International Telecommunication Union

ITU-T

TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU

Y.3152

(04/2019)

SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES

Future networks

Advanced data plane programmability for IMT-2020

Recommendation ITU-T Y.3152

Recommendation ITU-T Y.3152

Advanced data plane programmability for IMT-2020

Summary

Advanced data plane programmability (ADPP) as an underlying technology for network softwarization enhances software-defined networking (SDN) with more agility and flexibility to meet the requirements of IMT-2020 networks specified in Recommendation ITU-T Y.3150. Recommendation ITU-T Y.3152 defines the advanced data plane programmability technology, which allows network operators to benefit from a "top-down" design process by defining network processing behaviour in a high-level language. In other words, the advanced data plane programmability enables network operators to define specific data plane protocol (including packet formats) and to support extended network functionalities. The advanced data plane programmability leads to flexibility and automation, which allows network operators to fully exploit data plane resources to enable their network applications.

History

Edition	Recommendation	Approval	Study Group	Unique ID[*]
1.0	ITU-T Y.3152	2019-04-29	13	11.1002/1000/13893

Keywords

Data plane programmability, IMT-2020, network programmability, protocol independent.

[*] To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

Table of Contents

Recommendation ITU-T Y.3152

Advanced data plane programmability for IMT-2020

1 Scope

This Recommendation describes requirements, architecture, functionalities, and reference points of advanced data plane programmability for IMT-2020 networks, which supports the requirements of network evolution and accommodates convergent services in IMT-2020 networks.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; all users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T Y.3150]	Recommendation ITU-T Y.3150 (2018), *High-level technical characteristics of network softwarization for IMT-2020.*

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 control plane [b-ITU-T Y.2011]: The set of functions that controls the operation of entities in the stratum or layer under consideration, plus the functions required to support this control.

3.1.2 data plane [b-ITU-T Y.2011]: The set of functions used to transfer data in the stratum or layer under consideration.

3.1.3 network slice [b-ITU-T Y.3100]: A logical network that provides specific network capabilities and network characteristics.

NOTE 1 – Network slices enable the creation of customized networks to provide flexible solutions for different market scenarios which have diverse requirements, with respect to functionalities, performance and resource allocation.

NOTE 2 – A network slice may have the ability to expose its capabilities.

NOTE 3 – The behaviour of a network slice is realized via network slice instance(s).

3.1.4 network slice blueprint [b-ITU-T Y.3100]: A complete description of the structure, configuration and work flows on how to create and control a network slice instance during its life cycle.

NOTE – A network slice template can be used synonymously with a network slice blueprint.

3.1.5 network slice instance [b-ITU-T Y.3100]: An instance of network slice, which is created based on a network slice blueprint.

NOTE 1 – A network slice instance is composed of a set of managed run-time network functions, and physical/logical/virtual resources to run these network functions, forming a complete instantiated logical network to meet certain network characteristics required by the service instance(s).

NOTE 2 – A network slice instance may also be shared across multiple service instances provided by the network operator. A network slice instance may be composed of none, one or more sub-network slice instances which may be shared with another network slice instance.

3.1.6 network virtualization [b-ITU-T Y.3011]: A technology that enables the creation of logically isolated network partitions over shared physical networks so that heterogeneous collection of multiple virtual networks can simultaneously coexist over the shared networks. This includes the aggregation of multiple resources in a provider and appearing as a single resource.

3.1.7 software-defined networking [b-ITU-T Y.3300]: A set of techniques that enables to directly program, orchestrate, control and manage network resources, which facilitates the design, delivery and operation of network services in a dynamic and scalable manner.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 data plane orchestrator: A software-defined networking (SDN) control plane function that allows multiple data planes of network slices to operate on data plane elements.

3.2.2 programmable data plane: A data plane that provides the ability to support new protocols and data processing procedures at run time.

3.2.3 protocol independent instruction set: A set of protocol-independent instructions supported by data plane elements required for implementing a wide range of network functionalities.

NOTE – The instruction means a code in a program, which defines and carries out an operation.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

ADPP	Advanced Data Plane Programmability
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CP	Connection Point
DDoS	Distributed Denial of Service
eMBB	enhanced Mobile Broadband
ICN	Information Centric Networking
I/O	Input/Output
IP	Internet Protocol
mMTC	massive Machine Type Communication
NFV	Network Functions Virtualization
PDP	Physical Data Plane
uRLLC	Ultra-Reliable and Low Latency Communications
SDN	Software-Defined Networking

5 Conventions

In this Recommendation:

The keywords "is required to" indicate a requirement which must be strictly followed and from which no deviation is permitted, if conformance to this Recommendation is to be claimed.

