Supporting Seamless Virtual Machine Migration via Named Data Networking in Cloud Data Center

Ruitao Xie, Yonggang Wen, *Senior Member, IEEE*, Xiaohua Jia, *Fellow, IEEE*, and Haiyong Xie

Abstract—Virtual machine migration has been touted as one of the crucial technologies in improving data center efficiency, such as reducing energy cost and maintaining load balance. However, traditional approaches could not avoid the service interruption completely. Moreover, they often result in longer delay and are prone to failures. In this paper, we leverage the emerging named data networking (NDN) to design an efficient and robust protocol to support seamless virtual machine migration in cloud data center. Specifically, virtual machines (VMs) are named with the services they provide. Request routing is based on service names instead of IP addresses that are normally bounded with physical machines. As such, services would not be interrupted when migrating supported VMs to different physical machines. We further analyze the performance of our proposed NDN-based VM migration protocol, and optimize its performance via a load balancing algorithm. Our extensive evaluations verify the effectiveness and the efficiency of our approach and demonstrate that it is interruption-free.

Index Terms—Named-data networking, cloud data center, virtual machine migration

1 Introduction

Virtualization [1] has been touted as a revolutionary technology to transform data centers. With the growing demand of Internet services [2], more and more data centers were deployed globally. However, their utilization has found to be low, typically ranging from 10 to 20 percent, as reported in [3]. Virtualization technology, in which physical resources (i.e., computing, storage, networking) are partitioned and multiplexed, has been introduced to tackle the low-utilization problem in data centers. Specifically, a virtualized server, referred to as a virtual machine (VM), can be dedicated to a particular application, and virtual machine migration [1], [4] can be used to consolidate VMs from under-utilized physical machines in order to reduce the energy cost of data centers. It has been reported that VM migration can provide various benefits, such as load balancing [5] and performance optimization [6].

However, the VM migration imposes new challenges on data center operations. It results in an IP mobility problem [7], in which a VM migrated to another subnet would be assigned with a different IP address. The IP mobility problem brings in challenges on both live and offline VM migrations. In the live migration, a VM is migrated while its service is running. It would result in service interruptions. Since other VMs communicating with the migrated VM could not learn its new IP address timely. In the offline migration, a VM is migrated after it is shut down and assigned with a new IP address. As a result, the change of the IP address requires a manual reconfiguration of the network settings in all related VMs. Such a manual process is labor-intensive, time-consuming and error-prone, since the IP addresses are involved in various network, security and performance policies.

Various approaches have been proposed to address the VM migration problem. One approach, as in [8], [9], [10], attempted to solve the problem by adapting the existing mobile IP solution, i.e., the triangular routing. It not only introduces packet delivery delay but also results in excessive network traffic. Another approach [11] adapted the mobile IPv6 to the VM migration. It requires the migrated VM to advertise its new address to other communicating nodes. It could not avoid service interruptions, because the advertisement takes some time, during which the traffic to the migrated VM may be routed to its old location via outdated information. A different approach, as in [7], [12], [13], [14], used two addresses to solve the problem, an IP address used for identifying a VM and a location-specific address used for routing to a VM. The binding information between these two addresses is maintained in a central proxy. This approach can neither avoid service interruptions. Since it also takes some time to update the binding information. Moreover, the central approach is unscalable and prone to the single point of failure problem.

In this paper, leveraging the emerging technique named data networking (NDN) [15], [16], we propose a novel method to support an interruption-free VM migration. That is, we identify the communication with a VM via the name of the service running on it rather than its

- *R. Xie and X. Jia are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. E-mail: {ruitao.xie, csjia}@cityu.edu.hk.*
- *Y. Wen is with the School of Computer Engineering, Nanyang Technological University, Singapore 639798. E-mail: ygwen@ntu.edu.sg.*
- *H. Xie is with China Academy of Electronics and Information Technology, and University of Science and Technology of China. E-mail: haiyong.xie@acm.org.*

Manuscript received 12 May 2014; revised 19 Nov. 2014; accepted 27 Nov. 2014. Date of publication 3 Dec. 2014; date of current version 6 Nov. 2015. Recommended for acceptance by X. Gu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2377119

network address. The routing is directed via the service name. In this way, services and VMs are decoupled from their locations.

In the named-service framework, we aim to support a seamless VM migration protocol. First, we propose a dedicated named service routing protocol. Second, we propose a VM migration policy to support interruption-free service access. Finally, we conduct analyses and simulations to verify the benefits of our approach. The analysis and simulation results demonstrate the following advantages:

1) The VM migration is interruption-free;
2) The overhead to maintain the routing information by our routing protocol is less than classic NDN;
3) The routing protocol is robust to both link and node failures;
4) Our framework inherently supports the implementation of a distributed load balancing algorithm, via which requests are distributed to VMs in balance.

The rest of this paper is organized as follows. Section 2 presents the related works on solving the IP mobility problem in VM migrations. In Section 3, we briefly introduce the concept of NDN, and present a named-service framework. Section 4 presents the named-service routing protocol. Section 5 presents a seamless VM migration policy and analyzes its performance. Section 6 presents a distributed load balancing algorithm. The simulations and performance evaluations are presented in Section 7. Section 8 concludes the paper.

2 Related Works

We review the existing works on the IP mobility problem in DCN and categorize them into three lines of research.

The first line of work adapts the existing mobile IP solution, to VM migrations (as in [8], [9], [10]), in which all traffic to and from the migrated VM must traverse a home agent in the source subnet. This triangular routing not only introduces packet delivery delay but also results in excessive network traffic. Johnson et. al. in [11] adapted the mobile IPv6 to VM migrations. Although it can avoid the triangular routing, it requires the migrated VM to advertise its new address to other communicating nodes. This needs the migrated VM to be aware of the set of communicating nodes, which is impractical. In addition, it could not avoid service interruptions. Because the advertisement takes some time, during which the traffic to the migrated VM may be routed to its old location when the address has not been updated yet.

The second line of work uses two addresses to solve the problem (as in [7], [12], [13], [14]): an IP address used for identifying a VM and a location-specific address used for routing to a VM. As such, once a VM is migrated to another location, its IP address is retained, while its location-specific address is changed. The binding information between these two addresses is maintained in a central proxy. This approach can neither avoid service interruptions. Since it also takes some time to update the binding information in the central proxy and keep the binding information consistent within other communicating VMs. During the updating time, the traffic may be routed with outdated binding information. Moreover, the central approach is unscalable and prone to the single point of failure problem.

The third line of work adopts service names to access applications in DCN [17]. This approach still needs to maintain a binding between a service name and a list of network addresses of servers in a dedicated server. In this approach, first packet of each flow is sent to a service router to resolve the service name, and the following packets are routed via the resolved network address. It encounters the same problems as the above binding approach.

3 Named-Service Framework in DCN

We now apply the key design principles of NDN to address the IP mobility problem in VM migrations.

3.1 Named Data Networking: A Primer

NDN is a receiver-driven, data-centric communication paradigm [15], [16]. In NDN, each piece of data is uniquely identified by a location-independent name, and communication is performed by specifying the data names instead of the host locations. There are two types of NDN packets: *interest* and *data*. To request a data, a consumer puts the name of the data into an interest packet and sends it out. The routing nodes use the name to forward the interest packet toward the data producer. The data packet is returned along the reverse path of the corresponding interest packet, either from a data producer or a routing node that caches the data.

NDN adopts a novel routing protocol. Each routing node maintains three data structures: the forwarding information base (FIB), the pending interest table (PIT) and the content store. FIB is similar to that in IP-based networks, except that it contains name prefixes instead of IP address prefixes, and allows for multiple outgoing interfaces rather than a single one for each name prefix. PIT maintains an entry for every pending interest packet, so that data packets can be returned in reverse. The content store caches the data transferred by the routing node, which is different from traditional IP-based networks.

Based on these routing information, NDN routing node can process two types of packets as follows. First, when a node receives an interest packet, it looks up the longest-match information with the requested data name. It first checks whether there is a matching data in its content store. If any is found, the data is sent through the incoming interface of the interest packet. Otherwise, if there is a matching PIT entry, then the incoming interface of current request will be added to the list of incoming interfaces in this PIT entry. Otherwise, if there is a matching FIB entry, then the node forwards the interest packet to the interface determined by a particular forwarding policy, and creates a new PIT entry. If there is no matching for this interest packet, then it is discarded. Second, a data packet is returned to consumers following the reverse path of the corresponding interest packet. When a routing node receives a data packet, it looks up its PIT with the data name. If a matching PIT entry is found, the node sends the data packet to a list of interfaces in the entry. Then it caches the data in the content store and removes the PIT entry. Otherwise, the data packet is discarded.

