(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 February 2016 (25.02.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/026516 A1

(51) International Patent Classification:
*G06F 9/44* (2006.01) *H04L 12/24* (2006.01)

(21) International Application Number:
PCT/EP2014/067641

(22) International Filing Date:
19 August 2014 (19.08.2014)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **HUAWEI TECHNOLOGIES CO., LTD.** [CN/CN]; Huawei Administration Building, Bantian Longgang, Shenzhen, Guangdong 518129 (CN).

(72) Inventor; and
(71) Applicant *(for US only)*: **PORAT, Hayim** [IL/DE]; Huawei Technologies Duesseldorf GmbH, Riesstr. 25, 80992 Munich (DE).

(74) Agent: **KREUZ, Georg M.**; Huawei Technologies Duesseldorf GmbH, Messerschmittstr. 4, 80992 Munich (DE).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

*[Continued on next page]*

(54) Title: SOFTWARE DEFINED NETWORK CONTROLLER AND METHOD FOR ITS CREATION

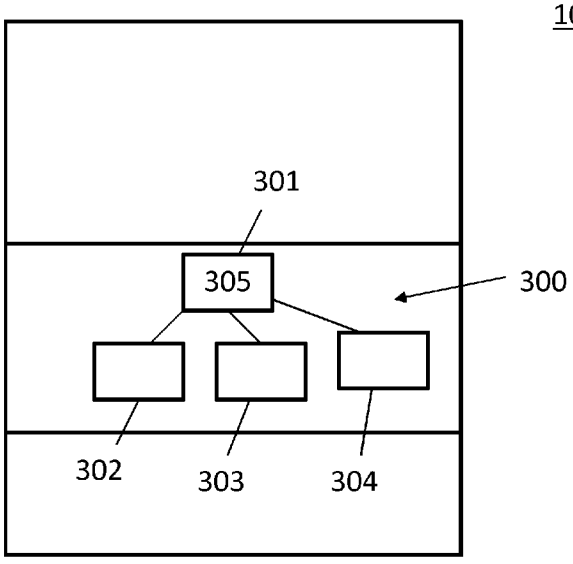


Fig. 3

(57) Abstract: The present invention provides an SDN controller 100 and a method for its creation. The SDN controller 100 is configured to fully replace a Neutron server in an Openstack networking service. The SDN controller 100 is provided with a unified data model 300 for all network devices in the network. In the unified data model 300, all network devices are abstracted, including, for instance, FEs, FWs and/or LBs. To this end, the unified data model 300 includes at least a root meta-class 301 defining a plurality of elementary attributes and/or operations 305, which are shared by all of the network devices. The unified data model 300 further includes a plurality of device classes 302, 303, 304, each device class representing one class of the network devices. The root meta-class 301 is inherited by each of the device classes 302, 303, 304.

WO 2016/026516 A1

**Published**:

— *with international search report (Art. 21(3))*

# SOFTWARE DEFINED NETWORK CONTROLLER AND METHOD FOR ITS CREATION

## TECHNICAL FIELD

The present invention relates to a software defined network (SDN) controller and a method for creating such an SDN controller. In particular, the present invention presents an SDN controller based Neutron server, i.e. a SDN controller configured to replace a Neutron server of an Openstack networking service.

## BACKGROUND

Openstack is a free and open source software cloud computing platform. The Openstack networking service is a standard service, which often involves deploying several processes across a plurality of network nodes. A main process of the Openstack networking service is a so called Neutron server, which is a Python daemon that exposes Openstack network application programming interfaces (API) to tenants of the networking service, and passes tenant requests to a suite of plug-ins for additional processing.

The Openstack networking service typically encompasses the components shown in Fig. 8. A Neutron server (formerly known as Quantum) runs on a controller node, in order to service the networking APIs and their extensions. The Neutron server is also used to enforce the network model and the IP addressing of each port. A Neutron server typically comprises at least one (Neutron) plug-in. The Neutron server requires access to a database for persistent storage, and requires access to a message queue for inter-communication.

A plug-in agent (Neutron agent) typically runs on a compute node, in order to manage and configure local virtual switches. The plug-in agent also requires message queue access.

A set of plug-ins / agents are applications that run on top of the neutron infrastructure. Among them are the dynamic host configuration protocol (DHCP) agent, which provides services to tenant networks, and also requires message queue access. Or the L3 agent that provides L3/NAT forwarding for external network access of virtual machines on tenant networks, and also requires message queue access. These agents run on the network node.

The Neutron server of the state of the art has several problems and drawbacks. For example, the Neutron server is only an API translation layer, which does not deploy any application logic, and does also not provide any other services, for example, to SDN applications. The only way to add applications or services is through the plug-in infrastructure. This, however, causes the problem that since there are no services provided by the Neutron server, the applications need to encapsulate all of the services. That is, the encapsulation needs to be done separately by each and every application.

Moreover, the state of the art Neutron server does also not provide an infrastructure for application chaining, so there is no coordination between applications that may interfere or interact with each other. In addition, applications may be partitioned only if they are operating in a "share nothing" mode using a Neutron scheduler service. In any case, applications do not control the data path as in SDN, and thus require that all processed traffic runs through network nodes, which may hinder applications like distributed virtual router (DVR).

Another disadvantage is that any addition or change of an application requires a complete compiling of a new version of the Neutron server. The Neutron server does further not support any large scale deployment, which requires clustering and high availability (HA) of the Neutron server.

In the state of the art some approaches were presented for addressing the above-mentioned problems. Two of these approaches are shown in the Figs. 9a and 9b, respectively. According to the approach shown in Fig. 9a, an external SDN controller 901 is connected through an Open Flow (OF) plug-in. Although this approach is quick and easy, and enables SDN integration, it has several drawbacks. Namely, redundant functions of virtual networking management are present in the Neutron server and in the SDN controller 901. Furthermore, both the Neutron server and the SDN controller901 need to control the switches, because both have overlapping functions with distinct

applications. Finally, there is a risk of inconsistency in application deployment, because both the SDN controller 901 and the Neutron server may support SDN applications.

According to the approach shown in Fig. 9b, an external SDN controller 902 is connected through the plug-in backend to the Neutron server, and has message queue access. However, this approach suffers the same drawbacks as the approach shown in Fig. 9a.

## SUMMARY

In view of the above-mentioned disadvantages and problems, the present invention aims to improve the state of the art. In particular, the object of the present invention is to provide another approach that allows full SDN integration into an Openstack networking service in a quick and easy manner without the above-mentioned drawbacks. By the full integration of SDN into the Openstack networking service, control of the network from a management plane through a control plane to a data plane should be enabled. Furthermore, SDN controller support for controlling and managing switching devices and other forwarding elements (FEs), but also other network elements such as load balancers (LBs) and firewalls (FWs) is desired.

The above-mentioned object of the present invention is achieved by the solution provided in the enclosed independent claims. Advantageous implementations of the present invention are further defined in the respective dependent claims. In particular the core idea of the present invention is to provide a full-featured SDN controller for replacing a Neutron server in an Openstack network service. In particular, the SDN controller is provided with a unified data model, which allows control of all network devices including LBs and FWs.

A first aspect of the present invention provides a software defined network, SDN, controller, comprising a unified data model for all network devices in a network, wherein the unified data model includes a root meta-class defining a plurality of elementary attributes and/or operations shared by all network devices, and a plurality of device classes each representing one class of the network devices, and wherein the root meta-class is inherited by each of the device classes.

The SDN controller is configured to enhance a Neutron server in an Openstack networking service, thereby providing the Openstack networking service with full SDN capability while retaining all of the Neutron server capabilities. By means of the unified data model, which provides a single root representation for all network devices in the network, the SDN controller is configured to control all network devices, including LBs, FWs etc., from a management plane. In particular, by the use of polymorphism, the SDN controller can reference actions and attributes in an easy manner across different network device classes, i.e. to different network devices. With the SDN controller of the present invention, newly added or changed applications can be simply registered on the SDN controller, and a complete recompiling of a Neutron server is unnecessary.

In a first implementation form of the SDN controller according to the first aspect, the SDN controller is configured as a network controller of a cloud management system, preferably of an Openstack cloud management system.

In a second implementation form of the SDN controller according to the first aspect as such or according to the first implementation form of the first aspect, the unified data model is contained in an abstraction layer of the SDN controller, and is a unified abstraction of all devices and functions of the network infrastructure.

The abstraction of all network devices allows an easy integration of new or changed applications or devices into the network service, regardless of their type or manufacturer. That is, by the abstraction of all network devices is ensured that devices of the same type, but e.g. of a different manufacturer, can be integrated easily into the networking service.

In a third implementation form of the SDN controller according to the first aspect as such or according to any implementation forms of the first aspect, the elementary attributes and/or operations shared by each network device include a primary capability, a port, a match rule for determining data traffic that is to be operated on, and an action to be taken on the determined data traffic.

