function is to abstract the resources to the SDN controller and enforce policy. This expansion of detail also introduces a master resource data base (RDB), the conceptual repository of all resource information known to the network element.

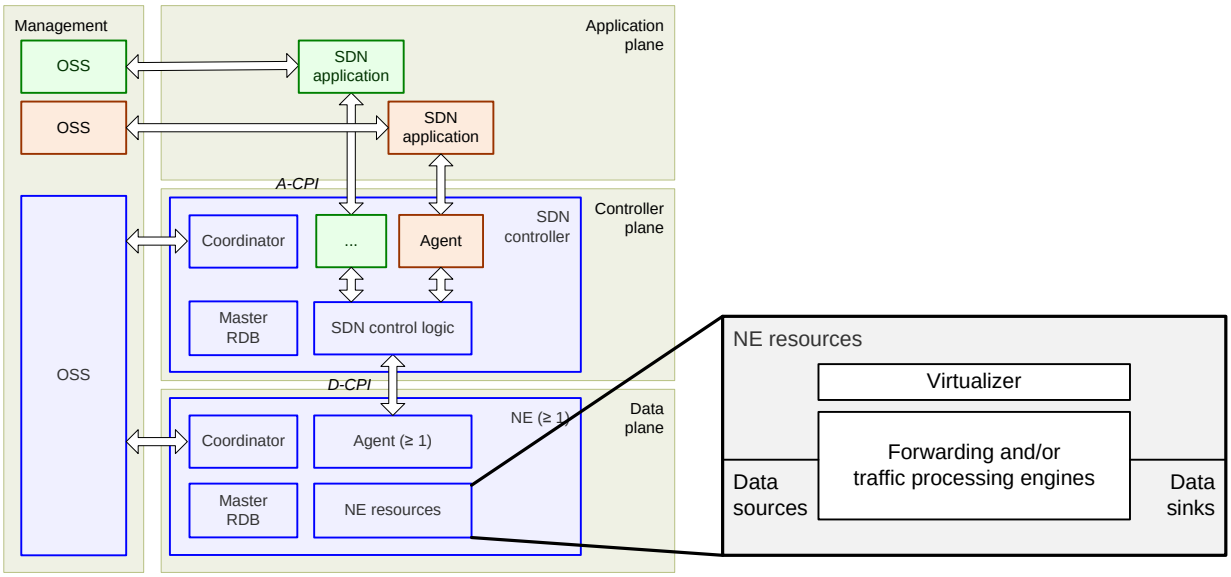

Figure 4.2 – NE resources detail

Software-defined networking concerns itself with traffic forwarding and traffic processing functions such as QoS, filtering, monitoring, or tapping. Traffic may enter or leave the SDN data plane via physical or logical ports, and may be directed into or out of forwarding or processing functions. Traffic processing might be exemplified by an OAM engine, an encryption function, or a virtualized network function [27]. Control of traffic forwarding or processing functions may be performed by an SDN controller or by separate mechanisms, possibly orchestrated in conjunction with the given SDN controller.

The data plane implements forwarding decisions made in the controller plane. In principle, it does not make autonomous forwarding decisions. However, the controller plane may configure the data plane to respond autonomously to events such as network failures or to support functions delivered by, for example, LLDP, STP, BFD, or ICMP.

The interface between data and controller planes (D-CPI) includes functions such as

- Programmatic control of all functions exposed by the RDB
- Capabilities advertisement
- Event notification

The data plane agent is the entity that executes the SDN controller's instructions in the data plane. The data plane coordinator is the entity by which management allocates data plane resources to various client agents and establishes policy to govern their use. Agents and coordinators serve the same purpose in every plane of the architecture. Both are discussed extensively in clause 5.

At the lowest layer of recursion, data plane resources are physical entities (including e.g., soft switches). At higher levels of abstraction, however, data plane resources need not be physical (e.g., virtual NEs). As with the other planes, the SDN architecture operates on an abstract model of the data plane, and as long as the functions advertised by the model are correctly executed, the architecture is blind to the difference. Virtualization is discussed in clause 4.5.

Management chooses which resources in which NEs are to be controlled by a given SDN controller, an operation described in clause 5.1. These resources are represented as a set of virtual NEs (VNEs), interconnected to form subnetworks. As illustrated in figure 4.14 on page 37, a VNE may itself be a subnetwork via successive abstraction.

The architecture imposes no restrictions on the technology of the data plane. The SDN controller may be used to program data planes implemented in existing technologies, such as DWDM, OTN, Ethernet, IP, etc., or in new data plane technologies that may evolve.

4.3 Controller plane

Although control is exercised to varying degrees in other planes (note), the SDN controller plane is modeled as the home of one or more SDN controllers. This clause describes the functional components of an SDN controller and its relation to other controllers and other administrative domains. As will subsequently emerge, not all responsibilities of the SDN controller can be allocated to specific functional components; the architecture sees no value in proliferating blocks beyond the current level.

> Note – This is the reason for the name *controller* plane, rather than the oft-used term *control* plane.

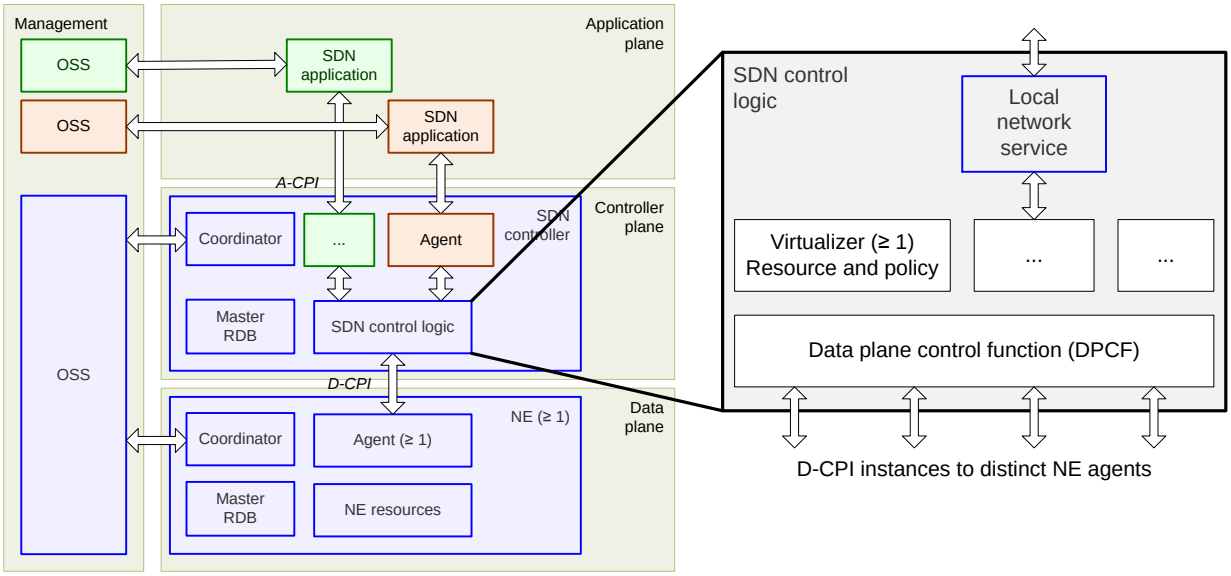

Figure 4.3 – SDN control logic detail

4.3.1 Overview

The architecture does not specify the internal design of an SDN controller. Functional components shown within an SDN controller, both above and throughout this document, are introduced only for explanatory purposes.

SDN control is logically centralized.

- A controller typically has subnetwork scope, spanning more than a single physical NE.
- There is no resource contention with other entities; the SDN controller regards itself as the owner of the virtual resources allocated to it by management.

Functions and services that are part of a controller's externally-observable behavior include full visibility of the information model instance under its control. Additional functions that may be required, depending on circumstances:

- Topology knowledge and path computation (the controller may also invoke an external service for these functions)
- The creation and maintenance of a further abstracted resource model for its applications, with resources bounded by enforced policy. Resource virtualization and control are potentially recursive.

An SDN controller is expected to coordinate a number of interrelated resources, often distributed across a number of subordinate platforms, and sometimes to assure transactional integrity as part of the process. This is commonly called orchestration. An orchestrator is sometimes considered to be an SDN controller in its own right, but the reduced scope of a lower level controller does not eliminate the need for the lower level SDN controller to perform orchestration across its own domain of control.

4.3.2 SDN controller

The SDN architecture does not specify the internal design or implementation of an SDN controller. It could be a single monolithic process; it could be a confederation of identical processes arranged to share load or protect one another from failures; it could be a set of distinct functional components in a collaborative arrangement; it could subscribe to external services for some of its functions, for example path computation. Any combination of these alternatives is allowed: the SDN controller is viewed as a black box, defined by its externally-observable behavior. Controller components are free to execute on arbitrary compute platforms, including compute resources local to a physical NE. They may also execute on distributed and possibly migratory resources such as on virtual machines (VMs) in data centers.

The architecture derives the required external behavior of an SDN controller from the principles of SDN (see clause 4.1).

The principle of logically centralized control is explored in detail below; here, it suffices to say that the SDN controller is understood to have global scope, for some value of globe, and that its components are understood to share information and state, such that no external block need concern itself with conflicting or contradictory commands from the controller. To the extent that the OSS affects resources or states, it is subject to the same coordination requirement with any SDN controllers that may be involved.

Multiple manager or controller components may have joint write access to network resources, but to comply with SDN principles, they must either

a) be configured to control disjoint sets of resources or actions, or
b) be synchronized with each other so that they never issue inconsistent or conflicting commands.

Note 1 – In distributed computing control of distributed network resources, strict state synchronization may carry excessive performance or complexity penalties. The implications of eventual (rather than strict) state convergence are a topic for further study.

Note 2 – The assumption of internal consistency in the controller is separate from the question of state consistency in the view of the underlying data plane resources. SDN controllers are always expected to be able to deal with related events that are asynchronously visible from various parts of the infrastructure.

Without intending to minimize their importance, issues such as bootstrap, synchronization, migration, backups, audits, controller software release management, etc., are internal to the black-box SDN controller, and are not a matter for the SDN architecture.

4.3.3 SDN controller functional components

Having just stated that the SDN controller is a black box, it is nevertheless useful to conceptualize a minimum set of functional components within the SDN controller (figure 4.3), namely data plane control function (DPCF), coordinator, virtualizer, and agent. Subject to the logical centralization requirement, an SDN controller may include arbitrary additional functions. A resource data base (RDB) models the current information model instance and the necessary supporting capabilities. Clause 5 discusses the RDB and its partitions in detail.

Data plane control function

The DPCF component effectively owns the subordinate resources available to it, and uses them as instructed by the OSS/coordinator or virtualizer(s) that controls them. These resources take the form of an information model instance accessed through the agent in the subordinate level.

Because the scope of an SDN controller is expected to span multiple (virtual) NEs or even multiple virtual networks (with a distinct D-CPI instance to each), the DPCF must include a function that operates on the aggregate. This function is commonly called orchestration. This architecture does not specify orchestration as a distinct functional component.

Coordinator

To set up both client and server environments, management functionality is required. The coordinator is the functional component of the SDN controller that acts on behalf of the manager. Clients and servers require management, throughout all perspectives on data, control and application plane models, so coordinator functional blocks are ubiquitous.

Further discussion of managers and management functions appears in clause 4.6.

Virtualizer

Note – The network function virtualization concept discussed in ETSI ISG NFV [27] differs from the virtualization concept as used in the SDN architecture. In the SDN architecture, virtualization is the allocation of abstract resources to particular clients or applications; in NFV, the goal is to abstract network functions away from dedicated hardware, for example to allow them to be hosted on server platforms in cloud data centers.

An SDN controller offers services to applications by way of an information model instance that is derived from the underlying resources, management-installed policy, and local or externally available support functions. The functional entity that supports the information model instance and policy at an A-CPI (application-controller plane interface) is called a virtualizer. It presents the local trust domain boundary to the corresponding agent, which represents the client's view of the information model instance.

A virtualizer is instantiated by the OSS/coordinator for each client application or organization. The OSS/coordinator allocates resources used by the virtualizer for the A-CPI view that it exposes to its application client, and it installs policy to be enforced by the virtualizer. The effect of these operations is the creation of an agent for the given client.

The virtualizer may be thought of as the process that receives client-specific requests across the A-CPI, validates the requests against the policy and resources assigned to the client, translates the request into terms of the underlying resources, and passes the results on to the DPCF and the D-CPI.

Virtualizer and DPCF and possibly other SDN controller functions must collaborate to provide features such as notification interpretation, resource sharing, implicit provider services, and transactional integrity.

Clause 4.5 describes virtualization in detail.

Agent

Any protocol must terminate in some kind of functional entity. A controller-agent model is appropriate for the relation between a controlled and a controlling entity, and applies recursively to the SDN architecture. The controlled entity is designated the agent, a functional component that represents the client's resources and capabilities in the server's environment.

An agent in a given SDN controller at level *N* represents the resources and actions available to a client or application of the SDN controller, at level *N*+1. An agent in the level *N*-1 data plane represents the resources and actions available to the given level *N* SDN controller. Even though the agent's physical location is inside the server's trust domain (i.e., on a server SDN controller platform), the agent notionally resides in the client's trust domain.

Other controller components

To avoid overspecification, the architecture only describes functions that are required of an SDN controller, but does not preclude additional functions. These may take the form of applications or features supported by the controller. These features may be exported to some or all of the server's external applications clients, or used internally by the provider administration for its own purposes.

As components of the SDN controller, such applications or features are subject to the same synchronization expectation as other controller components. To facilitate integration with third party software, the interfaces to such applications or features may be the same as those of others at the A-CPI.

The security aspects of such embedded applications are important to understand. Because they execute in the server's trust domain, they will be subject to the server's test, verification, audit and release management cycle.

4.3.4 Delegation of control

Although a key principle of SDN is stated as the decoupling of control and data planes, it is clear that an agent in the data plane is itself exercising control, albeit on behalf of the SDN controller. Further, a number of functions with control aspects are widely considered as candidates to execute on network elements, for example OAM, ICMP processing, MAC learning, neighbor discovery, defect recognition and integration, protection switching.

A more nuanced reading of the decoupling principle allows an SDN controller to delegate control functions to the data plane, subject to a requirement that these functions behave in ways acceptable to the controller; that is, the controller should never be surprised. This interpretation is vital as a way to apply SDN principles to the real world.

Criteria that encourage the controller to delegate a function to the data plane include:

- Rapid real-time response required to network events
- A large amount of traffic that must be processed
- Byte- or bit-oriented functions that do not readily lend themselves to packetization, for example repetitive SDH multiplex section overhead
- Low-value, possibly repetitive, predictable, well-understood, completely standardized behavior, for example encryption, BIP, AIS insertion, MAC learning, CCM exchanges

- Survivability or continuity in case of controller failure or re-initialization
- Functionality commonly available in data plane silicon, e.g., protection switching state machines, CCM counters and timers
- No perceived opportunity to add value by separating the function.

Assuming the raw data can be made available, an SDN controller always has the option not to delegate a control function, but to conduct the necessary operations itself. The criteria listed above affect whether such a choice is practical.

In deciding whether or not to delegate a function, the SDN controller must understand whether the behavior of a candidate delegated function fully satisfies its needs. This may require inbuilt or configured knowledge and/or query of candidate functionality and capability.

