Efficient Anonymous Communication in SDN-Based Data Center Networks

Tingwei Zhu, Dan Feng, *Member, IEEE, ACM*, Fang Wang, Yu Hua, *Senior Member, IEEE, Member, ACM*, Qingyu Shi, Jiahao Liu, Yongli Cheng, and Yong Wan

***Abstract*— With the rapid growth of application migration, the anonymity in data center networks becomes important in breaking attack chains and guaranteeing user privacy. However, existing anonymity systems are designed for the Internet environment, which suffer from high computational and network resource consumption and deliver low performance, thus failing to be directly deployed in data centers. In order to address this problem, this paper proposes an efficient and easily deployed anonymity scheme for software defined networking-based data centers, called mimic channel (MIC). The main idea behind MIC is to conceal the communication participants by modifying the source/destination addresses, such as media access control (MAC) and Internet protocol (IP) address at switch nodes, so as to achieve anonymity. Compared with the traditional overlay-based approaches, our in-network scheme has shorter transmission paths and less intermediate operations, thus achieving higher performance with less overhead. We also propose a collision avoidance mechanism to ensure the correctness of routing, and three mechanisms to enhance the traffic-analysis resistance. To enhance the practicality, we further propose solutions to enable MIC co-existing with some MIC-incompatible systems, such as packet analysis systems, intrusion detection systems, and firewall systems. Our security analysis demonstrates that MIC ensures unlinkability and improves traffic-analysis resistance. Our experiments show that MIC has extremely low overhead compared with the base-line transmission control protocol (TCP) (or secure sockets layer (SSL)), e.g., less than 1% overhead in terms of throughput. Experiments on MIC-based distributed file system show the applicability and efficiency of MIC.**

Manuscript received November 23, 2016; revised July 13, 2017; accepted September 7, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Chen. Date of publication October 3, 2017; date of current version December 15, 2017. This work was supported in part by the NSFC under Grant 61772216, Grant 61772222, and Grant 61772212, in part by the National High Technology Research and Development Program (863 Program) of China under Grant 2013AA013203, and in part by the Shenzhen Science and Technology Plan Project under Grant JCYJ20170307172248636. A conference paper [1] containing preliminary results of this paper appeared in ICPP 2016. *(Corresponding author: Dan Feng.)*

T. Zhu, D. Feng, Y. Hua, Q. Shi, and J. Liu are with the Key Laboratory of Information Storage System (School of Computer Science and Technology, Huazhong University of Science and Technology), Ministry of Education of China, Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China (e-mail: twzh@hust.edu.cn; dfeng@hust.edu.cn; csyhua@hust.edu.cn; qingyushi@hust.edu.cn; liujiahao@hust.edu.cn).

F. Wang is with the Key Laboratory of Information Storage System (School of Computer Science and Technology, Huazhong University of Science and Technology), Ministry of Education of China, Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China, and also with the Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen 518000, China (e-mail: wangfang@hust.edu.cn).

Y. Cheng is with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China (e-mail: chengyongli@hust.edu.cn).

Y. Wan is with the Computer Engineering College, Jingchu University of Technology, Jingmen 448000, China (e-mail: wanabc@hust.edu.cn).

Digital Object Identifier 10.1109/TNET.2017.2751616

***Index Terms*— Anonymity, data center, software-defined networking, in-network anonymous communication, distributed file system.**

I. INTRODUCTION

WITH the expansion of the scale, data centers are facing a growing number of security threats from internal components (such as compromised servers, switches). According to IBM 2015 Cyber Security Intelligence Index [2], 55% of all attacks and incidents monitored by IBM in 2014 were carried out by insiders. Moreover, the outside attackers can always hack into the internal network of their targets for data breach. For example, in the data breach of Target in 2013, the attackers gain access to the Target network through stolen HVAC vendor credentials [3], and then steal 40 million credit cards. As we can see, the internal network is untrustful, and more attentions should be placed on the security inside data centers.

When travelling through the untrustful network, it is important to protect the communication participants' identities and traffic patterns to conceal the activities of users. Even if the messages are encrypted, an adversary can still launch traffic-analysis attacks by examining the unencrypted information, like IP addresses, port, traffic rate or size. For example, an attacker can identify the originator and terminater of a flow by checking the source and destination addresses, and then reveal (or guess with a high probability) the ongoing operations between them by analyzing the traffic patterns. Further, the attacker can even know which user and application the communication participants belong to, as well as the scale or load of the application, through iterated traffic-analysis attacks. If the attacker aims to crash the target application or system, he can locate some key nodes of the system (like the Metadata Servers in distributed file systems) easily, and then launch active attacks, such as DoS/DDoS and Worms. If he aims for data breach, this can help locate the target servers.

A lot of anonymity systems have been proposed to conceal user identity and resist traffic-analysis attacks. Such systems attempt to facilitate anonymous communication by building mix- or relay-based overlay network, such as Mixminion [4], Crowds [5], Tor [6], Dissent [7], and etc. However, these systems are designed for the Internet environment, suffering from high overhead, and cannot meet the requirements of high bandwidth and low latency in the data center environment. For example, the most popular anonymity system Tor uses layer-encrypted packets and travels through multiple indirect

relays to conceal the endpoint's IP address. This approach will result in significant performance loss, since long end-to-end path length and cryptographic operations will cause high latency. Meanwhile, the indirectly traveling will incur redundant network traffic, consuming considerable network resources and reducing the total capacity of the data center network.

Most of the applications in data centers are performance sensitive, which require high bandwidth (e.g. video encoding systems) and low latency (e.g. web search systems) in transmission. Moreover, the computational and network resources are limited, and the overlay-based approaches are too expensive and will significantly reduce the capacity of the data center. Measurements show that Tor reaches 62 times higher in latency and 80% lower in throughput compared to the baseline TCP (Fig. 9 and Fig. 10(a)). Therefore, it is a challenge to provide an efficient and low overhead anonymity system which is suitable for the data center environment.

The widely deployed Software Defined Networking (SDN) [8] in data centers brings new idea for achieving anonymous communication. The SDN architecture makes the packets forwarding more flexible. The controller can install routing rules into switches in advance and the switches modify the packet header to hide the real participants of a flow, achieving anonymous communication.

To meet the requirements of anonymity within data centers, we present Mimic Channel (MIC), an efficient in-network anonymity system designed for data center environment, which can significantly reduce computation and network resource consumption with non-overlay architecture. The basic idea behind MIC is to conceal the sender and receiver of a flow by changing the addresses (such as MAC, IP and port) on multiple switches (not hosts). As a result, a flow can mimic flows of other participants. A flow in MIC is called an m-flow. The switch node which changes the packet addresses is called Mimic Node (MN). The fake addresses changed by an MN are called m-addresses. An MN can be regarded as a lightweight mix (or relay) node in traditional anonymity systems, but is built on a switch node in the network. MIC achieves in-network anonymous communication, and hence has much shorter forwarding paths and fewer intermediate operations than traditional overlay-based schemes. Therefore, MIC is more efficient and suitable for data center environments.

However, there are two technical challenges in the MIC design. **First**, in order to achieve better anonymity, the m-addresses should be real addresses in the same network. Therefore, we need to handle the potential conflicts between two different m-flows or between an m-flow and a common flow (non-mimic flow). To avoid these routing collisions, we propose a Collision Avoidance Mechanism and design an M-Address Generation Algorithm (MAGA) to map the m-addresses of different m-flows to disjoint address spaces. **Second**, in order to increase the usability and deployability, the MIC design should not incur any modification on commodity SDN switches, as well as achieving a certain level of traffic-analysis resistance. We employ Multiple M-flows, Dynamic M-Flows and Partial Multicast mechanisms to improve the traffic-analysis resistance of MIC.

The paper makes the following contributions.

- We reveal the potential security threats in non-anonymous data centers, and emphasize the importance of anonymous communication inside data centers.
- We propose an efficient anonymity scheme for SDN-based data centers, called MIC, which hides the communication participants by changing the packet header at multiple switch nodes along the transmission path. To address the challenge of routing collision, we design a Collision Avoidance Mechanism (Sec IV-B3). We also propose three mechanisms to enhance the traffic-analysis resistance for MIC (Sec IV-C) and present solutions to enable MIC co-existing with other systems like intrusion detection systems (Sec VI-A).
- We implement and evaluate MIC. Our security analysis and evaluations demonstrate that MIC can achieve session unlinkability and improve traffic-analysis resistance at low overhead. We also propose and implement MIC-based distributed file system (CapFS) to verify the applicability of MIC. The evaluation on MIC-based CapFS demonstrates that MIC can be easily deployed in distributed systems inside SDN-based data centers with negligible overhead.

The rest of this paper is organized as follow. Section II presents the background and motivation of this paper. Section III describes the system model, threat model, goals and assumptions. Sections IV describes the MIC design. In Section V, we discuss the security of MIC. We discuss the deployment and applicability issues of MIC in Section VI. Section VII describes the implementation details and our experimental evaluation of MIC. Section VIII describes the related work. Finally, we conclude our paper in Section IX.

II. Background and Motivation

Anonymity in Data Centers: The anonymity in data center is very important in breaking attack chains and guaranteeing user privacy. We take the data breach of Target in 2013 as an example to show how anonymity in the internal network helps to alleviate the attack. The attack is performed by the following steps [3].

