US 20220100432A1

(19) **United States**

(12) **Patent Application Publication**

Kim et al.

(10) **Pub. No.: US 2022/0100432 A1**

(43) **Pub. Date: Mar. 31, 2022**

(54) **DISTRIBUTED STORAGE SERVICES SUPPORTED BY A NIC**

(71) Applicant: **VMware, Inc.**, Palo Alto, CA (US)

(72) Inventors: **Jinpyo Kim**, Cupertino, CA (US); **Claudio Fleiner**, San Jose, CA (US); **Marc Fleischmann**, Palo Alto, CA (US); **Shoby A. Cherian**, Dublin, CA (US); **Anjaneya P. Gondi**, Fremont, CA (US)

(21) Appl. No.: **17/145,318**

(22) Filed: **Jan. 9, 2021**

Related U.S. Application Data

(60) Provisional application No. 63/084,513, filed on Sep. 28, 2020.

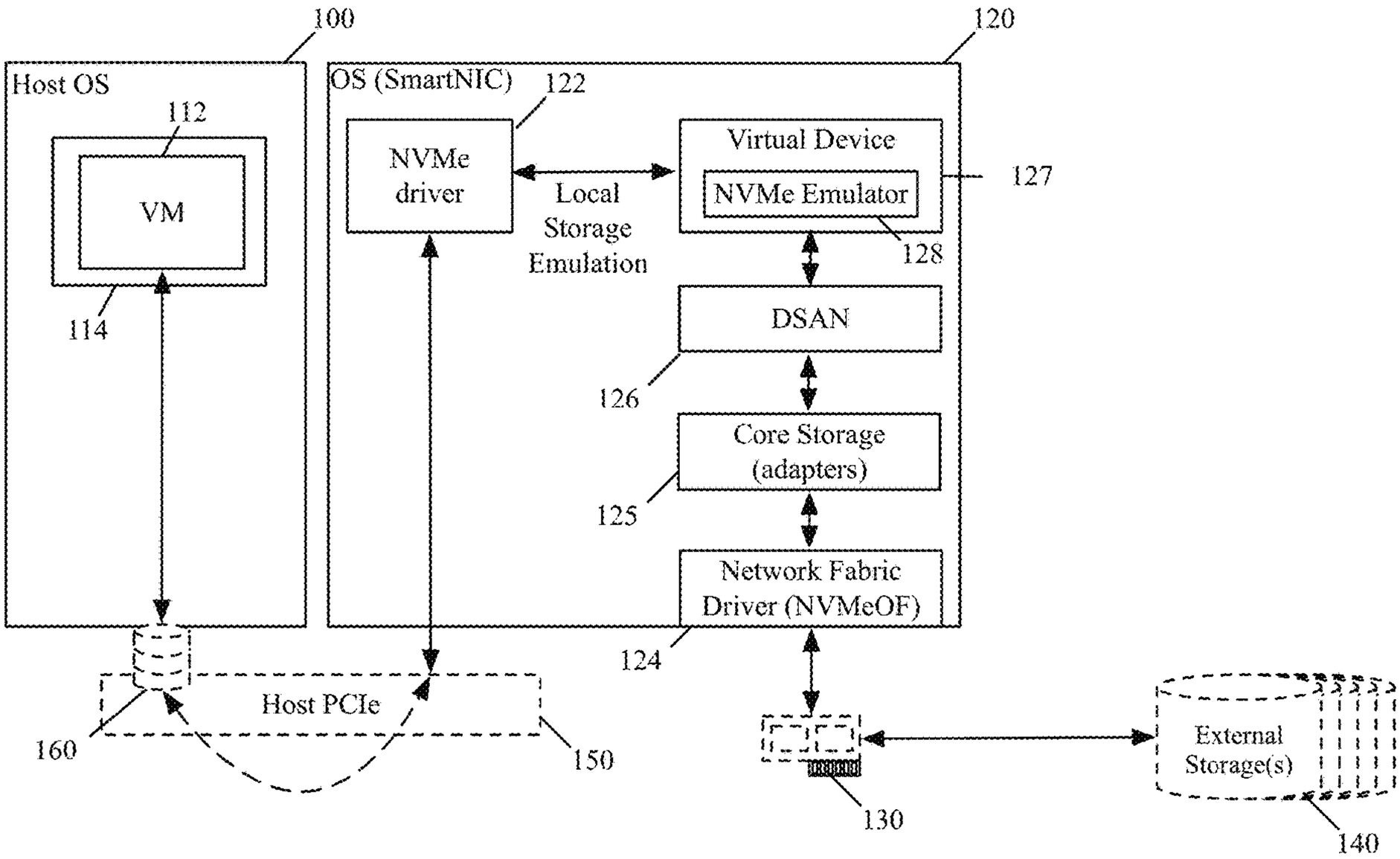
Publication Classification

(51) **Int. Cl.**
G06F 3/06 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 3/0664*** (2013.01); ***G06F 3/0641*** (2013.01); ***G06F 3/067*** (2013.01); ***G06F 3/0604*** (2013.01); ***G06F 3/0649*** (2013.01)

(57) **ABSTRACT**

Some embodiments provide a method of providing distributed storage services to a host computer from a network interface card (NIC) of the host computer. At the NIC, the method accesses a set of one or more external storages operating outside of the host computer through a shared port of the NIC that is not only used to access the set of external storages but also for forwarding packets not related to an external storage. In some embodiments, the method accesses the external storage set by using a network fabric storage driver that employs a network fabric storage protocol to access the external storage set. The method presents the external storage as a local storage of the host computer to a set of programs executing on the host computer. In some embodiments, the method presents the local storage by using a storage emulation layer on the NIC to create a local storage construct that presents the set of external storages as a local storage of the host computer.



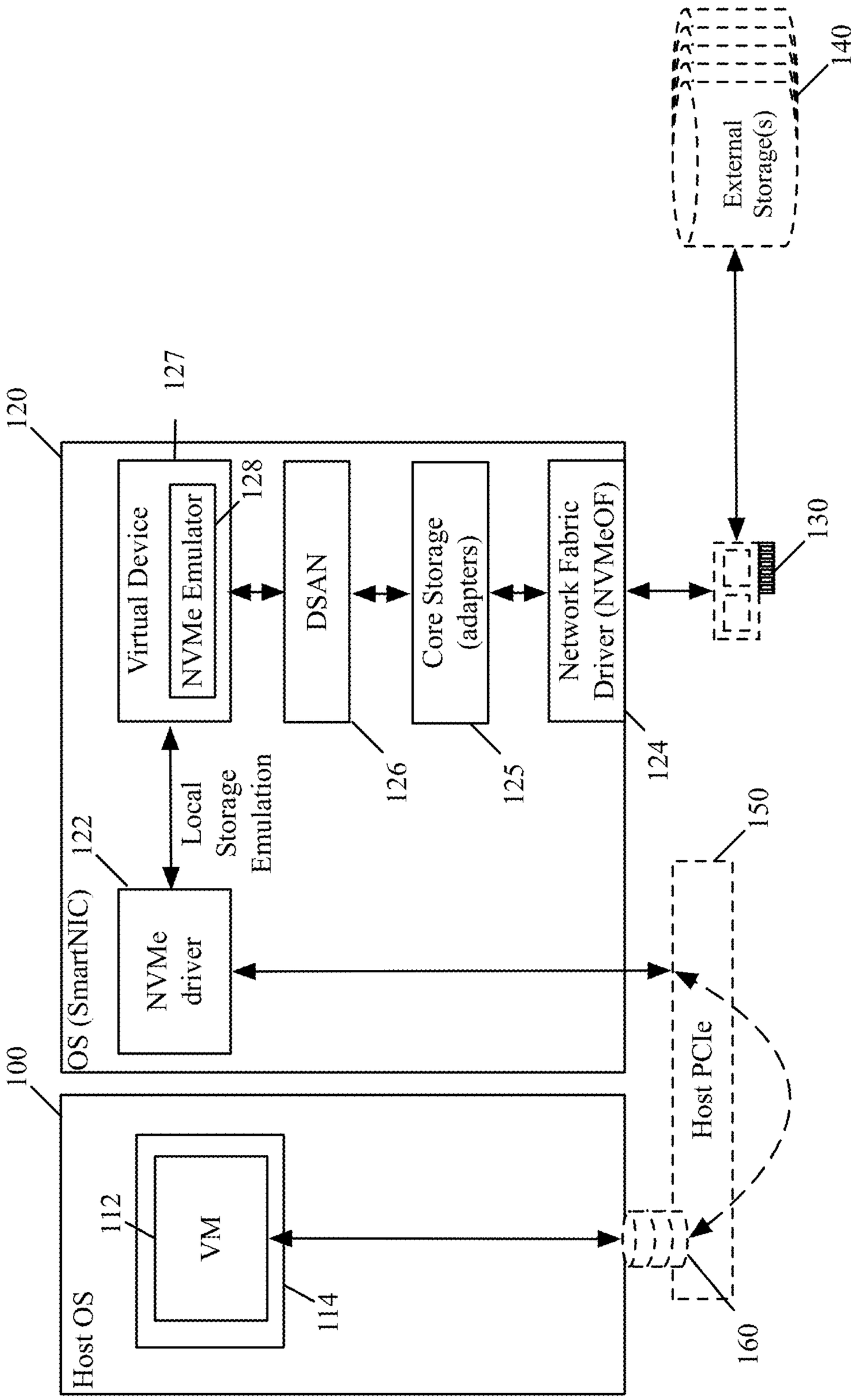

External Storage(s)
140
120
127
Virtual Device
NVMe Emulator
128
DSAN
Core Storage (adapters)
Network Fabric Driver (NVMeOF)
126
125
124
130
150
122
Local Storage Emulation
NVMe driver
OS (SmartNIC)
100
Host OS
112
VM
114
Host PCIe
160

Figure 1

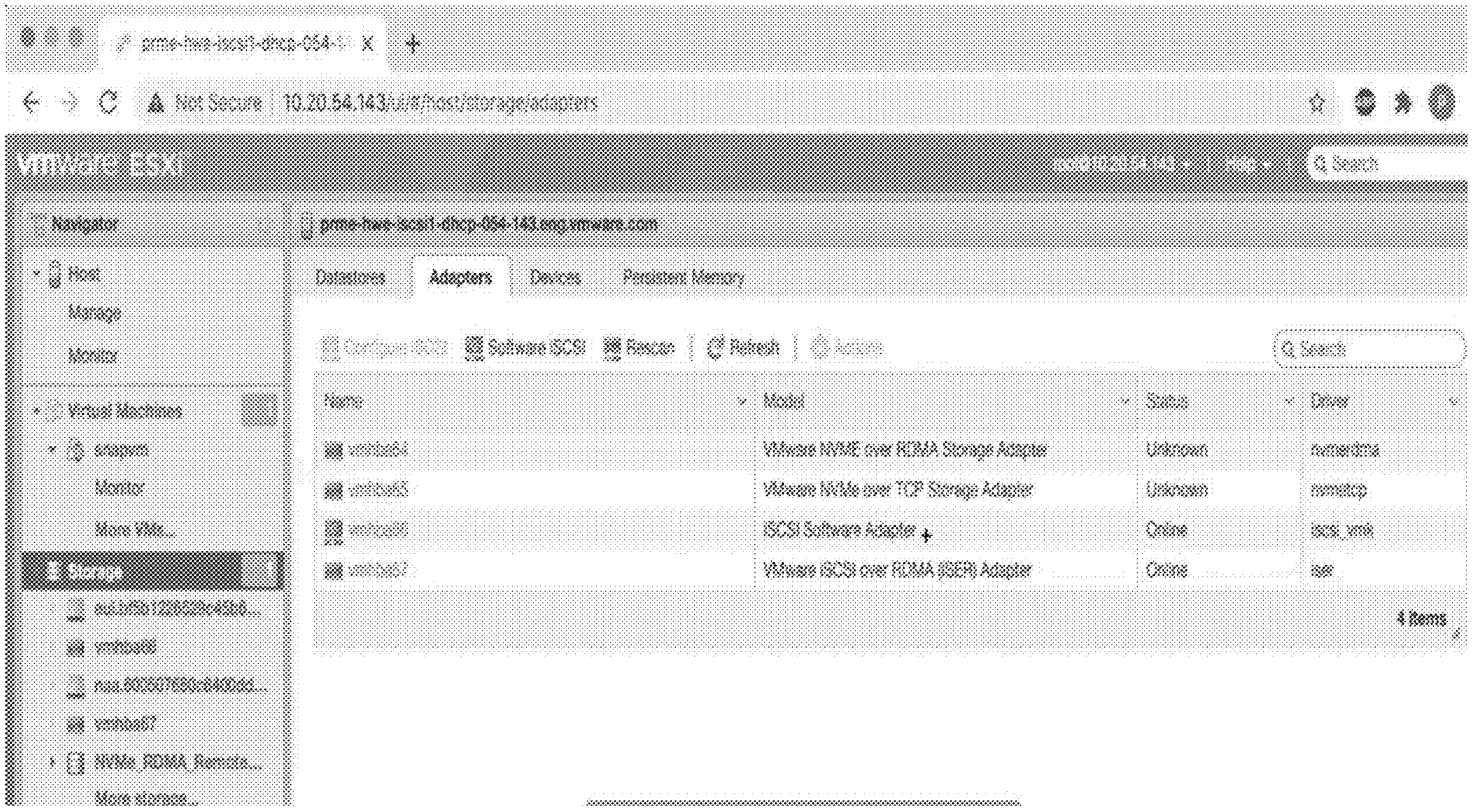

Not Secure | 10.20.54.143/ui/#/host/storage/adapters
vmware ESX
Navigator
Host
Manage
Monitor
Virtual Machines
Monitor
More VMs...
Storage
More storage...
Datastores
Adapters
Devices
Persistent Memory
Software iSCSI
Rescan
Refresh
Search
Name
Model
Status
Driver
VMware NVME over RDMA Storage Adapter
Unknown
VMware NVMe over TCP Storage Adapter
Unknown
iSCSI Software Adapter
Online
VMware iSCSI over RDMA (iSER) Adapter
Online
4 items

Figure 2

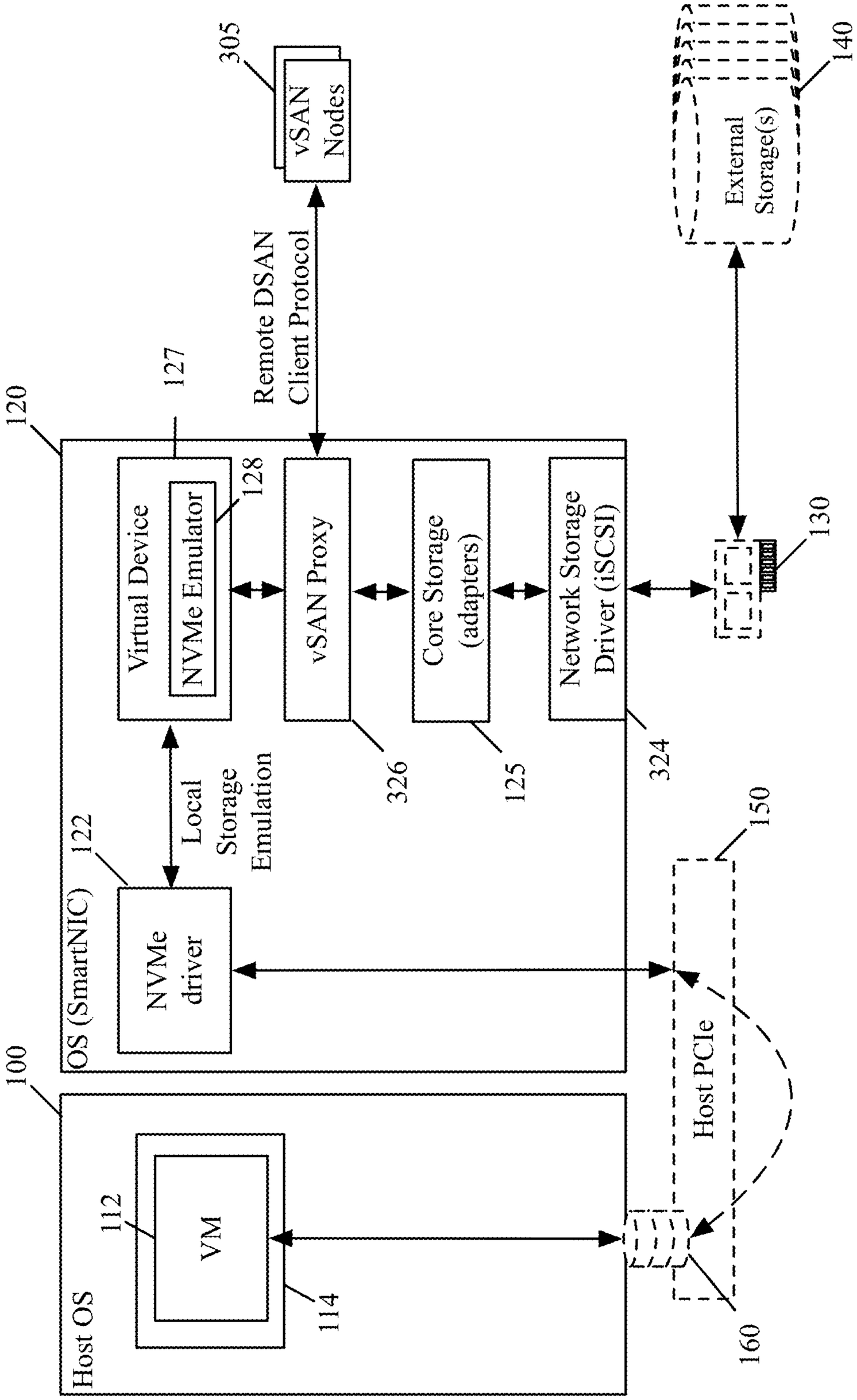

305
vSAN Nodes
Remote DSAN Client Protocol
External Storage(s)
140
120
127
Virtual Device
NVMe Emulator
128
vSAN Proxy
Core Storage (adapters)
Network Storage Driver (iSCSI)
326
125
324
130
150
122
Local Storage Emulation
OS (SmartNIC)
NVMe driver
100
Host OS
112
VM
114
Host PCIe
160

Figure 3

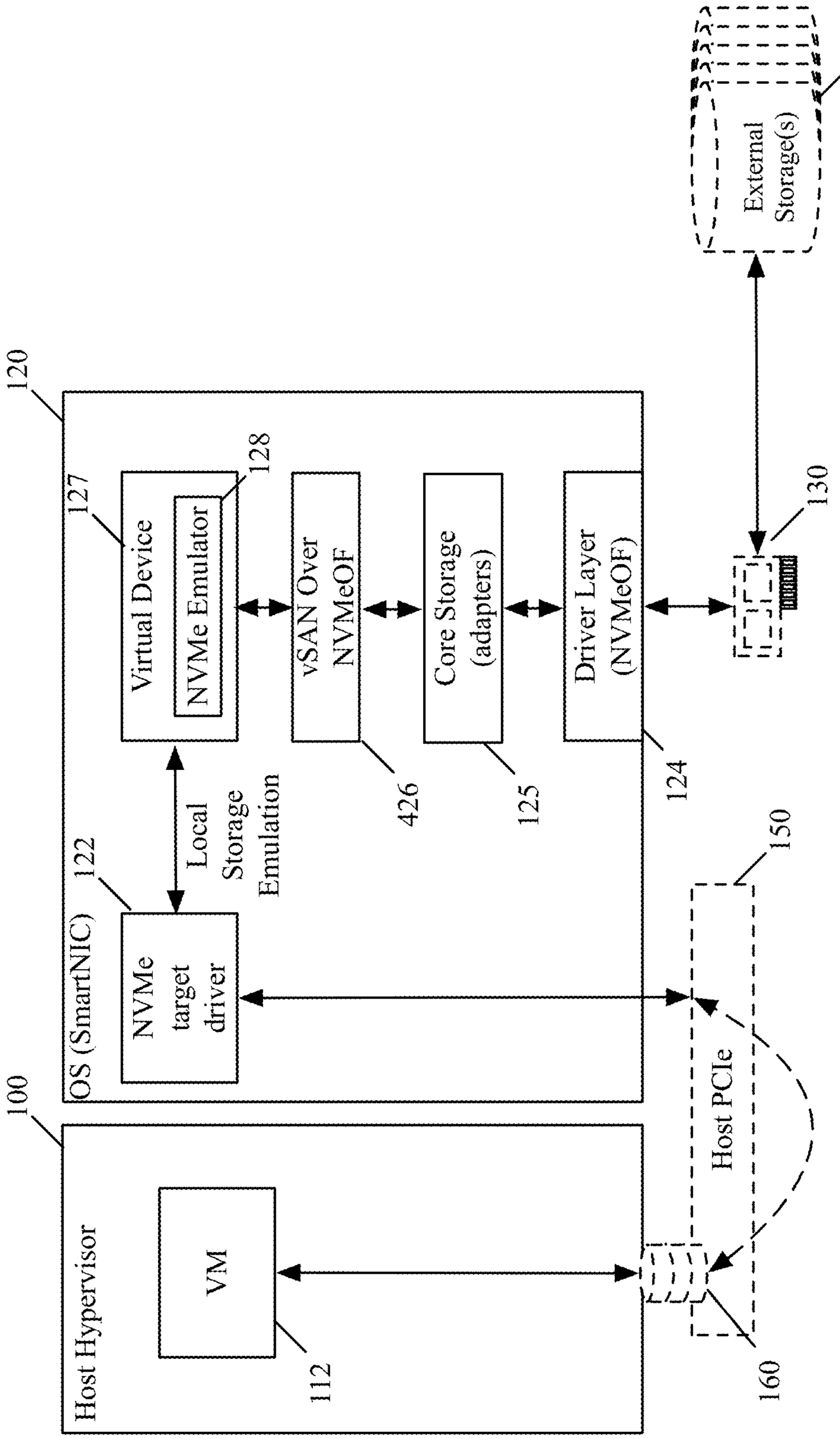

Host Hypervisor
VM
112
100
OS (SmartNIC)
NVMe target driver
122
Local Storage Emulation
Virtual Device
NVMe Emulator
127
128
vSAN Over NVMeOF
426
Core Storage (adapters)
125
Driver Layer (NVMeOF)
124
120
External Storage(s)
140
130
150
Host PCIe
160

Figure 4

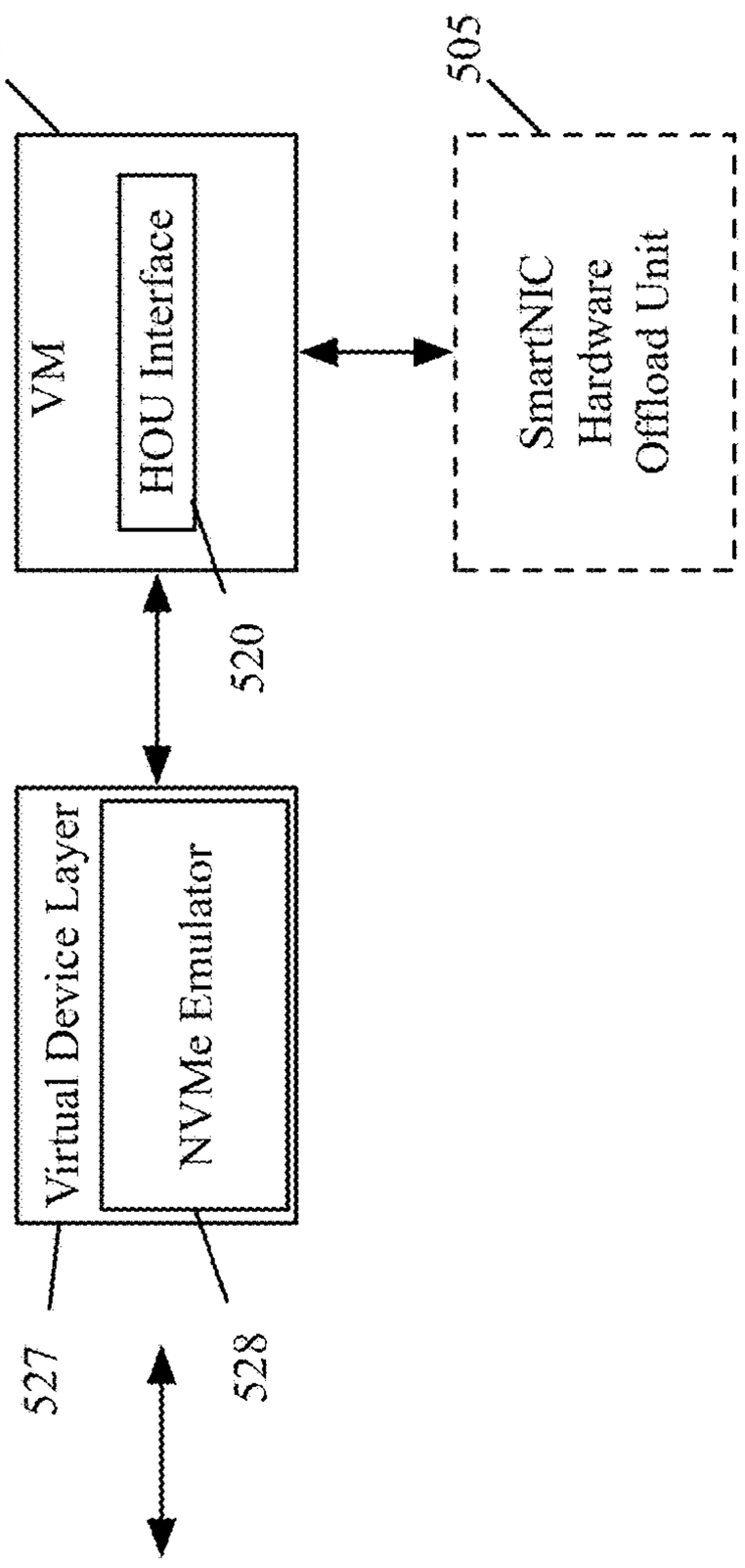
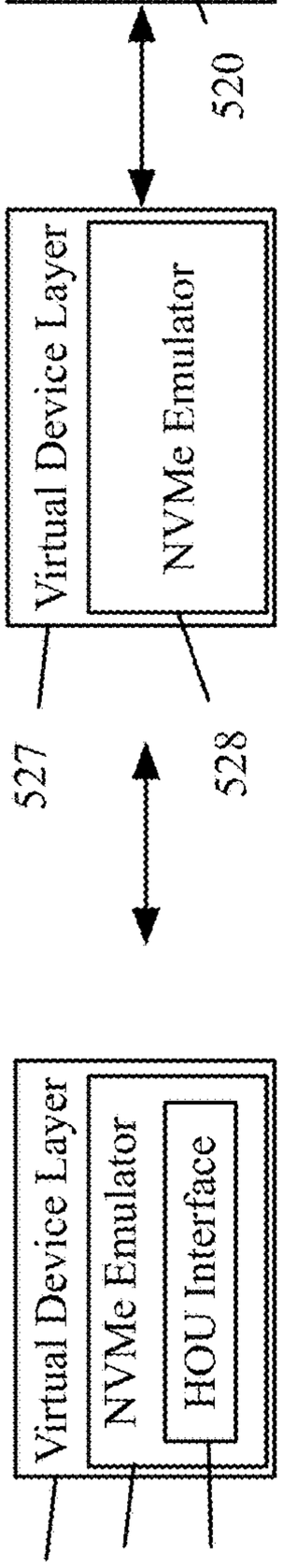