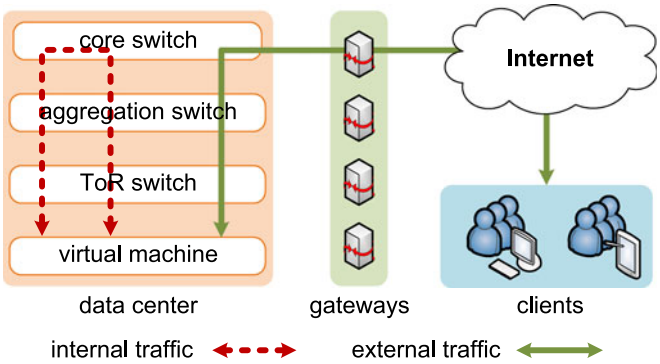


Fig. 1. Named-Service Framework in DCN. The gateways translate the IP-based external traffic into the name-based traffic.

3.2 Design Rationale

DCN is a good environment for applying the NDN designs. There are two reasons. First, the applications in DCN require efficient managements, such as VM migration, load balancing and caching. NDN can provide a systematic solution to address these issues. In contrast, existing approaches are point solutions, such as migration systems, load balancers and caching proxies. Second, DCN allows efficient network operations, due to its regular topology and well controlled machines.

NDN decouples location from identity and communication, and enables maintaining communication in dynamic environments, for example, when the host locations change.

Following this principle, we propose a *named-service framework* for DCN. In this framework, a service is identified by a location-independent name and is accessed via its name instead of the network address of its host VM. Therefore, clients can access the service regardless of where VMs are hosted or migrated. In addition, as in NDN, our framework can provide robust security and performance improvement by leveraging the features of data-based security and data caching.

3.3 Named-Service Framework in DCN

We next present a system-level description of our proposed named-service framework. As illustrated in Fig. 1, our framework consists of two components: i) the named service protocols within the data center, and ii) service gateway(s). The first component comprises a named-service routing protocol rendering the service to the requests and a VM migration policy supporting the VM migrations. The second component translates the IP-based external traffic into the named-service framework.

This framework can support two types of traffic in DCN: 1) traffic flowing between an external client and an internal VM, and 2) traffic flowing between two internal VMs [18]. The requests of internal traffic are directly presented in the named-service format, while the external requests enter in the framework through the dedicated gateways.

We make the following assumptions when designing the framework. First, a service is deployed on multiple VMs in DCN. This is reasonable since an application in DCN is usually provided by multiple servers for scalability and reliability. Second, the migrated VM does not store user context. As such, the requests from the same user can be handled by any VM that provides the service. This is reasonable because

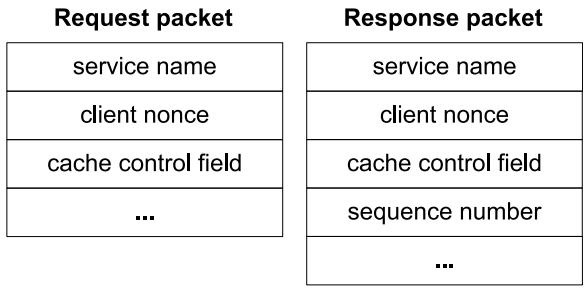


Fig. 2. Basic packet formats in named-service routing protocol.

majority traffic in data center are based on stateless protocols, such as HTTP [19]. Note that the first assumption ensures that other VMs can provide the service when a VM migrates, while the second assumption ensures that a request is location-independent, thus can be identified by a location-independent service name and routed in our framework.

3.4 Design Challenges

We address three issues when designing the named-service framework: an efficient and robust routing protocol, a flexible migration policy and performance optimization. First, we encounter a challenge distinctive from the classic NDN in designing a named-service routing in DCN: the low-overhead requirement. Specifically, in the classic NDN, the propagation of routing information results in a high overhead, since it involves every routing node. To reduce this overhead, a dedicated and efficient named-service routing protocol is designed, by taking advantage of the multi-rooted tree topology of DCN as in Fig. 4. Second, to handle the routing in sparsely-connected topology resulted from link and node failures, we propose a mechanism of route learning. Third, to support interruption-free service access, we propose a VM migration policy. Finally, to avoid link congestions and hot spots, we propose a distributed load balancing algorithm to distribute requests across multiple VMs and multiple paths.

4 Named-Service Routing Protocol

In this section, we propose a named-service routing protocol, a core component of the named-service framework. To meet the low-overhead requirement in DCN, we design a dedicated control message protocol to maintain routing information at routing nodes. Moreover, to improve the robustness of the routing protocol under link and node failures, we propose a mechanism of route learning. This section is organized as follows. We start with basic packet formats and data structures maintained at routing nodes. Then, we introduce the control message protocol. Next, we introduce the specific forwarding process to handle the packets.

4.1 Packet Format

Communications in the named-service routing protocol are performed over two types of packets: *request* and *response*, adopted from the classic NDN packet formats, as illustrated in Fig. 2. We explain each of the fields as follows.

The *service name* uniquely identifies a service running on multiple VMs in DCN. It is carried by both request and response packets. The naming system is an important

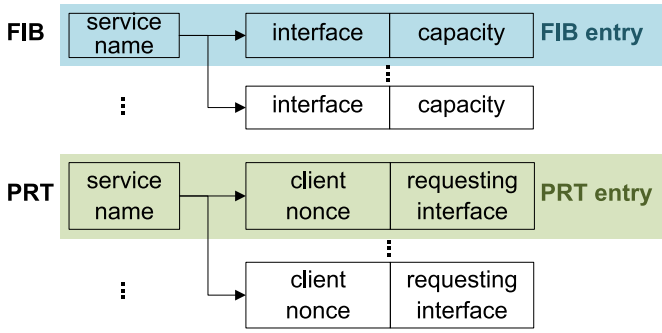


Fig. 3. Basic routing data structures in named-service routing protocol.

research issue in the NDN and is now still under active research [15]. Note that our named-service framework doesn't depend on any specific naming system.

The *client nonce* is a random number generated by the client, carried by both packets. It is used for two purposes. First, duplicated request packets have the same client nonce, thus they can be detected and discarded. Second, the destination of a response packet is identified via the nonce, so that it can be returned along the reverse paths of the associated request packet.

The *cache control field* denotes whether the packet is for a dynamic content (such as a search result) or for a static content (such as a piece of video). It is set by the client sending the request packet, and copied into the response packets returned. By detecting this field, the switches decide whether to cache the content or not. We focus on the case where the request is for a dynamic content in this paper. Our methods are easily extended to the static content case.

The *sequence number* is created by the responding service and carried in every response packet. It indicates the number of remaining response packets to be sent by the responding service. It is zero for the last packet. Since all the response packets corresponding to a request follow the same path back to the client, they are received by the switch in the same order. Thus, upon a switch receives and forwards the last response packet, it removes the related pending request table (PRT) entry.

In addition, more fields can be added for other manipulations, such as authentication, encryption and congestion control, as in classic NDN [15], [16].

4.2 Routing Data Structures

In the named-service routing protocol, similar to NDN, each routing node maintains three data structures: the forwarding information base, the pending request table and the Cache Store.

FIB is similar to that in NDN. As shown in Fig. 3, it consists of the service name and multiple outgoing interfaces for each service name. In addition, for each pair of name and interface, the FIB entry has a capacity value. It is used for a distributed load balancing algorithm, which will be discussed in Section 6. It represents the total capacity (i.e., maximum number of requests served per time unit) of the VMs that provide the service and are reachable through the interface in the FIB entry.

PRT is used to maintain an entry for every pending request packet, so that every response packet can be returned to the client in a reverse path. Each PRT entry records the service name, the client nonce identifying the client, and the requesting interface from which the request packet is received.

The Cache Store is used to cache the response data, if the cache control field in the response packet indicates that the data is cacheable.

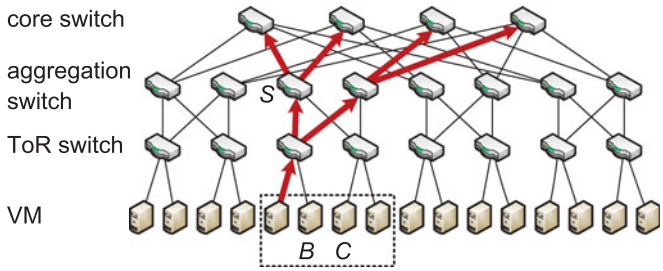


Fig. 4. Each switch forwards a control message to all of its upper-layer switches directly connected. For example, the control message to publish VM *A*'s service name is transferred in the direction of arrows. As such, each switch knows the service names of the VMs in its subtree. For example, aggregation switch *S* knows the service names of the VMs: *A*, *B*, *C* and *D*.