Step 1: Install Malware that Steals Credentials
Step 2: Connect Using Stolen Credentials
Step 3: Exploit a Web Application Vulnerability
Step 4: Search Relevant Targets for Propagation
Step 5: Steal Access Token from Domain Admins
Step 6: Create a New Domain Admin Account Using the Stolen Token
Step 7: Propagate to Relevant Computers Using the New Admin Credentials
Step 7.1: Bypassing Firewall and Other Network-based Security Solutions
Step 7.2: Running Remote Processes on Various Machines
Step 8: Steal 70M PII. Do Not Find Credit Cards
Step 9: Install Malware. Steal 40M Credit Cards

Step 10: Send Stolen Data via Network Share
Step 11: Send Stolen Data via FTP

In the **Step 4**, the attackers locate the relevant target servers by querying Active Directory[1] and obtain the respective IP addresses by querying the DNS server. In the **Step 7.1**, the attackers bypass the firewall and other network-based security solutions by propagating through a series of servers using the "Angry IP Scanner"[2] and "Port Forwarding utility",[3] which are based on IP addresses. In this attack, a precursory step for enabling the attackers accessing their target servers which hold credit cards is to obtain the IP addresses of the relevant servers. If anonymity is enabled, the attackers cannot obtain the IP addresses of the relevant servers directly, and it is hard for the attackers to bypass the firewall and other network-based security solutions as they has no idea of their targets' identifies. Therefore, anonymity in data centers can help to alleviate the attack dramatically.

The data center faces internal security threats. An adversary can collect or observe a large number of traffic information at any point of the network easily. For example, a hacker can take over a switch by telnet attack, thereby observing and analyzing the traffic patterns to launch traffic-analysis attacks. In some server-centric network topologies, such as BCube [9], a hacker can compromise a server, and analyze the traffic passing through it. In virtualized cloud data centers, a malicious user on a guest VM can attack or compromise the host hypervisor by "guest VM escape" [10], and then can easily observe the traffic of other VMs on the same host. Much information in the packet header is useful to the adversaries, for instance, the 'ports' will typically reveal the service type (the port 80 represents Web server). In addition, there are many known shortages in existing commercial cloud. For example, Ristenpart *et al.* [11] point out that the internal IP addresses are statically assigned to physical machines, and one can use the internal IP address to infer the instance type and availability zone of a target service in EC2 [12]. Therefore, it is important to protect the identity of a host inside data centers.

Unfortunately, traditional anonymity systems are designed for the Internet environment, which are not suitable for the data center environment. **First**, the applications in data centers have higher performance requirements than those in the Internet. All existing anonymity approaches are overlay-based, and hide the correspondence between input and output messages through hop-by-hop encryption. Therefore, they suffer from high performance overhead due to long transmission path and cryptographic operations. **Second**, the computational and networking resources are expensive in data centers. Redundant traffic in overlay architecture and multiple cryptographic operations will consume a lot of resources. Therefore, it is a challenge to achieve anonymous communication at low overhead. Fortunately, the data center is more controllable than the Internet, and it naturally faces much less security threats than the Internet, while tolerating looser threat model.

[1]https://en.wikipedia.org/wiki/Active_Directory
[2]http://angryip.org
[3]https://portforward.com

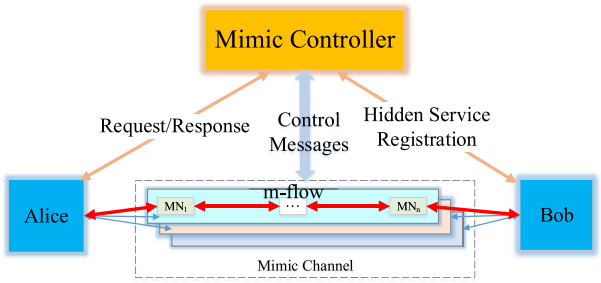


Fig. 1. The system model of MIC. The main idea behind MIC is to conceal the communication participants by modifying the source/destination addresses (such as MAC, IP and port) at multiple switch nodes (MNs).

This gives us a new design space for a lightweight anonymity system.

Software-Defined Networks: The Software-Defined Networking (SDN) architecture separates data plane from control plane, simplifying the network configuration, opening up the networking, and making the networking programmable. SDN dramatically simplifies the routing in data center networks (DCNs). A lot of researches on SDN-based DCNs, such as MCTCP [13], Hedera [14] and zUpdate [15], have demonstrated the feasibility and trend of integrating SDN into the data center. In this paper, we focus on the anonymity scheme in SDN-based data centers.

III. Problem Definition

In this paper, we study the anonymity system for data centers to achieve anonymous communication and enhance traffic-analysis resistance. More specifically, an anonymity system should conceal the end-hosts' identities and the real traffic patterns. Taking into account the features of data centers, we study an anonymity system that can provide a practical level of anonymity at minimal performance overhead.

A. System Model

The scheme proposed in this paper is designed for SDN-based data centers, and all the switches in this paper are SDN-enabled, which can modify the packet header. Mimic Channel (MIC) is a typical C/S model design, which consists of clients, MNs (Mimic Nodes) and an MC (Mimic Controller). As shown in Fig. 1, Alice is an initiator client who wants to communicate with Bob (the responder client) anonymously. She creates a transport channel between Bob and communicates with each other using MIC. A mimic channel consists of one or several end-to-end flows, called m-flows. Each m-flow travels through several MNs, which are specified by the MC. The MC, located in the SDN controller, calculates and manages the routing of each m-flow.

- The **clients**, including the initiators and the responders, can be any end-hosts in the network. An initiator establishes a mimic channel with a responder proactively before communication starts. Once a mimic channel is established, the communication pairs can exchange messages without revealing each other's identity.

- An **MN** is a lightweight mix or relay in the traditional anonymous systems, which can only modify the header of packets instead of operations like encryption/decryption, re-ordering, delaying and batching, and etc. The commercial switches generally have no advance intelligence, and our design goal is to minimize the overhead. Any switches in the network are potential MNs.
- The **MC** is responsible for calculating and managing the routing of each m-flow. It determines the MNs in each m-flow, and generates m-addresses for each MN. With the global view of the network and each m-flow, the MC is the core of MIC design.

At a high level, MIC has two phases, the channel establishment (Section IV-A1) and the data forwarding (Section IV-A2).

B. Threat Model

The goal of adversaries is to break the unlinkability of communication pairs, seeking to infer which pairs of clients are communicating. We assume an adversary who can compromise a part of switches, the initiator client or the responder client; and who can observe some fraction of network traffic.

Compromising the Switches: An adversary may compromise one or a plurality of switches (but not all), which may be MNs or common switches, seeking to observe and correlate the traffic via the switches.

Compromising the Client: An adversary may compromise the initiator (or the responder), seeking to obtain the identity of the responder (or the initiator). For example, a hacker compromises a client in a distributed storage system, and attempts to obtain information of other nodes (like the metadata servers or storage servers), to learn which points in the network to attack next.

Observing the Traffic: An adversary may observe and analyze the traffic at some points in the network. For example, the switches in data centers generally have port mirroring function, which is used for Intrusion Detection System (IDS). The adversary may use the port mirroring for traffic observing, or have compromised the existing IDS.

Like most of the prior practical anonymity schemes, MIC does not protect against a global adversary who can snoop on all paths or switches. A global adversary is unlikely in practice. Specifically, it is not easy to compromise a single switch, let alone all the switches in data centers. Moreover, an IDS generally monitors the traffic from only a few ports to reduce the overhead, and the port mirroring on most switches are disabled by default. Therefore, it is hard to observe the global traffic from all switches.

C. Goals

The main goal of MIC is to frustrate attackers from linking communication partners, achieving session unlinkability. MIC also aims to enhance resistance against size- or rate-based traffic-analysis. Moreover, MIC has the following design goals:

High Performance: Most of the applications in data centers are performance sensitive, requiring high bandwidth and low latency. For example, the web services are delay sensitive applications and the file services are bandwidth hungry applications.

Deployability: MIC design should require no kernel or switch modifications, and can deploy in common SDN-based data centers.

D. Assumptions

We assume the SDN controller (i.e., the MC), is secure, and all the communications between the SDN controller and the switches are secure. We believe these assumptions are reasonable. The SDN controller is the core of the network. Once the controller is compromised, the entire network will crash.

IV. Design of MIC

A. Overview

Similar to most of the previous anonymity systems, MIC has two phases, including the channel establishment and data forwarding.

1) Channel Establishment: In channel establishment, one or a set of bi-directional routing paths will be generated for each channel. Each m-flow has independent MNs and m-addresses. Specifically, when establishing a mimic channel, the initiator creates a *request* packet to the MC, and then the MC generates the corresponding routing before returning an acknowledgement to the initiator. The *request* packet contains the encrypted m-flow number, MN number and server address (or nickname). The MC calculates the forwarding path for each m-flow and chooses the specified number of switches as the MNs in each path. After all paths are generated, the MC sends an acknowledgement, which contains a set of entry addresses, to the initiator. The entry address is the first m-address in an m-flow from the initiator's view, which hides the address of the responder. In practice, the communication between the initiator and the MC can be realized by using the "packet in" mechanism, and a few UDP packets is enough.

2) Data Forwarding: After the mimic channel is established, the initiator or the responder can send messages anonymously through the channel. All the MNs will mimic the header of packets traveling through the path to hide the participants' identities. After the communication is completed, the sender will send a notification to the MC to facilitate channel management at the MC.

