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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 June 2015 (25.06.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/092954 A1

(51) International Patent Classification:
G06F 13/10 (2006.01) *G06F 13/00* (2006.01)
G06F 9/46 (2006.01) *H04L 12/717* (2013.01)

(21) International Application Number:
PCT/JP2014/005105

(22) International Filing Date:
7 October 2014 (07.10.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14/132,135 18 December 2013 (18.12.2013) US

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; New Orchard Road, Armonk, New York, 10504 (US).

(71) Applicant *(for MG only)*: **IBM JAPAN, LTD.** [JP/JP]; 19-21 Nihonbashi Hakozaki-cho, Chuo-ku, Tokyo, 1038510 (JP).

(72) Inventors: **DECUSATIS, Casimer M.**; 2455 South Road, Poughkeepsie, New York, 12601 (US). **KRISHNAMURTHY, Rajaram B.**; 2455 South Road, Poughkeepsie, New York, 12601 (US).

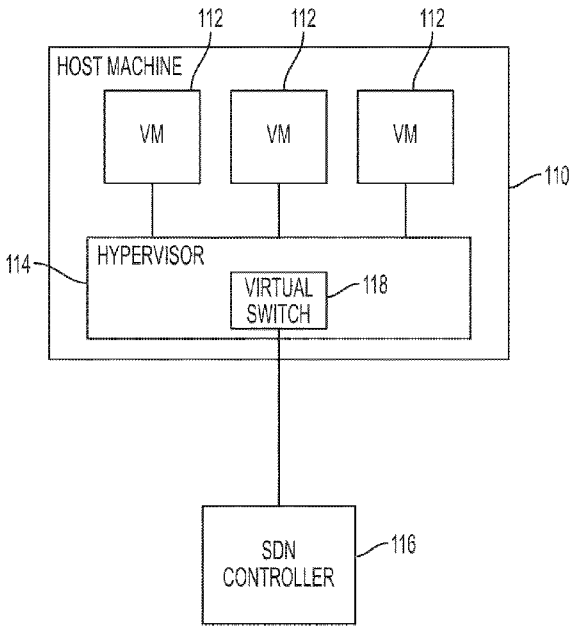
(74) Agents: **UENO, Takeshi** et al.; c/o Toyosu site, IBM Japan, Ltd., NBF TOYOSU CANAL FRONT Bldg., 6-52, Toyosu 5-chome, Koto-ku, Tokyo, 1358511 (JP).

(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, 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 NETWORKING (SDN) FOR MANAGEMENT OF TRAFFIC BETWEEN VIRTUAL PROCESSORS



(57) Abstract: An aspect includes receiving, at a software-defined networking (SDN) controller, an inquiry from a virtual switch executing on a host machine. The inquiry includes a request to identify a flow of a data packet received at the virtual switch from a source virtual processor. The source virtual processor is either a logical partition (LPAR) or a virtual machine (VM) executing under control of a hypervisor on the host machine. A destination virtual processor associated with the data packet is determined by the SDN controller. In addition, the SDN controller identifies the flow between the source virtual processor and the destination virtual processor. The flow includes a least one virtual port in the virtual switch. The SDN controller instructs the virtual switch to send the data packet from the source virtual processor to the destination virtual processor via the identified flow.

WO 2015/092954 A1

Published:

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

Description

Title of Invention: SOFTWARE-DEFINED NETWORKING (SDN) FOR MANAGEMENT OF TRAFFIC BETWEEN VIRTUAL PROCESSORS

Technical Field

[0001] The present invention relates generally to software-defined networking (SDN), and more specifically, to using SDN for management of traffic between virtual processors.

Background Art

[0002] Information technology (IT) resources, such as computer processors and networks, are being called upon to support ever greater processing demands, leading to the need for server footprints of increasing size to accommodate these expanding workloads. Virtualization provides a way to abstract the components of today's IT resources to consolidate, integrate, and simplify the required infrastructure and reduce the overall cost of IT resource ownership.

[0003] Server virtualization technology allows for the configuration and deployment of multiple logical server configurations on a common physical footprint to provide processing and usage benefits beyond those of the physical configuration. The physical server's resources are abstracted to accommodate the concurrent deployment of multiple instances of virtual processors. Each virtual instance, called a logical partition (LPAR) or virtual machine (VM), is capable of operating a separate operating system (OS) instance and its associated software stacks as if each instance was deployed on a separate physical server. This virtual view offers the benefit of not being restricted by the implementation or configuration of the underlying physical server resources. Each virtual processor instance provides a subset or superset of the various physical server resources that may be dedicated or concurrently shared by multiple LPAR or VM abstractions. By using processor virtualization technologies, the system's processors can be transparently multi-programmed and multi-processed by a virtualization hypervisor to optimize processor sharing by multiple LPAR or VM instances, thereby increasing processor utilization.

[0004] In traditional IT network architectures there is no centralized network control. Routing tables located locally in network devices, such as switches, bridges, gateways, routers, or firewalls, are individually configured to direct network traffic to neighboring nodes of the network. The network devices may make control decisions and forward network traffic accordingly. Traditional network architectures are contrasted with software-defined networking (SDN), where network traffic routing decisions are centrally controlled and made by a controller that creates tables to define

flow paths through the network. The controller decouples control decisions about where traffic is sent from network devices that forward traffic to a selected destination.

Summary of Invention

[0005] Embodiments include methods, systems, and computer program products for software-defined networking (SDN) for management of traffic between virtual processors. A method includes receiving, at a SDN controller, an inquiry from a virtual switch executing on a host machine. The inquiry includes a request to identify a flow of a data packet received at the virtual switch from a source virtual processor. The source virtual processor is either a logical partition (LPAR) or a virtual machine (VM) executing under control of a hypervisor on the host machine. A destination virtual processor associated with the data packet is determined by the SDN controller. The SDN controller identifies the flow between the source virtual processor and the destination virtual processor. The flow includes a least one virtual port in the virtual switch. The SDN controller instructs the virtual switch to send the data packet from the source virtual processor to the destination virtual processor via the identified flow.

