A Method for Managing a Data Center Network

Field of the invention

The present invention relates to a method for managing a data center network (DCN), a method for communicating through the DCN, a method for managing VM migration (VMM) in the DCN and an apparatus comprising the DCN.

Background of the Invention

Traditionally, VMs have been managed based on their network addresses (IP addresses) on DCNs. This can lead to several issues, for example, issues arising during VMM which has been touted as one of the crucial technologies in improving the efficiency in DCNs via for example, reducing energy cost, maintaining load balance, etc.

One of the challenges imposed by VMM is the IP mobility problem which arises when a VM migrated to another location is assigned with a different IP address. This causes previously established applications bound to the old IP address to become invalid. This IP mobility problem exists in both live and offline VMMs and is difficult to avoid. For example, when performing a live VMM, to avoid the IP mobility problem which can cause connection interruptions, the VM has to maintain the same IP address after it is migrated so as to achieve a service downtime that is unnoticeable by the user. However, it is extremely challenging to migrate a VM to a different subnet without changing its IP address. This is because normal network operations usually would not allow non-continuous IP assignment. When performing an offline VMM, the VM is first shut down and then assigned a new IP address at the destination location. To avoid the IP mobility problem due to this change in the IP address of the VM, a manual reconfiguration of the network settings in all related VMs is required. Such a manual process is labor-intensive, time-consuming and error-prone, especially

for the larger multi-tier applications, since the IP addresses are involved in various network, security and performance policies.

Traditional methods for addressing the IP mobility problem in VMM are mostly based on centralized solutions which often result in higher overhead and proneness to failures. Therefore, there exists a need to develop an efficient and robust method to manage VMs on DCNs so that seamless VMM can be achieved.

Summary of the invention

The present invention aims to provide a new and useful method for managing a DCN comprising a hierarchy of nodes.

As is well known in the art, a DCN is a communication network having a hierarchy of nodes (or tree topology) that interconnects a pool of resources such as computational, storage, network resources etc., whereas a VM is a software implementation of a computer, operating based on computer architecture (like a real computer) on which an operating system or program can be installed and run.

In the present invention, the DCN is configured to provide at least one service. The at least one service is associated with one or more VMs in the DCN. Requests and responses for the at least one service are respectively relayed downstream to and upstream from the one or more VMs through the nodes. Each node comprises information for facilitating routing of the requests and responses through the nodes.

In general terms, the present invention proposes that the above-described DCN is managed based on the services associated with the VMs on the DCN, rather than based on their network addresses.

Specifically, a first aspect of the present invention is a method for managing the above-described DCN, the method comprising: sending a control message from a VM through the nodes of the DCN, the control message indicating events related to the service associated with the VM; and at each node through which the control message is sent, updating the information based on the control message.

This method is advantageous as it decouples the services running on the VMs from the network addresses of the physical machines the VMs are on. As such, the services can be accessed by a client without knowledge of the network addresses where the VMs are hosted. With this, a more seamless VMM can be achieved.

A second aspect of the present invention is an apparatus comprising the above-described DCN wherein the DCN is managed by the method according to the first aspect of the present invention.

Brief Description of the Figures

An embodiment of the invention will now be illustrated for the sake of example only with reference to the following drawings, in which:

Fig. 1 illustrates a network according to an embodiment of the present invention, wherein the network comprises a DCN;

Fig. 2 illustrates two types of packets in a named-service format for communication through the DCN;

Fig. 3 illustrates entries in two data structures maintained by each node of the DCN;

Fig. 4 illustrates a flow diagram of a method for managing each VM on the DCN;

Fig. 5 illustrates a flow diagram of a method used by a node in the DCN for handling a client request;

Fig. 6 illustrates a flow diagram of a method used by a node in the DCN for processing a negative acknowledgement packet;

Fig. 7 illustrates a flow diagram of a method used by a node in the DCN for handling a response from a VM on the DCN; and

Fig. 8 illustrates a VMM strategy according to an embodiment of the present invention.

Detailed Description of the Embodiments

Network 100

Fig. 1 illustrates a network 100 according to an embodiment of the present invention. The network 100 comprises a data center 102 which in turn comprises a plurality of switches (namely, core switches, aggregation switches, ToR switches) and VMs. The core switches are upper layer switches of the aggregation switches and the aggregation switches are upper layer switches of the ToR switches. The data center is configured to provide at least one service, with each service associated with (more specifically, provided by) a VM in the data center 102. A particular service may be provided by one or more VMs in the data center 102. The network 100 further comprises a plurality of service gateways 104, the Internet 106 and Clients 108.