Notifications from delegated functions are recommended to follow a publish-and-subscribe model. Examples of notifications:

- A delegated control function may be asked to report
    - State and attribute value changes, such as port up/down (operational state: enabled/disabled),
    - Threshold crossing alerts (TCAs) against performance monitoring (PM) counters,
    - Hardware failure, recovery or installation, inasmuch as such activities affect the resources under the SDN controller's purview,
    - Manual and automatic protection switching events and results.
- The delegated control function in the data plane may execute standardized protocols and report intermediate or final results, exceptions or state changes. Examples include
    - Traffic encryption, including key exchange and update
    - CFM: 802.1ag or BFD
    - 802.1X authentication agent
    - GMPLS, providing the SDN controller with signaling access across administrative domain boundaries where SDN may or may not be supported

4.3.5 Shared resources

The data plane model of clause 4.2 assumes that resources are dedicated to clients or applications. In a dedicated resource model, there is limited opportunity to increase the usage efficiency of underlying resources, which is one of the anticipated benefits of SDN. One way to resolve this issue is contractually agreed best-efforts resource sharing, possibly with prioritized and weighted traffic contracts. An extension of the shared-resource feature is the possibility that the provider could maintain a pool of uncommitted resources for on-demand allocation (and billing) to clients on a first-come, first-served basis, or on a pre-negotiated schedule.

Maximizing resource usage implies that the resources in question may be oversubscribed. When a client requests a non-dedicated resource, the available performance may be less than desired, or the resource request may fail completely. Accordingly, the client must be prepared to deal with

exceptions. The acceptable degree of shared resource unavailability may be contractually committed by a provider to a client.

Another form of resource sharing occurs when some fraction of a resource is committed to each of several clients, for example 20% of the bandwidth of a link. This cannot be represented in the simple model that a client owns the entirety of some managed object instance. Some such resources may be allocated statically, rather than on demand, and may be immune to oversubscription.

The SDN controller has the responsibility to administer resources that are shared by more than one client or application, whether static or dynamic, taking a common view of the commitments made to the concerned clients. Changing requests and releases of shared resources may trigger reoptimization across the provider's network. The architecture does not allocate this responsibility to particular functional components of the SDN controller.

4.3.6 Multiple administrative domains

Figure 4.4 illustrates a case in which each of several administrations owns its own network, which are mutually interconnected. The topology is the same as in figure 2.1, but the colors have been changed to show different ownership of the various subnetworks.

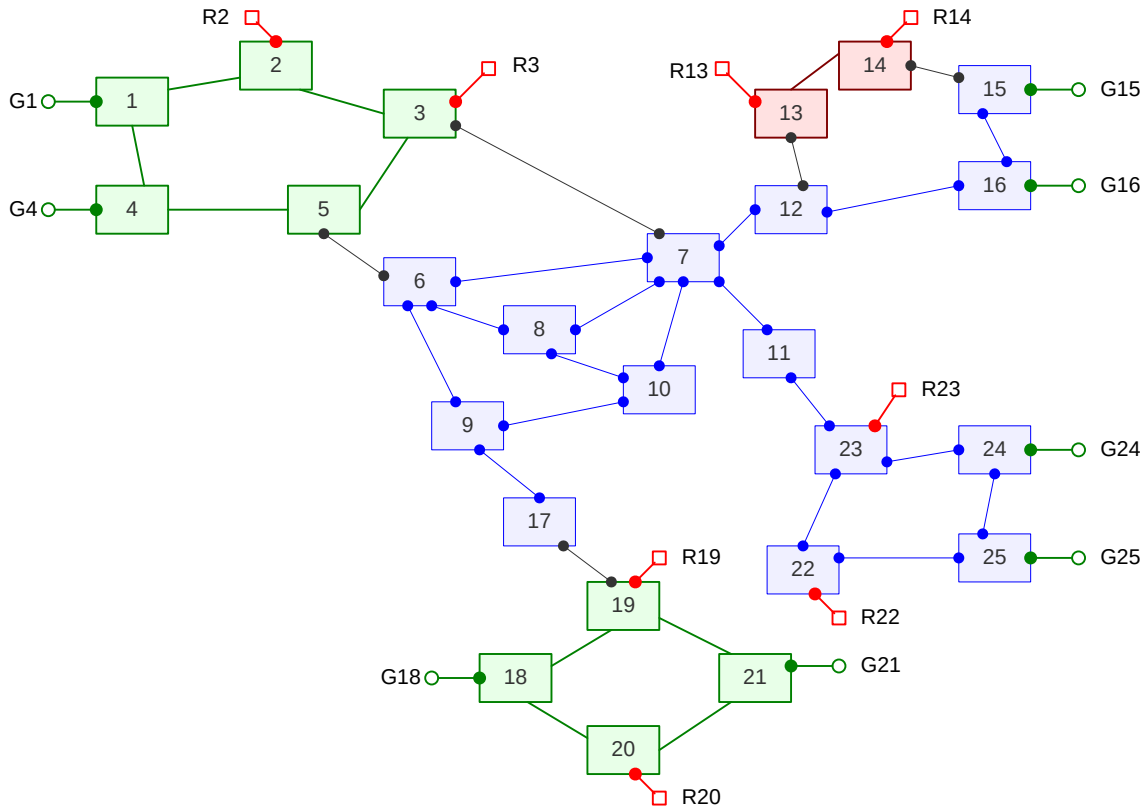

Figure 4.4 – Network with multiple owners

Figure 4.5 organizes these subnetworks into administrative domains (note), and abstracts the view to show only matters of concern to Red. That is, Red sees full detail of its own NEs and ports, but the Blue and Green domains are abstracted to the simplest possible representation.

> Note – Ownership is always a criterion for an administrative domain grouping. Network owners may define subordinate administrative domains for other reasons, for example scalability or technology.

Assume that there exists an SDN controller (SDNC) for each of the administrations Red, Blue, Green.

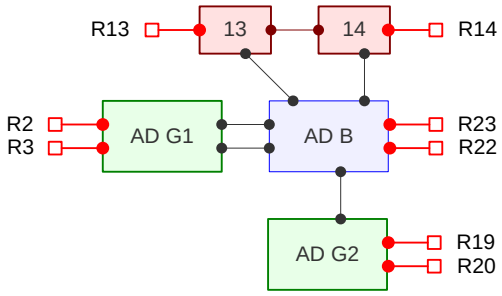

Figure 4.5 – Administrative domains of interest to Red

Figure 4.6 illustrates the options for Red's SDN controller associations.

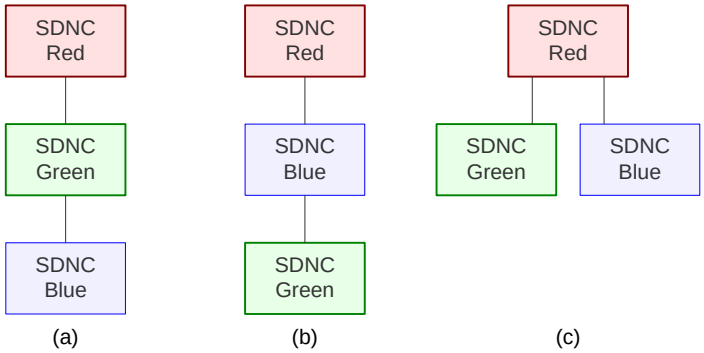

Figure 4.6 – Organization options

(a) Green may offer a service that seamlessly encompasses both Blue and Green domains. Blue resources are leased by Green on behalf of Red; Red has no business relationship with Blue, and interacts with Blue resources only by way of the passed-through virtualization provided by Green.

(b) There is no particular reason for Red to prefer Green as its service provider. The business and virtualization relationship could equally be in the other order.

(c) Red may have a contractual relationship with both Green and Blue. In this case, Red has visibility of the (three) links between Green and Blue domains, and expects to have some level of control over them. The finer granularity of this view may or may not permit Red to optimize its use of Green and Blue resources, according to criteria of its own choice, for example monetary cost, availability, latency.

In a fourth option, Red may have data plane handoff points in, or possibly transit connections through, administrative domains with which it has no SDN programmatic relationship (imagine that Blue did not offer SDN control visibility to its peers). The Red SDN controller may see this part of its network as statically provisioned connectivity, i.e., as a set of tunnels, or it may run conventional routing and signaling protocols to learn about reachability and establish connectivity to and through such domains.

Other clauses of the architecture document consider the case when an SDN controller offers services via its A-CPI, and in which the client for such services is in fact another SDN controller. As shown in figure 4.7, the (White) client controller, or a network-aware application, may orchestrate a number of server controllers. When server controllers are orchestrated from a common point, there is, in principle, no need for the server SDN controllers to communicate among themselves. Even here, however, optimizations may be available if the client controller enables limited communications between server controllers. The principle to be applied is that of logical centralization: the client controller may delegate control functionality elsewhere, as long as it retains full awareness of all state that is of interest to itself.

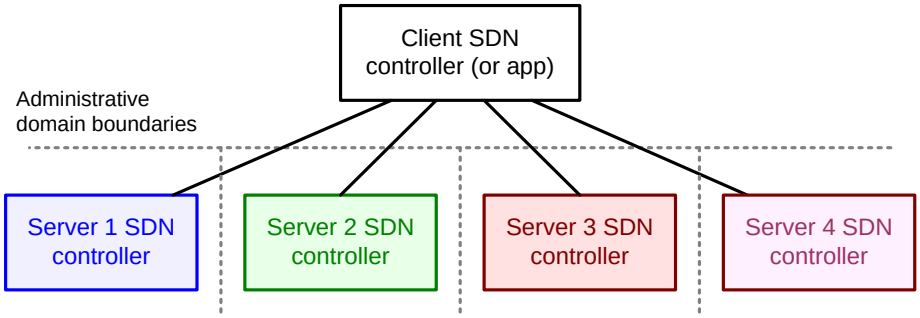

Figure 4.7 – Common controller coordination example

4.3.7 Controller-to-controller coordination

This clause further considers the case in which a number of SDN controllers exist as peers, but without overall orchestration from a superordinate SDN controller. Reasons to separate SDN controllers into distinct peer domains may include any combination of the following considerations:

- Controllers may be from different vendors who have not achieved full interoperability
- Controllers or underlying infrastructure may be owned or operated by different administrative organizations
- Controllers may have different technology or service functionality
- Scalability of network node count or geographic span, including the distinction between WAN and LAN
- Others

In the general case, a telecommunications service traverses multiple data plane network control domains (NCDs), which may include domains that are not under SDN control. These services require coordination between the associated SDN controllers, and also with non-SDN management, control or signaling. Peer-to-peer information exchange is generically referred to as controller-to-controller (C2C) communication (note).

Note – Controllers in a client-server relationship (as above) are also communicating, but the term C2C implies a peer-to-peer arrangement.

Figure 4.8 illustrates a simple set of network control domains NCD1..NCD4, together with their SDN controllers. In a hybrid of the previous arrangements, client White is shown with a direct relationship to Blue and Green, but only an indirect relationship to Red. NCD3 is exemplified as a domain without SDN control; services that terminate in NCD3 or traverse NCD3 must employ existing signaling or routing protocols (see below) or pre-negotiated agreements.

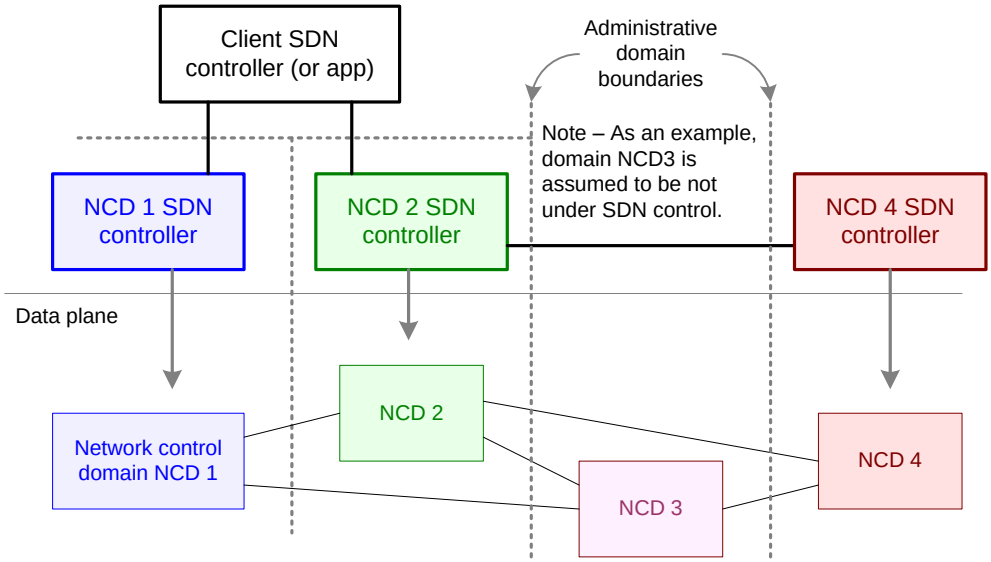

Figure 4.8 – Peer-peer controller coordination example

When controllers communicate across administrative domain boundaries that are also business boundaries, security and contractual issues of trust and information hiding become vital.

Information to be exchanged between controllers may include the following:

- SDN controller adjacency and capability discovery
- Data plane neighbor and topology discovery, to the extent agreed by policy
- State and attribute information, including the ability to subscribe to state and attribute change notifications, as agreed by policy
- Forwarding-relevant information, such as reachability at one or more layers
- Path computation information such as route cost, protection or restoration policies
- Other information such as OAM configuration, QoS assessment and reporting, usage information for billing

Operations may need to be synchronized in at least soft real time, for example when setting up a loopback point in one domain, then invoking a loopback test in a different domain, and finally releasing the loopback point.

These information exchanges are generally compatible with those of non-SDN network control domains, which use existing protocols (note). As of today, no C2C use case requirements have been identified that cannot be satisfied with existing protocols, possibly with minor extensions.

Feature negotiation and policy exchange are possible areas for further investigation. The need to develop a new protocol for SDN C2C purposes is a topic for further study as SDN matures.

> Note – Security for C2C associations that use existing protocols is understood to be a matter of existing specification and best practice. If new protocols were to be proposed, security considerations would be an important aspect of their standardization.

4.4 Application plane

Figure 4.9 expands the SDN application block from figure 3.3.

SDN principles permit applications to specify the resources and behavior they require from the network, within the context of a business and policy agreement. The interface from the SDN controller to the application plane is called the application-controller plane interface, A-CPI (note). Figure 4.9 shows that an SDN application may itself support an A-CPI agent, which allows for recursive application hierarchies, as explained in clause 4.1. Different levels of an application hierarchy are described as having various latitudes, depending on their degree of abstraction.

> Note – The SDN community often calls the A-CPI a northbound interface (NBI) or northbound API. Refer to clause 2.3.