Figure 5



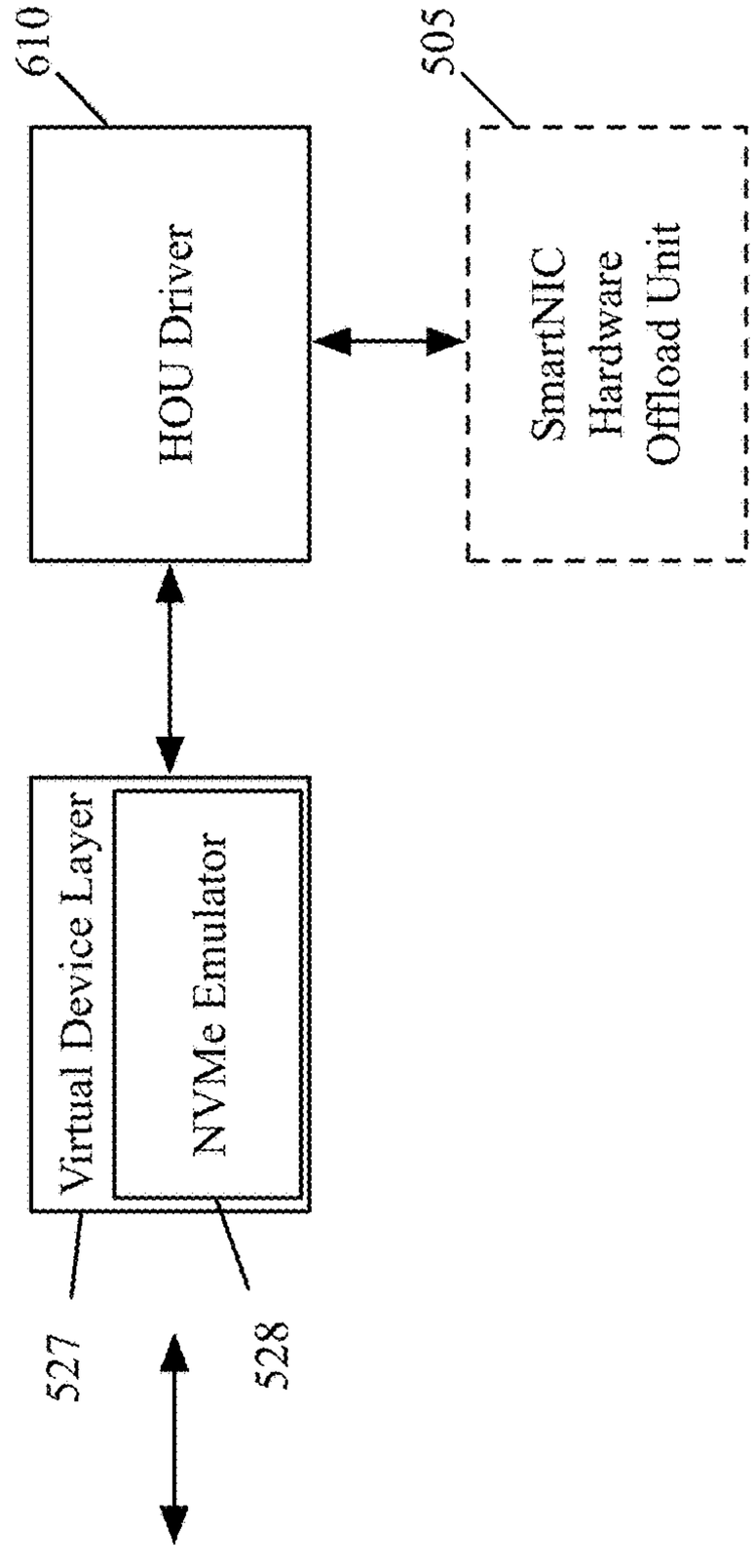


Figure 6

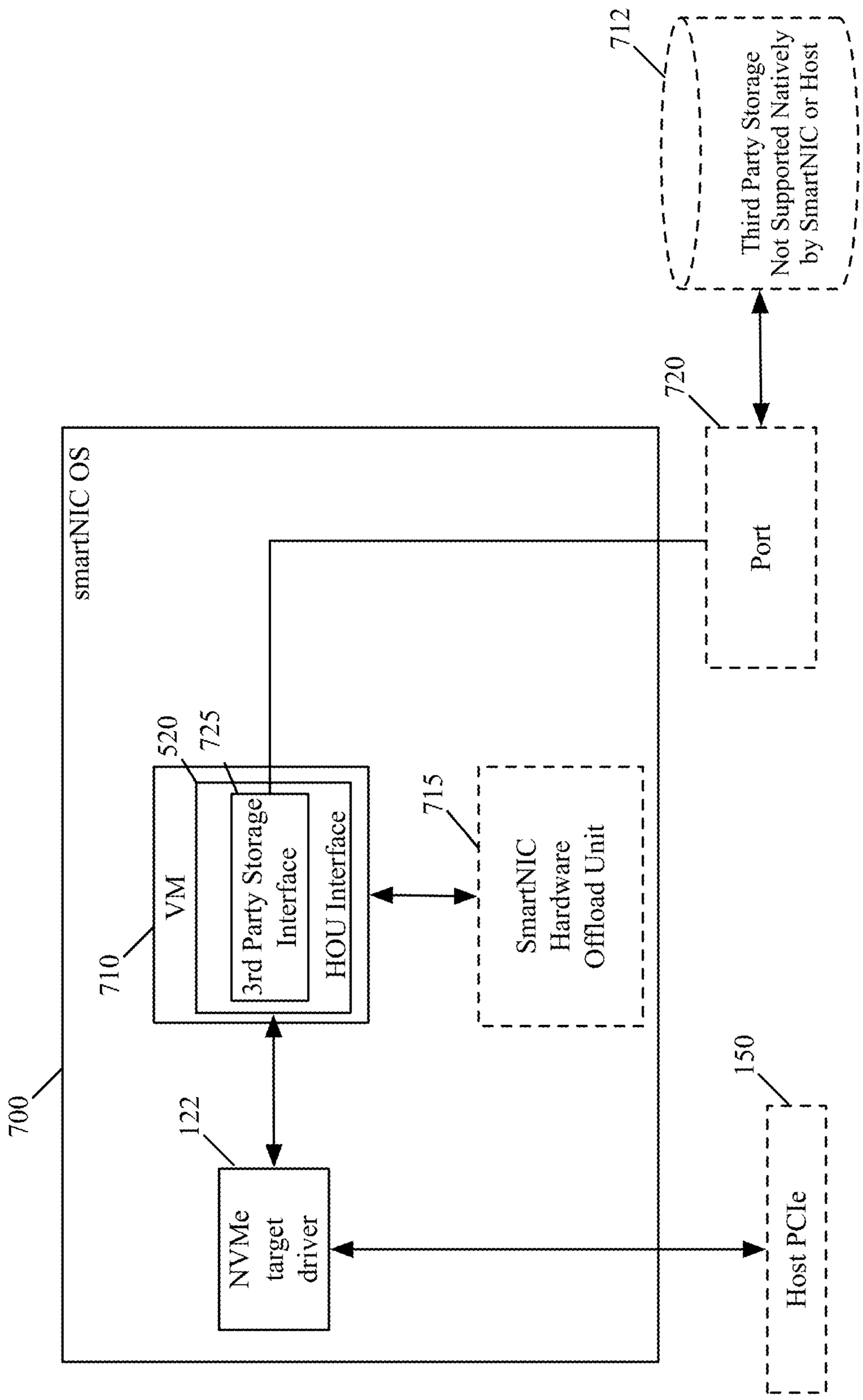

712
Third Party Storage
Not Supported Natively
by SmartNIC or Host
720
Port
smartNIC OS
520
725
710
VM
3rd Party Storage
Interface
HOU Interface
715
SmartNIC
Hardware
Offload Unit
700
122
NVMe
target
driver
150
Host PCIe

Figure 7

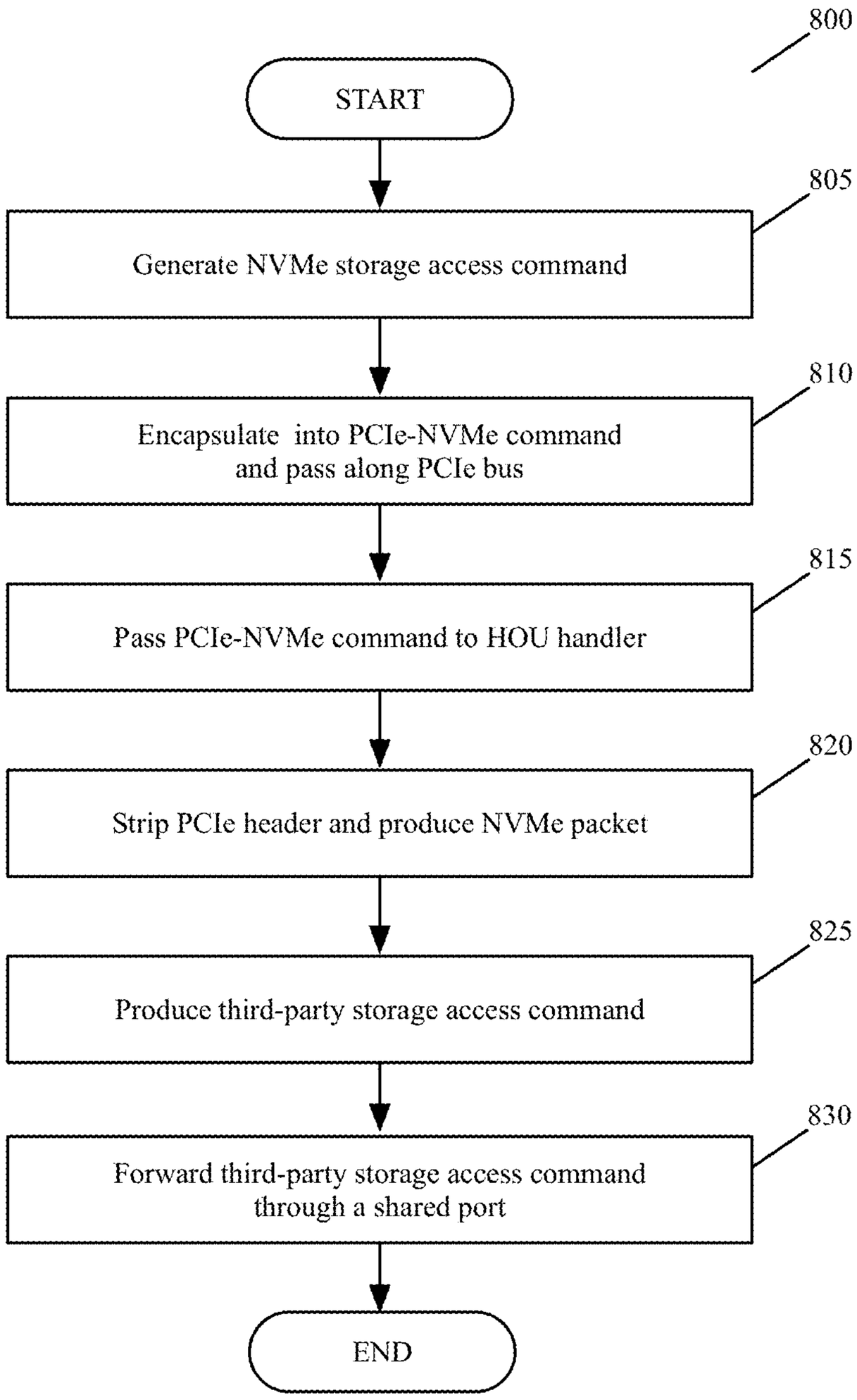

800
START
805
Generate NVMe storage access command
810
Encapsulate into PCIe-NVMe command and pass along PCIe bus
815
Pass PCIe-NVMe command to HOU handler
820
Strip PCIe header and produce NVMe packet
825
Produce third-party storage access command
830
Forward third-party storage access command through a shared port
END

Figure 8

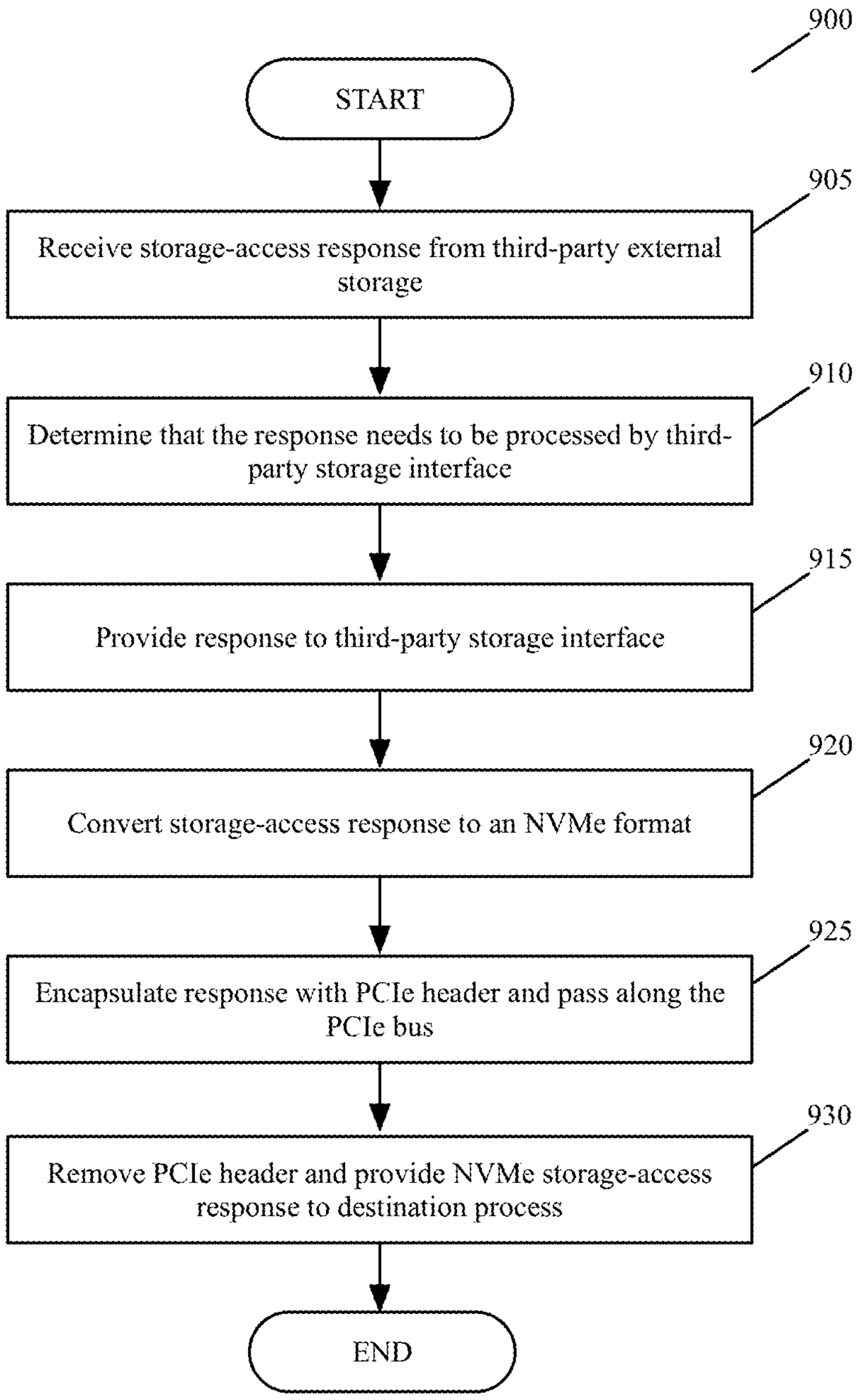

900
START
905
Receive storage-access response from third-party external storage
910
Determine that the response needs to be processed by third-party storage interface
915
Provide response to third-party storage interface
920
Convert storage-access response to an NVMe format
925
Encapsulate response with PCIe header and pass along the PCIe bus
930
Remove PCIe header and provide NVMe storage-access response to destination process
END

Figure 9

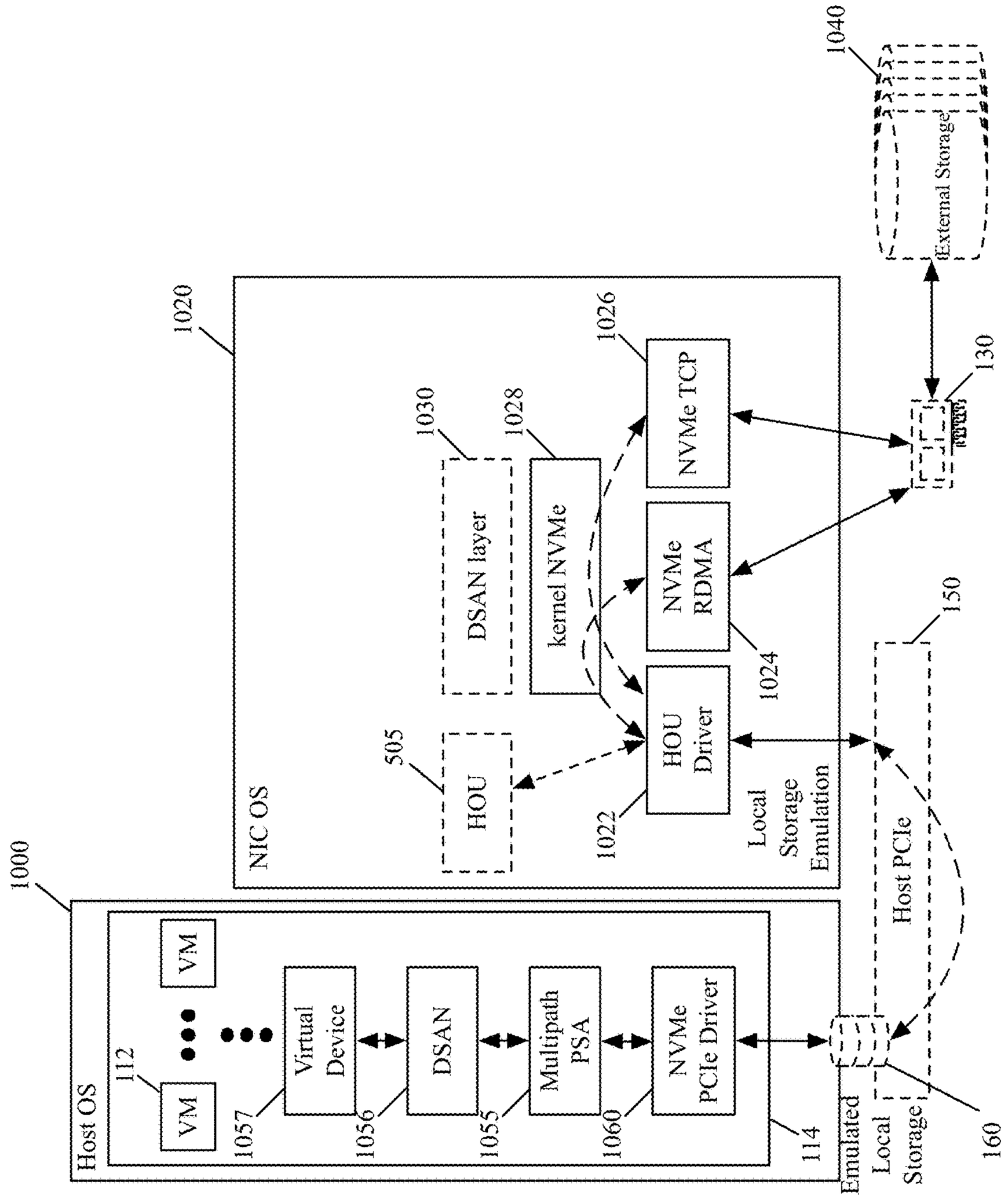

1000
Host OS
112
VM
VM
1057
Virtual Device
1056
DSAN
1055
Multipath PSA
1060
NVMe PCIe Driver
114
Emulated Local Storage
160
NIC OS
1020
505
HOU
1022
HOU Driver
1024
Local Storage Emulation
1030
DSAN layer
1028
kernel NVMe
1026
NVMe TCP
NVMe RDMA
150
Host PCIe
130
1040
External Storage

Figure 10

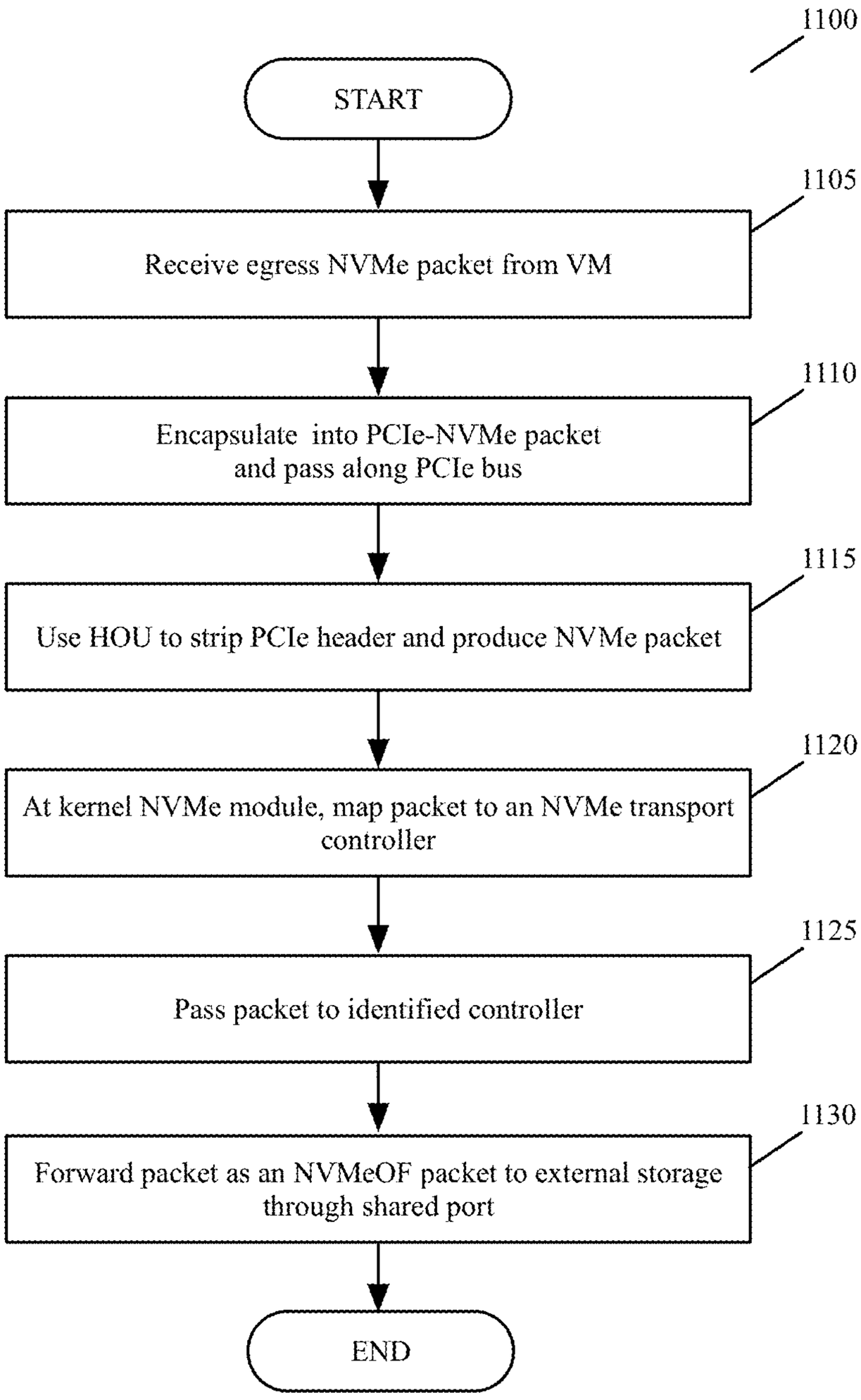

1100
START
1105
Receive egress NVMe packet from VM
1110
Encapsulate into PCIe-NVMe packet and pass along PCIe bus
1115
Use HOU to strip PCIe header and produce NVMe packet
1120
At kernel NVMe module, map packet to an NVMe transport controller
1125
Pass packet to identified controller
1130
Forward packet as an NVMeOF packet to external storage through shared port
END

