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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 January 2013 (03.01.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/002978 A2

(51) International Patent Classification: Not classified

(21) International Application Number: PCT/US2012/041043

(22) International Filing Date: 6 June 2012 (06.06.2012)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/169,024 27 June 2011 (27.06.2011) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventors: **FRIES, Robert**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **PARTHASARATHY, Srivatsan**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **SANGHVI, Ashvinkumar**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **RAMARATHINAM, Aravind**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **GRIER, Michael**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US).

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

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

[Continued on next page]

(54) Title: HOST ENABLED MANAGEMENT CHANNEL

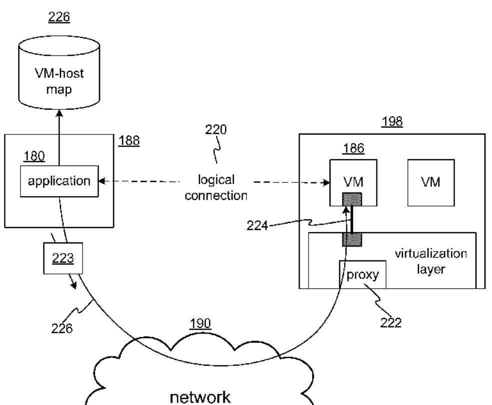


FIG. 4

(57) **Abstract**: A logical communication path is provided between a target virtual machine (VM) and a host or application communicating with the VM. The target VM runs on a hypervisor host that has a hypervisor and a proxy agent. The hypervisor manages execution of the VM. A mapping is maintained indicating which VMs execute on which hosts. When the host or application is to send a message or packet to the target VM, the mapping is consulted and the hypervisor host hosting the target VM is identified. The message or packet, which may identify the target VM, is transmitted to the hypervisor host. A proxy agent at the hypervisor host selects a communication channel between the hypervisor and the target VM. The hypervisor then passes the message or packet through the selected channel to the target VM.

WO 2013/002978 A2

HOST ENABLED MANAGEMENT CHANNEL

BACKGROUND

[0001] In the field of machine virtualization, virtual machines (VMs) have network functionality. That is, VMs may implement a network protocol stack to communicate via a network with other VMs or physical machines. For instance, virtualization hosts (e.g., Hyper-V (TM) hosts) may form part of a virtualization fabric that hosts guest VMs, where a Fabric Controller manages the virtualization fabric (as used in this Background "host" may refer to a Fabric Controller, for example, or any other computer). However, for various reasons, there may be no network connectivity between a host on a network and a VM, even though there is network connectivity between the host and a machine running the VM (to referred to as the "VM host"). For example, the VM might be on a Virtual Private Network (VPN) to which the host does not belong and the VM's network address may not be valid on the host's network. A firewall might block access to the VM from the hosts' network while allowing access on the VM host's network. A VM might simply be on a different network than the host that might need to communicate with the VM.

[0002] In some circumstances, it is desirable to communicate with a VM using a standard protocol such as HTTP (Hypertext Transfer Protocol), SOAP (Simple Object Access Protocol), WMI (TM) (Windows Management Instrumentation), the WS-Management protocol (transporting WMI calls over a SOAP based protocol via HTTP), and so forth. For example, in some data centers or clouds, VMs might have network agents or services running thereon that perform management functions (such as applying patches to a guest operating system, handling cloud fabric tasks, etc.), perhaps with one or more communication channels for control (e.g., WMI over HTTP) or data (BITS via HTTP). These management services or agents are controlled by a management application (e.g., a Fabric Controller), running on a controller host, for example. The management application sends packets, for example HTTP packets, to the VM's network address and the HTTP packets are delivered to the management agent. The management agents may perform functions in response to information in the payloads of the packets. However, when the management application does not have network connectivity to the VM, it is unable to invoke the management agents on the VM.

[0003] Techniques to enable communication with VMs via communication channels between hypervisors and the VMs are discussed below.

SUMMARY

[0004] The following summary is included only to introduce some concepts discussed in the Detailed Description below. This summary is not comprehensive and is not intended to delineate the scope of the claimed subject matter, which is set forth by the claims presented at the end.

[0005] A logical communication path is provided between a target virtual machine (VM) and a host or application communicating with the VM. For example, a path between virtualization host and a VM. The target VM runs on a hypervisor host that has a hypervisor and a proxy agent (e.g., an HTTP proxy). The hypervisor manages execution of the VM. A mapping is maintained indicating which VMs execute on which hosts. When the host or application is to send a message or packet to the target VM, the mapping is consulted and the hypervisor host hosting the target VM is identified. The message or packet, which may identify the target VM, is transmitted to the hypervisor host. A proxy agent at the hypervisor host selects a communication channel between the hypervisor and the target VM. The hypervisor then passes the message or packet through the selected channel to the target VM.

[0006] Many of the attendant features will be explained below with reference to the following detailed description considered in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The present description will be better understood from the following detailed description read in light of the accompanying drawings, wherein like reference numerals are used to designate like parts in the accompanying description.

[0008] Figure 1 shows an example virtualization layer.

[0009] Figure 2 shows processes and interactions of virtualization a layer in relation to virtual machines and virtual machine images.

[0010] Figure 3 shows an example of an application communicating with an agent running on a guest operating system hosted by a VM.

[0011] Figure 4 shows an overview of a logical communication path between application and a VM.

[0012] Figure 5 shows a client host initiating a connection with a VM.

[0013] Figure 6 shows a hypervisor host handling a packet from client host.

DETAILED DESCRIPTION

[0014] Embodiments discussed below relate to using internal communication channels on a VM/hypervisor host to allow external network communication. Discussion will begin with an overview of virtualization technology and virtualization layers (to also be referred to as hypervisors). An example of network communication between an application and a VM will be described next. An overview of a logical communication path using private channels on a hypervisor host will be explained. Finally, details of such a communication path will be described in detail, including an application at one end of the communication path and a hypervisor host (VM host) at another end of the communication path.