We take a simple example to illustrate how MIC works. Suppose two clients Alice (with IP address 10.0.0.1) and Bob (with IP address 10.0.0.8) are connected via three switches (S1, S2 and S3), as shown in Fig. 2. For the purpose of anonymous communication, Alice dose not send messages to Bob directly, but sends a *request* to the MC for constructing an anonymous path to Alice first. After receiving the *request* from Alice, the MC calculates the forwarding path to Bob. The switches along the path modify the packet header to conceal the identities of Alice and Bob. Specifically, suppose the packet header is denoted as a two-tuple $\langle src\_ip, dst\_ip \rangle$. The MC notifies Alice that he should send packets to the destination with address 10.0.0.2, i.e., the packet header of P1 is

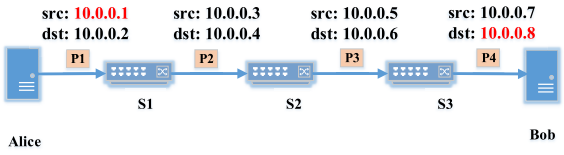


Fig. 2. An example of MIC. The intermediate switch nodes are not aware of the real 'src' 10.0.0.1 and 'dst' 10.0.0.8.

$\langle 10.0.0.1, 10.0.0.2\rangle$. The switch S1 modifies the header of P1, and forwards it to the next hop, i.e., the packet header of P2 is $\langle 10.0.0.3, 10.0.0.4\rangle$. Similarly, switches S2 and S3 modify the packet header to $\langle 10.0.0.5, 10.0.0.6\rangle$ and $\langle 10.0.0.7, 10.0.0.8\rangle$, respectively. It is worth noting that the last switch should modify the destination address back to the correct one, so that the receiver can handle the packets correctly without protocol stack or kernel modification.

B. Mimic Controller

The MC, located in the SDN controller, is the core of MIC. All the routings are calculated by the MC, and then are installed to the corresponding switches. The MC decides the forwarding path, the MNs and m-addresses for each m-flow, and has the global view of each channel. Specifically, the MC manages all the channel states, calculates and manages the routing, and handles the routing conflicts of each m-flow, ensuring the correctness of the network.

1) Channel Management: The MC needs to maintain the status of all mimic channels. When a mimic channel is constructing or the communication is finished, the initiator sends a *request* to inform the MC. Therefore, the MC can have the states of all m-flows. Thus, it can be seen that the MC needs to handle a large number of establishing and shutdown *requests* in massive short communication scenes. In order to reduce the overhead on the MC, we should reuse the mimic channel among the communications between the same participants. Therefore, in these scenarios, the sender does not send shutdown *request* to the MC immediately when the communication is finished. Instead, a dedicated module in the initiator will send notification to the MC periodically.

2) Routing Calculation: MIC achieves anonymous communication by elaborately-designed routing which changes the packet header at several switches while finally leading to the right destination. The MC obtains the global view of the network and calculates all-pairs equal-cost shortest paths after initialization. After receiving the *request* packet from an initiator, the MC generates the specified number of routing paths for m-flows. First of all, the MC gets the initiator and the responder's addresses, the m-flow number F and the MN number N from the *request* packet. If the responder is a hidden receiver, the MC should find the address of the receiver from a hidden service map. For each m-flow, the MC randomly selects a pre-calculated shortest path between the initiator and responder. If the path length is less than N, a new forwarding path with length larger than N will be calculated. After determining the routing path, the MC chooses N switches along the routing path as MNs. Then the MC determines the m-addresses on each MN. Finally, all the routings are installed to the corresponding switches. The MN number indicates the privacy level of an m-flow, and the more MNs will cause more overheads. We allow users to trade the privacy for performance. We will discuss how to generate m-addresses in Section IV-B3.

3) Collision Avoidance: All the m-addresses should be in the same network namespace (or subnet) to enhance the anonymity of the m-flow. Therefore, routing collision between two m-flows, or an m-flow and a common flow could happen, which will lead to errors.

Collision Examples: Routing conflicts could happen when two or more flows are forwarded through the same port at a switch. The following examples show three routing conflict scenes. To simplify the description, we assume the two-tuple $< src\_ip, dst\_ip >$ identifies a flow on each switch. (1) The packet addresses of two flows f_1, f_2, are changed to the same one on the same switch, as shown in Fig. 3(a). (2) The packet addresses of a flow f_1 are changed to the same as another flow f_2 on the same switch, as shown in Fig. 3(b). (3) The packet addresses of two flows f_1, f_2 are the same before they reach the same switch, but the switch does not change the addresses of both flows, as shown in Fig. 3(c). The root cause of routing conflicts is that, the m-flow will use variable addresses during communication. Therefore, an m-flow may occupy the addresses of a common flow, or two m-flows may use the same addresses simultaneously.

Collision Avoidance Mechanism: To avoid routing conflicts, we design a collision avoidance mechanism. The basic idea is to ensure each flow has a unique match entry on any switch.

First, to avoid collisions between a common flow and an m-flow, we use MPLS [16] label to distinguish them. Here we just use MPLS field for tagging, so that we can distinguish the flows carrying different three-tuple $\langle src\_ip, dst\_ip, mpls\rangle$. We divide the MPLS label into two disjoint categories, one used to mark the common flows (CF), and the other used to mark the m-flows (MF). Only the MC knows which MPLS labels are in CF and which are in MF. We will describe how to divide the MPLS label sets later.

Second, in order to avoid conflicts between different m-flows, we design an M-Address Generation Algorithm (MAGA). The main idea behind MAGA is to reasonably divide the address space into disjoint classes, and map the m-addresses of each m-flow (or mimic channel) into different address spaces. Therefore, for each m-flow, it can randomly select an m-address from its address space each time, avoiding collision with any other m-flows. For simplicity in description, we suppose each mimic channel contains only one m-flow. Specifically, for an m-flow, the real address is $\langle src\_ip, dst\_ip\rangle$, and an MN should convert the address into m-address $\langle m\_src\_ip, m\_dst\_ip\rangle$. In order to reduce the possibility of m-address collision among different m-flows, we add MPLS label for tagging. That is, we use the three-tuple $\langle m\_src\_ip, m\_dst\_ip, m\_mpls\rangle$ to uniquely identify an m-flow on each switch.

We use a hash function $f(x, y, z)$ to map the m-addresses of each m-flow to different address spaces. In that case, given

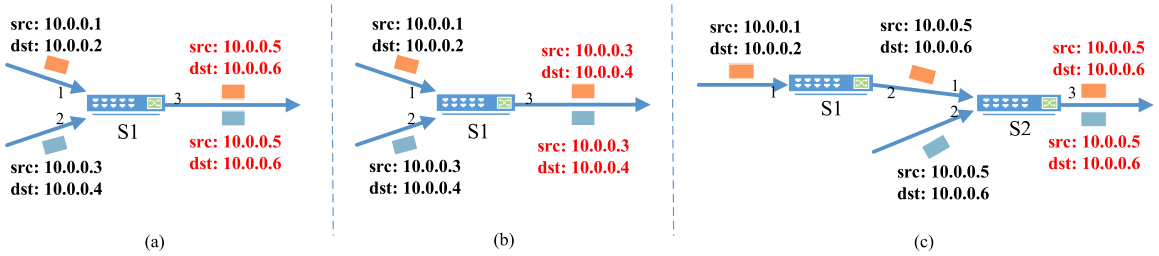


Fig. 3. Examples of routing collision. (a) Two different addresses are changed to the same one. (b) One address is changed to the same one with another. (c) Two flows with the same address are forwarded to the same port.

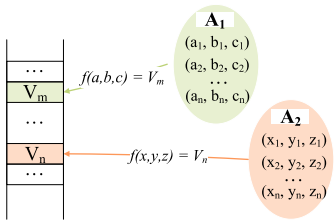


Fig. 4. Hash function demonstration.

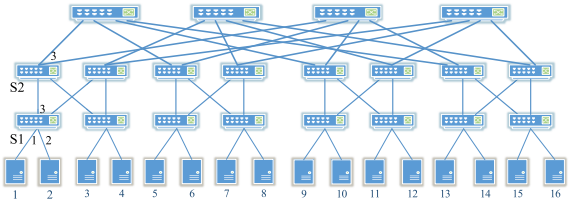


Fig. 5. Fat-tree Topology.

two different values V_m, V_n, we can get two disjoint three-tuple sets A_1 and A_2, which satisfy that for any $(a, b, c) \in A_1$, $(x, y, z) \in A_2$, satisfy $f(a, b, c) = V_m$ and $f(x, y, z) = V_n$, but $(a, b, c) \neq (x, y, z)$, as demonstrated in Fig. 4. Therefore, if we give each m-flow a unique ID, and let any m-address (x, y, z) of an m-flow satisfies $f(x, y, z) = ID$, the routing collision between different m-flows can be avoided.

The main point is to ensure that each m-flow has a unique ID. A simple method is to monotonically increase the ID when a new m-flow arrives, and recover the expired ID when an m-flow is closed. Performed naively, a global hash function for all MNs is enough. However, in this scheme, all m-addresses (on all MNs) are constrained by a single hash function, which will result in poor security. For example, an adversary can compromise an MN, and try to find out the hash function by analyzing the m-addresses on the MN. Once an adversary knows the hash function, he can associate the packets within the same m-address space to break anonymity.

To solve the above-mentioned issues and improve anonymity of MIC, we set an independent hash function for each MN rather than a uniform hash function for all. Therefore, the adversary cannot obtain all the hash functions on all MNs easily, so making it hard to associate with the m-flows. However, as each MN has an independent hash function, we can only ensure no conflicts among m-addresses within the same MN, but not that between different MNs. Fig. 3(c) shows an example of m-addresses conflict between two different MNs (if f_2 is an m-flow).