In particular by means of the match rule and the action to be taken, a single interface for controlling the different network devices, which are abstracted in the unified data model, is provided.

In a fourth implementation form of the SDN controller according to the third implementation form of the first aspect, the elementary attributes and/or operations shared by each network device also include a communication protocol, and a control protocol.

In a fifth implementation form of the SDN controller according to the third or fourth implementation form of the first aspect, in each of the plurality of device classes the primary capability of the network device is attributed, and specific data model attributes and/or operations of the network device are contained.

In a sixth implementation form of the SDN controller according to the third to fifth implementation forms of the first aspect, the match operation and action operation in the root meta-class reference simultaneously all network devices across the plurality of device classes, in order to cause the network devices to execute at least their primary capability on the determined data traffic.

Due to the above implementation forms, all network devices can be controlled from the management plane, and in a simultaneous manner. That means, if the match rule or the action is changed, all network devices are accordingly instructed in parallel, because the root meta-class attributes are inherited to each device class.

In a seventh implementation form of the SDN controller according to the first aspect as such or according to any implementation forms of the first aspect, the plurality of network devices comprises at least one of a FE, a FW, and a LB.

The SDN controller of the present invention is thus in contrast to an SDN controller of the prior art, which can only control forwarding devices.

In an eighth implementation form of the SDN controller according to the first aspect as such or according to any implementation forms of the first aspect, the SDN controller comprises at least one application programming interface, API, for registering at least one SDN application.

All service plug-ins existing in conventional Openstack networking services (e.g. L3 agent, DHCP agent etc.) can be reintroduced as pure SDN applications registered and running on the SDN controller. As a consequence, the service plug-ins do not cause

bottlenecks for data traffic, as their data path portion is offloaded to the data-plane according to the SDN paradigm.

In a ninth implementation form of the SDN controller according to the first aspect as such or according to any implementation forms of the first aspect, the SDN controller further comprises at least one application programming interface, API, for registering service plug-ins.

For example, standard SDN or OF service plug-ins may be registered on the SDN controller.

In a tenth implementation form of the SDN controller according to the first aspect as such or according to any implementation forms of the first aspect, the SDN controller further comprises at least one application programming interface, API, for registering a message queue, MQ, application.

Thereby, MQ connectivity for other Openstack elements and for a legacy Neutron agent is provided (the legacy Neutron agent may support non-forwarding elements like FWs). MQ connectivity is a significant enhancement to the conventional SDN model.

In an eleventh implementation form of the SDN controller according to the first aspect as such or according to any implementation forms of the first aspect, the SDN controller further comprises a driver wrapper for adapting Openstack vendor specific device plug-ins and/or drivers to be used by SDN applications.

In a twelfth implementation form of the SDN controller according to the first aspect as such or according to any implementation forms of the first aspect, the SDN controller further comprises a core layer containing a plurality of services including at least one of a topology service, routing service, chaining service, dispatching service, clustering service, and high availability service.

Thereby, the Openstack networking service is provided with full SDN capability. The additional services greatly improve the Openstack networking service.

A second aspect of the present invention provides a method for creating a software defined network, SDN, controller, the method comprising creating, in an abstraction layer of the SDN controller, a unified data model for all network devices in a network, including, in the unified data model, a root meta-class containing a plurality of

elementary attributes and/or operations shared by all network devices, and a plurality of device classes each representing one class of the network devices, and inheriting the root meta-class to each of the device classes.

In a first implementation form of the method according to the second aspect, the SDN controller is configured as a network controller of a cloud management system, preferably of an Openstack cloud management system.

In a second implementation form of the method according to the second aspect as such or according to the first implementation form of the second aspect, the unified data model is created in the abstraction layer of the SDN controller, and is a unified abstraction of all devices and functions of the network infrastructure.

In a third implementation form of the method according to the second aspect as such or according to any implementation forms of the second aspect, the elementary attributes and/or operations shared by each network device include a primary capability, a port, a match rule for determining data traffic that is to be operated on, and an action to be taken on the determined data traffic.

In a fourth implementation form of the method according to the third implementation form of the second aspect, the elementary attributes and/or operations shared by each network device also include a communication protocol, and a control protocol.

In a fifth implementation form of the method according to the third or fourth implementation form of the second aspect, in each of the plurality of device classes the primary capability of the network device is attributed, and specific data model attributes and/or operations of the network device are included.

In a sixth implementation form of the method according to the third to fifth implementation forms of the second aspect, the match operation and action operation in the root meta-class reference simultaneously all network devices across the plurality of device classes, in order to cause the network devices to execute at least their primary capability on the determined data traffic.

In a seventh implementation form of the method according to the second aspect as such or according to any implementation forms of the second aspect, the plurality of network

devices comprises at least one of a forwarding element, FE, a firewall, FW, and a load balancer, LB.

In an eighth implementation form of the method according to the second aspect as such or according to any implementation forms of the second aspect, at least one application programming interface, API, for registering at least one SDN application is created in the SDN controlled.

In a ninth implementation form of the method according to the second aspect as such or according to any implementation forms of the second aspect, at least one application programming interface, API, for registering service plug-ins is created in the SDN controller.

In a tenth implementation form of the method according to the second aspect as such or according to any implementation forms of the second aspect, at least one application programming interface, API, for registering a message queue, MQ, application is created in the SDN controller.

In an eleventh implementation form of the method according to the second aspect as such or according to any implementation forms of the second aspect, a driver wrapper for adapting Openstack vendor specific device plug-ins and/or drivers to be used by SDN applications is created in the SDN controller.

In a twelfth implementation form of the method according to the second aspect as such or according to any implementation forms of the second aspect, a core layer containing a plurality of services including at least one of a topology service, routing service, chaining service, dispatching service, clustering service, and high availability service is created in the SDN controller.

The above-described second aspect and its implementation forms achieve the same advantages as described in relation to the first aspect and its implementation forms, respectively.

A third aspect of the present invention provides a computer program comprising a program code for performing, when running on a computer, a method according to the second aspect as such or according to any implementation forms of the second aspect.

The third aspect achieves to the same advantages as described in relation to the second aspect and its implementation forms, respectively.

It has to be noted that all devices, elements, units and means described in the present application could be implemented in the software or hardware elements or any kind of combination thereof. All steps which are performed by the various entities described in the present application as well as the functionalities described to be performed by the various entities are intended to mean that the respective entity is adapted to or configured to perform the respective steps and functionalities. Even if, in the following description of specific embodiments, a specific functionality or step to be full formed by eternal entities not reflected in the description of a specific detailed element of that entity which performs that specific step or functionality, it should be clear for a skilled person that these methods and functionalities can be implemented in respective software or hardware elements, or any kind of combination thereof.

## BRIEF DESCRIPTION OF DRAWINGS

The above-described aspects and implementation forms of the present invention will be explained in the following description of specific embodiments in relation to the enclosed drawings, in which

Fig. 1 shows an SDN controller according to an embodiment of the present invention in an Openstack networking service.

Fig. 2 shows an SDN controller according to an embodiment of the present invention interacting with a cloud controller node, a message queue and a Neutron plug-ins agent in an Openstack cloud management system.

Fig. 3 shows an SDN controller according to an embodiment of the present invention.

Fig. 4 shows schematically a unified data model as used in an SDN controller according to an embodiment of the present invention.

Fig. 5 shows a specific unified data model as used in an SDN controller according to an embodiment of the present invention.

Fig. 6 shows a specific configuration of an SDN controller according to an embodiment of the present invention.

Fig. 7 shows a method for creating an SDN controller according to an embodiment of the present invention.

Fig. 8 shows an Openstack networking services according to the prior art.

Fig. 9a/9b show an Openstack networking services according to the prior art.

## DETAILED DESCRIPION OF EMBODIMENTS

Fig.1 shows an SDN controller 100 according to an embodiment of the present invention within a network, specifically within an Openstack networking service. The SDN controller 100 is configured to replace a Neutron server of such an Openstack networking service. Alternatively, a conventional Neutron server may be enhanced with functions that enable it to function as an SDN controller 100. Fig. 1 shows in particular some elements of an Openstack networking service, including an Openstack object store, Openstack image service, Openstack compute, Openstack block storage and Openstack identity service. The Openstack networking (i.e. the Neutron server of a conventional Openstack networking service) is replaced one-by-one with the SDN controller 100. It can be seen from fig. 1 that no reconfiguration of the remaining conventional Openstack networking service is necessary with the new SDN controller 100 based Neutron server.