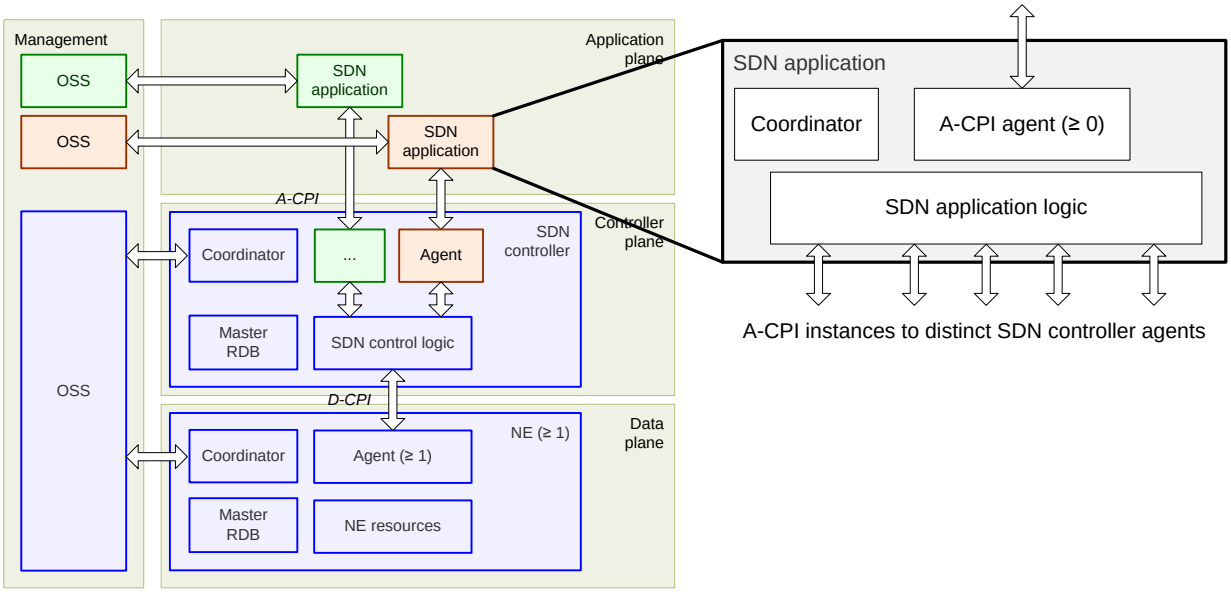

Figure 4.9 – SDN application detail

An SDN application may invoke other external services, and may orchestrate any number of SDN controllers to achieve its objectives. The OSS link and the coordinator function recognize that, like the other major blocks of the architecture, SDN applications require at least a certain amount of a priori knowledge of their environments and roles.

- An application plane entity may act as an information model server, in which case, it exposes an information model instance for use by other applications. Formally, the other applications are clients, which communicate to the SDN application server agent shown in figure 4.9.

- An application plane entity may act as an information model client, in which case it operates on an information model instance exposed by a server entity. The server entity may be an SDN controller or a subordinate application.
- An application plane entity may act in both roles simultaneously. For example, a path computation engine (PCE) may rely on an SDN controller for virtual network topology information (maintained in a traffic engineering database), while offering the SDN controller a path computation service.

Activity across the A-CPI typically includes queries or notifications about the state of the virtual network, and commands to alter its state, for example to create or modify network connectivity or traffic processing functions between network client layer (data plane) handoff points, with some specified bandwidth and QoS. The A-CPI may also be used for additional functions, for example as an access point to configure a service chain through one or more layer 4-7 services (note), or as an input to control virtualized network functions.

> Note – In terms of network behavior, service chaining is just the steering of traffic through an appropriate set of components. The added value at an A-CPI may be the ability to specify a sequence of component functions, expecting that the SDN controller will select the optimum instances of these functions and apply the pertinent traffic forwarding rules. The application could also support programming of component attributes, or even instantiate new virtualized network functions at optimum points in the topology.

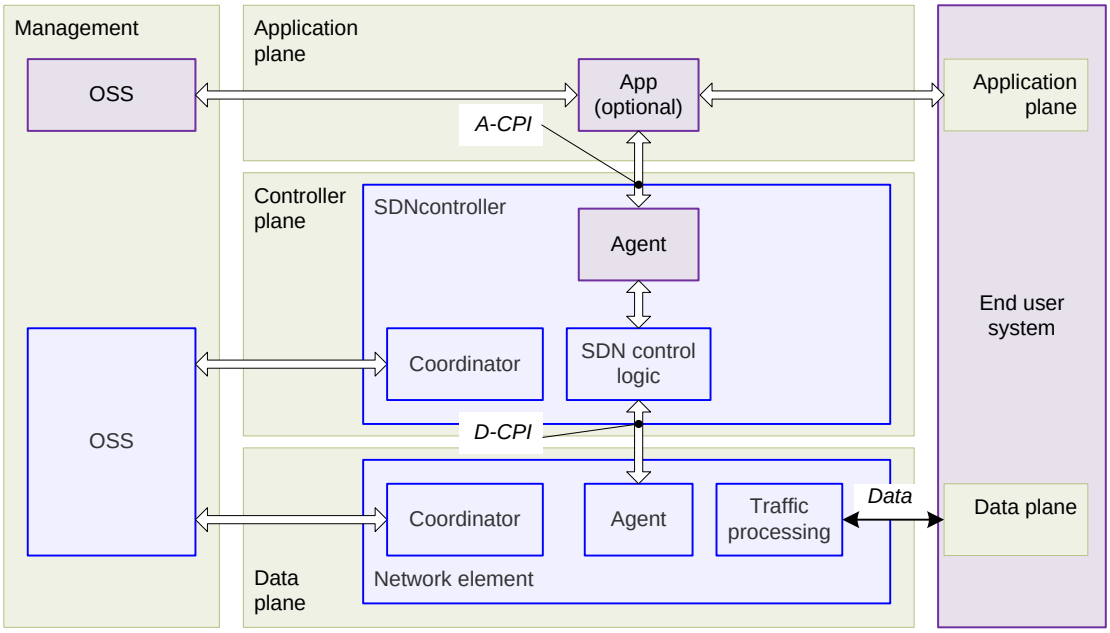

Figure 4.10 – Multi-plane end user system example

Figure 4.10 illustrates the possibility that an end user system may present both data plane and application plane aspects. An end host or a network appliance may fit this model. A firewall or DDOS detector would exemplify the network appliance case, and a customer terminal that was capable of signaling its existing or desired state is an example of the end host case (note).

Note – For example, the Microsoft use case in which Lync user terminals are capable of reporting or requesting service characteristics, whereupon a centralized coordination function may instantiate responses in the network resources.

Characteristics of the A-CPI are considered in more detail below (clause 6.8).

4.5 Virtualization

SDN control and management must be designed to operate on abstracted and virtualized resources, which ultimately encompass underlying physical resources, possibly through several successive levels of virtualization. This is done by way of a common information model that includes representation of physical hardware as a special case. This clause describes the capabilities necessary to virtualize network resources.

Recall that the data plane of an SDN is a network, a set of nodes that forwards traffic and may also produce, consume, store, or process traffic. Its nodes are network elements (NEs) interconnected by links. The NEs offer external data plane ports to client equipment and other networks. Because some of the anticipated benefits of SDN are based on centralized control, an SDN controller will generally control more than one NE.

In the following text, the abstraction and virtualization process is explained in a step-wise sequence, starting with a resource graph from a hypothetical provider environment (Blue), and extending to virtual network representations for specific customers, Green and Red.

Figure 4.11 reiterates figure 2.1 to illustrate the example network, owned by Blue. Rectangles indicate network elements (NEs). Lines designate links, which may be composite, which may be transitional, involving the same or different types of characteristic information, which may be protected, and which may be virtual (i.e., network client layers supported by further underlying network server layers). Open endpoints indicate data plane handoff points that are suitable for connection to external network equipment. The external ports are used by two clients, Green and Red.

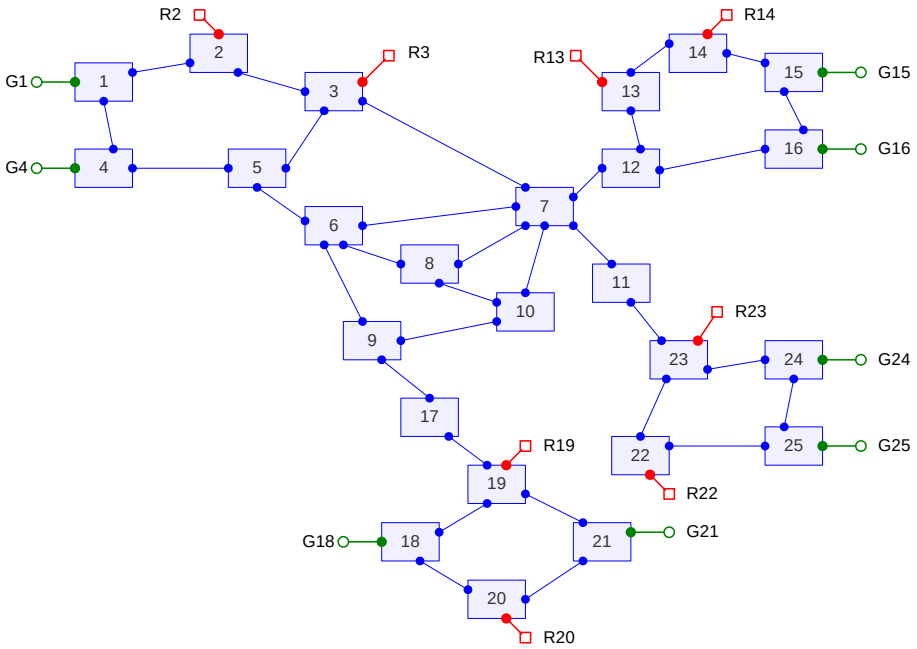

Figure 4.11 – An example abstract network

Considering the scale and complexity of a real network, it is clear that figure 4.11 is already an abstraction of underlying physical resources.

> Reminder: an *abstraction* is a representation of an entity in terms of selected characteristics, while hiding or summarizing characteristics irrelevant to the selection criteria. A *virtualization* is an abstraction whose selection criterion is dedication of resources to a particular client or application.

In figure 4.11, the selected abstraction characteristics reflect the intention to simplify the representation of a real-world network.

Figure 4.12 illustrates the same underlying network abstraction, now virtualized by provider Blue for client Green. The assumption here is that Blue reserves non-zero resources for Green in each of its NEs and links. The network topology therefore appears to be the same, but the NEs of figure 4.11 have been replaced with virtual NEs, VNEs (note). Ports previously shown for client Red have disappeared. For privacy reasons, Blue's commitments to Red *must* be concealed from this view.

> Note – The Blue NEs may already themselves be virtual.

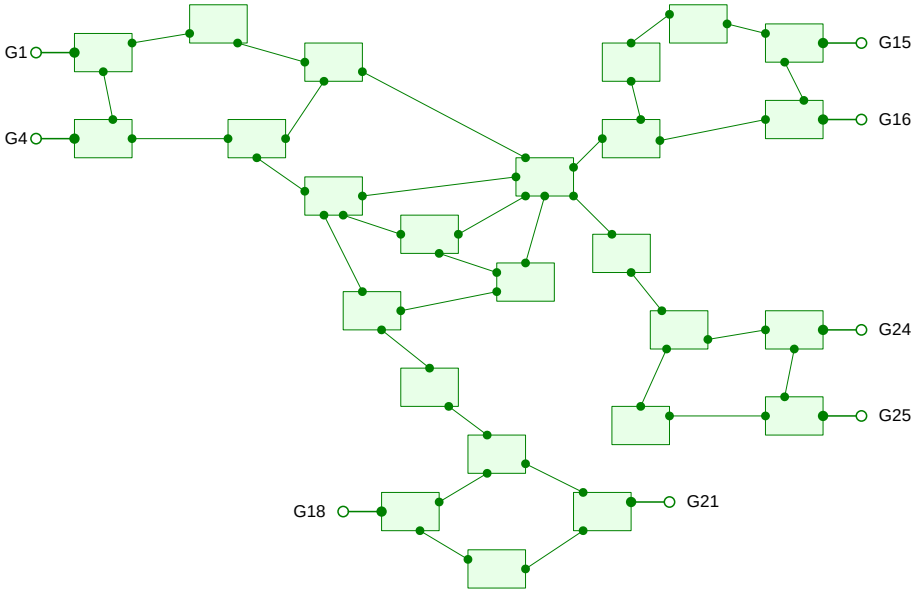

Figure 4.12 – Provider Blue's virtual network for client Green

The difference appears pictorially in the colors of the VNEs. In this case, a VNE represents a subset of the resources of an NE, the subset committed to client Green. Likewise, though not shown pictorially, the capacity and capability of links and exposed ports will have been reduced to retain only resources dedicated to Green.

As shown, the underlying network contains redundant resources that can visibly accommodate many of the visibly possible link and node failures. But suppose client Green was not prepared to pay for redundancy. Then, rather than the virtual network (VN) of figure 4.12, provider Blue might allocate resources only for a reduced VN as shown in figure 4.13, which provides the necessary connectivity, but without visible route redundancy (note).

> Note – Because links may be protected internally, it cannot be asserted that the resulting Green VN has no redundancy, but it clearly has lower availability than that of the previous topology. The cartoon is intended to suggest that the simpler VN may be unprotected.

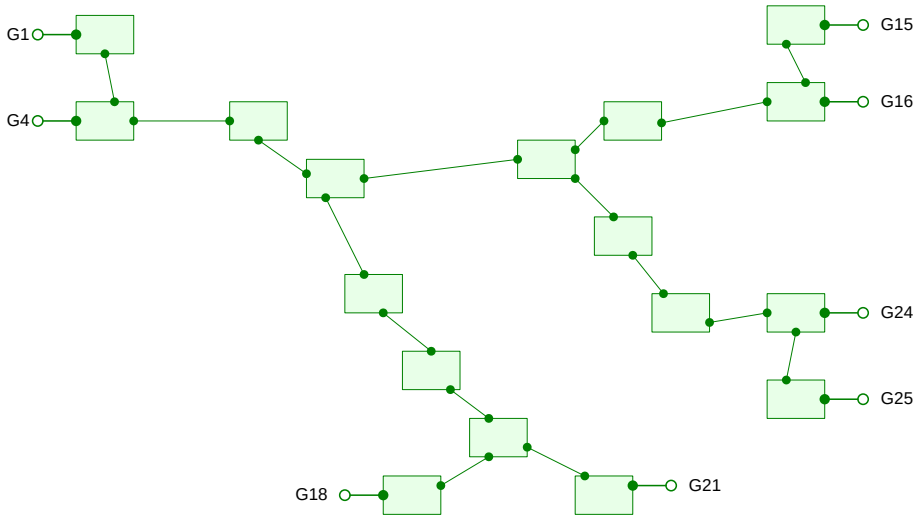

Figure 4.13 – Reduced cost, reduced availability provider VN for client Green

So far, the virtualizations all exist in the Blue network provider space, and serve as the means by which Blue identifies resources dedicated to client Green.

A VN (subnetwork) may be abstracted into a yet simpler VN. Provider Blue needs an internal map of all resources dedicated to client Green. However, Green may not care to see all of these resources; they may well be irrelevant to Green's network usage scenarios. Additionally, as a policy matter, Blue may be unwilling to expose details of its network. Therefore, the VN visible at the interface from Blue to Green may be further abstracted, i.e., be less detailed than Blue's own view. Figure 4.14 illustrates a possible VN offered by Blue to Green at their common CPI (see also figure 4.16). Each VNE is an aggregate of the resources reserved by Blue in the more granular model of figure 4.13.