The keywords "is recommended" indicate a requirement which is recommended but which is not absolutely required. Thus, this requirement need not be present to claim conformance.

The keywords "can optionally" indicate an optional requirement, which is permissible, without implying any sense of being recommended. This term is not intended to imply that the vendor's implementation must provide the option, and the feature can be optionally enabled by the network operator/service provider. Rather, it means that the vendor may optionally provide the feature and still claim conformance with this Recommendation.

6 Introduction

Network softwarization technologies including SDN, network functions virtualization (NFV), network slicing and their extensions are expected to support IMT-2020 mobile networks. However, a gap exists between the current projection of the deployment of SDN, NFV and network slicing technologies and the requirements for IMT-2020. In particular, the current SDN technologies primarily focus on the programmability of a control plane, and the existing SDN protocol specifications reflect a "bottom-up" design process in which the capabilities of a data plane (also known as the forwarding plane) are determined by fixed function chips with built-in network protocols. In order to support new protocols and architectures driven by use cases in IMT-2020 networks, further work in the data plane is needed.

Advanced data plane programmability (ADPP) as an underlying technology for network softwarization enhances the SDN with more agility and flexibility to meet the requirements of IMT-2020 networks [ITU-T Y.3150]. With the use of advanced data plane programmability technology, network operators benefit from a "top-down" design process by defining network processing behaviour in a high-level language. In other words, the advanced data plane programmability enables operators to define specific data plane protocol (including packet formats) and to support extended network functionalities. The ADPP brings the smooth evolution from existing protocols to future proof protocols, and supports resource slicing and isolation over the programmable data plane. The ADPP facilitates efficient and automated deployment of new network services, which permits developers to fully exploit data plane resources to enable their network applications.

Figure 6-1 shows targets of ADPP in the case of network slicing. Multiple network slice instances having their own data planes over infrastructure, generally consist of network, computation and storage resources. Functionalities of data plane elements, which have the capabilities of data forwarding, routing and/or data processing, are able to be changed by the framework of ADPP.

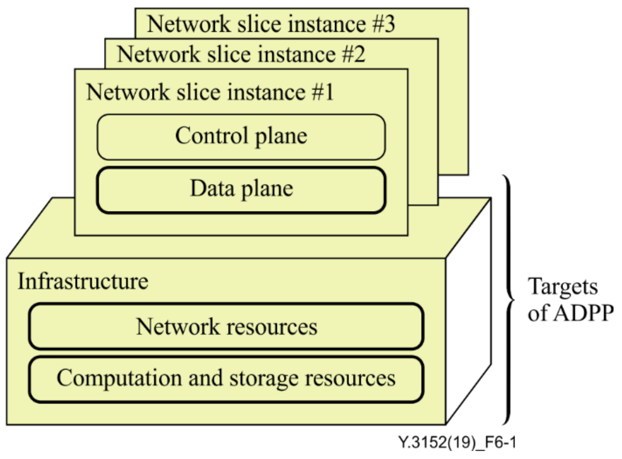
Network slice instance #3
Network slice instance #2
Network slice instance #1
Control plane
Data plane
Infrastructure
Network resources
Computation and storage resources
Targets of ADPP
Y.3152(19)_F6-1

Figure 6-1 – Targets of advanced data plane programmability

This Recommendation specifies requirements and architecture of advanced data plane programmability for IMT-2020 networks.

7 Requirements of advanced data plane programmability for IMT-2020 networks

7.1 Principal requirements

The IMT-2020 network is requested to have the capability to provide the ADPP that allows users to define and realize new protocols and mechanisms with the following principal requirements:

– A high-level programming environment is required to enable programmers to decide the behaviours of an entire network in a centralized way, rather than per-device programming.

– A set of application programming interfaces (APIs) for ADPP is required to support protocol-agnostic data forwarding and processing.

7.2 Requirements on data plane (i.e., instructions, pipelines)

The IMT-2020 network is requested to have the capability to provide data plane programmability that allows programmers to create, modify, or delete the packet forwarding and processing functions via protocol-agnostic programming APIs.

– It is required to have the ability to maintain an instruction set between an SDN controller and a data plane.

NOTE 1 – The instruction set describes the basic packet processing primitives, such as checksum and encapsulation, which is not designed for any specific protocols, services or applications.