Figure 11

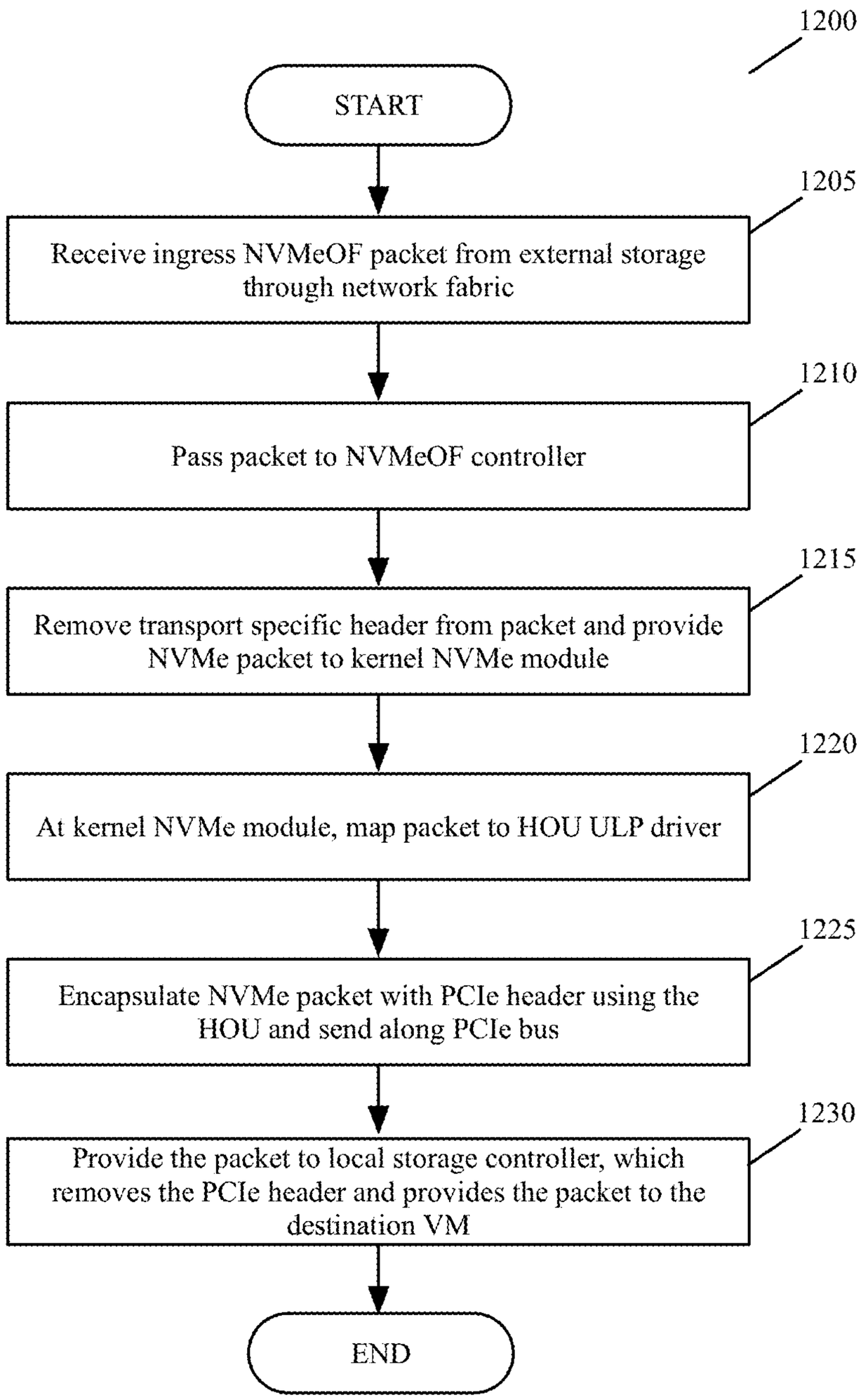

1200
START
1205
Receive ingress NVMeOF packet from external storage through network fabric
1210
Pass packet to NVMeOF controller
1215
Remove transport specific header from packet and provide NVMe packet to kernel NVMe module
1220
At kernel NVMe module, map packet to HOU ULP driver
1225
Encapsulate NVMe packet with PCIe header using the HOU and send along PCIe bus
1230
Provide the packet to local storage controller, which removes the PCIe header and provides the packet to the destination VM
END

Figure 12

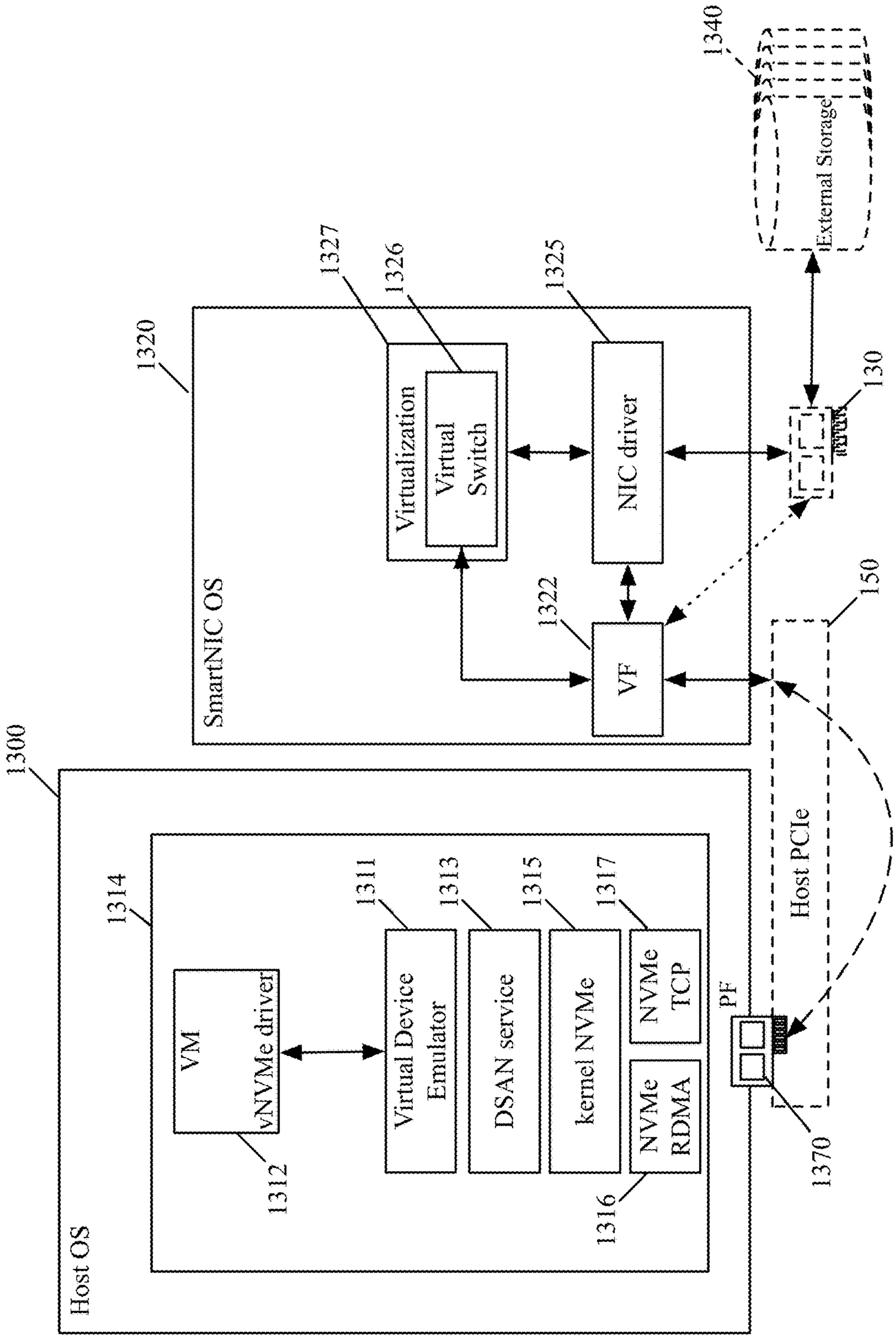

1340
External Storage
1327
1326
1325
1320
Virtualization
Virtual Switch
NIC driver
130
1322
VF
150
SmartNIC OS
1300
Host PCIe
1314
1311
1313
1315
1317
VM
vNVMe driver
1312
Virtual Device Emulator
DSAN service
kernel NVMe
NVMe TCP
NVMe RDMA
1316
PF
1370
Host OS

Figure 13

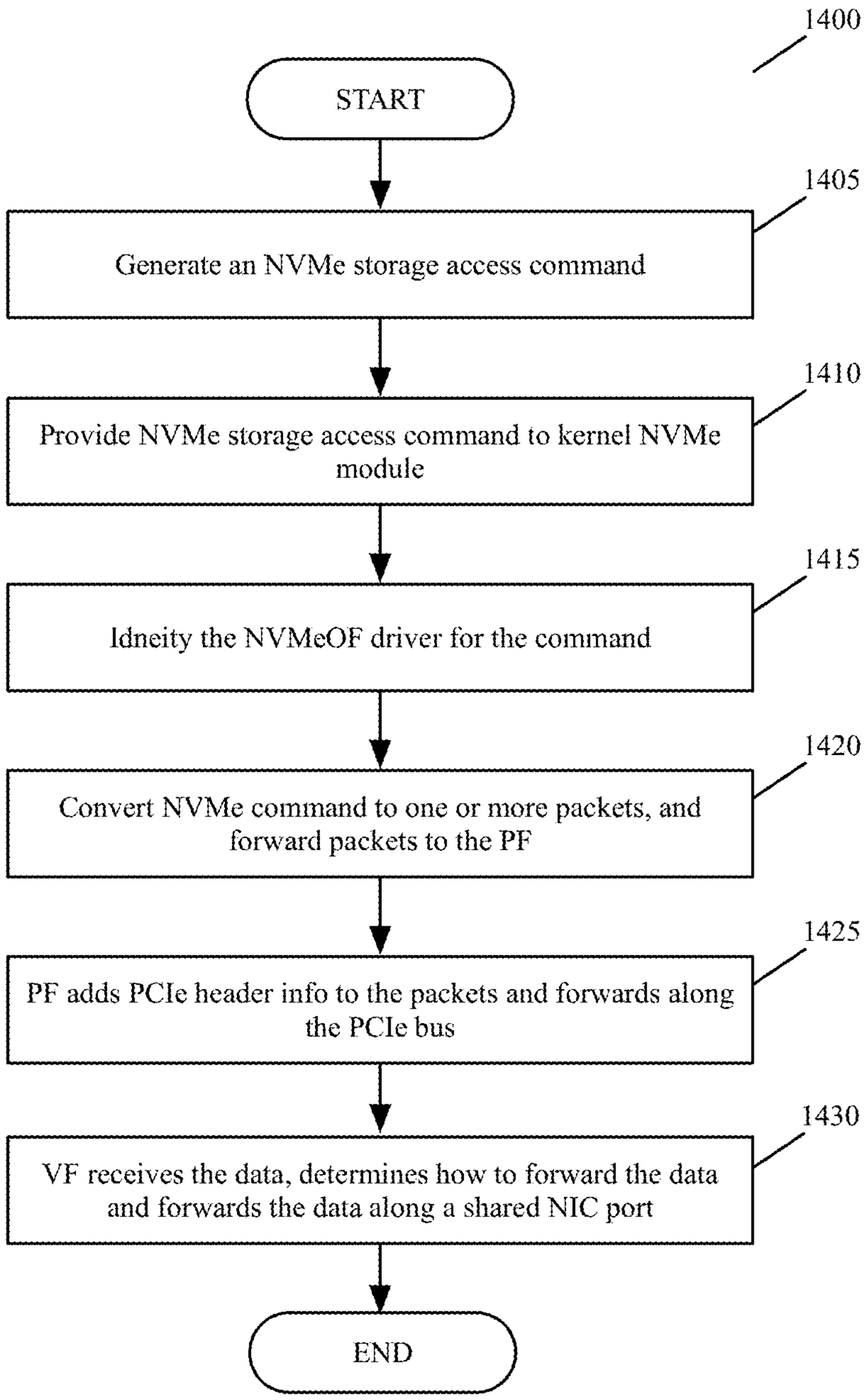

1400
START
1405
Generate an NVMe storage access command
1410
Provide NVMe storage access command to kernel NVMe module
1415
Idneity the NVMeOF driver for the command
1420
Convert NVMe command to one or more packets, and forward packets to the PF
1425
PF adds PCIe header info to the packets and forwards along the PCIe bus
1430
VF receives the data, determines how to forward the data and forwards the data along a shared NIC port
END

Figure 14

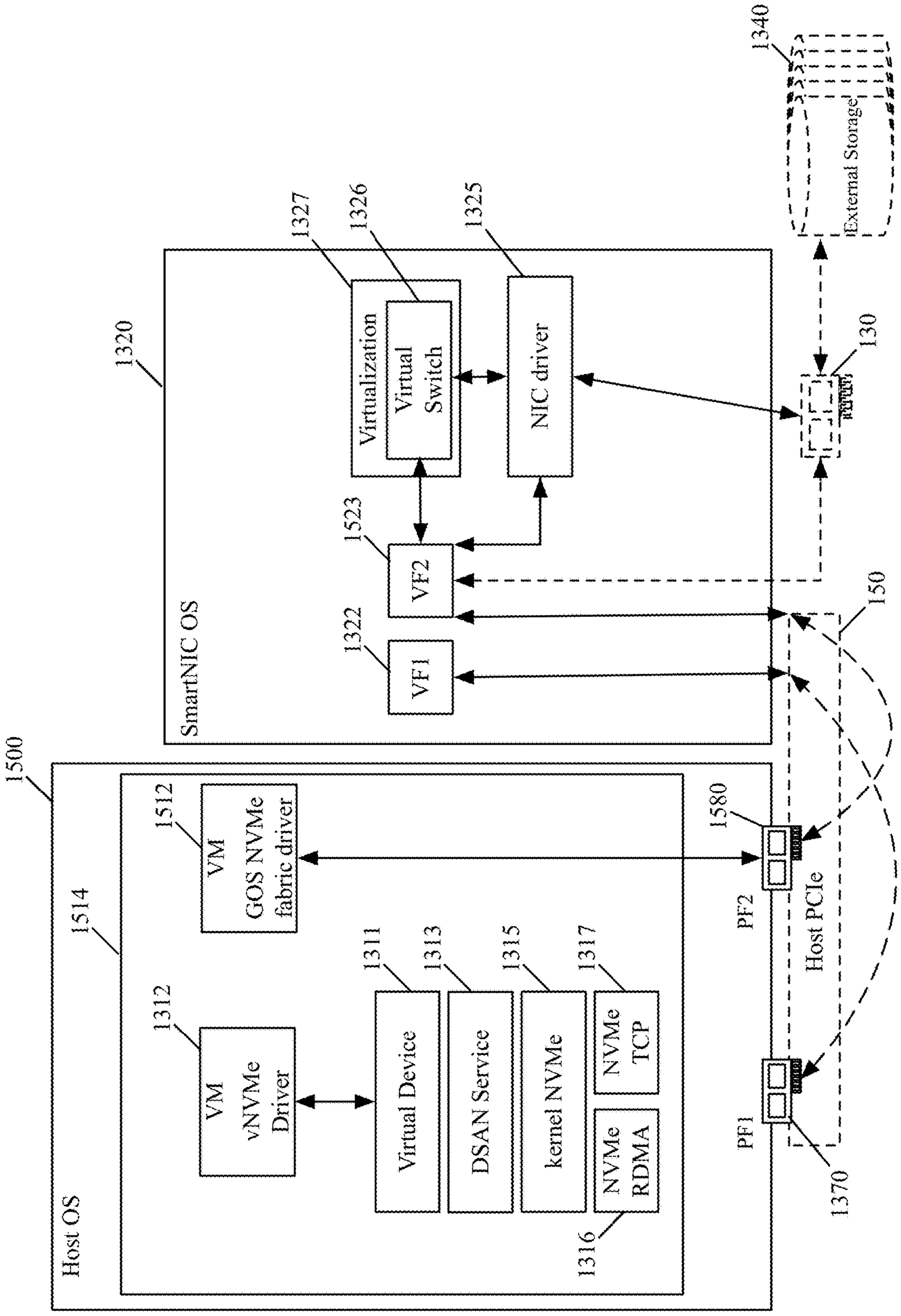

1340
External Storage
1327
1326
1325
1320
Virtualization
Virtual Switch
NIC driver
130
1523
VF2
1322
VF1
SmartNIC OS
150
1500
1512
VM
GOS NVMe fabric driver
1580
1514
1311
1313
1315
1317
PF2
Host PCIe
1312
VM
vNVMe Driver
Virtual Device
DSAN Service
kernel NVMe
NVMe TCP
NVMe RDMA
1316
PF1
1370
Host OS

Figure 15

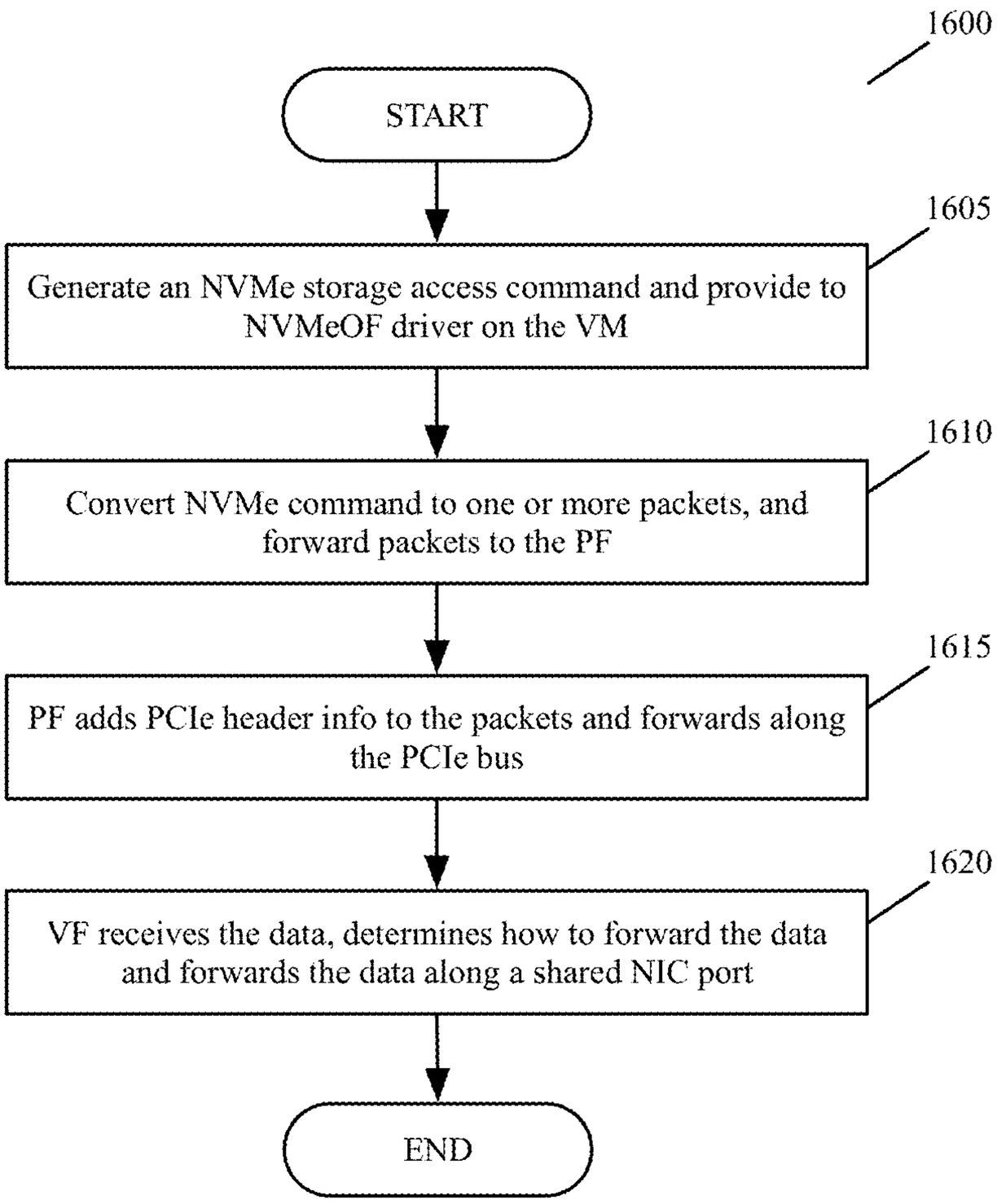

1600
START
1605
Generate an NVMe storage access command and provide to NVMeOF driver on the VM
1610
Convert NVMe command to one or more packets, and forward packets to the PF
1615
PF adds PCIe header info to the packets and forwards along the PCIe bus
1620
VF receives the data, determines how to forward the data and forwards the data along a shared NIC port
END

Figure 16

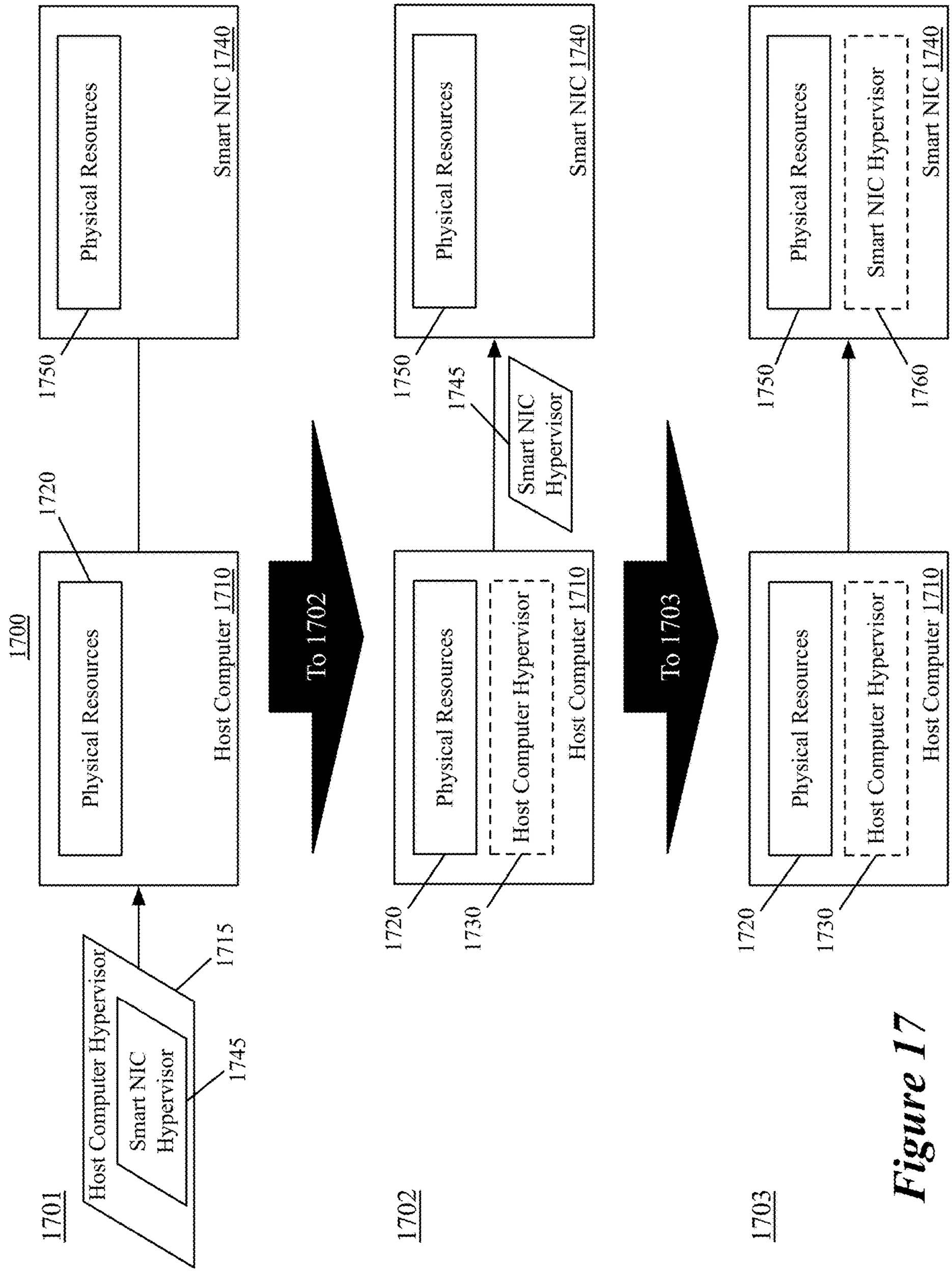

1700
Physical Resources
Host Computer 1710
1720
Smart NIC 1740
1750
1715
Host Computer Hypervisor
Smart NIC Hypervisor
1745
1701
To 1702
1702
1730
1760
To 1703
1703