To avoid this kind of conflicts, we use the MPLS label to ensure that the m-addresses between different MNs never conflict. Again, we divide the MPLS into multiple disjoint sets, and map the MPLS sets to each MN. Therefore, the m-addresses on different MNs have different MPLS labels, which will avoid m-addresses conflicts among different MNs. To ensure anonymity, for any given MPLS label, only the MC knows which MN the label corresponds to. Similarly, we use a hash function $g(x)$ to classify the MPLS sets, and map the sets to each MN. Specifically, each MN has a unique S_ID. For an MN M whose S_ID is S, if an MPLS label m satisfies $g(m) = S$, m is in the set for M.

Thus, the key point in MAGA is to build two hash functions $f(x, y, z)$ and $g(x)$. For $f(x, y, z)$, our goal is that for a given function value V, we can get a three-tuple (a, b, c) which satisfies $f(a, b, c) = V$. Therefore, function $f(x, y, z)$ must be reversible on at least one variable. In this case, we can first determine two variables randomly, and then determine the rest variable using the inverse function, and finally get the three-tuple m-address. In order to ensure all the variables of this function are integers, we use XOR or shift operation to build the function. For example, a simple $f(x, y, z)$ can be constructed as follow.

$$\begin{aligned} f(x,y,z) = {} & [(x \oplus A_0) \gg A_1] \oplus [(x \oplus A_2) \ll A_3] \\ & \oplus [(y \oplus B_0) \gg B_1] \oplus [(y \oplus B_2) \ll B_3] \\ & \oplus [(z \oplus C_0) \gg C_1] \end{aligned} \tag{1}$$

Then the inverse function for variable z is :

$$\begin{aligned} f_z^{-1}(v,x,y) = {} & v \oplus [(x \oplus A_0) \gg A_1] \oplus [(x \oplus A_2) \ll A_3] \\ & \oplus [(y \oplus B_0) \gg B_1] \oplus [(y \oplus B_2) \ll B_3] \\ & \ll C_1 \oplus C_0 \end{aligned} \tag{2}$$

A_0, A_1, A_2, A_2, B_0, B_1, B_2, B_3, C_0, C_1 are parameters, which can be different for different MN to build different hash functions.

To avoid an adversary distinguish the m-flows and common flows by observing the source/destination IP addresses, the m_src_ip and m_dst_ip should subject to different restrictions on different MNs. For example, for a Fat-tree topology as shown in Fig. 5, the source IP of packets forward out to port 3 should be restricted to $\{1, 2\}$ and $\{1, 2, 3, 4\}$, respectively at switch $S1$ and $S2$. Meanwhile, as previously described, the MPLS label should be restricted to different sets on different MNs to avoid m-addresses conflicts among different MNs. As a result, all the three

elements in $\langle m\_src\_ip, m\_dst\_ip, m\_mpls\rangle$ cannot be arbitrarily selected. To get a three-tuple which satisfies all the restrictions quickly, we divide the MPLS to two parts $mpls_1$ and $mpls_2$, of which the $mpls_1$ is subject to the restriction of distinguishing different MNs, but the $mpls_2$ is not. Therefore, getting a satisfied three-tuple $\langle m\_src\_ip, m\_dst\_ip, m\_mpls\rangle$ is equivalent to getting a four-tuple $\langle m\_src\_ip, m\_dst\_ip, mpls_1, mpls_2\rangle$. We construct a four variables hash function $F(\alpha, \beta, \gamma, \delta)$ and the inverse function for variable δ, $F_\delta^{-1}(v, \alpha, \beta, \gamma)$ similar to $f(x, y, z)$ and $f_z^{-1}(v, x, y)$, respectively. Finally, we first randomly select a qualifying m_src_ip, m_dst_ip, $mpls_1$, and then calculate out the $mpls_2$ using the inverse function $F_\delta^{-1}(v, \alpha, \beta, \gamma)$.

For $g(x)$, since there is only one variable, it is difficult to construct a function which meets the requirement. Hence, we divide the variable x into multiple independent variables in bits. For example, a simple solution is to divide the variable x into high bytes x_1 and low bytes x_2. Suppose the variable x has 32bits, x_1 is the high 16bits and x_2 is the low 16bits. Therefore, the function $g(x)$ is equivalent to $h(x_1, x_2)$. We can construct $h(x_1, x_2)$ and $h_{x_2}^{-1}(v, x_1)$ similar to $f(x, y, z)$ and $f_z^{-1}(v, x, y)$, respectively.

For an MN, if its S_ID is S, any MPLS label whose $mpls_1$ on it should satisfy $g(mpls_1) = S$. Given the hash value S, we first randomly select the high 16bits x_1, and then calculate out the corresponding low 16bits x_2 using the inverse function $h_{x_2}^{-1}(v, x_1)$, finally the $mpls_1 = x_1 \ll 16 + x_2$. It is worth noting that, in order to enhance security, we can make it harder for the adversary to obtain the hash function by dividing the variable x in the more random way, or dividing x into more sub-variables. Similarly, for the common flows, we assign a unique function value C to it, and let any MPLS whose $mpls_1$ satisfy $g(mpls_1) = C$ tags the common flows. The pseudocode of M-Address Generation Algorithm is shown in Algorithm 1.

Algorithm 1 M-Address Generation Algorithm

1: // Randomly select the m_src_addr and m_dst_addr;
2: $M.m\_src\_addr = RandomSelect()$;
3: $M.m\_dst\_addr = RandomSelect()$;
4: // Determine the $mpls_1$: (1). Randomly select the x_1; (2). Calculate out the x_2 using h^{-1};
5: $x_1 = RandomSelect()$;
6: $x_2 = h^{-1}(S, x_1)$;
7: $mpls_1 = combine(x_1, x_2)$;
8: // Calculate out the $mpls_2$ using F^{-1}
9: $mpls_2 = F^{-1}(V, M.m\_src\_addr, M.m\_dst\_addr, mpls_1)$;
10: $M.m\_mpls = combine(mpls_1, mpls_2)$;
11: **return** M;

C. Traffic-Analysis Resistance

An adversary may observe and correlate the traffic at some places (switches, links or servers) in the network, seeking to find out the communication participants or what operations are processing. To enhance traffic-analysis resistance, we employ three mechanisms, including the multiple m-flows, dynamic m-flows and partial multicast mechanisms.

Multiple M-Flows Mechanism: MIC aims to achieve anonymous communication with good performance and deployability which can be deployed and used in the practical data centers. The commercial SDN switches can only process the rules defined by southbound interfaces, like OpenFlow, but has no user-defined interfaces. Therefore, we do not delay, encrypt/decrypt or batch traffic on MNs, but just modify the packet header. To defend the size-based traffic analysis, we choose to mimic the traffic size at the source, which motivates us to employ multiple m-flows mechanism. Specifically, each mimic channel may consist of several m-flows, and each m-flow has independent routing path, MNs and m-addresses. The initiator divides the user data into slices, and each m-flow carries different amount of slices. As the traffic is divided into multiple pieces, an adversary cannot obtain the real size of the traffic unless he knows the m-flow number and has correlated all the m-flows.

Dynamic M-Flows Mechanism: To prevent an adversary from correlating the communication participants and obtaining the traffic patterns by long term traffic observing and iterated switch compromise, we should periodically update the m-flows' forwarding rules (mostly for the long term communications) during lifetime. Specifically, for an m-flow, the routing path, MNs and corresponding m-addresses will be changed. Performed naively, all the m-flows in a channel will be updated periodically in a synchronized way. However, an adversary may correlate the m-flows in the same channel by observing the flow changing frequency. For example, if an adversary observes that there are three flows disappear accompanied by three new flows emerging, then he can correlate the three flows to the same channel with high probability. Therefore, we update each m-flow in independent way, with randomized timing and variable frequency, to defend above-mentioned attacks.

Partial Multicast Mechanism: An adversary may observe all the ingress and egress traffic on an MN, and correlates the m-flow at the MN. Since the MN processes no cryptographic operations on packets, the packets in the same m-flow have the same payloads at each hop. An adversary can correlate with them by checking the payloads of each packets. MIC cannot defeat such end-to-end correlation, but uses partial multicast mechanism to maximally decrease the success rate of this correlation. More specifically, at an MN, we will replicate the input packet to multiple packets with different m-addresses, and send the packets out from different ports simultaneously. But only one of the output packets will finally reach the receiver, the others will be dropped in the next hop, as show in Fig. 6. This may be useful at the edge MNs.

D. Unlinkability

MIC achieves unlinkability by changing the packet header at multiple switches.

Sender Anonymity: MIC cannot hide the sender address if an adversary observes traffic at one point between the sender and the first MN. However, the goal in this paper is not

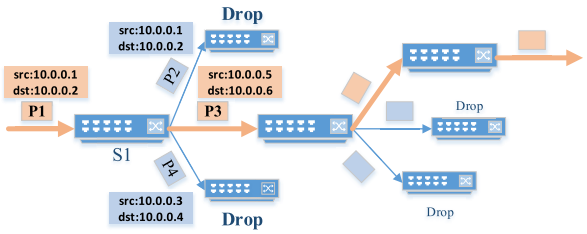


Fig. 6. Partial multicast demonstration. The MN S1 will forward out three packets P2, P3 and P4 when receiving P1, but P2 and P4 will be drop at next hop.

to provide strong anonymity at any cost, but to break the correlation between the sender and the receiver. In fact, as any switch in the network can be an MN, an adversary cannot tell whether the packet header has been modified by an MN, unless he compromises the first switch which direct links to the sender.