The following are requirements of the instruction set:

– The instruction set is independent of the northbound interface between the SDN controller and applications.

– No matter what kind of northbound interface is used, the SDN controller is required to be able to translate a forwarding process into protocol-independent instructions, and install them to data transport elements.

– Instructions can optionally apply any arbitrary packet field for manipulating the field data without specifically supporting the particular protocol packet formats for data plane.

NOTE 2 – Programmers can use the instruction set dynamically to build custom packet processing functions at any given time and as needed. The instruction set is not bound to any specific protocol. Therefore, any new protocol can be processed.

– It is recommended that any protocols, policies and services are realized using the combination of the instructions assembled by the SDN controller, i.e., future-proof operations can be achieved.

– The data plane is recommended to support a pipeline.

NOTE 3 – The pipeline contains multiple stages of forwarding tables, and supports the branch structure in a table in order to support the forwarding of matched packets to different tables for the next stage processing.

– The pipeline is recommended to be dynamically constructed and modified at run time.

NOTE 4 – The data plane should provide programming interfaces for the control plane to dynamically create and modify the pipeline, such as appending tables to the pipeline.

– The data plane is recommended to have the ability to provide differential services to different pipeline branches.

NOTE 5 – The data plane should provide programming interfaces to enable the control plane to specify the service parameters. For example, the maximum packets or bits are allowed to be forwarded per second, and the queuing priority defines output by rules in the pipeline branch.

– Instructions for dedicated time-sensitive functionalities are required for realizing the high performance of the data plane.

  NOTE 6 – An example in this regard, includes time-based scheduling functions for synchronous transmission and line-speed read/write functions for in-network storage.

7.3 Requirements on high-level network programming

IMT-2020 network is requested to have the capability to support a high-level network programming that should be independent from a specific network protocol and a network configuration (e.g., topology). Thus, the following requirements apply:

– The high level programming language is required to facilitate programmers to effectively program network functions with APIs.
– Programmers can optionally develop network programs to define the network protocol and describe a network policy with the high-level programming language.
– For the control of a configured switch data path, runtime libraries are recommended to be auto-generated from the configuration of data plane elements.
– Data transport element, such as a switch, is recommended to support reconfiguration during operation, and allow reconfiguration without interrupting packet processing.
– A compiler is required to have the capability to translate network programs into pipelines for individual data transport elements that correctly enforce the policies according to the network programs.
– The compiler is required to have the capability to maintain and optimize the pipelines by deploying or revoking forwarding rules that correspond to the status of network traffic in data transport elements dynamically.
– The compiler is required to be independent of switching platforms.

  NOTE – This means that the compiler works on a wide range of switching platforms including application specific integrated circuit (ASIC), network processors and software switches.
– The network programming is recommended to enable network operators to manage and orchestrate softwarized network components.
– It is recommended to deploy new protocols and network functionalities to the data plane seamlessly without interrupting existing network functionalities.

7.4 Requirements on data plane of network slice

IMT-2020 network is requested to have the capability for network slicing. The following requirements apply:

– The data plane of a network slice is recommended to preserve the programmability of data plane elements.
– The data plane of the network slice is recommended to customize logical topology, link and node capacity.
– The data planes of network slices are recommended to support diversified protocols and data transport mechanisms.
– Different data planes of network slices are required to be controlled by independent and isolated control planes for network monitoring and management.
– A data plane orchestrator is recommended to have the capabilities to create, modify, and remove data planes of network slices with customized logical network topologies, protocol stacks, links and node capacities.
– The data plane orchestrator is recommended to provide separation among multiple data planes of network slices.

NOTE – Each isolated data plane of network slices should be realized using an independent network protocol stack, network functionalities, and a control and management channel.

8 High-level architecture of advanced data plane programmability for IMT-2020 networks

8.1 Overview

The high-level architecture of ADPP in IMT-2020 is introduced as Figure 8-1.

The high-level architecture contains the following three functional layers:

– Application: Requests program and control plane to fulfil its own service requirements.

– Program and control plane: Includes high-level network programming environment, a SDN controller and a data plane orchestrator. This plane defines programmable data plane behaviours.

– Programmable data plane: Includes a protocol independent instruction set and data plane elements designed by the program and control plane.