4.3 Control Message and FIB Update

Next we discuss how FIB is populated and updated using the event-driven control messages. Via the control messages, a VM can report *service starting*, *service stopping* and *service capacity updating*. Specifically, a service starting message is sent, when a VM is allocated at the first time or migrated to a new location. Similarly, a service stopping message is sent, when a VM is about to migrate or stop its service. Moreover, a service capacity updating message is sent, when a VM's capacity changes. More functions can be added if necessary.

To reduce the overhead to propagate control messages, we implement a small-scale propagation. We allow each switch to forward a control message to all of its upper-layer switches directly connected. For example, in a DCN as shown in Fig. 4, the control message to publish VM *A*'s service name is transferred in the direction of arrows. This small-scale propagation results in *three properties* of the service names in FIB:

1) Each switch knows the service names of the VMs in its subtree, which consists of itself as the root node and all its descendant nodes in the topology (e.g., in Fig. 4, aggregation switch *S* can learn the service names of the VMs: *A*, *B*, *C* and *D*);
2) The closer a switch is to the VMs, the fewer service names it knows; while the closer a switch is to the core-switch layer, the more service names it knows;
3) In a richly-connected topology, such as the Clos and fat-tree topology as shown in Fig. 4, each core switch knows the service names in the entire network.

The FIB entries are updated as follows. A control message sent by a VM carries its service name and capacity value. In particular, a control message with the capacity value of zero reports the service stopping. Once a switch receives a control message, it proceeds two steps: 1) updates the FIB; 2) updates and forwards the message. First, it looks up a FIB entry by matching the entry's service name with the one carried in the

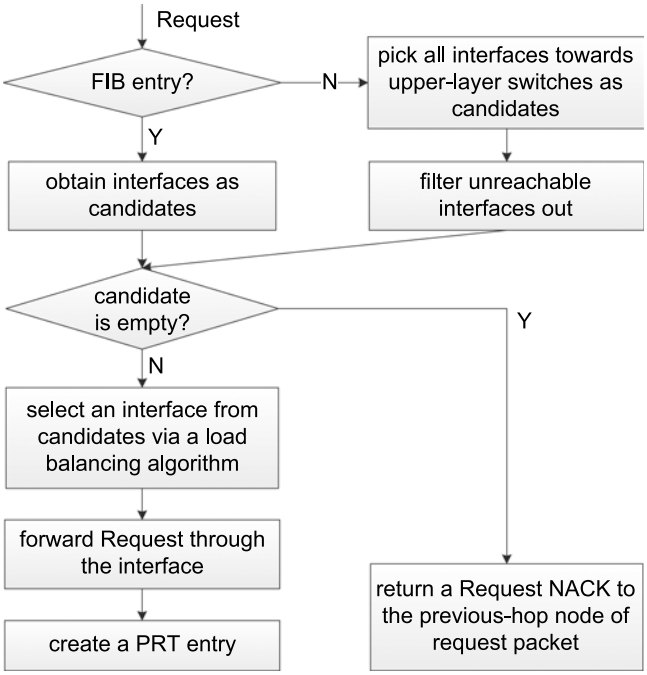


Fig. 5. The flowchart of handling a request packet.

message and matching the entry's interface with the one from which the message is received, respectively. If any entry is not found, then a new FIB entry is created and added into FIB. Otherwise, that is a FIB entry is found, if the message reports the service stopping, then the entry will be removed; if the message reports the service capacity updating, then the entry will be updated by replacing its capacity value with the one carried in the message. Second, this switch calculates the total capacity of the VMs that have this service name and are reachable through this switch. That is, it sums up all capacity values of this service name in its FIB. Then, it updates the control message by replacing the capacity field with the calculated total capacity, and forwards the updated message to all upper-layer switches.

This control message protocol results in lower overhead than the classic NDN [15], in which the routing information is propagated to every routing node. The overhead is affected by the transmissions of the messages rather than the message size, since a control message only carries a service name and a capacity value, and its size is small. For example, in the fat-tree network built with four-port switches as shown in Fig. 4, our protocol takes seven transmissions to propagate a control message; while the classic NDN takes 33 transmissions, which includes seven transmissions from the VM to all core switches and 26 transmissions from all core switches to all ToR switches. In general, for a fat-tree network built with k-port switches ($k \geq 4$) [20], via our protocol, a message is sent from a VM to its upper ToR switch, and then is transmitted following a full $\frac{k}{2}$-ary tree from the ToR switch to all core switches. Thus its number of transmissions is

$$\frac{k}{2} \cdot \frac{k}{2} + \frac{k}{2} + 1 = \frac{k^2}{4} + \frac{k}{2} + 1. \tag{1}$$

In comparison, via the classic NDN, a message is sent from a VM to its upper ToR switch, and then is transmitted along

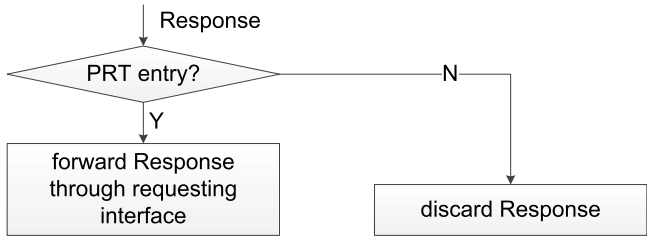


Fig. 6. The flowchart of handling a response packet.

all the links between all ToR switches and all core switches. Thus its number of transmissions is

$$k \cdot \left(\frac{k}{2}\right)^2 + \frac{k}{2} \cdot \frac{k}{2} \cdot k + 1 = \frac{k^3}{2} + 1. \tag{2}$$

Thus, our control message protocol can reduce the overhead from $O(k^3)$ to $O(k^2)$. Since there are $s = 5k^2/4$ switches in the fat-tree topology, equivalently the control message overhead is reduced from $O(s^{\frac{3}{2}})$ to $O(s)$.

4.4 Forwarding Process

In this part, we discuss the forwarding process to handle packets in our named-service routing protocol. Recall that via the preceding control message protocol, the service names in the FIB of each switch have three properties. Based on these properties, we propose the following forwarding process: upon receiving a request packet, if a switch knows the requested service name, it will forward this packet; otherwise, it will forward this packet to any upper-layer switch in order to find a route. As such, a request can be successfully forwarded. This is because, a request would arrive at a core switch, if no switch at previous hops knows the requested service name. Moreover, every core switch can forward the request, since it knows all service names (from property 3). However a core switch may not know all service names in some situations, we will discuss how to deal with it in Section 4.5.

We first explain how to handle the request and response packets in detail, as illustrated in Figs. 5 and 6 respectively. Because a service is provided by multiple VMs, there are multiple interfaces to forward a request. Thus, a set of interfaces are picked as candidates, and one of them is selected via a load balancing algorithm introduced in Section 6. To obtain the candidate interfaces, when a switch receives a request packet, it looks up the FIB entries by matching the entries' service name with the requested one. If some FIB entries are found, a set of interface candidates are obtained; otherwise, all interfaces connected to upper-layer switches are picked as candidates. After the request is forwarded to one of the candidate interfaces, a new PRT entry is created for returning the response packets later.

A response packet is returned to the client along the reverse path of the corresponding request packet, using the information in PRT. Specifically, when a switch receives a response packet, it looks up a PRT entry whose service name and client nonce match those carried in the response packet respectively. If any is found, then the switch will send the packet to the requesting interface in the entry found, and then remove the PRT entry if this is the last

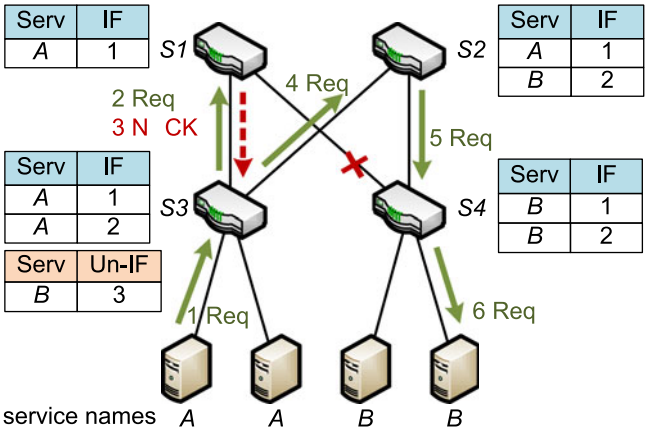


Fig. 7. An example of routing a request when a link fails (red cross). Each switch's FIB is shown beside it (Serv: service, IF: interface). Due to the link failure, switch S_1 knows only service name *A*, exclusive of service name *B*. Suppose service *A* would like to request service *B*. When switch S_1 receives the request for *B*, it has no interface to forward, so it returns an NACK back. When switch S_3 receives the NACK, it records the service name *B* and the unreachable interface (Un-IF for short) 3. Then it re-forwards the request through another interface.

response packet (with sequence number of 0). Otherwise, that is any PRT entry is not found, it will be discarded.