As shown in Fig. 1, there are two types of traffic flow in the network 100: 1) external traffic 110 i.e. traffic flowing between the Clients 108 and the VMs in the data center 102 and 2) internal traffic 112 i.e. traffic flowing from one VM to another VM in the data center 102. Both the external and internal traffic 110, 112 flow through the core switches, aggregation switches and ToR switches in the data center 102 to and from the VMs.

Communication through the network 100 is carried out based on a named-service framework. The framework comprises two components (i) a named service within the data center 102 and (ii) the service gateways 104.

Within the data center 102, communication with the VMs is carried out via a DCN having a multi-rooted tree topology comprising a plurality of nodes arranged in a hierarchical manner. In this document, an "upstream node" and a "downstream node" is defined such that for a particular node, its "upstream node" is its neighbouring node in the direction opposite to the request flow, whereas its "downstream node" is its neighbouring node in the direction along the request flow. Each switch in the data center 102 serves as a node in the tree topology of the DCN. "Upward interfaces" of a switch (node) refer to the interfaces connecting the switch (node) to the upper layer of switches (upper layer nodes) whereas "downward interfaces" of a switch (node) refer to the interfaces connecting the switch (node) to the lower layer of switches (lower layer nodes). A "layer" of switches is defined such that each switch in one "layer" has a direct link to a switch in the next "layer". Hence, each core switch in the upper-most layer has a direct link to an aggregation switch in the next layer and each aggregation switch in the next layer has a direct link to a ToR switch in the lowest layer. Two or more switches in one "layer" can be connected to the same switch in the next "layer".

As will be discussed below with reference to Fig. 5, in some situations, a switch receiving a *Request* packet may not be able to reach a VM providing the service for the *Request* packet. In these situations, the switch forwards the *Request* packet to one of its upward interfaces connected to an upper layer switch (see steps 506, 508, 510, 514). In other words, in these situations, even though the switch is a lower layer switch, it serves as the upstream node of the upper layer switch it forwards the *Request* packet to.

The named service runs a named-service routing protocol for communication through the DCN to, for example, render service for requesting and supporting VMM. This named-service routing protocol processes data in a named-service format. Requests of the internal traffic 108 are directly presented in the named-service format.

The Internet 106 serves as a medium for the external traffic 110 between the Clients 108 and the VMs in the data center 102. However, traffic through the Internet 106 is usually IP-based. The service gateways 104 are thus configured to translate the IP-based traffic from the Internet 106 into the named-service format to allow such traffic to be processed through the DCN within the data center 102. The service gateways 104 are also configured to translate traffic in the named-service format from the DCN to IP-based traffic to allow such traffic through the Internet 106 to the Clients 108.

In the named-service framework, each VM is associated with a service name indicating the service provided by the VM. Communication with each VM within the DCN is performed using the service name of the VM instead of the network address of the VM. In particular, routing to and from the VM is found by the service name of the VM. In this way, the VMs together with the services provided by the VMs are decoupled from the VMs' locations.

Named-Service Framework

Named-Service Format

As mentioned above, communication through the DCN is based on a named-service routing protocol configured to process data in the named-service format.

Fig. 2 illustrates two types of packets in the named-service format, namely the *Request* packet and the *Response* packet. A request from the Clients 108 is in the form of the *Request* packet whereas a response from the VMs in the data center 102 is in the form of the *Response* packet.

As shown in Fig. 2, both the *Request* and *Response* packets comprise a plurality of fields which are elaborated below.

Both the *Request* and *Response* packets comprise a service name. This field serves as an unique identifier of the service running on the VMs in the DCN.

Both the *Request* and *Response* packets also comprise an indicator in the form of a client nonce which is a random number generated by the Clients 108. This field serves two purposes. Firstly, it serves as a way to detect a duplicate *Request* packet (which will then be discarded). This is done by configuring the packets such that duplicate *Request* packets comprise the same client nonce. Secondly, the client nonce is used to identify the destination of a *Response* packet to allow the *Response* packet to be returned along the reverse path of its associated *Request* packet.

Both the *Request* and *Response* packets also comprise a cache control field which denotes whether the request or response is for dynamic content (such as a search) or for static content (such as a piece of video). This field is set by the Clients 108 sending the request. Each node in the DCN is configured such that upon receiving a *Response* packet, the node decides whether to cache the contents of the *Response* packet based on the cache control field of the *Response* packet.

The *Response* packet further comprises a sequence number. This field is created by the service running on the VM sending the *Response* packet and indicates the number of *Response* packets left to be sent by the VM. In particular, the sequence number of the last *Response* packet to be sent is zero. As mentioned above, all *Response* packets are returned along the reverse path of their respective associated *Request* packets. Further, the *Response* packets are received by each node of the DCN in the same order as that in which the node received the associated *Request* packets. The named-service routing protocol of this embodiment is configured such that it is not necessary to include the sequence number in the *Request* packet.