Fig. 2 shows how the SDN controller 100 according to an embodiment of the present invention interacts with a cloud controller node 201, a message queue 200 and a network node 202 (running e.g. a Neutron plug-in agent) of an Openstack cloud management system. Message queue connectivity (e.g. for Openstack elements) is enabled via the connection of the SDN controller to the message queue 200. Via the message queue 200, the SDN controller 100 can connect to the cloud controller node 201. Furthermore, the SDN controller 100 may connect to the Neutron plug-in agent on the network node 202. In comparison with the prior art solution shown in Fig. 9b, it can be seen that all service plug-ins of a conventional Openstack networking service (i.e. L3 agent, DHCP etc.) are reintroduced as SDN applications on the SDN controller 100, and do no longer act as a

bottleneck for data traffic. Specifics of the SDN controller 100 and SDN applications are explained in more detail below.

Fig. 3 shows a basic embodiment of an SDN controller 100 of the present invention as shown in the figs. 1 and 2. The SDN controller 100 is configured to fully replace a Neutron server in an Openstack networking service. The SDN controller 100 is provided at least with a unified data model 300 for all network devices in the network. In the unified data model 300, all network devices are abstracted, including, for instance, FEs, FWs and/or LBs. To this end, the unified data model 300 includes at least a root meta-class 301 defining a plurality of elementary attributes and/or operations 305, which are shared by all of the network devices. The unified data model 300 further includes a plurality of device classes 302, 303, 304, each device class representing one class of the network devices. The root meta-class 301 is inherited by each of the device classes 302, 303, 304.

Fig. 4 shows schematically a unified data model 300 as preferably included in the SDN controller 100 of fig. 3. The unified data model 300 is preferably cloud enabled and suitable for Neutron of Openstack. In particular, the unified data model 300 abstracts shared attributes of all network elements (network devices) including FEs, FWs, and LBs in the root meta-class 301. Specific attributes of the network elements are abstracted in the device classes 302, 303, 304. For instance, a FE is abstracted in a first device class 302 with flows 401 and tables 402. Further, a FW is for instance abstracted in a second device class 303 by a policy 403 and a rule 404. The unified data model 300 is preferably pluggable, i.e. may be easily extended to further network devices (i.e. without recompiling the SDN controller based Neutron server), and may model all types of network devices. In this way, a single control of all network devices by the SDN controller 100 is enabled, i.e. a control under the same infrastructure.

Fig. 5 shows a unified data model 300, which is a more specific illustration of the unified data model 300 shown in Fig. 4, and may be used in the SDN controller 100. In particular, the unified data model 300 includes again the root meta-class 301, which is valid for all network elements/devices, and defines a plurality of elementary attributes and an/or operations 305, which are shared by all network devices. The root meta-class 301 is also called network element. The root meta-class 301 may include as the attributes and/or operations 305, for instance, a primary capability, i.e. the main capability of a

network device, a port for getting network data, a match rule for determining data traffic at least to be operated on, and an action to be taken on the determined data traffic. The root meta-class 301 may preferably further include a communication protocol for understanding the network data, and a control protocol.

For each type of network device, the root meta-class 301 is inherited by a device class 302, 303, 304, respectively, wherein each device class represents a specific network device type. In each device class 302, 303, 304, the primary capability of the respective network device type is attributed, and specific data model attributes and/or operations 502, 503, 504 of the respective network device are contained. For example, in Fig. 5 the third device class 304 represents a LB. The primary capability of the LB is a stateful L7 load balancer. The specific attributes 504 of the LB include a link state and a balancing algorithm. The second device class 303 represents a FW. The primary capability of the FW is a stateful firewall. The specific attributes 503 of the FW include a rule, a policy and a log. The first device class 302 represents a FE. The primary capability of the FE is programmable frame forwarding. The specific attributes 502 of the FE include a flow and a table.

Thus, the unified data model 300 in the SDN controller 100 is a single root representation for all network devices, and is an abstraction of all devices and functions of the network infrastructure.

The unified data model 300 allows the SDN controller 100 the use of polymorphism, which enables referencing actions and attributes across different device classes with abstract operations. For example, the match operation and action operation abstracted in the root meta-class 301 may reference simultaneously network devices across the plurality of device classes 302, 303, 304, and may thus cause each of the respective network devices to execute at least its primary capability on the determined data traffic.

For example, in a cloud environment, two of the most fundamental actions are to define connectivity and security access rules between different virtual machines. On the abstract level, these two operations overlap. Therefore, as a typical policy, only actions are allowed from virtual machines that have connectivity between another. In conventional cloud environments, the two actions are modelled differently and controlled at different places. Synchronizing and coordinating the policy is therefore typically left to the user.

In contrast, the solution of the present invention allows modelling and controlling the two actions in the SDN controller. As mentioned above, in the unified data model of the present invention, a match and an action are abstracted at the root meta-class level. The match and action can thereby define the above-mentioned policy. For example, it can be defined as policy that only on data traffic originating from a certain IP address an action is taken. Then, each relevant network element (FE, FW, etc.) will implement its associated capabilities on the data traffic. The polymorphism that may thus be used reduces dramatically the need to integrate new functionalities into the SDN controller, enables the SDN controller to control all network elements using the unified data model.

Fig. 6 shows an SDN controller 100 according to an embodiment of the present invention. The SDN controller 100 of fig. 6 may be the SDN controller 100 of Fig. 3 shown in more detail. The SDN controller 100 has an abstraction layer 601, which unifies and abstracts all aspects of the network infrastructure, including functions such as FWs and LBs. The abstraction layer 601 includes preferably the unified data model 300 of all network devices.

Existing service plug-ins 606 (e.g. L3 agent, DHCP agent, FWaaS agent etc.) are preferably running as SDN applications on the SDN controller 100 shown in Fig. 6, and are registered to the SDN controller 100 via a north-bound interface (NBI) 603. The SDN controller 100 is preferably also provided with at least one legacy API 604 at the NBI 603, which is configured to function, for example, as the at least one (Neutron) plug-in of a conventional Neutron server, and allow the connection to legacy services.

The SDN controller 100 preferably also has south-bound interface (SBI) 607, which is preferably configured with multiple further plug-ins. The SBI includes preferably at least one API for registering service plug-ins, preferably SDN service plug-ins. Furthermore, the SBI contains an API for registering a MQ queue application (as shown in Fig.2). Furthermore, a driver wrapper 609 for adapting Openstack vendor specific device plug-ins and/or drivers, which may for instance be used by SDN applications, is provided on the SBI. Finally, also at the SBI a legacy API 608 is provided for allowing legacy plug-ins of conventional Neutron servers to plug-in to the SDN controller of the present invention.

The SDN controller 100 preferably also includes an internal database 602, in which the SDN controller 100 can store, for instance, information regarding SDN applications and

services running on it. The database 602 may also store application chains, in order to provide service chaining.

The SDN controller 100 further comprises preferably a core layer 605 containing a plurality of services including at least one of a topology service, routing service, chaining service, dispatching service, clustering service and high availability service.

Fig. 7 shows a method 700 for creating the SDN controller 100 of the present invention. In a first step 701, a unified data model 300 for all network devices is created, preferably in the abstraction layer 601 of the SDN controller 100. In a second step 702 a root meta-class 301 containing a plurality of elementary attributes and/or operations 305 shared by all network devices is included in the unified data model 300. In a third step 703, a plurality of device classes 302, 303, 304 each representing one class of the network devices is included in the unified data model 300. In a fourth step 704, the root meta-class 301 is inherited by each of the device classes 302, 303, 304.

The present invention may be applied to specific use cases. For example, the present invention may be applied to a DVR. In current Openstack networking service solutions, all data traffic, that crosses subnets, must pass through the network node. This makes the network node an unnecessary bottleneck, because the information it holds could be much more efficiently distributed. However, a conventional solution distributing the routing function to the compute nodes would require a compute node to exchange full routing information or store global routing information. This would put an unnecessary burden on the compute nodes in conventional solutions.

By using an SDN controller 100 according to an embodiment of the present invention as a full placement of a Neutron server, the data path may be distributed to the compute nodes, while the control part is maintained at the SDN controller level. Only required forwarding information base (FIB) segments may be sent to the compute nodes as required. Therefore, the network node is no longer a bottleneck, since not all routed data traffic must pass through it. The compute nodes are also not necessarily burdened, since they need not maintain routing protocols or large routing databases.

The present invention may also be applied to FW-as-a-Service (FWaaS). Conventionally, a FW behaviour is tightly coupled to the network behaviour. In conventional Openstack networking service solutions, the FWaaS is a separate and

disconnected service plug-in. A conventional Neutron server has the benefit of a common management platform for both networking and FWs. However, conventional SDN has no notion of FW as a controlled element but as an application associated with forwarding elements, and the FW must either run as a centralized FW, making it a bottleneck, or may be disconnected from the FW management. With the SDN controller 100 of the present invention, this problem can be solved by merging the two models, i.e. by merging a Neutron server and an SDN controller 100.

In summary, the present invention presents a full integration of SDN into Openstack networking service by means of the presented SDN controller 100. This enables controlling the whole network from a management plane through a control plane to a data plane. Furthermore, the SDN controller supports control and management of both switching devices and other network elements such as LBs and/or FWs.