Machine Virtualization

[0015] Figure 1 shows an example virtualization layer 100. A computer 102 has hardware 104, including a central processing unit (CPU) 106, memory 108, a network interface 110, non-volatile storage 112, and other components not shown, such as a bus, a display adapter, etc. The virtualization layer 100 manages and facilitates execution of virtual machines 114. Although not shown in Figure 1, each virtual machine 114 typically has an associated virtual disk image and a guest operating system. For brevity, the operating system and perhaps application software of a virtual machine 114 will sometimes be referred to as a guest, which is stored and executed from the virtual disk image associated with the virtual machine 114. For convenience, the term “hypervisor” will be used herein to refer to the various forms of virtualization layers. Moreover, as will be discussed below, virtual machines 114 are used to host elements of distributed applications.

[0016] The virtualization layer 100 may be of any variety of known or future implementations, such as Hyper-V Server (TM), VMWare ESX Server (TM), Xen, Oracle VM (TM), etc. The architecture of the virtualization layer may a hosted type, with a virtual machine monitor (VMM) running on a host operating system, or a bare-metal type with a hypervisor or the like running directly on the hardware 104 of the computer 102. As used herein, the term “virtual machine” refers to a system-type virtual machine that simulates any specific hardware architecture (e.g., x86) able to run native code for that hardware architecture; to the guest, the virtual machine may be nearly indistinguishable from a hardware machine. Virtual machines discussed herein are not abstract or process-type virtual machines such as Java Virtual Machines.

[0017] The virtualization layer 100 performs the basic function of managing the virtual machines 114 and sharing of the hardware 104 by both itself and the virtual machines 114. Any of a variety of techniques may be used to isolate the virtual machines 114 from the hardware 104. In one embodiment, the virtualization layer may provide different isolated environments (i.e., partitions or domains) which correspond to virtual machines 114.

Some of the virtualization layer 100 such as shared virtual device drivers, inter virtual machine communication facilities, and virtual machine management APIs (application programming interfaces), may run in a special privileged partition or domain, allowing for a compact and efficient hypervisor. In other embodiments, functionality for virtual machine management and coherent sharing of the hardware 104 may reside in a monolithic on-the-metal hypervisor.

[0018] Figure 2 shows processes and interactions of virtualization layer 100 in relation to virtual machines 114 and virtual machine images 140. The virtualization layer 100 performs a process 142 of starting and executing a virtual machine 114, possibly according to corresponding virtual machine configuration parameters. When a virtual machine 114 (VM) is started, the virtualization layer identifies an associated virtual machine image 140. In practice, any virtual machine image 140 can be used by any virtual machine 114. The virtual machine image 140 may be a specially formatted file (e.g., a VHD) on a file system 141 of the virtualization layer 100. The virtualization layer 100 loads the identified virtual machine image 140. The started virtual machine 114 mounts and reads the virtual machine image 140, perhaps seeking a master boot record or other boot information, and boots a guest operating system which begins executing.

[0019] The virtualization layer 100 manages execution of the virtual machine 114, handling certain calls to the guest's kernel, hypercalls, etc., and coordinating the virtual machine 114's access to the underlying hardware 104. As the guest and its software run, the virtualization layer 100 may maintain state of the guest on the virtual disk image 140; when the guest, or an application run by the guest, writes data to "disk," the virtualization layer 100 translates the data to the format of the virtual disk image 140 and writes to the image.

[0020] The virtualization layer 100 may perform a process 144 for shutting down the virtual machine 114. When an instruction is received to stop the virtual machine 114, the state of the virtual machine 114 and its guest is saved to the virtual disk image 140, and the executing virtual machine 114 process (or partition) is deleted. A specification of the virtual machine 114 may remain for a later restart of the virtual machine 114.

Overview of Communication to a Virtual Machine

[0021] Figure 3 shows an example of an application 180 communicating with an agent 182 running on a guest operating system (guest 184), hosted by a VM 186. The application 180, which may be a management application, for example, runs on a client host 188, which may be an ordinary computer with a network interface card (NIC) 189 to allow communication via a network 190. The client host 188 has a protocol stack comprised of various protocol implementations, including an application protocol implementation 192 (implemented by the application 180), a transport protocol implementation 194, and a network protocol implementation 196.

[0022] The guest 184 also has implementations of the above-mentioned protocols, as does hypervisor 196 on hypervisor host 198. The hypervisor host 198 is a computer running the hypervisor 196, which manages execution of the VM 186. The agent 182 (also referred to as "guest agent 182") resides on the guest 184 and may implement the same application protocol implemented by the application 180. The application 180 and guest agent 182 may be any variety of software, for instance background network services, interactive applications, executables, components of larger applications or suites, and so forth. In one embodiment, the application 180 is a virtual machine management application that manages VMs, and the agent 182 performs management functions in accordance with communications with the application 180.

[0023] Execution of the VM 186 is managed by the hypervisor 196, which may manage other VMs not shown in Figure 3. In a case where direct connectivity between client host 188 and VM 186 is possible, the application 180 and agent 182 communicate via the network 190 as follows. The application 180 forms an application message according to the application protocol 192 (e.g., an HTTP packet or message). The application 180 requests its local operating system to send the message to the network address (e.g., HTTP address) of the hypervisor host 198. The protocol stack of the local operating system opens a connection to the hypervisor host 198, encapsulates the application 180's message in a transport payload and the transport payload within a network packet 202. The network header thereof (which contains the network address of the hypervisor host 198) is routed through the network 190 to the hypervisor host 198. The hypervisor host 192 may pass the packet 202 to the VM 186 and in turn to the guest 184 and guest agent 182. Along the way, various payloads are extracted by the respective protocol implementations, and the guest agent 182 receives the transmitted application message (e.g., "command foo"). The process is similar but reversed when the guest agent 182 transmits an application message to the application 192.

[0024] As used herein, the terms "client," "client host," "application," and "agent," "hypervisor," and "hypervisor host" are used in their broadest senses. The particular platforms and software that communicate using techniques described herein are of minor significance. In fact, it may be notable that existing application-level software and protocols may use the communication techniques described below without significant modification (if any), in particular at the end that is communicating with a VM via a network (e.g., application 180). Moreover, while the HTTP, IP (Internet Protocol), and TCP/UDP (Transmission Control protocol/Universal Datagram Protocol) protocols are sometimes mentioned for illustration, the communication techniques described below may work with any standard networking protocols or versions thereof (e.g., SOCKS). Furthermore, for brevity, "HTTP," will is deemed to refer to versions or variants of HTTP as well as HTTPs (HTTP Secure).

