# Automatically configuring the network layer of data centers for cloud computing

C. Hu
M. Yang
K. Zheng
K. Chen
X. Zhang
B. Liu
X. Guan

*With the requirement of very large data centers for cloud computing, the challenge lies in how to produce a scalable and inexpensive design to interconnect a large number of servers in data centers while providing fault tolerance and high network capacity. Internet Protocol (IP)-based data center networks (DCNs) can support a sufficiently large number of machines and dynamic multi-path routing. However, the complex and error-prone manual configuration of IP-based DCNs hinders its deployment. In this paper, we propose DCZeroconf, a fully automatic IP address configuration mechanism, to eliminate the burden of manual configurations of the IP addresses of servers and switches in data centers. We have implemented a prototype system to validate the effectiveness of DCZeroconf via extensive experiments and simulations. The evaluation results demonstrate that DCZeroconf supports different topologies, and the assigned IP addresses can be automatically adjusted upon dynamic topology changes. In addition, the entire automatic process can be completed in seconds or less.*

## Introduction

The design of a more efficient and scalable data center network (DCN) has attracted tremendous interest in both research and operation communities, where the physical topology, addressing, and routing mechanisms are accentuated as the three primary issues to be considered [1, 2]. In general, most of the recent research proposals follow four directions. First, the scaling and multi-path routing problem is solved by adding Layer 3 Internet Protocol (IP) switches or routers to inherit the scalability characteristics and equal-cost multi-path (ECMP) forwarding features from IP networks. Given the proven spectacular success of IP networks for interconnecting billions of hosts in the Internet, many practical data centers are already constructed from multiple IP subnets [3, 4]. The second direction involves the construction of DCNs based on Ethernet, because of its low cost, high bandwidth, preset addressing, and automatic configurations. However, traditional Ethernet and its spanning-tree-based forwarding cannot support two required features of data centers: large-scale characteristics (thousands of millions of servers) and multi-path forwarding [5]. Therefore, the Ethernet proponents seek methods, such as EtherProxy [6], SEATTLE [7], and SPAIN [5], to scale the data centers and to utilize the redundant physical paths. The third direction is the new design of the physical DCN topology construction and addressing, as well as the routing algorithm, e.g., Fat-Tree [1], DCell [2], and BCube [8]. Each of these designs usually provides a unique interconnection topology, designs a specific addressing mechanism based on the topology, and proposes a specific routing mechanism according to the topology and the addressing. Most recently, a fourth direction of study focuses on a way to build hybrid electrical/optical data centers by making use of the very large bandwidth advantage of optical circuit switching technologies such as C-Through [9] and Helios [10].

Compared to other categories of approaches sketched above, IP-based DCNs can elegantly support a sufficiently large number of machines, along with dynamic, flexible load balancing and routing with available protocol stacks and devices. However, an underlying drawback of the IP-based large-scale DCNs is the complex configurations of its network layer. The IP addresses of the servers and intermediate Layer 3 switches or routers (in the remainder of this paper, we use the general term "switch" to refer to a Layer 3 switch) need to be configured before the routing

Digital Object Identifier: 10.1147/JRD.2011.2165751



**0018-8646/11/$5.00 © 2011 IBM**

table can be automatically computed. The configuration becomes more complex, difficult to manage, and error-prone for the network administrators, especially when a large number of virtual machines (VMs) dynamically migrate among physical servers. In this paper, we investigate the solution of automatically configuring the network layer of IP-based DCNs.

## Motivations and goals

Several benefits motivate us to seek a fully automatic solution that can configure the IP addresses for VMs, servers, and switches inside the IP-based DCNs. First, the cost of maintaining a data center, especially the overhead on manual IP address configuration, which currently accounts for 15%–30% of total capital expense,[1] can be significantly reduced. Second, manually configuring today's data networks is error-prone [11], and curbing manual configuration activities in DCNs can substantially minimize the risk of human errors. Third, the scale of DCN changes according to the customer's demand on applications, which requires frequent configuring and re-configuring of the IP addresses of the machines inside the DCN; therefore, the cost and overhead of evolving (i.e., changing the scale of) the DCNs can be reduced by an automatic address configuration mechanism.

The goal of this paper is to design a mechanism that requires zero manual intervention during the configurations on the IP addresses of DCNs with the following specific features. First, each VM/server and each switch port can be automatically and correctly configured with a proper IP address regardless of what physical topology is employed to interconnect the machines. Second, when the DCN topology changes, e.g., due to plugging in or removing a VM/host/switch, the IP addresses can be adaptively adjusted upon the change. Third, the mechanism is scalable to a large number of devices, e.g., hundreds of thousands of machines, and the configuration time for such a large DCN is as fast as seconds or less. Finally, the mechanism should be easy to deploy in today's DCNs and future DCNs. The main contribution of this paper is the design and implementation of an automatic address configuration mechanism for IP-based data centers, which achieves the above goals.

## Related work and challenges

Dynamic Host Configuration Protocol (DHCP) [12] is the most widely deployed scheme for automatic configuration of IP addresses within the same subnet. One or more DHCP servers are employed to record the available IP addresses and to eliminate allocation conflicts of addresses to hosts. When a new host joins the subnet, the host seeks a DHCP server and then requests an unused IP address from the DHCP server. The hosts and the DHCP servers should be in the same subnet so that the broadcast of the DHCP protocol messages can be received by a requesting host and the DHCP server. To apply DHCP into data centers, the configuration of the DHCP servers in each subnet, or the creation of virtual network cards on a single DHCP server for each subnet, also requires manual efforts in all the subnets, and the number of the subnets could be in the thousands. In addition, if the DHCP relay were employed in a large-scale date center with hundreds of thousands of machines, the inefficient global broadcast of DHCP messages would be a large traffic burden.

