分 类 号_____ 学号 D201277701

学校代码 10487 密级_____

華中科技大学

博士学位论文

基于软件定义网络的分布式存储系统优化技术研究

学位申请人： 朱挺炜

学 科 专 业： 计算机系统结构

指 导 教 师： 冯 丹 教授

答 辩 日 期： 2017 年 11 月 26 日

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Engineering

Research on Distributed Storage System Optimizations based on Software-Defined Networking

Ph.D. Candidate : Zhu Ting Wei

Major : Computer Architecture

Supervisor : Prof. Feng Dan

独创性声明

本人声明所呈交的学位论文是我个人在导师的指导下进行的研究工作及取得的研究成果。尽我所知，除文中已标明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：朱挺炜

日期：2017年 11月 26日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于

保密 □，在 ____ 年解密后适用本授权书。

不保密 ☑。

（请在以上方框内打“√”）

学位论文作者签名：朱挺炜

日期：2017年 11月 26日

指导教师签名：

日期：2017年 11 月 30 日

摘　要

互联网规模的急剧扩张与云计算技术的快速发展使得数据呈爆炸式增长。数据中心逐渐成为承载这个不断增长的数据宇宙的主要场所，预计到 2020 年约有 48% 的数据将存储在云数据中心内。分布式存储系统以其价格低廉、容量巨大、高可靠性以及高性能等优势成为存储数据的主要技术。随着新型存储介质存取速度的不断提升，存储系统中的网络传输逐渐成为整个系统的瓶颈。一方面，分布式存储系统的读写性能以及系统后台活动的性能都受存储网络带宽的限制；另一方面，随着规模的增长，分布式存储系统面临安全威胁的概率也将极大地增加。因此，研究如何提升分布式存储系统内的网络传输效率和降低存储系统遭受攻击的风险具有重要意义。本文针对数据中心环境下，围绕提升分布式存储系统中网络传输效率与安全性两方面展开了研究。具体而言，利用新型网络技术软件定义网络（Software-Defined Networking，SDN）实现了高效的网络组播传输方案、网络流量调度方案以及匿名通信方案，进而在分布式存储系统中实现更加高效与安全的网络传输。

提出一种拥塞感知的可靠组播方案 MCTCP，并且利用 MCTCP 实现组播多副本机制，从而有效地减少分布式存储系统中多副本写入过程的冗余报文，提升系统性能。MCTCP 通过扩展 TCP 协议来实现一对多模式下的可靠数据传输，并且利用 SDN 控制器来集中管理每个组播会话的状态。通过实时地监控网络链路状态，并且根据链路状态来实时调整组播会话的转发生成树以绕开拥塞或者失效链路，从而实现拥塞感知和高鲁棒性的路由转发。一方面，相比于传统的可靠组播方案，MCTCP 针对数据中心内分布式存储系统网络传输特征设计，具有比传统可靠组播方案更高的传输性能与更好的易用性。另一方面，基于组播的多副本机制比传统的多副本机制具有更少的冗余报文，从而达到更高的传输效率。基于组播的多副本机制中，MCTCP 方案比传统可靠组播方案达到更好的性能。实验表明，相比于原始版本分布式存储系统 HDFS-O，基于 MCTCP 的分布式存储系统 HDFS-M 多副本写操作带宽提高 1.5 倍。

提出一种动态的网络流量调度方案 MAX，可以在存储系统中实现跨前端与后端网络之间的网络流量调度，通过提高空闲网络资源的利用率来提升存储系统的恢复与重均衡性能。MAX 针对部署双网络的分布式存储系统设计。在每个存储节点上增加一个网络调度层，并使得存储系统的流量经由调度层转发后进入物理网卡。通过利用 SDN 集中化控制的能力，在 SDN 网络控制器上控制进入到调度层流量的路由来实现流量调度。因此，MAX 可以在不修改特定的分布式存储系统代码的情况下实现跨前端与后端网络间的动态流量调度。在存储系统前端无请求或者请求较低时，MAX 可以充分利用前端网络中的空闲网络带宽来优化系统的性能（如恢复与重均衡）。同时，提出一种基于优先级的路由方案保证后端网络中的流量不会影响前端网络请求的性能。实验表明，MAX 可以降低 Ceph 恢复和重均衡的时间，分别获得约 30% – 46% 和 30% – 43% 的时间节省。

提出一种基于全局路由冲突避免机制的网内匿名通信方案 MIC，并且利用 MIC 实现基于匿名通信的分布式存储系统（简称为匿名存储系统），通过匿名的方式有效提升存储系统的安全性。匿名存储系统中，存储节点成为动态的目标，从而扰乱攻击者的攻击链。MIC 的主要思想是通过在交换机上修改报文的源地址和目标地址来隐藏报文的发送端与接收端，从而达到匿名通信的目的。相比于传统的基于覆盖网络（overlay）的匿名方案，MIC 基于网内（in-network）的设计具有更短的传输路径和更少的中间操作，因此具有更高的性能。为了保证网络通信的正确性，设计了全局路由冲突避免机制，通过合理地分配每个流的地址来避免不同流之间的路由冲突。为了提升抗流量分析攻击能力，提出多匿名流与局部多播的机制。利用 MIC 实现了匿名存储系统 CapFS-M，从而实现存储系统中各个节点之间相互匿名，提升系统的安全性。实验表明，匿名存储系统 CapFS-M 相比于非匿名的 CapFS-O 引入带宽开销小于 1%。

关键词：分布式存储系统，软件定义网络，多副本，流量调度，匿名通信

Abstract

The rapid expansion of the Internet scale and the development of the cloud computing technology drive the data explosion. The data center is becoming the main venue for carrying the ever-growing digital universe, and about 48% of the data will be stored in cloud data centers by 2020. Distributed Storage System (DSS) is the main technology for storing the data volume due to its low price, massive capacity, high reliability, high performance and other advantages. As the speed of new storage medium continually improves, the network layer of the DSS has gradually becoming the bottleneck of the entire system. On the one hand, the performances of read/write operations as well as the background activities (such as recovery and re-balancing) of the DSS are limited by the storage network bandwidth. On the other hand, as the scale of the DSS increases, the probability of suffering from security attacks will also significantly increase. Therefore, it is important to study the issues that improving the efficiency of network transmission and reducing the risk of security attacks in DSSes. This paper focuses on improving the efficiency and security of the DSS in data centers by realizing efficient network transmission protocol, network scheduling scheme and secure transmission scheme based on the new network technology SDN (Software-Defined Networking).

Distributed storage systems typically adopt the data replication mechanism to improve data reliability. However, existing data replication schemes are implemented through multiple unicast transmissions, which will result in a large number of redundant network traffic, eventually leading to low performance in data replication. To address this problem, a congestion-aware and robust multicast protocol MCTCP and multicast-based data replication scheme which based on MCTCP are presented to improve the data replication performance in DSSes. The main idea behind MCTCP is to manage the multicast groups in a centralized manner, and reactively schedule multicast flows to active and low-utilized links,

by extending TCP as the host-side protocol and managing multicast groups in the SDN-controller. MCTCP achieves congestion-aware and robust routing by dynamically calculating and adjusting the multicast spanning trees (MSTs) according to the real-time network status, bypassing the congested and failed links. On the one hand, compared to the traditional reliable multicast schemes, MCTCP is designed for data replication scenes and is more efficient and easier to use. On the other hand, multicast-based data replication generates much less redundant network traffic than the traditional schemes, so that can achieve higher performance. Compared to the multicast-based data replication which based on the traditional multicast schemes, MCTCP-based data replication performs much better. Experimental results show that MCTCP-based HDFS (HDFS-M) outperforms the original HDFS (HDFS-O) up to 2 and 1.5 times in terms of lantency and bandwidth.

Mordern distributed storage systems typically deploy two networks, including the front-end and back-end networks, to improve performance. However, in the existing system design, the two networks are completely isolated, so the free network bandwidth across the two networks cannot be shared, resulting in considerably network resource waste. To address this issue, a dynamic network traffic scheduling scheme MAX is proposed, which can achieve network traffic scheduling across the front-end and back-end networks. MAX improves the distributed storage system performance by improving the free network bandwidth utilization. Specifically, MAX adds a virtual switch (vSwitch) between the storage system and the physical network card (NIC) on each storage node, so that all the traffic will travel through the vSwitch before reaching the physical NICs. Leveraging the SDN technology, a module in the SDN controller dynamically monitors all the traffic in vSwitches and calculates the placement for each large flow automatically. When the front-end network is free, MAX can make full use of the free network bandwidth in the front-end network to optimize the system performance (such as recovery and rebalancing). Moreover, to avoid the back-end traffic influencing the front-end requests performance, a priority-based routing scheme is proposed. Experimental results show that MAX decreases the recovery and rebalancing time in Ceph storage system by about $30\% - 46\%$ and $30\% - 43\%$, respectively.

As the scale increases, the distributed storage systems face much more security threats from the network and malicious users. In order to improve the security of the distributed storage system, an efficient anonymous communication scheme MIC and anonymous-based distributed storage system are proposed. Using anonymous communication between each node inside the DSS makes the storage nodes become dynamic targets, and thus disturbing the attacker's attack chain. The main idea behind MIC is to conceal the communication participants by modifying the source/destination addresses (such as MAC, IP and port) at switch nodes, so as to achieve anonymous communication. Compared with the traditional overlay-based anonymity approaches, the in-network based design used in MIC has shorter transmission paths and less intermediate operations, thus achieving higher performance with less overhead. To ensure the correctness of routing, a collision avoidance mechanism is propsed. To enhance the traffic-analysis resistance, two mechanisms, including multiple flow mechanism and partial multicast mechanism, are proposed. Based on MIC, an anonymous distributed storage system CapFS-M is presented. Experimental results show the efficiency of MIC and CapFS-M, e.g., achieving less than 1% overhead in terms of throughput.

Key words: Distributed Storage System, Software-Defined Networking, Data Replication, Traffic Scheduling, Anonymous Communication

目　录

1 绪论

随着互联网规模的急剧膨胀和应用类型的不断丰富，云计算应用模式快速普及，数据呈爆炸式增长。数据量的不断增长，加以物联网以及云计算技术的推动，使得数据中心成为了驾驭和承载这个数据时代的重要载体。IDC 预测数据将以每年超过 50% 的速率增长，到 2020 年将达到 44ZB[①]。HGST 的报告[②]指出，到 2020 年约有 48% 的数据将存储在云数据中心内。为了存储与处理这些不断增长的数据，分布式存储系统应运而生。与本地存储系统不同的是，分布式存储系统由多个存储节点组成，存储节点之间通过网络进行互联，从而具有大存储容量、高访问性能、高可靠性等优点。随着新型存储介质的不断发展，本地访问的速度不断增长，存储系统中网络逐渐成为系统的瓶颈。通过优化存储系统中网络传输的效率与安全性从而提升系统的整体性能与安全性具有重大的意义。软件定义网络技术（Software-Defined Networking，SDN）的兴起与发展为提升网络传输的效率和安全性提供了新的方法与思路，因此，利用 SDN 技术来优化存储系统也成为了新的研究方向。

1.1 基于对象的分布式文件系统及其网络技术研究

互联网应用的普及和高性能计算应用的发展，对存储系统的扩展性、容量、性能、可用性等提出了更高的要求。分布式文件系统经历了网络文件系统、SAN 文件系统两种架构的演进，到 2000 年，逐步发展出了基于对象的分布式文件系统架构。在之前的存储系统架构中，元数据服务器需要管理所有数据的物理视图，完成文件到磁盘的物理映射，因而成为阻碍系统扩展性的关键环节。对象存储技术将文件的物理视图下放到存储设备上，从而减少了元数据服务器的负担。其综合了“文件”的兼容共享与“块”的高性能等优势，从而可以实现较低成本下的高性能、大规模分布式文件系统。

① http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm
② http://dw.connect.sys-con.com/session/2049/Steve_Campbell.pdf

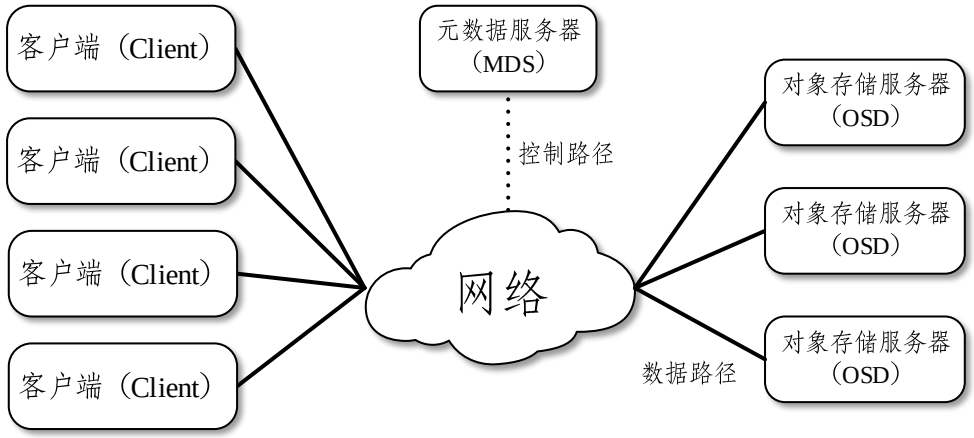


图 1-1 基于对象的分布式文件系统架构示意图

从架构上来看，基于对象的分布式文件系统采用三方架构，包括 MDS（MetaData Server）、OSD（Object Storage Device）和 Client，如图1-1所示。这种架构将控制路径（元数据请求等）与数据路径（数据请求等）分离开来，使得客户端可以直接访问存储服务器，而无需管理元数据的服务器进行数据转发。通过这种方式，将元数据的请求从数据请求的路径中分离开来，可以极大地提升系统的访问性能，提升系统的并发能力[1,2]。基于对象的分布式文件系统可以有效地存储与处理大数据时代产生的海量数据，已经成为当前数据中心内最为主要的分布式文件存储技术。代表性的基于对象的分布式文件系统包括 Linux 的 PVFS，Cluster 的 Lustre，Google 的 GFS（Google File System）以及其开源实现 HDFS，Redhat 的 Ceph 等。本实验室小组自主开发的 CapFS（Cappella File System）也属于基于对象的分布式文件系统。CapFS 由 Client（客户端）、OSS（对象存储服务器）和 MDS（元数据存储服务器）三部分组成，给用户提供标准的 Posix 接口。

PVFS[3,4] 最初是由 Walt Ligon 和 Eric Blumer 于 1993 年开发，作为 NASA 的一个基金来研究并行程序的 I/O 模式。2003 年 PVFS2①被推出，该系统采用新的设计，具有新的特性，包括对象服务器、分布式元数据服务器、支持 MPI、支持多种网络类型、具有高扩展性等。其主要的应用场景是针对数据挖掘、高性能计算等。于 2007

① http://dev.orangefs.org/old/documentation/releases/current/doc/pvfs2-guide/pvfs2-guide.php

年被移植以支持 IBM Blue Gene，后来被分成两个分支 “Orange” 和 “Blue”，于 2011 年 OrangeFS 被当作主线维护。

Lustre[5] 是由 Linux 与 Cluster 的合成词，其最初起源于 1999 年，由卡内基梅隆大学的 Peter J.Braam 开发。2001 年，Braam 开创自己的集群文件系统公司（Cluster File System Inc），于 2003 年发布 Lustre1.0。2007 年 Sun 公司收购集群文件系统公司，将 Lustre 加入到它的 ZFS 文件系统和 Solaris 操作系统中。2010 年 Oracle 公司收购 Sun 公司，开始管理和发布 Lustre。Lustre 主要应用于高性能计算领域。从 2005 年开始，至少有一半的 TOP 10 的超级计算机和超过 60 个 TOP 100 的超级计算机都使用 Lustre 来作为其存储系统。

GFS（Google File System）[6] 是由 Google 公司为其搜索引擎应用设计实现的一个分布式文件系统。其专门为 Google 的网页搜索的存储进行了优化设计，提供了专用的应用接口，而并不像 Lustre 这样的通用存储系统，提供标准的 POSIX 文件访问接口。GFS 的目标应用主要以大文件为主，访问模式主要是“一写多读”。因此，GFS 设计的性能目标是提供最大化的读写带宽。随着存储系统规模的增加，其部署成本是设计中考虑的另一个重要因素。GFS 可以完全运行在廉价的普通硬件设备上，而不需要昂贵的专用硬件设施。

HDFS[7] 是 GFS 的开源实现，是 Apache Hadoop 的分布式文件系统。其作为 Hadoop 大数据处理框架下的存储系统，支持 MapReduce 处理模型。HDFS 在设计上与 GFS 基本一致，包括一个 NameNode 节点，多个 DataNode 节点以及多个 Client 节点。与 GFS 上不同的是，HDFS 简化了写入的模型，只能支持单个客户端的写入，而不支持多个客户端同时写入同一个文件。从而在设计上，HDFS 比 GFS 更加简单。GFS 与 HDFS 都采用流水线的方式写入多个副本。

Ceph[8] 起源于加州大学圣克鲁斯分校（University of California, Santa Cruz）Sage Weil 博士论文研究。2007 年，Weil 博士毕业后专职研究 Ceph，并于 2012 年创建 Inktank Storage 为 Ceph 提供支持。2014 年 RedHat 公司收购 Inktank 公司，如今已经是开源项目，拥有非常活跃的开源社区。Ceph 已经加入到 Linux（从 2.6.34 开始）内核。Ceph 最初是一个分布式文件系统，提供标准的 POSIX 文件接口，如今发展成为提供块、文件和对象三种接口的统一存储系统。Ceph 主要有四个组件组成：监控集

群（Monitors），对象存储节点（OSD，Object Storage Device），元数据服务器（MDS）和客户端（Client）。Monitor 维护集群状态的各种表，包括监控群集图、OSD 表、归置组（Placement Group，PG）表和 CRUSH 表等；OSD 持久存储所有数据，处理数据的复制、恢复、回填和重均衡等；MDS 处理 Ceph 文件系统的元数据（只有文件系统需要 MDS，块、对象存储不需要 MDS）；Client 访问存储系统。Ceph 的主要特点包括：（1）利用 CRUSH[9] 算法计算每个对象的存储位置，以及（2）其可靠、智能的分布式对象存储引擎 RADOS[10]。CRUSH 算法将对象映射到一个归置组，然后再将归置组映射到存储节点 OSD 上。系统中所有成员，包括 Client，OSD, Monitor 和 MDS 都可以通过 CRUSH 算法获得一个对象的存储位置，从而使得系统的扩展、恢复、重均衡更加智能化。RADOS 智能地管理存储集群，使得系统可以自动地从不健康的状态（如节点失效）中恢复到正常，而无需管理员的手动干预。

分布式存储系统的网络层一直是影响系统性能的重要环节，因此，存储网络技术一直被广泛研究。目前，分布式存储系统网络层的优化技术研究主要集中在利用高速的 Infiniband 网络①与 RDMA（Remote Direct Memory Access）技术②来构建高性能的存储系统。InfiniBand 是针对服务器互连的高速网络技术，其设计愿景是取代 PCI、以太网以及光纤通道。IB 网络从 2000 年第一个版本发布起就具有很高的传输速率，目前 IB 网络产品能够达到 200Gbps 的传输速率。随着 IB 网络技术的发展，逐渐淡化了网络与总线之间的界线。IB 网络可以当作总线的延伸，同时也具有低时延特性。与总线中具有 DMA（Direct Memory Access）技术类似地，IB 网络具有 RDMA（Remote Direct Memory Access）功能。利用 RDMA 技术可以在无需 CPU 参与的情况下直接访问远程机器上的内存空间，从而能够在跨机器间通信时达到极低的时延。IB 网络的高带宽与低时延特性符合了很多存储系统的需求。因此，利用 IB 网络来构建分布式存储系统是提升存储系统性能的重要研究方向。

IB 网络一般给上层应用提供两种接口：（1）RDMA Verbs 接口③，即提供 RDMA 功能的编程接口和（2）套接字直接协议（Socket Direct Protocol, SDP）④，即将 Verbs 接

① https://en.wikipedia.org/wiki/InfiniBand
② https://en.wikipedia.org/wiki/Remote_direct_memory_access
③ https://thegeekinthecorner.wordpress.com/category/infiniband-verbs-rdma/
④ https://en.wikipedia.org/wiki/Sockets_Direct_Protocol

口封装成标准的 Socket 网络编程接口。Verbs 接口为应用程序提供了使用 RDMA 功能的编程能力。使用 SDP 可以让应用程序透明地利用 RDMA 通信机制来加速 TCP/IP 网络通信。

RDMA 的主要特点是可以在无需 CPU 干预的情况下实现访问远程机器的内存，从而达到低延迟的数据传输。RDMA 定义了一套异步网络编程接口，称为 RDMA Verbs，应用程序调用 Verbs 接口来获得 RDMA 的优势。最初，RDMA 技术是在 IB 网络上实现的，利用 IB 网络的能力来保证传输的可靠性。后来这项技术也在以太网上实现，包括 iWARP①和 RoCE(RDMA over Converged Ethernet)②两种。iWARP 是基于可靠传输协议（如 TCP 或者 SCTP）上的 RDMA 实现，RoCE 是基于用户数据报协议 UDP 上的实现。这两种协议都需要以太网卡硬件的支持。RDMA 技术在存储领域得到了广泛的应用，因而在 RDMA 之上研究出了一系列的协议，包括 iSER(iSCSI Extensions for RDMA)③、SMB Direct (SMB over RDMA)④、NFS over RDMA⑤ 等。

现有的主流分布式文件系统基本上都实现了对 IB 网络的支持与优化设计，例如 Ceph、HDFS、GlusterFS、PVFS、Lustre 和 Cappella 等。在这些分布式文件系统中，都是直接在该系统的网络模块中加入了对 RDMA 技术的支持。例如，CapFS 通过实现基于 RDMA verbs 接口的远程过程调用（RPC，Remote Procedure Call）层，来使得系统充分发挥 RDMA 带来的优势，提升整体性能。Ceph 通过一个支持 RDMA 的 RPC 中间层 Accelio⑥来利用 RDMA 的能力。GlusterFS 同样也实现了支持 RDMA 的通信层。与这些存储系统不同的是，Octopus[11] 充分考虑了 RDMA 技术与非易失存储器（NVM，Non-Volatile Memory）的特性，对文件系统内部机制进行了重新设计。其利用 RDMA 绕过传统的文件系统层直接访问共享的内存空间，从而充分发挥出 NVM 的性能。利用 RDMA 的原语来重新设计元数据机制，实现了低延迟的元数据 RPC 和低开销的分布式事务。

虽然利用更加高速的网络来提升分布式存储系统的性能是目前重要的研究路线。

① https://en.wikipedia.org/wiki/IWARP
② https://en.wikipedia.org/wiki/RDMA_over_Converged_Ethernet
③ https://en.wikipedia.org/wiki/ISCSI_Extensions_for_RDMA
④ http://www.mellanox.com/page/microsoft_based_solutions
⑤ https://docs.oracle.com/cd/E36784_01/html/E36825/rfsrefer-154.html
⑥ http://www.accelio.org

然而，随着分布式存储系统功能越来越复杂，其在网络中产生的流量模式也更加复杂。在很多情况下，单纯地提升网络的速度是不够的。一方面，提升网络速度往往会伴随着硬件成本的增加，因而还需要注重提升系统层面的流量传输效率；另一方面，提升网络速度无法优化分布式存储系统中的其它指标，例如安全性等。考虑以下几个问题：（1）随着分布式存储系统对数据存储可靠性要求的增加，在进行数据写入的时候往往需要写入多个副本。因此在网络层会产生复杂的网络流量，而并不只是简单的一对一的流量。如何高效地完成这种复杂模式下的网络流量传输是需要研究的问题。（2）随着分布式存储系统对性能和安全性要求的提升，在实际部署时往往会部署不止一个网络。如何充分地利用现有网络资源以提升存储系统整体性能也是需要研究的问题。（3）随着分布式存储系统规模的增加，系统遭受来自网络或者恶意用户攻击的概率也将增加。如何在尽量少地引入开销的情况下提升分布式存储系统抵御安全威胁的能力也是需要研究的问题。要解决上述问题，需要从另外一个层面去研究分布式存储系统的网络技术，即从网络的控制面去研究如何更加高效地去完成“一对多”模式的网络传输、如何动态地调度网络流量和如何实现更加安全的网络传输等。

为此，在分布式存储系统中，除了需要在网络的数据面进行研究（例如提升网络转发速率、减少延迟等），在网络的控制面上也需要更多的关注（例如实现自动高效的流量调度、提升安全性等）。近年来，一种新型的网络技术——软件定义网络（SDN, Software-Defined Networking）[12] 出现并且得到广泛地研究与应用。SDN 技术将网络的控制面与数据面分离，使得对网络的控制更加灵活高效。因此，SDN 正符合了分布式存储系统对网络的需求。利用 SDN 来优化分布式存储系统成为一个值得研究的方向。

1.2 软件定义网络及其应用研究

移动互联网、云计算和大数据等新技术的兴起与发展给网络技术带来了新的挑战。不断增长的网络规模与多样化的网络应用使得网络变得越来越复杂，逐步弱化了对网络的管理与控制。传统网络已经不适应新的业务需求，主要存在网络复杂、架

构封闭、难以管理、成本高等问题。传统的网络由大量功能单一、厂商绑定的专用硬件设备组成。网络一旦建好则难以变动，无法快速配置资源。网络之间资源难以共享，形成了大量独立封闭的网络和业务“烟囱”。最终导致网络难以支撑新应用、新业务的快速上线，同时也制约了网络技术的创新与发展。

1.2.1 软件定义网络概述

为了解决目前网络体系结构造成的管理复杂、难以创新、资源利用率低等问题，SDN 概念被提出。随着斯坦福大学相关研究者提出 OpenFlow[13] 技术，基于 OpenFlow 的 SDN 架构在业界被推广，得到广泛研究。

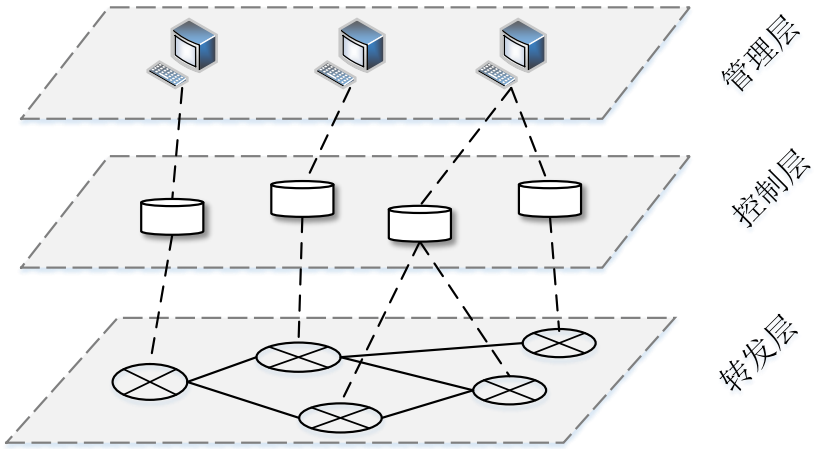


图 1-2 软件定义网络架构示意图

SDN 是一种新兴的网络架构与技术，其将网络的控制逻辑（控制面）与底层网络转发设施（数据面）分离以打破传统网络中的“垂直集成”，提供中心化的网络控制平面，从而使得网络的管理更加简单，具有更加灵活的可编程能力。网络管理员或者用户可以根据需求对网络进行编程，并且可以根据业务的需求灵活地调整网络的功能。对于所有网络设备的管理都可以在一个统一的控制平面上完成，从而极大地简化了网络的管理，进而减少网络的运维成本[14]。SDN 的架构如图1-2所示，主要包括三层，转发层、控制层与管理层。转发层是网络底层的硬件设备，负责网络报文的