4.5 Mechanism of Route Learning

Via the preceding forwarding process, a request packet would arrive at a core switch, if no switch at previous hops finds any FIB entry to forward this request. The request will be successfully forwarded, if the core switch knows the service name. It is usually true in a richly-connected network that every core switch can know the service names in the entire network. However, it is not true in some scenarios, such as when a link fails, or the network topology is sparsely connected. These scenarios result in that a top-layer switch cannot forward a request, and further a request may fail to reach a VM. For example, Fig. 7 shows a network topology which consists of four VMs providing two services named A and B and four switches $S_1 \sim S_4$. Each switch's FIB is shown beside it, where Serv represents service and IF represents interface. Due to the link failure between S_1 and S_4 (red cross in the figure), switch S_1 knows only service name A, exclusive of service name B. Suppose service A would like to request service B. When switch S_1 receives the request for B, it has no interface to forward.

To address this problem, we introduce the *request NACK* packet, inspired by [16]. That is, when a routing node cannot forward a request packet, it sends back a request NACK to the previous-hop node of the request packet, to let it try an alternative interface. However, this mechanism causes high NACK overhead and long latency for successive requests. Since although the node receiving the NACK uses another interface to forward the current request, it still possibly uses the unreachable interface to forward a new request. To address it, we further propose a mechanism of route learning, that is, recording the unreachable interfaces toward the upper-layer switches so that these interfaces can be filtered out when forwarding the requests afterwards. The reason why to record the unreachable interfaces instead of the reachable ones is that in a richly-connected data center network the number of unreachable interfaces is usually

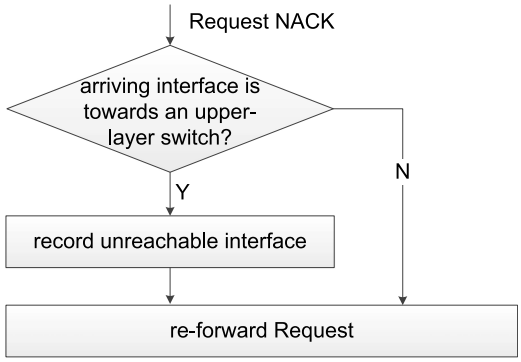


Fig. 8. The flowchart of handling a request NACK packet.

lower than that of reachable ones. It is noted that a request NACK packet carries the whole information in the request packet, so that a receiving node can retransmit the request without caching.

Next, we explain how to proceed the mechanism of route learning in detail. Each switch maintains a list of unreachable interfaces, in order to filter them out in forwarding the requests toward the upper-layer switches. It consists of the service name and the unreachable interface. As illustrated in Fig. 8, upon receiving an NACK packet, if its arriving interface is toward an upper-layer switch, then this node adds the entry $\langle servicename, arrivinginterface \rangle$ into the list of unreachable interfaces. To adapt this list to the network dynamics (e.g., a link failure recovery), we set an expiration time for each entry. An entry is removed from the list when its time expires. The setting of the expiration time depends on how fast a switch needs to adapt to network dynamics. The shorter the expiration time is, the faster a switch adapts to network dynamics, but it incurs more NACK overhead. After the route learning, the node obtains the request from the request NACK packet, and forwards the request again to another interface. For example, in Fig. 7, switch S_1 has no interface to forward the request for B, so it returns an NACK back. When switch S_3 receives the NACK, it records the service name and the unreachable interface (Un-IF for short): B and 3. Then it re-forwards the request through another interface. Eventually, this request arrives at service B. Moreover, subsequent requests for service B will not be forwarded to the unreachable switch S_1, since the unreachable interface is filtered out.

Here, we discuss how to filter unreachable interfaces out in the request forwarding. Recall that without the mechanism of route learning, if no FIB entry is found, then all interfaces connected to upper-layer switches are picked as candidates, as discussed in Section 4.4. To adapt to the new mechanism, an extra step has to be taken. That is, the switch looks up the list of unreachable interfaces, to find if there is any unreachable interface for the requested service. If any is found, then this unreachable interface is removed from the candidates, as illustrated in Fig. 5.

4.6 Discussions on Reliability

We discuss two mechanisms to improve the reliability of the named-service routing protocol. First, the outdated FIB entries due to node failures or control message losses can be updated by interface probing. That is, a switch can periodically send probing messages to its lower-layer switches or

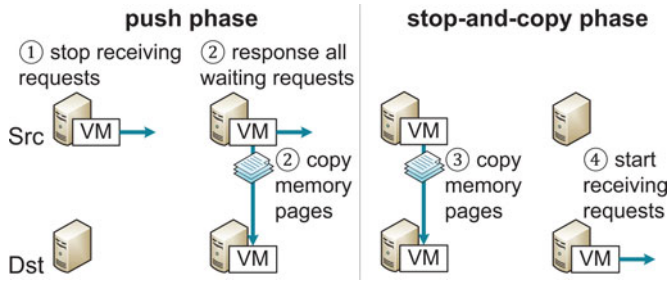


Fig. 9. The VM migration policy in named-service framework.

VMs to collect the latest service names and capacity information. The overhead can be controlled by limiting the probing frequency. Second, for packet losses, clients are responsible for re-sending the request when any response is lost.

5 Virtual Machine Migration Policy

In this section, we propose a seamless VM migration policy, based on the named-service routing protocol, and analyze its performance.

5.1 VM Migration Phases

In a typical VM migration, the memory transfer can be generalized into three sequential phases [4]:

- *Push phase.* In this phase, the source VM continues running while certain pages are pushed across the network to the destination;
- *Stop-and-copy phase.* In this phase, the source VM is stopped, pages are copied to the destination VM, then the new VM is started;
- *Pull phase.* In this phase, the new VM executes and, if it accesses a page that has not yet been copied, this page is pulled across the network from the source VM.

Most practical migration approaches select one or two of three sequential phases, as introduced in [4]. In next subsection, we propose a policy which involves in the first two phases, to support an interruption-free VM migration.

5.2 VM Migration Policy

The VM migration policy involves in the push phase and the stop-and-copy phase. As shown in Fig. 9, it consists of two rules: 1) at the beginning of the push phase, the source VM sends a service stopping message to stop receiving requests (①in the figure), and at the end of the stop-and-copy phase, the destination VM sends a service starting message to receive requests (④in the figure); 2) the source VM does not stop, that is, does not start the stop-and-copy phase, until it responses all waiting requests in its queue (② in the figure).

Recall that via the control message protocol, the service stopping message sent by the migrated VM is transferred from the bottom layer to the top layer. When a routing node receives the message, it updates its FIB. Instead of updating all the FIBs on all related nodes simultaneously, the upper-layer node updates its FIB latter than the lower-layer node, since there is transmission delay. This results in a situation that when a lower-layer node receives a request from an upper-layer node, all its FIB entries related to the requested service may have been removed so that it cannot forward this request. This is because when the upper-layer node forwards the request, it has not received the stopping message so that it has not updated its FIB yet. We can use the NACK mechanism (as discussed in Section 4.5) to deal with this situation. That is, the lower-layer node that cannot forward a request returns a request NACK to the upper-layer node to let it try another interface. As such, the request would be routed to a serviceable VM.

This migration policy is seamless. Since during the migration, all requests can be routed to serviceable VMs and responded successfully: 1) the requests received by the source VM before migrating are responded, as the second rule of the migration policy requires; 2) during the migration, requests are routed to the other serviceable VMs rather than the source VM via the load balancing algorithm directly or the NACK mechanism; 3) after the migration, the migrated VM becomes serviceable.

5.3 Performance Analysis

Via the VM migration policy, during the period between the time when the source VM sends the service stopping message and the time when all related FIBs are updated by the message, some requests may arrive at a switch that cannot forward them, so they incur request NACKs. In this section, we will analyze how many requests incur request NACKs in the worst case, and how much delay are resulted.

Let t_{req} denote the time when an external request is dispatched by a gateway, and t_{mig} denote the time when the migrated VM sends the service stopping message. Let d denote the single-hop transmission delay.