[0006] The subject matter which is regarded as embodiments is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The forgoing and other features, and advantages of the embodiments are apparent from the following detailed description taken in conjunction with the accompanying drawings in which:

Brief Description of Drawings

[0007] [fig.1]FIG. 1 depicts a system for implementing software-defined networking (SDN) for management of traffic between virtual processor on a single host machine in accordance with an embodiment;

[fig.2]FIG. 2 depicts a system for implementing SDN for management of traffic between virtual processors on multiple host machines in accordance with an embodiment;

[fig.3]FIG. 3 depicts a block diagram of functions performed by a virtual switch in accordance with an embodiment;

[fig.4]FIG. 4 depicts a block diagram of a SDN controller in accordance with an embodiment; and

[fig.5]FIG. 5 depicts a process flow for performing management of traffic between virtual processors in accordance with an embodiment.

Detailed Description of Embodiments

[0008] Exemplary embodiments relate to centralized software control of distributed logical partitions (LPARs) and virtual machines (VMs). An embodiment includes a software-defined networking (SDN) controller, at the network layer, that is capable of managing the virtual infrastructure of a network. In particular, the SDN controller can be capable

of recognizing LPAR and VM attributes established by a host machine in the network. In an embodiment, the SDN controller associates one-to-one, one-to-many, or many-to-one combinations of virtual processors (e.g., LPARs, VMs) and switches as a unified, distributed LPAR, and can programmatically manage LPAR traffic between distributed LPARs. This can be performed regardless of whether or not an LPAR is subsequently moved to another, different host machine in the network and then back to the original host machine. In order to manage traffic originating at a VM on a particular host machine, virtual identification tags can be assigned to VMs on that host machine which allows the SDN controller to manage/control the VMs like LPARs. For example, traffic may be disallowed from other distributed LPAR partitions in order to provide full isolation similar to traditional LPARs.

[0009] As used herein the term distributed LPAR (logical partition) refers to a subset of a computer's hardware resources (processor, memory, and storage) which is divided or virtualized into multiple sets of resources so that each set of resources can be operated independently with its own operating system instance and its own set of applications. The number of logical partitions that can be created depends on the system's processor model and available physical resources. Typically, partitions are used for different purposes such as database operation or client/server operation or to separate test and production environments. Each partition can communicate with the other partitions as if the other partition is in a separate physical machine.

[0010] Examples of the types of traffic that can be transmitted between the virtual processors include, but are not limited to, client/server traffic using Ethernet or similar packet-based networking protocols.

[0011] Turning now to FIG. 1, a system for utilizing a SDN controller 116 for managing traffic between virtual processors is generally shown in accordance with an embodiment. The system includes a host machine 110 that is executing multiple VMs 112 as well as a hypervisor 114 and a virtual switch 118. In an embodiment, the SDN controller 116 is in communication with the virtual switch 118 via secure link (e.g., an outband Ethernet link secured via some form of data encryption and/or endpoint authentication and a network connector such as an adapter or network interface card (NIC). The SDN controller 116 can be dedicated to managing traffic between VMs 112 on the host machine 110 or it can be used to manage traffic for multiple host machines. Alternatively, the SDN controller 116 can be used to provide control for a SDN network in addition to managing traffic between VMs. In an embodiment, the SDN controller 116 is located in the host machine 110 and is executing within the hypervisor 114 and/or within one of the VMs 112.

[0012] The host machine 110 can be implemented by any high speed processing device that supports virtual processors such as, but not limited to: a System Z(R) server, a System

X(R) BladeCenter(R) , or a hybrid processing device that includes a System Z server and a System X BladeCenter server coupled to each other. The host machine 110 can utilize hypervisor functions such as those provided by processor resource/system manager (PR/SM). Alternatively, the hypervisor 114 can be implemented by the z/VM software hypervisor.

[0013] In an embodiment, the virtual switch 118 is implemented by software that is executed by the hypervisor 114 to simulate a physical switch, such as an ethernet switch. Only one virtual switch 118 is shown in FIG. 1, however, other configurations may include multiple virtual switches 118 that are shared among VMs 112 or dedicated to a particular VM 112.

[0014] The example shown in FIG. 1 uses VMs 112 as an example of virtual processors executing on the host machine 110. In another embodiment, the virtual processors are LPARs.

[0015] Turning now to FIG. 2, a system for utilizing a SDN controller 216 for managing traffic between virtual processors that are located on two different host machines 210 is generally shown in accordance with an embodiment. The system includes host machines 210 that are executing LPARs 212 as well as hypervisors 214 and virtual switches 218. In an embodiment, the SDN controller 216 is in communication with the virtual switches 218 via a secure link and a network connector such as an adapter or network interface card (NIC). The SDN controller 216 can be dedicated to managing traffic between LPARs 112 on the host machines 110 or it can be used to perform other functions in a SDN network. In an embodiment, the SDN controller 216 is located in one of the host machines 210 and executing within one of the hypervisors 214 and/or within one of the LPARs 212.

[0016] The host machines 210 can be implemented by any high speed processing devices that support virtual processors such as, but not limited to: a System Z server, a System X BladeCenter, or a hybrid processing device that includes a System Z server and a System X BladeCenter server coupled together. The host machines 210 can utilize a PR/SM and/or z/VM software hypervisor. In an embodiment, the host machines 210 are connected together and acting as a single host machine.

[0017] Turning now to FIG. 3, a block diagram of a virtual switch, such as virtual switch 118 or 218 that may be implemented by exemplary embodiments is generally shown. An embodiment of the virtual switch is implemented by software instructions to provide the functions shown in the blocks of FIG. 3. The virtual switch functions shown in FIG. 3 include virtual switch logic 302, a secure link interface 304, a flow table 306, and virtual ports 310a-310n. The virtual switch logic 302 may be implemented in one or more processing circuits, where a computer readable storage medium is configured to hold instructions for the virtual switch logic 302 as well as

various variables and constants to support operation of the virtual switch. The virtual switch logic 302 forwards network traffic (e.g., packets) between the virtual ports 310a-310n as flows defined by the SDN controller.

[0018] The secure link interface 304 connects the virtual switch to the SDN controller. The secure link interface 304 allows commands and packets to be communicated between the SDN controller and the virtual switch using a SDN protocol. The secure link interface 304 can be controlled by executable instructions stored and executed as part of the virtual switch.