Structure of the DCN

As mentioned above, within the data center 102, communication with the VMs is carried out via a DCN having a multi-rooted tree topology comprising a plurality of nodes. Each node of this DCN comprises information for facilitating routing of requests and responses through the nodes. In particular, each node maintains three data structures, namely the Forwarding Information Base (FIB), the Pending Request Table (PRT) and the Cache Store.

Fig. 3 shows the structure of an entry in the FIB (FIB entry). As shown in Fig. 3, each FIB entry corresponds to a particular service provided by a VM in the DCN and comprises a service name ("name") of the particular service. Each FIB entry also comprises multiple outgoing interfaces ("interface") associated with the service name. Each interface in the FIB entry indicates a downward interface through which a VM providing the service (indicated by the service name) can be reached. Each FIB entry further comprises a capacity value ("capacity") associated with each interface for the purpose of a load balancing policy, which will be discussed later. This capacity value represents the total capacity (i.e. the maximum number of requests which can be served per time unit) of all the VMs in the DCN providing the service indicated by the service name and reachable through the associated interface.

Fig. 3 further shows the structure of an entry in the PRT (PRT entry). As shown in Fig. 3, each PRT entry comprises a service name ("name"), multiple indicators in the form of client nonces ("nonce"), incoming interfaces and outgoing interfaces. The PRT entry is used by the routing nodes to maintain an entry for every pending *Request* packet (i.e. *Request* packet which has not been responded to), so that every *Response* packet can be returned to the Clients 108 in the reverse path of its associated *Request* packet.

The Cache Store entry (not shown in Fig. 3) is used by a routing node to cache Response data in a *Response* packet it receives (if the *Cache Control Field* in the *Response* packet denotes that the data is to be cached).

Control Message Protocol

Fig. 4 illustrates a method 400 for managing the DCN based on the named-service framework. The method 400 takes advantage of the multiple paths in the multi-rooted tree topology of DCN. As shown in Fig. 4, the method 400 comprises steps 402 and 404.

In step 402, a VM is configured to send a control message through at least some of the nodes of the DCN whereby the control message indicates events related to a service provided by the VM. For example, the VM may send a *service starting* message indicating a *service starting* event when the VM is a VM allocated for the first time or a destination VM after a VMM. More specifically, this *service starting* message indicates that the VM will start providing its service. The VM may alternatively send a *service stopping* message indicating a *service stopping* event when the VM is about to migrate to another location, or is about to stop its service. More specifically, this *service stopping* message indicates that the VM will stop providing its service. Yet alternatively, the VM may send a *service capacity update* message indicating a *service capacity update* event when the capacity of the VM is about to change. Upon receiving the control message, each node in the DCN is configured to forward the control message to all upper layer nodes directly connected. In other words, the control message is forwarded hop by hop from the lower layer switches to the upper layer switches of the DCN until it arrives at a core switch.

With the control messages, each node is able to learn the service names of multiple VMs in the DCN tree topology which the node is in (with itself as the root node). In particular, in step 404 of method 400, at each node through which the control message from the VM is sent, information is updated based on the control message. More specifically, the FIB at the node is updated as follows. A control message sent by a VM carries the service name and capacity of the VM (a control message with a capacity value of zero is a *service stopping* message). Once a node receives the control message, it looks for a matching FIB entry

with a service name matching exactly the service name carried by the control message and an interface matching the interface the control message is received from.

If the control message received by the node is a *service starting* message and a matching FIB entry is not found, a new FIB entry is created (with its service name, interface and capacity value based on the control message) and added to the FIB at the node if the VM sending the *service starting message* is the only VM providing the service reachable from the node. However, if there are already VMs providing the same service as the VM sending the *service starting message* and reachable from the node i.e. there is already a FIB entry corresponding to this service, instead of creating a new FIB entry, a new interface with an associated capacity value equal to the capacity value carried by the control message is inserted in the corresponding FIB entry.

The following describes what happens if a matching FIB entry is found at the node.

If the control message is a *service stopping* message and the VM sending the control message is the only VM providing the service and reachable from the node, the matching FIB entry is removed. If the control message is a *service stopping* message and the VM sending the control message is not the only VM providing the service and reachable from the node, then only the information associated with the incoming interface of the control message is removed from the matching FIB entry.

If the control message is a *service capacity* update message, the matching FIB entry of the node is updated such that the capacity value associated with the interface, from which the control message is received, is replaced by the capacity value carried by the control message.