Zeroconf [13] can also be used for automatically assigning IP addresses. A Zeroconf-enabled host first randomly selects an address and validates its availability by broadcasting queries to the network. The address will be reserved for the host if no reply shows that the address has already been occupied; otherwise, the host randomly selects another address and repeats the validation process. Since the number of servers in a DCN could be as large as hundreds of thousands, naive modifications on switches to allow Zeroconf broadcasting inside such a large DCN will be costly and inefficient.

In the context of DCNs, Portland [14] and DAC [15] can automatically assign addresses. Portland develops a distributed location discovery protocol (LDP) to allocate the physical media access control (PMAC) addresses of servers and switches. PMAC is the private address dedicated for multi-rooted tree topology, e.g., Fat-Tree [1]. Therefore, the specific design of Portland on PMAC is not easily extended to other network topologies. DAC has a centralized server that learns the physical topology of the network and maps it to the logical addresses with the help of a blueprint. DAC is a generic method to configure all kinds of addressing on an arbitrary topology; however, it focuses on the initial setup of a DCN and does not seriously consider the cases in which the data center topology changes. To solve the topology-change problem with DAC, manual intervention is still required to input a new blueprint, which depicts the changed topology and corresponding addresses. Furthermore, the configurations of VMs by DAC are also left unknown in its proposal.

By reviewing the related work in the literature, we summarize the following challenges to achieve all the goals mentioned above. First, for scalability reasons, large-scale data centers are divided into a considerable number of subnets (e.g., thousands of subnets), and this feature limits the power of DHCP or Zeroconf in DCN, which can only solve the address conflicts in a single subnet.

Second, assigning IP addresses to intermediate devices like switches and routers also constitutes an indispensable configuration procedure to enable the devices to forward ordinary IP packets. However, the traditional DHCP or Zeroconf only configures the IP addresses for end hosts but not for the intermediate IP switches or routers used to build up the underlying communication channels.

[1] This information comes from an internal IBM report on information technology (IT) for cloud computing in 2009.

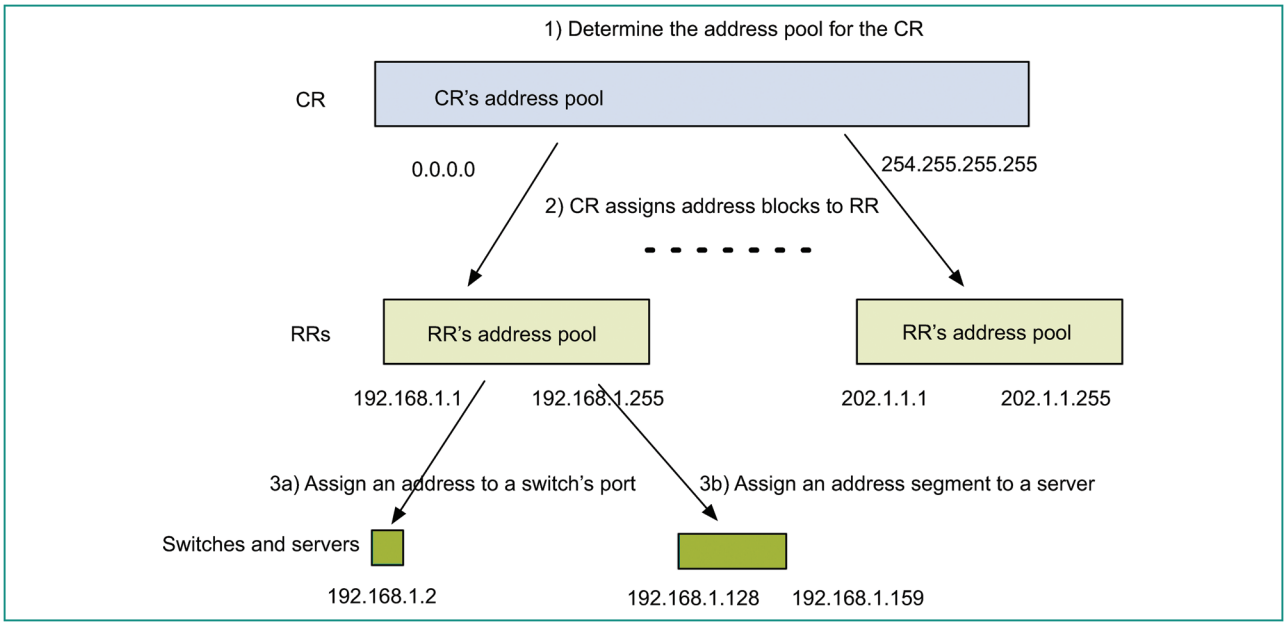


Figure 1

Addressing hierarchy.

Third, in the context of cloud computing, emerging applications or services will be constantly changed or reconfigured by the providers to adapt to new customer demands or marketing feedbacks [3]. Currently, the topology change is not uncommon in the data centers. New servers will be added to a data center if the computing demand on the data center is increased. As the demand continues to increase, the scale of the DCNs continues to evolve. Therefore, it is crucial, yet challenging, that the IP address configurations can automatically adapt to such topology changes.