Logical Communication Path, Application and Hypervisor Embodiments

[0025] Figure 4 shows an overview of a logical communication path 220 between application 180 and VM 186. A proxy agent 222 on the hypervisor host 198 bridges the client host 188 with the VM 186. It may be assumed that the VM 186 and the client host 188 have the necessary networking components (e.g., protocol stacks) to communicate, but are unable to communicate directly. For example, the network 190 may be unable to route network packets between them (e.g., either may be unaddressable on the network 190). However, at the client host 188, a network packet 223 may be directed to a network address of the virtualization layer (an address of the hypervisor host 198). When the packet 223 is received, the proxy agent 222 determines that the packet 223 is meant to be received by the VM 186 and causes the virtualization layer (hypervisor) to pass the packet through a private or local communication channel 224 to the VM 186.

[0026] In one embodiment, a VM-host map 226 contains information indicating which VMs reside on which hypervisor hosts. The client host 188 may use a known identifier of the VM 186 (possibly known by the application 180) to look up the network address of the hypervisor host 198 in the VM-host map 226. The identifier may be added to the packet 223 for use by the proxy agent 22. The client host 188 sends the packet 223 to the looked-up network address of the hypervisor host 198, which the network 190 uses to route 226 the packet 223 to the hypervisor host 198. As mentioned above, the proxy agent 222 uses the identifier of the VM 186 (e.g., from an HTTP CONNECT header) to cause the virtualization layer to pass the packet 223 to the VM 186.

[0027] Figure 5 shows client host 188 initiating a connection with VM 186. The client host 188 has access to the VM-host map 226. The client host 188 performs a process 250 for communicating with the VM 186. At step 252, a request 254 is received to communication with the particular VM 186, perhaps as part of the logic of application 180, or possibly received from an external entity. At step 256 an identifier of the VM 186 (e.g., "VM1" in Figure 5) is used to lookup the host on which VM 186 resides; hypervisor host 186. The lookup may return a network address or network hostname of the hypervisor host 198 (as used herein "network address" is assumed to include both numeric addresses as well as hostnames that can be resolved to numeric addresses). At step 258 packet 223 is formed, including the substantive payload (e.g., an application-protocol message). In an embodiment where HTTP is used, the packet 223 formed at step 258 is an HTTP packet and the hypervisor host's network address is included in the HTTP header. At step 260 the packet 223 is transmitted to the network 190 using the hypervisor's network address ("network addr1" in Figure 5," for example "128.1.2.3").

[0028] In one embodiment, process 250 may be performed fully or partly by the application 180. In another embodiment, the application 180 may act as a proxy or service for other applications that are to communicate with VM 186. Those applications pass the application 180 a VM identifier and a message body and the application 180 builds a packet for the body, adds the VM identifier, and transmits the packet to the corresponding hypervisor host. In yet another embodiment, rather than maintaining a lookup table (VM-host map 226), the VM identifiers may be globally unique hostnames registered with a DNS (Domain Name Service) server (possibly with local or limited scope to avoid conflicts) that maps to the network addresses of the hypervisor hosts that correspond to the VMs. In which case, when an application or client host wishes to communicate with a VM, it looks up the VM's identifier (e.g., a "fake" DNS name) via the local DNS server to obtain the correct hypervisor host's network address.

[0029] The form of the VM identifiers is not important as long as the proxy agent 222 and the client hosts/applications share the same names. The system may use any convention for naming VMs, for example a custom URI (Universal Resource Identifier) format such as "host#:vm#."

[0030] Figure 6 shows hypervisor host 198 handling packet 223 from client host 188. The proxy agent 222 operates in conjunction with the virtualization layer or hypervisor 192 to perform process 280. At step 282 the hypervisor host 198 receives packet 223, which includes the VM identifier (e.g., "VM1," "host1:VM1," etc.) of the target VM; VM 186. The proxy agent 222 extracts the VM identifier, and at step 283 looks up the VM identifier in a channel table 284. The channel table 283 maps communication channels 224, 224A to respective VMS 186, 186A. Each communication channel 224, 224A may have a pair of communication end points, including a hypervisor-side end point 286 and a VM-side end point 288 (in one embodiment the VM-side end point 286 is a virtual NIC of the VM 186). If step 283 discovers no communication channel, a step 290 is performed and a new channel is created. A reference to the new channel is then added to the channel table 284. Note that a virtual NIC, or management NIC, may be a virtual bus network adapter connected to an internal network switch to which the host and VMs are connected; VMs having automatic internal IP addresses (e.g., in the 169 range). In one embodiment, an ACL (access control list) is maintained in association with the internal network switch to prevent each of the guest VMs from communicating with each other without permission; host-to-VM communication is allowed but VM-VM communication is disallowed absent explicit permission in the ACL.

[0031] Having identified the correct communication channel 224 for the packet 223, at step 292 the hypervisor 192 and/or proxy agent 222 passes the packet 223 to communication channel 224. In turn, the VM 186 performs process 294. At step 296 the guest 184 receives the packet 223. At step 298, based on the packet 223, the guest 184 passes the packet 223 to the guest agent 182, which proceeds to service the substantive content of the packet 223 (e.g., executes "command foo"). For example, the application-protocol type of the packet 223 (e.g. HTTP) might be mapped by the guest to a port number that the guest agent 182 listens on. That is, the guest agent 182 may listen on specific ports (e.g., a designated WS-Management port 5986 for management, a BITS port 8114 for data, etc.). The proxy agent 222 (e.g., HTTP proxy) listens on the same ports (5986, 8114) of the external IP addresses (of the hypervisor host 186). The proxy agent 222 then forwards any incoming traffic onto the corresponding ports in the guest VMs, thus allowing multiplexing of various control and data traffic onto the guest VMs.

[0032] Regarding the communication channels, in one embodiment the communication channels are based on the same network and/or transport protocols used to deliver the packet 223 to the hypervisor host 198. In effect, a communication channel may be a limited private network connection between the hypervisor host 192 and the VM 186. In another embodiment, the proxy agent 222 may look into the packet 223 and change header information or otherwise modify the packet 223 before forwarding the packet 223 to the VM 186.