We take an example to illustrate that the external requests that satisfy

$$t_{mig} - 2d \leq t_{req} < t_{mig} \tag{3}$$

incur the request NACKs at the layer of ToR switch. As shown in Fig. 10, the network topology consists of a gateway, five switches $S_0 \sim S_4$ and 4 VMs. Only one out of two VMs connected to each ToR switch provides a service named A. Suppose the VM connected to switch S_3 is about to migrate. It sends a service stopping message at time t_{mig}. This message is transferred until it arrives at the core switch. As illustrated by the timeline in this figure, the service stopping message (StopMsg for short) arrives at switch S_3 at time $t_{mig} + d$. Upon its arriving, FIB entry $\langle$service A, interface 1$\rangle$ is removed from S_3. The stopping message arrives at switch S_1 at time $t_{mig} + 2d$, and then FIB entry $\langle$service A, interface 1$\rangle$ is removed from S_1. Suppose a request (Req for short) for A is dispatched at t_{req} by the gateway. It arrives at aggregation switch S_1 at time $t_{req} + 2d$ after two hops. If this time is between t_{mig} and $t_{mig} + d$, then S_1 may forward the request to S_3 based on the FIB information at that time. But when the request arrives at S_3 after d, the FIB entry of S_3 has been removed. As a result, S_3 cannot forward the request, so it returns an NACK back. When the NACK arrives at S_1 after one hop, although the FIB of S_1 has been updated, it can re-forward the request through another interface. Moreover, if the time $t_{req} + 2d$ is between $t_{mig} + d$ and $t_{mig} + 2d$, the request experiences the same

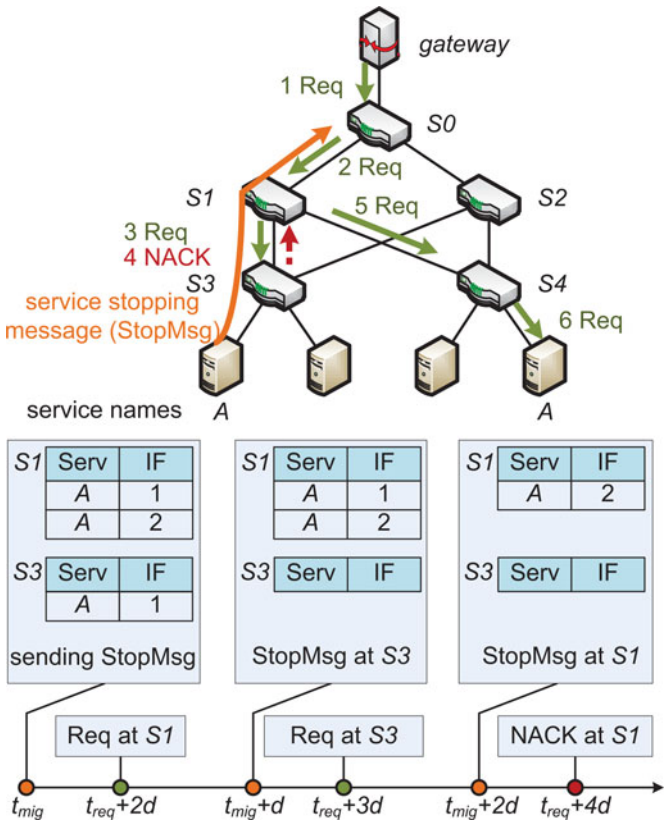


Fig. 10. An example of a request incurring a request NACK at the layer of ToR switch.

process. Thus, we conclude that if Eq. (3) is satisfied, then the request may incur a request NACK at the ToR layer.

In general, there are two necessary conditions for a request incurring a request NACK at a routing node: 1) the request arrives at the node later than or equal to the time when the node updates its FIB, that is, when the node receives the service stopping message; 2) the request arrives at the previous-hop node earlier than the stopping message. In Fig. 11, we show the arriving time of the request and the stopping message at each layer of nodes. Then we can obtain two necessary conditions. For the example in Fig. 10, the first condition requires that $t_{req} + 3d \geq t_{mig} + d$ and the second condition requires that $t_{req} + 2d < t_{mig} + 2d$. From them, we can obtain Eq. (3).

In this way, we can obtain that a request incurs a request NACK at a VM if

$$t_{mig} - 4d \leq t_{req} < t_{mig} - 2d, \tag{4}$$

and a request incurs a request NACK at an aggregation switch if

$$t_{mig} \leq t_{req} < t_{mig} + 2d. \tag{5}$$

Thus, the requests dispatched during the period $[t_{mig} - 4d, t_{mig} + 2d]$ with a length of $6d$ may incur the request NACKs. Let l_{VM} denote the load (i.e., average number of requests per time unit) toward the migrated VM. Then, the number of affected requests has an upper bound of

$$l_{\text{VM}} \cdot 6d. \tag{6}$$

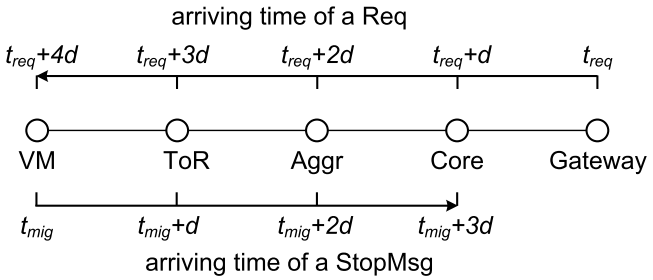


Fig. 11. The arriving time of a request (Req) and a service stopping message (StopMsg) at each layer of nodes.

The same upper bound can be obtained for the internal traffic in the same manner.

The request NACKs result in extra delay to serve requests. For the example in Fig. 10, a request NACK is returned from ToR switch S_3 to aggregation switch S_1, and it causes two extra hops for a round-trip. In the worst case, aggregation switch S_1 also has no interface to re-forward the request when it receives the NACK, so it also has to return an NACK to core switch S_0. As such, four extra hops are resulted for a round-trip. Similarly, in the worst case, six extra hops are resulted for a request incurring a request NACK at a VM, and two extra hops are resulted for a request incurring a request NACK at an aggregation switch. The similar results can be obtained for internal requests. In conclusion, the VM migration policy is seamless in the named-service framework, and it results in small extra delay for a small number of requests.

5.4 Performance in IP-Based Networks

Finally, we explain that although the similar VM migration policy can be used in the IP-based networks, it would result in service interruptions.

Recall that in the IP-based networks, a central proxy is responsible for binding the IP address and the location-specific address of each VM. The IP address is used for identifying a VM, while the location-specific address is used for routing to a VM. As such, once a VM is migrated to another location, its IP address is retained, while its location-specific address is changed. In this IP-based environment, a similar VM migration policy can be used. Instead of propagating the service stopping message and the service starting message to the switches, these messages are sent to the central proxy. Once the proxy receives the messages, it updates the corresponding binding information, so that the requests will be routed to the new location afterwards. It is noted that the central proxy has to keep the binding information within the VMs communicating with the migrated VM consistent with its own. Otherwise, via the inconsistent binding information, the internal requests between two VMs would be routed to wrong locations.

However, this migration policy could not avoid service interruptions. This is because in IP-based networks, the route is predefined rather than determined on the fly as in our framework. Thus, when a request arrives at the old location of a migrated VM, if this VM has stopped receiving new requests, then the request could neither be responded by the migrated VM nor be routed to another serviceable VM. Suppose the gateway connected to all the core switches operates as a central proxy. We can analyze how many requests

would not be responded in the same manner as previous. For the external requests, we conclude that the requests that are distributed to the migrated VM and are dispatched by the gateway during the period $[t_{mig} - 4d, t_{mig} + 4d]$ would not be responded, where t_{mig} and d are defined the same as previous. For the internal requests from other VMs to the migrated VM, the delay to maintain the consistency increases the amount of the requests not responded. Since the requests that are routed with the outdated binding information could not be responded.

To summarize, the performance gain of our method is significant when the request load is heavy. It is particularly true when VM migrations occur across subnets. Even for migrations within subnet, no penalty is incurred.

6 Performance Optimization

In this section, we introduce a distributed load balancing algorithm to improve the performance of the routing protocol, to avoid congestion and hot spots in particular.

Our algorithm is based on the field of capacity maintained in FIB. In a FIB entry $\langle$*service name, interface, capacity*$\rangle$, the field of capacity records the total capacity (i.e., the average maximum number of requests served per time unit) of all the VMs that provide the service and are reachable through the interface in this entry. It is maintained via the control message protocol as discussed in Section 4.3. In addition, we assign an equal capacity value to all the candidate interfaces that are connected to the upper-layer switches, in order to use the load balancing algorithm in this case.

The load-balancing decision for a service at a switch can be formulated as an optimization problem. Let l denote the request load (i.e., the average number of requests arriving per time unit) for the service arriving at the switch. Suppose there are n candidate interfaces to distribute the load, and each interface has an average capacity, that is, the ith interface has capacity c_i. Then, our problem is to distribute the load to the candidate interfaces such that the maximum utilization on the interfaces (denoted by u) is minimized:

$$\begin{array}{lll} \min & u & \\ s.t. & \sum_{i=1}^{n} l_i = l, & \\ & l_i/c_i \le u, & i = 1, \ldots, n, \\ & l_i \ge 0, & i = 1, \ldots, n. \end{array} \tag{7}$$

This problem is a classic resource allocation problem, widely observed in resource scheduling and network optimization. Many algorithms can solve it. We adopt *Weighted Round-Robin Algorithm* (WRRA) for its simplicity. In this algorithm, each interface has a weight w_i, which is equal to $c_i/\mathrm{GCD}(c_1, \ldots, c_n)$. GCD is greatest common divisor. This weight represents how many requests can be forwarded by an interface in each round. At the beginning of a round, each interface is assigned tokens whose number is its weight. At each round, the algorithm forwards an incoming request to each interface in a circular order unless an interface runs out of tokens. A round keeps until all interfaces run out of tokens.