The present invention has been described in conjunction with various embodiments as examples as well as implementations. However, other variations can be understood and effected by those persons skilled in the art and practicing the claimed invention, from the studies of the drawings, this disclosure and the independent claims. In the claims as well as in the description the word "comprising" does not exclude other elements or steps and the indefinite article "a" or "an" does not exclude a plurality. A single element or other unit may fulfil the functions of several entities or items recited in the claims. The mere fact that certain measures are recited in the mutual different dependent claims does not indicate that a combination of these measures cannot be used in an advantageous implementation.

# CLAIMS

1. Software defined network, SDN, controller (100), comprising

a unified data model (300) for all network devices in a network,

wherein the unified data model (300) includes a root meta-class (301) defining a plurality of elementary attributes and/or operations (305) shared by all network devices, and a plurality of device classes (302, 303, 304) each representing one class of the network devices, and

wherein the root meta-class (301) is inherited by each of the device classes (302, 303, 304).

2. SDN controller (100) according to claim 1, which is configured as a network controller of a cloud management system, preferably of an openstack cloud management system.

3. SDN controller (100) according to claim 1 or 2, wherein the unified data model (300) is contained in an abstraction layer (601) of the SDN controller (100), and is a unified abstraction of all devices and functions of the network infrastructure.

4. SDN controller (100) according to one of the claims 1 to 3, wherein the elementary attributes and/or operations (305) shared by each network device include

a primary capability,

a port,

a match rule for determining data traffic that is to be operated on, and

an action to be taken on the determined data traffic.

5. SDN controller (100) according to claim 4, wherein the elementary attributes and/or operations (305) shared by each network device also include

a communication protocol, and

a control protocol.

6. SDN controller (100) according to claim 4 or 5, wherein in each of the plurality of device classes (302, 303, 304)

the primary capability of the network device is attributed, and

specific data model attributes and/or operations (502, 503, 504) of the network device are contained.

7. SDN controller (100) according to one of the claims 4 to 6, wherein

the match operation and action operation in the root meta-class (301) reference simultaneously all network devices across the plurality of device classes (302, 303, 304), in order to cause the network devices to execute at least their primary capability on the determined data traffic.

8. SDN controller (100) according to one of the claims 1 to 7, wherein

the plurality of network devices comprises at least one of a forwarding element, FE, a firewall, FW, and a load balancer, LB.

9. SDN controller (100) according to one of the claims 1 to 8, further comprising

at least one application programming interface (603), API, for registering at least one SDN application (606).

10. SDN controller (100) according to one of the claims 1 to 9, further comprising

at least one application programming interface (610), API, for registering service plug-ins.

11. SDN controller (100) according to one of the claims 1 to 10, further comprising

at least one application programming interface (611), API, for registering a message queue, MQ, application.

12. SDN controller (100) according to one of the claims 1 to 11, further comprising

a driver wrapper (609) for adapting Openstack vendor specific device plug-ins and/or drivers to be used by SDN applications.

13. SDN controller (100) according to one of the claims 1 to 12, further comprising

a core layer (605) containing a plurality of services including at least one of a topology service, routing service, chaining service, dispatching service, clustering service, and high availability service.

14. Method (700) for creating a software defined network, SDN, controller (100), the method comprising

creating (701), in an abstraction layer (601) of the SDN controller (100), a unified data model (300) for all network devices in a network,

including (702, 703), in the unified data model (300), a root meta-class (301) containing a plurality of elementary attributes and/or operations (305) shared by all network devices, and a plurality of device classes (302, 303, 304) each representing one class of the network devices, and

inheriting the root meta-class (301) to each of the device classes (302, 303, 304).

15. A computer program comprising a program code for performing, when running on a computer, the method (700) according to claim 14.

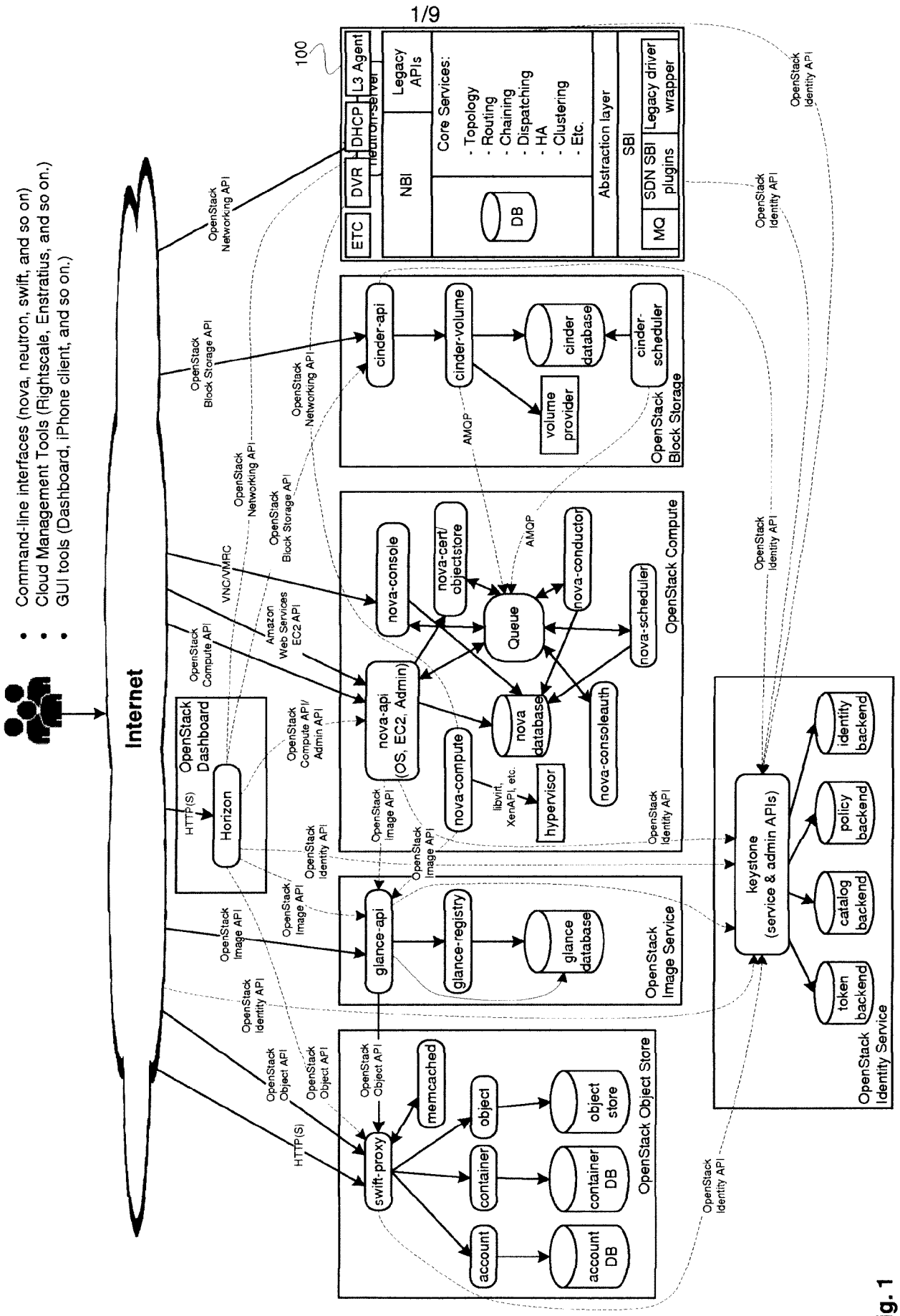
Command-line interfaces (nova, neutron, swift, and so on)
Cloud Management Tools (Rightscale, Enstratius, and so on.)
GUI tools (Dashboard, iPhone client, and so on.)
Internet
100
ETC
DVR
DHCP
L3 Agent
neutron-server
NBI
Legacy APIs
Core Services:
- Topology
- Routing
- Chaining
- Dispatching
- HA
- Clustering
- Etc.
DB
Abstraction layer
SBI
MQ
SDN SBI plugins
Legacy driver wrapper
OpenStack Networking API
OpenStack Identity API
OpenStack Block Storage API
cinder-api
cinder-volume
cinder database
cinder-scheduler
volume provider
AMQP
OpenStack Block Storage
OpenStack Networking API
OpenStack Block Storage API
VNC/VMRC
Amazon Web Services EC2 API
OpenStack Compute API
nova-console
nova-cert/ objectstore
Queue
nova-conductor
nova-scheduler
nova-api (OS, EC2, Admin)
nova database
nova-compute
libvirt, XenAPI, etc.
hypervisor
nova-consoleauth
OpenStack Compute
OpenStack Compute API/ Admin API
OpenStack Image API
OpenStack Identity API
HTTP(S)
OpenStack Dashboard
Horizon
glance-api
glance-registry
glance database
OpenStack Image Service
keystone (service & admin APIs)
identity backend
policy backend
catalog backend
token backend
OpenStack Identity Service
OpenStack Object API
memcached
swift-proxy
object
container
account
object store
container DB
account DB
OpenStack Object Store