## System overview

In this paper, we propose DCZeroconf to automatically configure the network layer of DCN. Before delving into the design details, we first define the terminologies as follows. In modern data centers, the servers are placed in *racks* and the servers in the same rack are usually connected to the same switch. Such a switch is known as a *top-of-rack* (TOR) switch. The servers and TOR switches form the access layer of a data center, and to connect the TOR switches in an access layer, more Layer 3 switches are employed. For the topology used to connect the racks, DCZeroconf makes no specific assumptions about it.

DCN should support rapid VM migration, which in turn calls for separating names from locations. We follow the naming convenience discussed in [3], where an application-specific address (AA) and a location-specific address (LA) are maintained for each VM. DCZeroconf will assign a LA to a physical server, and the LA is routable inside the data center. An AA will be assigned to a VM, with which the applications can identify a service running on a specific server/VM. When a VM is migrated from one server to another, the routable LA has to be changed, while the AA remains the same. Also, the mapping between the AA and LA will be changed, and the directory service will keep the latest mapping between AA and LA. With the mapping between an AA of a specific server/VM and the LA of the physical server it locates, one can find and reach the service running on that server/VM.

The DCZeroconf system is a two-tier mechanism that automatically configures the IP addresses inside the data center. We have a top configuration server called a *Central Regulator* (CR) and a configuration server called a *Rack Regulator* (RR) in each rack. Note that one RR is required for each broadcast domain if several broadcast domains are in a same rack. **Figure 1** provides an overview of how DCZeroconf works and can be understood as follows.

In the first step, the network administrator determines an available IP address pool that can be used in the data center and provides it as input to the CR, and this is the only manual effort required by DCZeroconf during the automatic configuration. For the example in Figure 1, the IP addresses for the servers inside the data center could be employed for internal use only, and a Network Address Translation (NAT) is functioning at the gateway; thus, the entire 32-bit address space (for IPv4) can be the pool for the CR.

Next, the CR partitions the available IP addresses into blocks and informs each RR regarding the block(s) that can

be used to configure the servers and switches within the corresponding rack. For instance, the CR will send an address block 192.168.1.1–192.168.1.255 to a RR when receiving a request.

After the RR is configured, it then automatically assigns the IP addresses to each VM, server, and switch. We may continue with the example above for which the address block 192.168.1.1–192.168.1.255 is assigned to the RR. The RR selects one address from the block to configure itself first, e.g., 192.168.1.1. Then, the RR replies to the address configuration request from the port of any switch with one of the addresses in the block, e.g., 192.168.1.2. In addition, the RR will send an IP address segment to any server in the same rack after receiving an address configuration request from the requesting server, e.g., 192.168.1.128–192.168.1.159. Each VM in that server will be allocated with one IP address from the address segment.

During the procedure associated with DCZeroconf, the difficulty lies in how to coordinate the address assignment between 1) CR and RR, 2) RR and servers/VM, and 3) RR and switches.

### Address assignment between CR and RR

To assist the negotiation between the CR and RR, connections between each RR and the CR are required that form a control plane path over the data path for traffic forwarding. The introduction of such an extra control plane is not associated with a large cost, and many literature studies make use of a center regulator/scheduler to control the switches/servers in a data center and indicate such a control plane [3, 14, 16]. In fact, there is no critical bandwidth requirement on a control plane, so that a commodity switch fabric can be used to construct the communication channel. Aside from connecting them with direct cables, a wireless networking with encrypted communication channel for configuration purposes could also be a viable option. Constructing such a wireless control plane network does increase capital investment, but it is relative small compared to the cost of manual efforts that could be reduced. Only one wireless network interface card (NIC) is needed for each rack, and in each rack there could be 30–126 servers. Suppose the cost of one server is $1,000, and the cost of one wireless NIC is $10. Even without considering the cost on the switches and routers, the increased cost compared to the cost of servers is only 0.008%–0.03%. In addition, the communication range of 802.11 could be 100 meters, i.e., it can serve a 20,000-$m^2$ square room, which is sufficient to accommodate a large DCN. Furthermore, there should be no concern with the configuration on wireless networking for each NIC. The same script could be copied to each RR and CR to connect to the access point when the server boots.

A RR is a lightweight process that can be run in any of the servers in a rack. A RR first requests an IP address block (a number of continuous IP addresses) from the CR, which can be assigned to VMs, servers, and switches. On receiving a valid request, the CR assigns RR with the IP address block as well as the corresponding lease (length of time when the allocation is valid). The lease is employed to detect any physical detachment of a certain rack, and each RR should renew its contract with the CR before the lease expires. Otherwise the CR will reclaim the assignment, and the corresponding address block is available for reassignment. The procedure is typically initiated immediately after RR being booted, and a RR can start the assignment of the IP addresses to VMs/servers in the same rack and switches upon receiving the allocated address block from the CR. If any message in each step is lost, retransmission will be activated in order not to fail the procedure.

### Address assignment between the RR and servers/VMs

The servers/VMs and RR in a rack should be in a flat Layer 2 network, i.e., the same Layer 2 broadcasting domain such as Ethernet or a virtual local area network (VLAN). When a RR is allocated with an IP address block from the CR, it works in a similar way as a DHCP server to assign IP addresses to the local servers/VMs. The only difference with a DHCP server is that RR can assign multiple IP addresses in a batch to a single server in the same rack, while a DHCP server assigns only one IP address to a single host. This difference is due to the existence of several VMs in a single server.