转发与策略的实施。控制层负责网络的控制逻辑，生成转发策略并向转发层下发控制策略。管理层允许用户或者管理员定义或者编写自己的网络应用。用户的网络应用通过北向接口（如 REST 接口）下发给控制层，再通过南向接口（如 OpenFlow 协议）下发到网络设备，从而简化网络功能的部署、加速业务的上线周期。

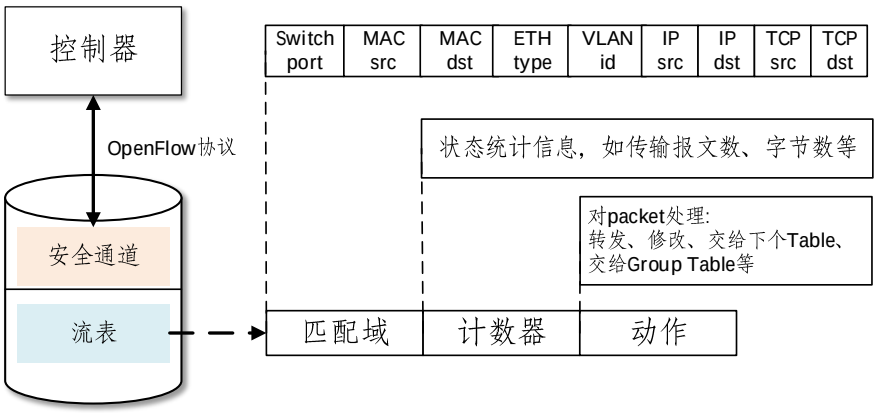


图 1-3 OpenFlow 交换机架构示意图

OpenFlow 最早由斯坦福大学的 Nick McKeown 教授等研究人员在 2008 年提出，其本身是一份设备规范，规定了作为 SDN 转发层基础设施 OpenFlow 交换机的基本组件和功能要求，以及用于由远程控制器对交换机进行控制的通信协议。OpenFlow 交换机的设计思想和整体架构如图1-3所示。其至少由三部分组成：流表（Flow Table），安全通道（Secure Channel）和 OpenFlow 协议（OpenFlow Protocol）。流表是 OpenFlow 交换机进行数据转发的匹配规则，包括匹配域（Match Fields）、计数器（Counters）和动作（Actions）三个字段（此为 OpenFlow1.0 版本，后续版本包括更多内容，如优先级等）。安全通道是 OpenFlow 交换机和控制器的通信通道，采用 TLS（Transport Layer Security，安全传输层协议）技术。OpenFlow 协议是用来描述控制器和 OpenFlow 交换机之间交互所用信息的接口标准，其核心是 OpenFlow 协议信息的集合。

1.2.2 软件定义网络相关应用研究

软件定义网络技术出现后，基于 SDN 的网络应用被广泛研究。利用 SDN 技术，在传统网络中无法实现或者难以实现的功能将变得可能或者更加简单高效。数据中

心是 SDN 应用最为广泛的场景[15]。本章主要考虑数据中心环境下，基于 SDN 在流量工程与安全性两方面应用的研究。

（1）流量工程

通过利用 SDN 的全局视图与可编程能力来实现高效的流量工程是最为广泛的研究方向之一。大量基于 SDN 的流量工程的研究被提出，这些研究的主要目标是通过调度流量来提升网络的利用率、降低网络能耗、保证网络流量的负载均衡和其它目标的功能实现与优化。ElasticTree[16] 提出一种数据中心内动态的能量管理器，通过动态地调节网络的活动情况，在保证网络需求的前提下最小化能量消耗。其基本原理是通过动态地调度网络流量，在满足流量需求的情况下，使用最少的交换机与链路参与转发所有流量，从而将空闲的交换机设备关闭来达到节能的目的。Hedera[17] 提出一种数据中心内的动态流量调度方案，通过周期性地监控与调整网络中每个大流的路由来避免或者降低大流之间的冲突，实现负载均衡，提升网络利用率。Hedera 利用 SDN 的全局视图，采用集中化的算法：全局最先匹配（Global First Fit）和模拟退火算法（Simulated Annealing），来计算每个大流的放置，从而提升网络的整体性能。MicroTE[18] 针对网络负载的特征，通过短期和局部的预测来使得网络调度适应负载的特点，从而提升流量调度的性能。文献[19]提出一种基于 OpenFlow 的在线服务请求负载均衡方法。与传统的基于服务器的请求负载均衡方案不同，其通过设计 OpenFlow 交换机上的流表来实现不同服务器节点之间的负载均衡，从而避免了传统方案的单点故障与瓶颈问题。Plug-n-Server[20] 针对类似于企业、校园网络的非结构化网络拓扑环境，提供 Web 服务请求的负载均衡，同时减少请求的时延。zUpdate[21] 利用 SDN 的中心化调度能力来实现数据中心内，在不影响性能情况下的设备更新。Google 与 Microsoft 公司都使用了 SDN 来实现它们在全球各个数据中心之间网络的流量工程。Google 的 B4[22] 网络使用基于 OpenFlow 的 SDN 技术实现广域网数据中心之间网络的流量工程和实时管控，把数据中心间的核心网络带宽利用率提高到了接近 100%。Microsoft 的 SWAN[23] 同样使用 SDN 来对数据中心之间的流量进行实时调控与重配置来提升网络资源的利用率，进而满足应用的性能需求。

（2）安全性

SDN 可以很方便地采集网络中的统计信息，并且允许用户或者管理员对网络设

备进行编程，因此可以很方便地用于提升网络的安全性。大量的研究专注于利用 SDN 来提升网络或者系统的安全性，主要包括安全策略的实施（访问控制、防火墙等）、Dos 攻击检测与缓解、入侵与异常检测等。

文献[24]提出一种基于 SDN 的弹性 IP 和安全组服务方案，通过控制 SDN 交换机上的流表来实现云环境下的防火墙。LiveSec[25] 提出一种可扩展、灵活的安全管理架构，可以实现针对大规模网络的全面安全保护。LiveSec 在网络中增加一个 Access-Switching 层（OpenFlow 控制层），实现以下功能：（1）网络租户之间端对端的细粒度交互式地安全策略；（2）分布式的负载均衡，支持安全组件的增量部署；（3）应用感知的网络可视化功能。OpenFlow 控制层由支持 OpenFlow 协议的虚拟交换机（OpenvSwitch）和无线路由器构成。在 LiveSec 架构下，可以实现网络边缘与内网全网的点到点的安全控制。

文献[26]提出一种基于 OpenFlow/NOX 的轻量化分布式拒绝服务攻击（DDos）的检测方法，具有比传统检测方法更小的开销。控制器可以直接高效地从（多个）OpenFlow 交换机上通过获取流表来获得报文特征，并且由智能的攻击检测机制进行处理。在检测到攻击后，可以立即下发相应的流表以及时处理该攻击。文献[27]通过利用 SDN 来优化传统远程触发黑洞（Remote Triggered Black Hole，RTBH）。RTBH 通过边缘路由器（Trigger Router）[28] 来将攻击流打上标记，将其引导到空的接口丢弃，从而缓解恶意流的攻击。但是 RTBH 只能丢弃报文而不能分析报文，可能会导致正常报文被丢弃。通过利用 SDN 可以实现动态的 RTBH 配置，并且实现流级（per-flow level）的匹配。

文献[29]提出一种基于 OpenFlow 的随机主机变化（OpenFlow Random Host Mutation，OF-RHM）方案来使得主机的 IP 可变，从而提升主机的安全性。OF-RHM 为主机周期性地赋予一个短期的虚拟地址 vIP（从空闲的地址空间中选取）。通信过程中在交换机中使用 vIP 进行转发同时保证报文正确地到达目标主机，从而扰乱攻击者的目标。CloudWatcher[30] 利用 SDN 将网络中的流路径进行监控与调度，使得流能够经过安全设备，从而实现细粒度的安全检查与分析。FlowNAC[31] 基于 SDN 实现细粒度的网络访问控制。

1.3 基于 SDN 的存储技术相关研究

随着存储设备性能的不断提升，网络逐渐成为存储路径中的瓶颈。同时，在云环境下，存储网络与应用网络融合已经成为未来发展的趋势。如何保证存储网络的带宽、时延、扩展性与安全性等已经成为存储面临的问题。SDN 技术的发展为提升存储网络效率与安全性等方面带来了新思路，因此基于 SDN 的存储技术也成为工业界与学术界研究的新领域与方向。

在工业界，大量企业通过利用 SDN 技术来提升其存储产品的能力。Coho Data[①]公司在其存储产品 DataStream 中使用到了 SDN 技术。SDN 的思想是把控制层和管理层从传统网络的数据层中抽象出来，从而使动态的远程编程控制成为可能。Coho Data 在设计 DataStream 时也遵循了这一抽象。在管理层，Coho Data 在主交换机上运行一个管理模块（SSAPP）来提供管理服务。在控制层，Coho Data 设计了 OpenFlow 控制器，利用分布式的协作服务在所有的存储节点上选出适当的节点来运行该控制器，智能地控制每个 SDN 交换机的路由。在数据层，每个存储节点跟控制层交互，智能地处理数据分布。该公司的产品主要技术发表为论文 Strata[32] 和 Mirador[33]。Strata 利用 SDN 来实现存储协议的虚拟化，从而在保留标准协议（如 NFS 协议）的前提下实现后端存储系统的弹性扩展。基于 OpenFlow 的 SDN 交换机动态地将前端请求重定向到相应的后端存储节点上。因此，Strata 可以对多个客户端提供一个单一的 IP 地址，同时实现存储系统的线性扩展。Mirador 进一步地公开了 Coho Data 的动态存储放置服务，根据不同的指标生成相应的策略，同时可以动态地迁移数据与客户端的网络连接。Jeda Network 公司[②]将 SDN 技术应用到存储网络中，推出 SDSN（Software Defined Storage Networks）。SDSN 提供一种基于软件来统一管理融合存储网络的解决方案。SDSN 不在 FCoE 交换机中使用控制平面，而将控制平面提取到虚拟机中，称为 FNC（Fabric Network Controller），并且可以驻留在任何虚拟化服务器上。SDSN 通过将 SDN 技术引入到存储网络中，可以极大地减少存储网络的投资成本与运营成本，同时也增加了存储网络的敏捷性与可扩展性。富士通[③]同样利用 SDN 技术来控

① http://www.cohodata.com/

② http://www.jedanetworks.com/resources/white-papers

③ http://www.fujitsu.com/global/about/resources/news/press-releases/2013/1209-01.html

制其存储流量的路由转发，以提升 LAN 和 SAN 融合网络的吞吐率。其指出在融合网络中，局域网（LAN, Local Area Network）流量可以通过以太网地址或者 IP 地址被 SDN 控制，而存储流量由于没有使用 IP 地址并不能被 SDN 控制。为此富士通实验室开发了在 FCoE 上利用 SDN 来对存储访问流量进行监测并控制的技术[34]，在以太网交换机中实现存储流量检测和管理（数据包转换和转发）功能。Plexxi 公司与存储供应商 SolidFire[35] 联合将各自的产品“关联驱动的网络”与“云设施中的全固态存储系统”结合起来，构建了一个网络可编程的 QoS 保障的存储解决方案。通过带宽保证、网络与存储设施之间更加紧密的访问接口来达到最大化的应用性能，从而提高网络设施的效率。其最终构成基于 SDN 的具有 QoS 保障的存储方案。综上所述，目前有很多企业已经在将 SDN 技术应用到存储中去。其中 Coho Data 最为典型，其利用 SDN 对网络流的控制能力，进行存储流量的精心控制，对客户端发起的存储请求进行重定向，以达到存储系统的负载均衡与线性扩展，同时对客户端透明。Jeda 与富士通都是针对融合网络中如何对基于 FCoE 的存储流量进行控制。Jeda 建构了一个专门用于控制 FCoE 的流量网络控制器，自动化地管理及部署存储。富士通在以太网交换机上实现了存储流量的检测与管理，合理的控制存储流以最小化网络拥塞。Plexxi&SolidFire 专注于利用 SDN 技术构建端到端 QoS 保证的存储。

在学术界，大量的研究将 SDN 应用到分布式存储系统中，从而优化存储系统的性能、扩展性等。Mayflower[36] 通过将 SDN 网络控制器与存储系统协同设计来使得存储系统的请求可以感知网络，从而可以根据网络情况进行智能地副本选取。与传统的分布式存储系统的架构相比，Mayflower 新增了一个在 SDN 控制器上的流管理器。流管理器可以智能地根据文件系统与网络的状态信息进行副本选取与流量调度，进而实现系统的全局优化。MetaFlow[37] 利用 SDN 将元数据的查询请求分发到相应的元数据服务器上，从而提升存储系统中的元数据查询性能。与传统的方法相比，MetaFlow 不需要额外的定位元数据所在服务器的步骤，允许客户端直接使用元数据服务器的 ID 与元数据服务器进行连接，并进行 I/O 操作，因此具有更小的时延。Smarteye[38,39] 提出一种基于 SDN 的网内重复数据删除方法。利用 SDN 的控制转发分离与可编程的能力，在 SDN 控制器上进行数据指纹检测。为了减少 SDN 控制器和交换机之间的通信开销，利用局部性原理在交换机上缓存文件信息进行初步的冗

余消除。通过将数据的重删过程放置到网络内部（in-network），既可以减少源端去重方案中的消息通信时延，也可以降低目的端去重方案中的网络传输与存储开销，因此是源端与目的端去重方案的折衷。

专利[40]提出在 SAN 存储系统中，计算节点与存储设备之间存在多条路径时利用 SDN 来选取最优的传输路径。专利[41]提出一种基于 SDN 的协议无关的存储访问方法。用户向存储系统发送请求时，请求的初始报文将会由 SDN 交换机打上协议标记转发到 SDN 控制器。SDN 控制器会根据协议标记向存储应用发送相应的初始报文。在具有众多存储应用并且各个存储应用使用不同协议的情况下，可以实现用户协议无关地访问到相应的存储。专利[42]通过利用 SDN 来优化对象存储系统（例如 Ceph）的读/写性能。SDN 控制器可以根据设置的 QoS 策略实时地生成最佳的路由，例如对 QoS 高的流分配更高的优先级等，从而提升系统的性能。

综上所述，利用 SDN 技术可以更加高效地实现存储系统中的数据传输，减少网络拥塞，提升元数据查询的效率等。除了现有的研究之外，利用 SDN 来优化分布式存储系统依然存在巨大的研究空间。分布式存储系统在数据的传输效率、网络资源的利用率和安全性方面都存在较大的问题。例如，分布式存储系统中还具有以下问题：（1）分布式存储系统中存在大量的“一对多”模式的流量，目前传输这些流量的方法存在较大的开销；（2）分布式存储系统经常部署两个独立的网络，两个网络之间不能共享资源从而造成较大的资源浪费；（3）分布式存储系统的安全性问题突出，无法抵御来自网络和恶意用户的攻击。本文受 SDN 在流量工程与安全性方面应用的启发，创造性地将 SDN 技术应用在分布式存储系统中，以更好地优化存储系统的性能与安全性。

1.4 本文主要研究内容

随着分布式存储系统中存储介质性能的不断提升，以及系统规模的不断扩张，网络将会逐渐成为制约系统性能提升的瓶颈。为了减少成本，数据中心的存储呈现向超融合的趋势发展。因此，存储与计算的网络将会融合在一起，如何提升存储数据在网络中的传输效率、减少网络拥塞，进而提升存储系统的整体性能变得非常重要。另

一方面，随着分布式存储系统规模的增长，存储系统的节点与承载该存储系统的网络拓扑的规模都将增加。这将伴随着更多的恶意用户通过入侵存储系统中的部分节点或者网络节点来攻击存储系统或者窃取数据的风险。如何提升存储系统的安全性也将是需要研究的重要课题。软件定义网络的出现使得网络的控制与转发更加灵活。用户可以通过在一个网络控制器上编程来控制每个流的的转发，从而可以实现的灵活流量调度与报文转发。利用软件定义网络可以使得分布式存储系统中的流量传输更加高效与灵活。因此，本文主要研究利用 SDN 实现高效、安全的网络应用，从而提升分布式存储系统中的流量传输效率与安全性。

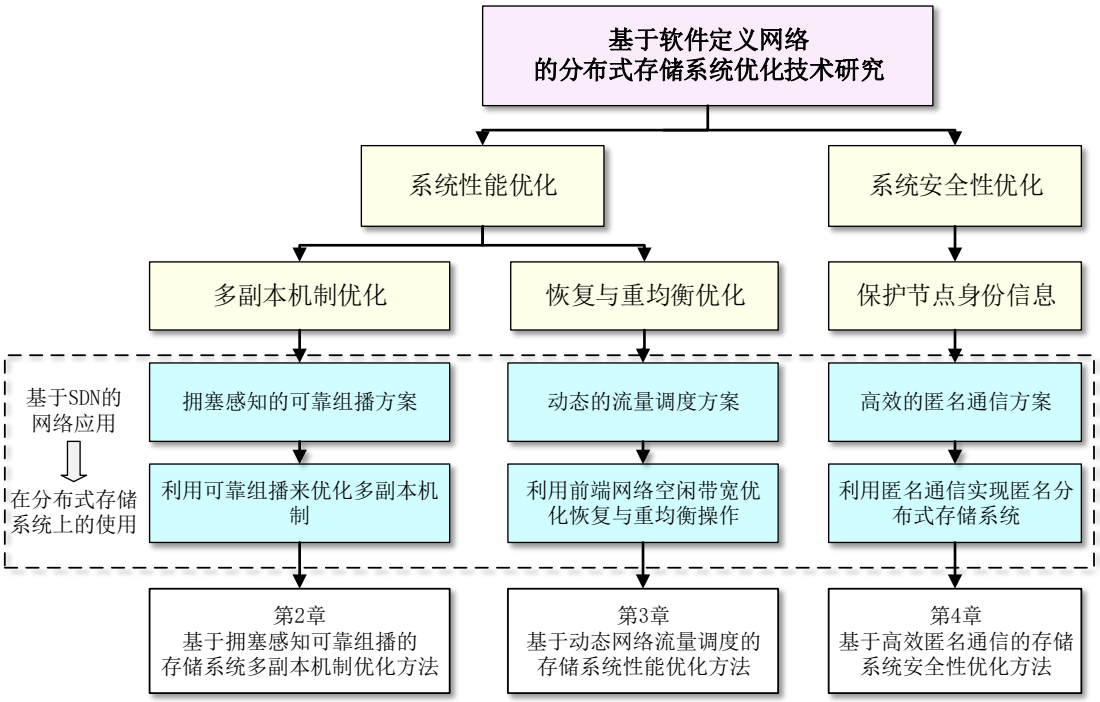


图 1-4 全文研究内容和组织结构图

图1-4给出了本文的论文组织结构，本文主要针对分布式存储系统中的多副本写操作、恢复与重均衡操作、安全性三个方面在网络层进行优化开展相关研究。具体的研究内容包括以下三个方面：

（1）基于拥塞感知可靠组播的存储系统多副本机制优化方法

分布式存储系统通常采用多副本机制来提升数据存储的可靠性。在进行数据多

副本写操作的过程中，客户端需要将一份数据同时写入到多个存储节点上，因此在网络中产生一对多模式的数据传输。现有的多副本机制中，都是采用基于单播的方式实现，即通过多次的一对一的数据传输完成。这个过程中将会产生大量的网络冗余数据传输，从而降低多副本写入的性能。针对这一问题，本文提出一种基于组播的多副本机制，通过使用可靠的组播协议来完成数据从客户端到多个存储节点的写入操作。然而，现有的组播协议在易用性与性能方面都难以满足需求。因此，本文提出基于 SDN 的拥塞感知组播方案 MCTCP，并利用其在分布式存储系统 HDFS 上实现组播多副本机制。MCTCP 的主要思想是通过扩展 TCP 协议来实现可靠的组播数据传输，并利用 SDN 来根据网络链路的状态实时地计算与调整组播转发树（Multicast Spanning Tree, MST）以绕开拥塞与失效的链路，从而达到更好的传输性能与鲁棒性。MCTCP 充分考虑了分布式存储系统多副本场景的特点，并在主机端处理与转发路由两方面都进行了性能优化，从而比传统可靠组播方案更加适合用于实现组播多副本机制。

（2）基于动态网络流量调度的存储系统性能优化方法

现有很多分布式存储系统通常部署两个网络，包括前端网络与后端网络，来提升系统的性能，例如 Ceph 和 Swift。用户的请求流量将使用前端网络，而系统的后台流量（如恢复、重均衡）将使用后端网络。然而，由于两个网络在部署上是完全隔离的，当有一个网络存在空闲带宽时，另一个网络的流量无法利用空闲带宽来提升性能，造成网络资源的浪费。针对这一问题，本文提出一种高效的动态流量调度方案 MAX，可以自动地发现存储系统中空闲的网络带宽，实现跨前端与后端网络间的网络流量调度。为了提升 MAX 的易用性，在网络层进行流量调度，而不需要修改存储系统的代码实现。MAX 在各个存储节点上加入一个网络调度层，使得所有存储系统的流量都将先进入到该调度层，经过 MAX 调度之后再发往相应的物理网卡，从而实现跨前端与后端网络间的流量调度。利用 SDN 的全局视图与可编程能力，可以实时地发现网络中的空闲带宽，并进行集中化的计算，提高流量调度的效率。通过将部分后端网络中的流量（例如恢复与重均衡操作）调度到前端网络中以利用其空闲带宽，可以有效地提升恢复与重均衡操作的性能。通过采用基于优先级的路由方案，可以保证调度到前端网络的流量不影响到前端用户请求的性能。

（3）基于高效匿名通信的存储系统安全性优化方法

随着系统规模的增加，分布式存储系统遭受来自网络与恶意用户的安全威胁也将增加。恶意用户可以通过控制分布式存储系统中某个节点（例如通过伪装成合法用户使用客户端节点），之后通过分析来确定数据所在节点的位置，然后进一步发动其他攻击来窃取数据或者破坏系统。为了抵御这类攻击，本文提出基于匿名通信的分布式存储系统（简称为匿名存储系统），通过匿名通信方案来实现存储节点之间相互匿名，提升系统的安全性。然而，现有的匿名通信方案开销巨大，无法满足性能需求。因此，本文提出一种基于全局路由冲突避免机制的匿名通信方案 MIC。其主要思想是通过在交换机上修改报文头部信息来隐藏报文的发送者与接收者信息，从而达到匿名通信的目的。相比于传统的基于覆盖网络（overlay）的方案，MIC 基于网内（in-nework）的设计具有更短的传输路径和中间操作，因此具有更小的开销，达到更高的性能。为了保证网络通信的正确性、避免路由冲突，MIC 设计了全局路由冲突避免机制。通过采用多个子流和局部多播机制在不修改交换机实现的前提下提升抗流量分析攻击的能力。由于 MIC 的性能开销极小，从而使得分布式存储系统可以在几乎不损失性能的前提下实现节点间的匿名通信，提升系统整体安全性。

1.5 论文组织结构

本文基于 SDN 对分布式存储系统中网络层进行了优化，主要包括性能优化与安全性优化两方面。第二、三章介绍了针对存储系统性能优化的两个研究工作，第四介绍了针对存储系统安全性优化的研究工作。具体的组织结构如下：

第一章首先介绍了基于对象的分布式文件系统及其网络技术研究；然后介绍了软件定义网络的相关概念与基于 SDN 的相关网络应用研究；之后介绍了基于 SDN 的存储技术相关研究；最后从全文的角度详述了本文的主要研究内容以及组织结构。

第二章首先介绍了拥塞感知的可靠组播的研究动机；然后详细介绍了拥塞感知的可靠组播方案 MCTCP 的设计与实现，以及基于 MCTCP 在 HDFS 上实现组播多副本机制的相关细节；最后分别对拥塞感知的组播方案 MCTCP 以及组播多副本机制进行了性能评估。

第三章首先介绍了跨双网络动态流量调度的研究动机；然后详细介绍了动态网络流量调度方案 MAX 的设计与实现，结合实例 Ceph 分布式存储系统进行了应用场景的分析说明；最后在 Ceph 上进行了 MAX 的性能评估。

第四章首先介绍了数据中心内高效匿名通信需求与匿名通信的相关研究；然后介绍了高效的匿名通信方案 MIC 的问题界定、设计与实现，并对 MIC 进行了安全讨论；再然后详细讨论了 MIC 在分布式存储系统上的应用；最后分别对 MIC 与匿名存储系统 CapFS-M 进行了性能评估。

第五章对全文的主要内容进行总结，指出了各个部分的创新点，然后对未来研究工作提出了展望分析。

2 基于拥塞感知可靠组播的存储系统多副本机制优化方法

分布式存储系统通常采用多副本机制来保证数据的可靠性和可用性，即一份数据将会被同时保存在多个存储节点上，从而保证在部分存储节点失效时，数据依然不会丢失。在多副本的写入过程中将会产生“一对多”模式的数据传输，从而在网络中产生大量的报文。然而传统的多副本机制往往都是采用多次单播的方式实现。对于N 个副本的写入将会产生 N 个点对点的数据传输，从而将会产生大量的网络冗余报文，增加网络资源的占用，造成网络拥塞，降低存储系统的性能。针对这个问题，本章提出基于组播的多副本机制，通过采用组播传输的方式来完成多副本数据的写入。然而现有的可靠组播传输方案在易用性与性能方面都难以满足需求。因此，本章提出基于 SDN 的拥塞感知的可靠组播方案（MCTCP，Multicast TCP），并利用其实现基于组播的多副本机制。MCTCP 的主要思想是通过扩展 TCP 协议来实现组播传输的可靠性，并利用 SDN 来根据网络状态实时调整组播的转发树来绕开拥塞和失效链路，从而提升传输性能与鲁棒性。

2.1 拥塞感知的可靠组播的研究动机

近年来，随着云计算技术的发展，数据中心内的应用得到极大地丰富，其中大量不同类型的分布式应用在网络中产生复杂的“一对一”和“一对多”模式的通信流量，对网络资源造成极大的冲击。更为严重的是，数据中心内部大量的一对多模式的网络流量依然采用低效的多次单播方式，在网络中产生大量的重复流量，造成网络资源的浪费，严重影响应用的性能。数据中心内存在大量典型的一对多组通信场景，很多分布式系统在运行过程中需要将相同的一份数据传输到多个远程节点上。分布式文件系统中采取多副本的机制以保障数据存储的可靠性，如大数据处理系统中的HDFS[7]、Google 公司的 GFS[6] 以及 Redhat 的 Ceph[8] 等。文件被切分为块，每个块按照一定的放置策略写入到多入存储节点上。大数据处理系统中将可执行的程序或

者共享数据分发到其他协作服务器上，如 Hadoop MapReduce 的 DistributedCache[①]和 Apache Spark 的 Broadcast variable[②]。搜索引擎将搜索的请求分发到多台索引服务器以提升查询速度，如 Google 和 Bing。