Receiver Anonymity: Receiver anonymity can be easily realized in MIC. Unlike the previous anonymity systems, MIC needs no additional rendezvous. The MC, which has the global view of each mimic channel can achieve the similar functionality as rendezvous or hidden service in traditional anonymity approaches. The hidden receiver first sends its contact information to the MC for anonymous service registration. The MC then adds the receiver to a hidden service map. The initiator client obtains the service name (or nickname) of the hidden receiver out of band and constructs a mimic channel using the service name. As the MC knows about the location and identify of the receiver, the channel can be constructed as normal.

V. Security Analysis

MIC is designed to achieve anonymous communication in SDN-based data centers. We discuss a variety of attacks within our threat model and how MIC withstands them.

Compromising Switches: An adversary may compromise one or several switches, which can be the common switches (non-MNs) or the MNs, along a transmission path. We consider the following cases. 1) If an adversary compromises a switch between the sender and the first MN, he can obtain the sender's address but not the receiver's; 2) If an adversary compromises a switch between the last MN and the receiver, he can obtain the receiver's address but not the sender's; 3) If an adversary compromises a switch between the first MN and the last MN, he can obtain neither the sender's nor the receiver's address. Therefore, the adversary cannot obtain both the sender and the receiver at any single point, and the global adversary is out of our threat model. As any switch in the network is likely to be an MN, an adversary cannot tell which is the first (or last) MN for a specific flow.

Compromising the Initiator or Responder: The adversary compromises the initiator (or the responder), seeking to obtain the identity of the node which is communicating with it, to determine the next attack target. If the responder is a hidden receiver, the initiator does not know the identity of the responder, and the responder has no idea of the initiator. Therefore, compromising the initiator (or the responder) cannot break the unlinkability of an m-flow.

Traffic Observing Attack: The adversary may observe (e.g., using the mirror ports in switches) the traffic on a switch, and analyze the traffic to correlate ingress and egress packets in the same flow. By iterated traffic analysis, the adversary may eventually correlate the entire m-flow. The observation of the global traffic in data centers is unproductive, since the mirror ports are not enabled on all switches by default. Our partial multicast mechanism helps to prevent the adversary correlating with ingress and egress packets at a single MN.

Size- or Rate-Based Traffic-Analysis: The adversary may count the packet number (or size) and transmission rate at various points, seeking to analyze the traffic patterns (size or rate) of a dedicated initiator (or responder), thereby inferring what operations or businesses are processing. Our multiple m-flow mechanism can reduce the effectiveness of this attack significantly. The adversary does not know the flow number within a channel, and it is hard to correlate the flows in the same channel even if he knows the number. Even if an adversary has obtained an m-flow's traffic pattern, he cannot know the channel's traffic pattern as well.

Denial-of-Service (DoS) Attack: The traditional overlay-based anonymity systems are vulnerable to DoS attacks. As the mixs (or relays) in a circuit are assigned by the initiator client, malicious users can easily launch DoS or DDoS attacks by creating a large mount anonymous request via the same mixs to consume their resources. As a result, due to the CPU-expensive asymmetric cryptography operations and limited packets forwarding capability (typically a mix has one or two network interfaces), the overlay-based anonymity systems face low vulnerability against DoS attacks. MIC is robust against DoS attacks in three aspects. 1) All the MNs are switch nodes, and thereby have strong packets forwarding capabilities. 2) All the transmission paths are determined by the MC, so users cannot assign the MNs or routing path, and all traffic will be evenly distributed to different transmission paths. 3) There is no cryptography operations on MNs, and therefore can reduce CPU consumption significantly.

Payload-Based Correlation: As there are no cryptographic operations on each MN, the adversary may correlate the packets by checking their payloads. MIC can reduce the effective of this correlation by the partial multicast mechanism. The adversary can observe multiple packets with the same payloads at different places, therefore he cannot confirm the correlation. Only when the adversary correlates the packets with the sender and receiver's real addresses, respectively, the adversary can reveal the identities of the communication participants. We are also considering of using the Internet Protocol Security (IPsec) to encrypt the payloads on each MN, so that the payload-based correlation can be prevented. However, this requires modifications on the SDN switches and will incur much higher overhead.

Anonymous Abuse: Most of the traditional anonymity systems are faced with the abuse issues. Users can send unlimited untraceable traffic to anonymity systems, causing issues like spamming,[4] sybil attack[5] and illegal trade. In data centers,

[4]https://en.wikipedia.org/wiki/Spamming
[5]https://en.wikipedia.org/wiki/Sybil_attack

anonymous abuse can cause great waste of resources. Unlike the previous anonymity systems, MIC provides anonymous communications to protect user privacy against malicious users or hackers but not the Cloud Service Providers (because the entire network in data centers must be managed by the CSP). Therefore, the MC, which has the global view of the network, can address the anonymous abuse issues effectively.

VI. Deployment and Applicability

In this section, we discuss how to enable MIC co-existing with other systems which are potential incompatible with MIC and how to deploy MIC in distributed systems, e.g. distributed file system CapFS.

A. Co-Existence With Traditional Systems

MIC modifies the packet header at switch nodes to achieve anonymity. However, many other systems in the network need to use or analyze the packer header information to facilitate their functions, such as the network packet analysis systems (e.g., Wireshark)[6] which will de-capsulate the packet header to display or analyze the network traffic, the intrusion detection systems (e.g., Snort)[7] which may check the packet header to detect malicious activity, and the firewall systems (e.g., Iptables)[8] which may check the packet address to prevent unauthorized communications. As MIC changes the packet header, the real packet header and traffic patterns are concealed. Therefore, exceptions or errors may occur in above-mentioned systems when they are deployed in the same network where MIC resides. All these systems are important components of the network, so MIC should be able to coexist with them.

In order to address the problem, we provide $interfaces$ to these systems for converting the m-flow to the original flow. The key point is to restore the packet header of the m-flow back to the original one, which is the reverse process of our M-address Generation Algorithm. Here we briefly summarize the workflow of converting the original flow into the m-flow and the reverse process, i.e., restoring the m-flow back to the original flow.

1) Converting the Original Flow Into the m-Flow: MIC translates the real packet header information into a fake one. Specifically, the three-tuple $< src\_ip, dst\_ip, mpls >$ is converted into an m-address $< m\_src\_ip, m\_dst\_ip, m\_mpls >$, and the other items are modified accordingly. The workflow of converting the original flow to m-flow can be found in Section IV-B3.

2) Restoring the m-Flow Back to the Original Flow: Similarly, we can restore the m-flow back to the original flow through the reverse process. Specifically, we can calculate the identity V of the flow using the m-address $< m\_src\_ip, m\_dst\_ip, m\_mpls >$, and then find the original header information through the flow identity V. Therefore, we need to store the original header of the flow (in an o_table) when generating an m-flow. The workflow of restoring an m-flow back to the original flow is shown as follows:

1) For an input packet, we first check the MPLS field. If the MPLS is null or satisfies $h(x_1, x_2) = C$, then the flow is a common flow.
2) Otherwise, the identity of the MN is $S = h(x_1, x_2)$. Then we find the hash function for the MN $F()$ using the S. The identity of the m-flow can be calculated as $V = F(m\_src\_ip, m\_dst\_ip, mpls_1, mpls_2)$.
3) Finally, the corresponding original header information of the m-flow can be found in the stored table o_table.

We can expose the *interfaces* for restoring the m-flow to the needed systems, so that MIC can co-exist with them. In order to ensure security, the systems which request for the *interfaces* should register in the MC controller and get the corresponding access permission before they use the *interfaces*. The MC controller can grant different levels of permissions for different systems. For example, for some traffic monitoring or intrusion detection systems, they only need to distinguish the real abnormal network behaviors and the m-flow, but do not need to know the exact packet information (such as the IP address and port). Therefore, we can provide them low permission level *interfaces*, so that they can use it to identify the m-flows without knowing the exact flow information. However, for the firewall systems, they have to know the exact packet information so that they can match the rules precisely (e.g. blocking all the flow from A to B). Therefore, we should give them the fully authorized *interfaces*, so that they can restore the m-flow to original flow.

B. Application Integration

MIC is designed for data center applications, which can be easily deployed in distributed systems. There are two ways in MIC deployment, including the full deployment and the lightweight deployment. In the full deployment, the applications in both the initiator and responder should modify their implementation to support MIC. The initiator can establish a mimic channel explicitly before data transmission, so that can use nicknames instead of read IP addresses to communicate with the responder, achieving receiver anonymity.