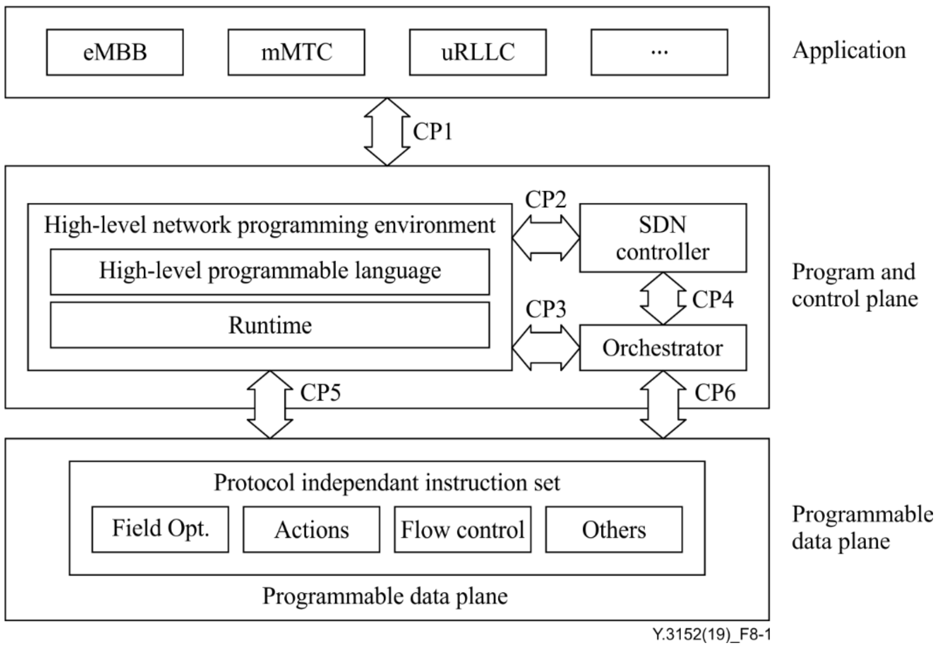

Figure 8-1 – High-level architecture of advanced data plane programmability

Due to the ADPP, the data plane orchestrator enables multiple data planes of network slices to coexist over the shared data plane resources, and for each network slice, a high-level programming environment and APIs are provided to manage device-level resources and per-flow operations. It is beneficial to support massive flexibility depending on different service scenarios and requirements, which is a major differentiation from existing networks.

The key components shown in Figure 8-1 are introduced in clauses 8.2 and 9.

8.2 High level network programming environment

The key components of a high-level programming environment are a high level programming language and a runtime system.

8.2.1 High level programming language

The high-level programming language should have at least two parts, that is, a packet definition and network policy description.

a) In the packet definition part, programmers are allowed to define header formats of network protocols that are supposed to support the protocol independent data plane. Programmers can define protocol headers with variable lengths and optional fields, and also define the protocol stack with variable sequences of protocol headers.

b) In the network policy description part, a programmer is allowed to describe a network policy using a set of protocol-agnostic programming APIs.

At least the following four types of APIs should be supported:

– Packet parsing APIs: A programmer can use the APIs to construct a packet parser from the packet definition part, and parses incoming packets accordingly. The packet parsing APIs are protocol agnostic, and they extract the header and field names from packet definition arguments.

– Packet field manipulation APIs: A programmer can use the APIs to arbitrarily manipulate a specified packet field, such as read, write, modify, shift, testify, insert, remove, etc.

– Path construction APIs: A programmer can use the APIs to construct a path on the network topology that packets should be forwarded through.

– Environment APIs: A programmer can use the APIs to query environment information such as the network topology and throughput.

8.2.2 Runtime system

A runtime system should fill a gap between the high-level programming language and low-level data-plane forwarding instructions. The runtime system produces and updates pipelines for data plane elements.

– When a policy is made or changed by a network program, the runtime system should update the policy in its memory.

– The runtime system generates optimized pipelines, which contain instructions for implementing policies, on data plane elements.

8.3 Data plane orchestrator

A data plane orchestrator has the following two major functionalities:

a) Management for the data plane of a network slice: The orchestrator manages the mapping between logical data plane elements of network slices and network resources. The following isolations should be managed:

   – Topology isolation: Adding, removing, or modifying network elements in one data plane of a network slice should not have any influence on data planes of other network slices.

   – Traffic isolation: Network packets belonging to one data plane of a network slice should not be forwarded to network elements that do not belong to the corresponding data planes.