Fig. 1

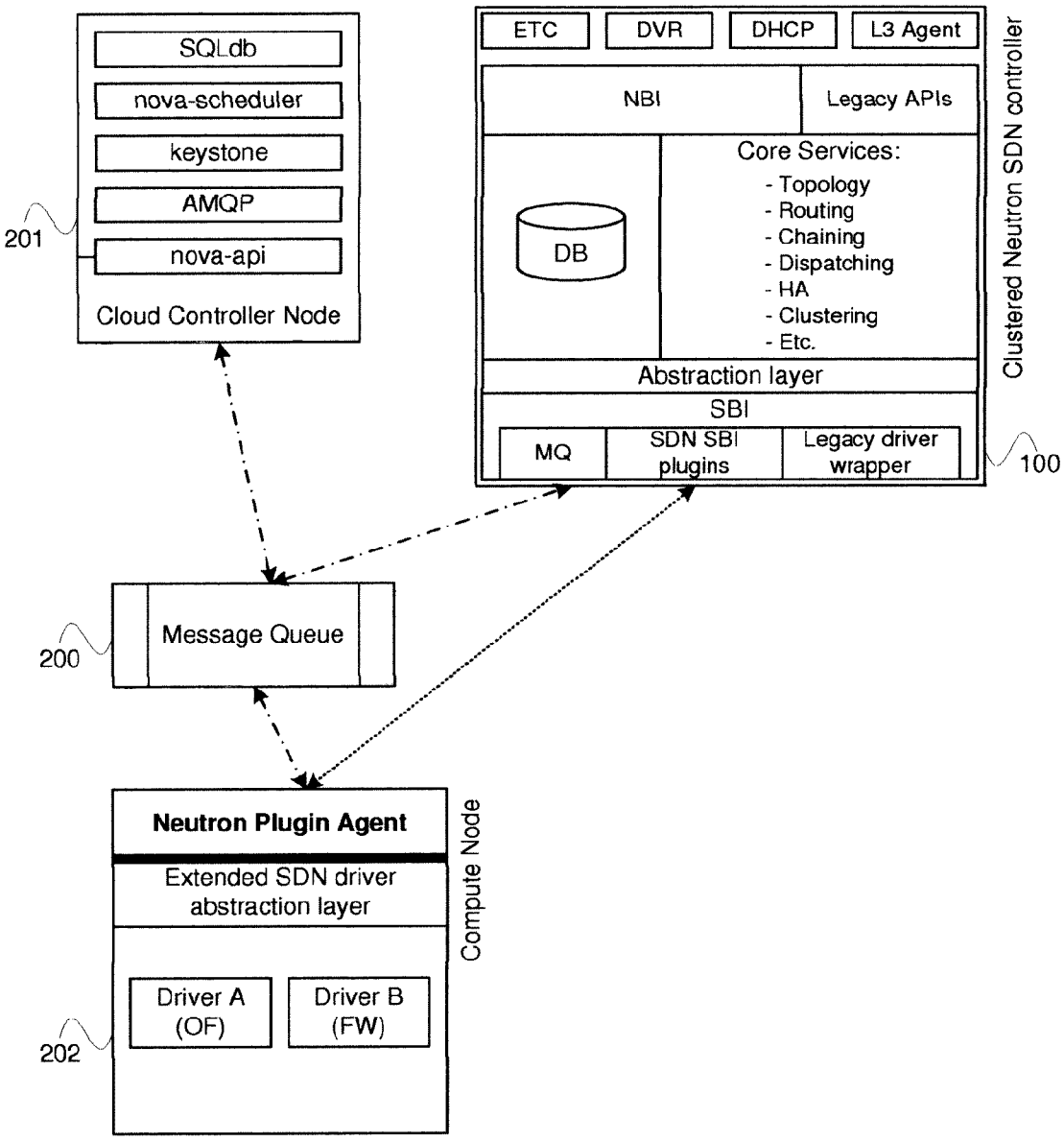
SQLdb
nova-scheduler
keystone
AMQP
nova-api
Cloud Controller Node
201
ETC
DVR
DHCP
L3 Agent
NBI
Legacy APIs
DB
Core Services:
- Topology
- Routing
- Chaining
- Dispatching
- HA
- Clustering
- Etc.
Abstraction layer
SBI
MQ
SDN SBI plugins
Legacy driver wrapper
Clustered Neutron SDN controller
100
Message Queue
200
Neutron Plugin Agent
Extended SDN driver abstraction layer
Driver A (OF)
Driver B (FW)
202
Compute Node

**Fig. 2**

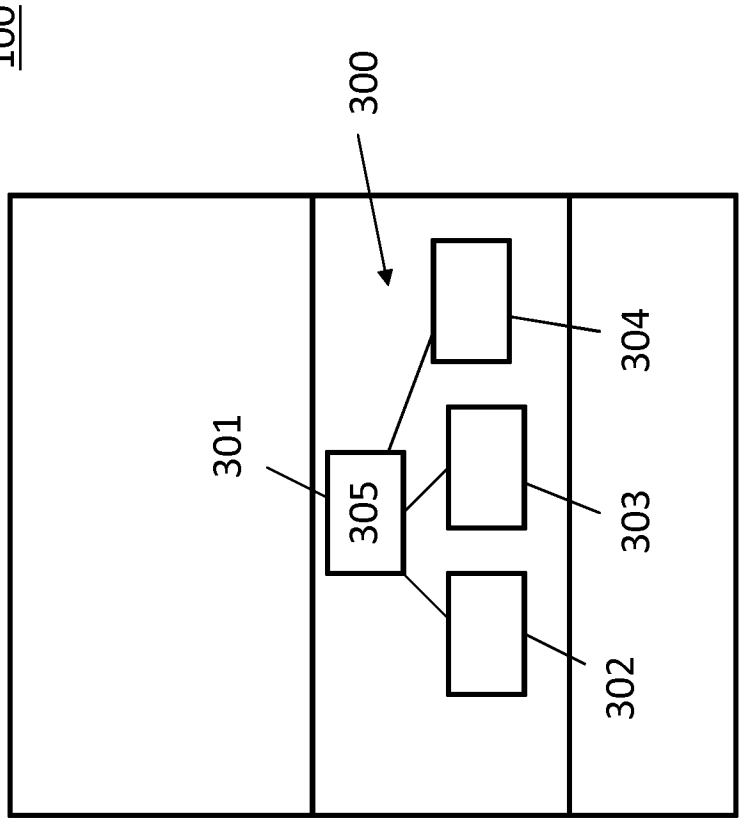


Fig. 3

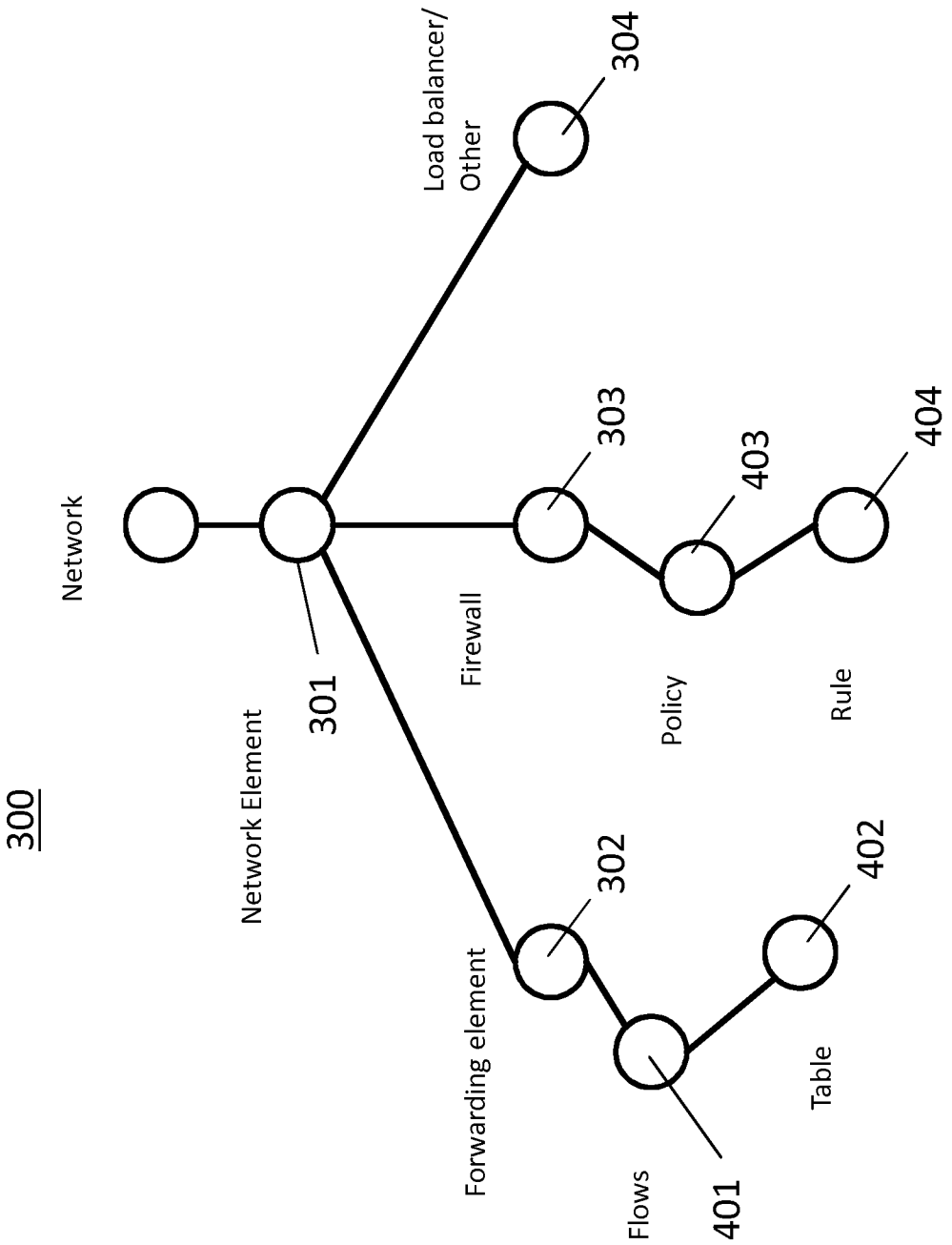
300
Network
Network Element
301
Load balancer/
Other
304
Firewall
303
Policy
403
Rule
404
Forwarding element
302
Table
402
Flows
401