Regardless of whether the control message is a *service starting* message, a *service stopping* message or a *service capacity* update message, after updating the FIB at the node, the total capacity of VMs providing the service indicated by the service name in the control message and reachable through the node is calculated from the updated FIB entry, in particular, by taking the sum of the capacities associated with all the interfaces of the updated FIB entry. The capacity value in the control message is then replaced with this calculated total capacity (the capacity value in the control message remains as zero if the matching FIB entry is removed). Then, the updated control message is forwarded to all the upper layer nodes. At each upper layer node, the above steps are repeated.

Named-Service Routing Protocol

The following describes the named-service routing protocol in greater detail. In particular, it describes how a request from the Clients 108 is communicated through the DCN to the VMs and how a response from a VM is communicated through the DCN to the Clients 108.

Handling Request Packets

Fig. 5 illustrates a method 500 used by a node in the DCN for handling a request from the Clients 108. The request is converted to a *Request* packet in the format as shown in Fig. 2 by one of the service gateways 104. In other words, the *Request* packet comprises a service name ("request service name"), a client nonce and a cache control field.

As shown in Fig. 5, the method 500 comprises steps 502 – 514 which will be described in detail below.

In step 502, upon receiving a *Request* packet, the node looks up its FIB and searches for matching FIB entries having a service name matching exactly the

request service name of the *Request* packet. In step 502, the node also looks up its PRT and search for a matching PRT entry with a service name and a nonce respectively matching exactly the request service name and nonce of the *Request* packet. If a matching PRT entry is found, this indicates that the *Request* packet is a duplicate *Request* packet and the *Request* packet is discarded.

The node then communicates with a further node based on the search results for the matching FIB entries, so as to further forward the *Request* packet through the DCN. In particular, if there are matching FIB entries, step 504 is performed. Otherwise, if there is no matching FIB entry, step 506 is performed.

In step 504, the interfaces of the matching FIB entries are obtained as candidate interfaces whereas in step 506, all upward interfaces of the node are obtained as candidate interfaces.

The further node the node is to communicate with is selected based on the candidate interfaces obtained. In particular, in step 508, tried candidate interfaces are filtered from the candidate interfaces to obtain a set of untried interfaces. The tried interfaces are interfaces to which the *Request* packet has been sent and from which a *Request NACK* packet has been received. The *Request NACK* packet will be discussed in greater detail later on.

The further node is selected based on whether there are untried interfaces. Specifically, if there is at least one untried interface, step 510 is performed. Otherwise, if there is no untried interface, step 512 is performed.

In step 510, an interface is selected from the untried interfaces. The selected interface links to the further node. The selection of the interface is done by a load balancing policy which is based on a design consideration to balance load among a set of VMs providing the same service in the DCN. More specifically, the load balancing policy uses the capacity values associated with the untried

interfaces in the corresponding FIB entries. As mentioned before, the capacity value in a FIB entry represents the total capacity (i.e., the maximum number of requests that can be served per time unit) of all VMs providing the service indicated by the associated service name and reachable through the associated interface. It is maintained and updated by the control messages as discussed previously. Moreover, to facilitate the load balancing policy in the upward forwarding, the same capacity value is assigned to all upward interfaces of the node when performing step 506.

The load-balancing policy can be formulated as an optimization problem. Given a request load l for a service (i.e. the number of requests for the service per unit time – this can be statistically determined) arriving at a node, and n available interfaces (i.e. untried interfaces), each of which has a capacity value of c_i, the problem is to distribute the load to each interface such that the maximum utilization on interfaces is minimized, as described in the following Equation (1).

$$\begin{aligned} &\min \ \max_{i=1}^{n} u_i \\ &s.t. \ u_i = l_i / c_i, \qquad i = 1,\ldots,n \qquad (1) \\ &\sum_{i=1}^{n} l_i = l \end{aligned}$$

where the utilization on each interface represents the load on the interface per unit capacity of the interface and is denoted as u_i. Minimization of the maximum utilization on interfaces can be achieved using several approaches, two of which are described below.

Random Choice Algorithm (RCA):

In this algorithm, each untried interface is assigned a probability p_i equal to $c_i / \sum_{i=1}^{n} c_i$. The forwarding interface i.e. the interface to which the *Request*

packet is to be forwarded is modelled as a discrete random variable obeying the distribution of $(p_i,\ldots,p_n)$. Each *Request* packet is forwarded to one of the untried interfaces randomly chosen.

Weighted Round-Robin Algorithm (WRRA):

In this algorithm, each untried interface is assigned a weight w_i equal to $c_i / GCD(c_i,\ldots,c_n)$ where GCD represents the Greatest Common Divisor. The weight of an interface represents the number of requests that can be forwarded by the interface in each round. At the beginning of a round during which a first *Request* packet is forwarded, each interface is assigned a number of tokens equal to its weight. During the round, the *Request* packets are forwarded to the untried interfaces in a circular order until an interface runs out of tokens. The round ends when all the interfaces have run out of tokens.