   – Resource isolation: The network resources allocated to one network slice should not be used to provide services to other network slices in order to avoid the risk of exceeding the capacities of the resources.

b) Runtime support for the data plane of a network slice: A runtime system is supported by the orchestrator to provide instructions or optimized pipelines, which include different protocols and network functionalities, for a specific data plane of a network slice.

8.4 SDN controller

An SDN controller should have the following capabilities for ADPP:

– Provide the message channel with the data plane elements.

– Handle the outages of resources (e.g., for switch and/or link).

– Other basic control plane functionalities such as topology discovery.

8.5 Programmable data plane

A programmable data plane is able to support new protocols and data processing procedures on data plane elements at run time. With ADPP, the application requirements are programmed by the program and control plane, and then deployed to a programmable data plane.

8.5.1 Protocol independent instruction set

A protocol independent instruction set is capable of handling both existing and new protocols. It allows programmers to program data path processing according to both existing and new packet formats and behaviours.

The protocol independent instruction set consists of low-level instructions that allow programming of a wide variety of data path functions, which are not limited to any pre-configured protocol behaviour in data transport elements. The primitive instruction set should include minimal instructions such as:

– pipeline manipulation;

– insert/remove header; set/copy field; add/sub/inc/dec field; boolean operations and logical shifts on fields;

– execution controls (e.g., jump, conditional jump, goto table);

– actions (e.g., forward, drop);

– some notion of function calls (e.g., invoking hardware accelerators).

9 Reference points for advanced data plane programmability

Data plane programmability uses the following reference points:

Connection point 1 (CP1) represents the reference point between an application layer and a program and control plane. The application layer implicitly or explicitly treats requirements relating to resource handling, and send it to program and control plane through CP1. The requirements can also define a new network protocol, and describe a new routing scheme, etc.

CP2 is the reference point between a high-level network programming environment and an SDN controller. The programming environment sends commands to the controller. The controller can then report events to the programming environment. For example, on receiving a controller's event requesting to setup a new network flow, the programming environment plans a path in data plane, and sends commands to the controller for enabling the path through CP2.

CP3 is the reference point between the programming environment and a data plane orchestrator. Through CP3, the programming environment makes the configurations for data planes of network slices, and events from data planes of network slices are reported to the programming environment. For example, if the data plane orchestrator receives the event that a switch on the data plane of a network slice is overloaded, the orchestrator reports it to the programming environment through CP3. The programming environment recalculates the resource allocation for the data plane of this network slice, and sends the updated configuration to the orchestrator through CP3.

CP4 is the reference point between the SDN controller and the orchestrator. The controller can send instructions to data plane elements (e.g., switches) of a network slice via CP4, and data plane elements of the network slice can also report events to the controller through CP4. For example, if a new network flow in a network slice should be setup, the controller can directly plan a path on the corresponding data plane or receive a planned path from the programming environment, and send instructions to the orchestrator for enabling the path through CP4.

CP5 is the reference point between the programming environment and programmable data plane elements. Through CP5, the programming environment can send instructions to elements of data plane, and elements of data plane can also report their events to the programming environment. For example, if a new network flow should be setup on data plane elements, programming environment can plan a path on the network, and directly sends instructions to data plane elements for enabling the path through CP5.

CP6 is the reference point between the orchestrator and data plane elements. Through CP6, configurations of data planes of network slices are sent to data plane elements. The data plane elements may report the events to the orchestrator. For example, if a link failure happens in the data plane of a network slice, the orchestrator receives the failure event through CP6 and reports it to the other functions. The orchestrator receives the new configurations from programming environment or the controller, and re-configure the data plane of this network slice through CP6.

10 Security considerations

The introduction of advanced data plane programmability inevitably raises security challenges. Different kinds of security threats should be considered.

Data plane is vulnerable to attacks. For example, it is possible for a malicious programmer to program data plane network elements to perform various attacks, such as eavesdropping on other applications' traffics, or launching a distributed denial of service (DDoS) attack. To provide counter measures, network applications should be granted appropriate privileges before invoking certain instructions on the data plane network elements. The programming and control plane should have defense mechanisms against the DDoS attacks, and in addition, access control to the control and programming plane is also required.

The programing environment is also subjected to undesirable behaviours. The presence of undesirable behaviours has caused numerous problems in some programming languages, including bugs and serious security vulnerabilities. There are a few places where evaluating a ADPP program can result in undesirable behaviours, such as, out parameters, uninitialized variables, accessing header fields of invalid headers, and accessing header stacks with an out of bounds index. The undesirable behaviours in ADPP should be preempted as much as possible, and the validation of programs, which may include programs' performance, should be conducted before their running. For example, given the concern for performance, compiler flags and/or programs can be defined to override the safe behaviour. Generally, it is expected that programmers should be guided toward writing safe programs, and encouraged to think of a way to allow exceptions.