Figure 17

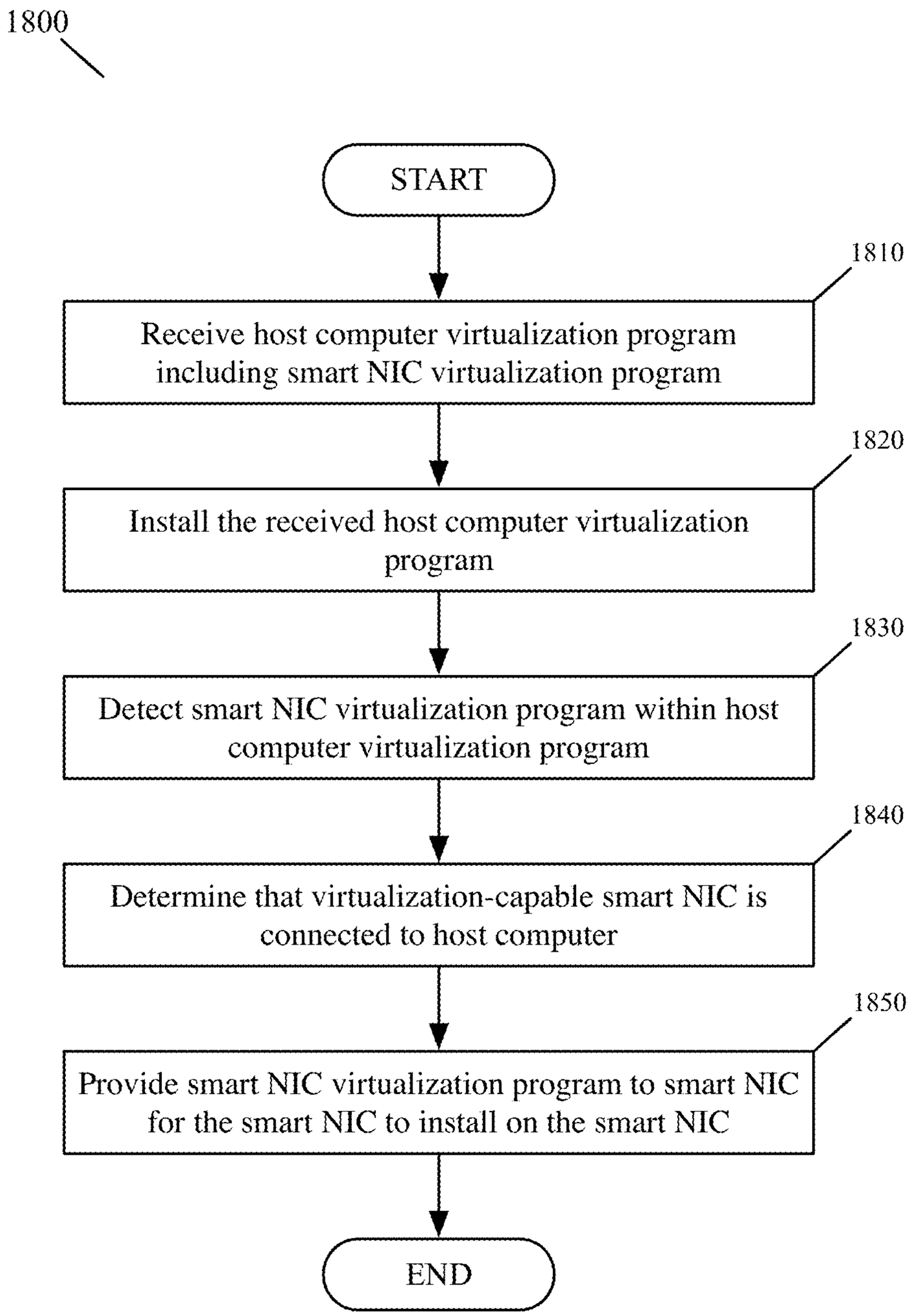

1800
START
1810
Receive host computer virtualization program including smart NIC virtualization program
1820
Install the received host computer virtualization program
1830
Detect smart NIC virtualization program within host computer virtualization program
1840
Determine that virtualization-capable smart NIC is connected to host computer
1850
Provide smart NIC virtualization program to smart NIC for the smart NIC to install on the smart NIC
END

Figure 18

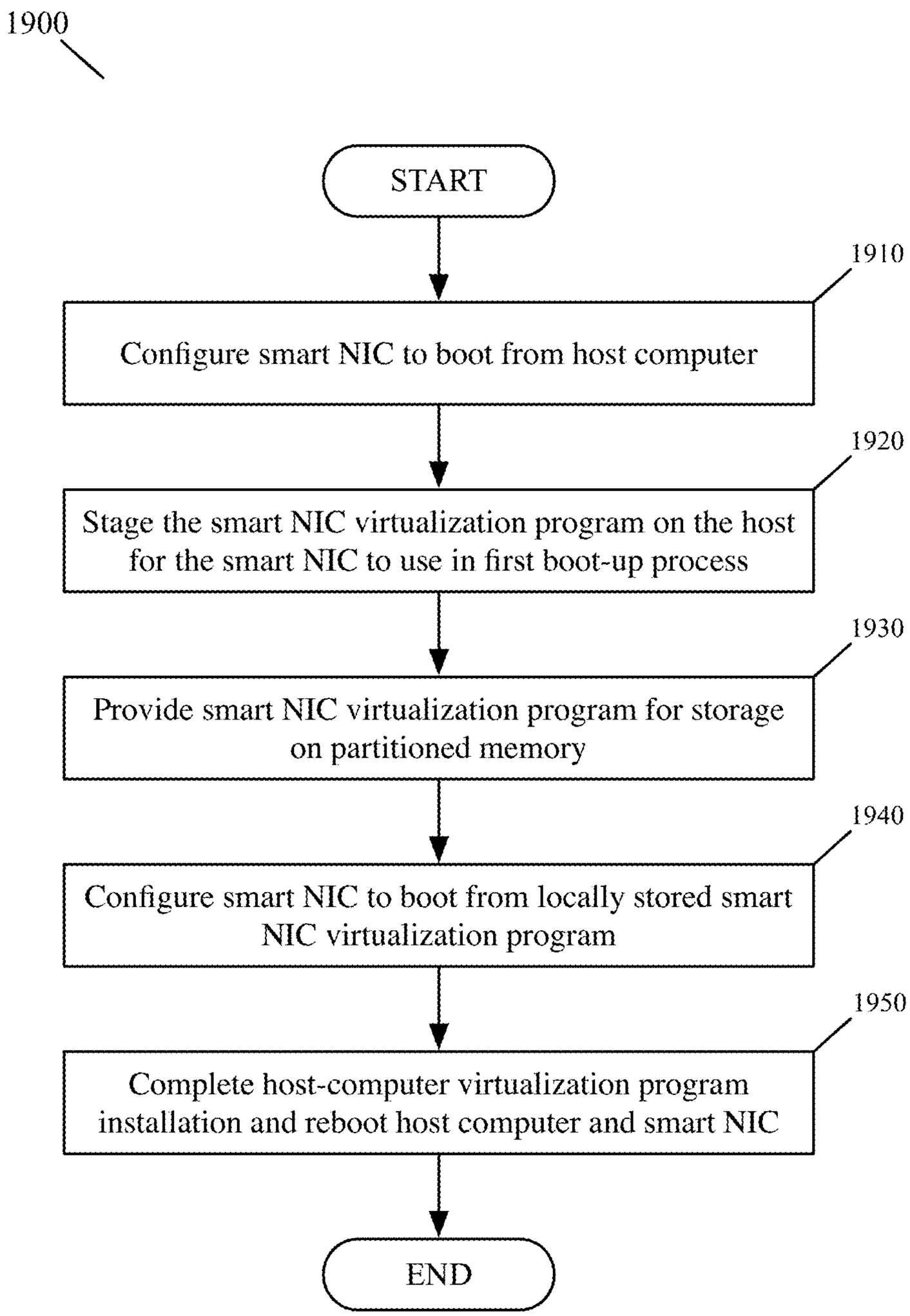

1900
START
1910
Configure smart NIC to boot from host computer
1920
Stage the smart NIC virtualization program on the host for the smart NIC to use in first boot-up process
1930
Provide smart NIC virtualization program for storage on partitioned memory
1940
Configure smart NIC to boot from locally stored smart NIC virtualization program
1950
Complete host-computer virtualization program installation and reboot host computer and smart NIC
END

Figure 19

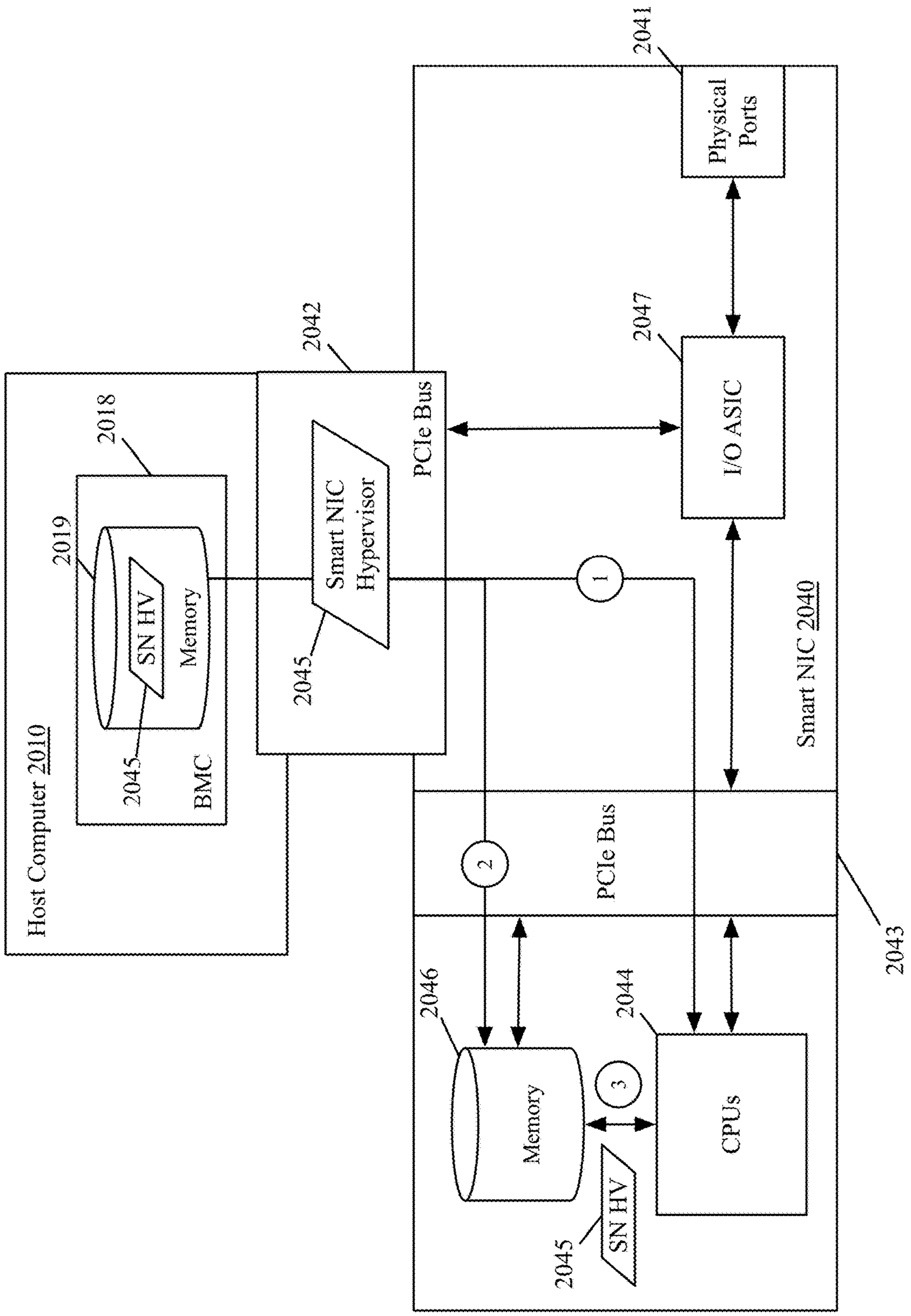

Host Computer 2010
2018
2019
SN HV
Memory
2045
BMC
2042
Smart NIC Hypervisor
2045
PCIe Bus
2041
Physical Ports
2047
I/O ASIC
Smart NIC 2040
2043
1
2
PCIe Bus
2046
Memory
3
2045
SN HV
2044
CPUs

Figure 20

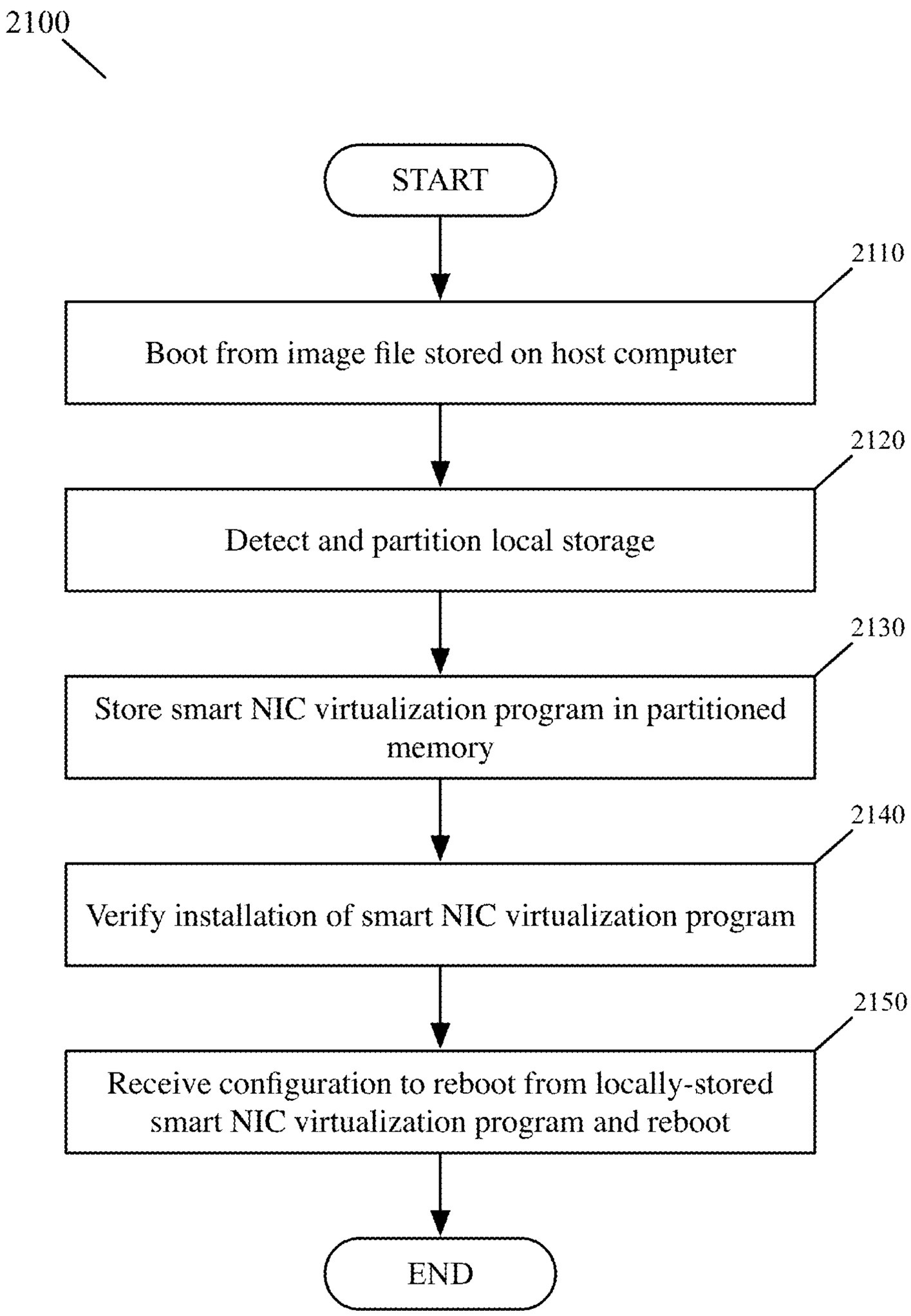

2100
START
2110
Boot from image file stored on host computer
2120
Detect and partition local storage
2130
Store smart NIC virtualization program in partitioned memory
2140
Verify installation of smart NIC virtualization program
2150
Receive configuration to reboot from locally-stored smart NIC virtualization program and reboot
END

Figure 21

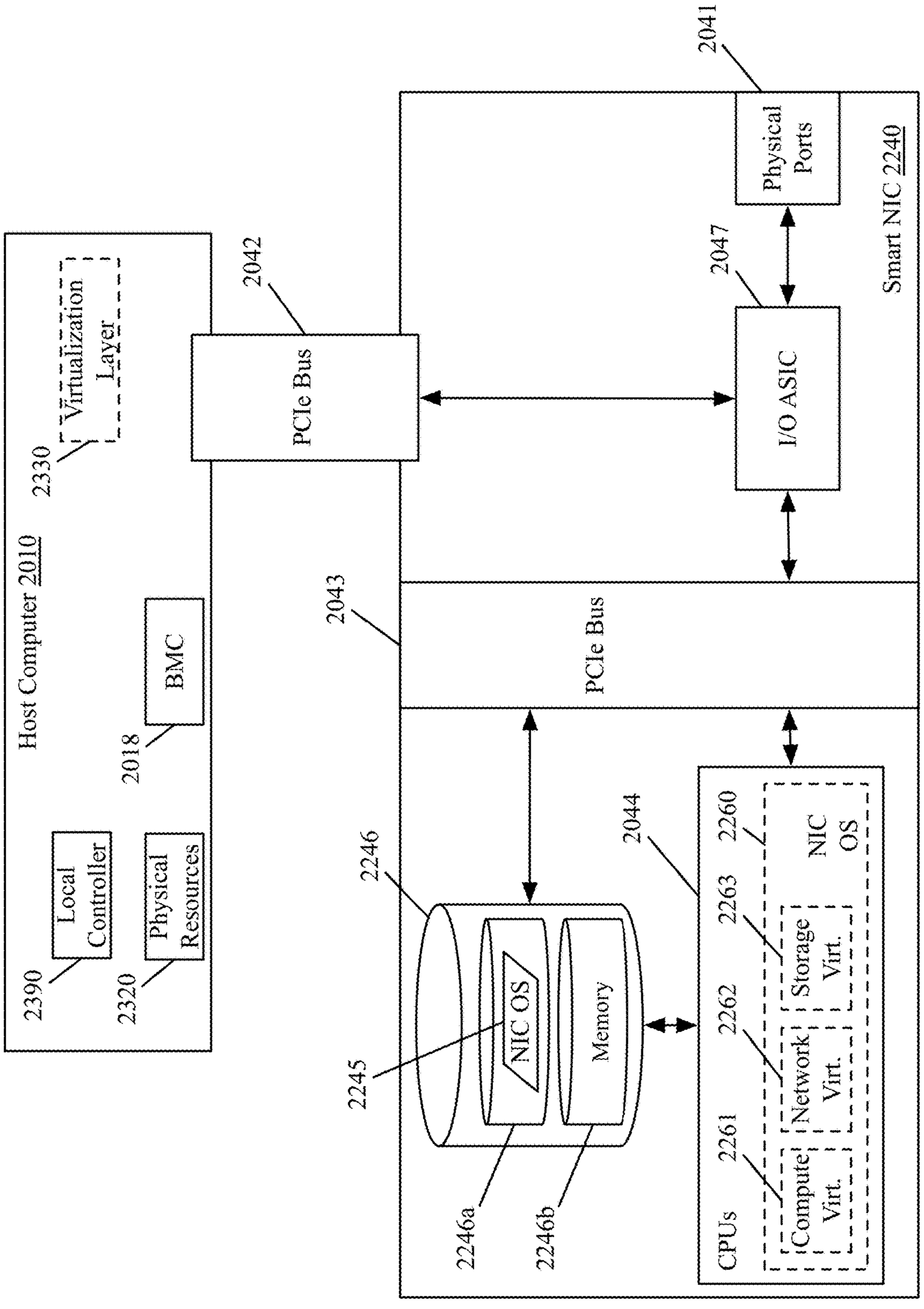

Host Computer 2010
2330
Virtualization Layer
2390
Local Controller
2320
Physical Resources
2018
BMC
2042
PCIe Bus
2043
PCIe Bus
2041
Physical Ports
2047
I/O ASIC
Smart NIC 2240
2246
2044
2245
NIC OS
Memory
2246a
2246b
CPUs
2261
Compute Virt.
2262
Network Virt.
2263
Storage Virt.
2260
NIC OS

Figure 22

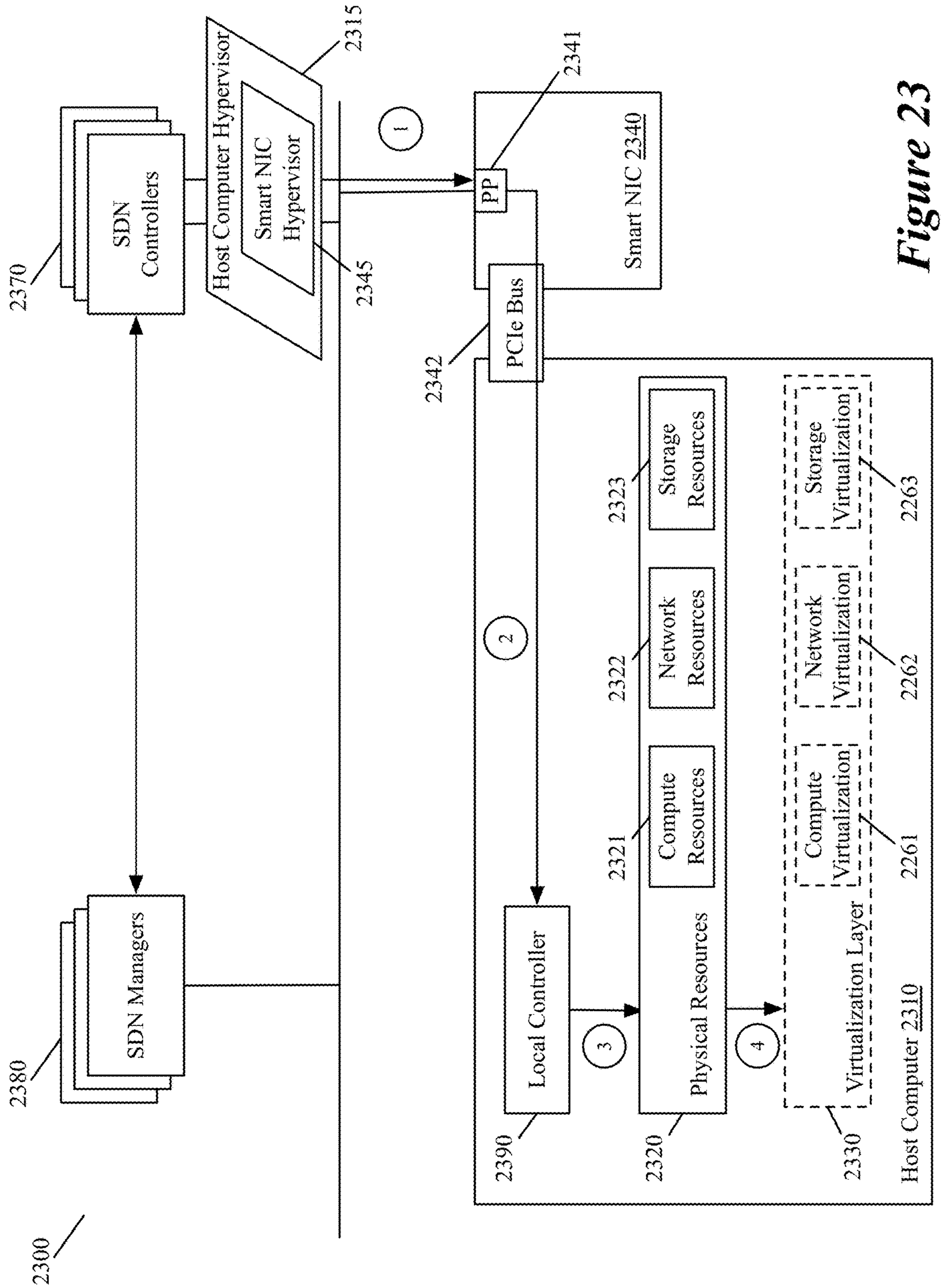

2370
SDN Controllers
Host Computer Hypervisor
Smart NIC Hypervisor
2315
2345
1
2341
PP
Smart NIC 2340
2342
PCIe Bus
2380
SDN Managers
2300
2
2321
Compute Resources
2322
Network Resources
2323
Storage Resources
2261
Compute Virtualization
2262
Network Virtualization
2263
Storage Virtualization
2390
Local Controller
3
2320
Physical Resources
4
2330
Virtualization Layer
Host Computer 2310