In the lightweight deployment (which is denoted as L-MIC), MIC eliminates the need of modification in the applications by sacrificing a certain level of anonymity. The L-MIC will automatically generate an m-flow for the session when the initiator creates a session between the responder, so that the applications can use MIC transparently. Users or administrators can specify the host pairs which should communicate using MIC in advance. When a Packet-in event is triggered for routing generation, the MC will check the packet header to decide whether to generate an m-flow. However, as no modification is involved in the initiator, the identity of the responder must be acquired by the initiator, so the receiver anonymity fails. Moreover, as both the sender and the receiver addresses are carried by the packets between the initiator and the first MN, once the first MN is compromised, the adversary can break the unlinkability of the flow. To relieve the effectiveness of this kind of attacks, L-MIC can be deployed in the virtualized

[6]https://www.wireshark.org/
[7]https://www.snort.org/
[8]https://en.wikipedia.org/wiki/Iptables

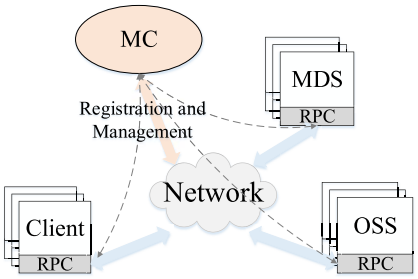


Fig. 7. The architecture of MIC-based CapFS.

environment, where the host server is under the protection of other security systems like intrusion detection systems (IDS). In this scenario, the virtual switches in the host server can be regarded as the first MN, so any nodes out of the host server cannot obtain both the sender and receiver identities.

In order to verify the applicability of the MIC, we integrate MIC in the RPC (Remote Procedure Call) [17], [18] and then use it to achieve anonymity in our distributed file system CapFS. RPC[9] is a widely used communication protocol in distributed systems, such as NFS.[10] It hides the complexity of the low-level network operations across different networks and provides uniform interfaces for upper-level applications, enabling the applications to invoke the remote procedures transparently just like local calls.

We build our distributed file system CapFS using the Sun's Transport-Independent RPC (TI-RPC) library.[11] CapFS consists of Clients, which provides POSIX-compliant file access interfaces for applications, OSSs, which store the data objects, and MDSs, which store the meta-data information of the data objects. The OSSs and MDSs are key nodes in the system. If an adversary want to crash the system, then he can first identify one of the MDSs by analyzing at the network or at one of the Client nodes and lunch DDoS attacks after that. MIC can protect the system against this kind of attacks effectively. In our anonymous CapFS, each communication inside CapFS is anonymous, therefore, it is hard for the adversaries to locate the key nodes of the system or have a sight of the system deployment details.

To achieve this, we first implement the MIC-based RPC, just replacing the socket interfaces in the RPC with the MIC socket-like interfaces. Then, we use the MIC-based RPC to replace the original RPC in CapFS. As no real IP address is used in the system, we have to assign each node a nickname, such as $mds0$, $osd0$, $client0$ and so forth. Each node should request to the MC for hidden service registration at initialization, so that others can reach it without knowing its real IP address. Therefore, each communication inside CapFS is achieved using the nicknames. As the MC keeps the identity of each node in the system, access control can also be enforced. The architecture of MIC-based CapFS is illustrated in Fig. 7.

VII. Evaluation

We build a test platform on Mininet [19]. The hardware consists of one server running Ubuntu 12.04.5 LTS operating system, with Intel (R) Xeon (R) E5-2620 @ 2.00GHz CPU, 32GB RAM. We install Mininet 2.2.0, Openvswitch 2.1.0, and Ryu 3.17 [20] on it. The network consists of 16 hosts interconnected using a Fat-tree of twenty 4-port switches, as shown in Fig. 5. We evaluate the performance of MIC compared with Tor, TCP and SSL in terms of route setup latency, transmission latency and throughput. MIC-TCP and MIC-SSL in our evaluation are two MIC versions which based on TCP and SSL, respectively.

MIC Implementation: MIC prototype consists of two modules: the user-end module and the MC module. We implement the user-end module on Linux platform. MIC employs typical C/S model, providing socket like programming APIs, and thus a programmer can use MIC for anonymous communication easily. We implement the MC on Ryu, a popular SDN controller platform. The communication between the client and the MC is encrypted using private key. When a client builds up a mimic channel for the first time, he should exchange a private key with the MC in advance using asymmetric encryption algorithms, like RSA [21] or D-H [22].

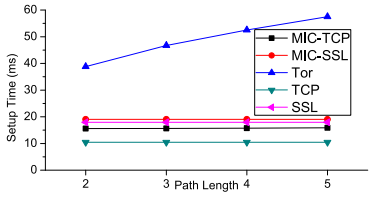


Fig. 8. Route setup time comparison among MIC, Tor, TCP and SSL.

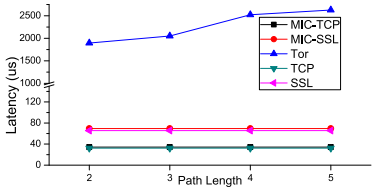


Fig. 9. Latency comparison among MIC, Tor, TCP and SSL.

A. Route Setup Latency

We evaluate the route setup latency of MIC, Tor, TCP and SSL. For MIC, we measure the "MIC_connect" function time on the initiator. We use the AES function in OpenSSL for encrypting/decrypting the *request* packet. For Tor, we measure the "connect" time on the client. Specifically, we redirect the traffic to our local Tor testbed by using the "torsocks" command, and vary the route length by modifying the "DEFAULT_ROUTE_LEN" in the Tor source code. We also evaluate TCP and SSL as the base line.

Fig. 8 plots the results of the route setup time varying the route length. The route length is the number of relay stages along the path. As one would expect, MIC outperforms Tor in route setup time, due to the more lightweight processing and shorter transmission path. The route setup time increases with increased route length in overlay-based Tor but remains nearly the same in in-network based MIC. That is because the

[9]https://en.wikipedia.org/wiki/Remote_procedure_call
[10]https://en.wikipedia.org/wiki/Network_File_System
[11]https://sourceforge.net/projects/libtirpc/

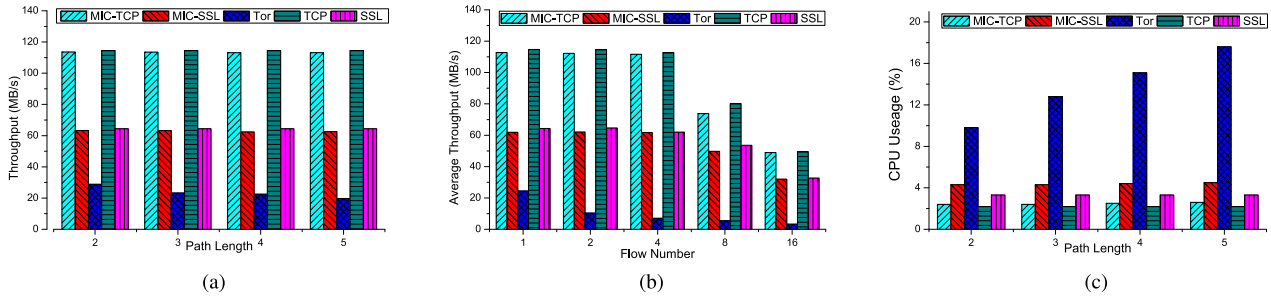


Fig. 10. Throughput comparison among MIC, Tor, TCP and SSL. (a) and (b) show the throughput comparison, (c) shows the CPU usage of evaluation (a).

operations on each MN are very lightweight, and the actual length of transmission path will not increase (significantly) with increased route length. Compare to the base line TCP and SSL, MIC requires additional time for sending request to the MC, therefore, resulting a little overhead.

B. Latency and Throughput

We evaluate the latency and throughput among MIC, Tor, TCP and SSL after the session is established. In the latency evaluation, we measure the time from when the sender sends 10 bytes data to the receiver until the receiver sends 10 bytes data back. Fig. 9 plots the results of latency. As can be seen from the results, MIC (including MIC-TCP and MIC-SSL) outperforms Tor significantly in terms of latency, and MIC-TCP is comparable with TCP, MIC-SSL is comparable with SSL. Compared to Tor, MIC has fewer cryptographic operations and shorter transmission path (the network paths and host protocol stacks), so that achieving lower latency. Compared to TCP (or SSL), MIC only incurs more "actions" in flow-table on MNs, whose overhead is substantially negligible.

In the throughput evaluation, we use Iperf for Tor and TCP test, and a modified Iperf for MIC and SSL. We first evaluate the throughput of one flow in different path lengths, and then evaluate the average throughput of various number of flows (the path length is set to default 3). Fig. 10(a) and (b) shows the throughput comparison among MIC, Tor, TCP and SSL. MIC achieves higher throughput than Tor due to its lightweight design. It's not a surprise to see Tor's average throughput decreases badly as the path length or flow number increases, as Tor employs the heavyweight overlay-based design. In Tor, each anonymous communication will occupy a large number of redundant network and computational resources than a common (non-anonymity) communication needs. Therefore, Tor will saturate the data center network quickly as the flow number increases, resulting in traffic congestion. However, MIC does not induce much additional length over the original (non-anonymity) path length, thereby can achieve high performance which is comparable with TCP (or SSL).

We also evaluate the overall CPU usage of MIC, Tor, TCP and SSL when performing the first throughput evaluation, as shown in Fig. 10 (c). The results show that the CPU overhead on MIC has a narrow increasement than TCP (or SSL) due to the extra operations on virtual switches. However, Tor suffers from extremely high overhead due to the significant redundant route paths and intermediate operations.

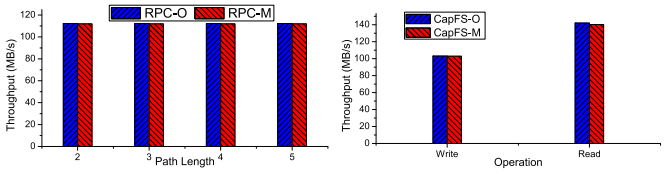


Fig. 11. Performance comparison among MIC-based applications. (a) Throughput comparison between RPC-O and RPC-M. (b) Write and Read throughput comparison between CapFS-O and CapFS-M.