[0033] In effect, from the viewpoint of application 180 and guest agent 182, the two are able to exchange application-level communications using ordinary network communication protocols and addresses, much as they might when a VM is directly accessible or addressable from the client host 188. With regard to receiving packets, the proxy agent 222 functions much as known proxies function.

[0034] As mentioned above, communications may also originate from within the guest or guest 184 agent 182 and be passed through the communication channel and virtualization layer to the proxy agent 222. For example, the guest agent 182 may be configured with a network address of client host 188. The proxy agent 222 in turn transmits the guest-originated packet to the client host 188.

[0035] In one embodiment, in order to provide visibility of VMs, hypervisor hosts (hosts on which VMs execute) may self-register a VM when a VM is created. For example, a hypervisor host might add a new entry to VM-host mapping table 226 for a new VM, identifying the host and the new VM.

Conclusion

[0036] Embodiments and features discussed above can be realized in the form of information stored in volatile or non-volatile computer or device readable media. This is deemed to include at least media such as optical storage (e.g., compact-disk read-only memory (CD-ROM)), magnetic media, flash read-only memory (ROM), or any current or future means of storing digital information. The stored information can be in the form of machine executable instructions (e.g., compiled executable binary code), source code, bytecode, or any other information that can be used to enable or configure computing devices to perform the various embodiments discussed above. This is also deemed to include at least volatile memory such as random-access memory (RAM) and/or virtual memory storing information such as central processing unit (CPU) instructions during execution of a program carrying out an embodiment, as well as non-volatile media storing information that allows a program or executable to be loaded and executed. The embodiments and features can be performed on any type of computing device, including portable devices, workstations, servers, mobile wireless devices, and so on.

CLAIMS

1. A method of providing network connectivity between a target virtual machine (VM) running on a first host and an application running on a second host, the method comprising:

receiving, via a network at the first host, a connection request initiated by the application, the request comprising information identifying the target VM;

forming a communication channel between a hypervisor managing the VM and the target VM; and

passing the request to the target VM through the channel.

2. A method according to claim 1, wherein the communication channel is accessible to only the hypervisor and the target VM such that another VM on the first host cannot use the communication channel to communicate with the target VM without authorization.

3. A method according to claim 1, wherein the packets comprise Internet Protocol packets, wherein the first host has a first Internet Protocol (IP) address, the second host has a second IP address, and the target VM has a third IP address, wherein the network is unable to route packets from the second IP address of the second host to the third IP address of the VM.

4. A method according to claim 3, further comprising sending packets from the second host to the first host by the second host addressing the packets to the first IP address.

5. A method according to claim 4, wherein the packets comprise payloads conforming to an application-level protocol, the method further comprising delivering the payloads to an agent executing on the VM after the packets have been received by the VM through communication channel.

6. A method according to claim 1, wherein the communication channel comprises an internal network switch of the hypervisor and a virtual network interface card provided by the hypervisor and assigned to the target VM.

7. A method according to claim 1, wherein the hypervisor and the proxy agent on the first host cooperate to allow the second host to communicate with the target VM using a standard network protocol.

8. One or more computer-readable storage media storing information to enable a computer to perform a process, the process comprising:

executing a virtual machine (VM) and a guest operating system on the VM, wherein the VM comprises an implementation of a network-level protocol and an implementation of a transport protocol, and wherein execution of the VM is managed by a hypervisor on the computer;

executing a proxy agent on the computer; and

receiving packets at the computer, the packets conforming to a protocol and having been addressed and sent to a network address of the computer using a network protocol, wherein the proxy agent determines that the VM is to receive the packets, and in response the hypervisor passes the packets to the VM through a private communication channel (224) between the VM and the hypervisor.

9. One or more computer-readable storage media according to claim 8, wherein the VM implements the network protocol and has a network address according to the network protocol, a host that transmitted the packets to the computer via the network does not have direct connectivity, via the network and the network protocol, with the network address of the VM.

10. One or more computer-readable storage media according to claim 8, the process further comprising maintaining, at the computer, channel information indicating which VMs correspond to which VMs managed by the hypervisor.

11. A method wherein a first host comprises a virtualization layer managing execution of virtual machines (VMs) on the first host, the method comprising:

establishing a communication channel between the virtualization layer and a VM comprising one of the VMs on the first host, the communication channel having a first end point in the virtualization layer and a second end point in the VM;

receiving packets at the first host from a second host via a network; and

enabling indirect network connectivity between the second host and the VM by a proxy agent executing on the first host causing the virtualization layer to pass the packets through one of the communication channels to the VM.