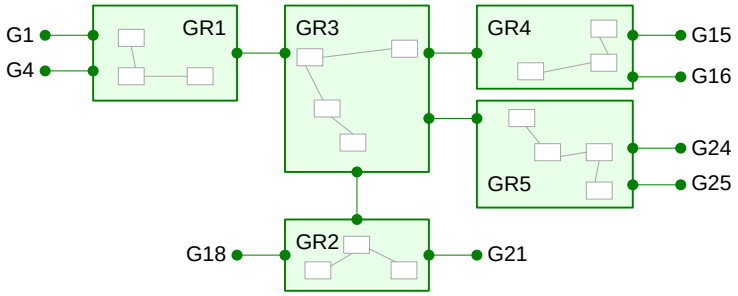

Figure 4.14 – Client Green's view of simpler VN

To summarize: the Blue SDN controller needs to know about both levels of abstraction, but the view of figure 4.13 is for its internal use. Naming of these internally reserved resources is a private matter for the Blue SDN controller. The managed object instances (ports, virtual NEs) visible to the Green SDN controller at the CPI (figure 4.14) are named according to the agreement between Blue and Green. Identifiers may be contractually pre-negotiated or negotiated over the CPI at run time.

As shown in figure 4.15, Green could also contract for the simplest possible VN, a single VNE as shown by the rectangle (client's view in green, provider's underlying view in gray). Client Green might specify that the VNE be called Green-1.

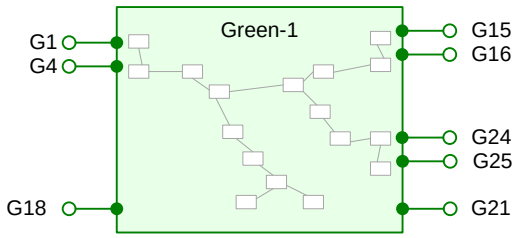

Figure 4.15 – Client Green's view of simplest possible VN

Suppose Green wishes to create a forwarding rule between, for example, its ports G1 and G25 (or between IP subnets accessible through G1, G25). To Green's SDN controller, given the Green-1 abstract NE of figure 4.15, this is a single forwarding rule in a single switch. Blue's SDN controller must intercept the command that establishes the forwarding rule, reinterpret it in the context of the virtual subnetwork (figure 4.13) dedicated to Green, then further map it onto its own Blue network (figure 4.11). If Green's action is expressed as an IP forwarding rule, Blue may implement the request in the underlying network as any combination of rules in L3 forwarding nodes, mappings into existing tunnels, or mappings into newly created tunnels. The following paragraphs consider these options in further detail.

One of the important reasons for virtualization is that Blue must isolate Green's traffic from that of other clients, often without the knowledge or active cooperation of Green. There are three cases, any of which may be used along a particular link in the end-to-end path.

(a) Case 1: Isolation may be achieved by physical means; e.g., if Green contracts for dedicated media, wavelengths/spectra or tributary time slots.
(b) Packet traffic may need to be isolated if it cannot be guaranteed that there are no address space overlaps among the different clients. Guarantees of address uniqueness may need to be enforced with access control lists (ACLs) at data plane handoff points. If encapsulation is needed, it can be performed in either of two ways:
    i. Case 2: By way of an additional encapsulation layer, for example with service VLAN IDs (S-VIDs) [10].
    ii. Case 3: Within the same layer by way of a mapping such as network address translation (NAT).

    The choice of encapsulation is a matter of Blue policy; its details are invisible to Green. At network points chosen by Blue, possibly but not necessarily the edge devices (the resources underlying ports G1 and G25), Blue adapts ingress traffic from Green to its isolated form, and reverses the adaptation for egress traffic to Green. The Blue infrastructure forwards encapsulated traffic across the network in Blue's address space.

Encapsulated isolation may take the form of tunnels in the Blue infrastructure. These are subnetwork connections in the Blue server layer that appear as simple links to Green. To establish a forwarding relationship, Green need only map traffic into the near- and far-end link

endpoints of the proper tunnel. Blue may pre-configure subnetwork connections as part of the VN it offers Green. If Blue can intercept Green's forwarding requests, Blue may also create tunnels dynamically. In either case, Blue would accompany tunnel creation with the necessary support features, for example OAM and protection. Tunnel setup and operation is invisible to Green.

The internal users of administration Green may have differing requirements for the use of their VN, for example simple connectivity requests (as above) or more detailed control. Figure 4.16 illustrates how Green may internally perform further abstraction from the simple VN of figure 4.14 to the single-NE view of figure 4.15. As shown, this abstraction (and its subsequent interpretations) can be done entirely within Green's own space, and completely without the knowledge of Blue. This might be appropriate if some Green applications care only about connectivity between edge ports, and other applications deal with performance optimization.

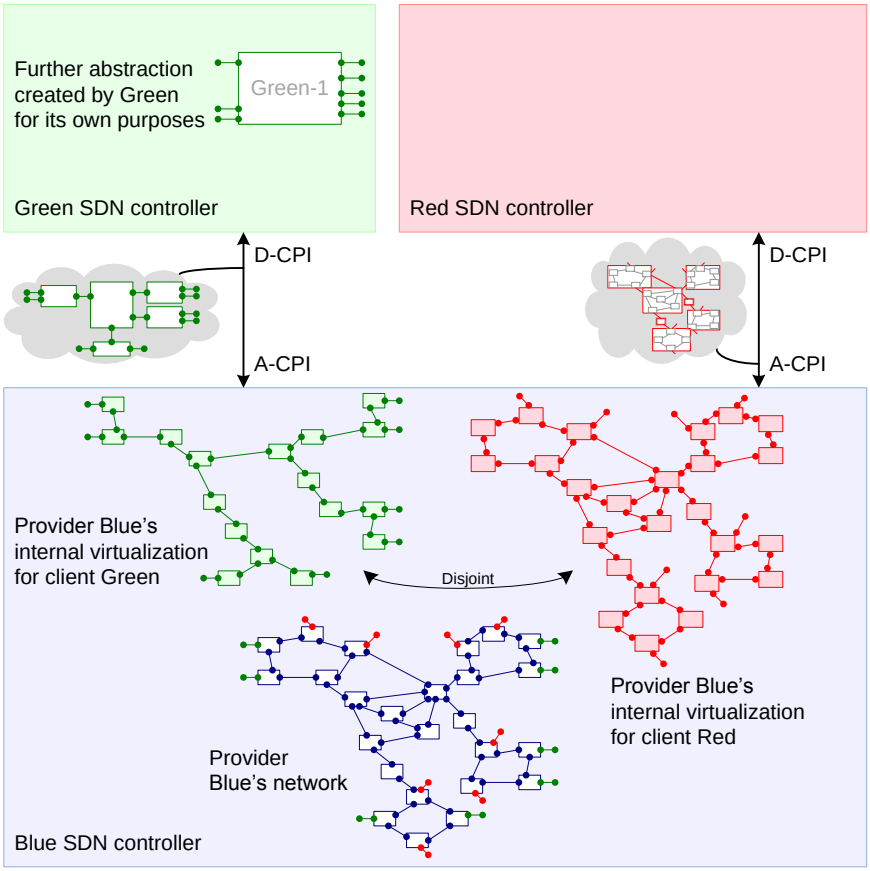

Figure 4.16 – Green's further abstraction, and VNs for Red

Figure 4.16 also illustrates how Blue may allocate resources that expose completely different VNs to another client, Red.

Suppose Green decided to upgrade to a higher availability service, say with (visible) route redundancy. Provider Blue would revise the underlying VN to restore some or all of the redundant resources from figure 4.12, but there would be no need to update the VN exposed by

Blue to Green at the CPI. Green would see no change except in observed availability, exactly as expected.

Further data plane considerations

The networks illustrated above have been severely abstracted to highlight the points under discussion. A less abstracted view might reveal important additional information, for example that the subnetworks were multi-layer (note), meaning that the ports and links might (or might not) support different characteristic information. Links might be composite, and might have multiple endpoints. Adaptation between layers might be required. A given layer might have layer-specific overhead that needed to be controlled, specifically for operations, administration and maintenance (OAM). Redundancy might be exploited for survivability at one or more layers.

> Note – Separate controllers for each technology in a layered network are sometimes proposed, for example between packet and optical layers. This architecture neither specifies nor precludes technology layer separation.

These aspects are important, but require extensive additional specification, and are thus covered in architecture and information models to be developed in domain-specific ONF working groups (WGs), e.g., optical transport, and wireless and mobile WGs. From the perspective of this document, it suffices to recognize the need for state and structure well beyond that implied by the simple view. As noted in clause 4.3, some of this state and structure may be delegated from the SDN controller into data plane hardware and software.

4.6 Management

Management covers infrastructure support tasks that are not to be done by the application, controller and data planes themselves. Management may also perform operations that the application, controller- and data planes are restricted from doing by policy or for other reasons. Perhaps the single most important reason to prevent a task from being executed by SDN components is that the SDN controller may reside in a customer trust domain, while business reasons mandate that core management and support functions be done within the provider trust domain. Although an agent policy could be devised that completely trusted its controller, the transparency policy and policy enforcement software would nonetheless have to be installed by the provider’s manager. For security reasons, the default behavior is recommended to be to expose nothing, rather than everything.

The SDN architecture recognizes classical management functions such as equipment inventory, fault isolation, software upgrade and the like, but regards them as largely out of scope of SDN. One of the perceived benefits of SDN is allowing clients (in foreign trust domains) to perform many of the actions that are today performed by management systems. The traditional OSS interface is expected to play a smaller role over the course of time, as customer applications take on more responsibility via SDN controllers.

Within the scope of SDN are the SDN-specific management functions, namely recording and expressing business relationships (policies) between provider and client, and configuring SDN entity environment and initialization parameters. This includes coordinating data plane handoff points, identification conventions, reachability and credentials among logical and physical

entities. The SDN architecture requires that this information be configured into the relevant SDN NEs, controllers, and applications, but does not specify the nature or structure of the OSSs.

In the general case, each client-server pair of data plane, controller and application level entities lies in a separate trust domain (see figure 4.1). Where a trust boundary exists in the SDN hierarchy, a corresponding trust boundary also exists in the management domain. Managers – called OSSs in this document – in different trust domains may need to exchange information, but this exchange is beyond the scope of the SDN architecture.

Two management roles are recognized: server manager and client manager. The responsibilities of the server manager are not the same as those of the client manager.

Responsibilities common to both managers

- Configuration of separate entities such that they can communicate with each other. This may include information such as identity, protocol selection, reachability, and security policy and credentials.

Responsibilities of the server manager

- Instantiation of an agent in the server environment, representing a client-specific environment in a real or virtual infrastructure. This includes resource allocation and policy installation, and possibly downloading of custom or special feature modules. Client-specific configuration could include a choice of protocol or release level (e.g., OpenFlow-switch 1.3). SDN control of the server's own interest is accommodated by an agent in the server's trust domain (note).

  Note – Clause 5 describes agent functions in detail.

- Updating client-specific resource allocation and policy over the course of time. This may result from events such as business renegotiation or network build out, or from the request and release of resources covered by contract but delivered and billed on demand. A special case is the deletion of everything related to a given client when the business agreement terminates.
- Auditing the compliance of resource allocations and policies to the business commitments. This includes confirming that resources are not double-booked and that the traffic of separate clients is mutually isolated. Tools for this purpose include notifications, for example of security alarms or connectivity faults.
- Subscribing to notifications and collecting statistics for purposes of SLA monitoring, security monitoring, fault management, billing, network planning, and others. These are existing functions that are expected to remain unchanged except perhaps in their details.

Responsibilities of the client manager

The client manager has much the same responsibility as the server manager, but from the inverse perspective.

- The client SDN controller (or application) may require information that cannot be discovered from the server, in particular about data plane adjacencies on its external network ports. If so, the manager must supply the information.

- Although the client SDN controller receives a view of the resources from its agent on the server, the client manager may wish to instantiate its own view of the contracted resources and policy. This could facilitate reconciliation or audit by the client SDN controller. Auditing of expected versus discovered resources and actions may be an important security feature.
- Both before and during operation, the server manager seeks assurance that the client gets no more service than contractually specified, while the client manager seeks assurance that it gets no less service than contractually specified. The client manager may poll for performance or state information, or subscribe to run-time exception and performance monitoring notifications from its agent on the server controller to help with this assessment.

A manager itself may be a business or operations support system (BSS/OSS), a network management system (NMS), or even an element management system (EMS). This document uses the term OSS to include all of these options. Further detail of OSS capabilities and inter-OSS communications is beyond the scope of this architecture.

As a special case, client and server may exist within the same trust domain, and it may be possible to simplify some of the architectural border crossings. Figure 4.17 illustrates a common management interface within a single trust domain, via the SDN controller.

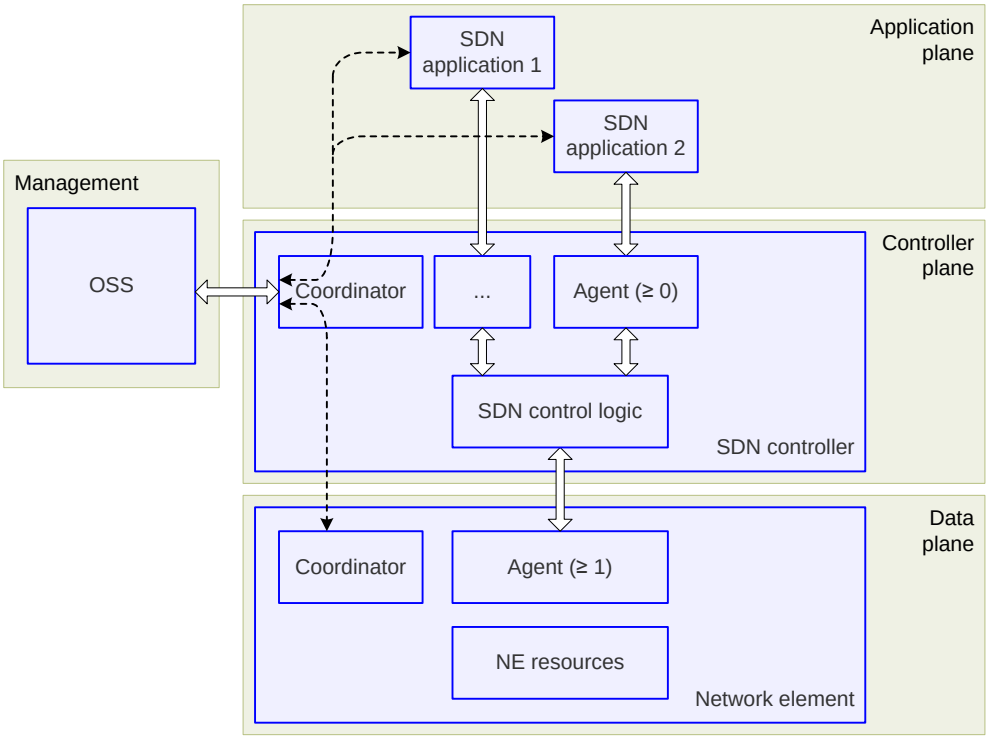

Figure 4.17 – Proxy management communications

The dotted lines in figure 4.17 illustrate how, within a common trust domain, the SDN controller may proxy management communications between an OSS and applications or network elements. This exemplifies a common case of SDN deployments in data centers (DCs). A typical application would be a cloud/DC management system, which is in the same trust domain as the

network elements and the SDN controller (note). The SDN controller implements some of the required management functions directly, and proxies those implemented elsewhere.

> Note – In this case, the cloud/DC management system would itself be responsible for security considerations related to the separate trust domains of the various cloud/DC tenants. In this sense, it acts as an SDN controller itself.

4.7 Information model

A variety of protocols may be used between the various components of a software-defined network, to serve a variety of purposes. While it is clearly desirable to minimize the number of protocols, the architecture does not mandate any particular protocols. What *is* essential is that all communicating entities share a common information model. This is not to be confused with a common data representation [7], which is a protocol issue. Indeed, when context permits, elements of the information model may be understood implicitly, rather than conveyed explicitly by protocol.