7 Performance Evaluation

In this section, we evaluate the performance and demonstrate the advantages of our framework via extensive simulations.

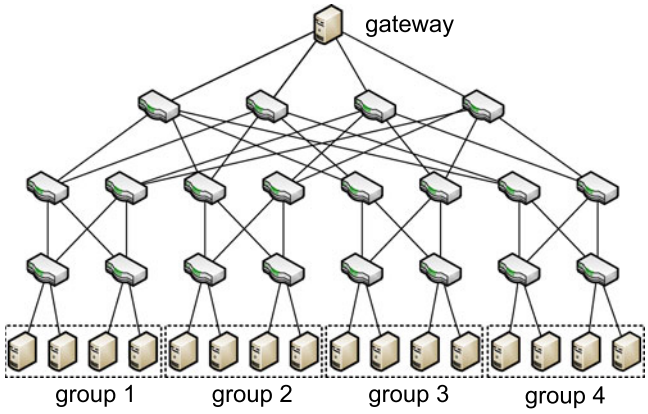


Fig. 12. The simulation topology.

7.1 Simulation Settings

We implement a simulation testbed based on ndnSIM, an ns-3 based NDN simulator [21]. The testbed has a fat-tree topology. It consists of 16 physical machines, three layers of switches, and a gateway which is connected to all the core switches, as shown in Fig. 12. We let each PM host a VM. Each switch has a buffer size of 20 packets. Each link has a bandwidth of 100 Mbps and a transmission delay of 1 ms (if not explicitly stated). The gateway dispatches a load of external requests. We set the request packet and the response packet to 30 and 1,024 bytes, respectively. In this testbed, we simulate the VM migration by a simple procedure, which consists of sending the service starting message, sending the service stopping message and a short interval between them. Other details, such as memory copy and image boot, are omitted.

The service time consumed by a VM to process a request follows an exponential distribution with an average of *1/capacity*. When a VM is busy, a request is added into its queue to wait the service. The maximum length of the queue is set to 500, to avoid the request loss due to the queue overflow in the simulations. The request queue is small in size since the length of a request packet is short. For example, it is 15 MB in our case. Thus, it is easy to increase the queue size to avoid queue overflow in practice.

This testbed serves two purposes: 1) to verify the named-service framework; 2) to evaluate the performance of the VM migration policy and the load-balancing algorithm.

7.2 Adaptive Named-Service Routing

We first verify that the named-service routing protocol can adapt to system dynamics, by varying the VMs' capacity. Specifically, we group every four neighboring physical machines into a cluster, as shown in Fig. 12. We assign a default capacity of 20, 40, 100 and 160 requests per second (req/s) to the VMs in the four groups, respectively. Then, at the 30th second we change each VM's capacity to 80 req/s, and at the 60th second we change the capacities of the VMs in the four groups to 160, 100, 40 and 20 req/s, respectively. In addition, we generate a Poisson-arrival load profile with an arrival rate of 640 req/s. That is, the inter-request time of the load follows an exponential distribution with an average of 1/640 s. The load is dispatched by the gateway. We repeat the simulation for 20 times, each running for 90 seconds, and report the average results.

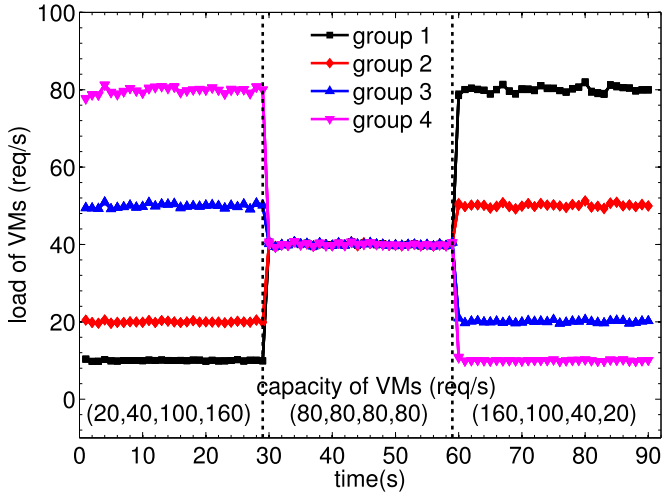


Fig. 13. The allocated load of individual VMs via the load balancing algorithm. The load is dispatched by the gateway. It is a Poisson process and has an average arrival rate of 640 req/s.

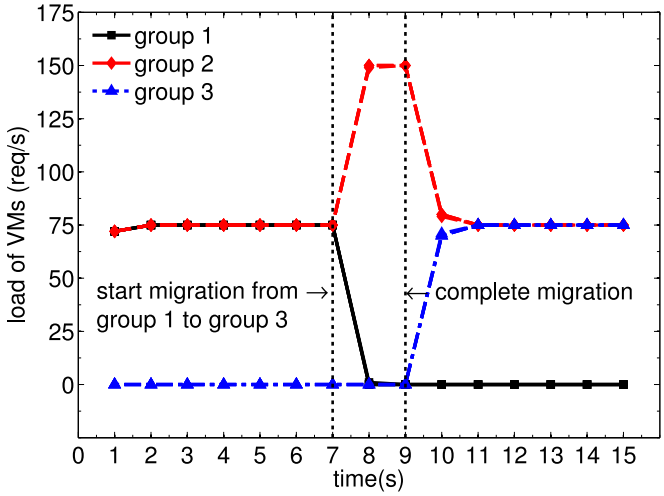


Fig. 15. The allocated load of individual VMs as the VMs are migrated from group 1 to group 3. The migration starts at 7th second, and ends at 9th second.

We show the allocated load of individual VMs in Fig. 13. We observe that the allocated load for each VM closely follows its service capacity. This clearly suggests that the named-service routing protocol in our proposed framework adapts to the system dynamics.

We also verify that the load balancing algorithm is effective. We define the utilization of a VM as the percentage of its busy time. We measure the utilization per second of individual VMs. The above simulation is repeated for 20 times, each running for 90 seconds. We first average the utilization per second over 20 times. Then, we report the statistics, that is, mean and standard derivation, of the utilization per second of all the VMs throughout the runtime in Fig. 14. We observe that the average utilization of the VMs is close to the optimum, that is, 0.5 in our case.

7.3 VM Migration Policy

In this section, we evaluate the performance of the VM migration policy. Same as above, we group every four neighboring physical machines into a cluster. At the beginning, we assign a VM on each physical machine in the first two groups. All the VMs provide the same service, and have the same capacity of 300 req/s. The gateway dispatches a constant request load of 600 req/s. We set the single-hop transmission delay to 10 ms. We migrate all the VMs in group 1 to the physical machines in group 3. The migration starts at 7th second, and ends at 9th second. We conduct a series of simulations to evaluate the load of individual VMs, the number of the requests that incur request NACKs, and the resulted delay.

First, we show the load of individual VMs in Fig. 15. It illustrates that the allocated loads on individual VMs follow the change of their capacities, as the VMs migrate. Specifically, the load allocated to the VMs in group 1 becomes 0 when the migration starts; all of the load is allocated to group 2 during the migration; and the load is distributed equally in groups 2 and 3 when the migration finishes.

Second, we evaluate the number of the requests that incur request NACKs and the resulted delay. As discussed in Section 5, this is because the upper-layer switches receive the service stopping message and update their FIBs latter than the lower-layer switches due to the transmission delay. We show the hop counts of individual request-response

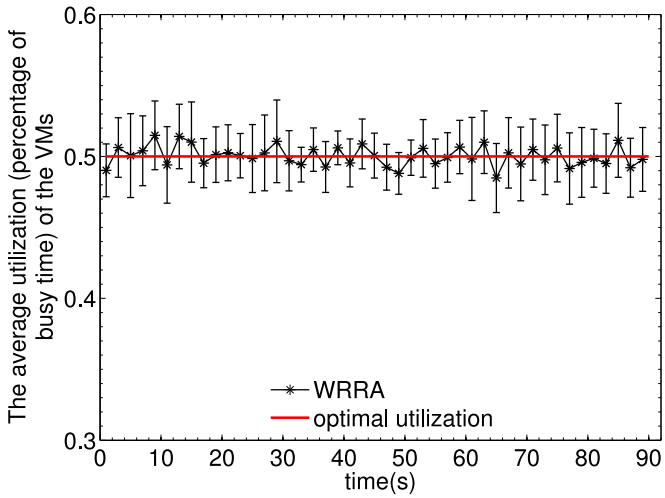


Fig. 14. The average utilization (percentage of busy time) of the VMs via the load balancing algorithm, where the bars demonstrate the standard deviation.