[0019] The flow table 306 defines supported connection types associated with particular addresses, virtual local area networks or virtual ports, for example. A flow may be defined as all network traffic that matches a particular header format, including use of wildcards. Each entry 311 in the flow table 306 can include one or more rules 312, actions 314, and statistics 316 associated with a particular flow. The rules 312 define each flow and can be determined by packet headers. The actions 314 define how packets are processed. The statistics 316 track information such as the size of each flow (e.g., number of bytes), the number of packets for each flow, and time since the last matching packet of the flow or connection time. Examples of actions include instructions for forwarding packets of a flow to one or more specific virtual ports 310a-310n (e.g., unicast or multicast), encapsulating and forwarding packets of a flow to the SDN controller, and dropping packets of the flow. Entries 311 in the flow table 306 can be added and removed by the SDN controller via the secure link interface 304. The SDN controller can pre-populate the entries 311 in the flow table 306. Additionally, the virtual switch can request creation of an entry 311 from the SDN controller upon receiving a flow without a corresponding entry 311 in the flow table 306.

[0020] A virtual machine has associated with it a number of virtual I/O interfaces, which transmit data packets to a virtual switch in the hypervisor in a manner similar to that used by a physical server connected to a physical switch (but without the need for cables external to the physical server). Each data packet is to be encapsulated with a virtual identification tag by the virtual I/O interface; this tag identifies the packet as coming from a particular virtual machine, and being destined for a particular virtual switch interface. The tags may contain other optional information such as qualify of service tags. The virtual switch is aware of these tags. The virtual switch is managed by the SDN controller, in a similar manner that a physical switch would be managed by an SDN controller, including functions such as providing flow tables to the switch, and implementing specific flows based on match-action tables defined in the SDN controller

[0021] Turning now to FIG. 4, a block diagram of a SDN controller such as SDN controller

116 or 216 is generally shown in accordance with an embodiment. The SDN controller can be embodied in any type of processing system and is depicted embodied in a general-purpose computer 401 in FIG. 4. Alternatively, the SDN controller can be implemented as software instructions and stored as part of a hypervisor or located in a virtual processor.

[0022] In an exemplary embodiment, in terms of hardware architecture, as shown in FIG. 4, the computer 401 includes processing circuitry 405 and memory 410 coupled to a memory controller 415, and an input/output (I/O) controller 435. The I/O controller 435 can be, for example but not limited to, one or more buses or other wired or wireless connections, as is known in the art. The I/O controller 435 may have additional elements, which are omitted for simplicity, such as controllers, buffers (caches), drivers, repeaters, and receivers, to enable communications. Further, the computer 401 may include address, control, and/or data connections to enable appropriate communications among the aforementioned components.

[0023] In an exemplary embodiment, a conventional keyboard 450 and mouse 455 or similar devices can be coupled to the I/O controller 435. Alternatively, input may be received via a touch-sensitive or motion sensitive interface (not depicted). The computer 401 can further include a display controller 425 coupled to a display 430.

[0024] The processing circuitry 405 is a hardware device for executing software, particularly software stored in storage 420, such as cache storage, or memory 410. The processing circuitry 405 can be any custom made or commercially available computer processor, a central processing unit (CPU), an auxiliary processor among several processors associated with the computer 401, a semiconductor-based microprocessor (in the form of a microchip or chip set), a macro-processor, or generally any device for executing instructions.

[0025] The memory 410 can include any one or combination of volatile memory elements (e.g., random access memory (RAM, such as DRAM, SRAM, SDRAM, etc.)) and nonvolatile memory elements (e.g., ROM, erasable programmable read only memory (EPROM), electronically erasable programmable read only memory (EEPROM), flash memory, programmable read only memory (PROM), tape, compact disc read only memory (CD-ROM), disk, hard disk drive, diskette, cartridge, cassette or the like, etc.). Moreover, the memory 410 may incorporate electronic, magnetic, optical, and/or other types of storage media. Accordingly, the memory 410 is an example of a tangible computer readable storage medium upon which instructions executable by the processing circuitry 405 may be embodied as a computer program product. The memory 410 can have a distributed architecture, where various components are situated remote from one another, but can be accessed by the processing circuitry 405.

[0026] The instructions in memory 410 may include one or more separate programs, each of

which comprises an ordered listing of executable instructions for implementing logical functions. In the example of FIG. 4, the instructions in the memory 410 include a suitable operating system (OS) 411, SDN control logic 412, and a flow monitor 413. The operating system 411 essentially controls the execution of other computer programs and provides scheduling, input-output control, file and data management, memory management, and communication control and related services. Although depicted separately, the SDN control logic 412 and flow monitor 413 can be combined or further subdivided. When the computer 401 is in operation, the processing circuitry 405 is configured to execute instructions stored within the memory 410, to communicate data to and from the memory 410, and to generally control operations of the computer 401 pursuant to the instructions.

[0027] In an exemplary embodiment, the computer 401 can further include a network interface 460 for coupling to the secure links of a SDN network or a host machine. The network interface 460 and components of the SDN network can be configured by the SDN control logic 412 according to flow tables 416. The flow tables 416 can be populated, or augmented, by virtual processor traffic management logic 418. The virtual processor traffic management logic 418 can include computer instructions that are used to identify data packets that are being sent between virtual processors (e.g., LPARs, VMs) on a host machine, to determine actions to take for virtual processor traffic, and/or to aid in initiating a virtual switch for transmitting the data packets.

[0028] In an embodiment, the SDN controller receives policy information or configuration data from an orchestration software such as, but not limited Veritas. The policy information can input to the SDN controller via an application programming interface (API) and then be used to determine communication paths between virtual processors. These communication paths can then be used by the SDN controller to instruct the virtual switch to set up a virtual port with dedicated links between the virtual processors having communication paths. In addition, the SDN controller can instruct the virtual switch to record the virtual ports, links and other pertinent data in the flow table on the virtual switch. The virtual ports and links can be set up as part of system initialization and modified during system run time. In addition, virtual ports and links can be set up in response to a particular data packet being received at the SDN controller. Alternatively, the configuration data can be manually entered into the SDN controller.