As opposed to assigning one IP address for a VM each time, we propose a "batch mode" that assigns multiple addresses at once, which is more efficient. A server in a rack first initiates a broadcasting *Discover* message seeking the RR server. On receiving the *Discover* message, the RR sends an *Offer* message to notify the requesting server. Then, the server sends a *Request* message to request an address from the RR, and the RR in turn replies with an Acknowledgement (ACK) message to assign a number of addresses, i.e., an *address blocks*. To identify an address block that can be allocated to the VMs on the requesting server, the *ACK* message contains the first IP address and the length of the address block. If no address block is available, a *Negative Acknowledgement* (NACK) message will be generated. Upon receiving the address block on the server side, a hypervisor in the server, which allows multiple VMs to run concurrently on a server, performs the address assignment among the local VMs in the server.

Original DHCP can be an alternative option for the operation between the RR and servers. The servers and the RR in a rack are in a same Layer 2 network. When the RR is equipped with the IP address block from CR, in order to assign IP addresses to servers in the same rack, it works like a classic DHCP server. Servers in each rack act as DCHP-enabled clients. If there are several VMs in one server, each VM triggers an IP address assignment process via DHCP.

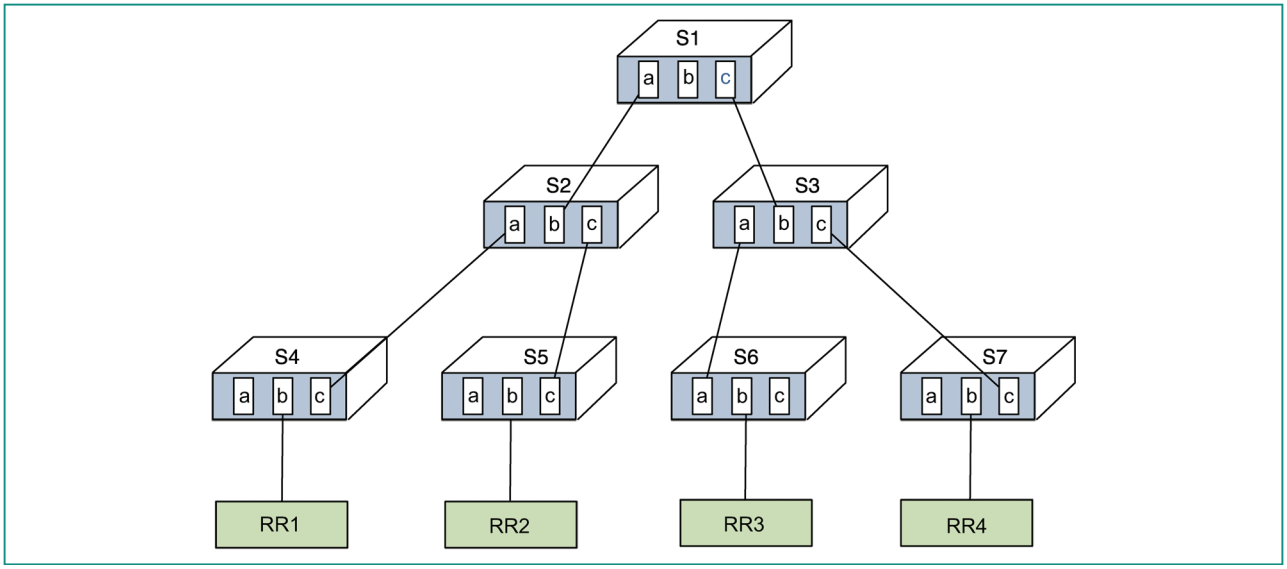


Figure 2

Example of a switch configuration. S1 through S7 in the figure are switches. In each switch, a, b, and c denote the different ports, and RR1 through RR4 are rack regulators.

Although the DHCP mechanism can be used between the RR and VMs, the "batch" mode mentioned provides a more efficient way that configures several machines simultaneously.

If one RR runs out of the available IP addresses that can be allocated to servers/VMs, it redirects the query message to another RR. The redirection follows a pre-determined order managed by a cyclic linked list and each RR is informed of the next RR to resort to when the CR configures the RR.

### *Address assignment between the RR and switches*

In this section, we first describe how the RRs and switches exchange messages followed by the communication among them to assign addresses. For the ease of presentation yet without loss of generality, we depict a simple tree-based topology in **Figure 2** as an example topology. S1 through S7 are switches, and a/b/c in each switch denotes the different ports. RR1 through RR4 are rack regulators.

The switches can query an arbitrary RR for obtaining available IP addresses. However, switches cannot form Layer 3 routing paths until they are assigned with IP addresses, and the switches also forbid the broadcast among themselves by default. A brute-force approach is to allow Layer 2 broadcasting of all the protocol messages in the entire DCN. Although the protocol messages in the control plane contribute relatively little traffic to the traffic in the data plane, brute-force broadcasting is inefficient and requires more time to perform the configuration. To enable a switch to communicate with a RR for requesting IP addresses, DCZeroconf follows a bottom-up approach and utilizes Layer 2 bridging to relay the DCZeroconf protocol messages among switches and RRs, akin to a path-vector protocol.