Appendix I

Potential use cases supported by advanced data plane programmability

(This appendix does not form an integral part of this Recommendation.)

This appendix introduces four typical use cases of advanced data plane programmability outlined in clauses I.1 to I.4.

I.1 Use case of line speed in network storage with ADPP

In the in-network caching, the switch and storage functions are usually deployed in a single node, while the throughput of switch is much larger than the storage. If they work together, the unbalance workload may result in very low storage efficiency. With ADPP, the in-network storage is divided into two stages, switch and input/output (I/O) operation, which are executed in switch end and storage end separately (see Figure I.1). Switch end implements packet-switching operations. The request packet that requires I/O operation is first translated into a new request packet based on the proprietary protocol by switch end, and forwarded to the storage end. This process can efficiently offload the switch end, and achieve high in-network caching performance.

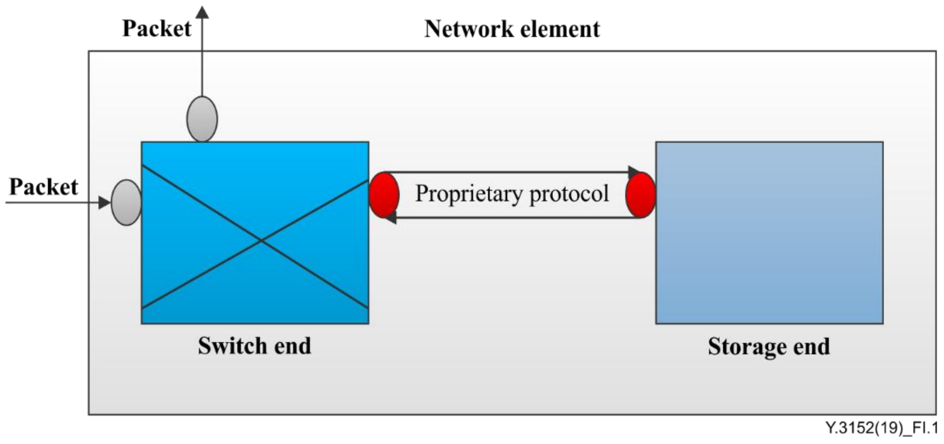

Figure I.1 – Use case of wire-speed in-network storage with ADPP

A typical procedure is as follows:

1) The request packet that requires cache operation (i.e., read) is translated into a new request packet which conforms to proprietary protocol by using protocol independent instruction set.
2) The new request packet is forwarded to storage end.
3) Once storage end accomplishes cache operation, a response packet containing the data of the I/O operation will be sent back to switch end.
4) The process of data packet that requires cache operation (i.e., write) is similar.
5) Switch end merely implements simple packet-switching operations, not including any block I/O operations, it is possible to guarantee wire-speed forwarding at switch end.
6) When designing proprietary protocol, there are two basic rules. One is that switch end can generate such a packet quickly by using protocol independent instruction. The other is that both switch and storage ends can process this packet quickly. Hence, a proprietary protocol packet has a forwarding-friendly and easily decoded format.

I.2 Use case of data security with ADPP

ADPP can provide the security protection mechanism of the data plane which is tailored according to the relevant security policies to meet differentiated data transmission protection requirements of different services.

The network nodes in the programmable data plane (see Figure I.2) can check identity and authentication using the instruction set without offloading mechanism. The data security specification in the application layer, such as data transmission protocol policy and distributed data authentication policy, will be processed by the high-level network programming environment or directly by the controller into the instruction set and installed into the nodes in the programmable data plane. One node in the programmable data plane will check the identity of a data packet to allow it to pass, or check the authentication to decide triggering an alert event.