这些组通信场景的以下特点为组播传输方案的设计提出了新的需求：**第一**，组播会话多为小组模式。分布式存储系统中一般采用 3 个副本，数据分析类应用中工作节点的数目一般为几十到几百个内[43]。**第二**，可靠性要求高。分布式存储系统中要求写入到所有节点上的数据都一致，不允许传输过程中数据丢失。**第三**，发送端发起传输。分布式存储系统中的传输是由客户端主动向存储节点中推送数据，接收端不能提前知道哪个客户端节点，在何时会向其发送数据。**第四**，高效率。分布式存储系统往往都部署在同一个数据中心内，因此其网络基础设施具有高带宽、低时延的特性。这就要求组播传输协议能够充分利用硬件的能力，发挥最高的性能。另一方面，数据中心内的网络流量呈现突发与不均衡的特性[44]。组播协议需要能够具有拥塞感知的能力，从而达到高效的路由转发。**第五**，鲁棒性。组播传输的过程中会创建组播转发树 MST（Multicast Spanning Tree），转发树中任何一条链路或者交换机失效都将会影响到传输性能。因此需要具有在链路失效时实时调整到正常链路的能力。

然而传统的可靠组播方案并不适应于上述场景，很难高效地传输上述场景中的一对多模式的流量，主要有以下几方面的原因。**第一**，大部分传统的可靠组播方案是接收端发起的，并且是在应用层上实现（基于 UDP 协议实现），因此并不适应分布式存储系统多副本的场景，并且具有比较大的性能开销。接收方发起模式中每个接收端需要提前获取想要订阅的组播地址，自由地加入和退出该组，典型的有 PGM[45,46]、ARM[47]、NORM[48]、TCP-SMO[49]、SRM[50]、RMTP[51]、TMTP[52]、RDCM[53] 等。这种模式更加适应于大组场景，发送端可以不需要知道接收端信息，实现组播带宽不随着接收端数目增加而减少。但是这种模式的组播设计不适合动态的小组场景，在该场景下，接收端难以在接收组播数据之前获取正确的组播地址。同时采用接收端自由加入组的方式，接收端在不同的时刻加入，难以保证组播传输对所有接收端完全可靠。**第二**，传统 IP 组播路由算法，例如 PIM-SM[54] 等，并不是为了最优路由转

① https://hadoop.apache.org
② http://spark.apache.org

发而设计的。它们并不能感知网络拥塞，从而在数据中心内突发与不可预测的网络环境[44]中容易造成性能下降。**第三**，传统组播管理协议，例如 IGMP[55] 等，并不能感知链路失效。在链路失效后无法及时地调整路由转发树，因此将会导致传输卡顿，性能下降以及上层应用中断。

软件定义网络 SDN[13] 技术的出现与发展，为解决组播协议的路由效率问题带来了新的思路。SDN 架构将网络的控制平面与数据平面分离，简化了网络的配置模式，增加了网络控制权的开放性。其在控制器端具有全局的网络视图，可以实时地监控网络状态，支持集中式的最优路由选择，这为组播的最优路径转发提供了绝佳的条件。利用 SDN 的全局视图能力，组播路由算法可以根据实时的网络拥塞状态来计算出（近似）最优的组播路由转发树，并且实时地对链路失效做出反应。SDN 技术的不断发展，使得其成为未来网络发展的趋势，其在数据中心内的部署将会变得越来越普遍，基于 SDN 架构的组播技术也将随着 SDN 技术的发展而不断得到应用。

利用 SDN 技术，可以获取网络的全局视图，使得组播转发树的生成变得更加高效，同时利用 SDN 的全局控制能力，可以很容易地满足组播协议的安全性。Avalanche[56] 与 OFM[57] 提出基于 SDN 的组播系统，用 SDN 控制器进行组播管理与路由，可以提升路由效率与组播的安全控制能力。类似地，CastFlow[58] 通过提前计算好发送端与接收端集合之间的所有可能的路由来加速组播事件的处理。Ge 等[59] 提出利用基于 OpenFlow 的动态组播生成树算法来提升转发性能。Shen 等[60] 提出一种近似算法，称为 RAERA （Recover Aware Edge Reduction Algorithm），来实现一种新的可靠组播树恢复感知的斯坦纳树（Recover-aware Steiner Tree, RST）。然而，现有的基于 SDN 的组播研究都只是针对组播路由，而并没考虑到传输协议。同时，它们也都不具有拥塞感知的能力，难以满足数据中心组播传输的性能需求。

为了满足数据中心内对组播传输的需求，本章提出 MCTCP，一种基于 SDN 架构的可靠、高效率、拥塞感知和高鲁棒性的组播方案。其主要思想是采用中心化的方法管理每个组播组的路由，并且根据网络中链路的利用率动态地调整组播的转发树以提升传输效率。为了减少组播协议在主机端的处理开销，MCTCP 扩展 TCP 协议以支持组播传输，保证报文传输的可靠性。与传统的基于 UDP 的可靠组播方案（应用层）相比，MCTCP 在传输层实现报文处理，因此具有更高的性能。MCTCP 针对数

据中心内分布式存储系统多副本操作的特性来设计，因而可以很方便地应用到分布式存储系统中。本章设计了基于组播的分布式存储系统多副本机制，并在 HDFS 上实现了原型系统。

2.2 拥塞感知的可靠组播方案

2.2.1 总体架构

拥塞感知的可靠组播方案 MCTCP 是一个单源可靠组播传输解决方案，包括两个模块：主机端处理模块（HSP, Host-Side Protocol）和组播管理模块（MGM, Multicast Group Manager）。主机端处理模块基于 TCP 协议之上作组播功能的扩展，利用 TCP 的连接机制、数据确认机制、数据重传机制和拥塞控制机制，实现“一对多”组播数据的可靠传输。组播管理模块负责 MCTCP 会话的组播转发树的生成与管理，在 MCTCP 建立连接的过程中生成合理的转发树，并在链路失效的时候更新相关的转发树。为使组播转发的过程中充分利用网络资源，最大化地避开网络拥塞，SDN 控制器实时地监控当前网络状况，在生成转发树的过程中加入对链路拥塞状况的考虑。

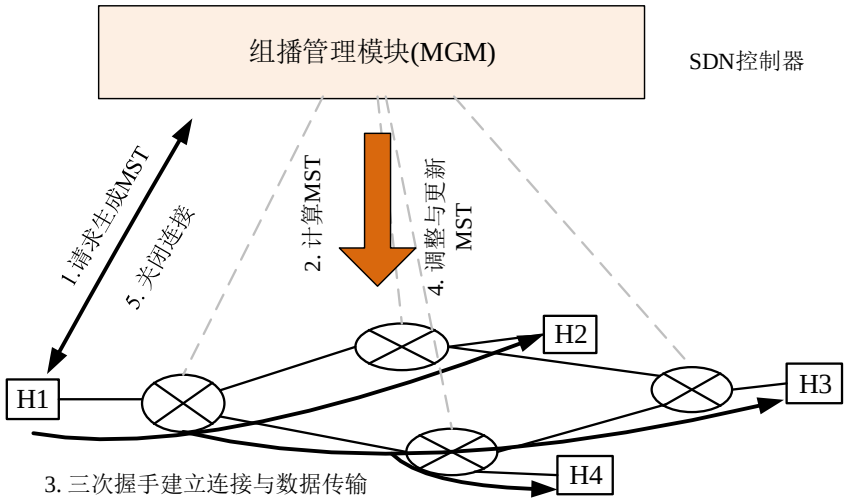


图 2-1 MCTCP 工作原理示意图

MCTCP 的工作原理示意图如图2-1所示，数据发送端显示地与多个接收端建立连接。第一，发送端向 MGM 模块发送请求以进行组播转发树（MST）的计算；第二，

MGM 模块计算 MST，并将其安装到相应的交换机上；第三，发送端开始进行三次握手，与多个接收端建立连接，并且在连接建立之后开始传输数据；第四，MGM 模块在检测到链路拥塞或者失效后自动更新相应的 MST；第五，在数据传输结束后，发送端通知 MGM 模块以结束会话连接。

2.2.2 主机端处理模块

由于 TCP 已经历经了四十多年的发展，各项机制已经进化得相当成熟，包括数据可靠性保障、拥塞控制和性能等方面。直接利用 TCP 的成熟的机制来扩展组播功能是设计 MCTCP 主机端处理模块的基本思路。主机端处理模块主要在 TCP 协议上进行组播功能的扩展。与 TCP 协议相似，发送源在发起连接之前先获取接收端的地址与端口号，然后与多个接收端建立连接。MCTCP 为非对称的单向传输协议，只允许发送端发起连接、传输数据和关闭连接。

连接建立过程

MCTCP 在建立组播会话时需要由组播管理模块生成最优的组播转发树，因此每个会话建立时需要通知组播管理模块。由于接收端并不提前获得组播地址，在进行三次握手的过程中，第一次握手需要通过接收端各自的单播地址进行通信。MCTCP 将组播地址通过 SYN 报文捎带给接收端。接收端接收到 SYN 报文之后，将组播地址加入本地的接收列表，从而可以接收该组报文。

依据通知组播管理模块的时机，提出两种方案：带外方案（握手之前通知）和带内方案（握手过程中通知）。带外方案是在进行握手之前，发送端向控制器发送请求，控制器在生成本次组播会话的转发树之后通知发送端开始握手过程。带内方案是将三次握手过程中的第一次握手报文 SYN 复用为通知报文。控制器在生成组播转发树之后，再分别将 SYN 报文以单播的方式发送到各个接收端。带外方案的优势是可以减少控制器的开销，但是每次连接时需要额外地与控制器进行一次通信，会带来额外的时延开销。这种方案适用于建立连接不频繁，而传输数据量大的场景。带内方案的优势是可以减少建立连接过程的时延，但是对于每个会话，控制器需要分发 SYN 报文到各个接收端，会给控制器带来较大的开销。这种方案适用于组成员数少（比如小于 10 个），对连接时延要求较高的场景。图2-2显示了连接建立的过程。

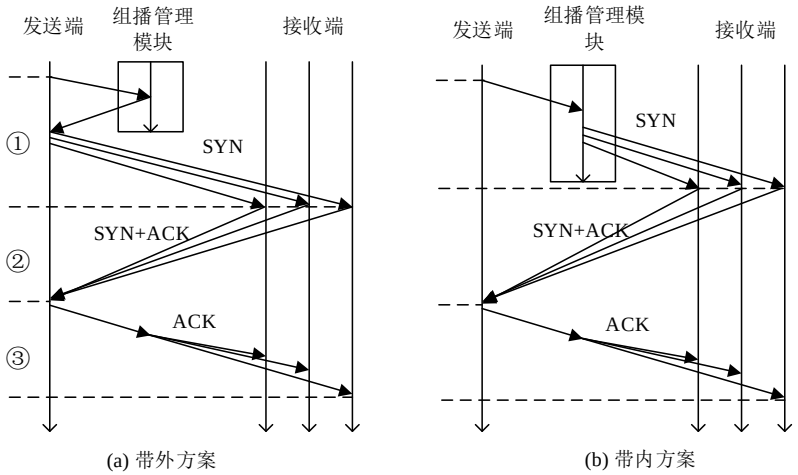


图 2-2 MCTCP 会话建立过程示意图（三个接收端）

数据传输过程

数据传输的过程中需要处理数据报文的确认、重传、拥塞控制以及节点失效。

（1）报文确认的处理

与 TCP 协议类似，MCTCP 在建立好连接之后，允许发送源向该连接发送数据。基于 TCP 的可靠数据保障机制，MCTCP 在发送源上需要维护滑动窗口，同时处理所有接收端的确认。接收端在接收到数据报文之后以累积确认的方式给发送源返回确认信息。发送源滑动窗口的调整受限于速度最慢的接收端，只有在接收到所有接收端的确认之后，才能相应地调整滑动窗口。由于 MCTCP 主要针对动态小组模式的传输场景，传统可靠组播方案中面临的确认风暴（ACK-implosion）问题可以忽略。同时先前已有研究证明了在普通的主机与 1Gbps 通用的网络环境下，完全可以容纳接收端数目小于 1000 规模的确认报文[49]。考虑到很多应用要求低时延，传统的为避免确认风暴而采用的分级确认方案并不适用，而这种简单直接的方式反而显得更加有效。

（2）数据重传的处理

若检测到报文丢失或者出现超时，则以组播的方式重传该报文。虽然以组播方式重传的方式在只有少数接收端丢包时会造成一定的资源浪费，但是 MCTCP 针对的主要是小组场景，复杂的重传机制带来的开销将会比重传带来的浪费更为昂贵。另外，MCTCP 通过高效的转发机制，最大化地避免网络拥塞，可以极大地减少网络丢包。

（3）拥塞控制方法

数据中心内存在各种交错复杂的网络流量，其中大部分流量都是由 TCP 协议承载的，因此当一种新的协议加入到这种环境时，是否 TCP 友好成为一种很重要的指标。很多传统的可靠组播方案没有拥塞机制，或者自己设计专门的拥塞算法。然而与成熟的 TCP 拥塞机制相比不管在性能还是在多种场景下的适用性上都稍显不足。MCTCP 可以直接使用 TCP 已有的所有拥塞控制算法，以最拥塞的接收端来决定组播会话的拥塞程度。直接使用 TCP 的拥塞机制还有一个好处，即 MCTCP 可以直接使用针对 TCP 拥塞进行研究的成果，不断地更新演进。

（4）节点失效处理

在数据发送过程中，若某个或者几个接收端在一定的阈值时间内未返回确认，则认为该接收端已经失效，并将其从组播会话中清理出去。出现这种情况很可能是接收端节点在运行过程中出现宕机或者网络故障。为了保证其它正常的节点能够继续接收数据，只能将故障节点从会话中清除，然后由上层应用程序来进行故障恢复。如分布式存储系统在进行三副本数据写入的过程中，出现某一个存储节点宕机，那么只能先完成两个正常节点的写入操作，然后再由分布式存储系统以自己定义的策略恢复或者重新选择节点进行数据写入。MCTCP 具有很好的灵活性，若在数据传输的过程中出现节点宕机，可以很容易地选取新的节点，重新建立连接进行数据传输。

在实现过程中，MCTCP 被实现为一种新的传输层协议（具有一个新的协议号），从而避免了直接在内核代码中进行修改。MCTCP 提供与 TCP 协议相同的语义，接收端需要提前进入监听状态，发送源在建立连接之前已经获得了各个接收端的 IP 地址及端口号信息。

（1）报文格式

为了简化系统实现的复杂度以及保证协议的兼容性，MCTCP 的报文格式与 TCP 的完全一致，采用在 TCP 报文选项字段里添加 MCTCP 相关的特性。MCTCP 中的选项字段格式如图2-3所示，其中 Kind 为选项类型，即为定义的 $TCPOPT\_MCTCP$，$length$ 为该选项长度，sub_kind 为 MCTCP 选项下的子选项，包括 $OPTION\_MCTCP\_XID$、$OPTION\_MCTCP\_MCADDR$、$OPTION\_MCTCP\_STATUS$、$OPTION\_MCTCP\_SENDER$ 等，$info$ 为相应子选项的内容。

Kind	Length	Sub_kind	Info

图 2-3 MCTCP 中的选项字段格式

目前 MCTCP 中定义了四种选项：

$OPTION\_MCTCP\_XID$，用于标识一个唯一的组，其长度为 7 字节，包涵 4 字节的 XID。该 XID 由发送源生成，在连接建立的过程中传递给接收端，用于标识该连接会话，同时也是所有接收端的起始序号。

$OPTION\_MCTCP\_MCADDR$，用于传递当前组播地址，其长度为 8 字节，包涵 1 字节的接收端 ID 和 4 字节组播 IP 地址。该选项用于发送源建立连接过程中的 SYN 报文，和接收端向发送源发送的报文中。组播 IP 地址用于标识当前报文属于哪个组。对于发送源的 SYN 报文，接收端 ID 为空；而对于接收端向发送源发送的报文，接收端 ID 用于标识当前报文是由哪个接收端发送，可以快速定位接收端，加速发送源对确认报文的处理。

$OPTION\_MCTCP\_STATUS$，用于标识当前报文是连接处理什么状态下发送的，其长度为 4 字节，包涵 1 字节的状态码。其作用是用于区分该报文被发出时连接处于何种状态，目前只使用了 SYN 请求这一种状态。因为 MCTCP 中接收端需要提前区分 SYN 报文与其它报文，SYN 报文的目标地址为接收端地址，而其它报文的目标地址为组播地址。

$OPTION\_MCTCP\_SENDER$，用于标识发送该报文的是否为发送源，其长度为 3 字节，一般可以有与其它选项复用。如果 MCTCP 选项值大于等于 128 则为发送源发送的报文，否则为接收端发送的报文。

（2）接收报文处理

Linux 内核中以 SK 结构（struct sock）唯一表示一个连接。报文从网络中到达主机时，首先需要找到该报文对应的连接。TCP 以（源地址，源端口，目标地址，目标端口）四元组来唯一标识一个连接会话。接收端接收到报文时，通过这四元组来查找对应的表示连接的 SK 结构。对于 MCTCP 来说，因为有上述复杂的发送源与接收端的区分处理，在主机接收报文的处理也需要明确发送源与接收端。在接收端接收到报文

时需要判断是否为 SYN 报文，因为 SYN 报文与其它报文的的目标地址不相同。SYN 报文的目标地址是接收端主机地址，而其它报文的目标地址是组播地址。对于 SYN 报文，直接从 TCP 的全局哈希表（$tcp\_hashinfo$）中查找对应的 SK 结构。而非 SYN 报文则从 MCTCP 专用的全局哈希表（$mctcp\_hash$）中查找对应的 SK 结构。在发送端，接收到来自各个接收端的 ACK 报文时，该报文的源地址为接收端的主机地址，目标地址为发送源的主机地址，而在发送源上表示当前 MCTCP 连接的 SK 结构的目标地址是组播地址。因此不能按照 TCP 的方式直接使用（源地址，源端口，目标地址，目标端口）四元组来查找 SK 结构。在发送源接收到报文时，先从报文的头部获取选项 $OPTION\_MCTCP\_MCADDR$ 字段，以组播地址代替源地址，即（组播地址，源端口，目标地址，目标端口）从 TCP 全局哈希表（$tcp\_hashinfo$）中查找。同时发送源需要处理来自多个接收端的 ACK 报文，通过 $OPTION\_MCTCP\_MCADDR$ 字段中的接收端 ID 来确定当前是哪个接收端的 ACK 报文。

（3）编程接口

MCTCP 采用通用的 socket 编程接口。在进行 MCTCP 编程时，接收端与 TCP 编程相同，只需要调用 listen() 函数进入监听即可。在发送源上，用户可以自己指定一个组播地址用于本次连接的地址，若不指定则 MCTCP 会自动选取一个随机的地址。同时调用 setsockopt() 函数指定接收端地址，如下所示，其中 mc_addr 为接收端地址列表。之后即可调用 connect() 函数建立连接，调用 send() 函数发送数据。

```
struct sockaddr_mc{
    uint16_t sin_port;
    struct in_addr sin_addr;
};
struct sockaddr_mc mc_addr[PEER_NUM];
setsockopt(fd, MC_IPPROTO_TCP, TCP_MCTCP_ADDR, &mc_addr,
  sizeof(mc_addr));
```

2.2.3 组播管理算法

MCTCP 采用集中式的组播管理，由 SDN 控制器完成组播会话的管理与路由管理功能，可以达到更好的灵活性与灵敏度。SDN 控制器具有网络的全局视图，可以实时监控网络状况，从而使得组播转发树可以有效地避开拥塞链路和失效链路。MCTCP 的组播管理模块可以根据链路状况合理地更新当前网络中所有组的转发树，以达到最优的转发效率。组播管理模块包括会话管理、路由管理与链路监控三个子模块，如图2-4所示。

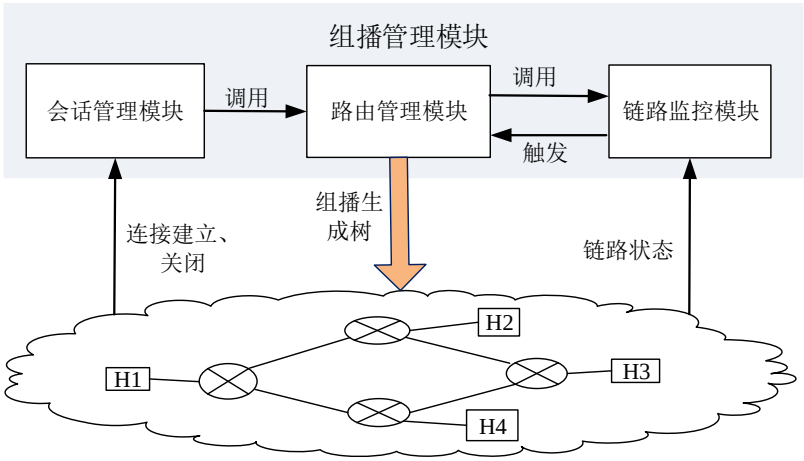


图 2-4 组播管理模块结构图

会话管理模块负责协助组播会话的建立与当前网络所有组播会话状态的维护。当一个组播会话建立或者关闭时，发送端将会通知会话管理模块。因此，会话管理模块可以跟踪维护每个组的状态。会话建立时将该组标记为活跃状态，会话关闭后将该组标记为不活跃状态。对于不活跃状态的会话，可以根据当前交换机的负载情况决定是否删除其对应的组播转发树。在交换机负载较轻时，可以将组播转发树保留一定时间，以便下次相同的会话再次建立时复用。另一方面，会话管理模块定时地清理不活跃状态的会话对应的组播转发树。

2.2.3.1 链路监控模型

链路监控模块负责定时地网络链路状态监控，根据网络的实时负载情况计算每条链路的权值。当有某链路的权值达到设定的阈值时，触发路由管理模块进行路由

更新。链路监控可以采用现有成熟的网络监控技术，如 sFlow、netFlow 等，也可以直接使用 OpenFlow 协议的“Port-Status”接口实现。

本章中采用 OpenFlow 协议的“Port-Status”接口实现，重点在于如何估计每条链路的负载权值。考虑网络链路带宽相同的全双工网络环境，假设链路的单向带宽为 B，链路的单向权值为 $weight$。对任一链路，通过统计获得时间间隔 Δt 内的传输总字节数为 M，则其传输带宽为 $R = M/\Delta t$，其对应的权值为 $W_m = R/B$。当 $W_m < 1$ 时，表明该链路的流量速率未达到链路带宽，则以测量到的传输速率计算该链路的权值，即 $weight = W_m$。当 $W_m = 1$ 时，表明该链路的流量速率达到链路带宽，则当前该链路上所测量到的传输速率并不能代表其真实的负载情况，其实际负载要比测量的负载 W_m 更重。一般而言，在链路上的传输带宽达到链路带宽时，该链路上的流数量越多则相应的负载越重。为此，需要根据链路上流的数目来估计权值。此处引入参数 F，表示一条流独占链路时能够达到的速率值。假设有 n 条流共享某一链路，那么这 n 条流可以产生的流量速率为 $B_n = n \cdot F$，则该链路的权值为 $weight = n \cdot F/B$。在实际的实现过程中，测量的速率 R 可能并不能精确地达到最大容量值 B，而只能达到一个比较接近的值。假设 $R = \alpha \cdot B$，如果 α 大于一个阈值 η，例如 95%，那么则认为流量已经达到链路容量。权值的计算总结为以下公式：

$$weight = \begin{cases} R/B & R < \eta \cdot B \\ max(1, n \cdot F/B) & R \geq \eta \cdot B \end{cases} \quad \text{(式 2.1)}$$

从公式2.1中可以看出，需要测量每条链路的传输速率 R，每条链路上的流的个数 n 以及一条流能够在网络中达到的速率 F。可以通过获取 SDN 交换机中的端口统计信息来计算每个链路的传统速率 R，例如利用 OpenFlow 中的“get_port_stats”功能。为了计算每条链路上的流的数目 n，可以在控制器上记录每个已经生成的流表，并且通过启用 “Flow Removed” 消息来让交换机在每个流表过期时自动地通知控制器。F 作为一个网络环境的参数，可以根据经验进行静态配置，也可以在网络运行过程中实时估计。链路监控模块定时地进行链路权值估计，得到每条链路的权值 $weight$，再与预先设定的阈值 $weight\_threshold$ 进行比较，若大于该值，表示链路过载，此时将触发路由管理模块进行路由调整。

2.2.3.2 拥塞感知的路由算法

路由管理模块负责组播转发树的生成及调整。当有新组播会话建立时，根据当前的链路状况生成开销最小的组播转发树；当有链路过载时，触发调整经过过载链路的组播转发树；当检测到链路失效时，更新所有经过失效链路的组播转发树。通过实时地根据链路状态来调整组播转发树从而可以实现拥塞感知的路由转发。路由管理模块分成两部分，路由生成与路由调整。在会话建立的过程中，需要快速生成转发树，而在链路拥塞或者链路失效的过程中，需要根据当前网络状况尽力而为地调整转发树，以达到更优的转发效果。

路由生成算法

每个组播会话的接收端成员都是由发送端来指定，因此 MCTCP 不允许成员的动态加入与离开。生成组播转发树的过程包括两个步骤，计算全局最短路径对（all-pair shortest paths）和计算转发树。在计算全局最短路径对时计算所有以边缘交换机节点为目标结点的最短路径，记为 GSP（Global Shortest Path）。在计算 GSP 时有两种算法，包括最短路径算法（SP, Shortest Path）以及最宽最短路径算法（SWP, Shortest Widest Path）。因此，基于这两种算法，在路由生成时有两种算法，包括基于 SP 的算法与基于 SWP 的算法。

（1）基于 SP 的算法（S-GSP 算法）。MCTCP 默认采用 SP 算法（例如 Dijkstra[61] 或 Floyd Warshall[62]）来进行 GSP 的计算。此处的最短路径指最小开销路径，即路径的总权值最小。在计算 GSP 之后，采用最小代价转发树算法（Minimum-cost Path Heuristic Algorithm, MPH[63] 计算最小代价组播转发树 MST。

假设 GSP 内，节点 i,j 之间的最小距离（代价）为 $w(i,j)$，组播发送端 s，接收端 $u \in U$。算法中考虑 u 到达 v 与 v 到达 u 的距离等价。MPH 算法的流程如下：1). 将组播成员接收节点都加入集合 M，然后把源节点加入转发树 T，计算 M 中所有接收节点到转发树 T 的最小距离和最短路径。2). 从 M 中取出到转发树 T 的距离最小的节点 u，并将该节点 u 和转发树 T 上到达 u 距离最小的节点 v 之间的路径加入到转发树 T 上，并将 u 从 M 中删除。3). 对于步骤 2 中每个新加入到转发树 T 的节点 x，检查其到达 M 中所有结点 m 的距离，若该距离小于转发树 T 到达 m 的距离，那么更新转发树 T 到达 m 的距离，并更新转发树 T 到达 m 的最短路径。4). 重复执行

步骤 2 和步骤 3，直到集合 M 为空。

（2）基于 SWP 的算法（W-GSP 算法）。MCTCP 应用的很多场景对组播传输带宽比较敏感，而上述算法计算出的组播转发树并未考虑瓶颈路径对组播传输带宽的影响。在 MCTCP 中，传输一份数据时，只有在所有的接收端都接收到数据之后才判定该数据传输成功。因此，在组播转发树内的任何一条瓶颈路径都能影响到该组的整体传输带宽。为了避免或者缓解瓶颈路径对组播带宽的影响，采用基于 Shortest Widest Path (SWP)[64] 的算法来生成组播转发树。