Each switch periodically sends a message to all its neighbors (i.e., servers and other switches), requesting "Can you reach a RR?" until the switch receives a reply indicating a valid next hop to a RR. During the bootstrapping phase, only the switches directly connected with a RR can receive a reply from the RR specifying, "I am a RR." Then, recursively, such a switch S will answer the requests from its neighbors with the path from S to the RR that S has discovered (S prepends itself in the path). If there is more than one path received by a certain switch S, only the first received path is used by S to reach the RR and is further propagated to its neighbors. The first received path is likely to indicate the path with the least delay from the switch to a RR, since switches relay answers with path information, hop-by-hop from the RRs. For example, consider Figure 2. S1 through S7 periodically send path queries. S4, S5, S6, and S7 first receive replies from RR1, RR2, RR3, and RR4, respectively. S2 will then receive path information of "S4 RR1" and "S5 RR2" from S4 and S5, respectively. S2 selects one of these two paths, and places itself in the path, i.e., "S2 S5 RR2."

RR only sends messages to switches after receiving messages from switches. When sending a message packet from a switch to the RR, the intermediate switches will add the path information to the packet. The path information can be treated as a stack. The source switch and all the intermediate nodes along the path to the RR will push their addresses to the stack. This path information is also kept in the packet back to

the switch from the RR. Thus, the reply messages from the RR to the switch always can find their reverse way, making use of the encapsulated path information in the packet. In the example of Figure 2, when S1 sends a message to RR, the packet carries the path information "S1 S2 S5 RR2." RR (here, in this example, is RR2) can compute the reverse path for its reply message "S5 S2 S1." The RR pops the first element and send reply packet to S5. Now, when S5 receive the packet, the reverse path information is "S2 S1." S5 also pops an element from the path stack and finds the next hop as S2. This process continues, and eventually the packet can reach S1. This part is similar to source-routing, and the switches do not keep a forwarding table but just examine the packets for the path information.

Address aggregation is usually used for the sake of scalability and is able to keep the routing table small. To utilize address aggregation for constructing a space-efficient routing table in each switch, the addresses of the two ends of a link between two switches are always within the same /31 prefix. (Here, the number following the slash is the prefix length, the number of shared initial bits, counting from the most-significant bit of the address. Thus, the /31 prefix is an address block with a 31-bit prefix.) Whenever a RR receives an address request from a switch S for a certain port p, RR returns two continuous IP addresses for this port and the port in the neighboring switch of S that connected with p. With the two returned IP addresses, the switch S selects one of them and attempts to assign this address to the connected port in the neighboring switch. The port of the neighboring switch accepts the assignment and configures its IP address as indicated in the message if this port has not received any address assignment from any RR. In case a certain port p of switch S receives two assignments from both the RR and a neighboring switch N, switch S compares the media access control (MAC) addresses of the two ends of the link. The address assigned from the switch port with a larger MAC address will be accepted. An ACK/NACK message will be then triggered to accept/decline the assignment from the RR or a neighboring switch.

Consider Figure 2 again as an illustration. S2.a requests a pair of addresses from the RR and the RR replies with a pair of addresses, $A_1$ and $A_2$. S2.a first selects one address, say $A_2$, to allocate to S4.b. If S4.b returns an ACK, S2.a configures itself with $A_1$ and sends an ACK to RR. In the case where S4.b sends another IP address, e.g., $A_3$ to S2.a, S2.a compares its MAC address with S4.b. If the MAC address of S4.b is larger than S2.a, S2.a selects $A_3$ as its address; otherwise, S2.a resends an address assignment message to S4.b, and an ACK is expected to arrive later.

If the RR does not find any available IP addresses that can be allocated, it redirects the query message to another RR. The redirection follows a predetermined order managed by the CR. The CR is aware of all the configured RRs, and it keeps a cyclic linked list for the RRs.

## Evaluations

The major performance metric of DCZeroconf is the time it takes to configure a given DCN. In this section, we first examine the configuration time on a small-scale testbed and then check the larger-scale DCNs via simulations. Moreover, experiments are also performed to investigate the configuration time of DCZeroconf when topology changes.

### *Experiments on testbed*

We first examine the performance of DCZeroconf on an eight-node prototyping testbed. In this testbed, we also emulate two racks, each of which has one RR and a server with two VMs. A CR is connected with two RRs, over a wireless connection. The wireless connection among CR and RRs is a 54-Mb/s 802.11g connection, and the connections among CR, RR, servers, and switches are all with 100-Mb/s Ethernet. In addition, three desktop computers are deployed with the eXtensible Open Router Platform (XORP) [17] and act as the Layer 3 switches. Switch no. 1 connects with Switch no. 2 and Switch no. 3, and Switch no. 2 or 3 has two links connecting with the two emulated racks.

The entire configuration of DCZeroconf contains four phases: 1) *RR configuration phase*, in which the CR configures the RRs; 2) s*erver configuration phase*, during which servers are configured; 3) c*ommunication channel construction (CCC) phase,* which builds the communication channel among switches and RRs for IP address allocation; and 4) *switch configuration phase* that configures the IP addresses for all the ports of each switch. We use $T_{RR}$, $T_{server}$, $T_{CCC}$, and $T_{switch}$ to denote the configuration time of the above four phases, respectively. Servers periodically send discover messages to RR before RR is configured, and switches will resend request messages until any reply is received in an interval. These two processes in our implementation have the same retry interval, which is denoted as $T_{interval}$. By performing experiments on the testbed, we are able to obtain the time of the four configuration phases, as well as the total time to configure all the switches/servers/VMs.

The experiments are divided into three groups. In each group, we repeat the experiments 10 times, and $T_{interval}$ is set to be 10 ms, 50 ms, or 100 ms in the three groups, respectively. We average the measured results in each group and illustrate them in **Table 1**. From the results, we make several observations. First, the value of $T_{interval}$ does not affect $T_{RR}$, $T_{switch}$, $T_{CCC}$, and $T_{server}$ significantly, which is in accord with the design of DCZeroconf.