After selecting an interface from the set of untried interfaces using the load balancing policy, step 514 is performed in which the *Request* packet is forwarded to the selected interface and a new PRT entry is created at the further node the selected interface links to. The new PRT entry comprises the request service name and client nonce of the *Request* packet, an incoming interface indicating the interface from which the *Request* packet is received and an outgoing interface indicating an interface to which the *Request* packet will be forwarded. If the selected interface is not an interface to a VM, method 500 is then repeated by the further node to further forward the *Request* packet.

If there is no untried interface available, this usually indicates that the node is unable to further forward the *Request* packet. A possible reason for this is that using the above-mentioned control message protocol, each node in the DCN learns the service names of the VMs in a tree topology with itself as the root node but does not know the service names out of this tree topology. As

mentioned above, if there is no untried interface available, step 512 is performed.

In step 512, the upstream node from which the *Request* packet is received (i.e. in the direction opposite to the request flow) is selected as the further node. The node sends a negative acknowledgement packet in the form of a *Request NACK* packet back to this upstream node. This is to notify the upstream node to try an alternative interface. The *Request NACK* packet carries the full contents of the *Request* packet (i.e. its request service name, client nonce and cache control field) so that the upstream node can retransmit the *Request* packet without having to cache the packet's contents. To allow the upstream node to differentiate between a *Request* packet and a *Request NACK* packet, each of these packets comprises a "packet type" bit indicating whether the packet is a *Request* packet or a *Request NACK* packet.

Upon receiving the *Request NACK* packet, the upstream node learns from the *Request NACK* packet. In particular, the FIB at the upstream node is updated based on the *Request NACK* packet such that the interfaces of the FIB entries associated with the request service name exclude the interface from which the *Request NACK* packet is received.

Fig. 6 illustrates a method 600 used by the upstream node for processing the received *Request NACK* packet. As shown in Fig. 6, method 600 comprises steps 602 – 608 which will be elaborated below.

In step 602, the upstream node looks up the FIB for a matching FIB entry with a service name matching exactly the request service name (which is the service name of the *Request NACK* packet since the *Request NACK* packet and the *Request* packet have the same contents).

If a matching FIB entry is found, step 604 is performed. In step 604, the interface from which the *Request NACK* packet is received is removed from the interfaces of the matching FIB entry.

If no matching FIB entry is found and the *Request* packet is sent from an upward interface of the upstream node, step 606 is performed. Otherwise, if no matching FIB entry is found and the *Request NACK* packet is not sent from an upward interface, no action is taken.

In step 606, a new FIB entry is created at the upstream node with the request service name and with the interfaces set as all the upward interfaces of the node (except the upward interface from which the *Request NACK* packet is received). In this new FIB entry, all the interfaces have the same capacity value.

Step 608 is performed after step 604 or 606. In step 608, the *Request NACK* packet is converted to a *Request* packet by changing the "packet type" bit of the packet. This *Request* packet is then resent to one of the remaining interfaces of the matching FIB entry or one of the interfaces of the new FIB entry. This interface is selected using the load balancing policy as mentioned above.

Handling Response Packets

Upon receiving a *Request* packet, a VM returns a *Response* packet to the Clients 108 through the DCN along the reverse path of the associated *Request* packet. This *Response* packet is in the format as shown in Fig. 2. In other words, the *Response* packet comprises a service name, a client nonce, a cache control field and a sequence number. This *Response* packet is then converted to IP-based traffic by one of the service gateways 104 for communication through the Internet 106 to the Clients 108.

Fig. 7 illustrates a method 700 used by a node in the DCN for handling a *Response* packet sent from a VM. As shown in Fig. 7, method 700 comprises steps 702 – 706 which will be described in detail below.

In step 702, upon receiving a *Response* packet, the node looks for a matching PRT entry with the service name and client nonce respectively matching exactly the service name and client nonce of the *Response* packet.

If a matching PRT entry is found, step 704 is performed. In step 704, the node sends the *Response* packet to the incoming interface associated with the matching client nonce of the matching PRT entry (this is the interface from which the associated *Request* packet was received at the node). Further in step 704, if the *Response* packet is the last *Response* packet to be sent (i.e. the *Response* packet has a sequence number of zero), the PRT entry is removed from the node.

Otherwise, if no matching PRT entry is found, step 706 is performed. In step 706, the *Response* packet is discarded.

VMM Strategy 800

In a typical VMM, the memory transfer can be generalized into three sequential phases:

1) *Push Phase*: in this phase, the source VM continues running while some memory pages are pushed across the network to the destination VM;

2) *Stop-and-Copy Phase*: in this phase, the source VM is stopped, further memory pages are copied to the destination VM, after which the destination VM is started;

3) *Pull Phase*: in this phase, the destination VM executes and, if the destination VM tries to access a memory page that has not yet been copied, this memory page is pulled across the network from the source VM.