首先，采用 SWP 算法来计算 GSP。给定两个限制 B_a 和 D_a，SWP 算法的目标是保证计算得到的路径的空闲带宽大于 B_a，路径长度小于 D_a。在该算法内部，其首先将空闲带宽小于 B_a 的链路从网络中剔除，再使用最短路径算法（例如 Dijkstra 算法）计算网络的最短路径。W-GSP 算法的目标是尽量避开瓶颈路径从而选取空闲带宽较大的路径。假设把空闲带宽小于 B_a 的链路作为瓶颈路径，那么可以直接使用 SWP 算法计算 GSP。然而，为了减少计算开销，在每个周期内只计算一次 GSP，而在该周期内可能有多个组播转发树需要进行计算。对于每个组播转发树，其瓶颈路径各不相同，因此，不能保证每个组都达到最大空闲带宽。

在使用 SWP 计算 GSP 时如何确定 B_a 值是面临的挑战。由于 W-GSP 算法的目标是尽量选取空闲带宽较大的路径来重新放置组播组以提升传输性能，因此，需要综合考虑当前现有组播组的传输带宽来确定 B_a 的取值。首先，链路的最小空闲带宽不得小于某个固定的阈值 $B_a = \eta \cdot B$（例如 $\eta = 0.1$），从而保证高负载的链路不会被选取。其次，最小空间带宽不得小于当前组播组传输速率的平均值，从而保证有足够的空间来调整组播转发树以优化组播传输带宽。因此，$B_a = max(\eta \cdot B, \overline{B_i})$，其中 B 是链路带宽，B_i 是组播会话 i 的带宽。

与 SWP 算法不同的是，W-GSP 算法希望尽量不选取传输带宽大于 B_a 的链路，而非不能选取。因此，在实现在过程中，对于权值大于 B_a 的链路，将其权值乘以 10。由于这些链路权值远大于其它链路，因此将不会被优先选取。若该链路为关键链路，那么其最终也将会被选取。在计算好 GSP 之后，任意两点之间的路径的带宽都大于 B_a，且总代价最小。为了得到最大带宽转发树，在计算组播转发树时使用 MPH 算法的变体（记为 WMPH 算法）。在 WMPH 算法的每个循环中，每次选取到达集合 M

中节点中具有最大空闲带宽的路径，而非最小开销的路径。如果存在多条具有相同最大空闲带宽的路径，则选取总代价最小的路径。

拥塞感知的路由调整算法

当链路监控模块检测到链路拥塞时，将触发路由调整。路由调整模块将获取经过拥塞链路的所有活动组播会话，重新计算每个会话的转发树。由于路由调整模块只能调整组播转发树，而不能调整除 MCTCP 外的流量，因此只能采用规避的方式进行调整，并不能保证总是达到最优。

算法 2.1 组播转发树调整算法

```
1: // 更新链路的权值，并且找到过载的链路
2: for l in L do
3:     l.weight = l.rate < B?l.rate/B : l.flow_num · F/B
4:     if l.weight > W_thr then
5:         C.append(l) // 将过载的链路存储在 C 中
6:     end if
7: end for
8: for g in G do
9:     if g.link in C then
10:        n = g.newMst() // 计算新组播转发树 MST
11:        // 检查是否需要更新组播转发树
12:        if CheckMST(n.L, g.L) then
13:            //B_g 为组 g 的传输带宽
14:            for l in n.L - g.L do
15:                l.weight+ = B_g/B
16:            end for
17:            for l in g.L - n.L do
18:                l.weight− = B_g/B
19:            end for
20:            g.update() // 更新组播转发树
21:        end if
22:    end if
23: end for
```

（1）基于 SP 的算法。对于基于 SP 的算法，通过比较计算出来的新 MST 与旧 MST 之间的总权值来决定是否需要更新该组的 MST。假设已经将组 G_1 调整到新的 MST，那么 G_1 将会在新 MST 中的“新链路”（即在新 MST 中但不在旧 MST 中的链路）上产生负载。因此，需要更新“新链路”的权值，即将“新链路”的权值加上 B_1/B，其中 B_1 为组 G_1 的带宽。之后，比较新 MST 与旧 MST 的总权值，如果

算法 2.2 CheckMST(L_{new}, L_{cur})

```
1: // 计算当前 MST 的总权值 L_cur
2: for l in L_cur do
3:     C_cur += l.weight
4: end for
5: // 计算新 MST 的总权值 L_new
6: for l in L_cur ∩ L_new do
7:     C_new += l.weight
8: end for
9: for l in L_new − L_cur do
10:    C_new += l.weight + B_g/B
11: end for
12: if C_cur − C_new ⩾ C_thr then
13:    return TRUE
14: end if
15: return FALSE
```

新 MST 与旧 MST 之间的总权值有降低，并且达到一定的阈值，则将组播调整到新的 MST 上。具体地，考虑一个 MST $M(V, L)$，其中 V 和 L 分别代表节点与链路的集合。任意链路 $l \in L$ 都具有权值 $w(l)$，用 C 表示 MST 的总权值，即 MST 中所有链路的权值之和。C_{thr} 表示 MST 调整需要达到的总权值降低的阈值。对于当前的 $M(V_{cur}, L_{cur})$，其总权值为 $C_{cur} = \sum^{l \in L_{cur}} w(l)$ 。对于新 MST $M(V_{new}, L_{new})$，其总权值为 $C_{new} = \sum^{l \in L_{new} \cap L_{cur}} w(l) + \sum^{l \in L_{new} - L_{cur}} (w(l) + B_1/B)$。当 $C_{cur} - C_{new} \geqslant C_{thr}$ 时，新 MST 将会被安装到交换机上。默认地，$C_{thr} = B_1/B$。

（2）基于 SWP 的算法。对于基于 SWP 的算法，不再用 MST 的总权值来判断是否需要进行调整，而是用 MST 中的空闲带宽，以及相比于当前组播的带宽提升空间来判断。如果新 MST 的空闲带宽 B_n 大于当前组播 G_1 的带宽 B_1，即 $B_n > B_1 \cdot (1+\alpha)(\alpha > 0)$，那么新 MST 将会被使用。这保证了新 MST 中具有足够的空闲带宽来提升组播 G_1 的传输带宽。默认地，$\alpha = 0.3$，其值也可以根据网络状态进行调整。

为减小调整开销，在每个调整的周期内只进行一次 GSP 计算。下面考虑多个组需要进行调整的情况。当一个组进行转发树调整之后，其会改变相关链路的权值。因此需要将这些变化考虑进去以指导后续组的调整。例如，假设某一个组 G_1 调整之前的转发树的集合为 $L_{cur} : \{l_1, l_2, l_3\}$, 调整之后的转发树集合为 $L_{new} : \{l_1, l_3, l_4\}$。那么 G_1 将原本在链路 l_2 上的流量切换到 l_4 上，则需要对 l_2 和 l_4 的权值 $weight$ 进行修正，

即 $w(l_2) = w(l_2) - B_1/B$ 和 $w(l_4) = w(l_4) + B_1/B$。因此在调整后面的组时，可以根据新的链路权值进行计算。路由调整的伪代码见算法2.1和算法2.2。

2.3 基于组播的分布式存储系统多副本机制

分布式存储系统往往通过多副本来保证数据的可靠性。最为常见的多副本策略有三种：（1）直接副本方式，即客户端直接写多个副本到各个存储节点，例如 SheepDog①，GlusterFS②等；（2）主备副本方式，即客户端将数据写入到一个主副本，再由主副本节点将数据写入到其他副本节点，例如 Ceph[8] 等；（3）链式副本方式，即客户端将数据写入到第一个副本，然后第一个副本节点将数据写入到第二个副本，依次类推直至数据写入到所有副本节点，例如 HDFS[7] 等。

对于直接副本方式，客户端同时将数据写入到 N 个存储节点，因此通常具有较小的写入延迟，假设为 T，但是吞吐率只能达到网络带宽的 $1/N$。对于主备副本方式，理论上具有直接副本方式两倍的写入延迟，即为 $2 \cdot T$，最高吞吐率为网络带宽的 $1/(N-1)$。而对于链式副本方式，其写入延时较高，为 $N \cdot T$，但是具有较高的吞吐率，能够达到 100% 的网络带宽。

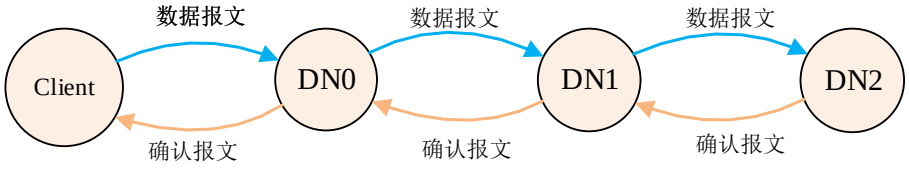
本章提出基于组播的多副本写入策略，通过组播的方式可以最大化网络带宽的使用效率。因此，组播多副本方式下，可以同时达到较低的延迟与较高的吞吐率。本章实现了基于 MCTCP 的 HDFS 多副本写功能，以优化 HDFS 写操作的时延，减少网络开销，提升网络的利用率。HDFS 多副本写的过程是一个数据中心内很典型的一对多数据传输，客户端在发起写请求之前先向 NN（NameNode）获取要写入的 DN（DataNode），然后再将数据传输到对应的 DN。

如图2-5 (a) 所示，原始版本的 HDFS 采用流水写入的方式，客户端将数据分成以包（packet）为单位进行传输。对于每个包，先被传输到 DN0，DN0 将数据存储并同时传输给 DN1，DN1 将数据存储并同时传输给 DN2。DN2 接收到数据后，向 DN1 发送确定报文，DN1 再给 DN0 发送确认，最后 DN0 向客户端发送确认，并结束一个包的写入过程。整个写入的过程可以看作是一个六段流水线，客户端在传输第一个包给

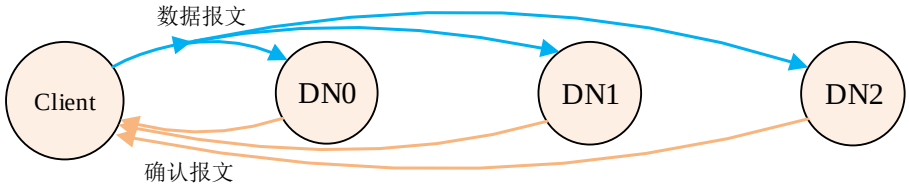
① https://sheepdog.github.io/sheepdog/

② https://www.gluster.org/

DN0 之后接着传输第二个包，直到所有包都传输完成。称原始的 HDFS 为 HDFS-O，HDFS-O 的写入过程流程很长，导致包传输的时延较长。另外，由于 HDFS-O 在传输数据的各个阶段都采用单播方式传输，在网络中会产生大量的重复报文，将增加网络开销。



(a) 链式多副本方案



(b) 组播多副本方案

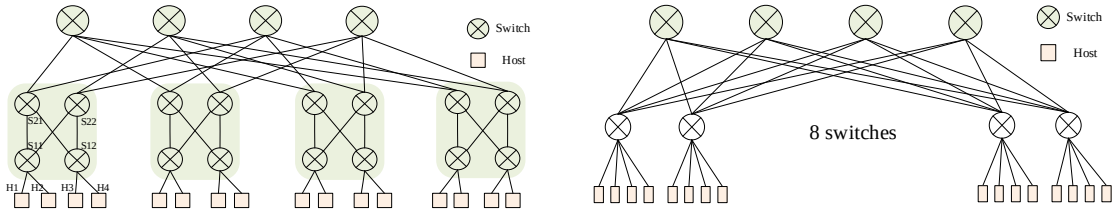
图 2-5 链式多副本方案与组播多副本方案数据流对比

为减少 HDFS 写操作的网络开销，利用 MCTCP 来实现 HDFS 多副本写操作，称之为 HDFS-M。如图2-5 (b) 所示，客户端将数据分成为包为单位进行传输，然后通过 MCTCP 将数据同时传输给 DN0，DN1，DN2。对于每个数据包，客户端只需要发送一次，由 MCTCP 来保证数据传输的可靠性，而每个 DN 接收到数据包后直接返回确认给客户端。HDFS-M 写操作的流程可以看作是两段流水线，客户端不断地发送数据包给 DN，直到完成。

与 HDFS-O 相比，HDFS-M 写操作的流程变短（从六段到两段），降低了每个包的延迟。同时，HDFS-M 采用组播的方式传输数据，极大地减少了网络中的冗余报文，降低了网络开销。类似地，也实现了基于 TCP-SMO[49] 的 HDFS 多副本，记为 HDFS-T。TCP-SMO 是一个传输层的可靠组播方案，由于其是接收端发起的组播方案，在实现时需要通过额外的通信机制来告知每个数据节点提前加入特定的组。

2.4 性能评价

测试平台在虚拟网络拓扑环境 Mininet[65] 上搭建。硬件平台为一台服务器，具体配置为：Intel(R) Xeon(R) E5-2620@ 2.00GHz CPU，32GB 内存。安装 Ubuntu 12.04.5 LTS 操作系统、Mininet 2.2.0①模拟软件和 Openvswitch 2.4.0[66] 虚拟交换机软件。SDN 控制器使用 RYU 3.17②。测试过程中使用两种数据中心内常用的网络拓扑，即 Fat-tree[67] 拓扑与 Leaf-Spine[68] 拓扑。Fat-tree 拓扑中 $k=4$，因此具有 20 个 4 端口的交换机，16 个主机，如图2-6a所示。Leaf-Spine 拓扑中包涵 4 个 spine 交换机，8 个 leaf 交换机，每个 leaf 交换机连接 8 个主机，如图2-6b所示。



(a) Fat-Tree 拓扑结构　　(b) Leaf-Spine 拓扑结构

图 2-6 测试中所使用的两种网络拓扑结构

测试过程中使用了两种 MCTCP 实现，分别为基于 SP 算法（记为 MCTCP-S）和基于 SWP 算法（记为 MCTCP-W）的版本。共进行了三种测试，包括基本测试、真实背景负载测试和多副本性能测试，同时还讨论了控制器的复杂度。

基本测试：主要评估 MCTCP 的数据传输带宽，以及在处理链路拥塞、链路失效时的效果和与 TCP 共享链路时的 TCP 友好性。与最为流行的应用层开源可靠组播 NORM[48]、openPGM③和一个类似的传输层可靠组播方案 TCP-SMO[49] 进行对比。此处只对比可靠组播方案进行性能对比测试。

真实背景负载测试: 利用两个真实数据中心负载作为背景负载，测试 MCTCP-S、MCTCP-W 与 TCP-SMO 在不同负载下的传输带宽。此处选用两种数据中心常见的负载：网页搜索（Web Search）[69] 和数据挖掘（Data Mining）[70]，在测试过程中生成这

① http://mininet.org/
② http://osrg.github.io/ryu
③ http://code.google.com/p/openpgm

两种模式的背景负载。

多副本性能测试：对比测试基于 MCTCP、TCP-SMO 的 HDFS 组播多副本机制与原始的流水式的 HDFS 多副本机制在真实数据中心负载下的性能。

控制器复杂度：讨论 SDN 控制器的复杂度，包括算法的运行时间、计算与网络的开销等。

2.4.1 基本测试

本节测试的目的是评估 MCTCP 的数据传输性能，在处理链路拥塞、链路失效情况下的效果和与 TCP 共享链路时的友好性（TCP-Friendliness）。先对比测试了 MCTCP 与两个最为流行的开源可靠组播方案 NORM、openPGM 以及 TCP-SMO 在无链路拥塞与具有拥塞时的结果，然后测试了 MCTCP 在处理链路失效以及与 TCP 协议共享链路时的结果。

首先，分别测试了 NORM、openPGM、TCP-SMO 与 MCTCP 的传输带宽，全部开启拥塞控制功能，发送速率为 500Mbps。本测试中接收端个数为 3，发送端为 $H1$，接收端为 $H2$,$H3$,$H4$。传输开始后，通过观察交换机中的流表确定当前传输所选取的组播转发树，第 20s 时在当前组播转发树的某一链路之间启动 Iperf 以模拟产生链路拥塞。Iperf 持续传输 15s。

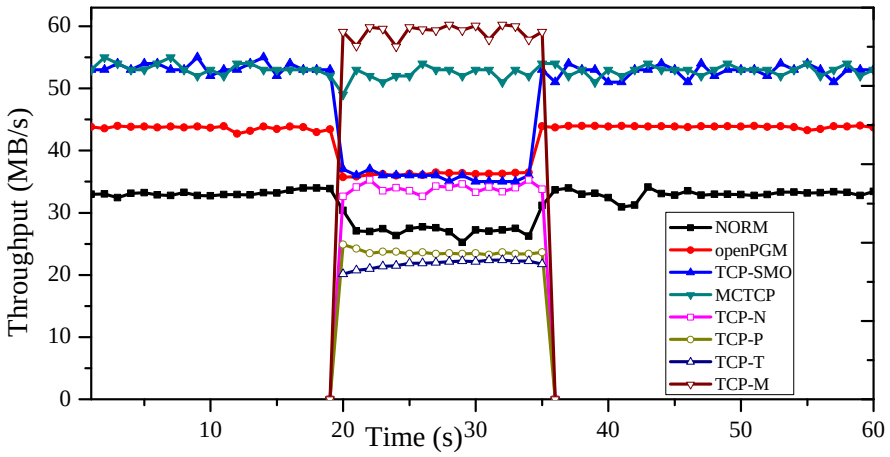


图 2-7 NORM、openPGM、TCP-SMO 和 MCTCP 的传输带宽对比

图2-7为该四个方案的带宽测试结果。图中 TCP-N，TCP-P，TCP-T 和 TCP-M 分

别表示在 NORM，openPGM，TCP-SMO 和 MCTCP 测试中的 TCP 流。从结果中可以看到，无链路拥塞时（即 0-20s 与 35-60s 之间）MCTCP 相比于 NORM 与 openPGM 分别提升了 60% 与 22% 的传输带宽。当链路出现拥塞时（即 20-35s 之间），MCTCP 几乎不出现性能下降，而 NORM、openPGM 与 TCP-SMO 的带宽分别下降了 17%、17% 和 33%，因此 MCTCP 的带宽分别为它们的 1.9、1.44 与 1.45 倍。MCTCP 性能优于其它三种方案的原因主要有以下两点。第一，MCTCP 在 Linux 内核下实现，是一个传输层的协议，而 NORM 与 openPGM 均为用户态实现的应用层协议（在 UDP 协议之上实现的）。因此 MCTCP 在处理数据传输、报文确认以及数据重传时具有更高的效率。第二，MCTCP 可以实时地检测链路状态，当发现拥塞时可以立即调整组播转发树以避开拥塞链路。因此，当链路出现拥塞时，MCTCP 可以很快地更新组播转发树，从而最小化因拥塞带来的性能损失。而另外三个对比的方案一旦建立好组播转发树，则不会中途更改，因此在拥塞出现时将会导致极大的性能下降。

然后，测试了 MCTCP 在处理链路失效和与 TCP 共享链路时的结果。由于上述三个对比方案都是基于传统的 IGMP 组管理协议设计的，在处理链路失效的情况时依赖于 IGMP 的机制。例如在二层数据中心内部，一般使用 IGMP snooping 进行组管理。IGMP snooping[①]中由 IGMP 查询器定期向本地网段内的所有主机与路由器（224.0.0.1）发送 IGMP 普遍组查询报文，以查询该网段有哪些组播组的成员。在查询的时间间隔内不会更新组播转发树，一旦出现链路失效至少需要等到下一次查询才能恢复。因此，NORM、openPGM 和 TCP-SMO 在处理链路失效时的恢复时间依赖于 IGMP 查询器的查询时间间隔，传统网络中一般为 10s-60s。因此本测试中并未测试 NORM、openPGM 与 TCP-SMO 处理链路失效的结果。

与前面的测试类似地，在 MCTCP 开始传输数据之后，观察到组播转发树 MST 为 $\{S11 \rightarrow S21, S21 \rightarrow S12\}$。在 20s 时刻，将 $S11 \rightarrow S21$ 断开，之后在 44s-74s 之间，在 H2 与 H3 之间启动 Iperf。测试过程中 TCP 与 MCTCP 都运行“reno”拥塞控制算法。图2-8为测试结果。从图中可以看到，在 20s 时刻，链路失效对 MCTCP 传输几乎不产生影响。这是由于当 MCTCP 检测到链路失效时，会立即触发路由调整，将组播转发树更新到正常的链路上。在 44s-74s 之间，由于 TCP 流量的出现，MCTCP

① https://en.wikipedia.org/wiki/IGMP_snooping

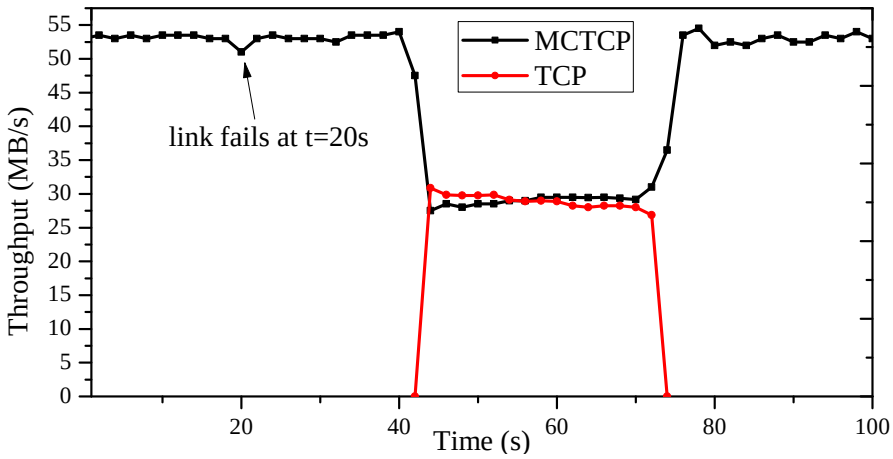


图 2-8 在产生链路失效以及与 TCP 共享时 MCTCP 的带宽。

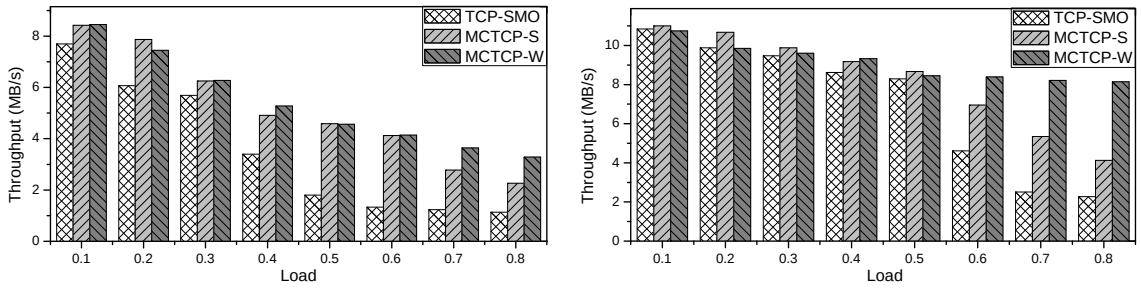
会调整拥塞窗口，与 TCP 共享网络资源。从结果中可以看到 MCTCP 与 TCP 平均分配带宽，具有较好的 TCP 友好性。

2.4.2 真实背景负载测试

本测试旨在测试 MCTCP 在真实的数据中心背景负载存在时的效果。因此考虑 MCTCP 与背景负载网络共享网络资源的情况，分别使用了两种负载模式：web search 与 data mining 负载。测试的节点被分为两组，一组用于生成背景负载（$GroupA$），另外一组用于组播测试（$GroupB$）。具体地，将所有偶数的节点加入 $GroupA$，而将所有奇数节点加入 $GroupB$。对于 $GroupA$，所有的节点都打开接收端口，并且节点 i 将节点 $(i+4)mod(numhosts)$ 发送数据。报文的长度符合上述两种负载的 CDF。

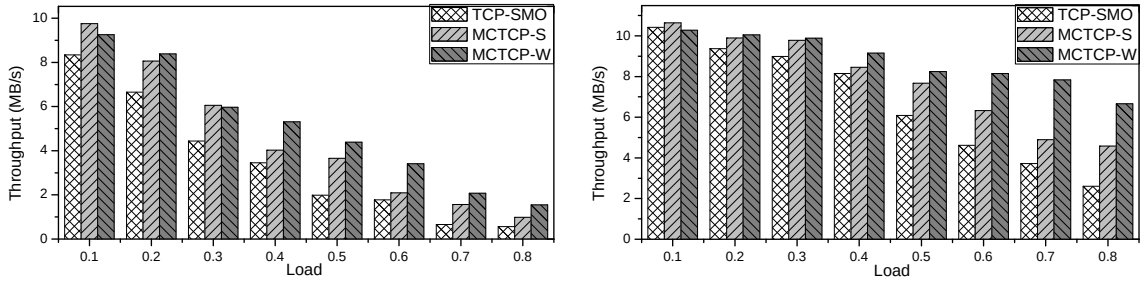
对比测试了 TCP-SMO，MCTCP-S 和 MCTCP-W 在不同背景负载情况下的吞吐率，其中负载的强度从 0.1 到 0.8。测试分别在两种数据中心典型网络拓扑结构下进行，即 Fat-Tree 与 Leaf-Spine 拓扑。分别在 Fat-Tree 与 Leaf-Spine 拓扑下启动两个和四个组播会话，每个组播包括一个发送端和三个接收端。同时保证单个组播内的接收端跨三个机架，并且不同的组之间的成员不重合。测试过程中运行 ECMP。MGM 模块的调度周期为 2 秒。

图2-9和图2-10显示了 TCP-SMO、MCTCP-S 与 MCTCP-W 在不同模式的背景负载下两种网络拓扑中的吞吐率。观察到以下几点结论：第一，MCTCP（包括 MCTCP-S



(a) Web server 模式背景负载　　(b) Data mining 模式背景负载

图 2-9 Fat-Tree 拓扑结构下，TCP-SMO、MCTCP-S 和 MCTCP-W 在两种模式的背景负载下的带宽



(a) Web server 模式背景负载　　(b) Data mining 模式背景负载

图 2-10 Leaf-Spine 拓扑结构下，TCP-SMO、MCTCP-S 和 MCTCP-W 在两种模式的背景负载下的带宽

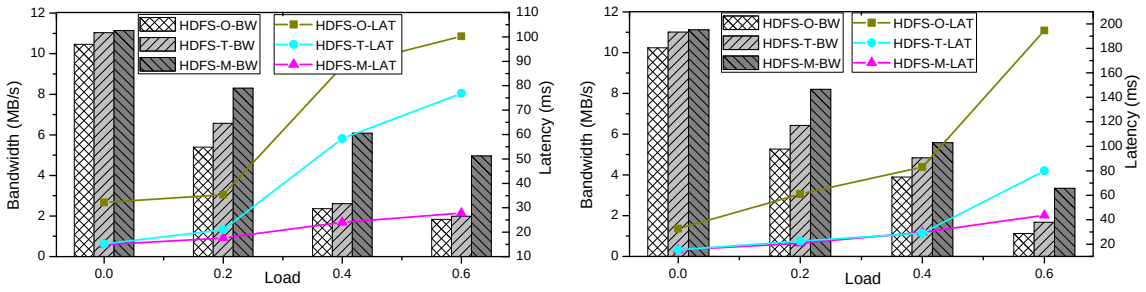
与 MCTCP-W）的带宽高于 TCP-SMO。具体地，相比于 TCP-SMO，在 web server 模式背景负载下，MCTCP-S 的带宽在 Fat-Tree 与 Leaf-Spine 拓扑下分别提升约 10% – 2.1 倍和 17% – 1.3 倍，MCTCP-W 的带宽均提升约 10% – 2.1 倍。在 data ming 模式背景负载下，MCTCP-S 的带宽在 Fat-Tree 与 Leaf-Spine 拓扑下分别提升约 1% – 1.1 倍和 2% – 76%，MCTCP-W 的带宽分别提升约 1% – 2.5 倍和 7% – 1.5 倍。第二，MCTCP-W 在大部分测试情况下比 MCTCP-S 性能更高，尤其在负载强度大于 0.5 的情况下。