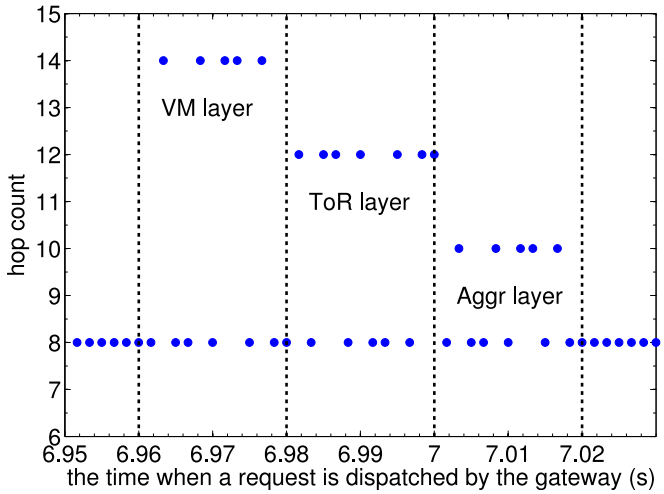


Fig. 16. The hop counts of individual request-response pairs.

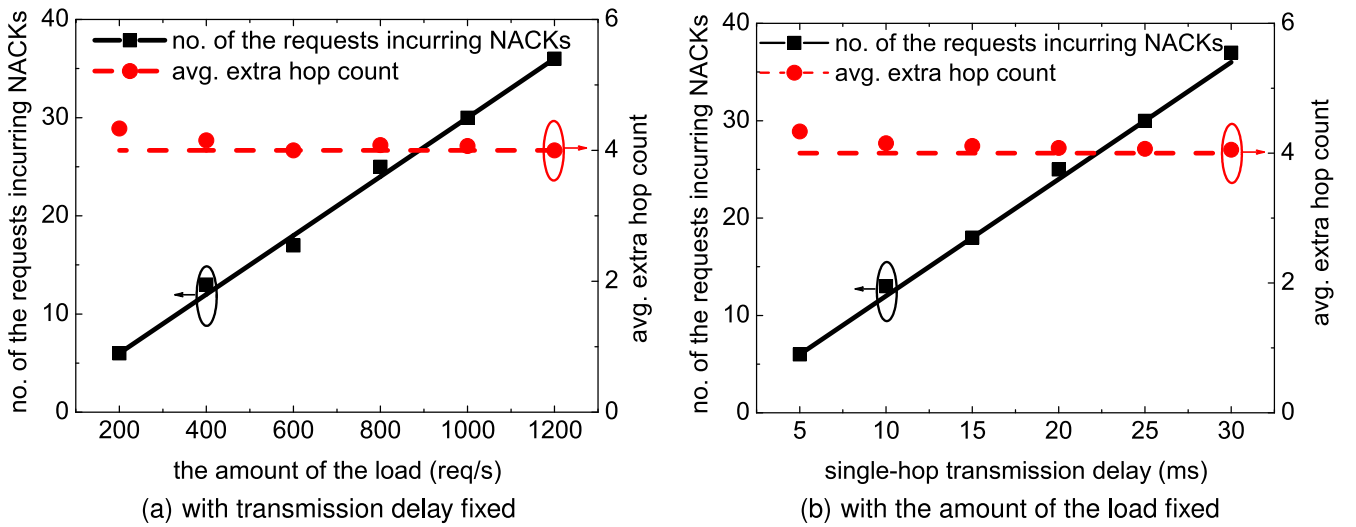


Fig. 17. The number of the requests that incur NACKs and the resulted extra hop count on average, as the amount of the load dispatched by the gateway or the single-hop transmission delay varies.

pairs around the migration time in Fig. 16. The x-axis is the time when a request is dispatched by the gateway, i.e., t_{req}. Recall that the time when the migration starts, i.e., t_{mig}, is 7 s and the single-hop transmission delay is 10 ms. As analyzed in Section 5.3, the requests that are dispatched in three intervals incur request NACKs at three layers of nodes, respectively. The three intervals are [6.96, 6.98), [6.98, 7) and [7, 7,02) and the three layers of nodes are VM, ToR switch and aggregation switch, respectively. Moreover, from the analysis, the resulted extra delay in the three cases are six, four and two hops, respectively. Since it takes eight hops for a request dispatched by the gateway to receive a response without any NACK. Thus, the delay of the request-response pairs in the three cases are 14, 12 and 10 hops, respectively. It is observed that the simulation results shown in Fig. 16 are consistent with these analysis results. It is also observed that in the three intervals only half of the requests incur request NACKs. This is because, the requests are distributed equally to the VMs in two groups via the load balancing algorithm, so only half of the requests that are distributed to the migrated VMs in the first group incur request NACKs. In conclusion, this simulation suggests that the migration policy is seamless, and only results in a short extra delay for a small number of requests.

Finally, we evaluate the impacts of the single-hop transmission delay and the amount of the load dispatched by the gateway on the performance of the VM migration policy. In this simulation, we first fix the transmission delay to 10 ms, and vary the amount of the load. Then we fix the amount of the load to 400 req/s, and vary the transmission delay. We show the number of the requests that incur request NACKs and the resulted extra hop count on average in Fig. 17. It is observed that the number of the requests that incur NACKs is about three times as many as the product of the transmission delay and the amount of the load. For example, when the transmission delay is 10 ms and the amount of the load is 400 req/s, the number of the requests incurring NACKs is about 12. Recall that only half of the load is distributed to the migrated VMs in group 1. Thus, the simulation result conforms with the analysis result in Eq. (6). In addition, it is observed that the resulted extra hop count on average is four hops, which also conforms with the analysis result.

7.4 Mechanism of Route Learning

In this section, we evaluate the mechanism of route learning in the case of link failures. Via this mechanism, when an NACK is received from an interface connected to an upper-layer switch, the unreachable interface is recorded. We compare it with the basic NACK mechanism, in which the unreachable interface is not recorded.

In this simulation, we use the topology as shown in Fig. 18. It consists of 8 VMs and 10 switches $S_1 \sim S_{10}$. The leftmost VM provides service A and the rightmost four VMs provide service B. Suppose service A would like to request service B. Service A sends a constant load of 600 req/s, and each service B provides an average capacity of 300 req/s. Suppose the link between switches S_1 and S_5 fails (red cross in the figure). As a result, switch S_1 do not know service B, by using the control message protocol. Thus the NACK mechanism has to be used to route the requests. We set the single-hop transmission delay to 1 ms, and run the simulation for 3 s.

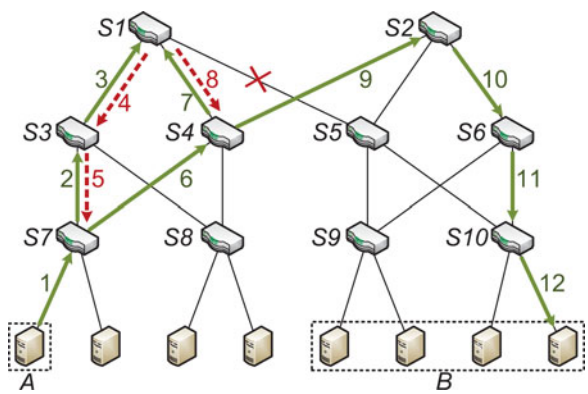


Fig. 18. The simulation topology for the mechanism of route learning, where the link between switches S_1 and S_5 fails (red cross). The arrows show a longest path to route a request from service *A* to service *B*, where the solid green ones represent forwarding requests and the dashed red ones represent forwarding NACKs respectively, and the numbers beside them represent the order of forwardings.

TABLE 1
The Percentage of the Requests Sent by Service *A* Taking Various Hops to Arrive at Service *B*

mechanism	6 hops	8 hops	10 hops	12 hops
with route learning	99.7	0.2	0.1	0
w/o route learning	0	0.2	0.1	99.7

We evaluate the first 1,000 requests that receive responses, and measure the percentage of the requests that take various hops to arrive at service B. We show the results in Table 1. It is observed that, by using the mechanism of route learning, 99.7 percent of requests take six hops to arrive at service B, by the shortest paths (e.g., service $A \rightarrow S_7 \rightarrow S_4 \rightarrow S_2 \rightarrow S_6 \rightarrow S_{10} \rightarrow$ service B); in comparison, when this mechanism is not used, 99.7 percent of requests take 12 hops to arrive at service B. The results demonstrate that the mechanism of route learning can reduce the delay to route requests.