Fig. 8 illustrates a VMM strategy 800 according to an embodiment of the present invention. As shown in Fig. 8, the VMM strategy 800 comprises two of the above-mentioned phases, specifically, the push phase 802 and the stop-and-copy phase 804.

At the beginning of the push phase 802, the source VM sends a first control message in the form of a *service stopping* message through the DCN to the core switches. This is so as to configure the nodes of the DCN to stop forwarding *Request* packets to the VM. The source VM then sends out *Response* packets for the remaining requests in the queue. Memory pages are then copied from the source VM to the destination VM.

During the stop-and-copy phase 804, further pages are copied from the source VM to the destination VM and at the end of the stop-and-copy phase 804, the new VM sends a second control message in the form of a *service starting* message through the DCN to the core switches. This is so as to configure the nodes of the DCN to start forwarding *Request* packets to the destination VM.

Occasionally, due to a communication delay in the DCN, it is possible that a *Request* packet arrives at an incident node which has no interface for further forwarding of the *Request* packet during VMM. This may be because all serviceable VMs under the tree of the incident node may have migrated away at the same time. Another reason for this may be that the arriving time of the *Request* packet may be after the *service stopping* message has reached the incident node and its previous-hop nodes. In this case, the FIB entries at the incident node and the previous-hop nodes for this service would have been removed in response to the *service stopping* message (if the source VM is the only VM providing the service). This phenomenon is usually referred to as a request-forwarding miss.

To address the above issues, the VMM strategy 800 uses the above-described named-service routing protocol to handle requests from Clients 108 during the

VMM so that the requests can be served in an interruption-free manner. More specifically, the VMM strategy 800 uses method 500 to handle *Request* packets during VMM. When an incident node which has previously received a *service stopping* message performs method 500 to process a *Request* packet, step 512 will be performed in which the incident node will send a *Request NACK* packet back to the upstream node from which it received the *Request* packet. The upstream node will then perform method 600 to process the *Request NACK* packet and locate an alternative interface (if any) for forwarding the *Request* packet. If the upstream node cannot locate an alternative interface, the upstream node will also send a *Request NACK* packet to a further upstream node which will then perform method 600 as well. This goes on until a route is found for the *Request* packet to reach a serviceable VM. Thus, using the above-described named-service routing protocol (in particular, the *Request NACK* packet) allows all requests to arrive at a serviceable VM successfully.

Alternatives

Various embodiments will be apparent to one skilled in the art.

For example, the ToR switches in the network 100 may be replaced by other types of access switches. The DCN in the data center 102 may also comprise nodes other than the switches shown in Fig. 1 and there may be a different number of service gateways than that shown in Fig. 1.

In addition, the named-service routing protocol may be configured to process data in formats other than the named-service format described above. For example, the *Request* packet and the *Response* packet may comprise more fields than those shown in Fig. 2. Some of these fields may be used for functions such as authentication, encryption and congestion control.

The nodes may also comprise data structures different from those shown in Fig. 3. In particular, the data structures may comprise more information so that more

functions can be performed through the DCN. The data structures may alternatively comprise less information. For example, the FIB entry need not comprise the capacity value (even though, this is not preferable). Also, depending on the services provided by the VMs in the DCN, a FIB entry can have only one interface. Similarly, a PRT entry can have only one set of nonce, incoming interface and outgoing interface associated with a service name.

Also, although only three types of control messages – the *service starting* message, the *service stopping* message and the *service capacity update* message are described above, other types of control messages may also be used by the VMs to perform other functions.

In the named-service routing protocol described above, if a node is unable to further forward a *Request* packet, the node sends a *Request NACK* packet to the upstream node from which the *Request* packet is received. In an alternative embodiment, the node may simply forward the *Request* packet upward to an upper layer switch to find an alternative route. This can be used for internal traffic in the DCN. However, although this may work well in a richly-connected topology e.g. in a Clos and fat-tree topology (since in such topology, any core switch can forward a request to a VM providing the service), in some DCNs with sparsely-connected topology, one or more of the core switches may not have information regarding some service names in the network. Further, occasionally, the FIB information may be outdated due to loss or communication delay of control messages, particularly in the process of VMM. This issue may be overcome by using the *Request NACK* packet in the manner as described above.

Further, in the named-service routing protocol described above, upon receiving the *Request NACK* packet, the upstream node learns from the *Request NACK* packet regarding the interfaces available for the service. In an alternative embodiment, the upstream node can simply try alternative interfaces without

updating its information. However, this can cause high NACK overhead for successive requests.