由于 MCTCP 可以实时地发现网络中的空闲链路，并将组播的转发树调整到当前拥塞更小的链路上去，从而可以在网络拥塞时比其它方案达到更好的性能。由于只有一半的节点在生成背景负载，因此网络中的流量并未使得网络链路达到饱和，从而留给 MCTCP 大量的调度空间。MCTCP-S 可以发现网络中具有更小的总代价的 MST，

从而让组播尽量处于一个较轻的拥塞状态下，提升组播传输的性能。然而 MCTCP-W 则是尽量找到网络中空闲带宽更大的 MST，从而最大化组播传输的带宽。当负载强度较小时（如小于 0.5），网络中的瓶颈链路比较少，从而基于 SP 的算法与基于 SWP 的算法效果相当。但当负载强度较大时，网络中大量的瓶颈链路将会极大地降低组播传输的带宽。根据木桶原理，组播转发树中最为拥塞的那条链路将会成为该组传输的瓶颈。因此，即使 MST 的总代价较小，其总带宽也不一定高。这种情况下，基于 SWP 的算法由于充分考虑到了 MST 中瓶颈链路的影响，从而具有更好的带宽，即 MCTCP-W 在负载强度大于 0.5 时性能明显高于 MCTCP-S。

2.4.3 多副本性能测试

本节在 web search 背景负载模式下对比测试了 HDFS-O、HDFS-T 和 HDFS-M 的性能。HDFS 默认 3 副本，其中一个副本放置在本机架的一个节点上，另一个放置在另一机架的一个节点，最后一个放置在另外一个相同机架的不同节点上。但是，在很多情况下，三副本需要写到三个远程的节点上，以保证系统的负载均衡或者提升数据的可靠性。为充分显示网络对 HDFS 写操作的影响，测试过程中将三个副本写到随机选取的远程节点上。测试过程中，数据写到内存。在前述两种网络拓扑结构下进行了测试，同时采用与前述相同的背景负载生成方法，分别测试了背景负载强度从 0 到 0.6 时的结果。当负载强度为 0 时，表示没有背景负载。



(a) Fat-Tree 拓扑结构　　(b) Leaf-Spine 拓扑结构

图 2-11 HDFS-O、HDFS-T 和 HDFS-M 在 Fat-Tree 与 Leaf-Spine 网络拓扑结构下的性能对比

测量了两个指标，每个包的时延和总带宽。包大小为 64KB，测试写数据量为 500MB。图2-11所示为包时延与总带宽测试结果。从结果中观察得到以下三点结论：

第一，基于组播的多副本比链式的多副本机制具有更好的性能，尤其在包的时延方面。链式多副本方案下，其包的时延基本上是组播多副本方案中的 3 倍以上。第二，在两种网络拓扑结构下，HDFS-M 都比 HDFS-O 在吞吐率上提高 50% – 1.5 倍，同时在包时延上降低了 50% – 72%。第三，在无背景负载时，两个基于组播的多副本方案 HDFS-M 与 HDFS-T 性能相当，而当有背景负载时，HDFS-M 比 HDFS-T 带宽提升 20% – 1.3 倍，时延降低 10% – 45%。

HDFS-M 性能优于 HDFS-O 与 HDFS-T 的原因主要有两点。第一，相比于 HDFS-O，HDFS-M 具有更短的数据传输与处理路径，并且利用组播技术减少了大量的网络冗余报文。在 HDFS-O 中，每个报文都将被串行地被多个副本节点处理，因此，其处理时延包括多次的传输与处理时延。另一方面，当网络拥塞时，多次的单播转发引起的大量冗余报文将会加剧网络拥塞，导致极大地性能下降。因此，HDFS-M 的时延比 HDFS-O 更低，同时更短的路径也意味着在网络中受到其它流量的影响的概率和程度也将极大地降低。第二，HDFS-M 可以根据当前网络拥塞情况进行合理的路由调整，可以找到最为空闲的链路进行转发，因此其具有更好的数据传输效率。HDFS-O 与 HDFS-T 都不能感知网络拥塞，无法调整组播转发树来规避拥塞链路。对于 HDFS-O，一方面在流水线的各个阶段都受到负载的影响，另一方面也不具有避开拥塞链路的能力，因此在链路存在空闲的情况下依然产生极大地拥塞。对于 HDFS-T，虽然与 HDFS-M 一样具有较短的传输路径，但是无法在网络拥塞时绕过拥塞链路，从而在背景负载下将会导致更大的性能下降。

2.4.4 控制复杂度分析

计算 MST 所需的全局最短路径（GSP，Global Shortest Path）会在控制器启动时和每个调整周期开始时计算好。从而当一个组到达时，可以立即生成其 MST。一个组播转发树计算的时间独立于网络拓扑的规模和当前组数的多少，而只与接收端数有关。

如图2-12所示为在 $k = 8$ 和 $k = 16$ 的 Fat-Tree 拓扑下不同接收端数目时处理一个组所需的时间。对于 $k = 8$ 的 Fat-Tree 拓扑，其总共有 32 个边缘交换机。因此，当接收端数目大于 32 时，组播的 MST 将会跨所有的机架，从而使得计算时间达到峰

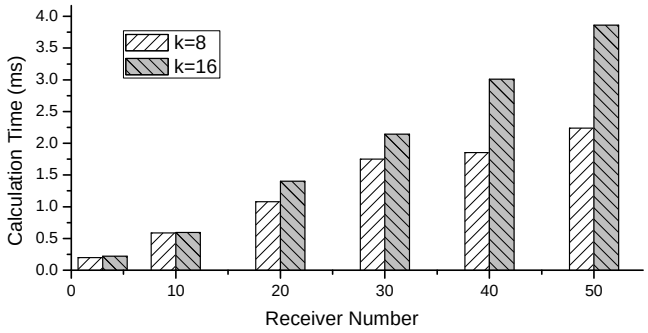


图 2-12 不同接收端数目时 MST 的计算时间

值。另外，值得注意的是，在生成一个组播组时，包括计算 MST 和安装到交换机两个过程。安装到交换机的时间与使用的控制器平台有关，并不在本文讨论的范围内。

表 2.1 不同网络拓扑时控制器的计算时间

拓扑	链路检测 (ms)	计算最短路径 (ms)	生成组播路由	
			计算路由 (ms)	安装路由 (ms)
k=8	38.41	1.789	0.615	13.929
k=16	154.254	25.544	0.722	13.53

在一个调整周期内，控制器首先需要获取当前链路的状态。本章使用“Port Status”接口获得每个交换机上的端口报文统计值，从而获得所有链路的状态。若检测到有链路拥塞，那么 MGM 模块会重新计算当前到达所有边缘交换机的最短路径 GSP。表2.1所示为在 $k=8$ 和 $k=16$ 的 Fat-Tree 拓扑下，检测链路状态和计算 GSP 的时间，以及生成一个组所需的时间。由于 RYU 实现机制的限制（单线程），测试得到的检测链路状态花费的时间较长。相信通过使用多线程可以极大地加快该过程。默认地，在每个调整周期内会计算完整的 GSP，从而在组数较多的场景下，可以加速调整过程。但是在组数较少的情况下，可以选择不进行完整的 GSP，而只计算生成 MST 所需的最短路径对。

从算法2.1中可以得出，MCTCP 的组播调整算法的时间复杂度为 $O(|L|+|G|\cdot O(MST))$。其中 $|L|$ 为链路数，$|G|$ 为需要调整的组数，$O(MST)$ 是计算单个 MST

表 2.2 不同的组数（g）时控制器的调整时间

拓扑	g=10(ms)	g=100(ms)	g=1000(ms)	g=10000(ms)
k=8	12.458	75.695	491.822	5189.126
k=16	62.951	111.259	635.594	5453.479

表 2.3 不同的组数（g）时控制器的 CPU 利用率与网络开销

拓扑	CPU 利用率 (%)		网络带宽 (KB/s)	
	g=100	g=1000	g=100	g=1000
k=8	1.20	4.70	54.8	55.2
k=16	3.10	5.80	292	293

的复杂度。对于 MPH 算法，计算一个 MST 的时间复杂度为 $O(mn^2)$，其中 m 为组播组中接收端数目，n 是网络中总的节点数。表2.2所示为在 $k=8$ 和 $k=16$ 的 Fat-Tree 拓扑下，调整不同组数所需的时间。从表中可以看出，调整时间与组数成线性增长关系，因此可以处理大量组数的场景。另外，还测试了在不同的组数时 $k=8$ 与 $k=16$ 的 Fat-Tree 拓扑下 CPU 与网络的开销。如表2.3所示，从结果中可以看出，控制器的计算与网络开销很小，几乎可忽略不计。

2.5 本章小结

为了优化分布式存储系统中多副本机制，本章提出基于组播的多副本写入策略，从而可以极大地减少网络中的冗余报文，提升数据写入的效率。然而，现有的可靠组播方案在易用性与性能方面都无法满足应用需求。因此，为了满足应用对可靠组播的需求，本章提出 MCTCP，一种基于 SDN 环境的可靠组播数据传输方案，主要适用于数据中心内大量的小组场景。其通过扩展 TCP 协议来实现一对多模式下的可靠数据传输，是一个由发送端发起的传输层协议。每个组播的转发树 MST 由中心化的组播管理模块（MGM）进行管理。利用 SDN 控制器的全局控制能力，MGM 可以根

据当前网络的实时状态来调度组播转发树以规避拥塞和失效链路。因此，MCTCP 在主机端处理与路由转发两个方面都具有很高的效率，可以很好的满足上层应用对组播传输的性能需求。实验表明，MCTCP 可以达到比现有所有可靠组播方案更好的传输性能。最后，将 MCTCP 应用于实现分布式存储系统 HDFS 的多副本机制。从测试结果可以看出基于组播的 HDFS 多副本机制比原始的链式多副本机制具有更好的性能（时延和带宽）。

3 基于动态网络流量调度的存储系统性能优化方法

为了提升存储系统的性能，很多分布式存储系统通常部署两个网络，包括前端网络与后端网络。用户的请求流量将使用前端网络，而系统的后台流量（如恢复、重均衡）将使用后端网络。部署两个隔离的网络既可以提升存储系统的性能，也可以在一定程度上提升系统的安全性。然而，由于两个网络在部署上是完全隔离的，将会导致网络资源的浪费。例如，当一个网络空闲而另一个网络过载时，空闲的网络资源将无法被充分利用。为了解决这个问题，本章提出一种跨两个网络的动态流量调度方案 MAX，可以自动地发现存储系统中空闲的网络带宽资源，并且充分利用该空闲带宽资源来提升存储系统的性能。在前端网络存在空闲带宽时，MAX 可以将部分恢复或者重均衡的流量调度到前端网络中，通过提升网络资源利用率来提升系统恢复或者重均衡的性能。本章实现了 MAX 并将其应用到 Ceph 存储系统中。实验结果表明 MAX 可以提升 Ceph 的恢复和重均衡的性能。同时在请求与恢复操作共存时，Ceph 可以在不影响前端请求的情况下减少恢复的时间。

3.1 跨双网络动态流量调度的研究动机

分布式存储系统中的网络流量分为两类，包括前端流量和后端流量。前端流量是客户端发起读写请求，以及存储系统给客户端返回请求结果的流量。后端流量是存储系统内部维护系统状态产生的流量，例如数据恢复、数据重均衡的流量。分布式存储系统中往往采用多副本或者纠删码机制来保证数据的可靠性。当节点失效时，存储系统需要及时地对丢失的数据进行恢复。在存储系统恢复的过程中，将在网络中产生大量的后端流量。当存储系统的容量达到饱和时，管理员需要向存储系统中加入新的存储节点以扩张系统的容量。当要缩减系统容量时，需要将系统中一些存储节点从系统中删除。节点的增加或者删除将会触发系统内部数据的重均衡，这同时会引起大量的数据迁移，产生大量的网络传输。随着分布式存储系统的规模的扩展，节点失效与重均衡场景将会变得更加普遍，从而使得存储网络成为系统瓶颈，影

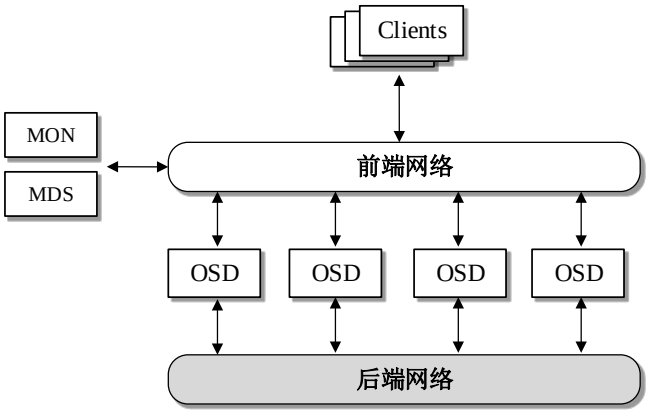


图 3-1 Ceph 存储系统的网络设置示意图

响请求的性能。

为了避免这些内部的网络流量影响到用户的请求的网络，在分布式存储系统的设计时，往往采用两个独立的网络，例如 Ceph[8]，Swift[71] 和 GlusterFS①。前端网络用于客户端请求的转发，而后端网络用于存储系统内部的网络流量传输。Ceph 存储系统的网络设置如图3-1所示，图中 MON 和 MDS 分别为 Ceph 的监控器与元数据服务器。其后端网络将对象存储节点（OSD, Object Storage Device）之间的复制、恢复或者重均衡流量从前端网络中分离过来②，从而减少了这些流量对前端应用请求的影响。另一方面，后端网络的存在也减少了部分安全攻击，例如 Dos（Denial of Service）等，对存储系统的影响。即使前端网络中的流量被攻击者扰乱，Ceph 系统中的监控与心跳信息依然可以通过后端网络正常工作，从而保证系统处于“active + clean”状态。除 Ceph 之外，Swift③与 GlusterFS④都具有相似的双网络设置。

然而，在很多情况下，前端网络与后端网络之间的负载不均衡。在系统恢复或者重均衡时，后端网络处于高负载，而前端网络可能处于空闲状态。例如，在一些 IOPS 密集型的应用场景下（如 Web server，OLTP 等），应用对带宽的需求不高，从而在大部

① https://www.gluster.org/
② http://docs.ceph.com/docs/master/rados/configuration/network-config-ref/
③ https://www.swiftstack.com/docs/admin/hardware.html
④ http://www.snia.org/sites/default/files/SDCIndia/2016/Presentations/Gluster%20Future%20Roadmap%20May%202016.pdf

分时间内前端网络中留有大量的空闲带宽。在 Ceph 文件系统中测试了 Filebench[①]的 Web server 负载，其总带宽为 11Mbps 左右。通常情况下，系统管理员往往会在前端应用无请求或者请求负载比较低时对系统进行扩容或者缩容。

如何充分利用分布式存储系统中空闲的网络带宽，从而在保证前端请求性能不受影响的情况下提升系统恢复与重均衡操作的性能是需要解决的问题。然而，现有的提升网络资源的利用率的方案都是针对单个网络内部设计的。Hedera[17] 通过周期性地重新计算网络中大流的放置来实现动态的网络流量调度，尽量避免大流之间的冲突，提升网络传输效率。MicroTE[18] 通过利用短期的局部流量预测来实现细粒度的流量工程。B4[22] 与 SWAN[23] 分别为 Google 与微软的数据中心间广域网互连网络，通过 SDN 中心化的调度来提升其网络利用率。然而现有的研究都是针对单个网络内的流量调度，无法实现跨双网络之间的流量调度。

为了解决分布式存储系统中在部署双网络时存在的网络资源浪费的问题，本章提出 MAX，在前端网络具有空闲带宽时，通过动态地调度后端网络的流量到前端网络以提升网络资源的利用率，从而提升系统恢复与重均衡性能。为了避免流量调度导致对前端请求的性能干扰，提出优先级路由的方式，保证前端请求的优先转发。MAX 是独立于具体的分布式存储系统，可以容易地部署在具有两个（或者多个）网络的分布式存储系统上。

3.2 动态网络流量调度方案

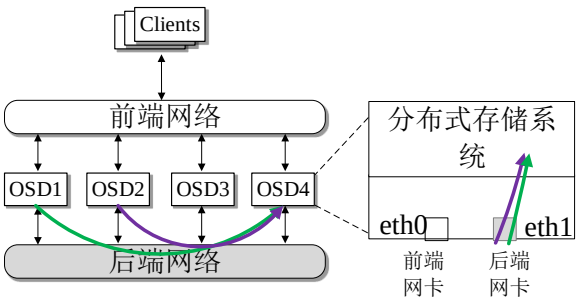
在设计分布式存储系统中跨前端与后端网络间动态的网络流量调度方案时，需要达到以下目标：**（1）易于部署**。需要不修改网络互连结构，比如不需要修改交换机与交换机之间、交换机与主机之间的互连拓扑，不需要修改交换机的配置。总之，MAX 需要能够直接部署在现有的通用硬件和通用的存储系统配置上。**（2）可移植性**。需要易于实现，不能修改存储系统的代码实现，从而具有良好的可移植性。不同的存储系统，例如 Ceph 或者 GlusterFS，能够直接部署 MAX 以优化性能。同时 MAX 的实现不需要修改操作系统内核，因此可以部署在不同的操作系统上。**（3）高效率**。

① https://github.com/filebench/filebench/wiki

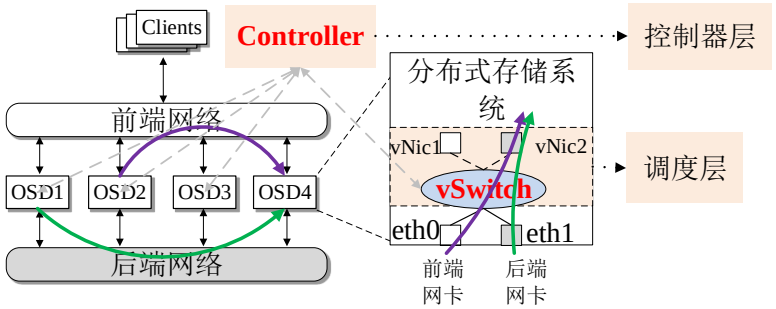
需要具有高效率，能够实时地监测前端网络中空闲网络带宽，并自动地调度相应的恢复或者重均衡流量以提升存储系统性能。同时需要保证网络资源的借用并不会影响到该网络原有流量的性能。为了达到这些设计目标，MAX 在网络层进行流量调度，并且采用中心化的调度方法来提升调度的效率。这里的网络层统称 I/O 路径上在存储系统应用层与存储节点物理网卡之间的层次。下面将介绍动态的网络流量调度方案 MAX 的系统架构。

3.2.1 系统架构

MAX 的系统架构如图3-2（b）所示。MAX 采用中心化的调度架构，包括位于每个存储节点上的*调度层*和一个独立的网络*控制器层*。



(a) Original



(b) With MAX

图 3-2 MAX 的架构及对比示意图

每个存储节点上具有两个物理网卡，分别用于连接前端网络（eth0）和后端网络（eth1）。MAX 在存储节点上安装一个虚拟交换机 vSwitch, 并且创建两个虚拟网卡 vNic1 和 vNic2。然后再将两个虚拟网卡、两个物理网卡连接到虚拟交换机 vSwitch。存储系统将使用虚拟网卡的 IP 地址，称为虚拟 IP 地址，分别为 vIP1 和 vIP2。存储

系统将虚拟 IP 地址当作它们的前端与后端网络的 IP 地址，从而所有的流量都将由虚拟网卡进入 vSwitch，再经过 vSwitch 调度之后才能进入到物理网卡。MAX 在控制器上动态地监听所有 vSwitch 的流量并计算出最优的路由，从而实现自动的流量调度。

以下举例说明 MAX 如何提升分布式存储系统的性能。假设当无前端用户请求，而有两条后端流量（恢复流量）分别为 $OSD1 \rightarrow OSD4$ 和 $OSD2 \rightarrow OSD4$。在原始的存储系统中，这两条流都将由后端网络转发，从而将在 OSD4 上的后端网卡上产生冲突，导致性能下降。而利用 MAX，其中一个流将会被调度到前端网络中，在 OSD4 上经由前端网卡进入到存储节点，从而避免了这两个流的冲突，如图3-2（a）所示。

由于单个网络内部的流量调度已经有大量的研究（例如 Hedera[17]），本章专注于跨两个网络之间的流量调度，而不关注单个网络内部的流量调度。因此，本章将两个网络都抽象成一个大交换机。MAX 的目标是提升网络资源的利用率，同时尽量保证两个网络之间的负载均衡。由于在存储系统中，用户请求无法到达后端网络，因此本方案可以具体化为在后端网络繁忙的时候，自动地发现前端网络中的空闲带宽，从而将部分后端网络流量调度到前端网络中，以提升系统后台活动（如恢复与重均衡）的性能。

3.2.2 网络控制器层设计

在网络控制器层，MAX 周期性地调整网络中各个流的放置来提升网络资源的利用率。与单个网络内的流量调度方案 Hedera 类似，MAX 只调度网络中存在的大流。由于 MAX 的主要目标是提升网络带宽的利用率，因此不对网络中的小流进行调度。MAX 的控制流程分为四个步骤：1、监听网络以探测出网络中的大流；2、估计各个大流的需求；3、计算每个大流的路由放置；4、将计算好的路由安装到相应的 vSwitch。

（1）流量监听

流量监听的主要目标是探测前端网络中的空闲带宽与发现后端网络中的所有大流。MAX 将会在所有存储节点上的 vSwitch 上监听经过其的流量。主要需要监听 vSwitch 上每个端口在调度周期内的传输量，以及 vSwitch 上所有流在调度周期内的传输量。可以利用 OpenFlow 的“get_port_stats”功能获得 vSwitch 上所有端口的传输量，利用“get_flow_stats”功能[72] 获得 vSwitch 上所有流的传输量。由于所有的流量

都经过 vSwitch，因此所需的统计信息可以一次性地从 vSwitch 上获得。对于一个流，如果测量得到其在一个调度周期内传输的速率大于一个阈值，则将该流标记为大流。默认地，该阈值为 $10\% \cdot B$，其中 B 为链路带宽。由于并不调度前端网络流量，因此前端网络的大流将被忽略。

（2）需求估计模型

在监听到所有存储节点上的前端与后端网络的速率之后，需要估计前端网络中的空闲带宽与后端网络中所有大流的需求。首先，需要计算前端网络中的空闲带宽。假设前端网络中，对于存储节点 i 的输入、输出网络速率分别为 RI_i 与 RO_i。那么两个存储节点 i，j 之间的前端空闲带宽为 $F(i,j) = B - max(RO_i, RI_j)$。由于在前端网络中可能存在被调度过来的后端流量，因此需要将后端流量排除。在流量监听的阶段，控制器层计算出了每个节点上的大流，以及它们各自的传输速率。假设被调度到前端网络的后端流量中，从节点 i 上发出的大流的的集合为 FO_i，发向节点 i 的大流的集合为 FI_i。则节点 i，j 之间的的空闲带宽修改为

$$F(i,j) = B - max(RO_i - \sum^{f \in FO_i} R_f, RI_j - \sum^{f \in FI_j} R_f)$$

其中 R_f 为流 f 的传输带宽。

其次，需要计算后端流量中所有大流的自然需求。这里使用 Hedera[17] 中提出的需求估计算法对每个大流进行需求估计。该算法假设所有流的性能瓶颈在于网络内部（而非主机节点上），从而估计出在当前网络拓扑下每个流可能达到的最大传输带宽。算法的输入为所有大流的集合，输出为每个大流的最大可能的传输带宽。

（3）流量放置算法

完成所有后端流量的需求估计之后，MAX 将根据每个流的需求来计算其应该放置在哪个网络中，以尽可能地提升网络资源的利用率。流量放置算法需要尽量保证两个网络的空闲带宽可以被均衡地使用。一种最为简单的方法就是将每个流放置到空闲带宽更大的网络中去。

然而，这种方法有以下几个缺点。第一，可能无法充分利用前端网络资源。例如，考虑三个节点 A/B/C，网络的链路带宽为 100Mbps。定义单个流在当前网络拓扑中能够达到的最大带宽为 R_{max}，并且假设 $R_{max} = 90Mbps$. 在前端网络中，链路 $A \leftrightarrow B$，

$B \leftrightarrow C$，和 $C \leftrightarrow A$ 的空闲带宽分别为 40Mbps，40Mbps 和 80Mbps。而在后端网络中，有四个大流 $A \rightarrow B$，$A \rightarrow C$，$C \rightarrow A$ 和 $C \rightarrow B$。通过需求估计算法计算得到这四个后端流的需求均为 0.5（即 50Mbps）。在放置流 $A \rightarrow B$ 时，前端网络与后端网络中的空闲带宽分别为 40Mbps 和 100Mbps，因此该流将被放置在后端网络中。然后后端网络的空闲带宽更新为 50Mbps。而当放置流 $A \rightarrow C$ 时，前端与后端网络的空闲带宽分别为 40Mbps 和 50Mbps，因此该流也将被放置在后端网络中。最后，两个大流都被放置在后端网络中，其总带宽为 100Mbps。然而，本例中最优的放置方案为将其中一个流放置在前端网络中，其总带宽可达到 130Mbps。

第二，结果与流放置的顺序关系较大。与上述例子相同地，在放置流 $C \rightarrow A$ 与 $C \rightarrow B$ 时，不同的放置顺序将会产生不同的结果。如果先放置流 $C \rightarrow A$，那么，在放置该流时，前端与后端网络的空闲带宽为 80Mbps 与 100Mbps，因此该流将被放置在后端网络中。然后流 $C \rightarrow B$ 也将被放置在后端网络中。而如果先放置流 $C \rightarrow B$，那么，流 $C \rightarrow B$ 将被放置在后端网络，而流 $C \rightarrow A$ 将被放置在前端网络中。

为了解决或者缓解上述两个问题，提出两点优化：首先，问题一的主要原因是没有考虑单个流可能的最大传输带宽，只是根据估计值来计算。此处假设所有大流能够达到的最大速率 R_{max} 相同。对于单个链路，如果后端网络中在其上放置的流量总和超过了链路带宽，那么需要保证至少有一个流被放置到前端网络中（如果前端网络中的空闲带宽大于一定的阈值 F_{thr}）。假设链路带宽为 B，对于一个流 f，它的前端与后端网络路径为 P_f 和 P_b，路径 P_f 上空闲带宽为 F。在路径 P_f 和 P_b 上放置的后端网络大流数目分别为 N_f 和 N_b。如果 $F > F_{thr}$，并且满足 $N_f = 0, (N_b + 1) * R_{max} > B$，那么流 f 将会被放置在前端网络中，即在 P_f 上。