This architecture models SDN operation as the manipulation of managed object instances (MOs) across the various interfaces. Operations on MO instances include the familiar CRUD: create, read, update, delete, as well as invocation of methods defined on the MO classes and subscription to their notifications. As such, information modeling is a core function of the architecture and its evolution over time.

Information models are a key component in describing the architecture. This architecture recommends judicious re-use of existing information models. It is not necessarily expected that models from wider industry sources be directly and completely imported into an SDN information model. Adaptation to the SDN context may take into account both the special features of SDN and evolving best practices. However, the resulting SDN model should be chosen and documented such that it can be readily understood and used outside the SDN community. This compresses the learning curve and encourages migration by facilitating integration into existing infrastructure.

New information modeling requirements that may emerge from ONF work should be fed back into industry standards forums as the preferred route toward standardization.

5 Control functions and interactions

Assisted by coordination, control is at the heart of SDN. Focusing on these, as well as on recursion, this clause adds another level of detail to the principles and components introduced in clause 4. To summarize the difference between coordination (management) and control:

- The coordinator executes functions associated with the allocation of resources to clients and the bounding of these resources by policy. These functions occur within a single trust domain.
- Once a resource has been assigned to a client, it is effectively owned by the client's SDN controller (DPCF), which may use it in any arbitrary way permitted by the agreed policy. Controllers are modeled as residents of trust domains separate from their controlled resources.

For clarity, the text of this clause is organized into four progressively more complex scenarios, each with capabilities greater than the last. Accordingly, some of the material is repetitive. The architecture supports the most complex of these scenarios, but it is understood that the complexity of a particular implementation may be reduced to a greater or lesser extent, depending on the business or organizational circumstances of the stakeholders.

The four scenarios are:

1. Single player SDN provider
2. SDN provider with SDN clients, with underlying network exposed
3. SDN provider with virtualized network, non-recursive
4. SDN provider with recursive virtualized network

As before, administration Blue is taken to be the lowest-level infrastructure owner, while Green and Red represent customers.

5.1 Single player SDN provider

Ultimately, all networking is based on a set of physical network elements (NEs). It is useful to start at this level, and consider the set of NEs that forms one or more subnetworks within the control domain of a single SDN controller. Figure 5.1 illustrates such a subnetwork, owned and operated by provider Blue. NEs 1..*n* constitute the network control domain (NCD) of SDN controller $SDNC_B$ (subscript B for Blue). Everything in this clause occurs in the Blue trust domain.

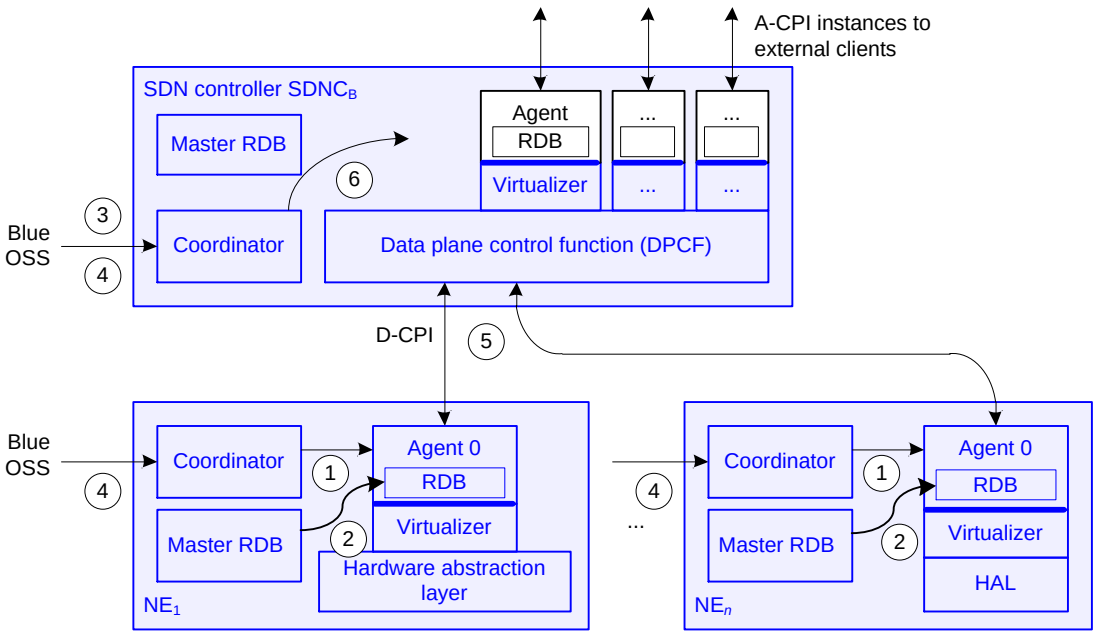

Figure 5.1 – SDN control of physical switches

Numbered circles indicate the logical sequence of activity when this network is placed under SDN control.

1. Each of the *n* NEs is understood to contain a coordinator function. It is unspecified how the coordinator is brought into existence. It may be instantiated by the Blue OSS, for example, or may exist as an artifact of the NE's software load.

   By way of the coordinator, the Blue OSS instantiates an agent on each NE, here designated agent 0 (note). The Blue OSS also instantiates a virtualizer to support agent 0; the virtualizer's function is to map resources assigned to agent 0 onto the hardware abstraction layer (HAL) of the NE.

   > Note – The value 0 is a convenient designator for this document, as a way to represent the provider's interest, not to be understood as a special reserved identifier.

2. Each of the *n* NEs is understood to contain its own master RDB, which models all of the resources in the NE (note 1). By way of the coordinator, the Blue OSS allocates resources from the master RDB to agent 0 (note 2), where they appear as an agent-local RDB. An agent represents the specific resources dedicated to a particular client, and represents an execution context for that client. In this case, agent 0 represents Blue's ownership of the NE's SDN resources.

   > Note 1 – It is unspecified how the NE's master RDB is initially populated, or indeed whether it even exists as a separate entity. It may be at least partly downloaded by the Blue OSS, for example, or may be populated by a discovery function in the NE at initialization time, or may simply be a view onto the NE's hardware and state.
   >
   > Note 2 – Not all NE resources need necessarily be subject to SDN control. One example of permanently off-limits resources is the identity, reachability and credentials package necessary for the NE to contact its OSS. Another example would arise if unsynchronized non-SDN protocols were responsible for some disjoint subset of the NE's resources (hybrid model).

3. The Blue OSS instantiates an SDN controller $SDNC_B$ on some platform or set of platforms. Its functions include a coordinator and a DPCF. Blue also initializes a controller master resource data base RDB in $SDNC_B$, and may partially or completely populate it from its own resource planning and inventory data base (note).

   > Note – NE and topology discovery may be available from the network itself (step 5). It would be appropriate for $SDNC_B$ to reconcile information discovered from the actual network against the provisioned RDB, and raise exceptions for positive disagreements.

4. Blue provisions controller and NE coordinators with the information they need to establish communications. Vital information includes identity, reachability (IP address, DHCP parameters, etc.), and security policy and credentials.

5. The NEs and the SDN controller establish communications. $SDNC_B$ reconciles its master RDB against the underlying resources, i.e., the union of the RDBs in the various NE agents 0. $SDNC_B$ may also discover or audit network topology or other meta-information that is not directly available from the NE agents 0.

These resources are now available for $SDNC_B$ to use in any way it desires. Interaction between controller and NE is modelled as a series of operations on the information model local to the agent 0 RDB in the NE.

6. By way of the coordinator in $SDNC_B$, the Blue OSS instantiates agents with virtualizers in $SDNC_B$, as needed to support the applications that it intends to offer to its application clients, and populates each with resources from the $SDNC_B$ master RDB, along with policy to govern the capabilities that it supports for that application client.

At this point, the Blue network is ready for SDN-controlled service to its application customers. As such, this configuration may represent the final target of some deployments (note). It is largely compatible with first-generation SDN specifications and implementations, which mostly contemplate implementation directly on hardware, and do not emphasize business or trust boundaries between control functions and the network.

Note – Figure 4.17 and figure 5.3 illustrate cases in which this configuration supports higher-layer applications that deal with multi-tenant issues.

Blue may also offer traditional telecommunications services from this configuration, that is, services controlled via a traditional management model, rather than by the SDN application model. Blue would be able to exercise SDN control of its NEs, but would not directly support network-aware applications. This could be a useful step in migrating the overall network toward SDN.

5.2 SDN provider with SDN clients, with underlying network exposed

Figure 5.2 shows the next step in the evolution of the scope and capability of the overall network. Here, provider Blue offers a virtual network SDN service to client Green, whose designator and subscript is G. The exposed virtual network is abstracted to include only selected ports and their supporting resources, but it rests directly on Blue's physical network elements. The VN exposed to Green could therefore be that of figure 4.12 or figure 4.13, but not the view of figure 4.14 or figure 4.15. Subsequent clauses relax this constraint.

Note – This configuration is not recommended for deployment in practice; it is included for explanatory reasons. Clause 5.3 describes the preferred implementation, which removes complexity from NEs in favor of the SDN controller, removes the constraint that a client virtual NE be contained in a single provider NE, and does not require direct connectivity between untrusted client SDN controllers and provider infrastructure.

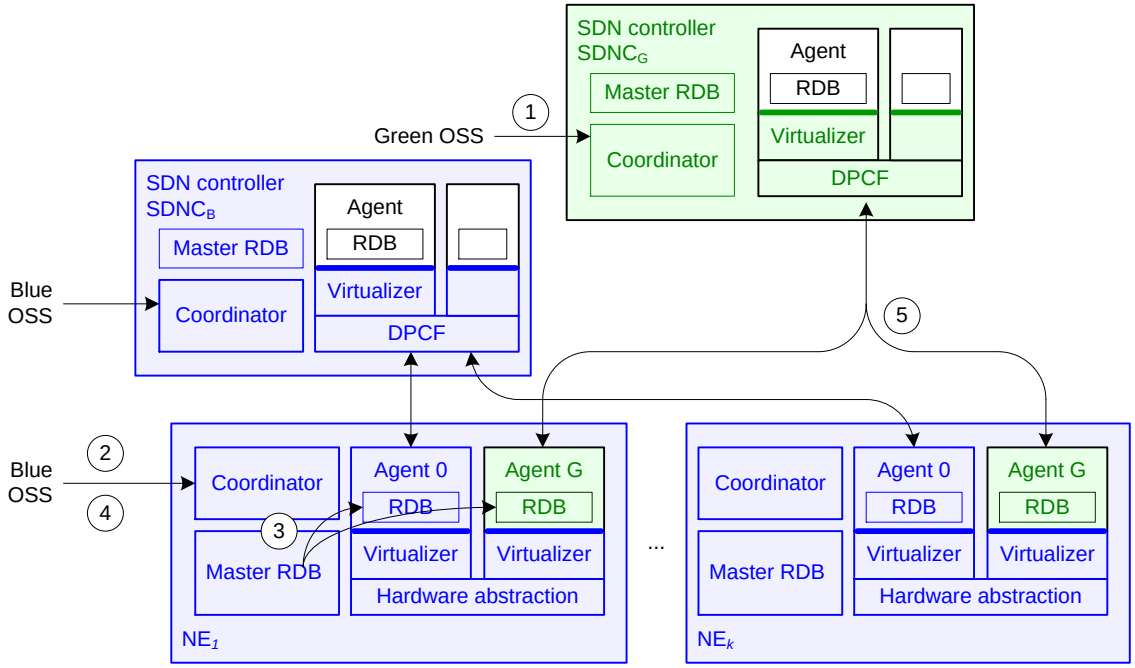

Figure 5.2 – Basic SDN network, adding client Green

Before network service is possible, Blue and Green must conclude a business and technical agreement. The ways in which this negotiation occurs are beyond the scope of this document. It is understood that both Blue and Green OSS are equipped with knowledge of their mutual technical commitments. As well as the necessary resources, the business agreement specifies data plane handoff points between Green's and Blue's networks. In some cases, these may be discoverable from the network itself, but they are usually expected to be provisioned into the controllers from the OSS (note).

> Note – As a matter of policy, it is expected that Blue will not expose its network ports promiscuously, nor will third party providers. If a signal from Green were observed on an unexpected Blue equipment port, for example because of mis-wiring, Blue would raise an exception, rather than accepting it, for security reasons if nothing else. Only in restricted environments such as data centers, might it be possible to regard ports as undifferentiated open-access pools.

The starting point is completion of the Blue configuration described in clause 5.1.

1. The Green OSS instantiates an SDN controller SDNC$_G$ on some suitable platform under its control. Maintaining architectural uniformity with clause 5.1, SDNC$_G$ includes a coordinator, a DPCF and at least the skeleton of a master RDB. Green may add virtualizers and application agents later, as described above in clause 5.1 step 6.

   The Green OSS may pre-populate its master RDB with an information model instance that represents part or all of the resources of the Blue network that have been assigned to it, along with auxiliary information such as permissions to take various actions on various resources (note). Examples of such actions include creating CFM MEPs or setting performance monitoring collection points with alert notification thresholds. These resources and policy

may guide $SDNC_G$'s execution, and may be used to audit the resources and capabilities that will become visible to Green at step 5.

Note – The Green master RDB could also contain information about directly controlled Green NEs, VNs contracted from other providers, or anything else of concern to itself.

The Green OSS provisions reachability and security policy into $SDNC_G$, which enables communication between the Green SDN controller and the various Blue NE agents.

2. Asynchronously, the Blue OSS instantiates an agent for client Green, designated agent G, on each pertinent NE.

3. In accordance with the business and technical agreement, the Blue OSS allocates resources and establishes policy for Green. The coordinator logically transfers resources to the agent G RDB. In figure 5.2, policy is illustrated as a heavy Blue bar that protects Blue from possibly unauthorized action attempts by Green (note). Policy enforcement resides in the virtualizer functional component.

   Note – The figures also show policy associated with Blue's own agents. Special aspects of provider policy are discussed below.

4. The Blue OSS provisions $SDNC_G$'s identity, reachability and security information into each NE. This permits communications between $SDNC_G$ and the agent G in each Blue NE. When Blue supports additional clients beyond Green, each client's agent-controller relationship has its own communications policy and credentials.

5. The Green SDN controller $SDNC_G$ and the Green agent in each NE establish mutual communications. $SDNC_G$ may upload or audit the RDB residing in each of the agents G. Having populated or reconciled its master RDB, $SDNC_G$ is then free to use the resources in any way it wishes.

The Green-Blue client-server relationship is many-to-many. Blue may support agents dedicated to additional clients according to the same process. The Green SDN controller may orchestrate any number of VNs from different providers, along with its own directly controlled NEs.

Green may now support its own application clients by instantiating a virtualizer and agent for each of them, and allocating resources from its own master RDB to the client agents.

Blue retains knowledge and ultimate power over all resources, whether assigned to another agent or not. It is not mandatory that a specialized Blue agent exist (note), or that it be designated 0. If such an agent exists, however, its policy may grant it extended or specialized scope and privilege. Provider agent policy may, for example, deny Blue operations on resources that have been assigned to Green, but accept a force option that overrides the deny.

Note – A case in which the Blue OSS would not instantiate a specialized Blue agent would occur if Blue's NE were not under SDN control, i.e., a hybrid NE.

Blue is responsible for dealing with issues that inalienably belong to the provider. Some of this responsibility may be directed from the Blue OSS through the NE coordinators; other operations may be directed by the Blue OSS, acting as an application client to $SDNC_B$, thence to the various agents 0, and executed within the bounds of associated privileged policies. Still other operations

may be driven by logic within the Blue SDN controller itself, for example reoptimizing resource allocations.