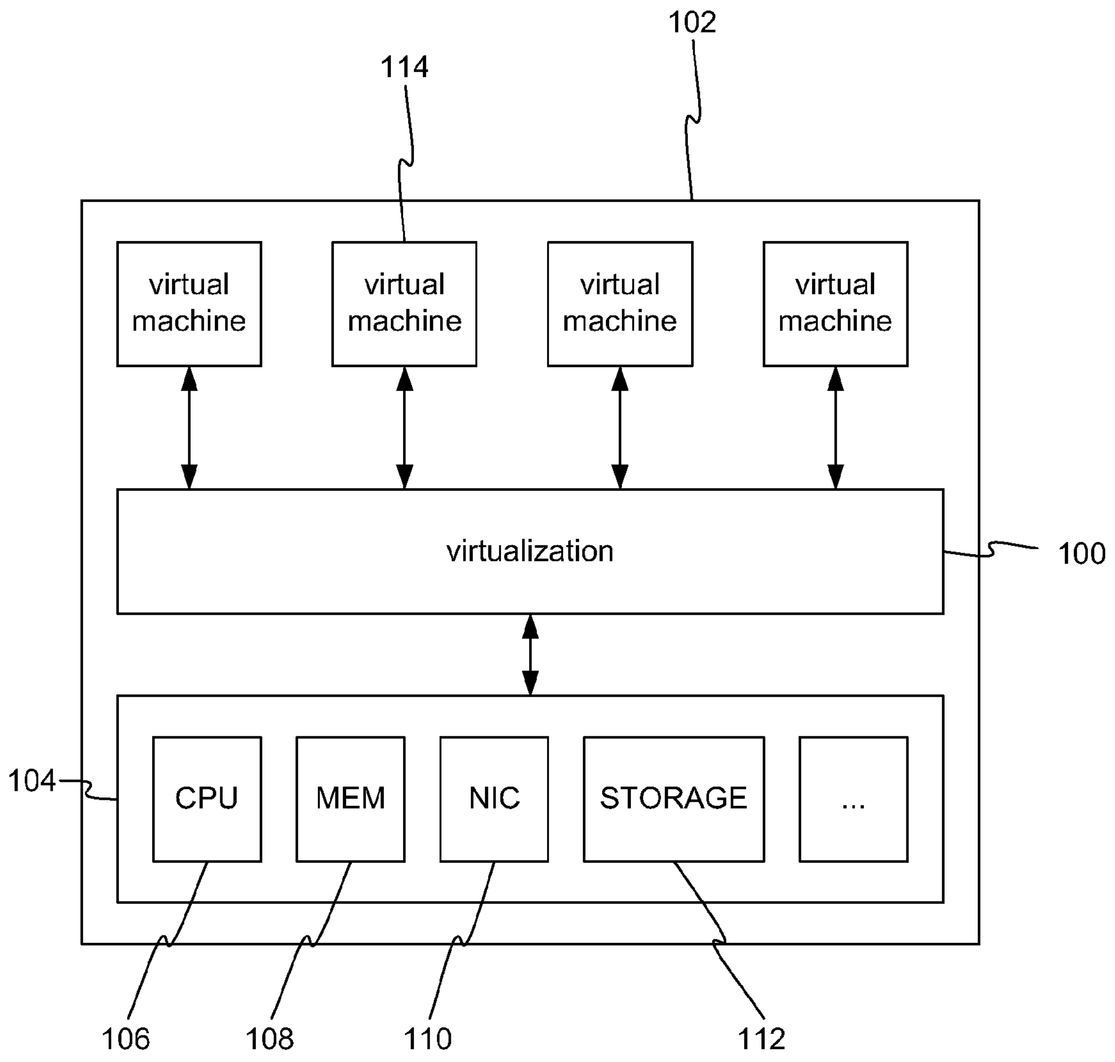
12. A method according to claim 11, the method further comprising maintaining association information associating communication channels between the virtualization layer and the VMs, respectively, and when a packet is handled by the proxy agent the proxy agent uses the association information to select a communication channel that will carry the packet to one of the VMs.

13. A method according to claim 12, wherein the proxy agent comprises a hypertext transfer protocol (HTTP).

14. A method according to claim 13, wherein the packets comprise HTTP headers that comprise identifiers that identify the VMs, and the proxy agent reads the HTTP headers,

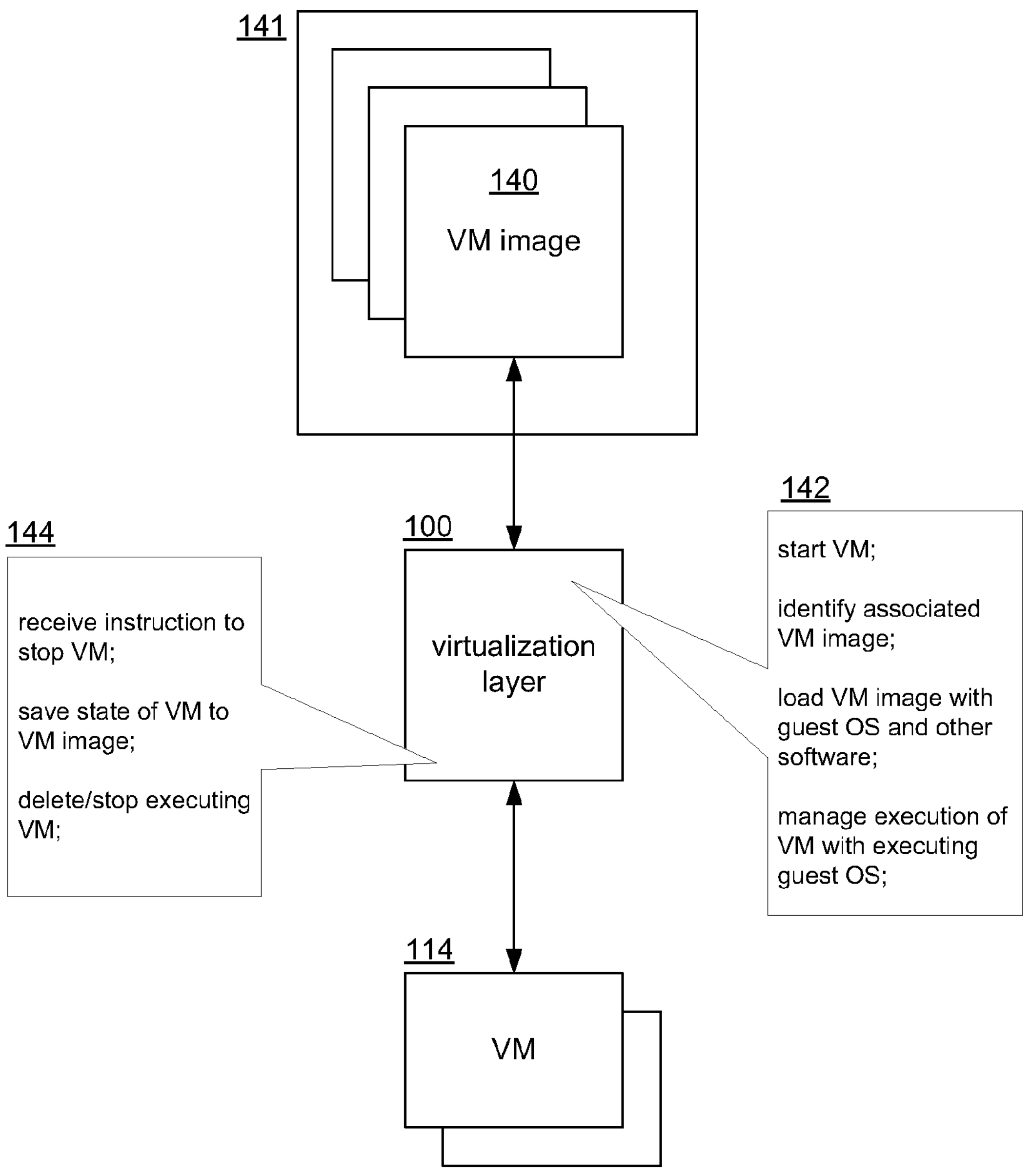
extracts the identifiers, and uses the identifiers to select which channels to receive the packets.