[0029] Turning now to FIG. 5, a process flow for management of traffic between virtual processors is generally shown. In an embodiment, the process described in FIG. 5 is performed by a SDN controllers described in FIGs. 1-4. At block 502, a request from a virtual switch to identify a flow of a data packet is received at the SDN controller. In an embodiment, the virtual switch received the data packet from a source virtual

processor (e.g. a LPAR or VM) and the virtual switch requires instructions from the SDN controller on how to process the data packet. In an embodiment, the virtual switch is executing under control of a hypervisor on a host machine. At block 504, the SDN controller can determine a destination virtual processor associated with the data packet, and at block 506, the SDN controller can identify the flow between the source virtual processor and the destination virtual processor. In an embodiment, the flow includes at least one virtual port. The SDN controller can use identifier tags assigned to the virtual processors to identify the flow. At block 506, the SDN controller instructs that the virtual switch to send the data packet from the source virtual processor to the destination virtual processor via the identified flow.

[0030] Technical effects and benefits include the ability to provide complete isolation between LPARs and VMs. Further, this approach includes the ability to address a number of issues with virtual machine management which are not as highly developed as logical partition implementations. For example, there are different types of virtual machines, some of which are designed to run a complete operating system and some of which are designed to run a specific program; our proposed approach facilitates workload optimization between different types of virtual machines and LPARs. Further, the virtual machine can implement an instruction set architecture which is different from that of the physical underlying system and from the LPARs; our proposed approach facilitates network management that can optimize traffic flows between different instruction set architectures. Further, when multiple VMs are concurrently running on the same physical host, each VM may exhibit a varying and unstable performance (i.e.variable speed of execution), which highly depends on the workload imposed on the system by other virtual machines, unless proper techniques are used for temporal isolation among virtual machines; our approach facilitates this type of temporal isolation.

[0031] As will be appreciated by one of average skill in the art, aspects of embodiments may be embodied as a system, method or computer program product. Accordingly, aspects of embodiments may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as, for example, a "circuit," "module" or "system." Furthermore, aspects of embodiments may take the form of a computer program product embodied in one or more computer readable storage device(s) having computer readable program code embodied thereon.

[0032] One or more of the capabilities of embodiments can be implemented in software, firmware, hardware, or some combination thereof. Further, one or more of the capabilities can be emulated.

[0033] An embodiment may be a computer program product for enabling processor circuits to perform elements of the invention, the computer program product comprising a computer readable storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method.

[0034] The computer readable storage medium (or media), being a tangible, non-transitory, storage medium having instructions recorded thereon for causing a processor circuit to perform a method. The "computer readable storage medium" being non-transitory at least because once the instructions are recorded on the medium, the recorded instructions can be subsequently read one or more times by the processor circuit at times that are independent of the time of recording. The "computer readable storage media" being non-transitory including devices that retain recorded information only while powered (volatile devices) and devices that retain recorded information independently of being powered (non-volatile devices). An example, non-exhaustive list of "non-transitory storage media" includes, but is not limited to, for example: a semi-conductor storage device comprising, for example, a memory array such as a RAM or a memory circuit such as latch having instructions recorded thereon; a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon; an optically readable device such as a CD or DVD having instructions recorded thereon; and a magnetic encoded device such as a magnetic tape or a magnetic disk having instructions recorded thereon.

[0035] A non-exhaustive list of examples of computer readable storage medium include the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a portable compact disc read-only memory (CD-ROM). Program code can be distributed to respective computing/processing devices from an external computer or external storage device via a network, for example, the Internet, a local area network, wide area network and/or wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface card in each computing/processing device receives a program from the network and forwards the program for storage in a computer-readable storage device within the respective computing/processing device.

[0036] Computer program instructions for carrying out operations for aspects of embodiments may be for example assembler code, machine code, microcode or either source or object code written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program

code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

[0037] Aspects of embodiments are described below with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions.

[0038] These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer program instructions may also be stored in a computer readable storage medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a particular.

[0039] The computer program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatus or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0040] The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block

diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

Claims

[Claim 1] A computer-implemented method of software-defined networking (SDN) for management of traffic between virtual processors, the method comprising:
receiving, at a SDN controller, an inquiry from a virtual switch executing on a host machine, the inquiry including a request to identify a flow of a data packet received at the virtual switch from a source virtual processor, the source virtual processor one of a logical partition (LPAR) and a virtual machine (VM) executing under control of a hypervisor on the host machine;
determining, at the SDN controller, a destination virtual processor associated with the data packet;
identifying, at the SDN controller, the flow between the source virtual processor and the destination virtual processor, the flow including a least one virtual port in the virtual switch; and
instructing the virtual switch to send the data packet from the source virtual processor to the destination virtual processor via the identified flow.

[Claim 2] The method of claim 1, wherein the destination virtual processor is one of a LPAR and a VM.

[Claim 3] The method of claim 1, wherein the destination virtual processor is executing under control of the hypervisor on the host machine.

[Claim 4] The method of claim 1, wherein the source virtual processor and the destination virtual processor are executing on different host machines having different architectures.

[Claim 5] The method of claim 1, further comprising initializing the virtual switch to support traffic being sent between the source virtual processor and the destination virtual processor, the initializing including reserving the virtual port for data packets being sent between the source virtual processor and the destination virtual processor.

[Claim 6] The method of claim 1, wherein the identifying is based on configuration data received via an application programming interface (API) at the SDN controller.

[Claim 7] The method of claim 1, wherein the source and destination virtual processors are assigned virtual processor identifier tags that are utilized by the SDN controller to identify the flow.

[Claim 8] A system for software-defined networking (SDN) for management of

traffic between virtual processors, the system comprising:
a SDN controller, the system configured perform a method comprising:
receiving an inquiry from a virtual switch executing on a host machine, the inquiry including a request to identify a flow of a data packet received at the virtual switch from a source virtual processor, the source virtual processor one of a logical partition (LPAR) and a virtual machine (VM) executing under control of a hypervisor on the host machine;
determining a destination virtual processor associated with the data packet;
identifying the flow between the source virtual processor and the destination virtual processor, the flow including a least one virtual port in the virtual switch; and
instructing the virtual switch to send the data packet from the source virtual processor to the destination virtual processor via the identified flow.

[Claim 9] The system of claim 8, wherein the destination virtual processor is one of a LPAR and a VM.

[Claim 10] The system of clam 8, wherein the destination virtual processor is executing under control of the hypervisor on the host machine.