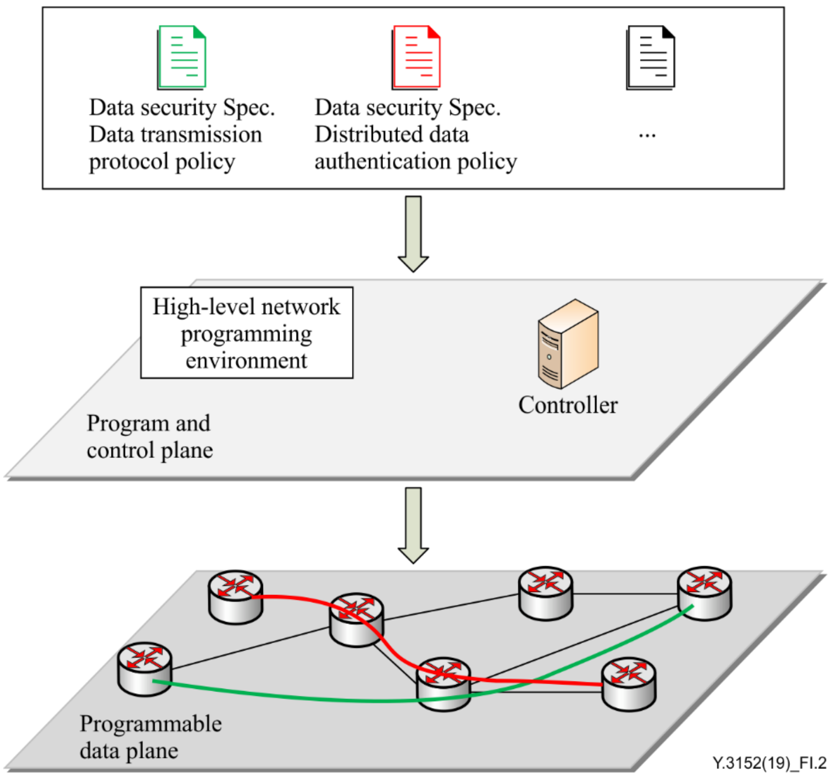

Figure I.2 – Use case of data security with ADPP

I.3 Use case of programming for ICN and IP network slice with ADPP

In the scenario of programming for information centric networking (ICN) and Internet protocol (IP) network slices, two types of network slices with different protocols should be created (see Figure I.3). With ADPP, developers can submit their requirements to the orchestrator via SDN controller to create the network slices, and they can program the specific routing policy on different data planes of network slices through the programming environment. Programs will be processed by the runtime and then deployed to the physical data plane (PDP).

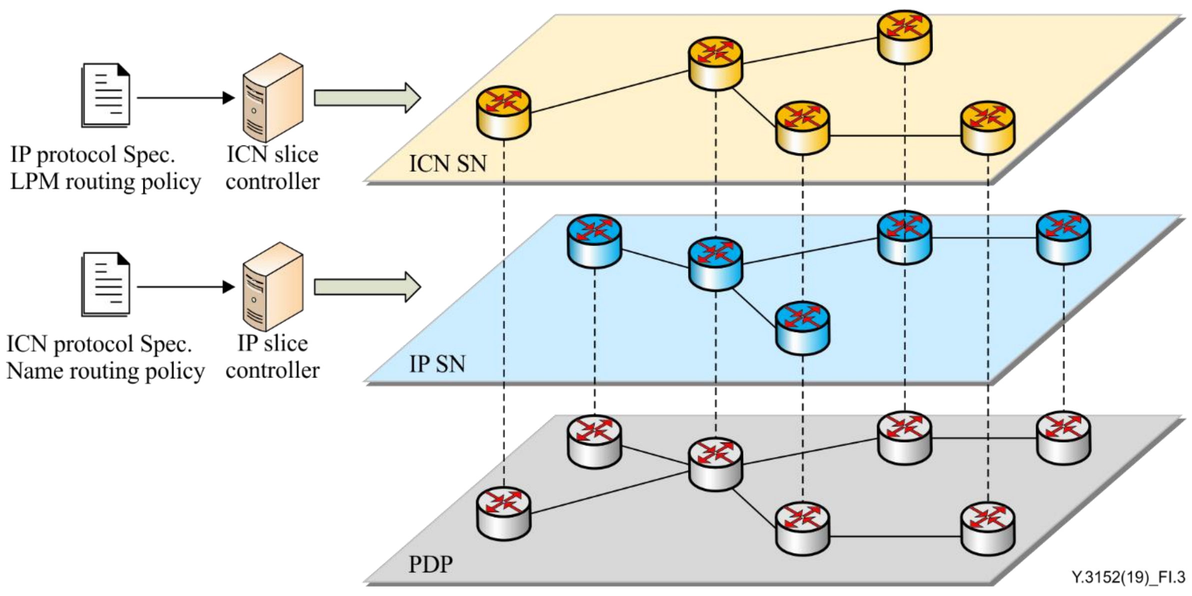

Figure I.3 – Use case of ICN and IP network slice programming with ADPP

A typical procedure is as follows:

1) The application developer of the IP and ICN applications submit their network slice requirement to the orchestrator via the SDN controller.