Fig. 4

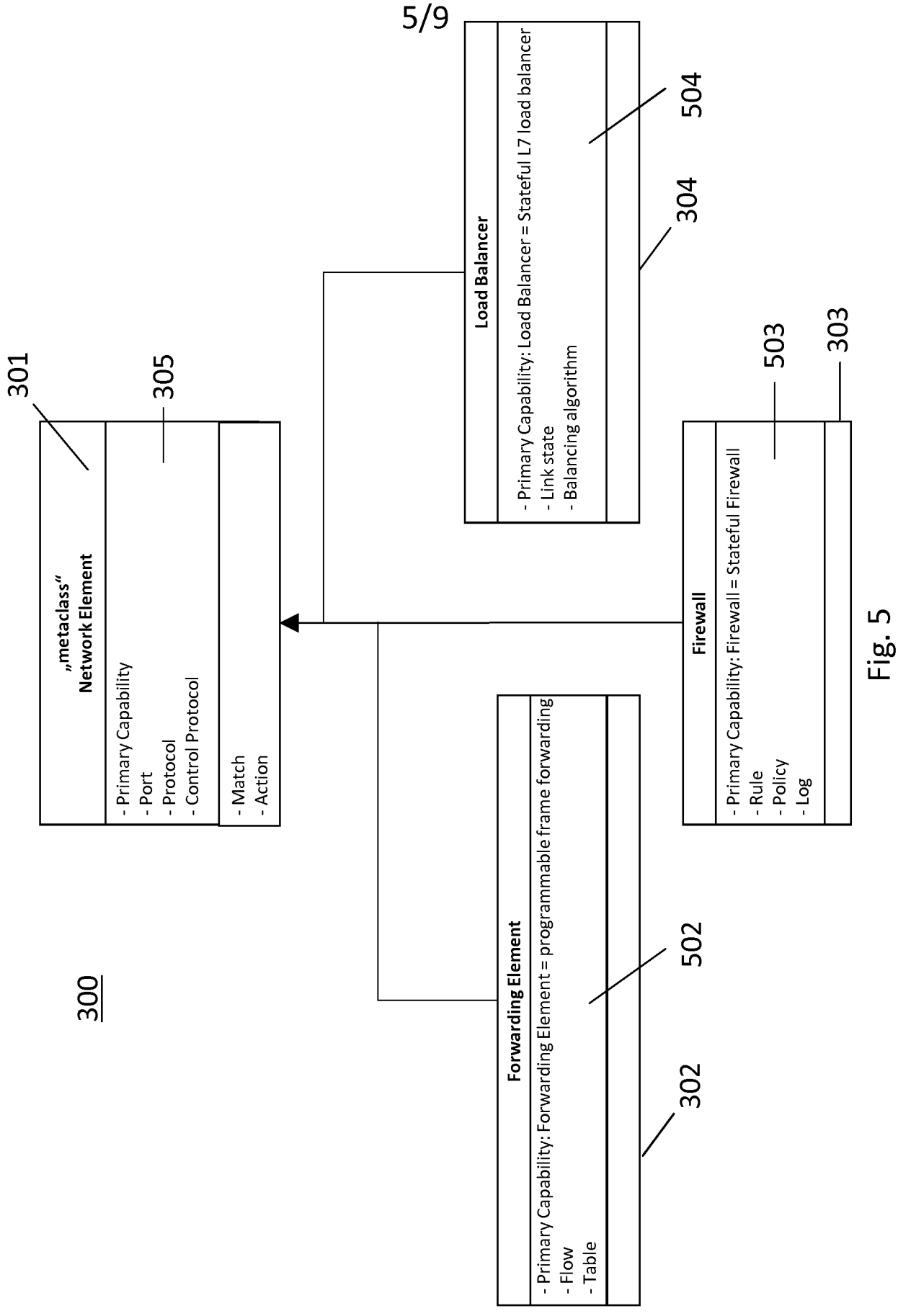
300
301
„metaclass“
Network Element
- Primary Capability
- Port
- Protocol
- Control Protocol
- Match
- Action
305
Load Balancer
- Primary Capability: Load Balancer = Stateful L7 load balancer
- Link state
- Balancing algorithm
504
304
Firewall
- Primary Capability: Firewall = Stateful Firewall
- Rule
- Policy
- Log
503
303
Forwarding Element
- Primary Capability: Forwarding Element = programmable frame forwarding
- Flow
- Table
502
302

Fig. 5

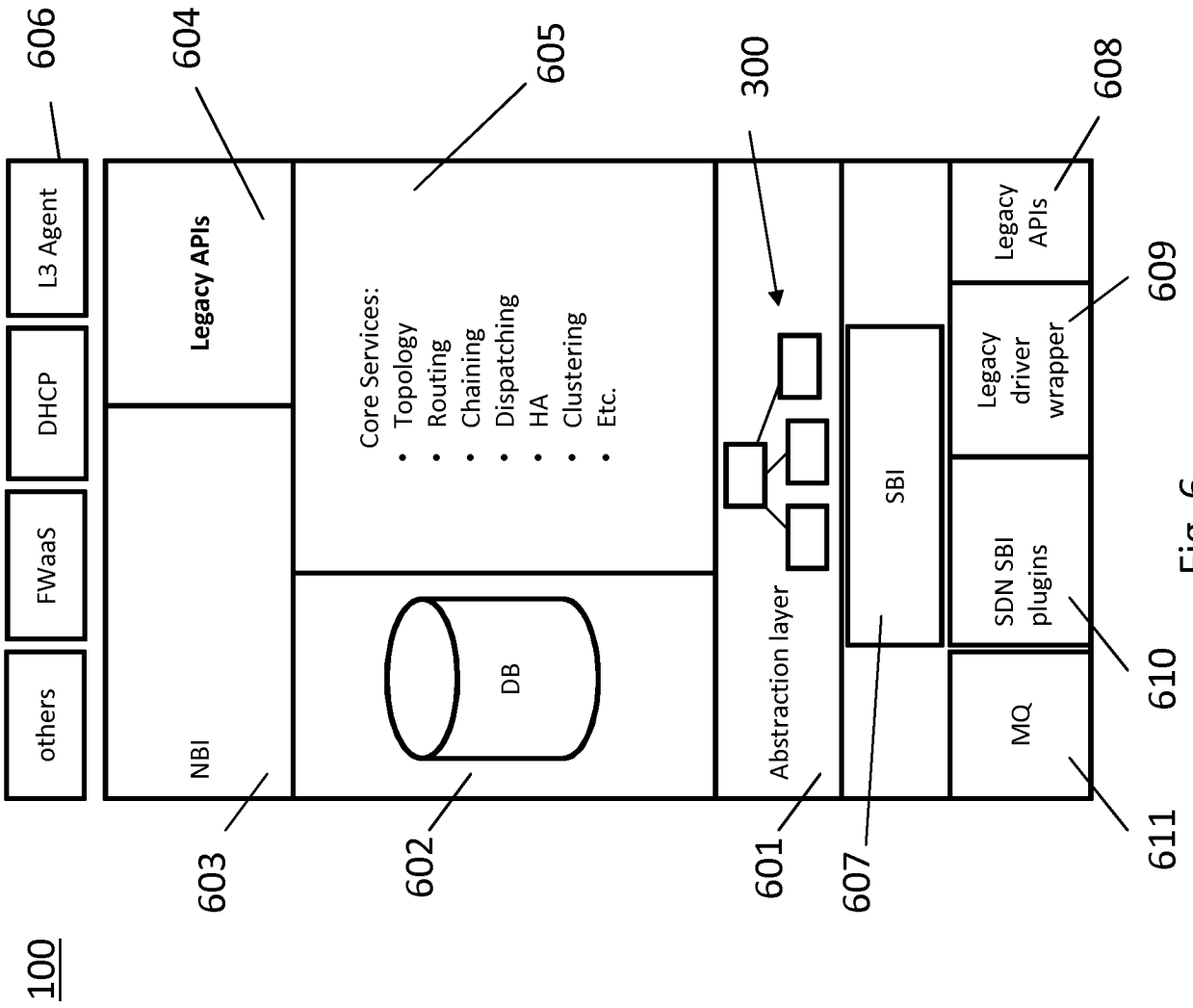
100
606
604
605
300
608
609
610
611
603
602
601
607
L3 Agent
DHCP
FWaaS
others
Legacy APIs
NBI
Core Services:
• Topology
• Routing
• Chaining
• Dispatching
• HA
• Clustering
• Etc.
DB
Abstraction layer
SBI
Legacy APIs
Legacy driver wrapper
SDN SBI plugins
MQ