[Claim 11] The system of claim 8, wherein the source virtual processor and the destination virtual processor are executing on different host machines having different architectures.

[Claim 12] The system of claim 8, wherein the method further comprises initializing the virtual switch to support traffic being sent between the source virtual processor and the destination virtual processor, the initializing including reserving the virtual port for data packets being sent between the source virtual processor and the destination virtual processor.

[Claim 13] The system of claim 8, wherein the identifying is based on configuration data received via an application programming interface (API) at the SDN controller.

[Claim 14] The system of claim 8, wherein the source and destination virtual processors are assigned virtual processor identifier tags that are utilized by the SDN controller to identify the flow.

[Claim 15] A computer program product for software-defined networking (SDN) for management of traffic between virtual processors, the computer program product comprising:

a tangible storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method comprising:

receiving an inquiry from a virtual switch executing on a host machine, the inquiry including a request to identify a flow of a data packet received at the virtual switch from a source virtual processor, the source virtual processor one of a logical partition (LPAR) and a virtual machine (VM) executing under control of a hypervisor on the host machine;

determining a destination virtual processor associated with the data packet;

identifying the flow between the source virtual processor and the destination virtual processor, the flow including a least one virtual port in the virtual switch; and

instructing the virtual switch to send the data packet from the source virtual processor to the destination virtual processor via the identified flow.

[Claim 16] The computer program product of claim 15, wherein the destination virtual processor is one of a LPAR and a VM.

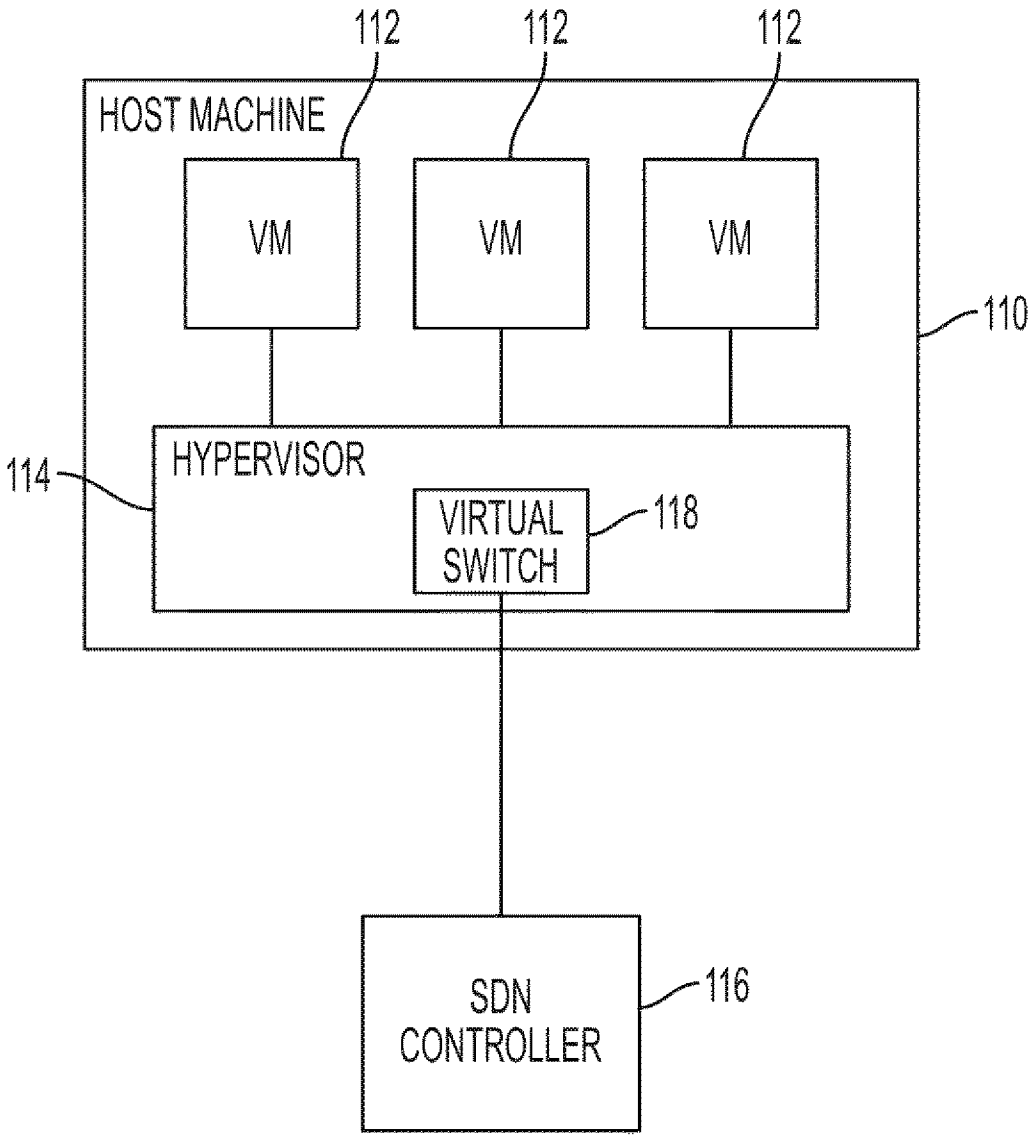
[Claim 17] The computer program product of claim 15, wherein the destination virtual processor is executing under control of the hypervisor on the host machine.

[Claim 18] The computer program product of claim 15, wherein the source virtual processor and the destination virtual processor are executing on different host machines having different architectures.