2) The orchestrator creates two data planes of network slices from the data plane elements, and it also creates virtual machines as the controllers for network slices.

3) The IP application developer composes the IP packet specification and the longest prefix match-based IP routing policy.

4) The programming environment of the IP network slice takes the programs as input and produce instructions to the orchestrator, which translate the instructions and deploy them to the physical data plane.

5) Similarly, the ICN application developer composes the ICN packet specification and the ICN routing policy, and the programs are compiled and executed in the programming environment of the ICN controller, and eventually produces instructions for deployment on the physical data plane.

I.4 Use case of multi-network access with ADPP

A user entity belonging to multiple networks, such as mobile network and Wi-Fi, is a typical scenario in IMT-2020. In this scenario, the user entity usually has multiple IP addresses.

Figure I.4 shows the architecture of multi-homing networks. The control plane is used to specify the forwarding rules of the flow and then deliver them to the switches of the data plane. The ADPP switches in data plane forwarding the packets according to the flow tables. Since the host has multiple accesses and addresses to the network, the server could use the multiple paths to transmit the content to the user. With the ADPP, the IMT-2020 network has the ability to operate the IP address, such as adding, deleting or changing, which may help the network to deliver the packets through the proper path to one or more specific destinations.

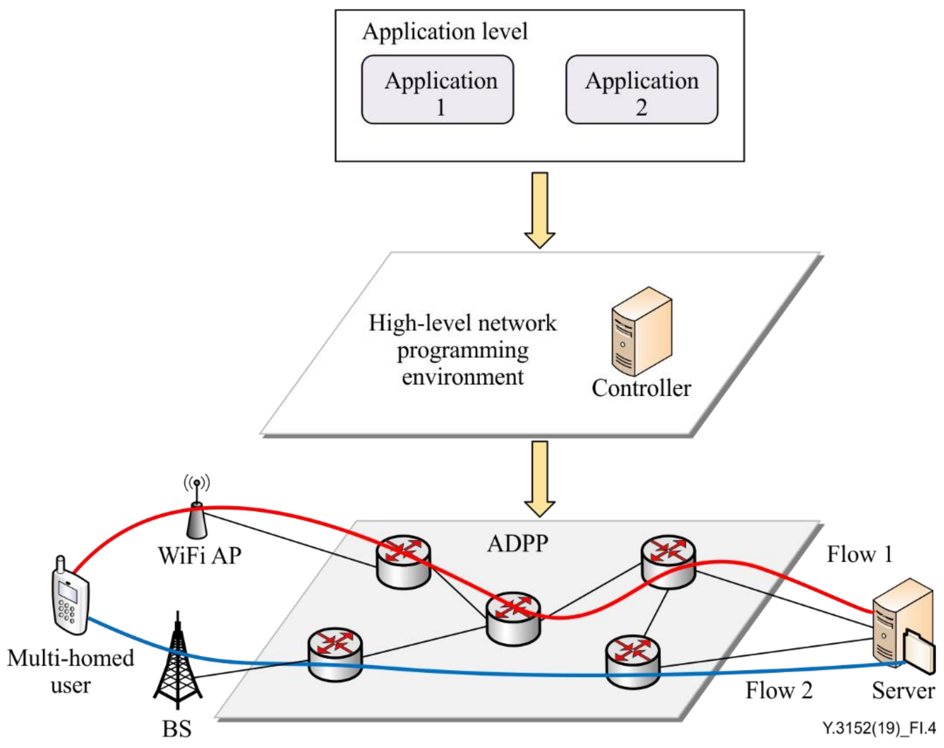

Figure I.4 – Use case of transmission in multi-homing network with ADPP

Bibliography

[b-ITU-T Y.2011]	Recommendation ITU-T Y.2011 (2004), *General principles and general reference model for Next Generation Networks*.
[b-ITU-T Y.3011]	Recommendation ITU-T Y.3011 (2012), *Framework of network virtualization for future networks*.
[b-ITU-T Y.3100]	Recommendation ITU-T Y.3100 (2018), *Terms and definitions IMT-2020 network.*
[b-ITU-T Y.3300]	Recommendation ITU-T Y.3300 (2014), *Framework of software-defined networking.*

SERIES OF ITU-T RECOMMENDATIONS

Printed in Switzerland
Geneva, 2019