C. Application-Based Evaluation

We evaluate the performance comparison between the original RPC (denoted as RPC-O) and the MIC-based RPC (denoted as RPC-M); the original distributed file system (denoted as CapFS-O) and the MIC-based CapFS (denoted as CapFS-M).

First, we evaluate the throughput of the RPC. During the evaluation, the sender transmits 1GB data to the receiver and each remote call carries 1MB data. Fig. 11 (a) polts the throughput comparison between the RPC-O and RPC-M. As can be seen from the result, the RPC-M is comparable with RPC-O in terms of throughput, as MIC only incurs little overhead over TCP.

Second, we evaluate the read and write performance of the CapFS. As MIC prototype is implemented in user space, we choose to use the user-space CapFS version which is based on FUSE (Filesystem in Userspace).[12] Unfortunately, we fail to mount our FUSE based Client in our Mininet test platform, so we further build a VM-based testbed. In this testbed, the host server runs Ubuntu 12.04 operating system, and three VMs, which are configured as MDS, OSS and Client nodes in CapFS respectively, are created by KVM (Kernel Virtual Machine).[13] The VMs are interconnected by Openvswitch instances installed on the host server. We use Iozone[14] benchmark for CapFS performance evaluation. Fig. 11 (b) plots the results. It's not a surprise to see CapFS-M is analogous to

[12]https://en.wikipedia.org/wiki/Filesystem_in_Userspace
[13]http://www.linux-kvm.org/page/Main_Page
[14]http://www.iozone.org/

TABLE I
PER-SWITCH RULE NUMBER VARYING THE TOPOLOGY SCALE AND HOST NUMBER

Topo	Hosts	Flows per host					
		Average			Max		
		1	5	10	1	5	10
k=4	16	7	38	74	12	52	86
k=8	128	15	79	158	30	110	196
k=16	1024	32	159	319	60	200	398
k=32	8192	64	319	639	104	406	740
k=48	27648	95	479	959	148	594	1100

TABLE II
GENERATION TIMES FOR DIFFERENT M-FLOW NUMBER

M-flow Number	1	2	3	4	5
Calculate(ms)	0.5	0.603	0.813	0.937	1.053
Install(ms)	4.887	9.595	14.384	19.113	23.716

CapFS-O in terms of read and write throughput, as negligible overhead is incurred in MIC and RPC-M.

In summary, from the evaluations presented above we can see that, MIC has very good applicability and negligible overhead, which can be easily deployed in most of the existing distributed systems inside data centers.

D. Scalability Analysis

To maximize the anonymity of the m-flows, the forwarding rules of each m-flow should be as randomized as possible. Therefore, MIC cannot leverage the wildcard rule to boost the forwarding efficiency, which will naturally cause large number of rules in switches. To mitigate this issue, we should balance the m-flows across each path so that each switch will carry the similar number of rules. Simply, when generating the forwarding routing, we randomly select a equal cost path for each m-flow. We evaluate the per-switch rule number when varying the topology scale (Fat-Tree topology) and host number, as shown in Table I. As the storage capacity of current SDN switches is around 1500 rules, the results indicate that MIC can work well in a large data center with 27648 hosts when 10 flows per host.

As can be seen from Section IV-B2, the time complexity of routing calculation of MIC is $O(|F|)$, where the $|F|$ is the m-flow number of a channel. By default, each MIC has one m-flow, and the m-flow number of a single MIC is generally less than 10. We evaluate the generation time of a mimic channel on the MC, as shown in Table II. The generation time includes routing calculation time and install time. The install time is depended on the design of the SDN controller and switches, which is not the concern of this paper. Moreover, much studies have tried to reduce the rule update latency on switches, such as RuleTris [23]. From the results in Table II we can see that the controller can process around 2000 flows per second. In our implementation, we use the Ryu open source SDN controller, which only supports single thread processing. We believe that, with multi-thread processing, the controller can process much more flows per second.

As we adopt the centralized approaches, MIC will naturally suffer from the single point failure and scalability issues. Fortunately, lots of efforts have been made on the scalability issues in SDN, such as distributed controllers [24]. MIC can be easily deployed on distributed controllers. As long as we ensure each MIC has a unique ID, our collision avoidance mechanism can guarantee the correctness of routing, and each mimic channel can be handled by a single controller independently. Therefore, we can assign a unique ID space for each controller to make MIC work among multiple controllers.

VIII. RELATED WORK

To protect the identity of the user or service provider and defeat traffic-analysis attacks, anonymity systems has been extensively studied. Prior anonymity systems are primarily based on Mix-net [25], DC-net [26], verifiable shuffles [27], [28] or broadcast (multicast). Existing anonymity systems can be divided into two categories in accordance with the latency: high-latency anonymity systems and low-latency anonymity systems.

High-Latency Anonymity Systems: These systems are mainly designed for applications which requires strong anonymity but can tolerate significant high latency, such as E-mail, including Babel [29], Mixmaster [30], Mixminion [4]. This systems are based on Mix-Nets, in which the messages are typically delayed for hours for batching to maximize anonymity and achieve strong traffic-analysis resistance against even a global adversary.

Low-Latency Anonymity Systems: These systems are mainly designed for interactive applications like web browsing and Internet chat. Anonymizer [31] is the simplest low-latency anonymity system, which has only one proxy. Onion routing [32] is a real-time variant of Mix-Net in early time. Before transmitting messages, the sender picks up a list of mixs (called relays) and constructs a bi-directional circuit with the receiver via the intermediate relays. The sender layered-encrypts the messages, and each relay decrypts them then forwards them to the next hop in the circuit. Each relay knows only its previous and next hops, but has no idea of the communication participants. The second-generation Onion Routing, Tor [6] is volunteer-based, and becomes the most popular anonymity system deploy in the real word. A large number of studies focus on the attacks on Tor, such as by cell-counting [33], cell manipulating [34] and flow correlation [35], [36]. Fu *et al.* [37], [38] also propose modeling and analysis on performance in mix networks.

In P2P architecture based anonymity systems, each node can be either the traffic initiator (or recipient) or forwarder. Crowds [5] hides the traffic originator among a large number of members. MorphMix [39] makes anyone can easily join the system instead of building static mix network, and provides collision detection mechanism to identify compromised paths to enhance robust. Tarzan [40] uses cover traffic to obscure traffic patterns to defeat global observers. Aqua [41] focuses on providing high-bandwidth and strong anonymity communication for BitTorrent. Herd [42] provides scalable and traffic-analysis resistant anonymity network for VoIP systems.

Hordes [43], P5 [44] and Herbivore [45] adopt multicast or broadcast mechanisms to achieve anonymity. Dissent [7] is built on DC-Net and verifiable shuffle, providing low latency, high scalability and strong anonymity. Information Slicing [46] tries to achieve anonymity communication without using public key through multi-path and secret sharing. LAP [47] provides low-latency and lightweight anonymity to protect daily online activities which are impatient to wait. iTAP [48] adopts the same idea of using SDN to prevent traffic analysis, but not focuses on data centers. As far as we are aware, MIC is the first anonymity scheme designed for data centers.

IX. Conclusion

We present MIC, an efficient anonymity scheme aimed for data center environment. Different from the traditional overlay-based architecture, MIC adopts an in-network design, which conceals the communication participants' identifies by modifying the source/destination addresses (e.g., MAC, IP and port) at switch nodes. To address the challenge of potential routing collision in MIC, we propose a routing collision avoidance mechanism. We also propose three mechanisms, the Multiple M-flows, Dynamic M-Flows and Partial Multicast mechanisms, to enhance the traffic-analysis resistance of MIC, and discuss the solutions to enable MIC co-existing with other systems. As a result, we can improve anonymity of applications within data centers at negligible overhead. Experimental results show that MIC outperforms Tor significantly in performance, and is comparable with TCP (or SSL). Moreover, we design and implement MIC-based distributed file system (CapFS-M), and the experimental results on CapFS-M show the applicability and efficiency of MIC.

References

[1] T. Zhu *et al.*, "MIC: An efficient anonymous communication system in data center networks," in *Proc. ICPP*, Aug. 2016, pp. 11–20.

[2] (2015). *IBM Security Services 2015 Cyber Security Intelligence Index*. [Online]. Available: http://www-03.ibm.com/security/data-breach/2015-cyber-security-index.html

[3] (2014). *The Untold Story of the Target Attack Step by Step*. [Online]. Available: https://aroundcyber.files.wordpress.com/2014/09/aorato-target-report.pdf

[4] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type III anonymous remailer protocol," in *Proc. Secur. Privacy*, May 2003, pp. 2–15.

[5] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for Web transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, pp. 66–92, 1998.

[6] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proc. USENIX Secur. Symp.*, Berkeley, CA, USA, 2004, p. 21.

[7] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in numbers: Making strong anonymity scale," in *Proc. OSDI*, Berkeley, CA, USA, 2012, pp. 179–192.

[8] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[9] C. Guo *et al.*, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. SIGCOMM*, New York, NY, USA, 2009, pp. 63–74.

[10] J. Kirch, "Virtual machine security guidelines version 1.0," Center Internet Secur., New York, NY, USA, White Paper., Sep. 2007. [Online]. Available: https://www.cisecurity.org/wp-content/uploads/2017/04/CIS_VM_Benchmark_v1.0.pdf

[11] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. CCS*, New York, NY, USA, 2009, pp. 199–212.