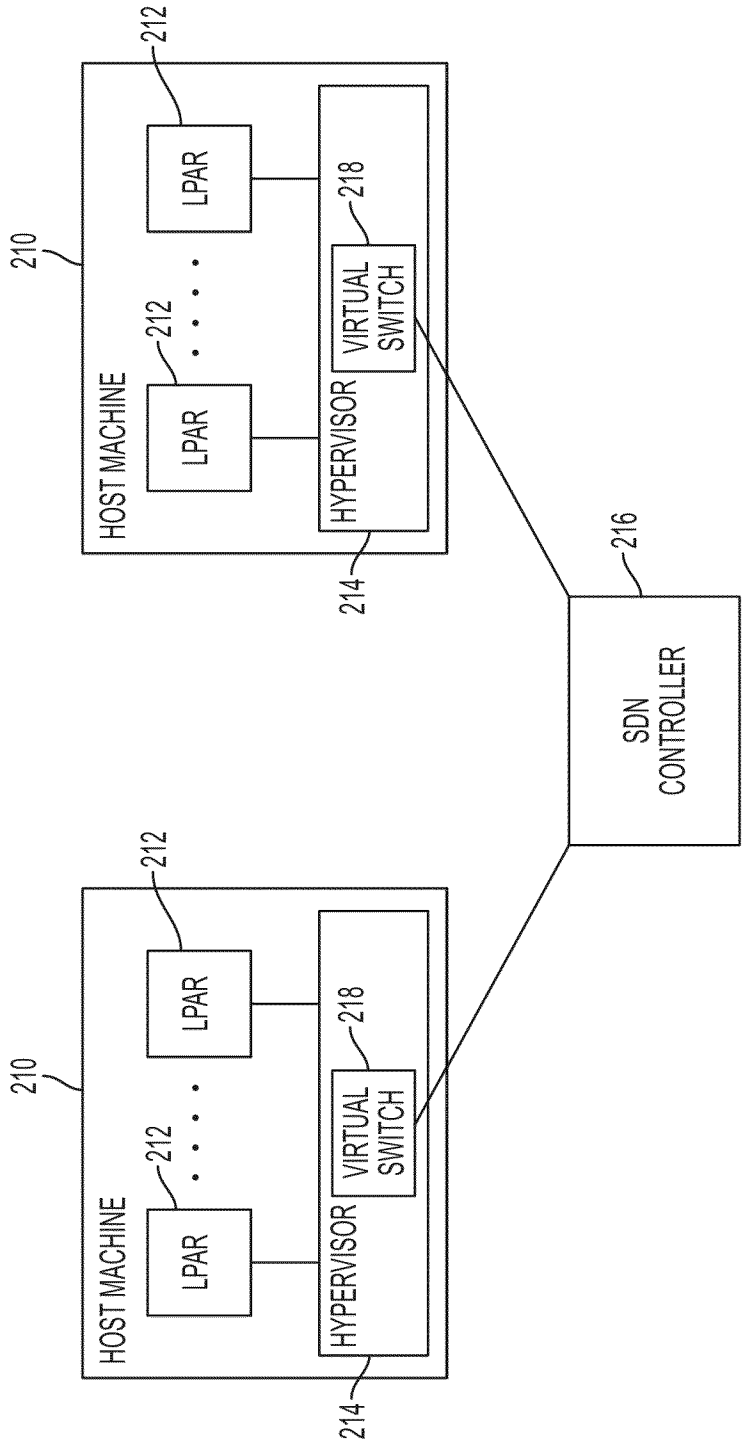
[Claim 19] The computer program product of claim 15, wherein the method further comprises initializing the virtual switch to support traffic being sent between the source virtual processor and the destination virtual processor, the initializing including reserving the virtual port for data packets being sent between the source virtual processor and the destination virtual processor.

[Claim 20] The computer program product of claim 15, wherein the identifying is based on configuration data received via an application programming interface (API) at the SDN controller.

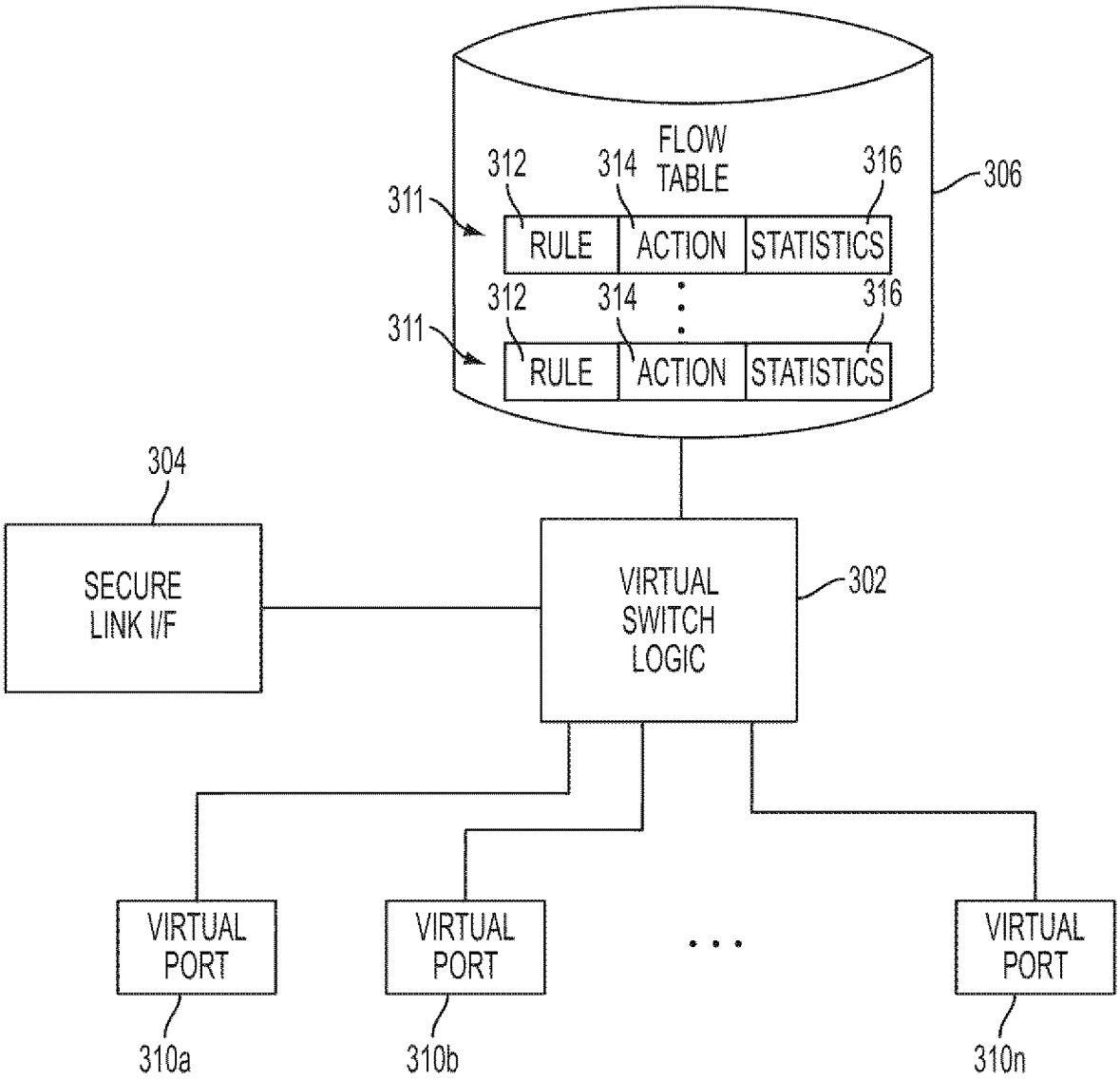
[Fig. 1]