其次，对于问题二，应当将可被调度的概率小的流先进行放置，即将前端网络空闲带宽小的流优先放置。从而那种前端网络空闲带宽较大的流将有更大的可能性被放置到前端网络中。

因此，本算法的流程如下：1）将所有的流根据其相对应的前端路径空闲带宽大小进行升序排序。2）依次放置已经排序的流。2-1）对于流 f，其对应的前端与后端网络路径为分别为 P_f 和 P_b。假设 P_f 和 P_b 上已经放置了 N_f 和 N_b 个大流，且 P_f 上的空闲带宽为 F。如果 $F > F_{thr}$ 并且 $N_f = 0, (N_b + 1) * R_{max} > B$，那么，该流 f 将

算法 3.1 流量放置算法

输入： 需要放置的流的集合, F

输出： 经过放置后的流的集合, F_s

```
1: 按照流所在路径上前端网络中的空闲带宽升序排序，得到 F_s
2: for each f ∈ F_s, in increasing order do
3:     if f.P_f.free > F_thr 且 f.P_f.num == 0 且 (f.P_b.num + 1) * R_max > B then
4:         f.place ← front
5:         f.P_f.free− = f.demand
6:     else
7:         if f.P_f.free > f.P_b.free then
8:             f.place ← front
9:             f.P_f.free− = f.demand
10:        else
11:            f.place ← back
12:            f.P_b.free− = f.demand
13:        end if
14:    end if
15: end for
16: return
```

被放置在前端网络。否则，将流 f 放置在具有更大空闲带宽的网络中。2-2）更新相关网络路径的空闲带宽。

其中 R_{max} 是根据经验设定的静态值，例如在千兆网络中，一个 TCP 流的速率可以达到接近链路带宽，因此 $R_{max} = 900Mbps$. 默认地，$F_{thr} = 10\% \cdot B$。本路由放置算法并不保证放置结果是最优的，但是具有很小的开销及很高的效率。路由放置算法的伪代码如算法3.1所示。

3.2.3 流量隔离机制

MAX 需要保证后端网络的流量在使用前端网络时不会影响到客户请求流量，导致客户请求性能下降。通过给前端流量与后端流量赋予不同的优先级可以解决这个问题。为此，在生成流表的时候设置两种不同优先级的队列。在前端网络中，客户请求的流量将进入到高优先级的队列中，而来自后端网络的流量（例如恢复流量、重均衡流量等）将进入到低优先级的队列中。通过设置不同的优先级，来自后端网络流量将不能影响到用户请求的性能，使得后端流量只能利用前端网络中的空闲带宽。

在存储节点上，可以使用软件功能在 vSwitch 的端口上加入优先级队列。例如 OpenvSwitch[66] 提供了优先队列的功能，则可以直接使用“ovs-vsctl”命令创建优先

级队列。大多数情况下，数据中心内部的交换机都支持优先级队列。因此可以直接使用交换机的优先级队列实现优先级转发。传统的数据中心交换机中通过匹配 IP 报文中的 DSCP 字段来实现优先级转发。因此，可以在 vSwitch 中修改各个报文的 DSCP 字段实现不同流量优先级的控制，例如，通过 OpenFlow 可以直接修改该字段。对于 SDN 交换机，可以使用“set_queue”功能对流的优先级进行控制。

然而，虽然当前的数据中心交换机大部分都支持优先级队列功能，但是也存在不支持该功能的情况。若前端网络中的交换机不支持优先级队列，那么就有可能会导致一定的性能干扰。存储系统中的前端请求主要分为两类，包括读请求与写请求。对于读请求，网络流量主要为上行流量，由于在 vSwitch 中具有优先级队列，因此后端网络依然不会影响到前端网络的性能。而对于写请求，由于无法保证在前端流量在网络内部下行流量的优先转发，可能会导致后端流量影响到前端网络的性能。然而对于写请求，性能的瓶颈在于后端网络，因此调度对于写操作的性能影响不大。并且，MAX 会实时地对流量进行重新放置，在前端网络无空闲带宽时会及时地将流量调度回后端网络中。

MAX 的原型在 Ubuntu 操作系统上实现。其主要包括两部分：调度层（虚拟交换机）与网络控制器层。虚拟交换机使用 OpenvSwitch 2.4.0①，控制器使用 RYU 4.8②。在虚拟交换机上，默认地，从 vNIC1 进入的流量将被转发到 eth0（前端网络），而从 vNIC2 进入的流量将被转发到 eth1（后端网络）。当后端网络的流量需要被调度到前端网络时，网络控制器将会在 vSwitch 上安装相应的流表，将相应的流转发到 eth0 端口。考虑到前端网络与后端网络的 IP 地址可能并不属于同一个子网，因此，MAX 将会把被调度的流量的 IP 地址修改成前端网络的 IP 地址。为了区分前端网络中的前端流量与后端流量，使用 MPLS（多协议标签交换，Multi-Protocol Label Switching）标志[73] 标记来区分不同的流量。

同时，MAX 需要给两个虚拟端口分配两个新的 IP 地址，称为虚拟 IP 地址。因此需要处理虚拟 IP 地址之间的连通性问题。虚拟 IP 地址可以与物理 IP 地址处于不同网段，以避免对物理 IP 地址空间的占用。因此，需要在虚拟交换机上实现虚拟 IP

① http://openvswitch.org/

② http://osrg.github.io/ryu

与物理 IP 之间的 NAT（网络地址转换，Network Address Translation）转换工作，从而实现不同结点之间的虚拟 IP 地址的互通。由于存储系统的存储节点上开启的监听端口并非一个固定端口，因此不能根据端口来判断一个报文是发向存储节点（物理 IP 地址）还是存储系统（虚拟 IP 地址）。为了解决这个问题，同样使用 MPLS 标记来区分这两种流量。这里将存储节点上的流量分为三类：存储系统前端网络的流量，存储系统后端网络流量与其它流量。前端网络流量与后端网络流量是由虚拟 IP 地址发出或者发向虚拟 IP 地址的流量，而其它网络流量是由存储节点上其它应用发出的流量。因此，对于这三种流量，分别给它们一个不同的 MPLS 标识。在报文发出的时候，虚拟机交换机可以给这三种流量分别打上相应的 MPLS 标记值。在接收到报文时，虚拟交换机根据报文上不同的 MPLS 值将报文分别转发到相应的端口上去。具体地，假设存储节点的物理网卡 IP 地址为 10.1.0.1（前端网络）与 10.2.0.1（后端网络），而虚拟网卡的 IP 地址为 11.1.0.1（前端网络）与 11.2.0.1（后端网络）。在虚拟交换机上，所有源 IP 地址为 11.1.0.1 的报文都将设置 MPLS 为 X，源地址为 11.2.0.1 的报文都将设置 MPLS 为 Y。同时，对于所有 MPLS 为 X 的报文都将转发到前端网络虚拟网卡，MPLS 为 Y 的报文转发到后端网络虚拟网卡。

3.3 动态网络流量调度在存储系统中的应用实例

本节以当前最为流行的开源分布式存储系统 Ceph 作为 MAX 的应用实例，通过具体地分析分布式存储系统的流量模式来分析 MAX 的应用场景。为了提升系统的性能和安全性，Ceph 推荐部署两个独立的网络，使得前端请求的流量与后端系统内部的流量隔离开来，从而即可以提升请求的性能，也可以提升恢复与重均衡的性能。下面将分析 Ceph 在写操作、恢复、重均衡操作时 Ceph 在两个网络中的流量状态。

（1）写操作

为了保证数据的可靠性，Ceph 支持两种数据冗余方式，包括多副本和纠删码。对于多副本机制，Ceph 在写的过程中采用主-备副本的机制，即客户端将数据写入到一个主 OSD，然后由该主 OSD 将数据复制到其它的副本 OSD 上。因此，对于每个写入的操作，将会有一份数据流量经过前端网络，而有 $n-1$ 份数据流量经过后端网络

（其中 n 为总副本数）。后端网络中的带宽将会成为系统的整体写吞吐量的瓶颈。随着副本数的增加，系统受后端网络带宽影响越大。为了最大化系统的吞吐量，可以将部分后端网络中的流量调度到前端网络中，从而达到两个网络之间的负载均衡。

（2）恢复操作

数据恢复过程是 Ceph 集群内产生网络流量最多的活动之一，因此需要消耗大量的网络资源。Ceph 集群会实时地检查系统的状态，维护一个系统的 OSDMap。当 OSDMap 更新时，Ceph 会更新所有 PG 的状态，进入 Peering 阶段。当一个 OSD 接收到 OSDMap 更新时，该 OSD 节点上的所有 OSD 列表有更新的 PG 都将需要重新同步信息 (re-peer)。对于一个 PG，其主 OSD 会计算出哪些 OSD 存在缺失对象，以及哪些对象已经缺失。在 Peering 阶段结束之后，所有存在缺失对象的 PG 将会重新恢复缺失对象。若主 OSD 上存在缺失对象，那么主 OSD 将从任意一个具有该对象的从 OSD 上拉取对象；若从 OSD 上存储缺失对象，那么主 OSD 将该对象推送到该从 OSD 上。对于副本模式，每份丢失的数据都将在后端网络中传输一次进行恢复，因此在后端网络中产生 1 倍的网络流量。而对于纠删码模式，当某个数据块丢失时，需要从其余所有具有该数据分块的 OSD 上获取数据块，从而需要在后端网络中产生 $k+m-1$ 倍的网络流量，其中 k 为分片数，m 为校验块数。

（3）重均衡操作

存储系统的扩容是非常重要的功能。在创建存储集群的时候往往无法精确地预测到应用或者业务在以后一段时间的存储容量，从而无法在创建的时候准确地配置系统的规模。另一方面，为了减少系统构建时的成本，用户往往不会在创建时就配置一个非常大的系统，而是采用在后期随着业务的增长而对系统做相应的扩展。当系统扩容或者缩容时，将会触发 Ceph 系统的重均衡机制。该操作也同样会导致 OSDMap 变更，从而触发集群内数据的移动。当有 OSD 节点加入或者退出集群时，Ceph 会通过 Crush 算法重新计算每个 PG 的 OSD 列表，对于 OSD 列表有更新的 PG，将会采用恢复的过程将数据重新迁移到新的 OSD 中。因此，对于有 OSD 节点加入到系统的情况，部分数据将会从原来的 OSD 上迁移到新加入的 OSD 中。对于有 OSD 节点退出集群的情况，退出的 OSD 上的数据将会从其在剩下的具有该数据的 OSD 上重新均衡到其它 OSD 节点上。

3.4 性能评价

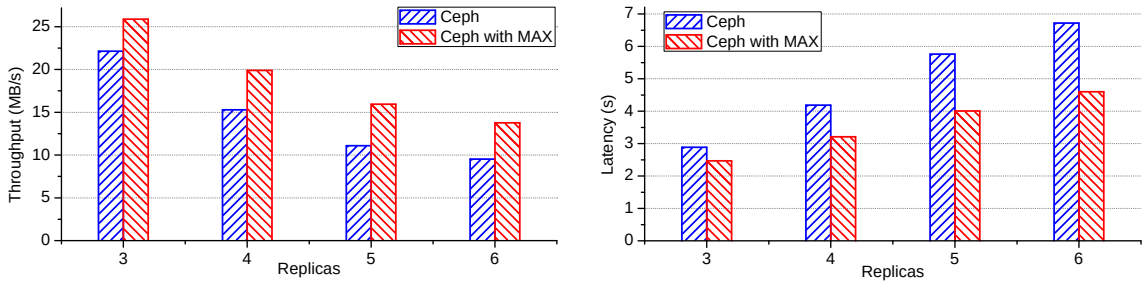
在测试过程中，假设存储系统中的网络带宽是系统的瓶颈。Ceph 测试集群使用 KVM[①]虚拟机搭建。虚拟机之间的网络互连拓扑为由 Openvswitch 2.4.0[②]组成。为了保证网络是系统中的瓶颈，将链路带宽设置为 100Mbps，而所有 KVM 的虚拟磁盘放置到 SSD 上（SSD 上的写带宽为 600MB/s）。本节分别测试了 Ceph 的写操作、恢复操作、重均衡过程在开启 MAX 和关闭 MAX（即为原始的 Ceph）时的性能。同时为了验证 MAX 是否会对客户请求带来性能影响，在运行前端应用负载时同时触发系统的恢复操作，从而对比开启 MAX 与关闭 MAX 时的应用性能与恢复性能。

3.4.1 基本测试

通过第3.3节的分析，可以发现在 MAX 可以优化后端网络流量比较重的操作。因此本节单独测试了 Ceph 系统中的写、恢复和重均衡操作。

（1）写操作

首先，测试了 Ceph 在不同的副本数的情况下，写操作的带宽与时延。测试环境包括 8 个 OSD，1 个 MON 和 1 个 Client。为了防止 Client 端的网络成为测试的瓶颈，将 Client 端的接入链路带宽设置为 1Gbps，而网络（前端和后端）的链路带宽为 100Mbps。在 Client 上运行“rados bench”命令进行写性能测试。



(a) 平均带宽　　(b) 平均时延

图 3-3 在不同的副本数情况下的平均带宽与平均时延

图3-3为 Ceph 在副本数为 3-6 个情况下的写操作平均带宽与平均时延。从测试结

① https://www.linux-kvm.org/page/Main_Page
② http://openvswitch.org/

果中可以看出，MAX 可以提升 Ceph 系统的整体写性能，提升写带宽约 16%-44%，并且副本数越多时性能提升的幅度越大。随着副本数目的增加，后端网络中将会产生越来越多的网络流量，使得前端网络与后端网络之间的负载变得越来越不平衡。后端网络的负载越来越高，而前端网络的将会变得空闲，从而导致系统的性能将会加剧下降。MAX 通过将后端的部分网络流量调度到前端网络中，从而使得两个网络中的负载变得更加平衡，最终将提升系统的性能。

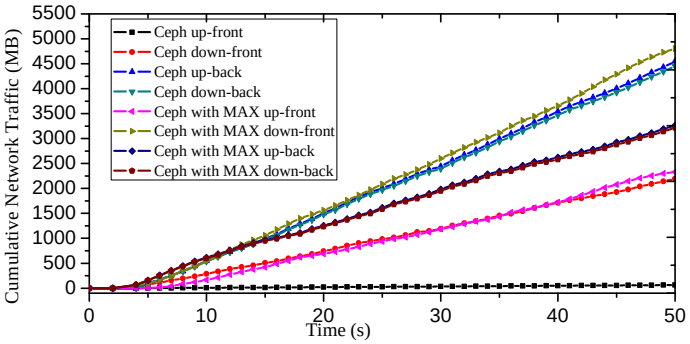
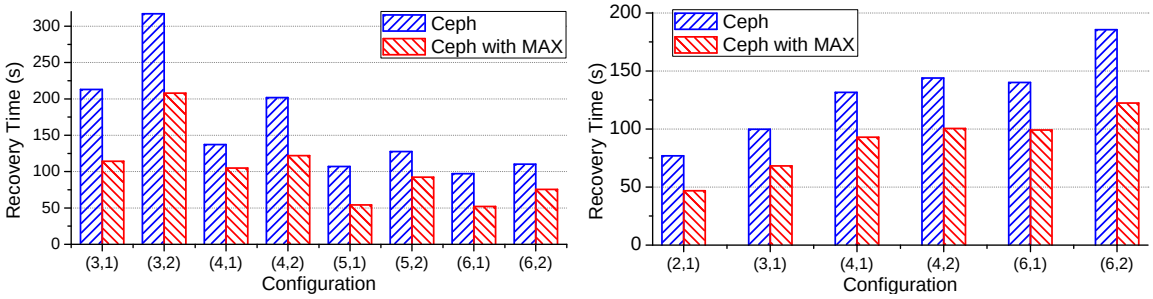


图 3-4 写操作测试过程中前端与后端网络中累积网络传输量对比

图3-4显示了在关闭与开启 MAX 时，两个网络中的吞吐量（副本数为 3）。在关闭 MAX 时，写操作的过程中前端网络的流量远小于后端网络，其中上行流量基本为 0，而下行流量是后端网络的 $1/(n-1)$ 倍（n 为副本数）。在开启 MAX 时，前端网络的流量更加接近于后端网络的流量。

（2）恢复操作

本节测试了 Ceph 存储系统在不同的设置时的恢复性能。分别测试了多副本与纠删码环境下的恢复性能。对于多副本的测试，副本数为 3。测试过程中 Ceph 的配置为 (M, N)，其中 M 为总 OSD 节点数，N 为失效的 OSD 数。测试开始时，将 N 个 OSD 设置为“down”状态，并向系统中写入 4G 数据（400 个对象）。写入完成之后，将 N 个被设置为“down”的 OSD 设置为“up”，统计系统恢复需要的时间。对于纠删码模式下的测试，所有的测试中总 OSD 数为 8，失效的 OSD 数为 1。分别测试了纠删码 (M, K) 时的恢复性能，其中 M 为数据块数目，K 为检验块数目。同样地，将 1 个 OSD 设置为“down”，然后向系统中写入 4G 数据（400 个对象）。写入完成之后，将被设置为“down”的 OSD 设置为“up”，计算恢复的时间。



(a) 副本模式下的恢复时间　　(b) 纠删码模式下的恢复时间

图 3-5 在不同的系统配置下的恢复时间测试

图3-5显示了在不同的配置下 Ceph 存储系统的恢复性能对比。从结果中可以看出，MAX 可以减少 Ceph 的恢复时间达 30%-46% 左右。随着存储系统的规模的增加，MAX 带来的性能提升趋于稳定。

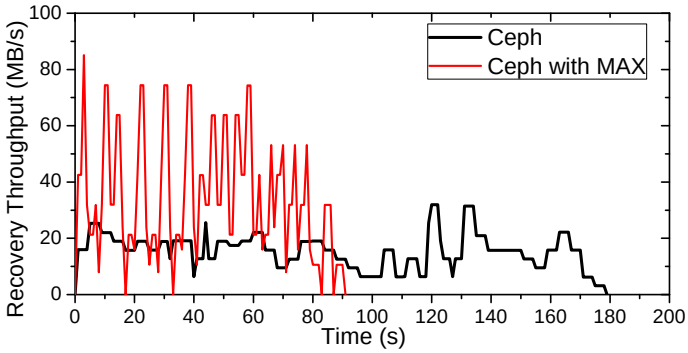


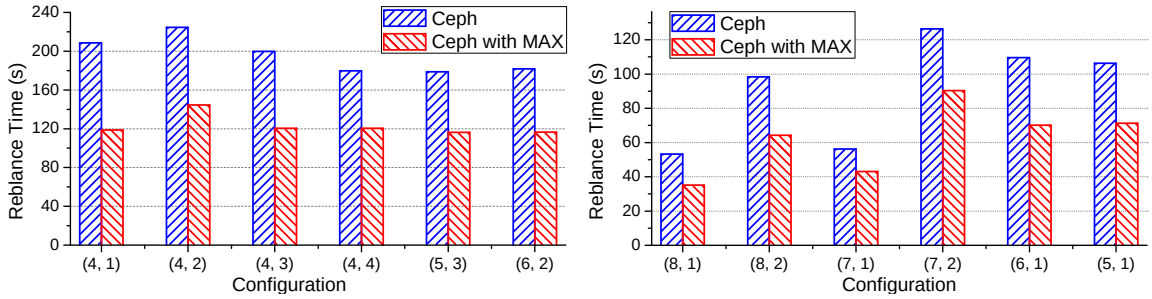
图 3-6 在（3，1）配置下恢复带宽对比

图3-6显示了（3，1）配置下副本模式在开启与关闭 MAX 时的恢复带宽对比。在无 MAX 时，即使前端网络无请求，Ceph 的恢复流量也只能使用后端网络资源。而当前端网络有空闲带宽时，MAX 可以将部分流量调度到前端网络中，从而提升系统中网络带宽利用率，进而提升恢复的性能。

（3）重均衡操作

本节测试了向 Ceph 加入以及删除 OSD 时 Ceph 系统重均衡所需的时间。测试系统采用三副本模式。分别测试了不同系统配置时的重均衡时间。系统配置为 (M, N)，其中 M 为系统原有 OSD 数，N 为增加或者删除的 OSD 数。对于每个实验，先搭建 M 个 OSD 的 Ceph 集群，向 Ceph 里写入 4GB 数据。然后再一次性往系统里增加 N 个 OSD，或者从系统中删除 N 个 OSD，之后统计系统完成重均衡所需的时间。

图3-7显示了在新增或者删除 OSD 时，Ceph 存储系统进行数据重均衡所需的时



(a) 增加 OSD 时重均衡时间　　(b) 删除 OSD 时重均衡时间

图 3-7 在不同的系统配置时重均衡时间测试

间。与恢复测试中得到的结果类似地，MAX 通过提升网络资源的利用率可以有效地降低系统重均衡的时间，达到约 30%-40% 的时间节省。

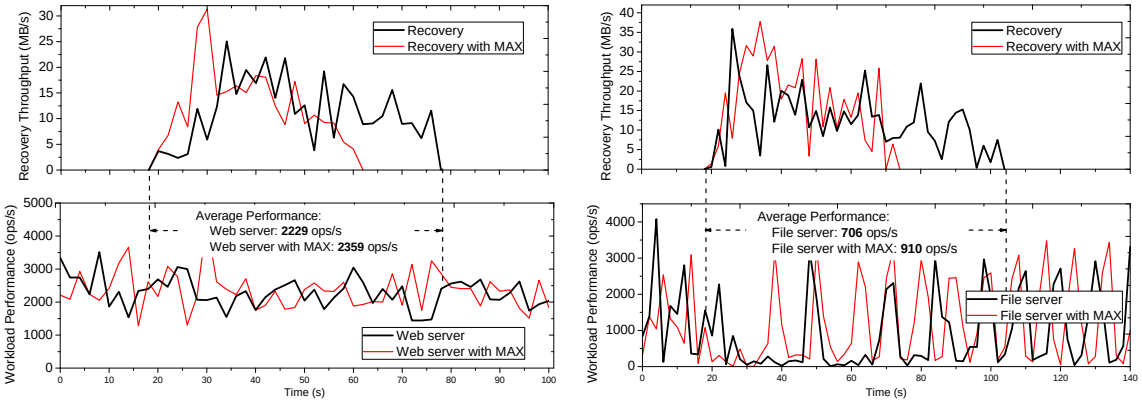
3.4.2 用户请求与恢复共存

MAX 主要的优点是能够充分发现并利用存储系统中空间的网络资源。尽管在很多情况下，系统恢复或者重均衡时前端网络没有客户端请求，然而更多的时候，系统恢复与客户请求是共存的。因此，需要确保 MAX 不会影响到客户请求的性能。MAX 采用一种基于优先级的方案来避免后端网络流量在前端网络中影响用户请求的性能。在本测试中，着重探索在后端的恢复操作与前端的应用请求共存时 MAX 的结果。

本节使用 Filebench[①]测试了两种典型的应用负载，包括 Web server 与 File server。对于 Web server 负载，其平均文件大小 $meanfilesize = 16KB$，读写比例为 10 : 1。对于 File Server 负载，其平均文件大小 $meanfilesize = 128KB$，读写比例为 1 : 2。测试环境中搭建了 6 个 OSD 节点的 Ceph 集群，在一个客户端挂载了 CephFS。然后使用 Filebench 来模拟 File server 与 Web server 两种应用。测试过程中，将一个 OSD 设置为“down”，然后开始运行 Filebench 测试。在 20s 之后，将被设置为“down”的 OSD 设置为“up”，从而系统自动会触发恢复操作。

图3-8显示了测试过程中负载的 IOPS 与恢复带宽。从图中可以看出，在恢复操作与这两种前端负载共存的情况下，MAX 依然可以降低 Ceph 系统的恢复时间，并且对前端应用不会造成明显的性能下降。具体地，在运行 Web server 和 File server 负载时，MAX 分别降低了 28% 和 36% 的系统恢复时间。由于这两种负载的平均文件

① https://github.com/filebench/filebench/wiki



(a) 运行 Web server 负载进行恢复操作　　(b) 运行 File server 负载进行恢复操作

图 3-8 前端负载与恢复共存时，前端负载的 IOPS 与恢复带宽

大小都比较小，因此它们都没有很高的带宽需求，即在前端网络上将留有大量的空闲带宽。MAX 正是通过充分利用这些前端网络中的空闲带宽来提升恢复的性能。

当恢复刚被触发时，两种负载都表现出了性能下降。然而，在开启 MAX 的情况下，应用负载的性能下降来得更加早些，这可能是由于 MAX 使得 Ceph 的“Peering”阶段更加早的完成。在开启 MAX 时，应用负载的平均 IOPS 并没有比没有启动 MAX 时的平均 IOPS 更低，相反，MAX 反而提升了 File server 的平均 IOPS（约 28% 的提升）。这是因为 File server 的读写比例为 1 : 2，是以写操作为主导的负载类型。而在 Ceph 存储系统中，当写操作的目标对象处于降级的状态时，将会阻塞写操作而优先恢复该对象。在对象的恢复操作完成之后才能继续进行写操作。因此，在系统正在进行数据恢复的时，写操作的性能将会严重下降。而 MAX 可以提升恢复操作的性能，这样也间接地提升了恢复时用户写操作的性能。对于 Web server 负载，由于其是读为主导的负载类型，因此 MAX 对其的性能几乎没影响。

3.4.3 开销分析

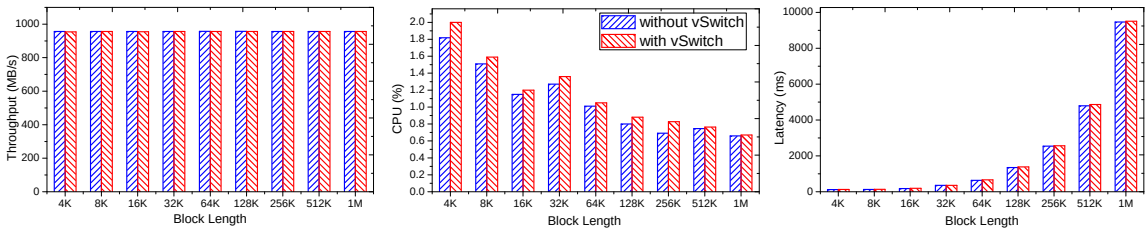
（1）控制器开销

MAX 使用 Hedera[17] 中提出的 Demand Estimation 算法进行大流的需求估计。该算法的时间复杂度为 $O(|F|)$，F 为大流的数量。由算法3.1可以看出，本章提出的流量放置算法的复杂度也为 $O(|F|)$，F 为网络中的大流数量。因此控制器端的时间复

杂度为 $O(|F|)$。从文献[17]中可以得到使用 Demand Estimation 算法，单个服务器在 5ms 内可以计算 25 万条大流的需求。因此一个普通的服务器即可满足该计算需求。

（2）转发开销

MAX 需要在存储节点上加入一个虚拟交换机。为了评估其引入的开销，本节测试了在存储节点上加入一个虚拟交换机（vSwitch）与不加虚拟交换机（vSwitch）时的网络传输性能、CPU 利用率以及请求/回复（request/response）的时延，如图3-9所示。