Figure 23

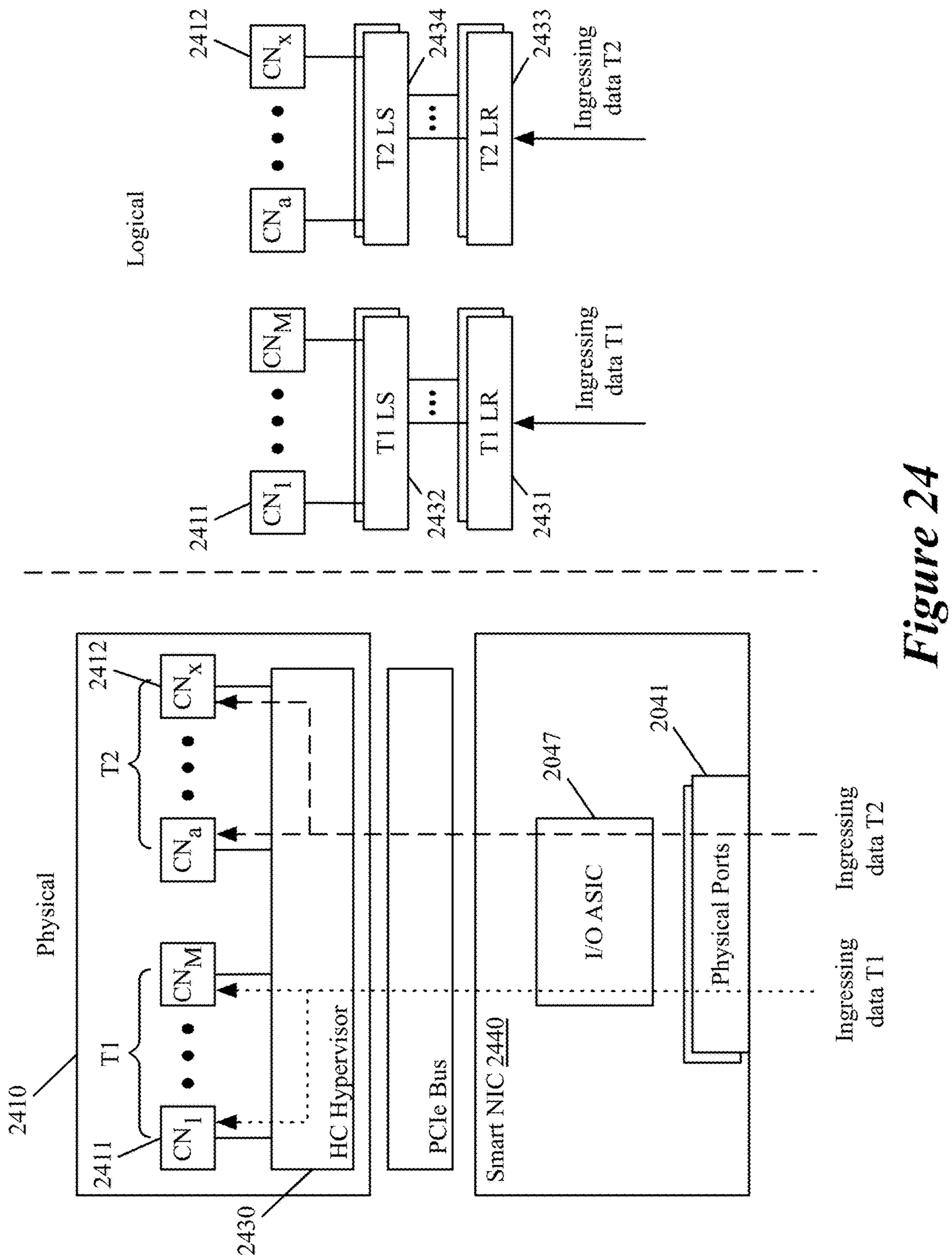

Logical
Physical
2410
2411
2412
T1
T2
CN1
CNa
CNx
CNM
HC Hypervisor
2430
T1 LS
T2 LS
2432
2434
T1 LR
T2 LR
2431
2433
Ingressing data T1
Ingressing data T2
PCIe Bus
Smart NIC 2440
2047
2041
I/O ASIC
Physical Ports
Ingressing data T1
Ingressing data T2

Figure 24

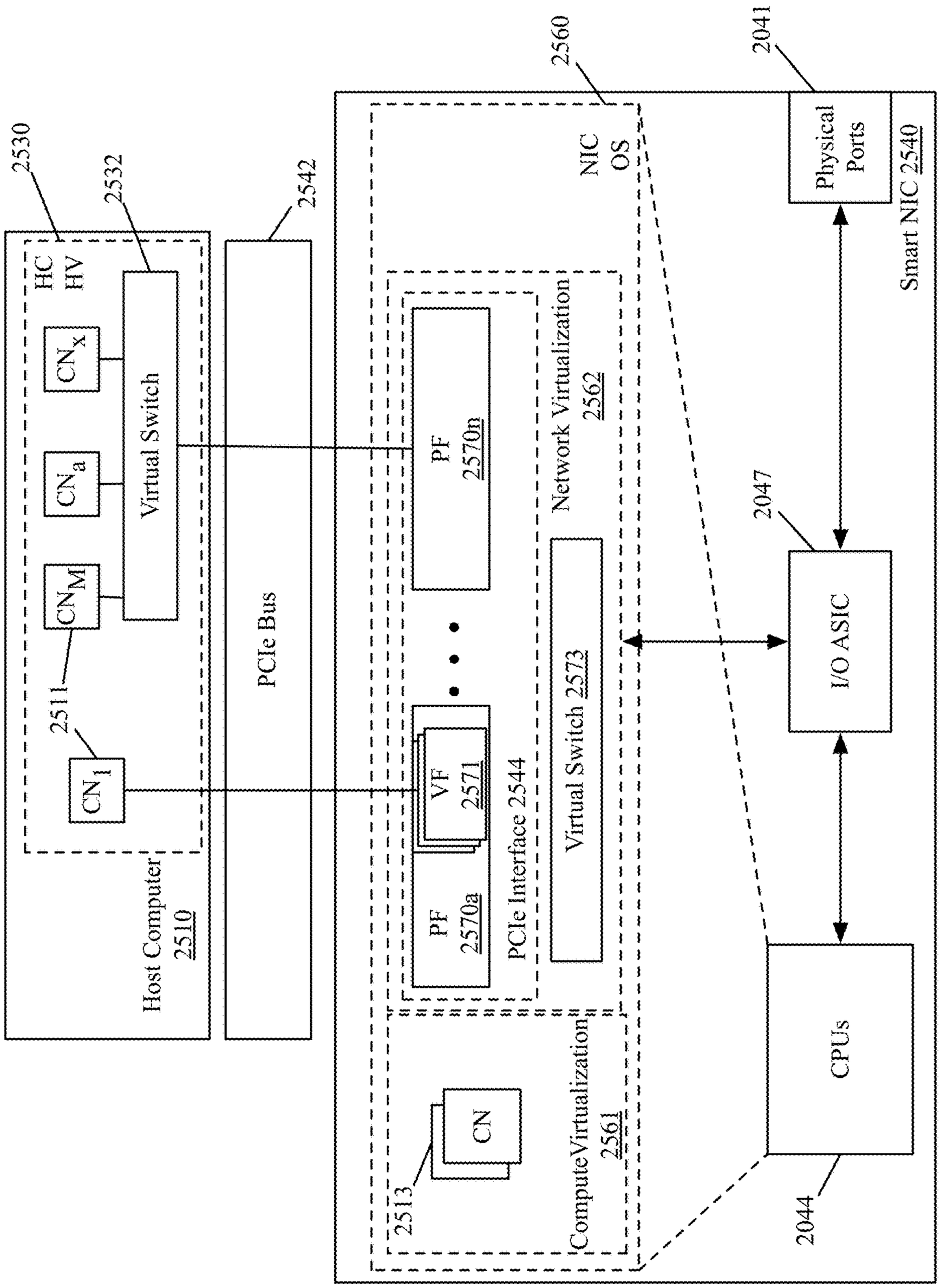

Host Computer 2510
HC HV
2511
CN_1
CN_M
CN_a
CN_x
Virtual Switch
PCIe Bus
2530
2532
2542
2560
2041
NIC OS
Physical Ports
Smart NIC 2540
PF 2570a
VF 2571
PF 2570n
PCIe Interface 2544
Virtual Switch 2573
Network Virtualization 2562
2047
I/O ASIC
2513
CN
ComputeVirtualization 2561
2044
CPUs

Figure 25

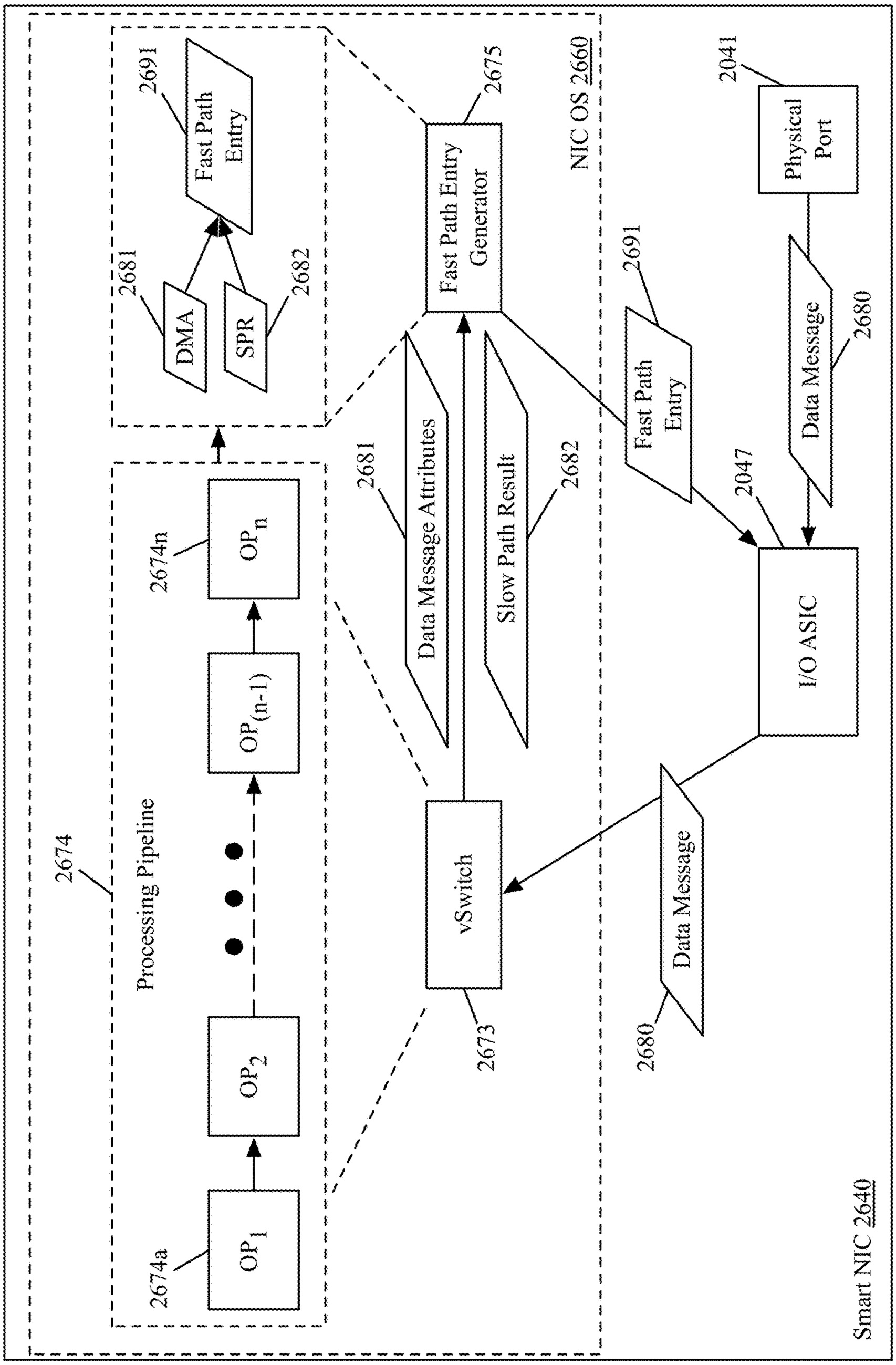

Smart NIC 2640
NIC OS 2660
2674
Processing Pipeline
2674a
OP1
OP2
2674n
OPn
OP(n-1)
2681
2682
2691
Fast Path Entry
DMA
SPR
2675
Fast Path Entry Generator
2673
vSwitch
Data Message Attributes
Slow Path Result
2041
Physical Port
2047
I/O ASIC
Data Message
2680
Fast Path Entry
Data Message

Figure 26

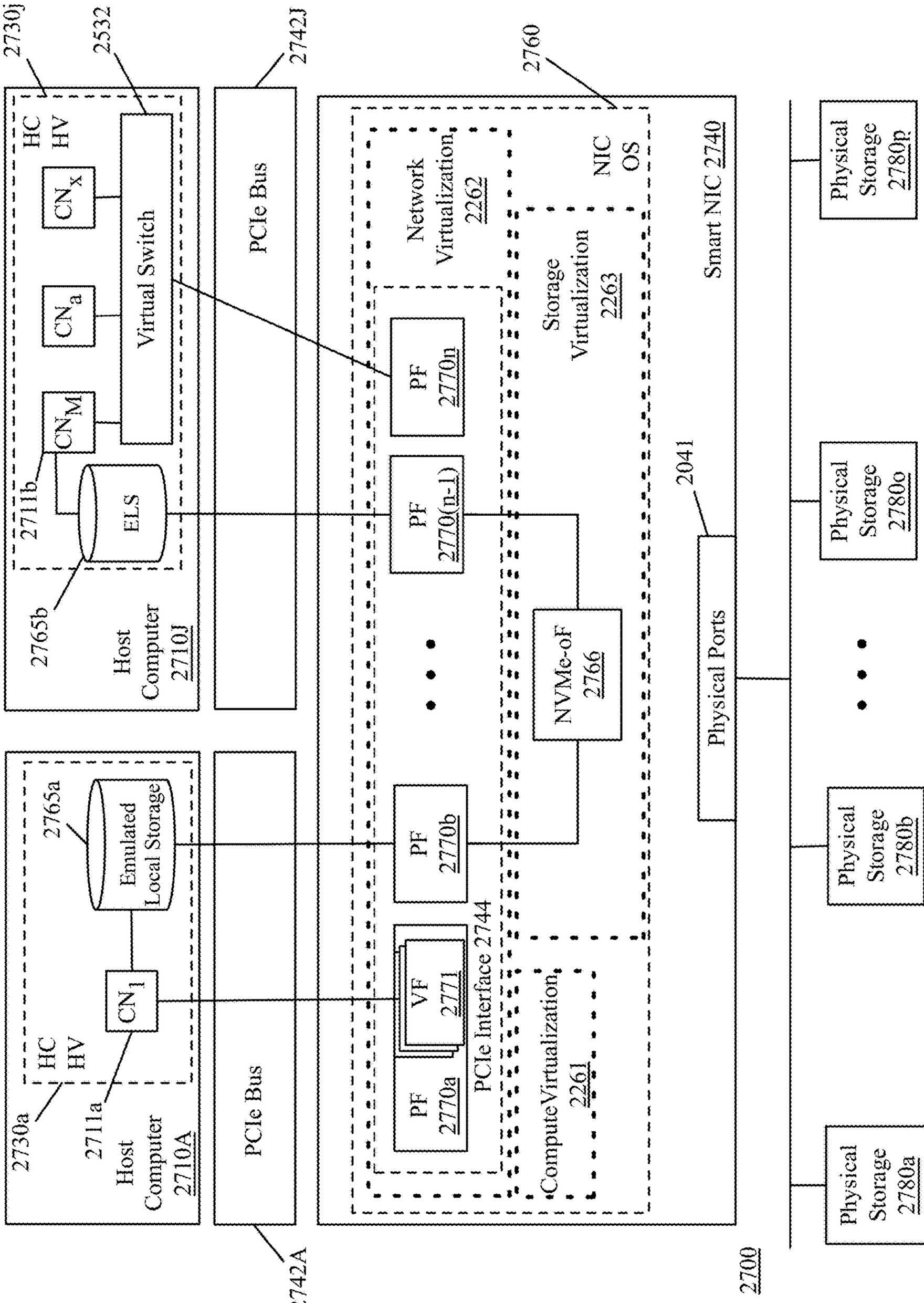

2730j
HC
HV
2532
CNx
CNa
Virtual Switch
2711b
CNM
ELS
2765b
Host Computer 2710J
PCIe Bus
2742J
2760
Network Virtualization 2262
Storage Virtualization 2263
NIC
OS
Smart NIC 2740
PF 2770n
PF 2770(n-1)
PF 2770b
NVMe-oF 2766
Physical Storage 2780p
Physical Storage 2780o
Physical Storage 2780b
2041
Physical Ports
2765a
Emulated Local Storage
2730a
HC
HV
2711a
CN1
Host Computer 2710A
PCIe Bus
2742A
PF 2770a
VF 2771
PCIe Interface 2744
ComputeVirtualization 2261
Physical Storage 2780a
2700

Figure 27

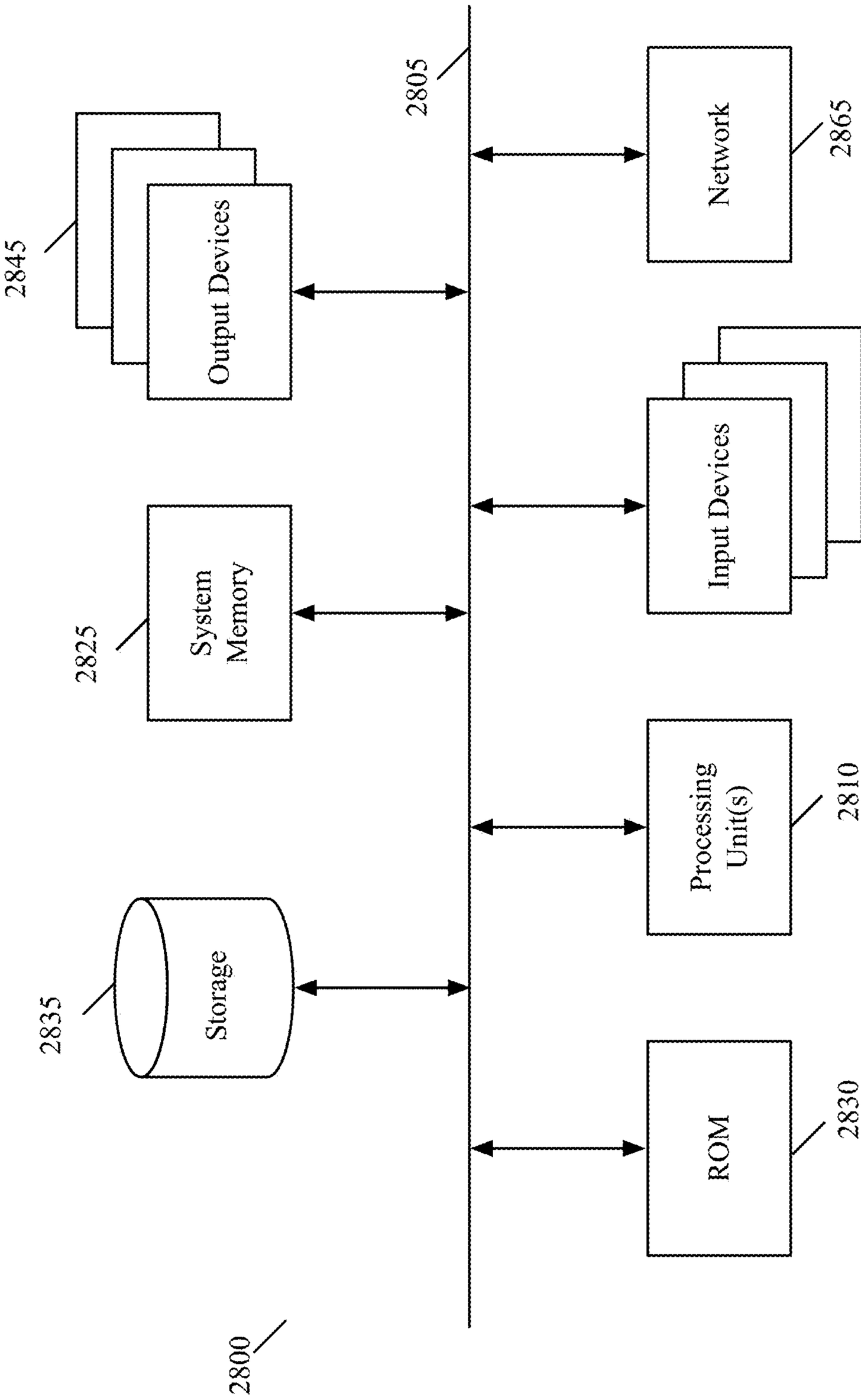

2805
Network
2865
2845
Output Devices
System Memory
2825
Input Devices
2840
Processing Unit(s)
2810
Storage
2835
ROM
2830
2800

Figure 28

DISTRIBUTED STORAGE SERVICES SUPPORTED BY A NIC

BACKGROUND

[0001] In recent years, there has been an increase in the use of hardware offload units to assist functions performed by programs executing on host computers. Examples of such hardware offload units include FGPAs, GPUs, smart NICs, etc. Such hardware offload units have improved performance and efficiency requirements of the host computers by offloading some of the operations that are typically performed by the host computer CPU to the hardware offload unit.

BRIEF SUMMARY

[0002] Some embodiments of the invention provide a method of providing distributed storage services to a host computer from a network interface card (NIC) of the host computer. At the NIC, the method accesses a set of one or more external storages operating outside of the host computer through a shared port of the NIC that is not only used to access the set of external storages but also for forwarding packets not related to the set of external storages or the distributed storage service. In some embodiments, the method accesses the external storage set by using a network fabric storage driver that employs a network fabric storage protocol to access the external storage set.

[0003] The method in some embodiments presents the external storage as a local storage of the host computer to a set of one or more programs executing on the host computer. In some embodiments, the local storage is a virtual disk, while the set of programs are a set of machines (e.g., virtual machines or containers) executing on the host computer. In some embodiments, the method presents the local storage by using a storage emulation layer (e.g., a virtual disk layer) on the NIC to create a local storage construct. In some embodiments, the emulated local storage (e.g., the virtual disk) does not represent any storage on the NIC, while in other embodiments, the emulated local storage also represents one or more storages on the NIC.

[0004] The method forwards read/write (R/W) requests to the set of external storages when receiving R/W requests from the set of programs to the virtual disk, and provides responses to the R/W requests after receiving responses from the set of external storages to the forwarded read/write requests. In some embodiments, the method translates the R/W requests from a first format for the local storage to a second format for the set of external storages before forwarding the requests to the external storage through the network fabric storage driver. The method also translates responses to these requests from the second format to the first format before providing the responses to an NIC interface of the host computer in order to provide these responses to the set of programs.

[0005] In some embodiments, the NIC interface is a PCIe (peripheral component interconnect express) interface, and the first format is an NVMe (non-volatile memory express) format. The second format in some of these embodiments is an NVMeOF (NVME over fabric) format and the network fabric storage driver is an NVMeOF driver. In other embodiments, the second format is a remote DSAN (distributed storage area network) format and the network fabric storage driver is a remote DSAN driver. The NIC in some embodiments includes a general purpose central processing unit (CPU) and a memory that stores a program (e.g., an NIC operating system) for execution by the CPU to access the set of external storages and to present the set of external storages as a local storage. In some embodiments, the NIC also includes an application specific integrated circuit (ASIC), which processes packets forwarded to and from the host computer, with at least a portion of this processing including the translation of the R/W requests and responses to these requests. The ASIC in some embodiments is a hardware offload unit of the NIC.

[0006] In addition to providing an emulation layer that creates and presents an emulated local storage to the set of programs on the host, the method of some embodiments has the NIC execute a DSAN service for the local storage to improve its operation and provide additional features for this storage. One example of a DSAN service is the vSAN service offered by VMware, Inc. The features of the DSAN service in some embodiments include (1) data efficiency processes, such as deduplication operations, compression operations, and thin provisioning, (2) security processes, such as end-to-end encryption, and access control operations, (3) data and life cycle management, such as storage vMotion, snapshot operations, snapshot schedules, cloning, disaster recovery, backup, long term storage, (4) performance optimizing operations, such as QoS policies (e.g., max and/or min I/O regulating policies), and (5) analytic operations, such as collecting performance metrics and usage data for virtual disk (IO, latency, etc.).

[0007] These services are highly advantageous for improving performance, resiliency and security of the host's storage access that is facilitated through the NIC. For instance, the set of host programs that access the emulated local storage do not have insight that data is being accessed on remote storages through network communications. Neither these programs nor other programs executing on the host in some embodiments encrypt their storage access, as the storage being accessed appears to be local to these programs. Hence, it is highly beneficial to use the DSAN services for the R/W requests and responses (e.g., its security processes to encrypt the R/W requests and responses) exchanged between the host and the set of external storages that are made to appear as the local storage.

[0008] The preceding Summary is intended to serve as a brief introduction to some embodiments of the invention. It is not meant to be an introduction or overview of all inventive subject matter disclosed in this document. The Detailed Description that follows and the Drawings that are referred to in the Detailed Description will further describe the embodiments described in the Summary as well as other embodiments. Accordingly, to understand all the embodiments described by this document, a full review of the Summary, Detailed Description, the Drawings and the Claims is needed. Moreover, the claimed subject matters are not to be limited by the illustrative details in the Summary, Detailed Description and the Drawing.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The novel features of the invention are set forth in the appended claims. However, for purpose of explanation, several embodiments of the invention are set forth in the following figures.

[0010] FIG. **1** illustrates one manner of using a smart NIC to emulate a local storage that represents several external storages to a virtual machine executing over a hypervisor of a host computer.

[0011] FIG. **2** illustrates examples of adapters emulated by the smart NIC.

[0012] FIGS. **3** and **4** illustrate two different ways that a DSAN service on a smart NIC serves as a vSAN node in some embodiments.

[0013] FIGS. **5** and **6** illustrate two different ways that the smart NIC of some embodiments uses to translate between the NVMe and NVMeOF storage formats.

[0014] FIG. **7** illustrates a VM that executes on a smart NIC to implement third party interface (protocols) that are needed to access a third party external storage and that are not natively supported by the smart NIC or the host.

[0015] FIG. **8** illustrates a process that some embodiments perform to handle egress communication from the host to a third party external storage.

[0016] FIG. **9** illustrates a process that some embodiments perform to handle ingress communication from the third party external storage to the host.

[0017] FIG. **10** illustrates a smart NIC emulating a local storage using an external storage and a hardware offload unit driver.

[0018] FIG. **11** illustrates a process that the smart NIC OS performs in some embodiments to process an egress communication from the host to an external storage for the example illustrated in FIG. **10**.