112
112
112
HOST MACHINE
VM
VM
VM
110
114
HYPERVISOR
VIRTUAL SWITCH
118
SDN CONTROLLER
116

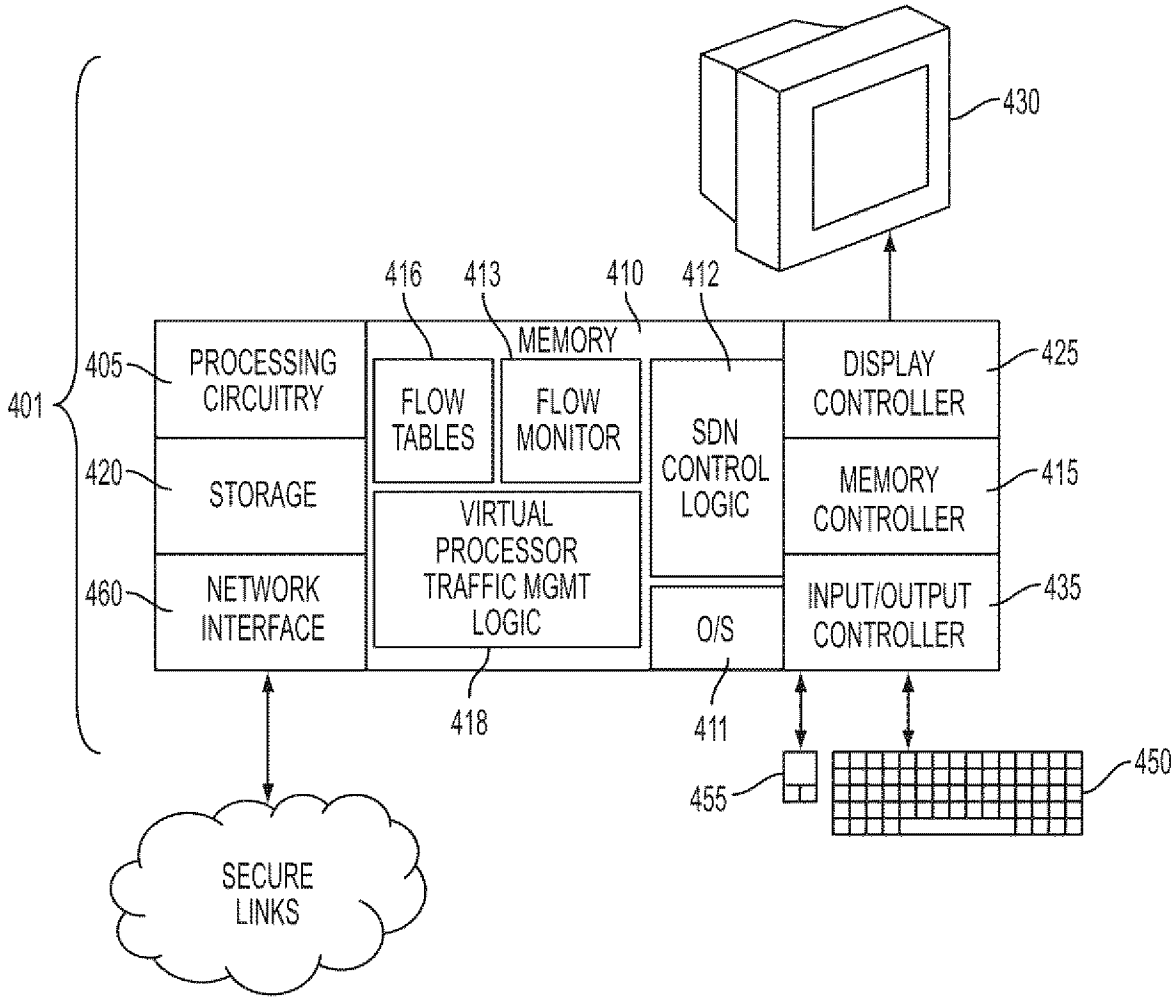
[Fig. 2]