15. A method according to claim 11, wherein a guest operating system of the VM implements a transport protocol and a network protocol to allow connectivity to the guest operating system via a network address of the guest, wherein the second host is unable to directly connect to the VM using the network protocol and the network address via the network, the wherein the method further comprises the second host addressing the packets to a network address of the first host.



RELATED ART

FIG. 1


141
140
VM image
100
virtualization layer
142
start VM;
identify associated VM image;
load VM image with guest OS and other software;
manage execution of VM with executing guest OS;
144
receive instruction to stop VM;
save state of VM to VM image;
delete/stop executing VM;
114
VM

RELATED ART

FIG. 2

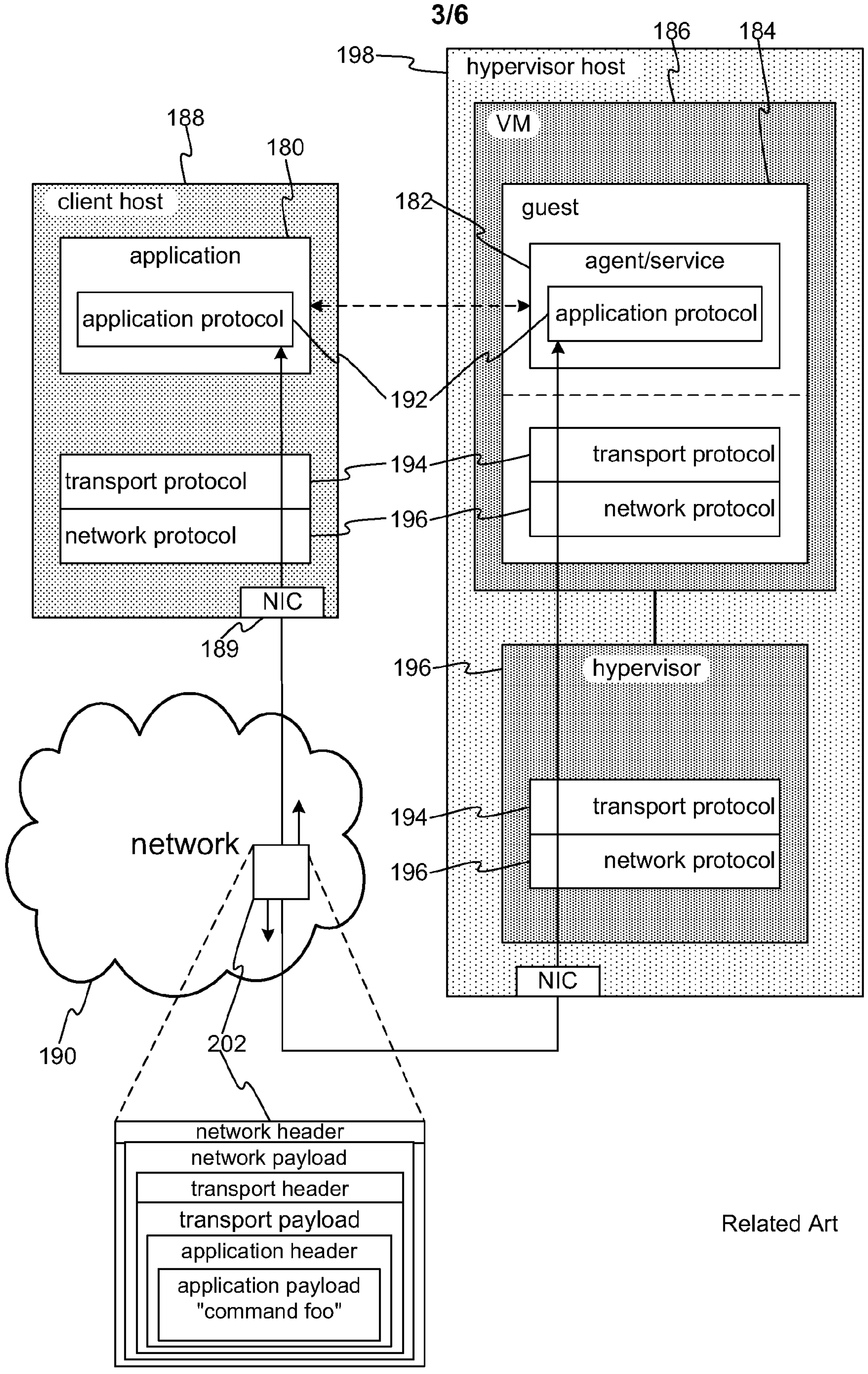

198
hypervisor host
186
184
VM
188
180
client host
guest
182
agent/service
application
application protocol
application protocol
192
transport protocol
transport protocol
194
network protocol
network protocol
196
NIC
189
196
hypervisor
network
transport protocol
194
network protocol
196
NIC
190
202
network header
network payload
transport header
transport payload
application header
application payload
"command foo"
Related Art