Blue provider functions include, for example:

- Administering resources that are implicitly or explicitly offered by Blue to Green as services. A primary example is a tunnel, which is a server layer network connection whose endpoints are visible to the client layer network. As well as the necessary connectivity, Blue would operate its own tunnel OAM, protection or recovery, performance monitoring, metering or policing, etc., as part of its service assurance process. Such server-layer details would be invisible to Green.
- Defect monitoring, alarm timing and declaration, propagation of failure information from the D-CPI through the virtualizers to the affected client agents.
- Infrastructure security monitoring, reporting of security alarms to the Blue OSS when exceptions occur. Security audit, security logging.
- Taking underlying Blue resources out of service (administrative lock) for maintenance purposes, and notifying the affected agents (operability state *disabled*).
- Allocating or reclaiming resources in the event of changes in the business relationship with the client, either transient (the client requests and releases scheduled or first-come-first-served resources within the scope of an umbrella business agreement) or permanent (client contracts for increased resources, no longer needs a given resource, or the client business relationship terminates). Resource reallocation may also result from network build-out or global network reoptimization.
- Administering resources that are contractually shared by several clients on a best-efforts basis, possibly with weights and priorities.
- OSS communications capabilities, as noted previously.
- Informing $SDNC_B$ of possible agent G failure.

At least some of these Blue functions affect the resources and state visible to Green. It is therefore necessary that the Blue agent 0 and coordinator be tightly integrated with agent G. The Blue agent 0 or coordinator may receive resource requests and exceptions from agent G, and exchange notifications or other information from the Blue resource space. In the example from above, resource lock by a Blue administrator must trigger at least a notification to $SDNC_G$, if not NE-local protection switching of Green resources.

Blue's integration with the agents G may be local to the NEs, or may occur in $SDNC_B$.

Policy is enforced in the virtualizer functional component. Characteristics of a policy include:

- It is common to allow a client to use any address space of its choice, at whatever layer it chooses, and to isolate clients by way of encapsulation (clause 4.5). The encapsulation technique is determined by provider policy; particular parameter values are stored in the agent RDB.

  Encapsulation is invisible to the client, both on the client's virtual data plane and on the controller plane. Commands and responses between the agent and the controller use the client's address space, but the agent must interpret them in light of the encapsulation scheme (note).

Note – If OpenFlow-switch is the control protocol, packet-in/packet-out functions as well as forwarding table entries must have their encapsulation stripped (or interpreted) in the northbound direction, and added in the southbound direction.

- Clients will wish to identify their resources according to their own conventions. Translation of identifiers between client and provider is done by the virtualizer. At least some of the client naming conventions can be captured and installed by the Blue OSS (in this case) in the RDB itself, as part of the business and technical negotiation on data plane interface points.
- The policy enforcement point must interpret events and actions in the context of the client, beyond just the translation of names. A hardware facility fault or administrative lock, for example, may translate into port down messages to several different client controllers, and may trigger any of a variety of protection or recovery actions in different clients. Assigning a resource to the client's agent is reported to the client as an object creation notification, while removing a resource is reported as an object deletion notification.

Both $SDNC_B$ and $SDNC_G$ may subscribe to notifications, invoke alarm reporting control, establish PM collection points with thresholds, etc., but Green only within the bounds of the resources it controls. As a special capability to facilitate SLA monitoring and troubleshooting between customer and provider contexts, $SDNC_B$ needs the ability to see the view (including notifications) presented to Green.

5.3 SDN provider with virtualized network, non-recursive

In clause 5.2, a client agent resides on each of the pertinent NEs. This constrains each virtualized resource to be wholly local to some individual NE. Conceptual resources that span multiple physical NEs must be expanded in the client SDN controller. The key difference in this clause 5.3 is that the client can contract for virtual resources that span multiple NEs, which are expanded on the server controller. Locating the provider's entire representation of the client's environment on the server SDN controller provides an integrated VN view to the client and also protects the server's NEs from direct exposure to the client's trust domain.

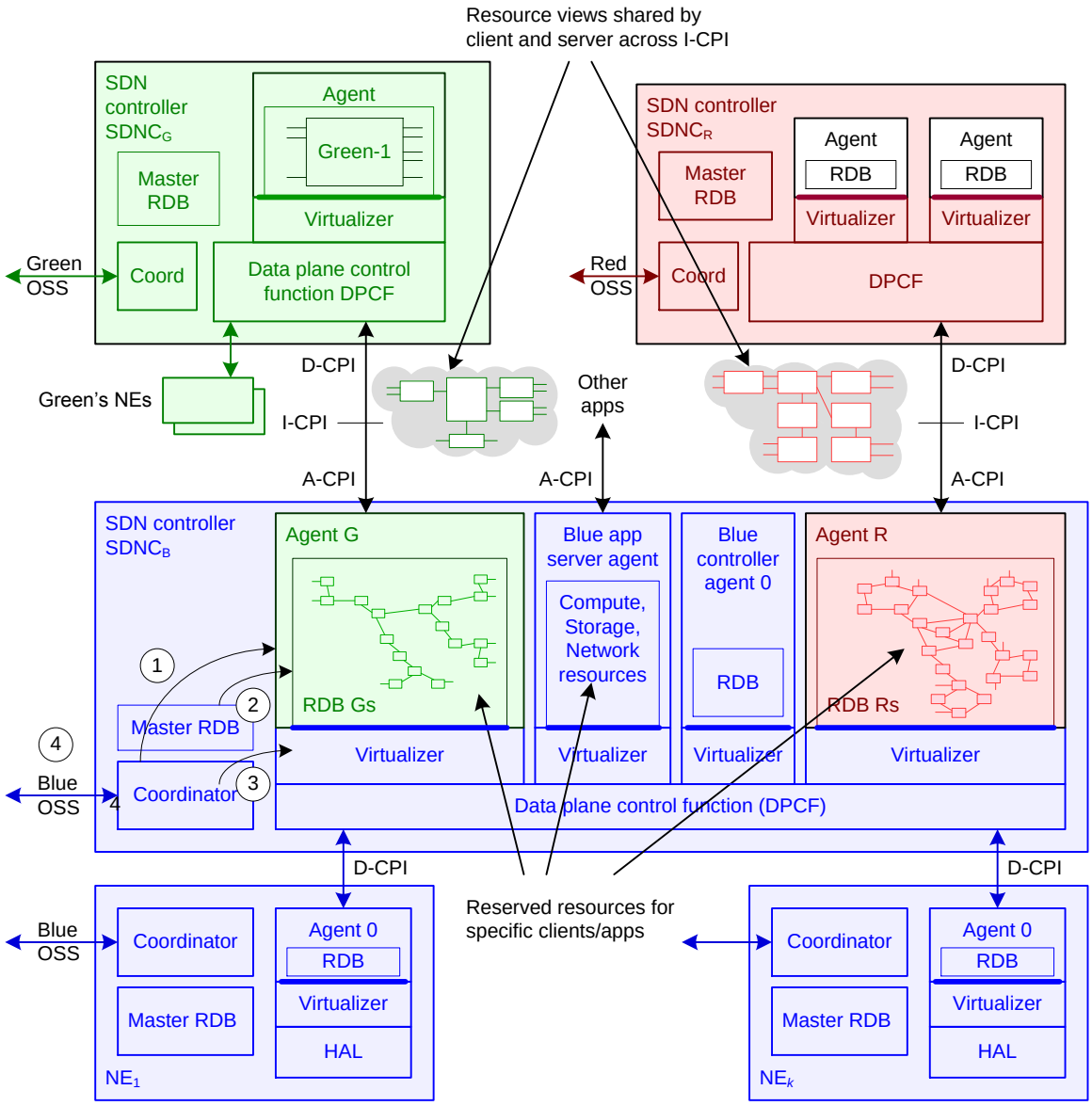

Figure 5.3 – Virtual network, single level control hierarchy

A VN contains at least one virtual NE (VNE). A VNE is an abstraction of a subnetwork, which is treated by a controller as if it were a physical NE. At one extreme (clause 5.2), there is a one-to-one mapping between each VNE and an underlying physical NE; at the other extreme, the entire VN is represented as a single NE (Green-1 in figure 4.15). Links internal to the VNE-subnetwork are concealed, while external ports are exposed. Unrestricted connectivity among like ports of a VNE (or unlike ports via adaptation) is not necessarily guaranteed. The ports on a VNE may ultimately be traceable to separate physical NEs. The cartoons shown at the I-CPI instances in figure 5.3 represent possible middle ground.

Green's SDN controller expects the virtual network to look like a real network, just as in clause 5.2. Green expects to conduct the usual operations, including, in particular, issuing simple forwarding rule entries against NE ports. Each such rule may, in fact, span an entire subnetwork. Blue implements these commands either by creating a series of forwarding entries in the

resources dedicated to Green (figure 4.13 implied here), or by mapping the command into pre-existing or dynamically created tunnels, or a combination of both.

Because a Green VNE may span multiple physical Blue NEs, it can happen that no physical NE is in a position to expand the resource abstractions desired by Green. The lowest level SDN controller whose view spans the given abstraction should be the one responsible for expanding the abstraction. Thus, the agent G environment necessarily migrates from the Blue NEs to the Blue SDN controller $SDNC_B$, where it appears as an RDB representing the entire Green VN, supported by a network virtualizer. These, in turn, deliver the necessary commands to an expanded number of Blue NEs. As well as expanding the scope of southbound commands from the Green SDN controller, the Blue virtualizer is responsible for consolidating information from the underlying network into a form that makes sense in terms of Green's virtual network. Figure 5.3 shows this situation.

Figure 5.3 also shows a second client, Red, with its own completely independent VN resource database, policy and virtualizer. For generality, the Green SDN controller is also shown as an orchestrator of its own NEs, as well as those visible from Blue.

Blue provisions client-specific information only on $SDNC_B$, not on Blue NEs. All actions on the Blue NEs are performed by the $SDNC_B$ DPCF onto the NE agents 0, over a trusted D-CPI that carries client information and actions only by way of the $SDNC_B$ virtualizers and policy enforcers. To the NEs, this is the same situation as described in clause 5.1.

The Blue OSS performs the following tasks for Green, and again for each additional client:

1. Instantiation of a client-specific agent on $SDNC_B$, for example agent G. This includes the creation of at least a skeleton master RDB.
2. Allocation of Blue resources from the master RDB in $SDNC_B$ to the agent G RDB by way of the coordinator functional component.

   As part of resource assignment, it may be necessary for Blue to establish tunnels and other hidden network services via $SDNC_B$ agent 0, along with the appropriate overhead (OAM, protection, PM). These represent simple link endpoint resources to Green, but to Blue, they are full connectivity services.
3. Instantiation of a virtualizer associated with the agent, and installation of policy. The virtualizer interprets client RDB operations to and from the underlying DPCF function. The virtualizer exposes Green's agent RDB to Green's SDN controller, possibly abstracted to less detail than the set of resources reserved in step 2, as suggested by the cartoon at the I-CPI. The difference in abstraction is a restricted view of the complete RDB contained within agent G.

   The virtualizer is also the policy enforcement point (PEP). The policy ensures that Green receives the contracted services, while protecting Blue's resources from misuse, intentional or otherwise. Policy also specifies what subset of the possible actions is available to Green, what information Green is permitted to query, and what notifications are available for Green's subscription. Policy defines the translation of identifiers between Green's designations and Blue's. In addition, the policy specifies how client traffic is encapsulated or translated, if it is necessary to isolate client address spaces from each other. As before, policy is shown as a heavy bar where Blue's virtualizer borders

Green's agent environment. The color indicates that the server (Blue) always owns and enforces the policy.

4. Establishment of IP connectivity, security credentials, etc., to permit $SDNC_G$ to communicate with its agent on $SDNC_B$, and thereby with its RDB.

   The Green OSS performs a complementary set of operations on $SDNC_G$. This is the same as described in clause 5.2, except that only one security association is essential (note).

   > Note – If the Green SDN controller functions in terms of individual sessions controlling individual NEs, it would be natural for it to establish a fully secured session to each of the Green VNEs (as in clause 5.2), not knowing or caring about their possible existence in a common agent RDB. Multiplexing these sessions into a single container session and a single security association would be desirable.

5. $SDNC_B$ may offer the added-value feature of responding to dynamic queries from $SDNC_G$ to claim or release additional resources. Such queries would have to flow through agent G, because there is no other connection between the Green environment and Blue.

   Access to additional resources would still be bounded by policy, but the policy would not necessarily have to be local to $SDNC_B$. It might be a meta-policy retained in the OSS, or possibly even negotiated by the respective operations support systems (OSS) on demand. In such a case, $SDNC_B$ might query the Blue OSS for a decision when it received such requests. When Blue grants or reclaims resources, it updates the $SDNC_B$ master RDB and the agent G RDB accordingly.

   Possibly through custom-downloaded feature packages, $SDNC_B$ may be able to respond to Green queries in the form of a what-if question, for example a request for Blue to propose resource options with a monetary or other figure of merit target, possibly constrained by maximum acceptable attribute bounds, from which Green intends to choose.

Figure 5.3 shows an agent 0 in the Blue SDN controller, which represents Blue's interest and ability to control Blue resources directly, either for Blue's own purposes or on behalf of the agents that serve Blue's clients (e.g., for tunnel establishment per step 2 above). The virtualizer and policy associated with this agent would be expected to have far greater scope and privilege than those associated with clients.

In addition, Blue may support applications directly from its SDN controller via one or more app server agents, each app defined in terms of an RDB and a policy.

The configuration of clause 5.2 is recommended for deployment only under special conditions. To summarize the advantages of this clause 5.3 configuration, as compared to that of clause 5.2:

- The Blue OSS need not reconfigure its NEs as clients come and go. All activity on Blue's NEs is conducted by way of the Blue NEs' agents 0, while $SDNC_B$ orchestrates all services delivered by Blue to its clients. Client-specific provisioning occurs only on $SDNC_B$.
- The granularity of Blue network virtualization is now completely arbitrary.

- NEs are no longer directly exposed to client SDN controllers, and no longer need agents or policies for clients. (Data plane interconnectivity is of course necessary, but carries far less security risk.)
    - In security terms, the network attack perimeter is reduced.
    - Overall complexity is essentially unchanged, but complexity has migrated into the SDN controller. NE software has become simpler.
    - Contention for finite physical resources, for example ports, queues or forwarding tables, are directly visible to the common controller $SDNC_B$ for arbitration and network-wide optimization.
- Non-SDN NEs may be able to support SDN principles with few, if any changes, because most of the SDN-specific features now reside in the controller.
    - It may be possible to adapt existing EMS or OSSs to mediate between SDN controllers and NEs, although possibly with penalties on performance or flexibility. This may be important in migrating legacy networks into SDN domains.

This clause 5.3 is the most important part of the controller discussion, because it localizes virtualization in the SDN controller. Clause 5.4 continues the exposition by describing recursive controller virtualization.

5.4 SDN provider with recursive virtualized network

Clause 4.1 explains the need for a recursive controller interface. The requirement for recursion implies that each level in the recursive hierarchy be sandwiched between like views, recognizing that special considerations may be appropriate at the lowest (physical) D-CPI and potentially at the final A-CPI handoff to a user who is not acting as a middleman. Support for recursion follows naturally from the paradigm that a client manipulates a common information model instance in a server at every CPI.