[0019] FIG. **12** illustrates a process that the smart NIC OS performs in some embodiments to process an ingress packet from an external storage to the host.

[0020] FIG. **13** illustrate one example of a smart NIC that is used with a host to perform storage emulation.

[0021] FIG. **14** illustrates a process performed to process an egress NVMe command by the smart NIC of FIG. **13**.

[0022] FIG. **15** illustrates another example of a smart NIC that is used with a host to perform storage emulation.

[0023] FIG. **16** illustrates a process that is performed to process egress packets from the VM.

[0024] FIG. **17** illustrates a system including a host computer and a connected smart NIC being configured with a host computer virtualization program and a smart NIC operating system.

[0025] FIG. **18** conceptually illustrates a process for installing programs enabling resource sharing on a host computer and smart NIC.

[0026] FIG. **19** conceptually illustrates a process that, in some embodiments, represents sub-operations of an operation described in relation to FIG. **18**.

[0027] FIG. **20** illustrates a simplified view of a host computer including a baseboard management controller (BMC) and connecting to the smart NIC through a PCIe bus.

[0028] FIG. **21** conceptually illustrates a process that is performed by the smart NIC to install the smart NIC operating system as part of the process described in FIG. **19**.

[0029] FIG. **22** illustrates a smart NIC after the installation is complete with its storage partitioned into a first partition storing the smart NIC operating system and a second partition.

[0030] FIG. **23** illustrates a system that includes the host computer, the smart NIC, a set of SDN controller computers, and a set of SDN manager computers.

[0031] FIG. **24** illustrates a host computer executing a host computer hypervisor and a set of compute nodes (CN_1-CN_M) for a first tenant ("T1") and a set of compute nodes (CN_a-CN_x) for a second tenant ("T2").

[0032] FIG. **25** illustrates a smart NIC providing compute virtualization and network virtualization to provide virtualized resources (e.g., compute nodes, physical functions, and a set of virtual functions) to be used by compute nodes executing on a host computer.

[0033] FIG. **26** illustrates an interaction between an I/O ASIC, a virtual switch, and a fast path entry generator, in some embodiments.

[0034] FIG. **27** illustrates a system including a smart NIC and a set of host computers connected to the smart NIC through two different PCIe buses.

[0035] FIG. **28** conceptually illustrates an electronic system with which some embodiments of the invention are implemented.

DETAILED DESCRIPTION

[0036] In the following detailed description of the invention, numerous details, examples, and embodiments of the invention are set forth and described. However, it will be clear and apparent to one skilled in the art that the invention is not limited to the embodiments set forth and that the invention may be practiced without some of the specific details and examples discussed.

[0037] Some embodiments of the invention provide a method of providing distributed storage services to a host computer from a network interface card (NIC) of the host computer. At the NIC, the method accesses a set of one or more external storages operating outside of the host computer through a shared port of the NIC that is not only used to access the set of external storages but also for forwarding packets not related to the set of external storages or the distributed storage service. The NICs are sometimes referred to herein as smart NICs as they perform multiple types of services and operations. In some embodiments, the method accesses the external storage set by using a network fabric storage driver that employs a network fabric storage protocol (e.g., NVMeOF) to access the external storage set.

[0038] The method presents the external storage as a local storage of the host computer to a set of programs executing on the host computer. In some embodiments, the local storage is a virtual disk, while the set of programs are a set of machines (e.g., virtual machines or containers) executing on the host computer. In some embodiments, the method presents the local storage by using a storage emulation layer (e.g., a virtual disk layer) to create a local storage construct that presents the set of external storages as a local storage of the host computer. In some embodiments, the emulated local storage (e.g., the virtual disk) does not represent any storage on the NIC, while in other embodiments, the emulated local storage also represents one or more storages on the NIC.

[0039] The method forwards read/write (R/W) requests to the set of external storages when receiving R/W requests from the set of programs to the virtual disk, and provides responses to the R/W requests after receiving responses from the set of external storages to the forwarded read/write requests. In some embodiments, the method translates the R/W requests from a first format for the local storage to a second format for the set of external storages before forwarding the requests to the external storage through the network fabric storage driver. The method also translates

responses to these requests from the second format to the first format before providing the responses to a NIC interface of the host computer in order to provide these responses to the set of programs.

[0040] In some embodiments, the NIC interface is a PCIe interface, and the first format is an NVMe format. The second format in some of these embodiments is an NVMeOF format and the network fabric storage driver is an NVMeOF driver. The NIC in some embodiments includes a general purpose central processing unit (CPU) and a memory that stores a program (e.g., an NIC operating system) for execution by the CPU to access the set of external storages and to present the set of external storages as a local storage. The NIC in some embodiments is implemented as a system on chip (SoC) with multiple other circuit components. For instance, in some embodiments, the NIC also includes an application specific integrated circuit (ASIC), which processes packets forwarded to and from the host computer, with at least a portion of this processing including the translation of the R/W requests and responses to these requests. This ASIC in some embodiments is a hardware offload unit (HOU) of the NIC, and performs special operations (e.g., packet processing operations, response/request reformatting operations, etc.).

[0041] In addition to providing an emulation layer that creates and presents an emulated local storage to the set of programs on the host, the method of some embodiments has the NIC execute a distributed storage area network (DSAN) service for the local storage to improve its operation and provide additional features for this storage. One example of a DSAN service is the vSAN service offered by VMware, Inc.

[0042] The DSAN services are highly advantageous for improving performance, resiliency and security of the host's storage access that is facilitated through the NIC. For instance, the set of host programs that accesses the emulated local storage does not have insight that data is being accessed on remote storages through network communications. Neither these programs nor other programs executing on the host in some embodiments encrypt their storage access, as the storage being accessed appears to be local to these programs. Hence, it is highly beneficial to use the DSAN services for the R/W requests and responses (e.g., its security processes to encrypt the R/W requests and responses) exchanged between the host and the set of external storages that are made to appear as the local storage.

[0043] Although the description of some embodiments refers to emulations of NVMe storage and NVMe storage protocol, in other embodiments other storage protocols may be emulated instead of or in addition to NVMe storages. Similarly, although the description refers to PCIe buses, in other embodiments, other system buses are used instead of or in addition to a PCIe bus. Although certain drivers and protocols are shown as being used by external storages in various embodiments, other embodiments use other drivers or protocols for external storage. The smart NICs described herein are described as having operating software. In some embodiments, this operating software is an operating system that has direct control over the smart NIC without an intervening program or hypervisor. In other embodiments, the operating software is a hypervisor that runs on top of another operating system of the smart NIC. Still other embodiments use just a hypervisor and no other operating system on the smart NIC.

[0044] FIG. **1** illustrates one manner of using a smart NIC to emulate a local storage **160** that represents several external storages **140** to one or more virtual machines **112** executing over the operating system (OS) **100** of a host computer. One example of such a machine is illustrated as a virtual machine (VM) **112**, which operates over a hypervisor **114** executing on the host OS **100**. The host computer has a set of processors that execute its OS, hypervisor and VM. This computer also includes a smart NIC that has a set of processors and a set of hardware offload units that assist in the operation of the host computer. Specifically, in addition to performing traditional NIC operations to forward packets to and from the host computer (e.g., between the machines executing on the host computer and machines executing on other host computers), the smart NIC performs storage emulation operations that represent multiple external storages **140** as the local storage **160** to the machines executing on the host computer. The smart NIC connects to PCIe bus **150** of the host.

[0045] The smart NIC in some embodiments is a system on chip (SoC) with a CPU, FPGA, memory, IO controller, a physical NIC, and other hardware components. The smart NIC has an operating system (OS) **120** that includes an NVMe driver **122** and a series of storage processing layers **124-127**. The discussion below collectively refers to the software executing on the smart NIC as the smart NIC OS **120**. However, in some embodiments, the smart NIC OS is a hypervisor, while in other embodiments a hypervisor executes on top of the smart NIC OS and some or all of the storage processing layers are part of this hypervisor. In the discussion below, the components that are attributed to the smart NIC OS **120** are components of the hypervisor **114** that serves as the smart NIC OS or executes on top of the smart NIC OS in some embodiments. In other embodiments, these are components of a smart NIC OS that is not a hypervisor. In still other embodiments, some of these components belong to the smart NIC OS, while other components belong to the hypervisor executing on the smart NIC OS.

[0046] The NVMe driver **122** is a driver for the PCIe bus **150**. This driver relays NVMe formatted R/W requests from the host hypervisor **114** to the storage processing layers, and relays responses to these requests from the storage processing layers to the host hypervisor **114**. The storage processing layers include an NVMeOF driver **124**, a core storage service **125**, a DSAN service **126**, and a virtual device service **127**. The virtual device service includes an NVMe emulator **128**.

[0047] The smart NIC OS **120** uses the NVMeOF driver **124** in some embodiments to access one or more external storages **140**. Specifically, the smart NIC OS **120** emulates a local NVMe storage **160** to represent several external storages **140** to the machines (e.g., VM **112**) executing on the host. From the host point of view, the VM **112** operates on the emulated local storage **160** as if it was a local NVMe storage connected through the PCIe bus **150**.

[0048] To access the external storages **140**, the smart NIC (e.g., the NVMeOF driver) uses one or more of its shared ports **130**. The shared ports are not only used for the purposes of accessing external storage **140**, but are also used for other purposes as well (e.g., used to forward packets to and from destinations other than the external storages). The NVMeOF driver **124** handles the NVMeOF protocols

needed for communicating with the external storages **140** through network fabric (e.g., through routers).

[0049] The smart NICs illustrated in FIG. **1** as well as in other figures perform operations other than storage emulation. For instance, the smart NICs perform regular packet processing in order to forward packets to and from other destinations outside of the smart NIC's host computer that are not external storages. Examples of such other destinations include machines executing on other host computers. However, the illustration presented in FIG. **1** and the other figures focus on the components of the smart NIC that facilitate the storage emulation operations in order not to obscure the description of some embodiments with unnecessary detail.

[0050] The core storage service **125** provides one or more core storage operations. One example of such operations are adapter services that allow the smart NIC to emulate one or more storage adapters, with each adapter logically connecting to one or more external storages **140** and facilitating a different communication mechanism (e.g., transport mechanism) for communicating with the external storages. FIG. **2** illustrates examples of such adapters. In this example, four adapters are illustrated. These include an RDMA storage adapter, a TCP storage adapter, an iSCSI adapter, and an iSER adapter.

[0051] Through this interface, an administrator in some embodiments can specify one or more adapters to use to access an external storage, or a set of two or more external storages. In some embodiments, more than one adapter is specified for an external storage when the administrator wants to specify a multipath pluggable storage architecture (PSA) approach to accessing the storage. Once the administrator specifies an adapter, a network manager that provides the interface sends the definition of the specified adapter to a network controller, which then configures the smart NIC to implement and configure a new driver, or reconfigure an existing driver, to access the external storage according to the adapter's specified definition. Different methods for configuring a smart NIC in some embodiments are described below.

[0052] The DSAN service **126** provides one or more DSAN operations to improve the operation of the emulated local storage **160** and provide additional features for this storage. These operations are performed as the emulated local storage is not really local but rather an emulation of one or more external storages. As such, the DSAN service **126** addresses one or more things that can go wrong in accessing such a virtual "local" storage.

[0053] For instance, in some embodiments, the DSAN service provides data resiliency and I/O control that are not generally needed when a host machine is accessing a physical local storage over NVMe. A local drive is not subject to interception over a network and is not prone to packet duplication in the manner of packets sent over a network. These issues arise from emulating the local storage using external storage accessed over a network, therefore the DSAN layer **126** resolves such issues before the data is presented to the higher layers.

[0054] In some embodiments, the DSAN operations include (1) data efficiency processes, such as deduplication operations, compression operations, and thin provisioning. (2) security processes, such as end-to-end encryption, and access control operations, (3) data and life cycle management, such as storage vMotion, snapshot operations, snapshot schedules, cloning, disaster recovery, backup, long term storage, (4) performance optimizing operations, such as QoS policies (e.g., max and/or min I/O regulating policies), and (5) analytic operations, such as collecting performance metrics and usage data for virtual disk (10, latency, etc.).

[0055] One example of a DSAN service **126** is the vSAN service offered by VMware, Inc. In some such embodiments, the smart NIC includes a local physical storage that can serve as a vSAN storage node. In other embodiments, the smart NIC does not have a local physical storage, or has such a storage but this data storage cannot participate as a vSAN storage node. In such embodiments, the smart NIC serves as a remote vSAN client node, and its vSAN layer is a vSAN proxy that uses one or more remote vSAN nodes that perform some or all of the vSAN operations and then direct the vSAN proxy what to do.

[0056] FIG. **3** illustrates such an approach. As shown, the vSAN proxy **326** uses a remote vSAN client protocol to communicate with the other vSAN nodes **305**, which direct the vSAN operations of the vSAN proxy. The vSAN nodes **305** provide some or all of the external storages **140** in some embodiments of the invention. In this example, the network storage driver **324** is an iSCSI driver, although other network storage drivers are used in other embodiments.

[0057] In other embodiments, the DSAN service of the smart NIC does not use a remote vSAN client protocol to communicate with the other vSAN nodes. For instance, as shown in FIG. **4**, a DSAN service **126** in some embodiments uses a vSAN over NVMeOF protocol **426** to communicate with the other vSAN nodes. This protocol is defined in some embodiments to allow the smart NIC to be a vSAN node that does not have a local physical storage, or has such a storage but this data storage cannot participate as a vSAN storage node. In some embodiments, the emulated local storage **160** (that is defined by one or more external storages **140** through emulation operations of the NVMe emulator **128** of the virtual device service **127** of the smart NIC OS **120**) serves as the local storage that allows the smart NIC to be a vSAN node.

[0058] The virtual device service **127** has an NVMe emulator **128** that emulates the local NVMe storage **160** to represent the set of external storages **140** that are accessed through the NVMeOF driver **124** and the intervening network. As part of this emulation, the virtual device layer **127** maps outgoing NVMe access commands to external storage access commands, and the incoming external storage responses to an NVMe memory response. When multiple external storages are used, this mapping involves mapping between a storage location in the emulated local storage **160** and a storage location in one or more external storages **140**. One example of a virtual device emulator that can be used for the NVMe emulator is the virtual device emulator of the vSphere software of VMware, Inc.

[0059] Part of the NVMe emulator's operation also involves this emulator using the hardware offload unit (e.g., an ASIC) of the smart NIC to convert the NVMe access commands from an NVMe-PCIe format to an NVMe format, and to convert the external storage responses received at the emulator **128** from the NVMe format to an NVMe-PCIe format (e.g., to remove PCIe header information from outgoing commands, and to add PCIe header information to incoming responses). This is further described below by reference to FIGS. **5** and **6**.

[0060] The host OS **100**, the hypervisor **114** or the VM **112** in some embodiments have their own drivers (not shown) for sending and receiving data through the PCIe bus **150**. The host OS **100**, the hypervisor **114** or the VM **112** treats the virtual local storage **160** as a physical local storage, without having to deal with the operations that the smart NIC performs to send data to and receive data from the set of external storages **140**.

[0061] DSAN services **126** (such as the remote DSAN client of FIG. **3** or the vSAN over NVMeOF of FIG. **4**) are two ways of offering disaggregated storage services. Today, many DSANs (e.g., VMware's vSAN architecture) are part of a hyper-converged solution, in which each vSAN node offers both storage and compute functionality. As illustrated by FIGS. **3** and **4**, disaggregated storage in some embodiments refers to storage in a system which has some DSAN nodes (e.g., some hardware boxes) that provide only compute functionality and no storage functionality. In some embodiments, one or more DSAN nodes only offer storage functionality and no compute functionality. Such a disaggregated system allows more flexibility in datacenters by allowing the operators of the datacenters to add more storage boxes than compute boxes or more compute boxes than storage boxes, whichever is necessary, rather than adding additional compute boxes with storages whenever additional capacity of only one of those resources is necessary.

[0062] FIGS. **5** and **6** illustrate two different ways that the smart NIC of some embodiments uses to translate between the NVMe and NVMe-PCIe formats (e.g., to remove PCIe header from outgoing storage access commands and to add PCIe header information to incoming storage responses). Both of these techniques use a hardware offload unit (HOU) **505** of the smart NIC to perform these operations. This HOU is an ASIC that has multiple packet processing stages that can be configured to remove or add PCIe headers to storage commands and responses to and from the external storages. In both approaches illustrated in FIGS. **5** and **6**, the NVMe emulator **128** uses an HOU interface **520** to communicate with the HOU **505**.

[0063] In FIG. **5**, the HOU interface executes on a VM **510** that executes on the smart NIC. The smart NIC OS is a hypervisor and the VM **510** executes on top of this hypervisor in some embodiments. As shown, the NVMe emulator **528** of the virtual device layer **527** communicates with the HOU interface **520** to forward storage access commands and responses for processing by the HOU **505** and to receive processed commands and responses from the HOU **505**. In other embodiments, the smart NIC executes the HOU interface on machines (e.g., Pods or containers) other than VMs. One example of an HOU interface and the HOU are the Snap software and hardware offered by Nvidia, Inc. In some embodiments, the HOU Snap software operates on the VM as it requires a different OS (e.g., require Ubuntu) than the smart NIC OS (which might be ESX offered by VMware, Inc.).

[0064] In some embodiments, a smart NIC is able to employ HOU drivers that are adapted to the smart NIC OS (e.g., HOU drivers supplied along with the smart NIC operating software or subsequently downloaded, etc.) as the interface with the smart NIC HOU. The HOU drivers that are adapted to run directly on a particular type of operating software are referred to as being "native" to that operating software. In FIG. **6**, the HOU interface **520** is implemented as a native HOU driver **610** of the smart NIC. This approach works when the driver is available natively for the smart NIC OS. Otherwise, the driver has to operate in a VM **510** as in FIG. **5**.

[0065] More generally, a VM is used by the smart NIC of some embodiments to perform other processes and/or support other protocols that are not natively supported by the smart NIC in some embodiments. For instance, FIG. **7** illustrates a VM **710** that executes on a smart NIC OS **700** to implement third party interface **725** (e.g., third party storage protocol) that is needed to access a third party external storage **712** and that is not natively provided by the smart NIC OS or the host OS. In this example, the third party storage interface **725** is part of an interface **520** for a HOU **715** of the smartNIC.

[0066] At the direction of the HOU interface **520** (also called the HOU handler), the HOU **715** performs storage command and response processing operations needed to implement the third party storage protocol and to convert between the command and response formats of the host's local storage (e.g., its NVMe local storage) and the third party external storage **712**. As shown, the third party storage interface **725** passes storage access commands and receives storage access responses from a shared port **720** of the NIC.

[0067] FIG. **8** illustrates a process **800** that some embodiments perform to handle egress communication from the host to a third party external storage. As shown, the process **800** starts (at **805**) when a workload VM or an application running on the host generates an NVMe command (with data). At **810**, the NVMe command is then encapsulated into a PCI-NVME command (i.e., encapsulated with a PCIe header) at a local storage controller of the host computer, and is forwarded along the PCIe bus to the smart NIC **700**. At the smart NIC **700**, the PCI-NVMe command is passed (at **815**) to the HOU handler **520** running inside of the VM **710**.

[0068] Next, at **820**, the third party storage interface **725** strips off the PCI Headers and passes NVMe command back to the HOU handler **520**. To do this, the third party storage interface **725** uses the HOU in some embodiments. The HOU handler next uses (at **825**) the smart NIC HOU to change the format of the NVMe command to a command that comports with the third party storage **712**, and passes (at **830**) this command to the third party storage **712** along a shared port of the smart NIC. In some embodiments, the command is passed to the third party storage **712** as one or more packets transmitted through the network fabric.

[0069] FIG. **9** illustrates a process **900** that some embodiments perform to handle ingress communication from the third party external storage to the host. As shown, the process **900** starts (at **905**) when it gets a storage-access response (e.g., a Read response) through a shared port of the NIC from the third party external storage **712**. At **910**, the smart NIC OS determines that the storage-access response is from a third party external storage that needs to be processed by the third party storage interface **725** of the HOU handler **520**.

[0070] At **915**, the HOU Interface **520** gets the storage-access response and provides it to the third party storage interface **725**, which then converts (at **920**) the storage-access response from a third party format to an NVMe format and passes the storage-access response back to the HOU interface **520**. Next, at **925**, the HOU interface encapsulates the NVMe storage-access response with a PCIe header, and is passed to the host's local storage controller

along the PCIe bus **150**. The local storage controller then removes (at **930**) the PCIe header, and provides the NVMe storage-access response to a workload VM or an application running on the host.

[0071] As described with respect to FIG. **1**, the smart NICs of some embodiments provide a DSAN service to perform various security and efficiency operations for the virtual local storage that is emulated with one or more external storages. However, the smart NIC in other embodiments bypasses the DSAN layer that performs DSAN operations in order to increase the speed of data transfer. Instead of the HOU drivers described above, some such embodiments use other protocols. For example, some embodiments use HOU upper level protocols (ULP). Upper level protocols (e.g., IPoIB, SRP, SDP, iSER, etc.) facilitate standard data networking, storage and file system applications to operate over InfiniBand.

[0072] FIG. **10** illustrates a smart NIC emulating a local storage for the VMs **112** of a host **1000** by using an external storage **1040** and a HOU driver **1022**. Like the above-described HOU interfaces and drivers, the HOU driver **1022** forwards data messages for processing by the smart NIC HOU **505** and receives processed data messages from the HOU **505**. Specifically, the HOU driver **1022** uses the HOU **505** to perform the packet processing needed to convert between the data message NVMe formats and the NVMe PCIe formats. In this example, the HOU driver **1022** exchanges data messages with a kernel NVMe layer **1028**, which exchanges data messages with an NVMe RDMA driver **1024** and/or an NVMe TCP driver **1026**. The NVMe RDMA and TCP drivers send and receive data messages to and from external storage **1040** through an intervening network fabric (e.g., intervening routers and switches).

[0073] One advantage of the approach of FIG. **10** is that the smart NIC **1020** transfers data quickly between the host and the external storage **1040** that is used to emulate the host's local storage **1020**. This transfer is fast because it uses the kernel NVMe **1028** as a bridge and it does not use a DSAN layer **1030** on the smart NIC OS **1020**. This embodiment can tap into NVMe RDMA offload capability by using the NVMe RDMA driver **1024**. In some embodiments, the HOU of the smart NIC **1020** can strip Ethernet headers from an incoming data packet, identify the particular NVMe PCIe controller (here, the HOU ULP driver **1022**) that needs to receive the packet, and pass the packet to that NVMe PCIe controller. Thus, the smart NIC CPU cost of bridging through the kernel NVMe layer **1028** is minimal. This speed comes at a cost of other features, such as bypassing the DSAN service **1030** which provides useful security and performance operations for the emulated local storage.

[0074] In the example of FIG. **10**, the DSAN module **1056**, the virtual device emulator **1057** and the multipath PSA service **1055** are provided for one or more VMs **112** through the host hypervisor **114**. Specifically, in this example, a multipath PSA layer **1055** exists between the VMs **112** executing on the host OS **1000** and the NVMe PCIe driver **1060** of the OS. Through this PSA layer **1055**, the host can use multiple paths to the same external storage by using different NVMe PCIe drivers executing on the host OS **1000** (although only one NVMe PCIe driver **1060** is shown in FIG. **10**). In other words, for the multi-pathing, different PCIe drivers are also used in some embodiments to access the same external storage through different paths. Also, in some embodiments, the different NVMe PCIe drivers are used to emulate different local storages from different external storages **1040**.

[0075] The virtual device emulator **1057** is used to emulate a local virtual disk from several external storages **1040** for one or more VMs **112**. As mentioned above, the vSphere software's virtual device layer is used to implement the virtual device emulator of the host hypervisor or smart NIC hypervisor in some embodiments. In some embodiments, the same or different PCIe drivers **1060** are used to access different external storages **1040** that are used to emulate one virtual disk. The DSAN module **1056** performs DSAN services like those described above for the emulated local storages.

[0076] In some embodiments, the host hypervisor and smart NIC hypervisor can be configured to provide different storage services for different workload VMs **112**. For instance, the storage access commands and responses for one workload VM is processed by the storage services **1055-57**, while the storage access commands and responses for another workload VM skip these storage services. Similarly, the storage access commands and responses of one workload VM is processed by the storage services **125-127** of the smart NIC as shown in FIG. **1**, while the storage access commands and responses of another workload VM are just processed by the kernel NVMe module **1028** and NVMeOF drivers **1024** and **1026** of FIG. **10**.

[0077] FIG. **11** illustrates a process **1100** that the smart NIC OS **1020** performs in some embodiments to process an egress communication from the host to an external storage for the example illustrated in FIG. **10**. As shown, the process starts (at **1105**) when an NVMe command (with data) is generated by a VM **112** on the host. This packet (at **1110**) is encapsulated with PCIe header information to produce a PCIe-NVMe command (with data) at a local storage controller (not shown) of the host, and is passed along to the PCIe bus **150**. Next, at **1115**, the HOU driver **1022** (e.g., HOU ULP driver) receives this command through the PCIe bus **150**, and uses the HOU to strip out the PCI headers and produce the NVMe command (with data).

[0078] At **1120**, the HOU driver **1022** passes the NVMe command to the kernel NVMe module **1028**, which maps this packet to an NVMeOF transport controller. The kernel NVMe module **1028** in some embodiments is transport agnostic, and can be configured to use any one of a number of different NVMe transport drivers. At **1120**, the kernel NVMe **1028** identifies the NVMeOF controller (i.e., NVMe RDMA controller **1024** or NVMe TCP controller **1026**) that needs to receive this NVMe command. This identification is based on the NVMe command parameters that identify the transport protocol to use. These command parameters are provided by the host's multipath PSA layer **1055**.