Second, the time for the "CCC phase" and the "Switch configuration phase" is much larger than the other two phases, because the protocol messages in these two phases are propagated hops away, while the "RR configuration phase" and "Server configuration phase" only exchange protocol messages in a same broadcast domain. In addition,

**Table 1** Time (ms) consumed during the configuration on the testbed.

| | $T_{interval}$ | $T_{CCC}$ | $T_{RR}$ | $T_{switch}$ | $T_{server}$ | $T_{total}$ |
|---|---|---|---|---|---|---|
| Experiment #1 | 10 | 180.6 | 2.5 | 12.2 | 2.5 | 200.1 |
| Experiment #2 | 50 | 170.0 | 2.7 | 11.6 | 2.6 | 223.9 |
| Experiment #3 | 100 | 171.9 | 2.5 | 12.4 | 2.4 | 266.5 |

CCC phase" needs to record the forwarding paths and port into its local memory. This is why the dominant time in the entire configuration appears to be the construction of the communication channel.

Third, we analyze the range of the configuration times and check whether our testing falls into the range. The "RR configuration phase" and the "CCC phase" can be started at the same time. Once RRs being configured, the "Server configuration phase" starts to assign IP addresses to servers/VMs, while the "Switch configuration phase" starts to work when "RR configuration phase" and "CCC phase" are completed. Further, with the desperate retry interval $T_{interval}$ and the second observation, we derived the range of the entire configuration time as the following:

$$T_{CCC} + T_{switch} \leq T_{total} \leq T_{CCC} + T_{switch} + T_{interval}. \quad (1)$$

The measured results in Table 1 are obviously in this range. Moreover, as indicated in (1) and Table 1, increasing $T_{interval}$ increases the total configuration time. A very short $T_{interval}$ will increase the burden on servers, switches and the network, and we set it to be 100 ms in the remainder of the evaluation experiments.

### Simulations

We analyze the parameters required in the simulation and then determine the values by measuring using the testbed. By substituting the values of these parameters, we estimate the configuration time of DCZeroconf using larger-scale DCNs.

In [15], the authors evaluate DAC under several data center topologies. To compare with DAC, we also examine the configuration time of DCZeroconf on the same topologies. The experimental topologies include BCube [8], Fat-Tree [1], VL2 [3], and DCell [2]. Although in the original proposals of BCube, Fat-Tree, and DCell, the authors have also introduced specific addressing methods instead of IP addressing, the physical topologies of BCube, Fat-Tree, and DCell could also be used to employ IP-based DCNs.

We use simulation to estimate the time for DCZeroconf to complete the configuration of an IP-based DCNs on different topologies, and the detailed results are illustrated in **Table 2**. The results of DAC are cited from [15],

**Table 2** Configuration time (ms) in different topologies. The reader is referred to [15] for an explanation of the numbers in parentheses in the first column.

| *Topologies* | *No. of devices* | *DAC* | *DCZeroconf* |
|---|---|---|---|
| BCube(4,4) | 2,304 | 310 | 144.2 |
| BCube(8,4) | 53,248 | 2,000 | 166.9 |
| Fat-tree(20) | 2,500 | 262 | 192.9 |
| Fat-tree(100) | 262,500 | 6,223.6 | 247.3 |
| VL2(20,100) | 52,650 | 606.7 | 211.6 |
| VL2(100,100) | 252,650 | 2,032.3 | 227.6 |
| DCell(2,3) | 2,709 | 353 | 295.8 |
| DCell(6,3) | 3,807,349 | 45,578.1 | 3,276.3 |

which utilizes the specific addressing for the corresponding topology. The difference in addressing does differ with the protocol messages, but we believe that this effect is minor since the length of address does not vary very much. Furthermore, since DAC does not support server virtualization, the number of VMs is set to one in all the simulations using DCZeroconf.

Table 2 shows that the configuration time of DCZeroconf is shorter than the delay of DAC, and the gap is substantial for large topologies. The reason for the significant gaps in large topologies comes from the computation of a "mapping" between logical identification (ID) and physical ID for DAC. It is demonstrated in [15] that the time for mapping dominates the entire configuration time for large topologies. DCZeroconf does not consider such mapping during its configuration. However, as we mentioned in the introduction, it is just one of the useful characteristics of IP-based DC, which does not require us to embed the location information into the network ID.

Note that, when we set our simulation environment, all the parameters are measured from the test-bed with 100-Mb/s links. If the connections increase to 1 Gb/s as researchers in a DAC paper [3] used for their experiments, the configuration time of DCZeroconf could be shorter.

### Configuration time when topology changes

It is quite common that a data center will enlarge its scale by gradually adding new servers and switches. In this section, we evaluate the configurations time in such cases.