Fig. 6

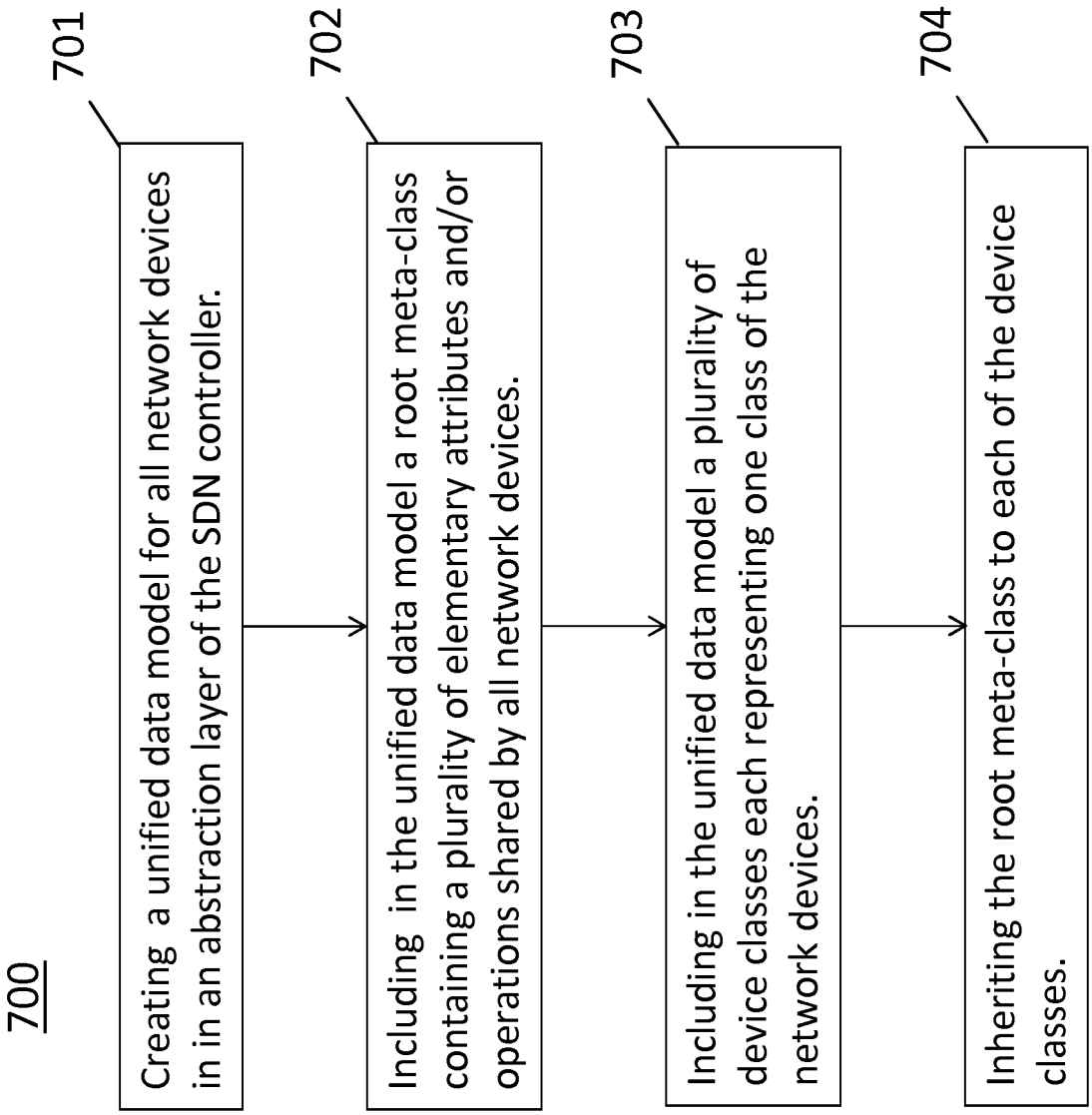
700
701
Creating a unified data model for all network devices in in an abstraction layer of the SDN controller.
702
Including in the unified data model a root meta-class containing a plurality of elementary attributes and/or operations shared by all network devices.
703
Including in the unified data model a plurality of device classes each representing one class of the network devices.
704
Inheriting the root meta-class to each of the device classes.

Fig. 7

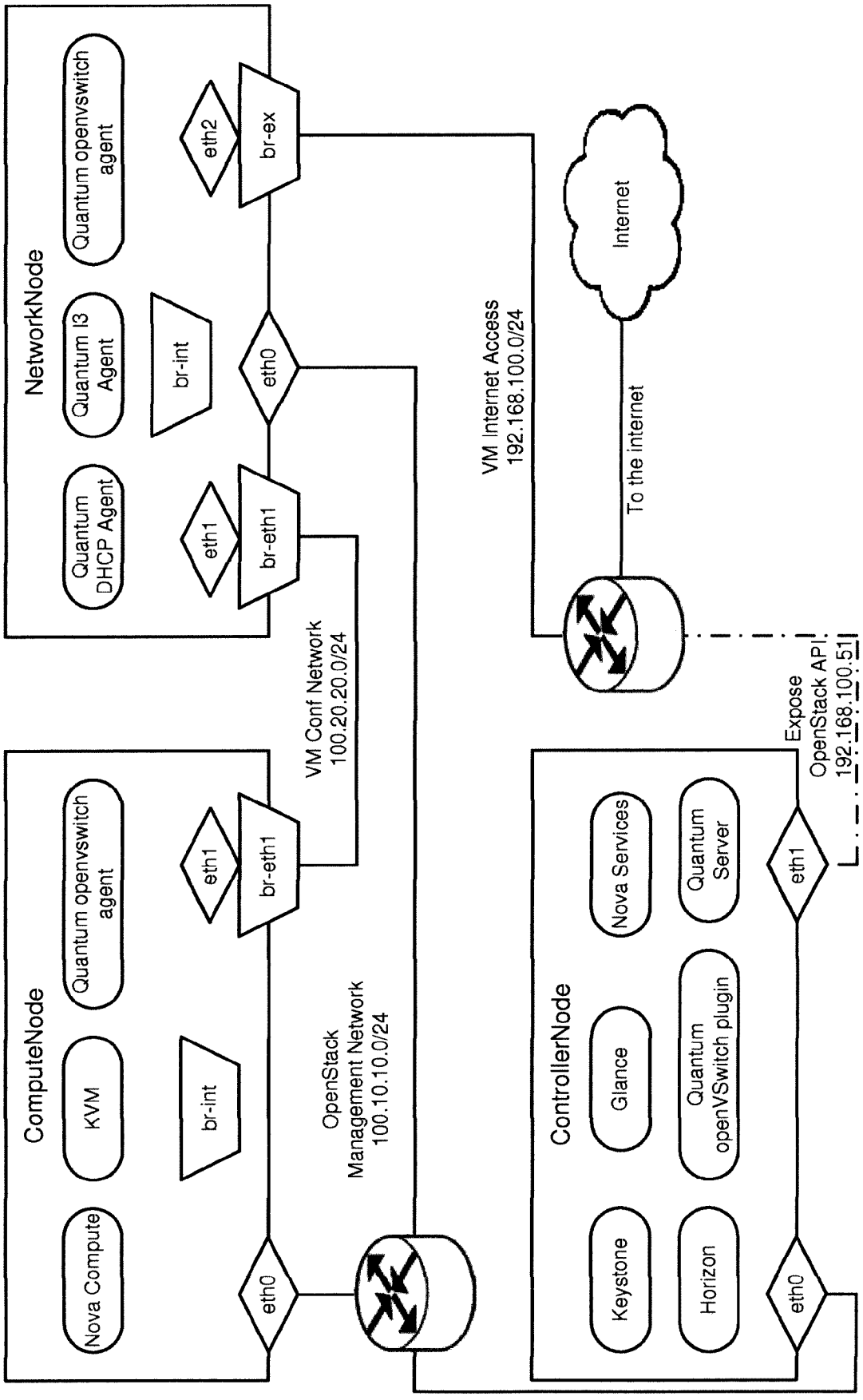
NetworkNode
Quantum openvswitch agent
Quantum l3 Agent
Quantum DHCP Agent
br-int
eth2
br-ex
eth0
eth1
br-eth1
Internet
VM Internet Access
192.168.100.0/24
To the internet
VM Conf Network
100.20.20.0/24
ComputeNode
Quantum openvswitch agent
KVM
Nova Compute
br-int
eth1
br-eth1
eth0
OpenStack
Management Network
100.10.10.0/24
ControllerNode
Nova Services
Quantum Server
Glance
Quantum openVSwitch plugin
Keystone
Horizon
eth1
eth0
Expose
OpenStack API
192.168.100.51