Recursive virtualization implies that any SDN controller's local view appears to be a D-CPI on its south side, or an A-CPI to its north, but may be an I-CPI from a wider perspective. Figure 5.4 shows how interfaces appear from a global, multi-level perspective. It should be recognized that a given VN may include both physical and virtual resources, for example if a provider offers some services from its own physical network, but subcontracts tunnels from a carrier's carrier. As such, interface designations such as D-CPI may be intrinsically ambiguous, partly physical, partly virtual.

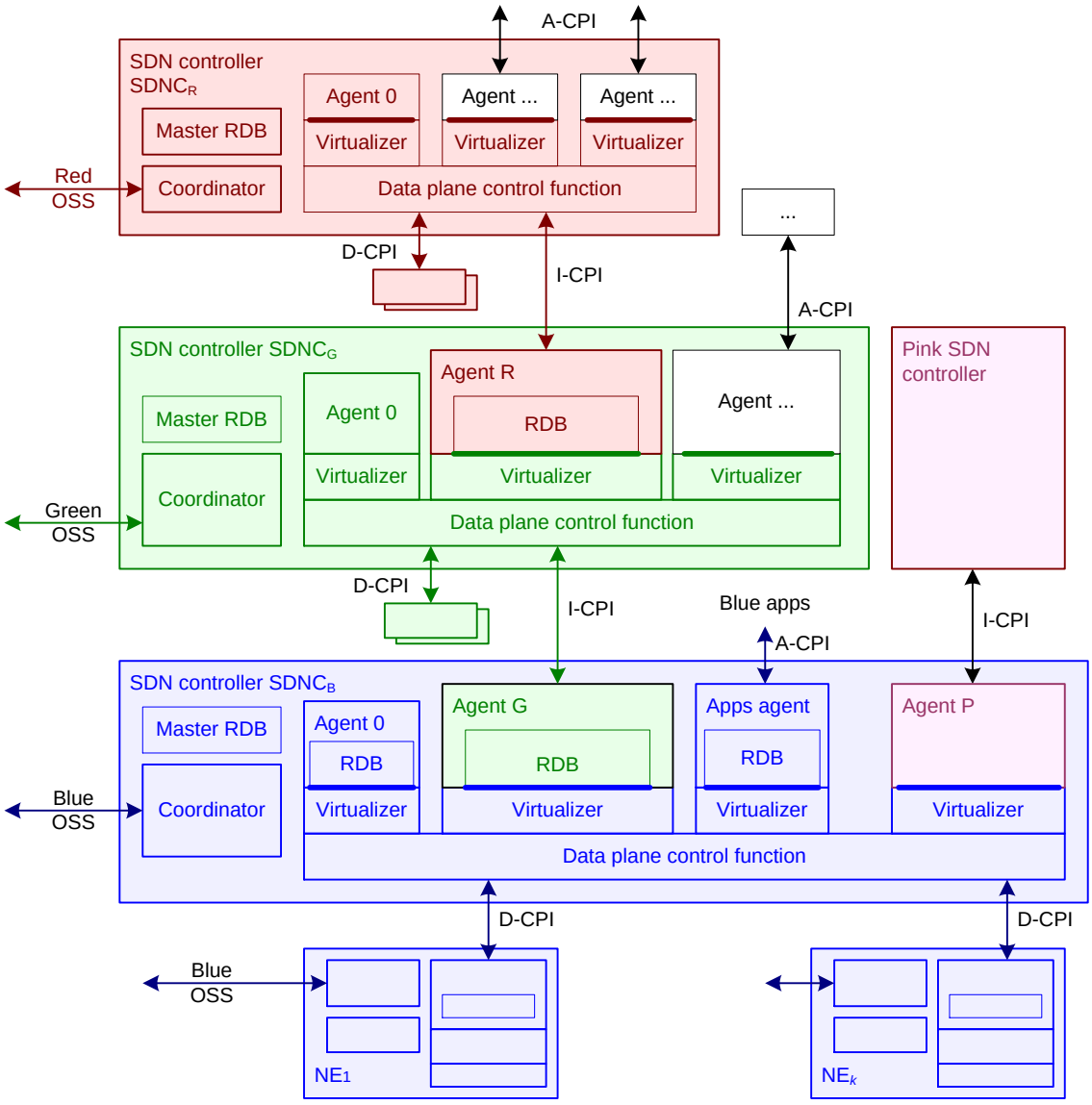

Figure 5.4 – Multi-level hierarchical architecture

Figure 5.4 illustrates a recursive client-server model. Blue owns a set of resources, shown as the lowest level of the figure, but not necessarily based 100% on a physical network. Blue delivers virtual network services to one or more clients, Green and Pink, for example. But in contrast to clause 5.3, Red is now a customer of middleman Green, rather than a direct customer of provider Blue.

Green's OSS and SDN controller behave the same with regard to the Blue server as noted in previous clauses; likewise Red's SDN controller and the Green server. The client (e.g., SDNC$_R$) has no visibility of possible further recursive levels below the server (e.g., SDNC$_G$), and the server (e.g., SDNC$_B$) has no knowledge of the applications offered by its client (e.g., SDNC$_G$).

5.5 Summary

By way of its OSS, a provider allocates resources and establishes an environment for each of its clients or applications.

Generically, an SDN controller acts as a server to expose a set of managed object instances to its applications, and satisfies the needs of these applications by acting as a client to some set of managed object instances in the data plane infrastructure, such as to satisfy the demands of its applications. Both application and data plane may be virtualized, leading to a recursive server-client hierarchy (see figure 4.1).

In a dedicated resource model, the server allocates a pre-negotiated set of resources (managed object instances) to its client. The set of available resources does not change except as the result of business relationship changes.

The dedicated resource model may be extended to allow for shared resources. In the shared-resource extension, the client may be allowed by provider policy to create additional managed object instances that represent resources, or to set attribute values that expand the capacity or capability of existing resources. The client may also be able to surrender resources whose cost is no longer justified.

The client SDN controller has full control of the resources allocated by the server.

Both client and server are responsible for ensuring transactional integrity, as required, across their respective domains of control.

The server and each of its clients exists in separate trust domains. A common trust domain is a special case.

The server enforces policy that guarantees the contracted level of service to the client, while simultaneously protecting itself and other clients from rogue behavior by the client.

The server is responsible for isolating clients' traffic. By agreement, the server may provide underlying network services to the client, such as tunnels or protection. These functions are invisible to the client.

Client code should not run within the server's trust domain. However, it might be allowed for the client to download applications software onto a computing platform that also supported part or all of the server's SDN controller. Because the information and state synchronization required of an SDN controller could not then be satisfied, such client code would be considered external to the SDN controller. Such code would then interface to the SDN controller via the controller's A-CPI reference point. The trust domain boundary at the A-CPI recognizes the undesirability of executing client code in the server's trust domain.

Although figure 5.4 and the previous figures show functional components inside the SDN controller, these are for explanatory purposes only, and are not architectural mandates. The SDN controller is a black box. Nothing normative can be said about its internal structure or interfaces, but the logical centralization principle requires that the functions share an information model instance and synchronize each other with regard to state. The SDN controller may contain other functional components, or may cooperate with external functional components such as a PCE.

6 Implementation considerations

While this clause cannot guide currently-existing implementations, it suggests criteria for the evaluation of current implementations, and factors to be considered in their evolution. Given its orientation to the future, it also identifies work in progress, or work that has yet to be undertaken.

6.1 Security

SDN security requirements may differ from those of a classical network due to their inherent characteristics and implementation choices. Depending on its physical implementation, centralized control may expose a single high-value asset to attackers, as distinct from a larger number of autonomous assets in a distributed control domain. Because logically centralized controllers are nevertheless likely to be implemented in distributed fashion, they may have additional implementation-related vulnerabilities that are not visible to the SDN architecture. That is, the architecture models the SDN controller as a single entity, implying a single secured session between controller and data- or application-plane entities, whereas the actual implementation may require multiple communications sessions because of physical distribution, each of which requires strong security. Operators are expected to mitigate some of the threats to a logically centralized controller by deploying SDN controllers within their secure computing environments.

A new class of threat arises because a software-defined network explicitly offers programmatic access to client controller plane or applications entities. These clients are typically separate organizational or business entities. This new business model presents requirements that do not exist within closed administrative domains, in terms of protecting system integrity and third-party data, in particular to ensure that business management and real-time control information of one entity is fully isolated from that of all others. Useful experience may be gained from existing automated interfaces between customer and provider business support systems.

On the other hand, the programmability feature also provides opportunities to enhance the security posture of networks. For example, it may be possible to use SDN techniques to construct a data plane security solution that is able to coordinate both network and security devices to detect and react to attacks in a more flexible way. However, the implementation of new data plane security functionality should not be achieved at the expense of overall system integrity and security.

Isolation of traffic between tenants is an existing security topic. In an SDN context, there are expected to be more components that could affect isolation, interacting more dynamically than in non-SDN networks. Both standards and operational practices need to ensure that isolation is not compromised.

Given the interconnection of different companies and organizations encouraged by SDN, the architecture is strongly driven by notions of trust domains with well-defined boundaries. A uniform interface model assists in thinking holistically about security issues, while strong boundaries help protect the rest of the network from trust domains with inadequate or compromised security. However, such trust domains cannot block threats posed by attackers who gain access to the SDN trusted domain (within the operator’s trust boundary), or who exploit weaknesses at applications or inter-domain interfaces exposed external to the trust boundary. The

architecture therefore requires strong authentication and robust security at all interfaces. Not unique to SDN is the fact that insiders represent a significant security threat, and that operator error threatens system integrity. To address this, the architecture should include strong identity and credential management functions that secure all entities and their associated state.

These and other factors point out the importance of audits to assure that the processes are working as expected. It may be required to allow real-time monitoring by a trusted third party. Real-time audits may be useful, and audit logs may be needed for forensic analysis and legal recourse. As such, it is important that the audit process and the logs themselves be protected from tampering and corruption. Implementations should restrict operator and process access to minimum authorized privilege levels, and partition the effect of impairments that may be caused by trusted actors.

Based on their own criteria, operators may choose to reduce the security restrictions on certain interfaces, for example at physically secure management ports on NEs and servers. It is important to understand the security implications associated with such implementation choices. In general, security is improved where security and administrative functions are manageable at scale, with minimal human intervention throughout their entire life cycles.

Experience has demonstrated the difficulty of retrofitting security capabilities into existing technologies (DNS and BGP are notable examples). However, SDN interfaces and protocols are being developed in the recognized context of escalating exploitation of technical and process deficiencies, with increasingly severe consequences. Therefore, it is critical that weaknesses previously addressed by non-SDN architectures not be repeated when building the SDN framework by securing the network architecture with a holistic approach to designing protocols and interfaces.

6.2 Flexibility

Decoupling of control and data planes encourages evolution of network infrastructure toward arrays of similar boxes, each with predictable, if not identical, feature capability, with all specialized features provided either outside the SDN domain, or better yet, by way of virtualized network functions. It remains to be seen how much of this can be achieved, even by individual operators, much less by the overall community, and how soon the necessary investment can be put in place. For the foreseeable future, heterogeneity is a fact of life in the network, which must be addressed by SDN.

A wide variety of network elements may thus be brought into the scope of SDN. It is theoretically possible to spontaneously install an arbitrary NE under the scope of an arbitrary SDN controller, at risk of incompatible capabilities, methods or protocols. In reality, however network planning will ensure a priori that, in addition to common protocols, the given NE and the given SDN controller jointly share the necessary feature richness and control capability. Different NE capabilities also pose a challenge to network virtualization, but it is a service planning issue, not a run-time issue. At run time, exceptions between controller and infrastructure may occur due to faults or resource performance or capacity limitations, but unsupported-capability exceptions should never occur.

It is necessary that underlying NEs expose their capabilities to the SDN controller, at least for audit and verification purposes. When the client view is of an abstracted NE or a non-degenerate

subnetwork, it may have unique capability and connectivity constraints that need to be reflected upward by the server controller. For example, even on a physical NE, some ports may be able to adapt traffic between two given types of characteristic information, while others may not. Another example is that some internal subnetwork connectivity may be protected, or possible to protect, while other connectivity may not.

Similar factors pertain to the A-CPI, complicated by the wide range of possible customers and applications and the expectation for rapid service fulfillment. It may be appropriate for providers to publish catalogs that define their commitments to support particular object models (possibly in the form of APIs). Where a catalogued entry suffices, the customer can gain service almost immediately, especially if the provider's SDN controller supports on-demand installation of feature packages and makes them available to a client's agent. Custom features and interfaces could still require development effort on behalf of both provider and customer. Over the course of time, the catalog would be expected to grow, until most customers could be satisfied with catalog offerings.

The point of this clause is to caution against the idea that arbitrary infrastructure, controllers and applications can be combined without careful planning and validation.

6.3 Distributed controller considerations

This document does not discuss the details of distributed controllers or distributed state, which involves describing the well-known challenges of implementing distributed systems. Essential aspects involve satisfying key distribution transparencies that hide various aspects of the underlying distribution from the user. Commonly cited transparencies (not an exhaustive list) include those of access, location, concurrency, replication, failure, and migration.

- Access transparency: having no apparent difference between local and remote access methods (syntactic and semantic consistency)
- Location transparency: ability to access any system component without needing to be aware of its location (details of topology of no concern to the user)
- Concurrency transparency: capability, for example, for various applications to access shared data/objects without creating interference among them
- Replication transparency: ability of a group of components to provide a single interface to others (e.g., if a system provides replication for availability or performance reasons, it should not concern the user)
- Failure transparency: ability for the failure of a component to be hidden from others, which can be achieved by using replication transparency (a group of system components providing a single interface to others can be made aware of the failure of one member and coordinate among themselves to assure others are not aware of it)
- Migration transparency: support for hiding of system component migration or reconfiguration, e.g., to provide better performance, reliability, etc. (should be of no concern to the user)

Considerable research has been, and is being, devoted to achieving various distribution transparencies, which is beyond the scope of this document.

6.4 Controller deployment

An SDN controller may use an SDN data plane for some or all of its internal or external interfaces, as long as the SDN controller does not rely for its connectivity on the operability of the data plane that it controls; otherwise, the SDN controller may find itself stranded or irrecoverably fragmented.

This must be considered when planning the deployment of SDN controllers relative to their controlled subnetworks, particularly if the SDN controllers are distributed, with some or all components located in clouds and subject to migration.

Many SDN scenarios optimize network resources, but assume connection-oriented services, in which an application establishes a route, a capacity and a QoS before beginning the data transfer. Real-time responsiveness of the controller to connection setup requests is generally not a factor in these cases. Other SDN use cases imagine real-time analysis of traffic flows and network load that may be at least partially chaotic (i.e., fractal, self-similar at any scale), with optimization in a closed feedback loop. It needs to be kept in mind that, in addition to measurement time and actuation time, the minimum time constant of such a feedback loop is affected by communications latency between network infrastructure and SDN controller. Additional delay would be incurred if the SDN controller were distributed and if feedback control required communication within the distributed components of the controller.

6.5 Interworking with non-SDN environments

Clause 5.1 describes how an SDN-controlled infrastructure can serve non-SDN application, while this clause extends the discussion in clause 5.3 about overlaying SDN applications on a non-SDN infrastructure. By bringing pre-existing networks under the umbrella of SDN principles, a provider can offer an SDN environment to its customers. That is, a customer may see SDN at an A-CPI, while the provider implements SDN on a partially or completely non-SDN network. The provider may do this by way of adaptation middleware, existing management or control systems, subcontracts to lower-level providers, or any combination thereof.