When the mechanism of route learning is not used, the majority of requests take the longest paths (12 hops), as shown by the arrows in Fig. 18. This is because, at switch S_7, once a request is re-forwarded through the reachable interface, a new request is forwarded through the unreachable interface by the load balancing algorithm. Thus almost all requests are forwarded through the unreachable interface first and then re-forwarded through the reachable interface. The same procedure happens at switches S_3 and S_4. As a result, almost all requests take 12 hops to arrive at service B.

7.5 Scalability

Finally, we evaluate the scalability of our system in various scale of networks. We simulate the networks that are built with four-port switches, eight-port switches and 12-port switches respectively. A fat-tree topology that consists of k-port switches has $k^3/4$ physical machines [20], thus these three topologies have 16, 128 and 432 physical machines respectively. Same as in Section 7.2, we evaluate the average utilization. We let each PM host a VM, and set each VM's capacity to 80 req/s. In addition, we generate a Poisson-arrival load profile with an arrival rate of $160k$ req/s, such that the optimal utilization is 0.5. We show results in Fig. 19. It is noted that the performance keeps stable as the time evolves, so we only illustrate the results in three time slots. As shown in Fig. 19, the utilization is close to the optimum in large-scale networks.

8 Conclusion

In this paper, we investigate the problem of seamless VM migrations in the DCN. Leveraging the benefit of decoupling a service from its physical location in the emerging technique named data networking, we propose a named-service framework to support seamless VM migrations. In comparison with other approaches, our approach has following advantages: 1) the VM migration is interruption-free; 2) the overhead to maintain the routing information is less than that caused by classic NDN; 3) the routing protocol is robust to both link and node failures; 4) the framework inherently supports the implementation of a distributed load balancing algorithm, via which requests are distributed to VMs in balance. The analysis and simulation results verify these benefits.

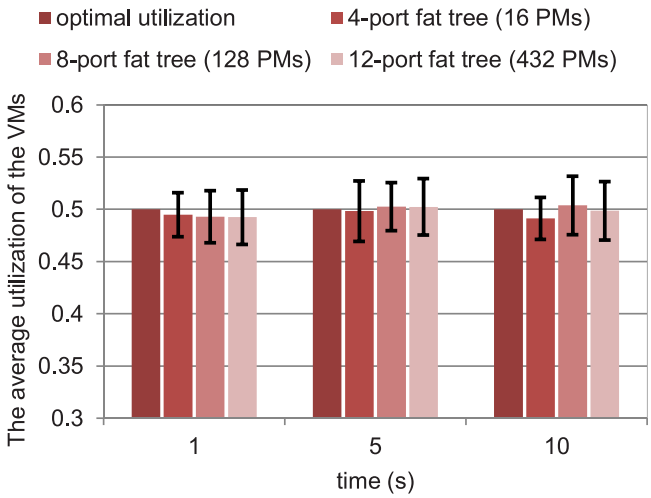


Fig. 19. The average utilization of the VMs in various scale of networks, where the bars demonstrate standard deviation.

9 Acknowledgments

This work was supported by grant from Research Grants Council of Hong Kong [Project No. CityU 114713], and was partially supported by Singapore EMA under Energy Innovation Research Programme (EIRP02)-Smart Grid. Ruitao Xie is the corresponding author.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003.

[2] Y. Wen, X. Zhu, J. Rodrigues, and C. W. Chen, "Cloud mobile media: Reflections and outlook," *IEEE Trans. Multimedia*, vol. 16, no. 4, pp. 885–902, Jun. 2014.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. 2nd Conf. Symp. Netw. Syst. Des. Implementation*, 2005, pp. 273–286.

[5] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *Proc. ACM/IEEE Conf. Supercomput.*, 2008, pp. 53:1–53:12.

[6] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *Proc. INFOCOM*, Apr. 2011, pp. 1098–1106.

[7] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 67–74, Jan. 2010.

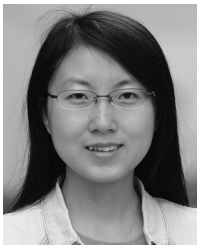
[8] E. Silvera, G. Sharaby, D. Lorenz, and I. Shapira, "IP mobility to support live migration of virtual machines across subnets," in *Proc. The Israeli Exp. Syst. Conf.*, 2009, pp. 13:1–13:10.

[9] K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Live data center migration across WANs: A robust cooperative context aware approach," in *Proc. SIGCOMM Workshop Internet Netw. Manag.*, 2007, pp. 262–267.

[10] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *Proc. 3rd Int. Conf. Virtual Execution Environ.*, 2007, pp. 169–179.

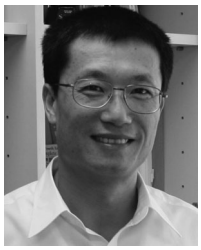
[11] E. Harney, S. Goasguen, J. Martin, M. Murphy, and M. Westall, "The efficacy of live virtual machine migrations over the internet," in *Proc. 2nd Int. Workshop Virtualization Technol. Distrib. Comput.*, 2007, pp. 8:1–8:7.

[12] V. Mann, A. Vishnoi, K. Kannan, and S. Kalyanaraman, "CrossRoads: Seamless VM mobility across data centers through software defined networking," in *Proc. IEEE Netw. Oper. Manag. Symp.*, 2012, pp. 88–96.
[13] Microsoft, "Whitepaper: Windows server 2012 server virtualization," 2012.
[14] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, Aug. 2009.
[15] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th Int. Conf. Emerging Netw. Exp. Technol.*, 2009, pp. 1–12.
[16] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Comput. Commun.*, vol. 36, no. 7, pp. 779–791, 2013.
[17] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Y. Ko, J. Rexford, and M. J. Freedman, "Serval: An end-host stack for service-centric networking," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, pp. 7–7.
[18] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.
[19] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 267–280.
[20] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 63–74.
[21] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," University of California, Los Angeles, NDN, Tech. Rep. NDN-0005, Oct. 2012.



Ruitao Xie received the PhD degree in computer science from the City University of Hong Kong in 2014, and BEng degree from the Beijing University of Posts and Telecommunications in 2008. She is currently a senior research associate in the Department of Computer Science at the City University of Hong Kong. Her research interests include cloud computing, distributed systems, and wireless sensor networks.

Yonggang Wen (S'99-M'08-SM'14) received the PhD degree in electrical engineering and computer science (minor in western literature) from the Massachusetts Institute of Technology (MIT), Cambridge, USA. He is an assistant professor with the school of computer engineering at Nanyang Technological University, Singapore. Previously, he has worked in Cisco to lead product development in content delivery network, which had a revenue impact of three Billion US dollars globally. He has published more than 100 papers in top journals and prestigious conferences. His latest work in multi-screen cloud social TV has been featured by global media (more than 1,600 news articles from over 29 countries) and recognized with ASEAN ICT Award 2013 (Gold Medal) and IEEE Globecom 2013 Best Paper Award. He serves on the editorial boards for the *IEEE Transactions on Multimedia*, *IEEE Access Journal*, and *Elsevier Ad Hoc Networks*, and was elected as the chair for IEEE ComSoc Multimedia Communication Technical Committee (2014-2016). His research interests include cloud computing, green data center, big data analytics, multimedia network, and mobile computing. He is the senior member of the IEEE.



Xiaohua Jia received the BSc degree in 1984 and the MEng degree in 1987 from the University of Science and Technology of China, and the DSc degree in 1991 in information science from the University of Tokyo. He is currently a chair professor with the Department of Computer Science at the City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, and mobile computing. He is an editor of the *IEEE Internet of Things*, *IEEE Transaction on Parallel and Distributed Systems* from 2006 to 2009, *Wireless Networks*, *Journal of World Wide Web*, *Journal of Combinatorial Optimization*, etc. He is the general chair of ACM MobiHoc 2008, a TPC co-chair of IEEE GlobeCom 2010 Ad Hoc and Sensor Networking Symposium, an area-chair of IEEE INFOCOM 2010 and 2015. He is the fellow of the IEEE.

Haiyong Xie received the B.S. degree from USTC in 1997, and the MS and PhD degrees in computer science in 2008 and 2005, respectively, both from Yale University. He is currently the executive director for the Cyberspace and Data Science Laboratory, the Chinese Academy of Electronics and Information Technology and a professor in the School of Computer Science and Technology at University of Science and Technology of China (USTC) in Hefei, China. Prior to his present position, he was the principal researcher for Huawei US Research Labs and the P4P Working Group (P4PWG) and distributed computing industry association. He proposed P4P (proactive provider participation in P2P) to coordinate network providers and peer-to-peer applications in a seminal paper published in ACM SIGCOMM 2008, led and conducted original research and large-scale tests on P4P. Encouraged by and based upon his research and results on P4P, the P4PWG was formed to promote academic studies and industrial adoptions of P4P, which was later adopted by IETF to form a new Application Layer Traffic Optimization (ALTO) Working Group. His research interest includes content-centric networking, software-defined networking, future Internet architecture, and network traffic engineering.