[12] *Amazon Elastic Compute Cloud (Amazon EC2)*. Accessed: 2017. [Online]. Available: http://aws.amazon.com/ec2/

[13] T. Zhu *et al.*, "MCTCP: Congestion-aware and robust multicast TCP in software-defined networks," in *Proc. IWQoS*, Beijing, China, Jun. 2016, pp. 1–10.

[14] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, Berkeley, CA, USA, 2010, p. 19.

[15] H. H. Liu *et al.*, "zUpdate: Updating data center networks with zero loss," in *Proc. SIGCOMM*, New York, NY, USA, 2013, pp. 411–422.

[16] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," IETF, Tech. Rep. RFC 3031, Jan. 2001.

[17] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Trans. Comput. Syst.*, vol. 2, no. 1, pp. 39–59, Feb. 1984.

[18] R. Thurlow, "RPC: Remote procedure call protocol specification version 2," IETF, Tech. Rep. RFC 5531, May 2009.

[19] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proc. CoNEXT*, New York, NY, USA, 2012, pp. 253–264.

[20] *Ryu*. Accessed: 2017. [Online]. Available: http://osrg.github.io/ryu

[21] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[22] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[23] X. Wen *et al.*, "RuleTris: Minimizing rule update latency for TCAM-based SDN switches," in *Proc. ICDCS*, Jun. 2016, pp. 179–188.

[24] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. HotSDN*, New York, NY, USA, 2014, pp. 1–6.

[25] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[26] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *J. Cryptol.*, vol. 1, no. 1, pp. 65–75, 1988.

[27] J. Furukawa and K. Sako, *An Efficient Scheme for Proving a Shuffle*. Berlin, Germany: Springer-Verlag, 2001, pp. 368–387.

[28] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *Proc. CCS*, New York, NY, USA, 2001, pp. 116–125.

[29] C. Gulcu and G. Tsudik, "Mixing e-mail with babel," in *Proc. Symp. Netw. Distrib. Syst. Secur.*, Feb. 1996, pp. 2–16.

[30] U. Moller and L. Cottrell. (Jan. 2000). *Mixmaster Protocol—Version 2 Draft*. [Online]. Available: http://www.eskimo.com/rowdenw/crypt/Mix/draft-moeller-mixmaster2-protocol-00.txt

[31] *Anonymizer*. Accessed: 2016. [Online]. Available: https://www.anonymizer.com/

[32] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *Proc. Int. Workshop Inf. Hiding*, London, U.K., 1996, pp. 137–150.

[33] Z. Ling *et al.*, "A new cell-counting-based attack against Tor," *IEEE/ACM Trans. Netw.*, vol. 20, no. 4, pp. 1245–1261, Aug. 2012.

[34] X. Fu and Z. Ling, "One cell is enough to break tor's anonymity," in *Proc. Black Hat Tech. Secur. Conf.*, Feb. 2009, pp. 578–589.

[35] Y. Zhu, X. Fu, R. Bettati, and W. Zhao, "Anonymity analysis of mix networks against flow-correlation attacks," in *Proc. GLOBECOM*, vol. 3. Nov. 2005, pp. 1–5.

[36] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao, "On flow correlation attacks and countermeasures in mix networks," in *Proc. 4th Int. Conf. Privacy Enhancing Technol.*, Berlin, Germany, 2005, pp. 207–225.

[37] X. Fu, W. Yu, S. Jiang, S. Graham, and Y. Guan, "TCP performance in flow-based mix networks: Modeling and analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 695–709, May 2009.

[38] R. Pries, W. Yu, S. Graham, and X. Fu, "On performance bottleneck of anonymous communication networks," in *Proc. IEEE IPDPS*, Apr. 2008, pp. 1–11.

[39] M. Rennhard and B. Plattner, "Introducing morphmix: Peer-to-peer based anonymous Internet usage with collusion detection," in *Proc. WPES*, New York, NY, USA, 2002, pp. 91–102.

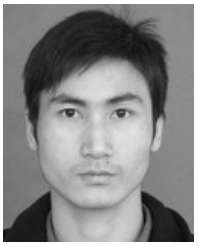
[40] M. J. Freedman and R. Morris, "Tarzan: A peer-to-peer anonymizing network layer," in *Proc. CCS*, New York, NY, USA, 2002, pp. 193–206.

[41] S. Le Blond *et al.*, "Towards efficient traffic-analysis resistant anonymity networks," in *Proc. SIGCOMM*, New York, NY, USA, 2013, pp. 303–314.

[42] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt, "Herd: A scalable, traffic analysis resistant anonymity network for voip systems," in *Proc. SIGCOMM*, New York, NY, USA, 2015, pp. 639–652.

[43] B. N. Levine and C. Shields, "Hordes: A multicast based protocol for anonymity," *J. Comput. Secur.*, vol. 10, no. 3, pp. 213–240, Sep. 2002.

[44] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P5: A protocol for scalable anonymous communication," in *Proc. Secur. Privacy Symp.*, 2002, pp. 58–70.
[45] S. Goel, M. Robson, M. Polte, and E. Sirer, "Herbivore: A scalable and efficient protocol for anonymous communication," Cornell Univ., Ithaca, NY, USA, Tech. Rep. 2003-1890, 2003.
[46] S. Katti, J. Cohen, and D. Katabi, "Information slicing: Anonymity using unreliable overlays," in *Proc. NSDI*, Berkeley, CA, USA, 2007, p. 4.
[47] H.-C. Hsiao *et al.*, "Lap: Lightweight anonymity and privacy," in *Proc. SP*, Washington, DC, USA, 2012, pp. 506–520.
[48] R. Meier, D. Gugelmann, and L. Vanbever, "iTAP: In-network traffic analysis prevention using software-defined networks," in *Proc. SOSR*, New York, NY, USA, 2017, pp. 102–114.



Tingwei Zhu received the B.E. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2012, where he is currently pursuing the Ph.D. degree in computer architecture. He has several publications in major journals and international conferences, including IWQoS, ICPP, and JNCA. His interests include software-defined networking and distributed storage systems.

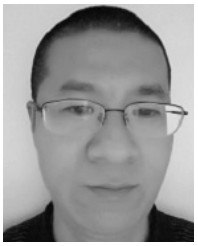
Dan Feng (M'05) received the B.E., M.E., and Ph.D. degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1991, 1994, and 1997, respectively. She is currently a Professor and a Vice Dean of the School of Computer Science and Technology, HUST. She has over 100 publications in major journals and international conferences, including the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the *ACM* TRANSACTIONS ON STORAGE, JCST, FAST, USENIX ATC, ICDCS, HPDC, SC, ICS, IPDPS, and ICPP. Her research interests include computer architecture, massive storage systems, and parallel file systems. She is a member of the ACM. She serves on the program committees of multiple international conferences, including SC 2011 and 2013 and MSST 2012.

Fang Wang received the B.E. degree, the master's degree in computer science, and the Ph.D. degree in computer architecture from the Huazhong University of Science and Technology (HUST), China, in 1994, 1997, and 2001, respectively. She is currently a Professor of computer science and engineering with HUST. She has over 50 publications in major journals and international conferences, including FUTURE GENERATION COMPUTER SYSTEMS, *ACM* TRANSACTIONS ON ARCHITECTURE AND CODE OPTIMIZATION, *SCIENCE CHINA Information Sciences*, the *Chinese Journal of Computers*, HiPC, ICDCS, HPDC, and ICPP. Her interests include distribute file systems, parallel I/O storage systems, and graph processing systems.

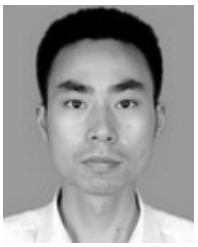
Yu Hua (SM'13) received the B.E. and Ph.D. degrees in computer science from Wuhan University, China, in 2001 and 2005, respectively. He is currently a Professor with the Huazhong University of Science and Technology, China. He has over 80 papers to his credit in major journals and international conferences, including the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, USENIX ATC, USENIX FAST, INFOCOM, SC, ICDCS, ICPP, and MASCOTS. His research interests include computer architecture, cloud computing, and network storage. He is a Senior Member of the CCF, and a member of the ACM and the USENIX. He has been on the organizing and program committees of multiple international conferences, including INFOCOM, ICDCS, ICPP, RTSS, and IWQoS.

Qingyu Shi received the B.E. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2014. He is currently pursuing the Ph.D. degree in computer architecture with the Wuhan National Laboratory for Optoelectronics. His interests include software-defined networking and network storage system.

Jiahao Liu received the B.E. degree in computer science and technology from the China University of Mining and Technology, Xuzhou, China, in 2014. He is currently pursuing the Ph.D. degree in computer architecture with HUST. His interests include distributed file system and software-defined storage.



Yongli Cheng received the B.E. degree from Chang'an University, Xi'an, China, in 1998, and the M.S. degree from Fuzhou University, Fuzhou, China, in 2010. He is currently pursuing the Ph.D. degree in computer architecture with the Huazhong University of Science and Technology, Wuhan, China. He is currently a Teacher with Fuzhou University. He has several publications in international conferences, including HPDC and IWQoS. His current research interests include computer architecture and graph computing.



Yong Wan received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, China, in 2013. He is currently an Assistant Professor with the School of Computer Engineering, Jingchu University of Technology. His research interests include computer networks and protocols, high-performance network cluster, and parallel and distributed systems.