210
212
LPAR
LPAR
HOST MACHINE
HYPERVISOR
VIRTUAL SWITCH
218
214
216
SDN CONTROLLER

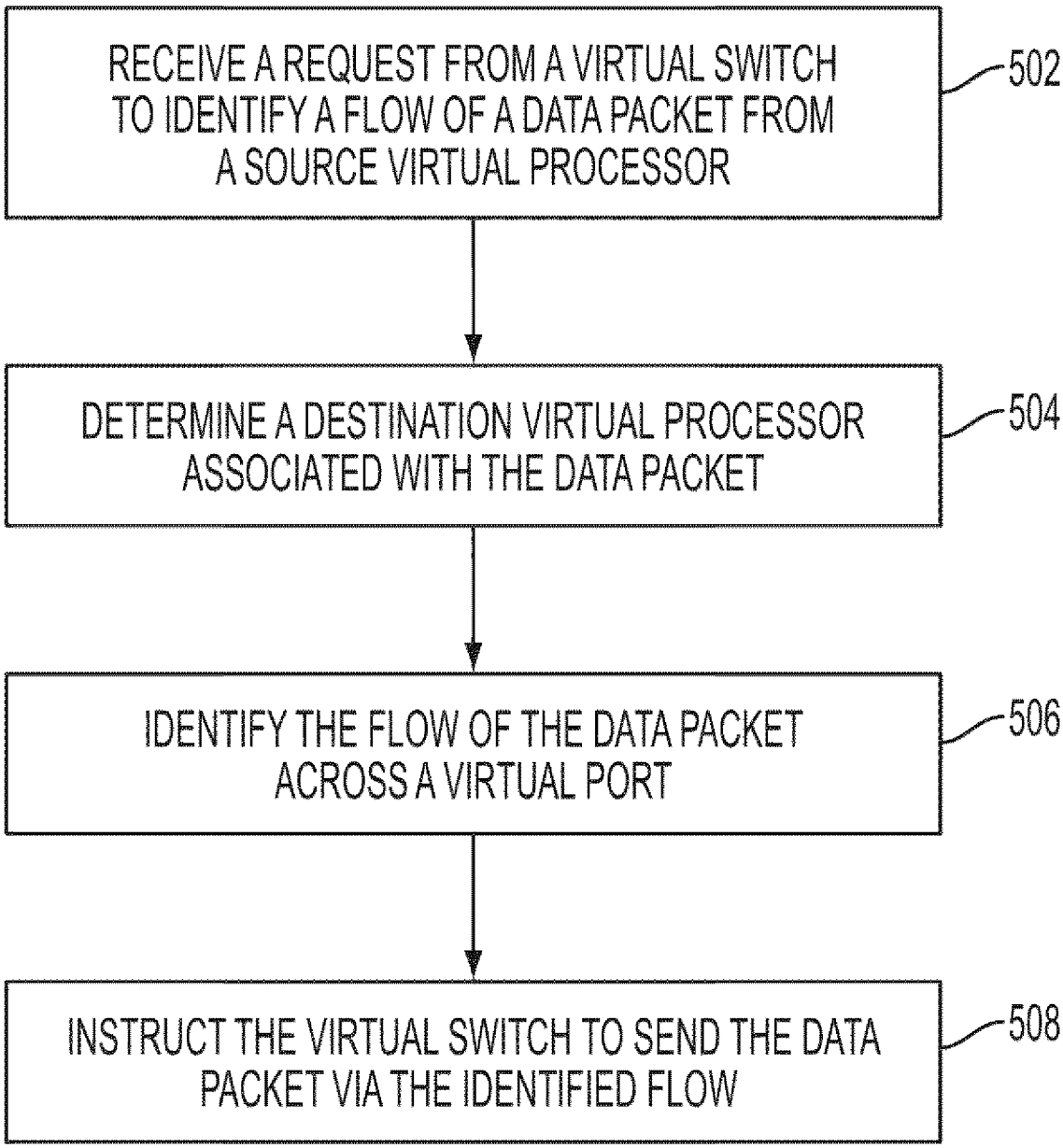
[Fig. 3]


FLOW TABLE
306
311
312
314
316
RULE
ACTION
STATISTICS
311
312
314
316
RULE
ACTION
STATISTICS
304
SECURE LINK I/F
VIRTUAL SWITCH LOGIC
302
VIRTUAL PORT
VIRTUAL PORT
VIRTUAL PORT
310a
310b
310n

[Fig. 4]


401
405
PROCESSING CIRCUITRY
420
STORAGE
460
NETWORK INTERFACE
416
413
410
412
MEMORY
FLOW TABLES
FLOW MONITOR
VIRTUAL PROCESSOR TRAFFIC MGMT LOGIC
SDN CONTROL LOGIC
O/S
418
411
DISPLAY CONTROLLER
MEMORY CONTROLLER
INPUT/OUTPUT CONTROLLER
425
415
435
430
455
450
SECURE LINKS

[Fig. 5]


RECEIVE A REQUEST FROM A VIRTUAL SWITCH TO IDENTIFY A FLOW OF A DATA PACKET FROM A SOURCE VIRTUAL PROCESSOR
502
DETERMINE A DESTINATION VIRTUAL PROCESSOR ASSOCIATED WITH THE DATA PACKET
504
IDENTIFY THE FLOW OF THE DATA PACKET ACROSS A VIRTUAL PORT
506
INSTRUCT THE VIRTUAL SWITCH TO SEND THE DATA PACKET VIA THE IDENTIFIED FLOW
508

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2014/005105

A. CLASSIFICATION OF SUBJECT MATTER

Int.Cl. G06F13/10(2006.01)i, G06F9/46(2006.01)i, G06F13/00(2006.01)i, H04L12/717(2013.01)i

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)

Int.Cl. G06F13/10, G06F9/46, G06F13/00, H04L12/717

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

Published examined utility model applications of Japan 1922-1996
Published unexamined utility model applications of Japan 1971-2015
Registered utility model specifications of Japan 1996-2015
Published registered utility model applications of Japan 1994-2015

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2011/0299537 A1 (Nakul Pratap SARAIYA, Lawrence Mcgoff MATTER, Alok RISHI) 2011.12.08, paragraphs. [0021] to [0025], [0036]; figs. 1, 3 (No Family)	1-20
A	EP 2651081 A1 (NEC CORPORATION) 2013.10.16, paragraphs. [0029], [0031], [0055], [0057], [0071] to [0074]; fig. 2 & WO 2012/077603 A1 & US 2013/0254891 A1 & CN 103250392 A & JP 2014-147120 A	1-20
A	US 2011/0032944 A1 (Uri ELZUR, Patricia Ann THALER, Hemal SHAH) 2011.02.10, paragraphs. [0026], [0027], [0054]; figs. 1, 3B (No Family)	6, 13, 20

☑ 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
21.01.2015	03.02.2015

Name and mailing address of the ISA/JP
Japan Patent Office
3-4-3, Kasumigaseki, Chiyoda-ku, Tokyo 100-8915, Japan

Authorized officer
KOBAYASHI, Hidekazu 5T 3449
Telephone No. +81-3-3581-1101 Ext. 3568

Form PCT/ISA/210 (second sheet) (July 2009)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2014/005105

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2012/0079478 A1 (CISCO TECHNOLOGY, INC.) 2012.03.29, paragraph. [0017]; fig. 2 & WO 2012/039792 A1 & CN 103141058 A	7, 14
A	EP 2485155 A1 (NEC CORPORATION) 2012.08.08, figs. 1, 12 & WO 2011/037148 A1 & US 2012/0185856 A1 & CN 102576343 A & JP 2011-070549 A	4, 11, 18