There already exist management protocols that communicate with data plane resources, for example NETCONF [8], SNMP, Corba, TL1, even command line interfaces. These protocols are suitable for setting up and monitoring quasi-static resources on non-SDN network elements, the information from which can then be made available to SDN controllers or exposed to customer applications. These protocols and information flows exist in currently deployed network elements, and in providers' OSS infrastructure. With no clear business case to replace them, they will be left in place indefinitely. It is therefore essential to exploit SDN principles in a mixed environment as much as may be possible, while recognizing that a full SDN environment is a long-term goal. The important principles to apply are unified control and global view.

Unified control means that a NE need never concern itself with conflicting demands on its resources. There are at least two ways in which this can be implemented.

1. An SDN controller could connect to the NE by way of a mediator function, for example an existing EMS, to translate its instructions into terms understood by the NE. No changes would need to be made to the NE; the mediator platform would need to be upgraded to perform the agent and virtualizer functions described in clause 5.
2. The NE could be upgraded to support the architecture described in clause 5, but some resources could be kept out of bounds. These resources would be controlled through non-SDN protocols, while SDN controllers were granted control of strictly separate resources.

In both cases, the conceptually monolithic SDN controller would remain responsible for maintaining a common, self-consistent view of information and state of the resources under its control.

The global view principle means that the SDN controller has full knowledge of its domain, where the meaning of *full knowledge* may vary depending on the need. For purposes of this clause, the question is how the SDN controller acquires this knowledge and keeps it up to date. Queries, polls or notifications from the network itself are of course an important source of information. Management systems may instantiate or update information in an SDN controller, for example about inventory. It was previously noted that an SDN controller may run routing or signaling protocols to communicate outside its own technology or administrative domain.

Interconnected SDN and non-SDN domains are likely to exist for some time to come. It will be appropriate for SDN controllers, or their associated back-office network and service planning systems, to understand how to construct services that continue to use existing NE capacity while exploiting the flexibility of SDN where it can add value. As described in clause 6.2, this implies intelligence in network and service planning to allocate functionality to the various nodes in a virtual network.

6.6 Management

SDN-specific management functions are described extensively above. In addition, network elements will continue to require all of the existing management functions, such as equipment installation and inventory and software upgrade. The protocols mentioned in clause 6.5 will continue to be used indefinitely in non-SDN NEs. Some of them, for example NETCONF, are candidates for use in SDN environments, as well.

The following topics also imply the continued presence of the OSS, but the precise division of effort between OSS and SDN controller is for further study, and may differ according to operator-specific criteria. It is expected that, over time, many current management functions will become the responsibility of SDN controllers and applications. Indeed, existing managers may find themselves in the role of application clients to SDN controllers.

Without intending to be a complete list, management functions include:

- Infrastructure maintenance: fault analysis, diagnostics, alarm correlation and management. Subscribing to PM threshold crossing alerts, monitoring PM counts. SLA monitoring with adjustments to billing records for SLA violations.
- Logging, especially security logs. Log storage, upload and retention.
- Configuration and service persistence, backup, restoration, auditing (clause 6.9).
- Traffic analysis, network and equipment planning, installation, inventory.

- Software distribution and upgrade. The centralized, global view available to the SDN controller may make it easier to temporarily route traffic around NEs that are scheduled for software upgrade or extensive equipment changes, thus reducing or possibly avoiding service disruptions that can occur in the existing network.

6.7 Inter-domain control communication

Taken to its extreme, the principle of logically centralized control suggests that a single SDN controller have worldwide scope. However, a real-world SDN controller probably doesn't encompass all endpoints of interest to its clients. Client services often extend into other technical or administrative domains. These domains may include non-SDN networks.

Particularly when interfacing with non-SDN-controlled networks that are already in place and working, it may be appropriate that an SDN controller run a variety of existing protocols, for example BGP or GMPLS. When talking directly to another SDN controller, the appropriate interface is a matter for further study, but may well be extensions of these, or other, existing protocols, for example the path computation element communication protocol, PCEP [9].

Following the principle of parsimony, new protocols should be developed only upon convincing evidence that no existing protocol can be suitably used or adapted.

6.8 Application-controller plane interface capabilities

The SDN service view may be very different from the underlying resource view. The descriptions in clause 5 use the wholesaler-middleman model to argue that the view, and the interface, at each level of hierarchy may be the same. From another perspective, the view and the interface may be quite different. This occurs when the SDN controller offers its clients something different from just a further abstraction of the same resources. The information needed for this may be configured, may be available from other sources, or may be an emergent property of the underlying resources.

- The A-CPI should support the ability to provide transactional integrity for applications that require it.

In general, applications are operated in their own trust domains, separate from the SDN controller trust domain. The A-CPI is strongly recommended to use the same agent-policy interface defined for other CPIs.

- The A-CPI must support strong security features when it crosses trust domain boundaries.

SDN is intended to offer its clients a wide variety of feature capabilities. Even ignoring the straight pass-through model, this variety makes it difficult to specify a single, universal A-CPI. Applications may support interfaces to other applications or SDN controllers in the role of clients, servers, or peers, and possibly in different roles at various times.

Networks have a limited set of functionality, which applications will wish to exploit in one way or another. Some of these requirements may be derived from clause 5. Not every application will require all of these features, and not every SDN controller implementation necessarily need support all of them. Downloadable feature packages assist with the desire to keep SDN applications open-ended.

The A-CPI specification should allow the following capabilities. Depending on its specific purpose, an instance of an A-CPI may not necessarily support all capabilities.

- To expose a full resource view, including virtual network topology.
- To directly expose the low-level information model.
- To expose abstracted views of its resources. This should be done with a specialization of the same information model that exposes full detail.

Many of the following functions illustrate client capabilities that, if offered by an SDN controller to its applications, would require support.

- To allow an application to set and query any attribute or state within the scope of its control.
- To allow an application to control traffic forwarding: to select traffic according to a set of criteria, to modify or adapt the traffic if necessary, to forward it to a given set of egress points. Selection criteria may be as simple as an ingress port or as complex as a vector that includes match fields up to and including layer 7, potentially spanning multiple packets of a flow.
- To allow an application to propose a traffic forwarding construct that requires the use of new or existing resources according to specified figures of merit, and to receive one or more offers from the controller. The client may propose and accept in a single request, or may review the offers and accept zero or one of them.
- To allow an application to invoke and control standardized functions such as STP, MAC learning, ICMP, BFD/802.1ag, 802.1X, etc.
- To allow an application to subscribe to notifications of faults, attribute value changes, state changes and threshold crossing alerts (TCAs).
- To allow an application to configure performance monitoring (PM) collection points, with thresholds, and to retrieve current and recent results.
- To allow an application to invoke and control traffic-processing functions through exchanges of opaque data blocks.

Separate work in ONF is assessing use cases for the A-CPI, which may extend or refine these criteria.

6.9 Network initialization

The behavior of an NE (agent) that loses contact with its SDN controller is subject to policy. Packet-oriented NEs may continue as before, with or without forwarding entry time-out, or may stop forwarding packets immediately. Typically, circuit-switching NEs would be expected to continue carrying traffic as before. OAM functions such as protection switching and alarm declaration would usually continue to operate. Operators may have specific requirements for such behavior, and implementers may allow for management provisioning of such aspects.

Related to these requirements are the questions of persistence, how much data should be stored on the NE, how it is to be backed up and restored, and how much functionality the NE should assume when it reinitializes locally, before having a controller connection. Whether this should be specified by the architecture is for further study.

If an SDN-controlled NE continues to function, or re-boots with a certain amount of functionality restored from local persistent memory, it will be necessary that the controller audit and reconcile the NE against some configuration and service database. Storage, backup and restoration of such a database is not presently specified.

These requirements imply considerable complexity. It may be acceptable for some kinds of SDN-controlled NE to maintain essentially no persistent data, and to initialize in a null state. This option would imply a longer time before service was available. The consequent deterioration of service availability or quality may need to be addressed.

In a hierarchical control arrangement, it is possible that a failure or reinitialization could affect only one, or a few, of the necessary levels between application and hardware. Recovery via protection or re-routing may be possible in some cases. In any case, it should be pre-negotiated how each level will behave in the event of the loss and recovery of its neighboring level. Factors for consideration include:

- The level of persistence of the state and behavior of one level during the absence of its neighbor.
- Assurance that local or network-wide initialization or restoration does not strand resources.
- Whether the neighbor can or should retain some useful state when it reinitializes.
- How the neighbors reconcile state when they re-connect.

The effect of NE or subnetwork reinitialization on external applications is not specified by the architecture, but may depend on service availability commitments. In any event, it is expected that hands-off recovery of affected applications be supported.

6.10 Integration with other initiatives

Several SDN-related activities are under way throughout the community. As a continuing effort, it is desirable that this architecture understand and integrate the value added by each of these activities. Contributions are solicited on all of the following topics, as well as additional list items.

Note – The inclusion of a reference is not to be understood as an endorsement by ONF.

Standards development organizations and industry forums

IETF

ALTO [15]

CCAMP [16]

FORCES [17]

I2RS [18]

NETCONF [19]

Netmod [23]

NVO3 [20]

PCE [21]

SFC [22]

IRTF SDNRG [38]

OIF – Optical Internetworking Forum [24]

MEF – Metro Ethernet Forum [25]

ITU-T [26]

ETSI ISG NFV – Network functions virtualisation [27]

Broadband Forum [39]

TM Forum [35]

Open-source work

Floodlight [29]

OpenStack [13]

OpenDaylight [30]

Open Vswitch [31]

OVDSB [32]

Ryu [33]

Trema [34]

FlowVisor [28]

6.11 Protection and restoration

Data plane protection and restoration models are well established for networks at all standardized layers. The introduction of SDN does not visibly imply changes to these standards. Rapid protection switching implies that the associated protocols and state machines continue to reside on network elements themselves, with the SDN controller responsible for pre-computing recovery resources, provisioning recovery behavior and subscribing to notifications.

At higher layers, the SDN controller may be responsible for restoring traffic itself, possibly by re-computing paths and re-routing traffic, possibly by re-optimizing resource allocation on a global basis, and possibly by triggering the migration of VMs to different physical locations in the cloud.

When underlying resources are shared by more than one client, the employment of redundancy must be planned to satisfy the needs of all such clients. This may be done in any combination of several ways:

1. Group clients into resource pools according to common availability and recovery time requirements.
2. Protect the underlying resources according to the most stringent requirement of any of the clients, and let the others ride free.
3. Offer a default level of shared resource protection, and require clients with more stringent needs to subscribe to dedicated resources.
4. Other.

7 Back matter

7.1 References

The following references may be useful background for SDN implementers.

[1] ONF, Software-Defined Networking: The New Norm for Networks (2012), available at https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[2] ONF, OpenFlow switch specification (2013), current and previous versions available at https://www.opennetworking.org/sdn-resources/onf-specifications

[3] ONF, OF-Config, OpenFlow management and configuration protocol (2013), current and previous versions available at https://www.opennetworking.org/sdn-resources/onf-specifications/openflow-config

[4] ONF, SDN architecture overview, available at https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf

[5] ITU-T G.805 (2000), Generic functional architecture of transport networks

[6] ITU-T G.800 (2012), Unified functional architecture of transport networks

[7] IETF RFC 3444 (2003), On the difference between information models and data models

[8] IETF RFC 6241 (2011), Network configuration protocol (NETCONF)

[9] IETF RFC 5440 (2009), Path Computation Element (PCE) Communication Protocol (PCEP)

[10] IEEE 802.1Q, MAC bridges and virtual bridge local area networks

[11] IETF RFCs, available through http://www.rfc-editor.org/rfc.html

[12] IEEE 802 standards and MIBs, available through http://standards.ieee.org/about/get/802/802.html

[13] Openstack, information available at http://www.openstack.org/

[14] Tussle in Cyberspace: Defining Tomorrow's Internet (2002), available at https://www.cs.duke.edu/courses/compsci514/cps214/spring09/papers/p347-clark.pdf

[15] IETF Alto working group, information available at https://tools.ietf.org/wg/alto/

[16] IETF Ccamp working group, information available at https://tools.ietf.org/wg/ccamp/

[17] IETF Forces working group, information available at https://tools.ietf.org/wg/forces/

[18] IETF I2rs working group, information available at https://tools.ietf.org/wg/i2rs/

[19] IETF NETCONF working group, information available at https://tools.ietf.org/wg/netconf/

[20] IETF Nvo3 working group, information available at https://tools.ietf.org/wg/nvo3/

[21] IETF PCE working group, information available at https://tools.ietf.org/wg/pce/
[22] IETF SFC working group, information available at https://tools.ietf.org/wg/sfc/
[23] IETF Netmod working group, information available at http://tools.ietf.org/wg/netmod/
[24] OIF, information available at http://www.oiforum.com/public/impagreements.html
[25] MEF, information available at http://metroethernetforum.org/
[26] ITU-T recommendations, information available at http://www.itu.int/en/ITU-T/publications/Pages/recs.aspx
[27] ETSI ISG NFV, information available at http://www.etsi.org/technologies-clusters/technologies/nfv
[28] FlowVisor, information available at http://onlab.us/flowvisor.html
[29] Floodlight, information available at http://www.projectfloodlight.org/floodlight/
[30] OpenDaylight, information available at http://www.opendaylight.org/
[31] Open Vswitch, information available at http://openvswitch.org/
[32] IETF RFC 7047, OVSDB, http://tools.ietf.org/html/rfc7047
[33] Ryu, information available at http://osrg.github.io/ryu/
[34] Trema, information available at http://trema.github.io/trema/
[35] TM Forum, general information available at http://www.tmforum.org/InformationFramework/1684/Home.html
[36] NGMN multi-SDO project, project material available at http://webapp.etsi.org/meetingDocuments/ViewDocumentList.asp?MTG_Id=30828
[37] NGMN, NG converged operations requirements, http://www.ngmn.org/uploads/media/NGMN_Next_Generation_Converged_Operations_Requirements_-_Final_Deliverable.pdf
[38] IRTF SDNRG, information at http://irtf.org/sdnrg
[39] Broadband Forum, technical reports available at http://www.broadband-forum.org/technical/trlist.php

7.2 Release history

ONF2013.225.14	Draft for formal review	17 Feb 2014
ONF2013.225.15	.14 comments resolved	18 Mar 2014
ONF2013.225.16	Last call draft for comments	21 Mar 2014
ONF2013.225.18	Released by arch WG	14 Apr 2014
Issue 1	Published	5 June, 2014

7.3 Contributors

Malcolm Betts, ZTE
Nigel Davis, Ciena
Rob Dolin, Microsoft
Paul Doolan, Coriant
Steve Fratini, Ericsson
Dave Hood, Ericsson
Mandar Joshi, Fujitsu
Kam Lam, Alcatel-Lucent

Ben Mack-Crane, Huawei
Scott Mansfield, Ericsson
Pascal Menezes, Microsoft
Sriram Natarajan, NTT
Manuel Paul, Deutsche Telekom
Makan Pourzandi, Ericsson
Jennifer Rexford, Princeton University
Jonathan Sadler, Coriant
Sibylle Schaller, NEC
Fabian Schneider, NEC
Silvio Shefer, ECI
Peter Smith, Microsoft
Jean Tourhilles, HP
Tina Tsou, Huawei
Eve Varma, Alcatel-Lucent
Maarten Vissers, Huawei
Marc Woolward, Goldman Sachs
Zhang Dacheng, Huawei