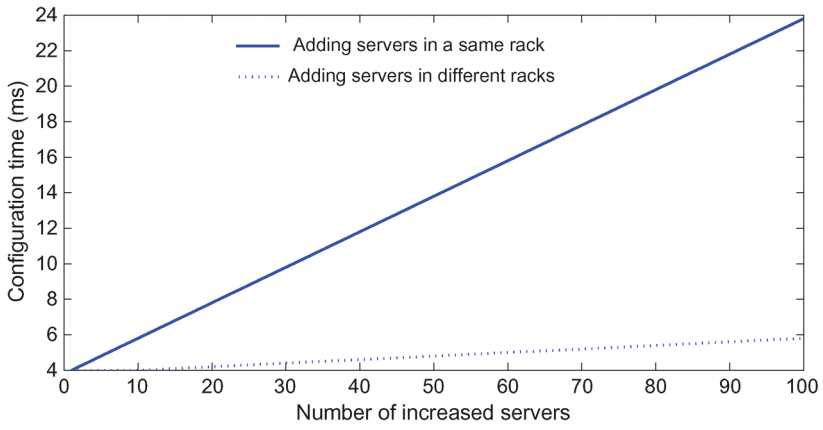


Figure 3

Configuration time when adding servers.

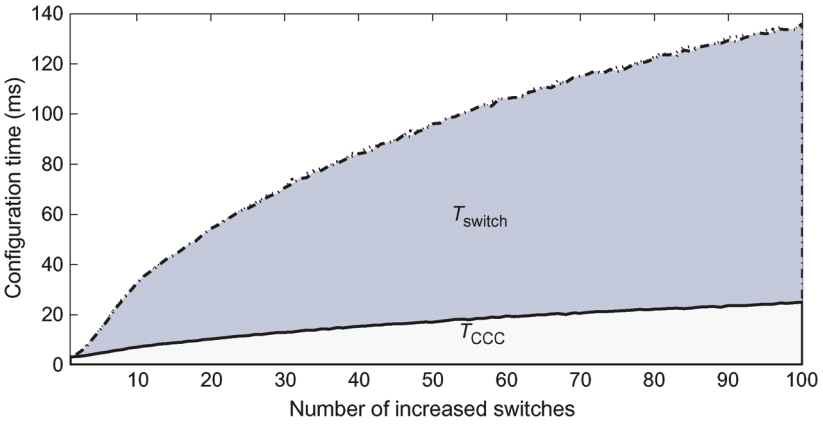


Figure 4

Configuration time when adding switches.

First, we study the case when adding servers. The required configuration time versus the number of the increased servers is depicted in **Figure 3**. This figure shows two extreme cases when adding servers. The solid line represents the case in which all the servers are added in a same rack. The dashed line indicates the results for the case when all the servers are evenly added to all the racks (in this experiment, the number of racks is 20). The increased configuration time is very small and even when we increase by 100 servers in one rack, the configuration time is less than 24 ms. The results demonstrate a linear bound of the scalability when adding servers: the configuration time linearly increased with the increase of the number of the servers. Since the configuration in different racks can be performed in parallel, the slope of the dash line is much smaller than the solid line. Please note that the configuration time of any other tested cases is within the range of these two curves in Figure 3.

Second, we investigate the performance of DCZeroconf when adding switches. The required configuration time versus the number of the newly added switches is depicted in **Figure 4**. In this simulation, we first generate a partially connected DCN and then add switches to randomly selected locations. It is worth noting that in this experiment, $T_{switch}$ and $T_{CCC}$ do not apply, and the configuration time consists of $T_{switch}$ and $T_{CCC}$. We run the experiment for 1,000 times, and the results in Figure 4 are the average value (the result is for a classical tree-based topology). The shaded area in Figure 4 indicates the difference between $T_{switch}$ and $T_{CCC}$. The total configuration time, as well as $T_{switch}/T_{CCC}$,

increases sublinearly with the number of the added switches, and this fact indicates that DCZeroconf is scalable with the enlargement of the data center. In addition, the entire configuration time including $T_{CCC}$ and $T_{switch}$ is less than 136 ms if we add 100 switches.

## Conclusion

In this paper, we have designed and implemented the DCZeroconf mechanism to automatically configure the IP addresses inside the data center. DCZeroconf successfully achieves the goals illustrated in the Introduction. First, it eliminates the possible IP address conflicts in different subnets without any assumption on the data center topology. Second, as the topology changes, DCZeroconf does not need to reconfigure the entire network; instead, RR only assigns the new devices with IP addresses. Third, the configuration of DCZeroconf is fast. It requires 200.1–266.5 ms to configure a real testbed with eight nodes and about 3.3 seconds in the simulation with more than 3 million devices. The experiments on the topology changes also indicate its scalability to large-scale DCNs (e.g., clouds of clouds). Finally, DCZeroconf is incrementally deployable in existing and future DCNs, as it only requires minimal modifications on the software stack of the current switches and servers.

## Acknowledgments

This work was partly supported by National Nature Science Foundation of China (NSFC) (60903182, 60921003, 60873250, 60736027), 973 plan (2007CB310702), 863 plan (2007AA01Z480), Tsinghua University Initiative Scientific Research Program, open project of State Key Laboratory of Networking and Switching Technology, and 111 International Collaboration Program of China.