FIG. 3

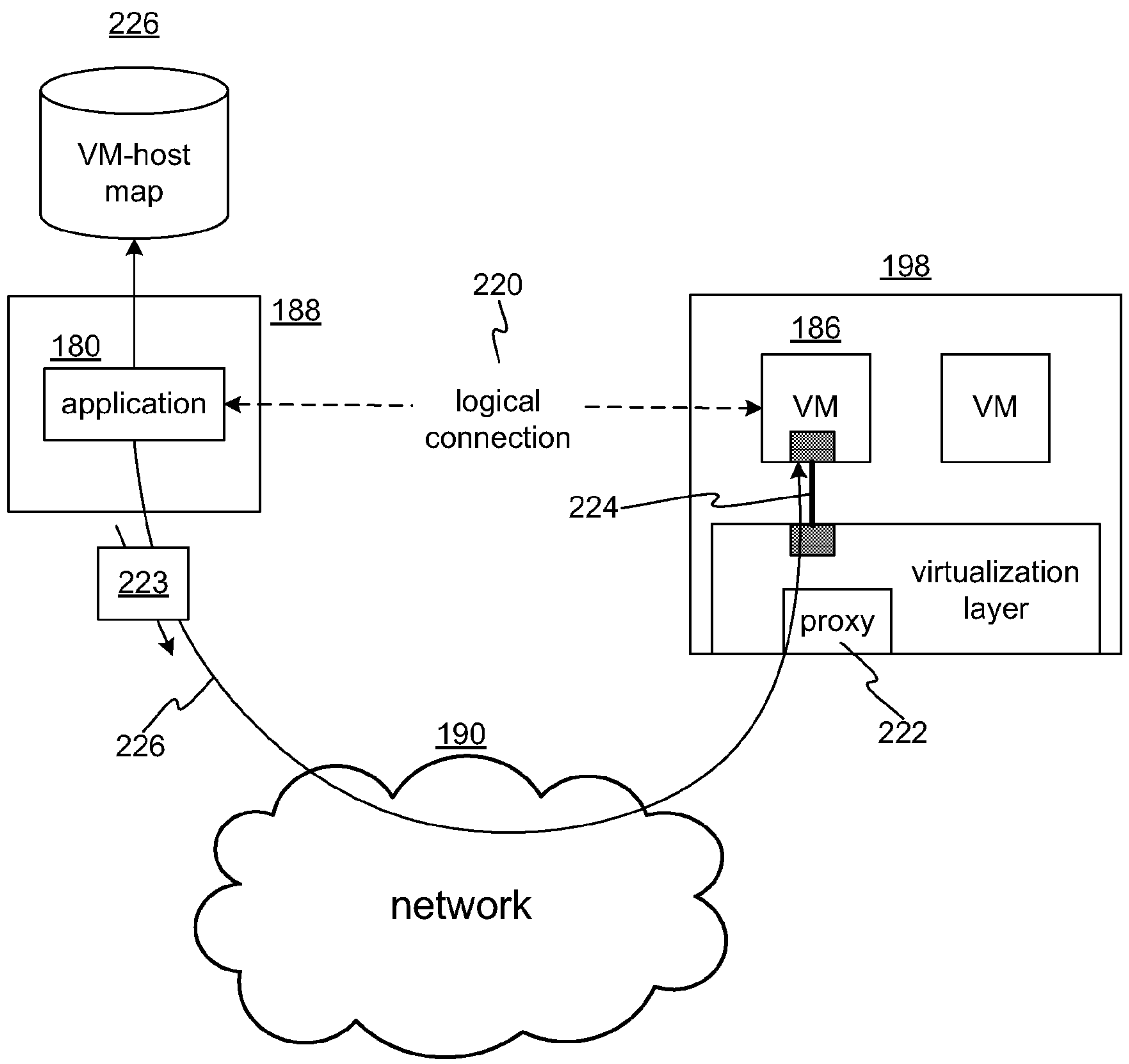

226
VM-host map
188
180
application
220
logical connection
198
186
VM
VM
224
virtualization layer
proxy
222
223
226
190
network