(a) 传输带宽对比　　(b) CPU 利用率对比　　(c) 传输时延对比

图 3-9 存储节点上加入一个额外的虚拟交换机（vSwitch) 时的开销

该测试使用 netperf 2.6.0[①]完成。在时延的测试中，测试了从发送端发送“块大小”字节数给接收端，直到接收端返回“块大小”字节数给发送端时所需的时间。另外，还测试了在加入一个 vSwitch 与不加 vSwitch 时，两个存储节点之间的“ping”时延，分别为 0.493ms 与 0.468ms。从结果中可以看出，在存储节点上加入一个 vSwitch 时对系统引入的带宽开销为 0.1% 左右，时延开销小于 5%。

3.5 本章小结

为了解决分布式存储系统在部署两个网络（前端与后端网络）时存在的网络资源浪费问题，本章提出 MAX，一种可以跨前端与后端网络之间的动态流量调度方案。MAX 可以自动地探测前端网络中的空闲带宽，并且动态地调度后端网络流量到前端网络中以充分利用其空闲带宽，从而可以提升后端活动（如恢复与重均衡操作）的性能。为了达到易于部署的效果，MAX 在网络层进行调度，从而不需要修改任何存储

① http://www.netperf.org/netperf/

系统的代码实现。具体地，在每个存储节点上加入一个虚拟交换机，并将存储系统的所有流量都导入到该虚拟交换机。流量由虚拟交换机转发之后再进入到相应的物理网卡。通过采用集中化的控制器来控制每个存储节点上虚拟交换机上的路由，从而可以实现跨两个网络之间的流量调度。为了避免两种流量共享前端网络时导致的相互干扰，使得存储系统的用户请求受到后台流量的影响，MAX 采用优先级的方式进行转发。给用户请求赋予更高的优先级可以在不影响前端请求性能的情况下充分利用前端网络中的空闲带宽。在 Ceph 存储系统上进行测试，结果表明 MAX 可以极大地提升系统恢复与重均衡的性能。

4 基于高效匿名通信的存储系统安全性优化方法

随着分布式存储系统规模的扩长，来自网络和恶意用户的安全威胁将极大地增加。一方面，攻击者可能发起针对存储系统以破坏为目标的攻击（例如 DDos 攻击），以使得存储系统无法正常对外提供服务。另一方面，攻击者可能发起针对存储系统中数据窃取的攻击，从而获得存储系统中的敏感数据。为了提升分布式存储系统抵御这类攻击的能力，本章提出基于匿名通信的分布式存储系统（匿名存储系统），通过使用匿名通信来保护分布式存储系统中各个节点的身份与位置信息，从而提升系统被攻击的难度。通过让存储节点匿名的方式，使得攻击者难以定位到需要攻击的目标节点，例如元数据存储节点或者存储敏感数据的存储节点，从而提升了攻击的难度。然而，现有的匿名通信方案存在着巨大的开销，难以在分布式存储系统中使用。为此，本章针对基于 SDN 的数据中心环境，提出一种基于全局路由冲突避免机制的匿名通信方案（MIC，Mimic Channel）。其主要思想是通过在交换机上修改报文的源、目标地址来隐藏报文的发送者与接收者，从而达到匿名通信的目标。利用 MIC 实现了匿名存储系统 CapFS-M，可以在引入极小开销的情况下提升存储系统的安全性。

4.1 数据中心内高效匿名通信需求

随着数据中心规模的扩张，数据中心面临着越来越多的来自内部的安全威胁。据 IBM 安全报告[①]指出，2014 年记录到 55% 的攻击来自具有内部访问的人员。同时，攻击者在进行攻击时，往往都需要通过一定的手段进入到目标的内部网络。例如，2013 年的 Target 公司信用卡泄露事件中，攻击者先通过窃取其空调系统供应商的凭证侵入到其内部网络，再窃取了其 40M 信用卡数据[②]。因此，更多的关注需要放在保护数据中心内部网络安全上。

用户通常在云数据中心内部署应用，以达到快速上线、减少成本的目的，如 Web

① http://www-03.ibm.com/security/data-breach/2015-cyber-security-index.html

② https://aroundcyber.files.wordpress.com/2014/09/aorato-target-report.pdf

服务、大数据处理系统和分布式存储系统等。然而这些用户系统在数据中心内部的通信存在极大的安全威胁。黑客、内部人员或者恶意用户可能会在网络中间节点进行监听、分析网络传输的数据。一方面，如果用户的数据传输过程中未加密，那么这可能造成用户的数据被窃取。另一方面，即使这些数据都经过加密之后再进行传输，攻击者也可以通过检查报文中未加密的 IP 地址和流量模式等信息发动流量分析攻击。例如，攻击者可以通过报文中的源、目标地址信息获得本次通信的发送者和接收者，再通过分析它们之间通信的速率，数据长度等信息分析出它们可能正在进行何种操作。进一步地，攻击者可以分析出哪些节点属于某一个用户，在这些节点上部署了什么系统以及该该系统的规模。如果攻击者的目的是使某个用户系统崩溃，那么可以分析出该系统的一些关键节点（例如分布式存储系统中的 MDS 节点等），再对其发动主动攻击，如 DoS/DDoS①、Worms②等。

在互联网环境下，研究人员提出匿名通信方案来防止流量分析攻击，保护用户身份信息。这些匿名通信方案通过构建由多个转发节点组成的覆盖网络（Overlay Network）来达到匿名目的，如 Mixmaster[74]、Mixminon[75]、Crowds[76]、Tor[77] 和 Dissent[78] 等。同样地，匿名通信在数据中心内也能够起到重要的作用。以下通过 2013 年 Target 公司的信用卡泄露事件为例说明匿名通信如何能够增加攻击的难度。该攻击的流程[79] 如表4.1所示。

在步骤 4 中，攻击者通过查询 Active Directory③来定位相关的目标服务器，并通过查询 DNS 服务器来获取相应的 IP 地址。在步骤 7.1 中，攻击者通过在一系列的节点之间进行扩散来绕过防火墙和其它的网络安全设施。该过程中利用“Angry IP Scanner④”和“Port Forwarding utility⑤”工具完成，而其基本的原理是扫描网络中的 IP 地址与端口，以获得通过抵达目标节点的可用的路径。由此可以看出，步骤 7.1 中能够最终访问到目标服务器的前提条件是获得了目标节点的 IP 地址。如果攻击者无法获得目标服务器的 IP 地址，那么其现有的攻击方法将难以访问到目标服务器。匿

① https://en.wikipedia.org/wiki/Denial-of-service_attack
② https://en.wikipedia.org/wiki/Computer_worm
③ https://en.wikipedia.org/wiki/Active_Directory
④ http://angryip.org
⑤ https://portforward.com

表 4.1 2013 年 Target 公司的信用卡泄露事件攻击流程

步骤 1：安装恶意软件以窃取登入凭证
步骤 2：利用窃取的凭证进入内部网络
步骤 3：发现一个网页应用的漏洞
步骤 4：搜索相关的攻击目标以进行扩散
步骤 5：从 Domain 管理员上窃取访问令牌
步骤 6：利用窃取的访问令牌创建新的 Domain 管理员账户
步骤 7：利用新创建的账户扩散到相关的计算机
步骤 7.1：绕过防火墙与其他网络安全设施
步骤 7.2：在不同的机器上运行远程程序
步骤 8：窃取 70M PII，但未找到信用卡
步骤 9：安装恶意软件，成功窃取 40M 信用卡
步骤 10：将窃取的数据通过网络共享发送出去
步骤 11：将窃取的数据通过 FTP 发送

名通信的目标正是保护通信过程中通信双方的身份信息，最主要的即为主机的 IP 地址。如果数据中心内使用了匿名通信，那么攻击者将无法轻易地获得主机的 IP 地址信息，从而可以有效地增加攻击的难度。

同样地，在分布式存储系统中，通过匿名通信的方式来保护系统中每个节点的身份信息也显得尤为重要。一方面，攻击者难以在网络中分析出系统的节点信息，从而无法定位到存储系统中的一些重要节点，例如元数据服务器等。另一方面，恶意用户无法通过侵入某个客户端节点来定位敏感数据所在存储节点的身份信息。

数据中心内部存在巨大的安全隐患，然而目前并没有任何部署于数据中心内部的匿名通信方案。攻击者可以通过网络中任何一点获得大量的网络通信信息。例如，黑客可以通过“telnet attacks”来控制某个交换机，从而窃取经过该交换机的报文，或者发动流量分析攻击。在一些以服务器为中心的网络拓扑中，如 Bcube[80] 等，黑客可

以通过控制某一服务器节点，分析经过其的所有报文。在虚拟化的云数据中心内，恶意用户可以通过虚拟机逃逸（vm escape）[81] 来攻击宿主机，然后可以很容易地监听或者分析同一宿主机上的其它所有虚拟机的流量。报文头部中很多信息对攻击者来说是非常有用的，例如“端口”信息往往可以揭露出上层应用的类型（端口 80 一般为网页服务）。另外，现有的云数据中心的设计存在不足。例如，文献[82]指出 EC2① 内虚拟机的 IP 分配是静态的。任何人都可以根据 EC2 内虚拟机的 IP 信息映射出虚拟机的类型和所属区域。因此，保护数据中心内主机身份信息显得非常重要，需要设计一种适合于数据中心内的匿名通信方案。

4.2 匿名通信方案的相关研究

为保护用户的身份信息或者服务提供者的身份信息和抵御“流量分析”攻击，匿名通信被广泛研究。传统的匿名通信方案主要基于 Mix-Net[83]，DC-Net[84]，verifiable shuffles[85,86] 和广播（broadcast）来实现的。Mix-Net 通过对报文在多个“mix”节点上进行加密/解密、延迟、乱序或批处理等操作来隐藏发送端与接收端的关联。DC-Net 和 verifiable shuffles 使得在不进行批处理的情况下保证高匿名性和抗流量分析攻击。现有的匿名通信方案按照延迟可以分为两类：高延迟匿名方案和低延迟匿名方案。

（1）高延迟匿名方案

高延迟匿名方案主要针对匿名性要求高、实时性要求较低的应用，如 Email 等，主要包括 Babel[87]、Mixmaster[74]、Mixminion[75]。这类系统通常是基于 Mix-Net，在“mix”节点（转发节点）上对报文进行加密/解密、延迟和批处理以达到最大化的匿名性和抗流量分析攻击能力。但是这些系统需要牺牲大量的性能，因此并不适合交互式或在线应用。

（2）低延迟匿名方案

低延迟匿名方案主要面向对传输的实时性有较高要求的应用，如网页浏览和聊天等。对于基于 Mix-Net 的低延迟匿名方案，为了达到低延迟传输，一般不对报文进行延迟和批处理，因此一般无法抵御全局的攻击者。对于基于 DC-Net 和广播（broadcast）

① http://aws.amazon.com/ec2/

的低延迟匿名方案，一般可以抵抗很强的攻击者，但是一般需要牺牲系统的带宽或者扩展性。

Anonymizer①是一个最简单的低时延匿名方案，只有一个中间代理。由于所有报文都经过同一个代理转发，因此只能忍受较松的威胁模型。一旦攻击者控制该代理或者分析其输入和输出流量，那么将会打破其匿名性。Onion routing[88]（洋葱路由）是早期的一个针对实时应用的 Mix-Net 变体。其在传输数据之前，由发送端指定多个转发节点，并建立起与接收端之间的双向环路。环路建立之后，发送端可以向接收端传输数据。发送端对报文进行多层加密，每个转发节点对报文进行一次解密，并转发给下一个节点。每个转发节点只知道它在环路中的上一跳和下一跳，而并不知道流量的真实发送者与接收者。Tor[77] 是第二代基于 Onion Routing 算法的匿名通信方案，通过自愿者贡献转发节点与网络带宽的方式广泛部署在互联网中，是目前最为流行的匿名通信方案[89,90]。Tor 在原始的洋葱路由之上增加了中心化的目录服务器来维护转发节点列表和集合点来实现隐藏服务（即实现接收端匿名）。

P2P 架构下的匿名方案中，所有节点既可以是流量的发起者或接收者，也可以是转发者。Crowds[76] 利用大量的节点来隐藏流量真实的发送者。每个转发节点可以自主地决定是否将报文提交给最终的接收端，或者将其转发给下一个转发成员。因此所有节点都不知道它的上一跳是流量的发起者还是转发者。MorphMix[91] 使得任意节点可以轻易地加入到系统中，同时提供冲突检测机制来鉴别被控制的路径。Tarzan[92] 利用“cover traffic”来扰乱流量模式，使得全局攻击者无法鉴别流量的发起者与接收者，从而提供低延迟的端对端的匿名通信。Aqua[93] 专注于为 Bit Torrent 提供低延迟、强匿名性的传输服务。其在核心链路上采用加密、“chaffed flows”来实现统一的传输速率，在边界链路上利用大量客户端类似的流量模式来达到 k-anonymity（即攻击者无法确定 k 个客户端中哪个正在进行通信）。Herd[94] 为 VoIP 系统提供可扩展的抗流量分析的匿名服务。

Hordes[95]、Herbivore[96]、P5[97] 采用组播或者广播的机制来实现匿名通信。Hordes 基于 Crowds 并且通过组播转发的方式来隐藏发送者。P5 通过构建层级的广播通道架构来提高扩展性，其中每个客户端可以加入两到三个不同的通道。Herbivore 基于 DC-

① https://www.anonymizer.com/

Net 和广播机制来构建高效的匿名方案。Dissent[78] 基于 DC-Nets 和 verifiable shuffles 实现具有强匿名性的低延迟、高带宽、可扩展的匿名通信。其可以提供可证明的匿名，在保证用户强匿名性的同时避免滥用行为。Information Slicing[98] 通过多路传输和秘密共享的方式，企图在不使用密钥的情况下实现匿名通信。LAP[99] 提供轻量级的匿名服务保证较低的延迟。其采用报文自带路由和最小化冗余路径的方式来减少中间路由需要保证的状态和端到端传输时延。

上述所有匿名通信方案都是针对互联网环境设计的，其实现都是基于覆盖网络（Overlay），主要针对 Email、Web 浏览、P2P 等应用。然而，在数据中心内部，应用对网络传输的时延、带宽需求远高于前面这些应用。基于 Overlay 的匿名通信方案不能满足云数据中心环境的应用需求，因此，需要设计一种适用于数据中心环境的匿名通信方案。数据中心内的匿名通信方案需要具有极低的开销，因此也使得设计面临巨大的挑战。幸运的是，数据中心环境相比于互联网环境具有更好的可控性，面临的安全威胁要比互联网环境少得多。例如，Tor 中采用自愿者提供转发的机制，这样恶意用户可以很简单地伪装成一个转发节点。而数据中心内的节点都是比较受控的，攻击者一般需要提前利用其它攻击方式才能控制交换机或者主机。

为了满足数据中心内应用对匿名通信的需求，本章提出拟态通道（Mimic Channel，MIC），一种网内（in-network）高效率匿名通信方案，具有非覆盖网络的架构，可以有效减少资源占用。

4.3 高效匿名通信方案问题界定

本章研究数据中心内匿名通信方案，以保证在数据中心内节点之间通信的匿名性。考虑到数据中心的特点，需要在提供合理的安全性的前提下最小化性能开销。

4.3.1 系统模型

MIC 的基本思想是通过在多个交换机节点（而非主机节点）上变换数据报文源、目标地址（包括 MAC 地址，IP 地址，端口号等）来隐藏报文的真实发送者与接收者。这里称一个经过多次地址变换的流为拟态流（m-flow），进行地址变换的节点为拟态

节点（Mimic Node，MN）。拟态节点可以当作一个传统匿名方案中轻量化的“mix”或者“relay”节点，在拟态节点上变换后的地址为拟态地址（m-address）。

本方案为针对基于 SDN 的网络环境设计，文中交换机统指 SDN 交换机，具有修改报文头部信息的功能。MIC 匿名通信方案为典型的 C/S 模型，包括客户端、拟态节点和拟态控制器。客户端即为可以发送或者接收数据的节点。拟态节点为网络中任意的交换机节点，每个拟态流上的拟态节点由拟态控制器指定。拟态控制器位于 SDN 控制器上，计算生成通信所需要的流表。MIC 的系统模型如图4-1所示。

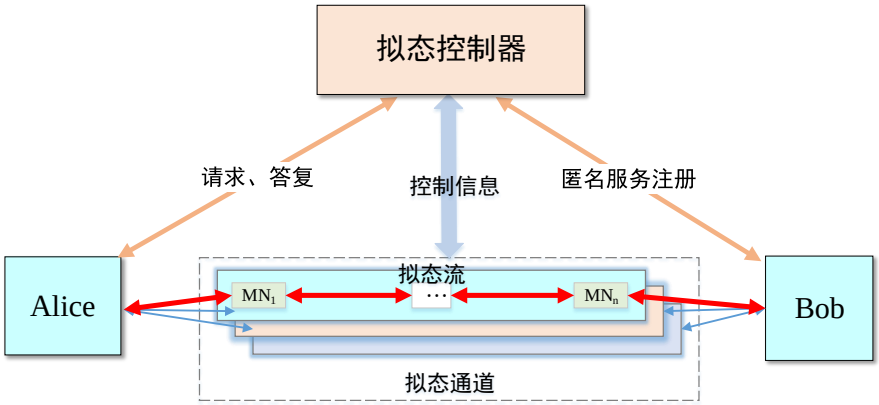


图 4-1 MIC 的系统模型

客户端为进行数据发送或者接收的节点，包括发送端和接收端。发送端可以主动地建立与接收端之间的拟态通道，通道建立之后，双方可以进行匿名通信。发送端可以指定一个接收端和通道的子流数目，向拟态控制器发起请求。拟态控制器则会生成相应的流表，其中在拟态节点上的流表会修改报文的头部信息。

拟态节点一般为 SDN 交换机，其可以对网络报文的头部信息进行修改，以隐藏报文真实的发起端与接收端。拟态节点是一个轻量级的“mix”节点，并不能实现传统匿名方案中“mix”节点所具有的如对报文加密/解密、改变报文长度、延迟乱序等操作。然而 MIC 设计的宗旨是最小化开销，因此并不需要在拟态节点上对报文进行加密/解密等操作，并且可以实现满足数据中心需求的匿名性。

拟态控制器位于 SDN 控制器，负责生成、更新和管理每个拟态通道及其子流。其知道所有拟态通道的真实参与者，所有的拟态节点和所有的流表信息。因此拟态控制器是 MIC 的核心，是实现匿名通信的关键。

拟态通道包括两个阶段，通道建立与通道转发。通道建立过程中发送端需要向拟态控制器发送请求，其通信过程使用公钥加密以保证安全性。拟态控制器在接收到请求之后生成相应的流表，其中控制器与所有交换机都通过加密通道进行通信。

4.3.2 威胁模型

假设攻击者的目标是打破通信的匿名性，分析出通信的发送端与接收端。攻击者可以控制交换机节点、发送端或者接收端；或者在网络中对某些端口进行监听，分析流量模式，具体如下：

控制交换机：攻击者可能控制一个或者多个交换机，其中可能是拟态节点或者普通交换机，企图通过分析经过其所控制的交换机的流量，分析出哪些节点之间正在通信，以及相应的通信模式（如通信的报文长度、速率等）。

控制客户端：攻击者可能控制了发送端（或者接收端），并企图获得接收端（或者发送端）的地址信息。例如，黑客控制了分布式存储系统中的某一个客户端节点，并企图获得该系统中的其它节点（如元数据服务器、数据存储服务器）的信息，从而确定下一步攻击的目标。

流量监控：攻击者可能在网络中对某些端口进行监听，获取并分析其流量。例如，数据中心内交换机一般具有端口镜像功能 (port mirroring[①])，供入侵检测系统（IDS，Intrusion Detection System[②]）进行流量监控分析。攻击者可能利用端口镜像功能对网络进行监控，或者控制了已有的入侵检测系统，获得流量监控的能力。

与大部分现有的匿名通信方案类似，MIC 假设攻击者只能控制部分交换机，并不具有全局的网络视图。并且只能在部分节点上进行监控，而不能监控所有节点上的流量。这种假设是比较实际的，因为攻击者必须通过其它攻击手段才能控制交换机，所以控制交换机对攻击者来说有一定的困难。另一方面，入侵检测系统 IDS 为了减少开销，一般只在少数几个节点上开启流量监控，并不会在所有的节点上开启监控。因此，攻击者若要监控所有节点依然面临巨大的困难。

MIC 的目标是打破通信参与者之间的关联，实现会话不可链接性。同时 MIC 也

① https://en.wikipedia.org/wiki/Port_mirroring
② https://en.wikipedia.org/wiki/Intrusion_detection_system

设计用于抵御部分流量分析攻击，比如基于报文长度的流量分析攻击。另外，MIC 需要达到以下设计目标：

高性能： 数据中心大部分为性能敏感的应用要求低延迟和高带宽。分布式存储系统对网络传输的延迟和带宽都具有很高的要求。

易于部署： 需要能够快速简易部署，不需要修改交换机实现，内核协议等等。

在设计 MIC 方案时，需要作以下假设：（1）假设 SDN 控制器是安全的。这个假设在实际中是成立的，因为 SDN 控制器是 SDN 网络中的核心，一旦 SDN 控制器被攻击，那么整个网络都将瘫痪。（2）假设 SDN 控制器与 SDN 交换机之间的通信通道为安全通道，SDN 控制器通道安全通道给 SDN 交换机下发流表。

4.4 高效匿名通信方案

4.4.1 总体设计

MIC 是基于 SDN 环境下的匿名通信方案，其核心思想是在多个交换机上修改报文的头部信息，打破会话的可链接性，达到匿名通信的目的。其由拟态控制器生成每个拟态流的路由信息，包括转发路径、拟态节点及相应的地址变换信息。只有拟态控制器知道每条流的完整信息，而每个转发节点（包括普通的交换机节点和拟态节点）只知道报文转发的下一跳或者头部变换信息（如果有的话），而并不知道报文的真实源、目标地址。

MIC 的设计面临着两大挑战。**首先**，为达到更好的匿名效果，拟态地址为与真实地址同一网段的任意地址，因此需要处理不同拟态流（或者非拟态流）之间可能出现的路由冲突问题。为了避免路由冲突，设计了一种全局路由冲突避免机制，其核心是通过一种拟态地址生成算法来将不同拟态流的拟态地址映射到不同的地址空间上去，从而避免发生冲突。**其次**，为了增加 MIC 的易用性，减少部署难度，需要在不修改交换机实现的前提下实现匿名通信方案，同时保证一定的抵御流量分析攻击能力。为增强 MIC 抗流量分析攻击的能力，提出多拟态流机制与局部多播机制。与传统的匿名方案相似地，MIC 包括两个阶段，即通道建立与数据传输。

4.4.1.1 通道建立阶段

通道建立过程中，需要建立一组双向的传输路径，每个方向上包括指定数目的拟态流，每个拟态流都具有独立的传输路径、独立的拟态节点和相应的拟态地址。具体地，发送端发起通道建立请求，拟态控制器接收请求之后生成相应的路由流表。请求报文中需要指定拟态流和拟态节点的数目。拟态控制器根据指定的拟态节点数目，在每个拟态流的转发路径上选取指定数目的交换机作为拟态节点。拟态节点上的流表与普通的交换机上的流表不同，具有修改报文头部信息等动作。拟态控制器建立好流表之后，向发送端返回每个拟态流的入口地址（Entry address）。入口地址是发送端看到的各个拟态流的地址，即第一个拟态地址，发送端可以通过该地址向接收端发送数据。若要建立可靠数据传输通道，发送端通过入口地址分别与每个拟态流建立 TCP 连接。

由于发送端发起的请求报文相当于一个拟态通道开启的标识，攻击者可能在网络中通过监控该报文以获得“当前正在开启一个拟态通道”的信息。为了提升安全性，发送端向拟态控制器发送请求报文的过程并不直接向控制器地址发送报文，而是利用 SDN 交换机的“packet-in”功能，将该报文转发到拟态控制器。具体地，发送端在发起拟态通道时，不是每次向某一个固定的地址发送报文，而是随机地向网络中一个地址发送报文。如果在某一个交换机上触发了“packet-in”消息，则拟态控制器可以接收到该报文，并返回正确的结果。如果没有在任何一个交换机上触发“packet-in”消息，则可能转发到该地址所在的主机，那么可能收不到回复。为了避免后面一种情况，可以采用同时向多个 IP 地址发送报文，这样只要有一个触发“packet-in”消息都可以完成请求。另一方面，请求报文发送的目标 IP 地址尽量选取不在本机 ARP 表中的地址。这样就能确保报文可以发送到拟态控制器。同时多个报文也起到一定的干扰作用。进一步地，发送端节点可以定期地生成一些小报文来隐藏真实的“请求报文”。由于请求报文长度较小，对网络资源占用可以忽略不计。

4.4.1.2 数据传输阶段

通道建立之后，发送端可以通过拟态通道向接收端发送匿名数据。传输路径上的每个拟态节点将会修改报文的头部信息来隐藏通信参与者的身份信息。数据传输

完成之后，发送端会通知拟态控制器，以方便拟态控制器管理当前通道状态。

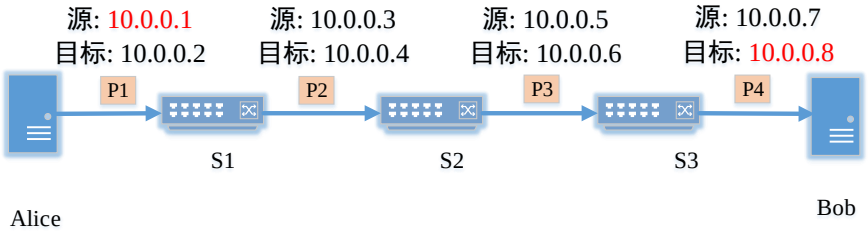


图 4-2 一个简单的 MIC 例子

举一个简单的例子来说明 MIC 的工作原理。假设用户 Alice（ip 地址为 10.0.0.1）与 Bob（ip 地址为 10.0.0.8）之间通过三个交换机（S1,S2,S3）连接，其中每个交换机具有修改报文头部的能力，如图4-2所示。为了达到匿名通信的目的，Alice 并不直接向 Bob 传输报文，而是先向网络控制器发起请求，并告知其要与 Alice 通信。网络控制器收到消息之后生成到达 Bob 的正确传输路径，并在沿路的多个交换机上修改报文的头部，使得攻击者无法获得报文的真实地址。

具体地，假设报文的头部为二元组《源 IP 地址，目标 IP 地址》。控制器在生成好传输路径之后通知 Alice 应该向地址为 10.0.0.2 发送报文，即图中的报文 P1 为 <10.0.0.1, 10.0.0.2>。报文 P1 到达交换机 S1 之后，其头部被修改为 <10.0.0.3, 10.0.0.4>，并发送到下一个交换机。同理，交换机 S2，S3 将报文头部修改成 <10.0.0.5, 10.0.0.6> 和 <10.0.0.7, 10.0.0.8>。值得注意的是，最后一次修改必需将目标地址修改成正确的接收端地址，从而接收端能够在不修改内核实现的情况下正常地接收与处理该报文。

4.4.2 路由生成算法

拟态控制器是 MIC 实现匿名性的核心，其具有每个流的全局视图，通过给交换机下发足以正确转发报文的最少信息来隐藏流真实的参与者。拟态控制器位于 SDN 控制器上，具有网络的全局视图，可以对流表进行控制。拟态控制器需要进行拟态通道的管理、路由生成与更新，以及处理流表冲突、保证网络传输的正确性。