[0079] The kernel module (at **1125**) passes the NVMe command to the identified NVMeOF controller, which then generates one or more NVMeOF packets to forward (at **1130**) the NVMe command to the destination external storage through a shared port of the smart NIC. As mentioned above, both NVMe RDMA **1024** and NVMe TCP **1026** are provided by the smart NIC OS **1020** for accessing remote external storages **1040** through the shared port(s) **130** of the smart NIC. In some embodiments, the kernel NVMe **1028** works like a multiplexer that provides NVMe storage access to the HOU driver **1022** using different transports, such as NVMe RDMA **1024** and NVMe TCP **1026**, at the same time. After **1130**, the process **1100** ends.

[0080] FIG. **12** illustrates a process **1200** that the smart NIC OS **1020** performs in some embodiments to process an ingress packet from an external storage to the host. As shown, the process starts (at **1205**) when an external storage **1040** generates and forwards an NVMeOF command (with data) that is received as a set of one or more network packets at a shared port of the smart NIC through network fabric (e.g., through one or more switches and/or routers). The port (at **1210**) passes the received packet to the NVMe RDMA controller **1024** or NVMe TCP controller **1026** depending on the transport protocol used by the external storage. The NVMe controller (at **1215**) receives the NVMeOF packet in its transport specific format, removes the transport header data, and provides an NVMe command (with data) to the kernel NVMe **1028**.

[0081] At **1220**, the kernel NVMe **1028** maps the received NVMe command to the HOU driver **1022** as the NVMe command needs to go to host. In some embodiments, the kernel NVMe **1028** creates a record when it was processing an egress packet at **1125** and uses this record to perform its mapping at **1220**. In some embodiments, the kernel NVMe **1028** provides the NVMe command to the HOU driver **1022** with the controller of the emulated local storage **160** as the command's destination. At **1225**, the HOU driver **1022** then encapsulates the NVMe command with a PCIe header by using the smart NIC's HOU and then sends the NVMe command along the host PCIe to the local storage controller of the emulated local storage **160**. The host PCIe then provides (at **1230**) the NVMe command to the local storage controller through the NVMe PCIe driver **1060**. This controller then removes (at **1230**) the PCIe header and provides the NVMe command to the destination VM **112**. The process **1200** then ends.

[0082] In some embodiments, the smart NICs are used as storage access accelerators. FIGS. **13** and **15** illustrate two such examples. FIG. **13** illustrates how a smart NIC serves as a network accelerator to one or more workload VMs **1312** executing over a host hypervisor **1314** that operates over a host OS **1300**. In this example, the remote storage services protocol is running inside the host and the smart NIC OS **1320** just runs network accelerators. Also, the host hypervisor **1314** provides emulation services in this example that allow it to present one or more external storages **1340** as a local storage to a VM **1312**. In some embodiments, the hypervisor **1314** is the ESX hypervisor of VMware, Inc. In some such embodiments, a virtual NVMe device emulation module **1311** of the VMware vSphere software provides the NVMe device emulation that presents multiple external storages **1340** as a single local NVMe storage to the VM **1312**.

[0083] In some embodiments, the hypervisor **1314** also includes the DSAN service layer **1313**, which provide distributed storage services for the emulated local NVMe storage. As mentioned above, the distributed storage services in some embodiments account for the VM **1312** having no knowledge regarding the plurality of external storages being used to emulate the local storage. These DSAN service improve this emulated storage's operation and provide additional features for it. Examples of such features in some embodiments include (1) data efficiency processes, such as deduplication operations, compression operations, and thin provisioning, (2) security processes, such as end-to-end encryption, and access control operations, (3) data and life cycle management, such as storage vMotion, snapshot operations, snapshot schedules, cloning, disaster recovery, backup, long term storage, (4) performance optimizing operations, such as QoS policies (e.g., max and/or min I/O regulating policies), and (5) analytic operations, such as collecting performance metrics and usage data for virtual disk (10, latency, etc.) One example of a DSAN service is the vSAN service offered by VMware vSphere software. The DSAN service layer **1313** also includes a multipathing PSA layer in some embodiments.

[0084] The DSAN service module **1313** receives and sends storage related NVMe commands from and to the kernel NVMe module **1315**. The kernel NVMe module **1315** interacts with either the NVMe RDMA driver **1316** or NVMe TCP driver **1317** to receive and send these NVMe commands. These drivers exchange these NVMe commands with the smart NIC OS **1320** through one or more virtual functions (VFs) **1322** defined for these drivers on the smart NIC OS.

[0085] In some embodiments, the smart NIC OS can present the smart NIC as multiple physical functions (PF) connected to the host computer. The PCIe bus **150**, in some embodiments, allows for the creation of these PFs. A PF, in some embodiments, can be further virtualized as multiple virtual functions (VFs). More specifically, in some embodiments, physical functions and virtual functions refer to ports exposed by a smart NIC using a PCIe interface to connect to the host computer over the PCIe bus. A PF refers to an interface of the smart NIC that is recognized as a unique resource with a separately configurable PCIe interface (e.g., separate from other PFs on a same smart NIC). In some embodiments, each PF is executed by the processing units (e.g., microprocessors) of the host computer.

[0086] The VF refers to a virtualized interface that is not fully configurable as a separate PCIe resource, but instead inherits some configuration from the PF with which it is associated while presenting a simplified configuration space. VFs are provided, in some embodiments, to provide a passthrough mechanism that allows compute nodes executing on a host computer to receive data messages from the smart NIC without traversing a virtual switch of the host computer. The VFs, in some embodiments, are provided by virtualization software executing on the smart NIC. In some embodiments, each VF is executed by the processing units (e.g., microprocessors) of the smart NIC.

[0087] The VFs and PFs, in some embodiments, are deployed to support storage and compute virtualization modules. For example, a PF or VF can be deployed to present a storage or compute resource provided by the smart NIC as a local device (i.e., a device connected to the host computer by a PCIe bus). Defining such VFs are further described below.

[0088] The PF **1370** on the host has the corresponding VF **1322** on the smart NIC. The PF **1370** represents a shared NIC port to the NVMeOF drivers **1316** and **1317**, which run on the host and convert the NVMe storage access commands to network packets. These drivers use this representative port **1370** to forward storage access packets to an external storage through the VF **1322** of the smart NIC **1320**, and to receive storage access response packets from the external storage **1340** through the VF **1322** of the smart NIC **1320**.

[0089] When the VF **1322** does not know how to process a packet (e.g., when it receives a first packet of a new flow for which it does not have a forwarding rule), the VF passes the packet through a "slow-path" that includes the virtual

switch **1326** of the virtualization layer **1327**, which then determines how to forward the packet and provides the VF with forwarding rule for forwarding the packet. On the other hand, when the VF **1322** knows how to process a packet (e.g., when the VF receives another packet of a flow that it has previously processed and/or for which it has a forwarding rule), the VF passes the packet through a "fast-path," e.g., passes a packet of a previously processed flow directly to the NIC driver **1325** for forwarding to an external storage **1340**. Accordingly, in the example illustrated in FIG. **13**, the VF **1322** is a network accelerator that facilitates the forwarding of the packets related to the external storages.

[0090] In some embodiments, the VF **1322** uses the smart NIC HOU **505** to perform its fast path forwarding. When the HOU is not programmed with flow-processing rules needed to process a new flow, the VF **1322** in some embodiments passes the packet to the virtualization layer **1327**, which either identifies the flow-processing rule for a rule cache or passes the packet to a manager (executing on the smart NIC or on an external computer) that then determines the flow processing rule, and passes this rule back to the virtualization layer to use to forward the packet and to program the HOU. Once programmed, the VF can use the HOU to process subsequent packets of this flow.

[0091] FIG. **14** illustrates a process **1400** performed to process an egress NVMe command by the smart NIC **1320** of FIG. **13**. In this example, the VM **1312** is presented an NVMe device through a virtual NVMe device emulation provided by the hypervisor **1314** (e.g., provided by a virtual NVMe device emulation module of the vSphere software of VMware Inc.). The NVMe device present in VM **1312** generates (at **1405**) an NVMe command (with data). The VM's NVMe driver passes (at **1410**) this NVMe command through the virtual device layer **1311** and the DSAN service layer **1313**, to the kernel NVMe module **1315**. At **1415**, the kernel NVMe module **1315** identifies the NVMeOF controller that needs to process this NVMe command, and provides the packet to this controller **1316** or **1317**.

[0092] The NVMEoRDMA **1316** or NVMEoTCP **1317** module running on the host (at **1420**) converts the NVMe command to one or more NVMe network packets (NVMeOF packets) and passes the packets to a PF **1370** of the PCIe bus **150**. At **1425**, the PF **1370** adds PCIe header information to the NVMe network packets, and then passes the packets along the PCIe bus **150**. The PCIe bus **150** creates a mapping between the PF **1370** and the VF module **1322** running on the smart NIC. Hence, the VF module **1322** receives each NVMeOF packet through the PCIe bus **150**.

[0093] At **1430**, the VFI module **1322** then transfers the NVMeOF packet either directly through the fast path to the NIC driver **1325**, or indirectly to the NIC driver **1325** through the slow path that involves the virtual switch **1326**. The NIC driver **1325** then forwards the NVMeOF packet through a shared port of the smart NIC, so that this packet can be forwarded through intervening network fabric (e.g., intervening switches/routers) to reach its destination external storage **1340**. In some embodiments, the fast-path processing of the VF **1322** allows the VF to directly pass the packet to the shared port of the smart NIC. The process then ends.

[0094] FIG. **15** illustrates another way of using the smart NIC as a network accelerator in some embodiments. In this example, one VM **1512** executes the NVMeOF driver, so that it cannot only bypass the DSAN service layer **1313**, but also the kernel NVMe **1315** and NVMeOF drivers **1316-17** of the host hypervisor **1514** that executes over a host OS **1500**. This approach provides the fastest access for a VM to one or more external storages through the VM's NVMeOF driver, which in this example is a GOS NVMe fabric driver. However, in this approach, no local storage is emulated for the VM **1514** by either the host or the smart NIC. This VM simply accesses the external storages through its NVMeOF driver. Specifically, in the example of FIG. **15**, the GOS NVMeOF driver inside the VM **1512** presents the NVMe device to the VM **1512**. Also, a PF **1580** is directly assigned to VM **1512** using a passthrough mode, such as SRIOV Mode or SIOV Mode.

[0095] For the PF **1580**, the smart NIC OS in FIG. **15** defines a VF **1523** to process the packets associated with the VM **1514**. In both FIGS. **13** and **15**, the smart NIC OS has a virtual switch **1326** to perform software switching operations and a network virtualization **1327** layer to perform network virtualization operations for the smart NIC. In some embodiments, these operations are analogous to the operations that traditionally have been performed on host computers to provide software switching and network virtualization operations. The smart NIC OS **1320** also has a NIC driver **1325** to communicate with the external storages **1340** through one or more ports of the smart NIC.

[0096] FIG. **16** illustrates a process **1600** that is performed to process egress packets from the VM **1512**. As shown, the process starts (at **1605**) when an application running on the VM **1512** generates an NVMe command (with data), and provides this command to the NVMeOF driver executing on this VM. This driver then coverts (at **1610**) the NVMe command to a set of one or more network packets, which it then provides to the PF **1580** directly.

[0097] The PF **1580** provides (at **1615**) the set of network packets that contains the NVMe command (with data) to the VF**2** **1523**, which is a high speed network adapter provided by the smart NIC **1320**. As described above for VF **1322** and operation **1430** of FIG. **14**, the VF**2** **1523** (at **1620**) transfers the set of network packets (containing the NVMe command/data) through the direct fast path or the indirect slow path, to a shared NIC port for forwarding to an external storage **1340**. In some embodiments, the shared NIC port can be used by both VFs **1322** and **1523** as well as other modules of the smart NIC for other forwarding operations.

[0098] The smart NIC operating system in some embodiments is provided with the host-computer hypervisor program as part of a single downloaded package. For instance, some embodiments provide a method for provisioning a smart NIC with a smart NIC operating system for enabling resource sharing on the smart NIC connected to a host computer. The method, in some embodiments, is performed by the host computer and begins when the host computer receives (1) a host-computer hypervisor program for enabling resource sharing on the host computer and (2) the smart NIC operating system. In some embodiments, the host-computer hypervisor program includes the smart NIC hypervisor program. The host computer then installs the host-computer hypervisor program and provides the smart NIC operating system to the smart NIC for the smart NIC to install on the smart NIC. One of ordinary skill in the art will appreciate that a hypervisor program is used as an example of virtualization software (e.g., software enabling resource sharing for a device executing the software).

[0099] The smart NIC, in some embodiments, is a NIC that includes (i) an application-specific integrated circuit (ASIC), (ii) a general purpose central processing unit (CPU), and (iii) memory. The ASIC, in some embodiments, is an I/O ASIC that handles the processing of packets forwarded to and from the computer and is at least partly controlled by the CPU. The CPU executes a NIC operating system in some embodiments that controls the ASIC and can run other programs, such as API translation logic to enable the compute manager to communicate with a bare metal computer. The smart NIC also includes a configurable peripheral control interconnect express (PCIe) interface in order to connect to the other physical components of the bare metal computer system (e.g., the x86 CPU, memory, etc.). Via this configurable PCIe interface, the smart NIC can present itself to the bare metal computer system as a multitude of devices, including a packet processing NIC, a hard disk (using non-volatile memory express (NVMe) over PCIe), or other devices.

[0100] Although not necessary for managing a bare metal computer, the NIC operating system of some embodiments is capable of executing a virtualization program (similar to a hypervisor) that enables sharing resources (e.g., memory, CPU resources) of the smart NIC among multiple machines (e.g., VMs) if those VMs execute on the computer. The virtualization program can provide compute virtualization services and/or network virtualization services similar to a managed hypervisor. These network virtualization services, in some embodiments, include segregating data messages into different private (e.g., overlay) networks that are defined over the physical network (shared between the private networks), forwarding the data messages for these private networks (e.g., performing switching and/or routing operations), and/or performing middlebox services for the private networks.

[0101] The host-computer hypervisor program and the smart NIC operating system, in some embodiments, are programs that do not have previous versions installed on the computer or the smart NIC. In other embodiments, the host-computer hypervisor program and the smart NIC operating system received by the host computer are update programs for previously installed versions of the host-computer hypervisor program and the smart NIC operating system. After a host-computer hypervisor program and the smart NIC operating system are received, the host computer, in some embodiments, receives an additional program for updating the smart NIC operating system and provides the received program to the smart NIC for the smart NIC to update the smart NIC operating system.

[0102] In some embodiments, after receiving the host-computer hypervisor program and the smart NIC operating system, the host computer detects (or determines) that the host computer is connected to the smart NIC. In some embodiments, the connection is made over a standard PCIe connection and the smart NIC is detected as a peripheral device that supports the installation of the smart NIC operating system. The host computer provides, based on the detection, the smart NIC operating system to the smart NIC for the smart NIC to install. In some embodiments, the smart NIC operating system is sent to the smart NIC along with an instruction to the smart NIC to install the smart NIC operating system.

[0103] In some embodiments, the host computer includes a local controller that receives the host-computer hypervisor program and the smart NIC operating system. The local controller, in some embodiments, provides the host-computer hypervisor program and the smart NIC operating system to a compute agent that installs the host-computer hypervisor program on the host computer to enable the host computer to share resources among a set of compute nodes (e.g., virtual machines, containers, Pods, etc.). The host-computer hypervisor program and the smart NIC operating system are particular examples of virtualization software that is used, in some embodiments, to enabling resource sharing for the host computer and smart NIC, respectively.

[0104] As mentioned above, the smart NIC in some embodiments includes a set of ASICs, a general purpose CPU, and a memory. The set of ASICs, in some embodiments, includes an ASIC for processing packets forwarded to and from the host computer as well as other ASICs for accelerating operations performed by the smart NIC on behalf of the host computer (e.g., encryption, decryption, storage, security, etc.). The smart NIC operating system, in some embodiments, includes virtualization programs for network virtualization, compute virtualization, and storage virtualization. The virtualization programs, in some embodiments, enable sharing the resources of the smart NIC among multiple tenants of a multi-tenant datacenter.

[0105] The network virtualization program provides network virtualization services on the smart NIC. The network virtualization services, in some embodiments, include forwarding operations (e.g., network switching operations and network routing operations). The forwarding operations are performed, in some embodiments, on behalf of multiple logically separate networks implemented over a shared network of a datacenter. Forwarding packets for different logical networks, in some embodiments, includes segregating packets for each logically separate network into the different logically separate networks. Forwarding operations for the different logical networks, in some embodiments, are implemented as different processing pipelines that perform different sets of operations. The different sets of operations include, in some embodiments, different logical packet forwarding operations (e.g., logical switching, logical routing, logical bridging, etc.) and different middlebox services (e.g., a firewall service, a load balancing service, etc.).

[0106] The compute virtualization program, in some embodiments, provides virtualized compute resources (virtual machines, containers, Pods, etc.) that execute over the compute virtualization program. The storage virtualization program, in some embodiments, provides storage virtualization services on the smart NIC. The virtualized storage, in some embodiments, include one or multiple of virtual storage area networks (vSANs), virtual volumes (vVOLs), and other virtualized storage solutions. The virtualized storage appears to the connected host computer as a local storage, in some embodiments, even when the physical resources that are the backend of the virtualized storage are provided by a distributed set of storages of multiple physical host computers.

[0107] FIG. **17** illustrates a system **1700** including a host computer **1710** and a connected smart NIC **1740** being configured with host computer virtualization software **1730** and a smart NIC operating system **1760**. Host computer **1710** includes a set of physical resources **1720** and smart NIC **1740** includes a separate set of physical resources **1750**. The set of physical resources **1720** of the host computer **1710**, in some embodiments, includes any or all of a set of

general purpose central processing units (CPUs), memory, and storage. The set of physical resources **1750** of the smart NIC **1740**, in some embodiments, include any or all of a set of general purpose central processing units (CPUs), application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), memory, and storage. The configuration of the host computer **1710** and smart NIC **1740** depicted in FIG. **17** will be described in relation to FIG. **18**.

[0108] FIG. **18** conceptually illustrates a process **1800** for installing programs enabling resource sharing on a host computer and smart NIC. The program for enabling resource sharing on the host computer, in some embodiments, is one of a hypervisor program, a virtual machine monitor, or other virtualization software. In some embodiments, the program for enabling resource sharing on the host computer executes as an operating system (OS) executing directly on the hardware of the host computer, while in other embodiments, the program executes as a software layer on top of an OS. Similarly, the program for enabling resource sharing on the smart NIC, in some embodiments, executes as an operating system (OS) executing directly on the hardware of the smart NIC, while in other embodiments, the program executes as a software layer on top of an OS.

[0109] The process **1800**, in some embodiments, is performed by a host computer (e.g., host computer **1710**) that in some embodiments, is a host computer (e.g., an x86 server) provided by a datacenter provider. The process **1800** begins by receiving (at **1810**) a host-computer virtualization program (e.g., host-computer hypervisor program **1715**) that includes a smart NIC operating system (e.g., smart NIC hypervisor program **1745**). The host-computer virtualization program (e.g., host-computer hypervisor program **1715**) and smart NIC operating system (e.g., smart NIC hypervisor program **1745**), in some embodiments, are installer programs that install virtualization software (e.g., a software virtualization layer or a virtualization OS). The host-computer virtualization program, in some embodiments, is received from a network controller computer to configure the host computer to support virtualized compute nodes, storage, network cards, etc., to be implemented on the host computer for a virtual or logical network associated with the network controller computer.

[0110] Stage **1701** of FIG. **17** illustrates a host computer **1710** that has not yet been configured by virtualization software receiving a host-computer hypervisor program **1715** that includes smart NIC hypervisor program **1745**. In some embodiments, the two programs **1715** and **1745** are received separately (e.g., simultaneously as part of a package or sequentially). In other embodiments, the program received (at **1810**) is an update to one or both of the host-computer virtualization program and the smart NIC operating system. The update, in some embodiments, is processed as an update to the host-computer virtualization program even in the case that the update includes only updates to the smart NIC operating system.

[0111] After receiving (at **1810**) the host-computer virtualization program and the smart NIC operating system, the host computer then installs (at **1820**) the received host-computer virtualization program (e.g., host-computer hypervisor program **1715**) on the host computer. The virtualization program, in some embodiments, is a hypervisor such as ESXi™ provided by VMware, Inc. or other virtualization programs. As shown in stage **1702** of FIG. **17**, the host computer **1710** installs a host computer hypervisor **1730** (dashed lines in FIG. **17** indicating software executing on a device) after receiving the host-computer hypervisor program **1715**. After installing the host-computer virtualization program **1715**, the host computer **1710** is able to provide virtual resources (e.g., compute nodes, virtual switches, virtual storage, etc.) based on the physical resources **1720** of the host computer **1710**.

[0112] After, or as part of, installing (at **1820**) the host-computer virtualization program, the host computer detects (at **1830**) that the smart NIC operating system is included in the host-computer virtualization program. In some embodiments, detecting (at **1830**) that the smart NIC operating system is incorporated in the host-computer virtualization program includes a set of operations to perform to program any virtualization-capable smart NICs connected to the host computer. The set of operations, in some embodiments, includes an operation to detect whether a virtualization-capable smart NIC is connected to the host computer.

[0113] The host computer determines (at **1840**) that a virtualization-capable smart NIC is connected to the host computer. In some embodiments, determining (at **1840**) that a virtualization-capable smart NIC is connected to the host computer is part of the installation process for the host-computer virtualization program. Determining (at **1840**) that a virtualization-capable smart NIC is connected to the host computer, in some embodiments, is based on a set of components exposed to the host computer by the smart NIC. In some embodiments, the host-computer virtualization program (e.g., an ESXi™ installer) queries a baseboard management controller (BMC) of the host computer to determine (at **1840**) that the smart NIC is compatible with the smart NIC operating system (e.g., a smart NIC operating system (OS) such as ESXio™). In some embodiments, a virtualization-capable smart NIC is identified to the connected host computer during a previously performed process that configures the connection between the host-computer virtualization program computer and the smart NIC.

[0114] After determining (at **1840**) that a virtualization-capable smart NIC is connected to the host computer, the host computer provides (at **1850**) the smart NIC operating system to the smart NIC for the smart NIC to install a virtualization layer to enable the smart NIC to share resources on the smart NIC. FIG. **17** illustrates, in stage **1702**, that the host computer **1710** sends the smart NIC hypervisor program **1745** to smart NIC **1740** for the smart NIC **1740** to install a smart NIC operating system. In stage **1703** of FIG. **17**, the smart NIC **1740** installs smart NIC hypervisor **1760** to enable virtualization of the physical resources **1750** of the smart NIC **1740**.

[0115] In some embodiments, providing (at **1850**) the smart NIC operating system for the smart NIC to install the smart NIC operating system includes multiple sub-operations. FIG. **19** conceptually illustrates a process **1900** that, in some embodiments, represents the sub-operations that are included in operation **1850**. FIG. **19** is described, at least in part, in relation to FIGS. **20-22**. Process **1900** begins by configuring (at **1910**) the smart NIC to boot from an image stored on the host computer. In some embodiments, the host-computer virtualization program invokes BMC APIs to configure (at **1910**) the smart NIC to enable a unified extensible firmware interface (UEFI) SecureBoot on the smart NIC.

[0116] After configuring the smart NIC to enable booting from an image stored on the host computer, the smart NIC operating system is staged (at **1920**) on the host computer for the smart NIC to use in an initial boot-up process. The host-computer virtualization program, in some embodiments, invokes BMC APIs to stage (at **1920**) the smart NIC operating system (e.g., ESX.io) in BMC storage as an image file (e.g., as an ISO, DD, tgz, or zip file) for the smart NIC to perform the initial boot-up of the smart NIC operating system.

[0117] FIG. **20** illustrates a simplified view of host computer **2010**, including BMC **2018** connecting to smart NIC **2040** through PCIe bus **2042**. FIG. **20** illustrates that the BMC **2018** provides a smart NIC operating system **2045** stored in BMC memory **2019** to the CPU **2044** of the smart NIC **2040** for the CPU **2044** to perform a first boot from the image stored in BMC storage **2019** (operation "1"). FIG. **20** illustrates components of the smartNIC, which will be described below by reference to other figures.

[0118] At **1930**, the process **1900** provides the smart NIC virtualization program for storage on partitioned memory. FIG. **21** conceptually illustrates a process **2100** that is performed by the smart NIC to install the smart NIC operating system as part of process **1900** of FIG. **19**. In some embodiments, the smart NIC, at this point, performs (at **2110**) a boot sequence for the smart NIC operating system from the BMC storage of the host computer. During the initialization, the local storage (e.g., embedded multi-media controller (eMMC), or other memory) is detected and partitioned (at **2120**). In some embodiments, the local storage is detected based on initialization scripts of the smart NIC operating system. The detected storage is then partitioned.

[0119] The smart NIC operating system (e.g., ESX.io bootloader and system modules) is then stored (at **2130**) in the local partitioned storage. In some embodiments, the smart NIC operating system is copied to the smart NIC local storage for storing (at **2130**) from the host computer based on a process of the smart NIC operating system. In other embodiments, the host-computer virtualization program detects that the smart NIC has booted from the image and partitioned the storage and provides the smart NIC operating system to the smart NIC for storage (at **2130**). FIG. **22** illustrates a smart NIC **2240** after the installation is complete with its storage **2246** partitioned into a first partition **2246***a* storing the smart NIC operating system (NIC OS) **2245** and a second partition **2246***b*.