Advantages

The named-service framework has several advantages which will be elaborated below.

For example, different from the traditional communication methods in DCNs, the named-service framework uses the services provided by the VMs instead of the network addresses of the VMs to communicate through the DCN. This decouples the services running on the VMs from the network addresses of the physical machines the VMs are on. As such, the services can be accessed by a client without knowledge of the network addresses where the VMs are hosted.

The named-service framework is particularly useful during VMM. In particular, no binding is needed between the VMs' identifiers (service names) and VMs' location-specific addresses in the above-mentioned VMM strategy 800. This can help achieve cost savings as it is not necessary to manage and update mapping data after a VMM. Also, the strategy 800 allows VMM to be performed with minimal service interruption, thus achieving seamless VMM. The VMM strategy 800 is also compatible with several existing VMM methods.

The named-service protocol is efficient and takes advantage of the multi-rooted tree topology of the DCN. For example, this tree topology is used to train the DCN to learn the services provided by the VMs on the DCN. In particular, the FIB entries of the nodes in the network are populated using event-driven control messages from the VMs while these messages travel through the nodes. This helps to achieve a low overhead.

Further, when forwarding *Request* packets through various nodes of the DCN, the named-service routing protocol uses a distributed load balancing policy

when selecting the interfaces to forward the *Request* packets to. This allows the load from the Clients 108 to be distributed among multiple VMs and multiple paths. In turn, this helps to avoid congestion and hot spots for arbitrary traffic patterns in the DCN. Also, neither a centralized controller nor a load balancer is needed as load balancing is considered when each node forwards a request, so requests are distributed among VMs in balance during the routing of the request.

The named-service routing protocol described above also makes use of *Request NACK* packets to inform upstream nodes when a node is unable to further forward a *Request* packet. As mentioned above, the use of the *Request NACK* packet helps overcome issues that arise when core switches do not have the information regarding a particular service. Further, configuring the upstream node to learn from the *Request NACK* packet can reduce the NACK overhead for successive requests.

Embodiments of the present invention can be deployed in many commercial DCNs. In the embodiments, a service can be accessed by clients without knowledge of the network addresses of the VMs on which the service is running. Furthermore, the service can be accessed by the client with minimal interruption during VMM. This seamless and flexible VMM achieves energy savings as well as a better performance in commercial DCNs.

Claims

1. A method for managing migration of a source virtual machine in a data center network, wherein the data center network comprises a hierarchy of nodes, wherein the data center network is configured to provide at least one service, the at least one service being associated with one or more virtual machines in the data center network, wherein requests and responses for the at least one service are respectively relayed downstream to and upstream from the one or more virtual machines through the nodes, each node comprising information for facilitating routing of the requests and responses through the nodes; and

wherein each node of the data center network comprises a Forwarding Information Base (FIB) with each FIB entry having:

a service name identifying a particular service associated with a virtual machine in the data center network; and

at least one interface associated with the service name, the at least one interface indicating an interface through which the virtual machine providing the particular service can be reached; and

wherein the data center network is managed according to a method comprising:

(1a) sending a control message from a virtual machine through the nodes of the data center network, the control message indicating events related to the service associated with the virtual machine; and

(1b) at each node through which the control message is sent, updating the information based on the control message

wherein updating the information in (1b) comprises updating the FIB at the node, and wherein during the migration, communication through the data center network is performed according to a method comprising the following steps at a node of the data center network:

(1c) receiving a *Request* packet comprising a request service name from an upstream node;

(1d) locating matching FIB entries having the service name matching the request service name; and

(1e) communicating with a further node based on the locating in step (1d) for further forwarding of the *Request* packet through the nodes.

2. A method according to claim 1, wherein the FIB entry further comprises a capacity value associated with the at least one interface, the capacity value indicating a number of requests that can be served per time unit by one or more virtual machines in the data center network associated with the service and reachable through the at least one interface.

3. A method according to claim 1, wherein step (1e) comprises:

if matching FIB entries are located, obtaining interfaces of the matching FIB entries as candidate interfaces;

if no matching FIB entry is located, obtaining upward interfaces of the node as candidate interfaces;

selecting the further node based on the candidate interfaces obtained; and

communicating with the selected further node.

4. A method according to claim 3, wherein selecting the further node comprises:

filtering tried interfaces from the candidate interfaces to obtain untried interfaces, the tried interfaces comprising interfaces to which the *Request* packet has been sent and from which a negative acknowledgement packet has been received; and

selecting the further node based on whether there are untried interfaces.

5. A method according to claim 4, wherein selecting the further node based on whether there are untried interfaces comprises:

if there are untried interfaces, selecting a node connected to an interface from the untried interfaces as the further node; and

if there is no untried interface, selecting the upstream node from which the *Request* packet is received as the further node.