Fig. 8

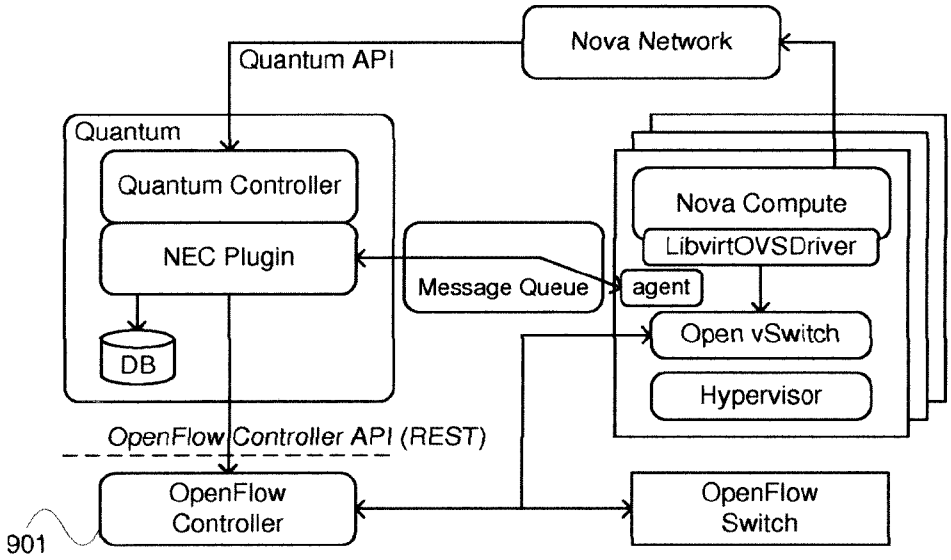
Nova Network
Quantum API
Quantum
Quantum Controller
NEC Plugin
DB
Message Queue
Nova Compute
LibvirtOVSDriver
agent
Open vSwitch
Hypervisor
OpenFlow Controller API (REST)
OpenFlow Controller
OpenFlow Switch
901

***Fig. 9a***

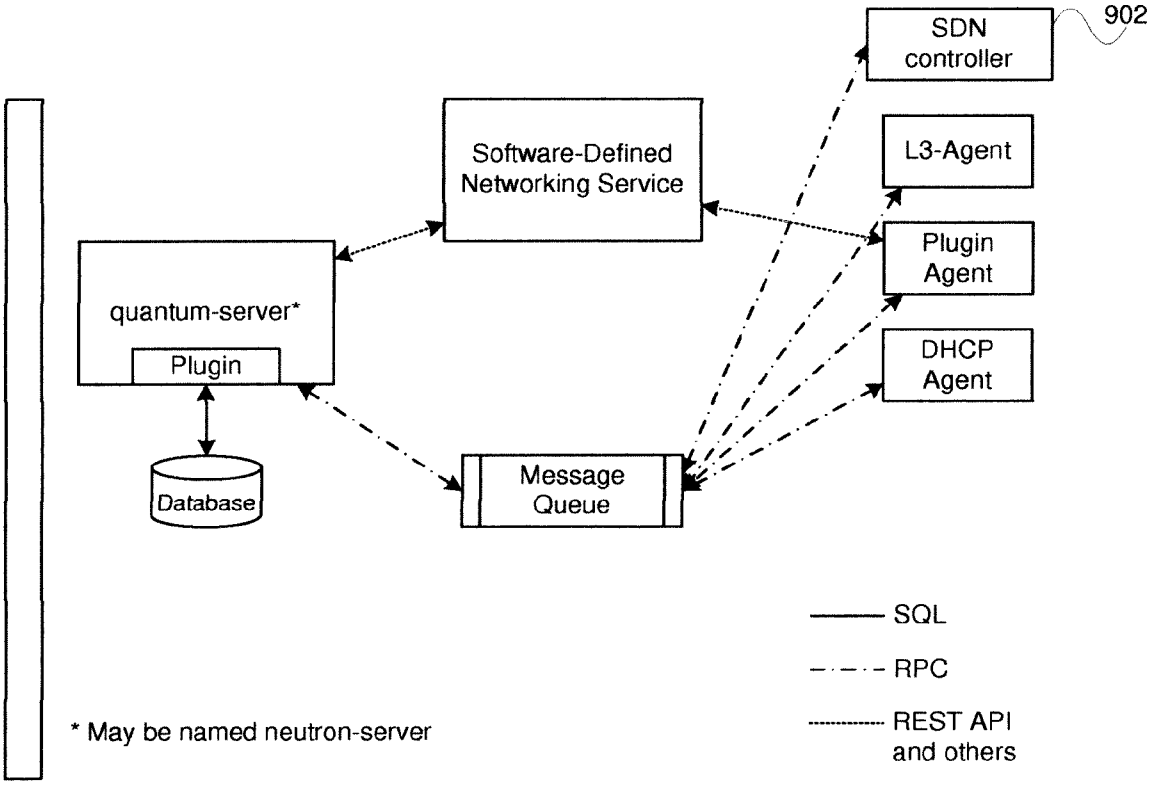
SDN controller
902
Software-Defined Networking Service
L3-Agent
Plugin Agent
DHCP Agent
quantum-server*
Plugin
Database
Message Queue
SQL
RPC
REST API and others
* May be named neutron-server

**Fig. 9b**

# INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2014/067641

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/44 H04L12/24
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal, IBM-TDB, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | US 2014/112190 A1 (CHOU WU [US] ET AL) 24 April 2014 (2014-04-24) paragraph [0022] | 1-15 |
| A | US 6 249 291 B1 (POPP NICOLAS [US] ET AL) 19 June 2001 (2001-06-19) column 18, line 39 - line 50 | 1-15 |
| A | US 2013/262685 A1 (SHELTON JAMES H [US] ET AL) 3 October 2013 (2013-10-03) paragraph [0007] - paragraph [0010] | 1-15 |
| A | US 5 493 680 A (DANFORTH SCOTT H [US]) 20 February 1996 (1996-02-20) column 14, line 25 - line 42 | 1-15 |

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 22 October 2014 | 29/10/2014 |

| Name and mailing address of the ISA/ | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040,<br>Fax: (+31-70) 340-3016 | Suciu, Radu |

1

# INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2014/067641

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| US 2014112190 A1 | 24-04-2014 | US 2014112190 A1 | 24-04-2014 |
| | | WO 2014063605 A1 | 01-05-2014 |
| US 6249291 B1 | 19-06-2001 | US 6249291 B1 | 19-06-2001 |
| | | US 2002007376 A1 | 17-01-2002 |
| | | US 2006184887 A1 | 17-08-2006 |
| | | US 2007113192 A1 | 17-05-2007 |
| | | US 2007113193 A1 | 17-05-2007 |
| | | US 2007192709 A1 | 16-08-2007 |
| US 2013262685 A1 | 03-10-2013 | CN 103460184 A | 18-12-2013 |
| | | EP 2625601 A1 | 14-08-2013 |
| | | US 2013262685 A1 | 03-10-2013 |
| | | WO 2012047756 A1 | 12-04-2012 |
| US 5493680 A | 20-02-1996 | JP H06103075 A | 15-04-1994 |
| | | US 5493680 A | 20-02-1996 |