[0120] FIG. **20** illustrates that after providing the smart NIC operating system **2045** to the CPU **2044** (in operation "1"), the smart NIC operating system is provided to the memory **2046** (as operation "2"). In this example, the smart NIC OS **2045** is provided through the external PCIe bus **2042** and an internal PCIe bus **2043**. The internal PCIe bus **2043** connects the I/O ASIC side of the NIC (including ASIC **2047** and physical ports **2041**) with the CPU side of the NIC (including CPU **2044** and memory **2046**). While FIG. **20** and several other figures described below use an internal PCIe bus **2043**, other embodiments do not use an internal PCIe bus. For instance, some embodiments implement the NIC as a system-on-chip and have the different circuits of the NIC communicate through other interconnects and buses.

[0121] The smart NIC operating system then verifies (at **2140**) that the installation was successful. In some embodiments, verifying (at **2140**) that the installation was successful includes verifying that the smart NIC device and functions are successfully enumerated. The verification (at **2140**), in some embodiments, is based on a set of post-installation scripts. In some embodiments, the verification includes a communication to the host-computer virtualization program installation process that the installation on the smart NIC was successful.

[0122] The host computer BMC then configures (at **1940**) the smart NIC to boot from the local copy of the smart NIC operating system. FIG. **20** illustrates that after the smart NIC operating system is stored in the memory **2046**, the CPU accesses the smart NIC operating system (SN HV **2045**) from the memory **2046** (as operation "3").

[0123] The host computer then completes (at **1950**) the installation of the host-computer virtualization program and reboots the host computer and the smart NIC. These operations (**1940** and **1950**) are reflected in process **2100** of FIG. **21** in operation **2150**, in which the smart NIC is configured to boot from the locally-stored smart NIC operating system and is rebooted. In some embodiments, the host computer is rebooted first to complete the installation of the host-computer virtualization program, and the host computer and smart NIC are then rebooted again to complete the installation of the smart NIC operating system. The host computer, in some embodiments, is rebooted first and then initiates a reboot of the smart NIC (from the smart NIC operating system stored in the memory of the smart NIC). In embodiments in which the smart NIC supports compute nodes of multiple tenants, an attempt to install the smart NIC operating system by another tenant's host-computer virtualization program installer is blocked. In some embodiments, the installation by a second tenant is unnecessary and would destroy any virtualization already performed for the first tenant. In such embodiments, the smart NIC or the host-computer virtualization program installer is programmed in such a way to determine if the smart NIC operating system is already installed. Additionally, the smart NIC or the host-computer hypervisor program installer, in some embodiments, is programmed to identify the tenant that installed the smart NIC operating system to allow updates to the smart NIC operating system made by that tenant.

[0124] As illustrated in FIG. **17**, the host-computer hypervisor program **1715** and the smart NIC hypervisor **1745**, in some embodiments, are programs that do not have previous versions installed on the host computer **1710** or the smart NIC **1740**. In other embodiments, the host-computer hypervisor program **1715** and the smart NIC hypervisor **1745** received by the host computer **1710** are update programs for previously-installed versions of the host-computer hypervisor program **1715** and the smart NIC hypervisor **1745**. In yet other embodiments, after a host-computer hypervisor program **1715** and the smart NIC hypervisor **1745** are received, the host computer **1710**, in some embodiments, receives an additional program for updating the smart NIC hypervisor **1745** and provides the received program to the smart NIC **1740** for the smart NIC **1740** to update the smart NIC hypervisor **1745**.

[0125] FIG. **22** illustrates a smart NIC **2240** after a set of configuration processes similar or identical to those described above in relation to FIGS. **18**, **19**, and **21**. After installing the smart NIC operating system **2245**, the CPUs **2044** execute a NIC operating system **2260** (e.g., a hypervisor, virtualization OS, or virtual machine monitor, etc.) that includes, in some embodiments, a compute virtualiza-

tion module **2261**, a network virtualization module **2262**, and a storage virtualization module **2263**. In some embodiments, a smart NIC operating system **2260** supports only a subset of these functions, supports additional functions, or supports a different combination of functions. The network virtualization module (or capability) **2262**, in some embodiments, is used to present the smart NIC **2240** as multiple physical functions (PF) connected to a single host computer (e.g., a server) or a set of host computers. Each PF, in some embodiments, can be further virtualized as multiple virtual functions (VFs).

[0126] As used in this document, physical functions (PFs) and virtual functions (VFs) refer to ports exposed by a smart NIC using a PCIe interface to connect to a host computer (or set of host computers) over a PCIe bus. A PF refers to an interface of the smart NIC that is recognized as a unique resource with a separately configurable PCIe interface (e.g., separate from other PFs on a same smart NIC). The VF refers to a virtualized interface that is not fully-configurable as a separate PCIe resource, but instead inherits some configuration from the PF with which it is associated while presenting a simplified configuration space. VFs are provided, in some embodiments, to provide a passthrough mechanism that allows compute nodes executing on a host computer to receive data messages from the smart NIC without traversing a virtual switch of the host computer. The VFs, in some embodiments, are provided by virtualization software executing on the smart NIC. The VFs and PFs, in some embodiments, are deployed to support the storage and computer virtualization modules **2263** and **2261**. For example, a PF or VF can be deployed to present a storage or compute resource provided by the smart NIC as a local device (i.e., a device connected to the host computer by a PCIe bus).

[0127] The smart NIC **2240** also includes a local memory **2246** and a set of general purpose CPUs **2044** that are used to install (and support) the virtualization layer **2330**, which enables resource sharing of elements on the I/O portion and a compute portion of the smart NIC (e.g., the CPUs **2044**, memory **2246**, etc.). As shown, smart NIC operating system **2245** is stored in memory **2246** (and more specifically, in memory partition **2246***a*) which communicates with the CPUs **2044** to execute the smart NIC operating system **2245** to install the NIC operating system **2260** (e.g., ESX.io). In some embodiments, the memory **2246** is an embedded multi-media controller (eMMC) memory that includes flash memory and a flash memory controller. The memory **2246** and the CPUs **2044** communicate, in some embodiments, with other elements of the smart NIC **2240** over an internal PCIe bus **2043**.

[0128] Smart NIC **2240** also includes an I/O ASIC **2047** (among a set of additional ASICs or field-programmable gate arrays (FPGAs) not shown) that can be used to accelerate data message forwarding or other networking functions (encryption, security operations, storage operations, etc.). A set of physical ports **2041** that provide connections to a physical network and interacts with the I/O ASIC **2047** is also included in smart NIC **2240**. The I/O ASIC and physical ports that are depicted in FIG. **20** perform a similar operations.

[0129] The host computer and smart NIC, in some embodiments, are elements of a datacenter that implements virtual networks for multiple tenants. In some embodiments, the virtual networks implemented in the datacenter include one or more logical networks including one or more logical forwarding elements, such as logical switches, routers, gateways, etc. In some embodiments, a logical forwarding element (LFE) is defined by configuring several physical forwarding elements (PFEs), some or all of which execute on host computers or smart NICs along with deployed compute nodes (e.g., VMs, Pods, containers, etc.). The PFEs, in some embodiments, are configured to implement two or more LFEs to connect two or more different subsets of deployed compute nodes. The virtual network in some embodiments, is a software-defined network (SDN) such as that deployed by NSX-T™ and includes a set of SDN managers and SDN controllers. In some embodiments, the set of SDN managers manage the network elements and instruct the set of SDN controllers to configure the network elements to implement a desired forwarding behavior for the SDN.

[0130] FIG. **23** illustrates a system **2300** that includes the host computer **2310**, the smart NIC **2340**, a set of SDN controller computers **2370**, and a set of SDN manager computers **2380**. The set of SDN manager computers **2380** implement a management plane for a particular SDN (e.g., a cloud provider SDN, or a tenant SND executed in the cloud or in a private datacenter). The set of SDN manager computers **2380** receive input from a user to implement a certain SDN configuration including, in some embodiments, configuration for a set of LFEs, a set of compute nodes, and a set of storage resources. The set of SDN manager computers **2380** communicate the desired configurations to the set of SDN controller computers **2370** implementing a control plane for the SDN. The set of SDN controllers **2370** generate configuration data for a set of host computers (including host computer **2310**) and provide control messages to a local controller **2390** on the host computer **2310** to configure a set of the network elements specified by a user. In some embodiments, the SDN manager computers **2380** and SDN controller computers **2370** are the NSX-T™ managers and controllers licensed by VMware, Inc.

[0131] As shown, the set of SDN controller computers **2370** send a host-computer hypervisor program **2315** to a local controller **2390** of host computer **2310** through smart NIC **2340** (using physical port (PP) **2341** and a PCIe bus **2342**). In some embodiments, the host-computer hypervisor program **2315** is an installer program executed by the compute resources **2321** of host computer **2310** to install a virtualization layer **2330** (e.g., a hypervisor such as ESXi™ provided by VMware, Inc.) to enable the physical resources **2320** of host computer **2310** (including compute, network and storage resources **2321**, **2322**, and **2323**) to be shared among multiple virtualized machines.

[0132] Local controller **2390** receives the host-computer hypervisor program **2315** and provides it to the physical resources **2320** (e.g., runs the host-computer hypervisor program **2315** using the compute resources **2321** of the host computer **2310**). Based on the host-computer hypervisor program **2315**, a virtualization layer **2330** is installed on the host computer **2310** (shown using dashed lines to distinguish between hardware and software of the host computer **2310**). While virtualization layer **2330** is shown as including a compute virtualization module **2261**, a network virtualization module **2262**, and a storage virtualization module **2263**, in some embodiments, a virtualization layer **2330** supports only a subset of these functions, supports additional functions, or supports a different combination of functions. As

described above in relation to FIG. **18**, as part of executing host-computer hypervisor program **2315** to install the virtualization layer **2330**, the host computer **2310** will provide smart NIC operating system **2345** to smart NIC **2340** for the smart NIC **2340** to execute (e.g., install a smart NIC virtualization layer).

[0133] FIG. **24** illustrates a host computer **2410** executing a host computer hypervisor **2430** and a set of compute nodes (CN_1-CN_M) **2411** for a first tenant ("T1") and a set of compute nodes (CN_a-CN_x) **2412** for a second tenant ("T2"). FIG. **24** also illustrates a logical view of the separate logical networks defined for T1 and T2. As shown, the logical networks for the separate tenants include a set of logical routers and logical switches (**2431** and **2432** for T1 and **2433** and **2434** for T2, respectively) that connect the compute nodes of the tenant. The different logical networks are both implemented by the host computer hypervisor **2430** and, in some embodiments, the smart NIC **2440**. The host computer hypervisor **2430**, in some embodiments, includes a virtual switch (e.g., a software switch) that implements the LRs and LSs for the different tenants on the host computer **2410**. In some embodiments, the I/O ASIC **2047** of the smart NIC **2440** is configured by the host computer hypervisor **2430** to perform logical routing and logical switching operations for the separate tenants. In other embodiments, the I/O ASIC **2047** of the smart NIC **2440** is configured by a hypervisor (not shown) of the smart NIC **2440**.

[0134] The I/O ASIC **2047** of the smart NIC **2440** and the host computer hypervisor **2430**, in some embodiments, implement separate processing pipelines for the separate tenants (e.g., the separate logical networks). Data messages, e.g., ingressing data messages T1 and T2, are segregated into the different processing pipelines of the different logical networks of the different tenants, in some embodiments, based on logical network identifiers (e.g., virtual local area network (VLAN) or virtual extensible LAN (VXLAN) identifiers).

[0135] FIG. **25** illustrates a smart NIC **2540** providing compute virtualization **2561** and network virtualization **2562** to provide virtualized resources (e.g., compute nodes **2513**, physical functions **2570***a-n*, and a set of virtual functions **2571**) to be used by compute nodes **2511** executing on a host computer **2510** (depicted as executing within a host computer hypervisor **2530**). In some embodiments, compute nodes **2513** are edge gateway machines that provide gateway services for compute nodes **2511** executing on host computer **2510**.

[0136] Network virtualization **2562** provides a virtualized PCIe interface that presents the PCIe bus **2542** as including a set of physical functions (PFs **2570***a-n*) as defined above and, for a set of physical functions, a set of virtual functions **2571**. Both the host computer hypervisor **2530** and NIC OS **2560** execute a virtual switch **2532** that provides logical routing and logical switching operations for compute nodes (virtual machines, container, Pods, etc.). In some embodiments, a virtual switch **2573** on the smart NIC **2540** provides logical forwarding operations for compute nodes on both the smart NIC **2540** and on the host computer **2510**. In some embodiments, the virtual switch **2573** interacts with the I/O ASIC **2047** to perform data message processing offload (e.g., flow processing offload) on behalf of the host computer **2510**.

[0137] FIG. **26** illustrates an interaction between an I/O ASIC **2047**, a virtual switch **2673**, and a fast path entry generator **2675**, in some embodiments. In some embodiments, I/O ASIC **2047** is configured to perform fast path processing for data messages to and from compute nodes executing on host computers connected to smart NIC **2640**. In some embodiments, for first data messages in a data message flow (e.g., data message **2680**), the I/O ASIC **2047** is programmed to provide data message **2680** to a virtual switch **2673** executing in the NIC OS **2660**. The virtual switch **2673** processes the data message **2680** through a processing pipeline **2674**. Processing pipeline **2674** includes operations **2674***a-n*. The operations **2674***a-n*, in some embodiments, include a set of logical forwarding operations (logical switching, bridging, routing, etc.). In some embodiments, the operations **2674***a-n* also include a set of middlebox services (e.g., firewall, load balancing, deep packet inspection, etc.) enabled for a particular logical network (e.g., belonging to a particular logical tenant associated with the data message **2680**). The processing pipeline **2674**, in some embodiments, identifies a particular set of data message attributes **2681** used to identify a data message flow or set of data message flows to which the data message **2680** belongs and determines a particular set of actions (e.g., slow path result **2682**) to take for future data messages matching the identified data message attributes **2681**. The data message attributes **2681** and slow path results **2682** are then provided to fast path entry generator **2675** to be combined into a fast path entry **2691** to be programmed into I/O ASIC **2047** to process future data messages having attributes that match the identified data message attributes **2681**.

[0138] FIG. **27** illustrates a system **2700** including a smart NIC **2740** and a set of host computers **2710**A-J connected to the smart NIC **2740** through two different PCIe buses **2742**A and **2742**J. The smart NIC **2740** has a NIC OS **2760**. Each PCIe bus **2742** is used to present virtualized elements of the smart NIC **2740** as different devices connected to the host computers **2710**A-J. For example, PF **2770***a* and **2770***n* present as a NIC, while PF **2770***b* and **2770**(*n*−1) present as connected storage devices. PF **2770***a*, as shown, presents a set of VFs **2771** as passthrough ports bypassing a virtual switch (not shown) of host computer **2710**A. PF **2770***b* and **2770**(*n*−1) appear to the host computers **2710**A and **2710**J, the hypervisors **2730***a-j* executing on these computers, and the compute nodes **2711***a-b* that execute on the hypervisors **2730** as (emulated) local storages **2765***a* and **2765***b* connected by a PCIe bus **2742**. The emulated local storages **2765** may appear as a local storage, a virtual storage area network, or virtual volume. In some embodiments, the storage virtualization **2263** backs the emulated local storage **2765** with a virtualized storage using non-volatile memory express (NMVe) or NVMe over fabrics (NVMe-OF) **2766**. The virtualized storage, in some embodiments, communicates with an external storage that is located on multiple physical storage devices **2780***a-p*. The communication, in some embodiments, uses NVMe-oF based on remote direct memory access (RDMA) or transport control protocol (TCP).

[0139] FIG. **28** conceptually illustrates an electronic system **2800** with which some embodiments of the invention are implemented. The electronic system **2800** can be used to execute any of the control, virtualization, or operating system applications described above. The electronic system **2800** may be a computer (e.g., a desktop computer, personal computer, tablet computer, server computer, mainframe, a blade computer etc.), phone, PDA, or any other sort of

electronic device. Such an electronic system includes various types of computer readable media and interfaces for various other types of computer readable media. Electronic system **2800** includes a bus **2805**, processing unit(s) **2810**, a system memory **2825**, a read-only memory **2830**, a permanent storage device **2835**, input devices **2840**, and output devices **2845**.

[0140] The bus **2805** collectively represents all system, peripheral, and chipset buses that communicatively connect the numerous internal devices of the electronic system **2800**. For instance, the bus **2805** communicatively connects the processing unit(s) **2810** with the read-only memory **2830**, the system memory **2825**, and the permanent storage device **2835**.

[0141] From these various memory units, the processing unit(s) **2810** retrieve instructions to execute and data to process in order to execute the processes of the invention. The processing unit(s) may be a single processor or a multi-core processor in different embodiments.

[0142] The read-only-memory (ROM) **2830** stores static data and instructions that are needed by the processing unit(s) **2810** and other modules of the electronic system. The permanent storage device **2835**, on the other hand, is a read-and-write memory device. This device is a non-volatile memory unit that stores instructions and data even when the electronic system **2800** is off. Some embodiments of the invention use a mass-storage device (such as a magnetic or optical disk and its corresponding disk drive) as the permanent storage device **2835**.

[0143] Other embodiments use a removable storage device (such as a floppy disk, flash drive, etc.) as the permanent storage device. Like the permanent storage device **2835**, the system memory **2825** is a read-and-write memory device. However, unlike storage device **2835**, the system memory is a volatile read-and-write memory, such a random access memory. The system memory **2825** stores some of the instructions and data that the processor needs at runtime. In some embodiments, the invention's processes are stored in the system memory **2825**, the permanent storage device **2835**, and/or the read-only memory **2830**. From these various memory units, the processing unit(s) **2810** retrieve instructions to execute and data to process in order to execute the processes of some embodiments.

[0144] The bus **2805** also connects to the input and output devices **2840** and **2845**. The input devices **2840** enable the user to communicate information and select commands to the electronic system. The input devices **2840** include alphanumeric keyboards and pointing devices (also called "cursor control devices"). The output devices **2845** display images generated by the electronic system **2800**. The output devices **2845** include printers and display devices, such as cathode ray tubes (CRT) or liquid crystal displays (LCD). Some embodiments include devices such as a touchscreen that function as both input and output devices.

[0145] Finally, as shown in FIG. **28**, bus **2805** also couples electronic system **2800** to a network **2865** through a network adapter (not shown). In this manner, the computer can be a part of a network of computers (such as a local area network ("LAN"), a wide area network ("WAN"), or an Intranet, or a network of networks, such as the Internet. Any or all components of electronic system **2800** may be used in conjunction with the invention.

[0146] Some embodiments include electronic components, such as microprocessors, storage and memory that store computer program instructions in a machine-readable or computer-readable medium (alternatively referred to as computer-readable storage media, machine-readable media, or machine-readable storage media). Some examples of such computer-readable media include RAM, ROM, read-only compact discs (CD-ROM), recordable compact discs (CD-R), rewritable compact discs (CD-RW), read-only digital versatile discs (e.g., DVD-ROM, dual-layer DVD-ROM), a variety of recordable/rewritable DVDs (e.g., DVD-RAM, DVD-RW, DVD+RW, etc.), flash memory (e.g., SD cards, mini-SD cards, micro-SD cards, etc.), magnetic and/or solid state hard drives, read-only and recordable Blu-Ray® discs, ultra-density optical discs, any other optical or magnetic media, and floppy disks. The computer-readable media may store a computer program that is executable by at least one processing unit and includes sets of instructions for performing various operations. Examples of computer programs or computer code include machine code, such as is produced by a compiler, and files including higher-level code that are executed by a computer, an electronic component, or a microprocessor using an interpreter.

[0147] While the above discussion primarily refers to microprocessor or multi-core processors that execute software, some embodiments are performed by one or more integrated circuits, such as application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs). In some embodiments, such integrated circuits execute instructions that are stored on the circuit itself.

[0148] As used in this specification, the terms "computer", "server", "processor", and "memory" all refer to electronic or other technological devices. These terms exclude people or groups of people. For the purposes of the specification, the terms display or displaying means displaying on an electronic device. As used in this specification, the terms "computer readable medium," "computer readable media," and "machine readable medium" are entirely restricted to tangible, physical objects that store information in a form that is readable by a computer. These terms exclude any wireless signals, wired download signals, and any other ephemeral signals.

[0149] This specification refers throughout to computational and network environments that include virtual machines (VMs). However, virtual machines are merely one example of data compute nodes (DCNs) or data compute end nodes, also referred to as addressable nodes. DCNs may include non-virtualized physical hosts, virtual machines, containers that run on top of a host operating system without the need for a hypervisor or separate operating system, and hypervisor kernel network interface modules.

[0150] VMs, in some embodiments, operate with their own guest operating systems on a host using resources of the host virtualized by virtualization software (e.g., a hypervisor, virtual machine monitor, etc.). The tenant (i.e., the owner of the VM) can choose which applications to operate on top of the guest operating system. Some containers, on the other hand, are constructs that run on top of a host operating system without the need for a hypervisor or separate guest operating system. In some embodiments, the host operating system uses name spaces to isolate the containers from each other and therefore provides operating-system level segregation of the different groups of applications that operate within different containers. This segregation is akin to the VM segregation that is offered in hypervisor-virtualized environments that virtualize system

hardware, and thus can be viewed as a form of virtualization that isolates different groups of applications that operate in different containers. Such containers are more lightweight than VMs.

[0151] Hypervisor kernel network interface modules, in some embodiments, are non-VM DCNs that include a network stack with a hypervisor kernel network interface and receive/transmit threads. One example of a hypervisor kernel network interface module is the vmknic module that is part of the ESXi™ hypervisor of VMware, Inc.

[0152] It should be understood that while the specification refers to VMs, the examples given could be any type of DCNs, including physical hosts, VMs, non-VM containers, and hypervisor kernel network interface modules. In fact, the example networks could include combinations of different types of DCNs in some embodiments.

[0153] While the invention has been described with reference to numerous specific details, one of ordinary skill in the art will recognize that the invention can be embodied in other specific forms without departing from the spirit of the invention. For instance, several examples were provided above by reference to specific distribute storage processes, such as vSAN. One of ordinary skill will realize that other embodiments use other distributed storage services (e.g., vVol offered by VMware, Inc.). The vSAN and vVol services of some embodiments are further described in U.S. Pat. Nos. 8,775,773 and 9,665,235, which are hereby incorporated by reference. Thus, one of ordinary skill in the art would understand that the invention is not to be limited by the foregoing illustrative details, but rather is to be defined by the appended claims.

We claim:

1. A method of emulating a local storage for a host computer comprising a network interface card (NIC), the method comprising

on the NIC:

deploying a storage emulator first program to emulate a local storage, from a plurality of external storages accessed through the NIC, for a set of processes executing on the host computer; and

deploying a storage service second program to perform a set of distributed storage services that account for the set of processes on the host computer lack of knowledge regarding plurality of external storages being used to emulate the local storage.

2. The method of claim **1**, wherein the set of distributed storage services comprises data efficiency processes including at least one of deduplication operations and compression operations.

3. The method of claim **1**, wherein the set of distributed storage services comprises security processes including at least one of end-to-end encryption and access control operations.

4. The method of claim **1**, wherein the set of distributed storage services comprises data and life cycle management, including at least one of storage vmotion, snapshot operations, snapshot schedules, cloning, disaster recovery, backup, and long term storage.

5. The method of claim **1**, wherein the set of distributed storage services improve performance, resiliency and security of the set of host processes access the plurality of external storages through the NIC.

6. The method of claim **1**, wherein the set of distributed storage services comprise encryption services encrypt the read and write requests and responses that are exchanged between the set of host processes and the plurality of external storages that are made to appear as the local storage.

7. The method of claim **6**, wherein the storage service second program is a proxy program that uses a storage service program executing on another device to provide one or more of the distributed storage services in the set of distributed storage services.

8. The method of claim **6** further comprising deploying a network fabric driver on the NIC to access the plurality of external storages through one or more intervening networks.

9. The method of claim **8**, wherein the network fabric driver is a non-volatile memory express over fabric (NVMeOF) driver.

10. The method of claim **9**, wherein the storage service second program communicates through the NVMeOF driver with other storage service programs that provide the distributed storage service for a plurality of compute and storage nodes.

11. The method of claim **1**, wherein the set of processes comprise an operating system executing on the host computer.

12. The method of claim **1**, wherein the set of processes comprise a set of machines executing on the host computer.

13. The method of claim **1**, wherein the set of processes comprise a hypervisor executing on the host computer.

14. A non-transitory machine readable medium storing sets of instructions for emulating a local storage for a host computer comprising a network interface card (NIC), the sets of instructions for execution by at least one processing unit of the NIC, the sets of instructions for:

emulating a local storage, from a plurality of external storages accessed through the NIC, for a set of processes executing on the host computer; and

performing a set of distributed storage services that account for the set of processes on the host computer lack of knowledge regarding plurality of external storages being used to emulate the local storage.

15. The non-transitory machine readable medium of claim **14**, wherein the set of distributed storage services comprises data efficiency processes including at least one of deduplication operations and compression operations.

16. The non-transitory machine readable medium of claim **14**, wherein the set of distributed storage services comprises security processes including at least one of end-to-end encryption and access control operations.

17. The non-transitory machine readable medium of claim **14**, wherein the set of distributed storage services comprises data and life cycle management, including at least one of storage vmotion, snapshot operations, snapshot schedules, cloning, disaster recovery, backup, and long term storage.

18. The non-transitory machine readable medium of claim **14**, wherein the set of distributed storage services improve performance, resiliency and security of the set of host processes access the plurality of external storages through the NIC.

19. The non-transitory machine readable medium of claim **14**, wherein the set of distributed storage services comprise encryption services encrypt the read and write requests and responses that are exchanged between the set of host processes and the plurality of external storages that are made to appear as the local storage.

20. The non-transitory machine readable medium of claim **14**, wherein the set of instructions further for using a network fabric driver on the NIC to access the plurality of external storages through one or more intervening networks.

* * * * *