6. A method according to claim 5, wherein selecting an interface from the untried interfaces comprises selecting the interface based on capacity values associated with the untried interfaces in the corresponding FIB entries.

7. A method according to claim 6, wherein selecting the interface based on capacity values associated with the untried interfaces in the corresponding FIB entries comprises distributing a request load of the service to each untried interface such that maximum utilization on the untried interfaces is minimized, wherein the utilization on each interface is the load on the interface per unit capacity of the interface.

8. A method according to any one of claims 5 – 7, wherein communicating with the further node comprises:

if there are untried interfaces, forwarding the *Request* packet to the further node; and

if there is no untried interface, sending a negative acknowledgement packet to the further node, the negative acknowledgement packet comprising contents of the *Request* packet.

9. A method according to claim 8, wherein each node of the data center network comprises a Pending Request Table (PRT), wherein the *Request* packet further comprises an indicator of a destination of the *Request* packet and wherein if there are untried interfaces, the method further comprises creating a PRT entry at the further node, the PRT entry comprising:

the request service name of the *Request* packet;

the indicator of the destination of the *Request* packet;

an incoming interface indicating the interface from which the *Request* packet is received; and

an outgoing interface indicating an interface to which the *Request* packet will be forwarded.

10. A method according to claim 8, wherein if there is no untried interface, the method further comprises updating the FIB at the further node based on the negative acknowledgement packet.

11. A method according to claim 10, wherein updating the FIB at the further node further comprises updating the interfaces of the FIB entries such that the interfaces associated with the request service name exclude the interface from which the negative acknowledgement packet is received.

12. A method according to claim 11, wherein the negative acknowledgement packet comprises the request service name and wherein updating the interfaces of the FIB entries comprises:

locating a matching FIB entry having a service name matching the request service name; and

if a matching FIB entry is located, removing from the interfaces of the matching FIB entry, the interface from which the negative acknowledgement packet is received.

13. A method according to claim 12, wherein updating the interfaces of the FIB entries further comprises:

if a matching FIB entry is not located, creating a new FIB entry at the further node, the new FIB entry comprising:

the request service name; and

interfaces comprising all upward interfaces of the further node, except the interface from which the negative acknowledgement packet is received.

14. A method according to claim 13, wherein the capacity values associated with the interfaces of the new FIB entry are equal.

15. A method according to claim 13 or 14, further comprising converting the negative acknowledgement packet to a *Request* packet and forwarding the *Request* packet to one of the remaining interfaces of the matching FIB entry or one of the interfaces of the new FIB entry.

16. A method according to any one of claims 9 – 15, further comprising:

receiving a *Response* packet comprising a service name and an indicator from a downstream node of the data center network, the indicator indicating a destination of the *Response* packet;

locating a matching PRT entry having a service name and an indicator respectively matching the service name and indicator of the *Response* packet; and

forwarding the *Response* packet to the incoming interface associated with the matching indicator of the matching PRT entry.

17. A method according to claim 16, further comprising removing the matching PRT entry if the *Response* packet is the last *Response* packet to be sent.

18. A method according to claim 1 or 2, further comprising:

sending a first control message from the source virtual machine through the nodes, the first control message indicating that the source virtual machine will stop providing a service;

copying memory pages from the source virtual machine to a destination virtual machine;

sending a second control message from the destination virtual machine through the nodes, the second control message indicating that the destination virtual machine will start providing the service.

19. A method according to claim 18, wherein each control message comprises a service name and wherein sending the first or second control

message through the nodes further comprises the following steps at each node through which the control message is sent:

locating a matching FIB entry having a service name and an interface respectively matching the service name of the control message and the interface from which the control message is received; and

if a matching FIB entry is found, updating or removing the matching FIB entry based on the control message.

20. A method according to claim 19, wherein the control message comprises a capacity value and the source virtual machine is one of multiple virtual machines providing the service in the data center network, and wherein updating or removing the matching FIB entry based on the control message further comprises:

replacing the capacity value associated with the matching interface of the matching FIB entry with the capacity value of the control message.

21. A method according to claim 19 or 20, wherein the method further comprises the following step at the node:

if a matching FIB entry is not found, creating a new FIB entry for the service.

22. An apparatus comprising a data center network having a hierarchy of nodes,

wherein the data center network is configured to provide at least one service, the at least one service being associated with one or more virtual machines in the data center network, wherein requests and responses for the at least one service are respectively relayed downstream to and upstream from the one or more virtual machines through the nodes, each node comprising information for facilitating routing of the requests and responses through the nodes; and wherein migration of a source virtual machine in the data center network is managed according to a method of claim 1 or 2.