## References

1. M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. SIGCOMM*, Seattle, WA, 2008, pp. 63–74.
2. C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A scalable and fault-tolerant network structure for data centers," in *Proc. SIGCOMM*, Seattle, WA, 2008, pp. 75–86.
3. 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," in *Proc. SIGCOMM*, Barcelona, Spain, 2009, pp. 51–62.
4. D. Roisman, "Data center top-of-rack switch redundancy models," in *Proc. NANOG 46*, Philadelphia, PA, 2009.
5. J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. Mogul, "SPAIN: COTS data-center Ethernet for multipathing over arbitrary topologies," in *Proc. NSDI*, San Jose, CA, 2010, pp. 265–280.
6. K. Elmeleegy and A. Cox, "EtherProxy: Scaling Ethernet by suppressing broadcast traffic," in *Proc. INFOCOM*, Rio de Janeiro, Brazil, 2009, pp. 1584–1592.
7. C. Kim, M. Caesar, and J. Rexford, "Floodless in SEATTLE: A scalable Ethernet architecture for large enterprises," in *Proc. SIGCOMM*, Seattle, WA, 2008, pp. 3–14.
8. C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. SIGCOMM*, Barcelona, Spain, 2009, pp. 63–74.
9. G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. E. Ng, M. Kozuch, and M. Ryan, "C-Through: Part-time optics in data centers," in *Proc. SIGCOMM*, New Delhi, India, 2010, pp. 327–338.
10. N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proc. SIGCOMM*, New Delhi, India, 2010, pp. 339–350.
11. C. Kim, "Scalable and efficient self-configuring networks," Ph.D. dissertation, Princeton Univ., Princeton, NJ, 2009.
12. R. Droms, "Dynamic host configuration protocol," RFC 2131, 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2131.txt
13. B. A. S. Cheshire and E. Guttman, "Dynamic configuration of IPv4 link-local addresses, Zeroconf," DFC 3927, 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3927.txt
14. R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer-2 data center network fabric," in *Proc. SIGCOMM*, Barcelona, Spain, 2009, pp. 39–50.
15. K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu, "Generic and automatic address configuration for data center networks," in *Proc. SIGCOMM*, New Delhi, India, 2010, pp. 39–50.
16. M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, San Jose, CA, 2010, p. 19.
17. XORPDownload code of Xorp. [Online]. Available: http://www.xorp.org/downloads.html

*Received January 31, 2011; accepted for publication March 19, 2011*

**Chengchen Hu** *MOE Key Lab for Intelligent Networks and Network Security, Department of Computer Science and Technology, School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'An, 710049 China; also guest researcher with SKLNST Lab, Beijing University of Posts and Telecommunications, Beijing 100876, China (huc@ieee.org).* Dr. Hu received his Ph.D. degree from the Department of Computer Science and Technology of Tsinghua University in 2008. He worked as an assistant research professor in Tsinghua University from June 2008 to December 2010 and is currently an Associate Professor in the Department of Computer Science and Technology of Xi'an Jiaotong University. His main research interests include computer networking systems and network measurement and monitoring.

**Mu Yang** *Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (yangmu266@gmail.com).* Mr. Yang is an undergraduate student in the Department of computer science and technology, Tsinghua University.

**Kai Zheng** *IBM Research Division, China Research Laboratory, Beijing 100074, China (zhengkai@cn.ibm.com).* Dr. Zheng received his M.S. and Ph.D. degrees both in computer science from Tsinghua University, Beijing, China, in 2003 and 2006, respectively. He is currently working with IBM Research China. His research interests include high-speed packet forwarding, pattern matching associated with network-security issues, and new data center network architecture design.

**Kai Chen** *Department of Electrical Engineering and Computer science, Northwestern University, Evanston, IL, 60201 USA (kchen@northestern.edu).* Mr. Chen is currently a Ph.D. student in the EECS Department at Northwestern University, Evanston, IL. Prior to this, he received his B.S. and M.S. degrees in computer science in

2004 and 2007, respectively, both from University of Science and Technology of China, Hefei, China. He is interested in finding simple yet elegant solutions to real networking and system problems.

**Xin Zhang** *Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213 USA (xzhang1@cs.cmu.edu).*
Mr. Zhang is currently a Ph.D. candidate in the computer science department at Carnegie Mellon University since 2006, working with Dr. Adrian Perrig and Dr. Hui Zhang. He received his B.S. degree in Department of Automation in 2006 from Tsinghua University. His academic interests include network security, applied cryptography, Internet routing, and clean-slate network architectural design. He is also passionate about web technologies and web application development.

**Bin Liu** *Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (liub@tsinghua.edu.cn).*
Professor Liu is currently a full Professor in the Department of Computer Science and Technology, Tsinghua University. His current research areas include high-performance switches/routers, network processors, high-speed security, and making Internet use more energy efficient. Professor Liu has received numerous awards from China including the Distinguished Young Scholar of China.

**Xiaohong Guan** *MOE Key Lab for Intelligent Networks and Network Security, School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'An, 710049 China (xhguan@mail.xjtu.edu.cn).* Dr. Guan received his B.S. and M.S. degrees in control engineering from Tsinghua University, Beijing, China, in 1982 and 1985, respectively, and his Ph.D. degree in electrical engineering from the University of Connecticut in 1993. He was a senior consulting engineer with PG&E (Pacific Gas and Electric Company) from 1993 to 1995. He visited the Division of Engineering and Applied Science, Harvard University, from January 1999 to February 2000. Since 1995, he has been with the Systems Engineering Institute, Xi'an Jiaotong University, and was appointed IEEE fellow in 2007, Cheung Kong Professor of Systems Engineering in 1999, and dean of the School of Electronic and Information Engineering in 2008. Since 2001, he has been the director of the Center for Intelligent and Networked Systems, Tsinghua University, and served as head of the Department of Automation, 2003–2008. He is an Editor of *IEEE Transactions on Power Systems* and an Associate Editor of *Automatica*. His research interests include optimization and security of networked systems, computer network security, and sensor networks.

# AUTHOR QUERY

## AUTHOR PLEASE ANSWER QUERY

Note that Ref. [17] has a web address that is not available. Please check.

## END OF AUTHOR QUERY