FIG. 4

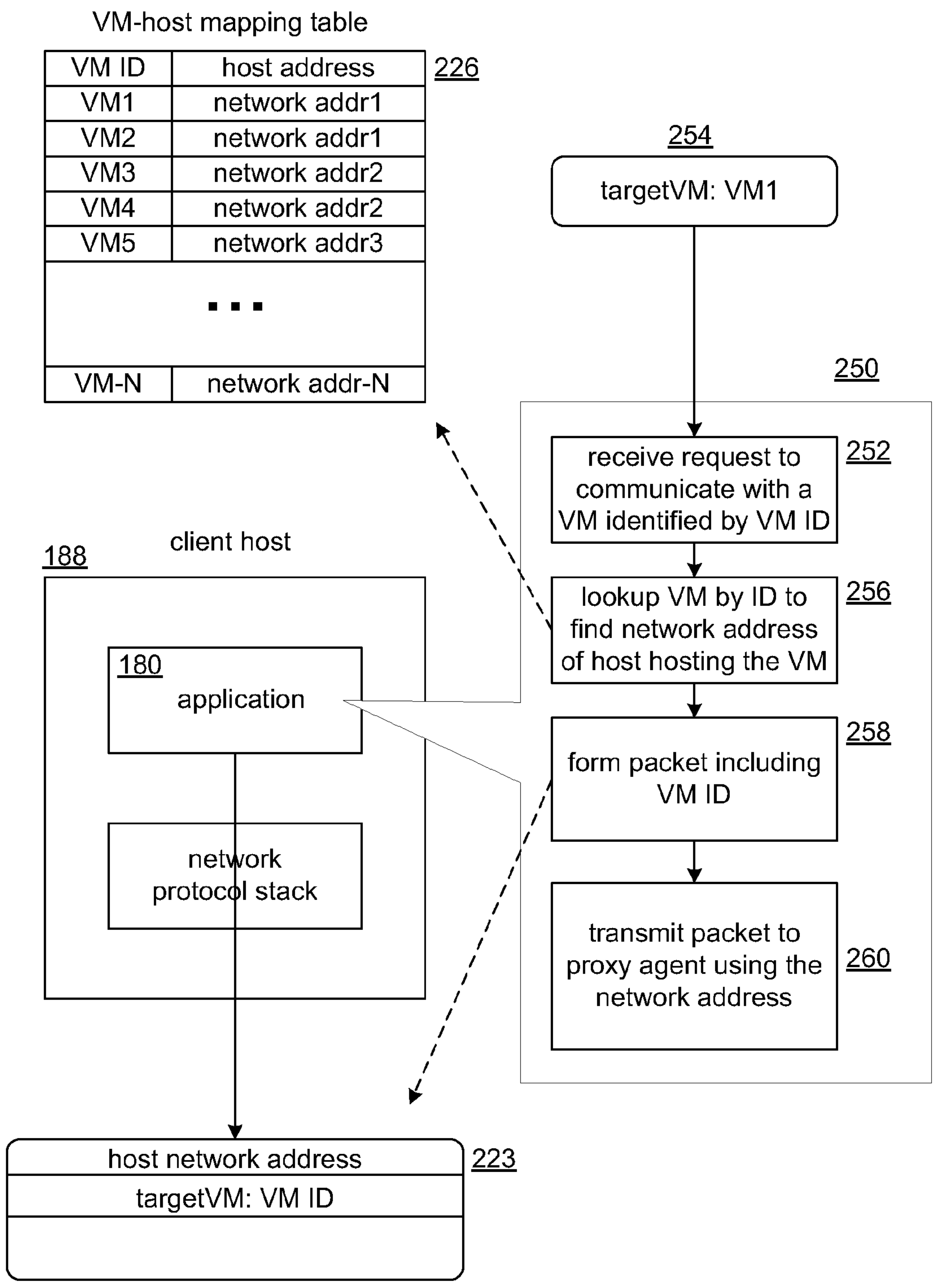

VM-host mapping table
VM ID
host address
VM1
network addr1
VM2
network addr1
VM3
network addr2
VM4
network addr2
VM5
network addr3
VM-N
network addr-N
226
254
targetVM: VM1
250
receive request to communicate with a VM identified by VM ID
252
lookup VM by ID to find network address of host hosting the VM
256
form packet including VM ID
258
transmit packet to proxy agent using the network address
260
client host
188
180
application
network protocol stack
host network address
targetVM: VM ID
223

FIG. 5

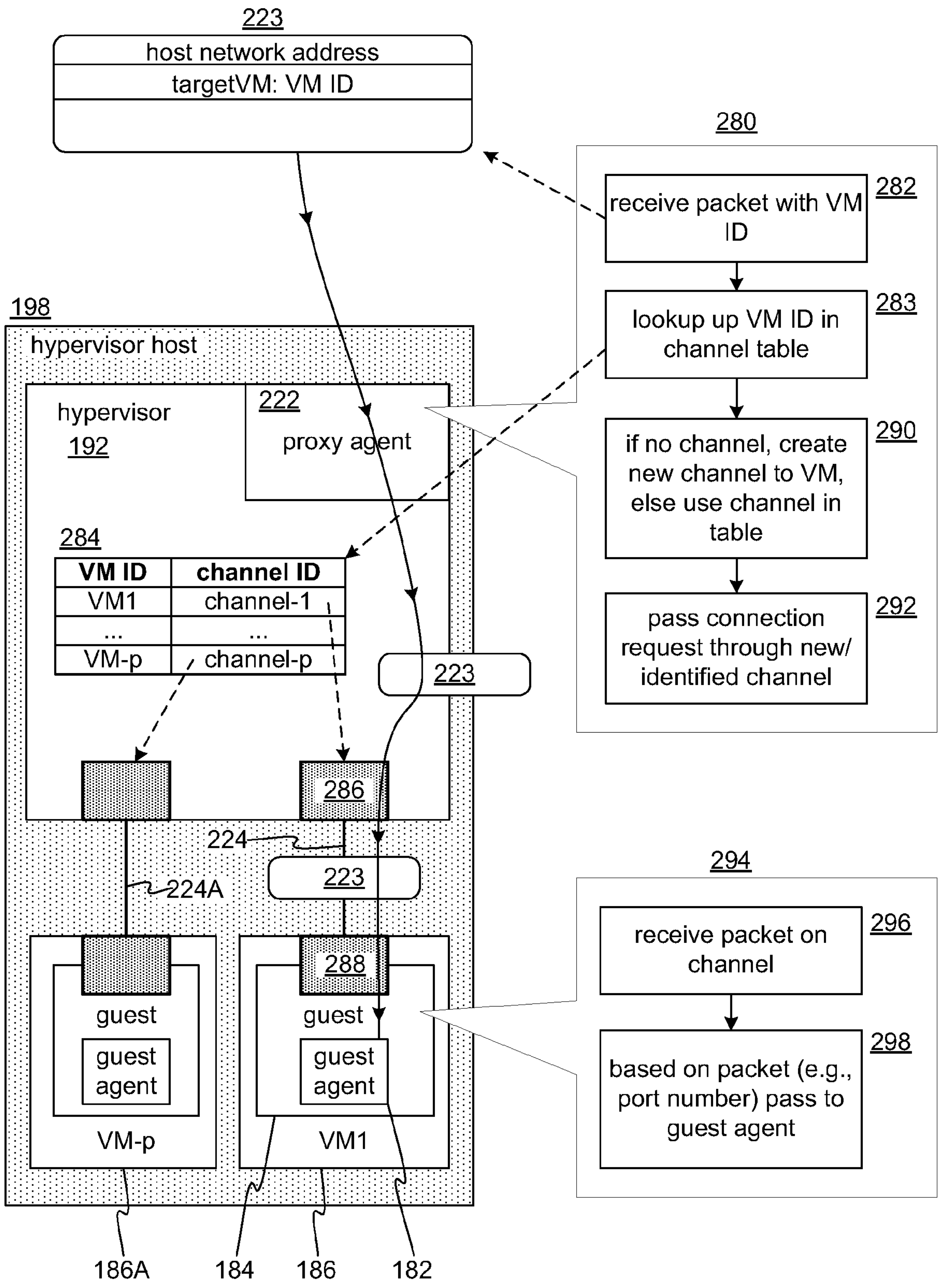

223
host network address
targetVM: VM ID
280
receive packet with VM ID
282
lookup up VM ID in channel table
283
if no channel, create new channel to VM, else use channel in table
290
pass connection request through new/ identified channel
292
198
hypervisor host
hypervisor
192
222
proxy agent
284
VM ID
channel ID
VM1
channel-1
...
...
VM-p
channel-p
223
286
224
224A
223
288
guest
guest agent
VM-p
guest
guest agent
VM1
294
receive packet on channel
296
based on packet (e.g., port number) pass to guest agent
298
186A
184
186
182

FIG. 6