当拟态通道建立时，发送端会向拟态控制器发送建立请求。当一个拟态通道传输完成时，发送端会发送关闭通知给拟态控制器。由此可以看出，在大量的短事务场

景下，拟态控制器需要处理大量的新建与关闭请求。为了减少拟态控制器的开销，对于相同两个节点之间重复的事务请求可以尽量重用一个拟态通道。因此，在这种场景下，当一个传输事务完成时，发送端并不立即发送通知给拟态控制器。而是由专门的功能模块定时地向控制器发送通知。

算法 4.1 路由生成算法

输入:

最短等价路径集合 P
发送端与接收端的标识 c 和 s
拟态流的数目 F
拟态节点的数目 N

输出:

每个拟态流的路由 R

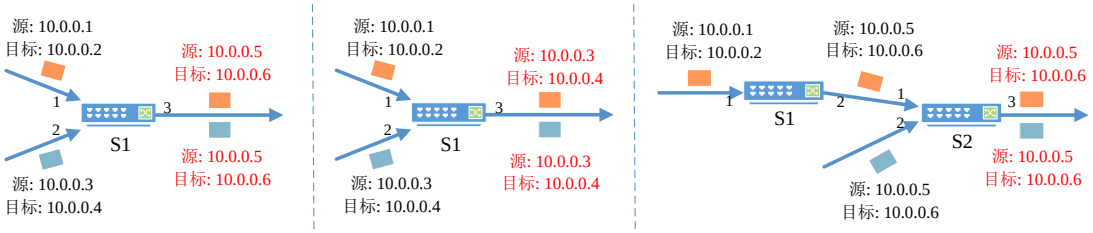
for $i = 0; i < F; i++$ **do**
  $F\_ID = GetFlowID(c, s)$
  // 从 P 中随机选取 c 到 s 的路径
  $path = RandomSelect(P_{c,s})$
  if $path.length < N$ **then**
    $path = PathCal(c, s, N)$
  end if
  $R[i].path = path$
  // 随机选取 N 个交换机作为拟态节点
  for $j = 0; j < N; j++$ **do**
    $mns[j] = RandomSelect(path.switches)$
    $S\_ID = GetMnID(mns[j])$
    $mns[j].maddress = GenAddr(V\_ID, S\_ID)$
  end for
  $R[i].mns = mns$
end for
return R

MIC 通过精心设计每个拟态流的传输路径、拟态节点、及相应的拟态地址变换来达到匿名通信的目的。控制器在启动时会获得网络的全局视图，并计算出任意两个交换机节点之间的所有等价最短路径（例如利用 Floyd-Warshall 算法）。在接收到客户端向路由模块发送请求之后，控制器需要生成相应数目的拟态流。首先，从请求报文中获得拟态流数目 F 和每条流的拟态节点数 N。如果接收端为一个匿名的服务端，那么控制器将会从已经注册的匿名服务的表中获得接收端的地址。对于每一条拟态流，从已经计算好的发送端与接收端的等价最短路径中随机选取一条。如果发送端与接收端的最短路径长度小于拟态节点数 N，则需要重新计算出一条长度大于

N 的路径。在选定路径之后，随机选取路径上的 N 个交换机节点作为拟态节点，并且确定每个拟态节点上的拟态地址，再安装相应流表到交换机。算法4.1显示了路由计算的伪代码，将会在第4.4.3节中介绍如何生成拟态地址。

4.4.3 全局路由冲突避免机制与拟态地址生成算法

拟态流的主要思想是利用拟态地址来隐藏真实的地址。为了保证最佳的匿名性，拟态地址可以是与真实地址同一网段的任意地址。因此，就有可能出现拟态流在某个交换机上的拟态地址与其它拟态流或者普通流（非拟态流）相同，从而发生冲突，最终导致出错。下面为三种路由冲突的情况的例子。为简化描述，假设二元组 <src_ip, dst_ip> 唯一标识一个流。



(a) 两个拟态流的地址同时被修改成相同的拟态地址

(b) 一个拟态流的地址被修改成与另外一个流相同的地址

(c) 具有相同拟态地址的两个流被转发到同一个端口上

图 4-3 路由冲突例子

（1）两个拟态流在同一个拟态节点上将地址变换成相同的拟态地址，并从同一端口转发。如图4-3（a）所示，两个拟态流 $f1$，$f2$ 分别从端口 1、2 进入交换机 S1，都从端口 3 转发出去。交换机 S1 将两个流的拟态地址都变换为 <10.0.0.5, 10.0.0.6>, 那么下一跳交换机将无法区分这两个流的报文，造成路由冲突。

（2）一个拟态流在一个拟态节点上进行地址变换后，在相同的拟态节点上与另一个流（可能是普通流，也可能是拟态流）地址相同，并从同一端口转发。如图4-3（b）所示，流 $f1$ 从端口 1 进入交换机 S1，从端口 3 转发出去，并不进行地址变换。而流 $f2$ 从端口 2 进入交换机 S1，地址被修改成与 $f1$ 相同的地址，并且同样从端口 3 转发。那么流 $f1$ 和 $f2$ 的路由将发送冲突。

（3）两个拟态流在不同的拟态节点上变换成相同的地址，并在另外一个交换机上转发到同一个端口。如图4-3（c）所示，流 $f1$ 和 $f2$ 分别在交换机 S1、S2 上将地址

修改成 <10.0.0.5, 10.0.0.6>，在交换机 S3 上通过同一个端口 3 转发，那么流 $f1$ 和 $f2$ 将发生路由冲突。

为了防止路由冲突，设计了一种全局路由冲突避免机制。**首先**，为避免普通流与拟态流之间的冲突，利用 MPLS（多协议标签交换，Multi-Protocol Label Switching）标志[73] 来进行区分。这里将 MPLS 分成不相交的两类：一类用于标记普通流（CF，Common Flows），一类用于标记拟态流（MF，Mimic Flows），只有拟态控制器知道哪些标记普通流，哪些标记拟态流。后面将介绍如何进行 MPLS 划分。控制器在生成普通流的路由时，每次随机从 CF 集合中选取一个 MPLS 值。由于普通流之间不存在冲突的可能性，因此不需要处理普通流之间的冲突。而普通流与拟态流具有不同的 MPLS 标识，从而可以很方便地区分普通流与拟态流，从而可以避免普通流与拟态流之间的冲突情况。

其次，为了避免不同拟态流之间的冲突，设计了拟态地址生成算法（M-address Generation Algorithm，MAGA）。其主要思想是合理划分每个拟态流在拟态节点上的变换地址空间，使得不同的拟态流在拟态节点上变换后的地址空间不重合。对于一个拟态流，其真实地址为 $< src\_ip, dst\_ip >$，拟态节点需要将其变换成拟态地址 $< m\_src\_ip, m\_dst\_ip >$。为了减少不同流之间拟态地址冲突的可能性，增加 MPLS 标识，即使用三元组 $< m\_src\_ip, m\_dst\_ip, m\_mpls >$ 唯一标识一个拟态流。

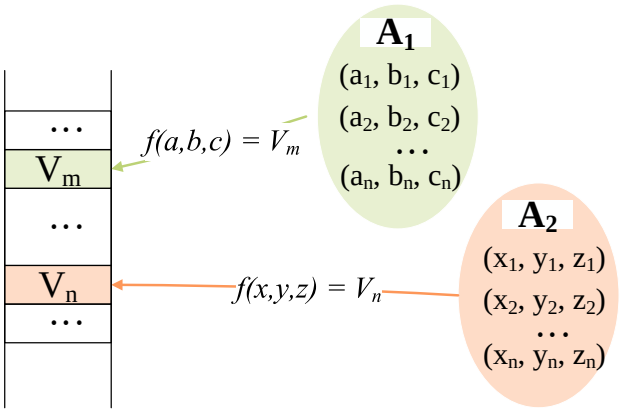


图 4-4 哈希映射示意图

为此，通过构建哈希函数 $f(x, y, z)$ 来将每个拟态流的拟态地址映射到不同的地址空间。哈希函数 $f(x, y, z)$ 需要满足以下条件：给定任意两个不同的的哈希值 V_m 和

V_n，可以获得两个不同的集合 A_1 和 A_2，使得所有 $(a,b,c) \in A_1$ 和所有 $(x,y,z) \in A_2$，满足 $f(a,b,c) = V_m$ 和 $f(x,y,z) = V_n$，但是 $(a,b,c) \neq (x,y,z)$，如图4-4所示。因此，只要令每个拟态流具有唯一的标识 ID, 且其所有拟态地址 (a,b,c) 满足条件 $f(a,b,c) = ID$，则不同的拟态流之间的拟态地址将不会冲突。那么现在需要解决的主要问题是如何保证每个拟态流的 ID 唯一。一种简单的方法是按照拟态流到达的顺序递增其 ID 值，即每到达一个新拟态流，其 ID 值为当前未被使用的最大 ID，并将 ID 加一。因此可以保证每个拟态流具有唯一的 ID 值，其拟态地址不会与其他拟态流冲突。

最简单的方式下，只需要设置一个全局的哈希函数即可以保证一个拟态流在网络中任意的交换机上都不会与其它拟态流发生冲突。然而，这种方案使得所有拟态流的地址变换都遵循相同的规则，减小了被攻击者分析破解的难度。一旦攻击者知道哈希函数，那么可以很容易地在多个点关联出同一个拟态流的报文，打破匿名性。为了解决上述的问题，提升系统的匿名性，需要为每个拟态节点设置不同的哈希函数，而不用全局统一的哈希函数。攻击者将难以同时获得所有拟态节点上的哈希函数，从而无法轻易地破解系统的匿名性。然而，当每个拟态节点上都具有不同的哈希函数时，只能保证同一个拟态节点上（生成）的拟态地址不发生冲突，而无法保证不同的拟态节点之间的拟态地址不发生冲突。后文中所有的“交换机上的拟态地址”指“在该交换机上变换后生成的拟态地址”。如图4-3（c）所示即为两个不同拟态节点上的拟态地址发生冲突的情况（如果 $f2$ 为一个拟态流）。

为了避免这种冲突情况，同样使用 MPLS 标志来保证每个交换机上变换后的拟态地址不会与其他交换机上变换生成的拟态地址冲突。因此，给每个交换机分配不相交的 MPLS 集合，从而使得任何一个交换机上的拟态地址中 MPLS 值与其它交换机上的不相同。为了保证匿名性，只有控制器知道每个 MPLS 对应哪个交换机。类似地，构建哈希函数 $g(x)$，对于一个交换机（标识为 S），则其上计算出的所有拟态地址中 MPLS 值必须满足 $g(x) = S$。

因此，本方案中的关键问题是如何构建两个哈希函数 $f(x,y,z)$ 和 $g(x)$。对于 $f(x,y,z)$，需要能够实现：给定任意哈希值 V，可以获得满足条件的拟态地址三元组 (a,b,c)。因此 $f(x,y,z)$ 必须至少对于某一变量可逆。从而可以先确定三个变量中两个变量，再通过 $f(x,y,z)$ 的反函数计算出另外一个变量的取值，最终获得一组满足

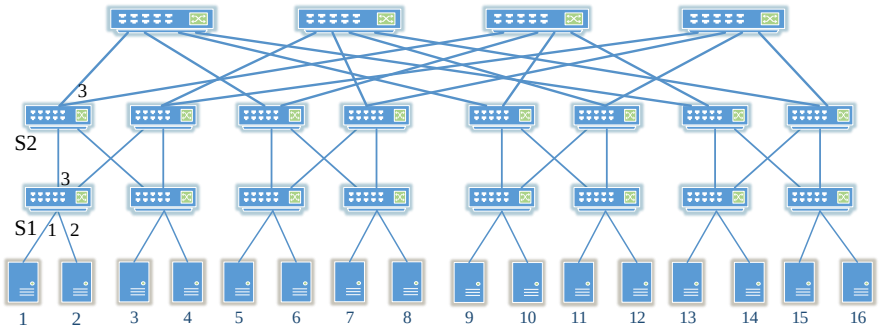


图 4-5 Fat-tree 拓扑结构

条件的拟态地址。为了保证各个变量的取值为整数，在构建 $f(x,y,z)$ 时采用基于异或、移位的方法。例如，一个简单的 $f(x,y,z)$ 可以如下构建：

$$\begin{aligned} f(x,y,z) =& [(x \oplus A_0) >> A_1] \oplus [(x \oplus A_2) << A_3] \\ & \oplus [(y \oplus B_0) >> B_1] \oplus [(y \oplus B_2) << B_3] \\ & \oplus [(z \oplus C_0) >> C_1] \end{aligned} \quad \text{(式 4.1)}$$

则对于变量 z 的反函数为:

$$\begin{aligned} f_z^{-1}(v,x,y) =& v \oplus [(x \oplus A_0) >> A_1] \oplus [(x \oplus A_2) << A_3] \\ & \oplus [(y \oplus B_0) >> B_1] \oplus [(y \oplus B_2) << B_3] \\ & << C_1 \oplus C_0 \end{aligned} \quad \text{(式 4.2)}$$

其中 $A_0, A_1, A_2, A_2, B_0, B_1, B_2, B_3, C_0, C_1$ 为参数，每个拟态节点上可以选取不同的参数来构建不同的哈希函数。

实际上，为了保证更好的匿名性，每个拟态节点上的拟态地址的源、目标 IP 都具有不同的限制。例如，对于一个 Fat-Tree 拓扑结构（如图4-5所示），交换机 S1 上的端口 3 发出报文的源地址一般为 1 或 2。这样可以达到拟态流报文与非拟态流报文相似，使得攻击者无法轻易地区分出哪个是拟态流，哪个是普通流。同时，如前面所述，每个拟态节点上的 MPLS 取值具有不同的限制，保证不同的拟态节点上的拟态地址不发生冲突。因此，三元组 $< m\_src\_ip, m\_dst\_ip, m\_mpls >$ 中三个元素的值都不能够进行任意取值。为了能够快速地获得满足条件的三元组，将其中的 MPLS

（m_mpls）值分成两部分，包括 mpls1 与 mpls2，其中 mpls1 的取值受限于哈希函数 $g(x)$ 以区分不同交换机，而 mpls2 可以任意取值。因此，地址三元组等价于一个四元组 $< m\_src\_ip, m\_dst\_ip, mpls1, mpls2 >$。为了获得满足条件的四元组，类似地构建了哈希函数 $F(\alpha, \beta, \gamma, \delta)$，其等价于 $f(x, y, z)$。最终，可以先随机地选取符合条件的 m_src_ip, m_dst_ip 和 $mpls1$，再使用反函数 $F_{\delta}^{-1}(v, \alpha, \beta, \gamma)$ 计算出 $mpls2$ 的值。

对于哈希函数 $g(x)$，由于只有一个变量，难以直接构造出一个满足条件的函数。因此，将变量 x 按位 (bits) 分成多个独立的变量，进而构建类似于 $f(x, y, z)$ 的多变量哈希函数。例如，一种简单的方案是将变量 x 的高字节与低字节作为两个独立的变量。假设变量 x 有 32 位，则 $x1$ 为高 16 位，$x2$ 为低 16 位，哈希函数 $g(x)$ 等价于函数 $h(x1, x2)$。同样地，一个简单的 $h(x1, x2)$ 可以如下构建：

$$\begin{aligned} h(x1, x2) =& [(x1 \oplus A_0) >> A_1] \oplus [(x1 \oplus A_2) << A_3] \\ & \oplus [(x2 \oplus B_0) >> B_1] \end{aligned} \quad \text{(式 4.3)}$$

其对于 $x2$ 的反函数为：

$$\begin{aligned} h_{x2}^{-1}(v, x1) =& v \oplus [(x1 \oplus A_0) >> A_1] \oplus [(x1 \oplus A_2) << A_3] \\ & << B_1 \oplus B_0 \end{aligned} \quad \text{(式 4.4)}$$

算法 4.2 拟态地址生成算法

输入:

拟态流的标识, V

交换机的标识, S

输出:

该交换机上拟态流的拟态地址, M

1: // 随机选取符合条件的 m_src_addr 和 m_dst_addr
2: $M.m\_src\_addr = RandomSelect()$
3: $M.m\_dst\_addr = RandomSelect()$
4: // 计算 $mpls_1$: (1) 随机选取 x_1; (2) 利用哈希函数的反函数 h^{-1} 计算 x_2
5: $x_1 = RandomSelect()$
6: $x_2 = h^{-1}(S, x_1)$
7: $mpls_1 = combine(x_1, x_2)$
8: // 利用 F^{-1} 计算 $mpls_2$
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

对于一个拟态节点，若其标识为 S，其对应的 MPLS 中的 $mpls1$ 值应该要满足 $g(mpls1) = S$。给定 S，先随机地选取 $mpls1$ 的高 16 位 $x1$，再通过反函数 $h_{x2}^{-1}(v, x1)$ 求出低 16 位 $x2$，则 $mpls1$ 的值为 $mpls1 = x1 << 16 + x2$。值得指出的是，为了保证安全性，可以通过随机地分隔变量 x，或者增加子变量的数目来增加被攻击者通过分析获得哈希函数的难度。同样地，对于普通流，为其赋予唯一的标识 C，所有 $mpls1$ 满足 $g(mpls1) = C$ 标识普通流。拟态地址生成算法的伪代码如算法4.2所示。

4.4.4 抗流量分析攻击机制

攻击者可能在网络中某处（或者多处）进行流量监听，分析报文的时序、速率和长度等信息，以进一步推测出流量的参与者和操作的类型等。为了抵御这类流量分析攻击，提升被分析的难度，提出以下两种机制: 多拟态流和局部多播机制。

（1）多拟态流机制

MIC 的设计目标是在不修改交换机实现的前提下提供合适的匿名通信能力。由于目前主流的 SDN 交换机一般只支持 OpenFlow 协议规定的功能，而并不具备用户自定义可编程的能力。因此，基于 SDN 交换机的拟态节点并不能像传统的“mix”节点一样对报文进行延迟、加/解密、合并等操作。MIC 中拟态节点只能修改报文的头部信息，而不对报文的时间和长度等信息进行修改。那么，要打乱一个流的传输速率与长度，只能够在发送端进行相应的操作。为此，提出多拟态流机制来提升攻击者发动流量分析攻击的难度。具体地，每个 MIC 由多个拟态流组成，每个拟态流经由不同的路径。发送端在发送数据时，将数据分块成条带，再将这些条带映射到不同的拟态流上进行传输。采用多拟态流机制的优势包括两方面。第一，将用户数据通过多路径传输，可以有效增加被同时窃取的难度，抵御窃取内容的攻击。第二，将流量分到不同的流上去，攻击者难以获得真实的流量长度与速率，除非其能够知道有多少条子流，并关联所有子流。

（2）局部多播机制

攻击者可能监听一个拟态节点的所有输入与输出报文，关联出该节点上的拟态流，并通过迭代攻击的方式关联完整的流。在一个拟态节点上只有少数拟态流时，攻击者可以通过分析输入与输出报文中不匹配的报文，关联头部变换后的拟态流报文。

同时，由于拟态节点并不对报文进行加密、解密操作，因此报文在网络中各处内容相同。攻击者可以通过对比端到端的报文内容来关联它们。

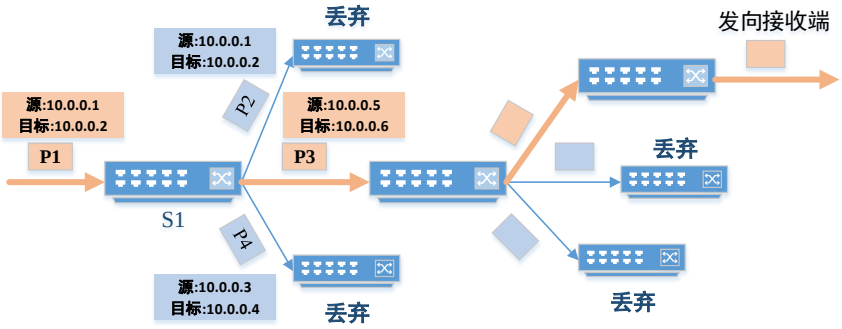


图 4-6 局部多播机制示意图

MIC 并不能抵御这种端到端的关联攻击，但是采用局部多播机制来最大化地减少这种关联的成功率。具体地，在拟态节点上将拟态流的报文头部进行修改，得到多个具有不同报文头部的报文，并将它们（包括原始的报文）从不同的端口发出。其中只有一个报文最终能够达到接收端。例如，如图4-6所示，一个拟态流的报文 P1：<10.0.0.1, 10.0.0.2> 从端口 1 进入，经过拟态节点之后分别从端口 2，3，4 输出报文 P2：<10.0.0.1, 10.0.0.2>, P3:<10.0.0.3, 10.0.0.4> 和 P4：<10.0.0.5, 10.0.0.6>。只有 P3 最终可以达到接收端，而 P2 和 P4 将会在下一个跳（或者多跳）被丢弃。攻击者无法轻易地确定哪个才能到达接收端。

4.5 高效匿名通信方案安全讨论

MIC 是设计用于保证数据中心内通信的匿名性。下面讨论几种在数据中心实际可行的攻击。

控制交换机。攻击者可能控制了传输路径上的某一个或者多个交换机，其中可能是普通交换机或者拟态节点。考虑以下几种情况：1）若攻击者控制了拟态流中发送端到第一个拟态节点之间的节点，攻击者可以获得流量的发送者，而无法获得接收者；2）若攻击者控制了拟态流中最后一个拟态节点到接收端之间的节点，攻击者

可以获得流量的接收者，而无法获得发送者；3）若攻击者控制了拟态流中第一个拟态节点与最后一个拟态节点之间的节点，则攻击者无法获得流量的发送者和接收者。因此，攻击者在网络中任何一处都无法同时获得某一个通信的发送端与接收端。由于网络中任何一个交换机都可能是拟态节点，因此攻击者无法确定哪个是第一个或者最后一个拟态节点（如果只控制了一个交换机节点）。

控制发送端或接收端。攻击者控制了发送端（或接收端），企图获得与其通信的接收端（或发送端）节点，从而确定下一个攻击目标。如果接收端为隐藏接收者，则发送者无法获得接收者的地址，同时接收端也并不知道真实的发送端地址。因此，控制发送端或接收端无法打破传输的不可链接性。

流量监听。攻击者通过某种手段（如通过控制交换机的镜像端口 mirror ports）监听一个或者多个交换机节点的流量，企图进行流量分析或者关联流的发送端与接收端。攻击者可能获得一个或者少数几个交换机节点的输入和输出流量，通过分析输入与输出报文的头部信息来关联属于同一个拟态流的报文。由于数据中心内交换机的镜像端口只有在少部分节点上会开启，因此攻击者需要获得全局的流量是不现实的。MIC 采用局部多播的方案来增加关联的难度，从而使得攻击者无法在单个节点上关联同一个拟态流的输入与输出报文。

基于长度和速率的流量分析。攻击者可能在网络中统计每个流的报文长度与传输速率，企图获得某个发送端或者接收端的流量模式，进而推测其正在进行何种操作。MIC 采用多拟态流的机制可以有效地增加这类攻击的难度。攻击者并不知道一个通道使用了几个流进行传输，并且无法将不同的流关联到同一个通道上去。因此，即使攻击者获得了某一个单独的流的流量模式，也无法获得整个通道的。

Dos 攻击。传统的基于“overlay”的匿名通信方案中，由于发送端指定传输路径中的“mix”（或者“relay”）节点，恶意用户可以很容易地发动 DoS 或者 DDoS 攻击。一个恶意用户可以指定特定的“mix”节点，从一个或者多个发送端向其发起大量请求来占用该“mix”节点上的资源，从而达到拒绝其它用户访问的目的。由于“mix”节点上需要进行大量的 CPU 密集性的加密/解密运算，并且只具有有限的数量转发能力（一般只具有一到两个网络端口），因此对于这种攻击比较脆弱。MIC 可以有效地抵御这种 Dos 攻击：1）所有转发节点均为交换机节点，具有很强的转发能力；2）所

有传输路径由拟态控制器决定，用户无法指定特定的拟态节点，因此所有流量都会平均地分配到不同的传输路径上；3）拟态节点上不需要对每个报文进行加密与解密操作，可以有效地减少 CPU 开销，提升转发性能。另一方面，拟态控制器也可以通过进行访问控制来过滤恶意用户发起的匿名通信请求。

基于 RTT 的位置推断攻击。攻击者可能控制发送端（或者接收端），并通过测量 RTT 来分析出另一方的位置或者减少搜索的范围。例如，在 Fat-Tree 拓扑中，同一个机架内的两个节点比不同机架之间的距离要短。若攻击者发现某一个通信对之间的 RTT 比其它的 RTT 都小，那么就很可能推测出两者位于同一个机架内。为了抵御这类攻击，MIC 在生成转发路径时，尽量保证所有路径长度相当（如 Fat-Tree 任意两个节点的距离为 6 跳）。

滥用攻击。传统的匿名方案都面临着滥用的问题。匿名方案中，用户可以随意地发送不可追踪的流量，造成很大的资源浪费和管控问题。数据中心内匿名通信的主要目标是对恶意用户或者第三方黑客保持匿名，而并不能对服务提供商匿名（因为整个网络必须由服务提供商来控制）。在 MIC 中具有全局的控制器，使用控制器可以有效地解决匿名滥用的问题。

从上述安全讨论可以看出，MIC 可以抵御攻击者在控制部分交换机或者客户端时的关联攻击，保护通信参与者的身份信息；针对基于长度和速率的流量分析攻击有一定的抵御能力；可以很好地抵御针对匿名方案本身的 Dos 攻击与滥用攻击。

4.6 基于匿名通信的分布式存储系统

MIC 是针对数据中心内应用设计的，具有很好的易用性，可以比较方便地应用于分布式存储系统中。本节实现了基于匿名通信的分布式存储系统，以提升分布式存储系统的安全性。首先，在 RPC（远程过程调用，Remote Procedure Call）[100,101] 上整合了 MIC，然后在实验室开发的分布式文件系统 CapFS 上实现了匿名存储系统 CapFS-M。

RPC[①]是广泛应用于分布式系统（例如 NFS, Lustre）的通信协议。其为上层的分布式系统提供统一的调用接口，隐藏了复杂的网络操作。使得程序设计者可以专注于程

① https://en.wikipedia.org/wiki/Remote_procedure_call

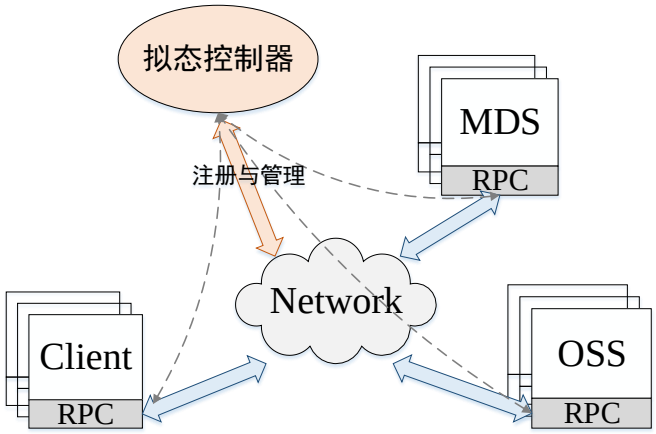


图 4-7 基于匿名通信的分布式存储系统 CapFS

序逻辑的设计，而无需考虑具体的底层网络操作。课题组在开源的 TI-RPC（Transport-Independent RPC）[①]之上构建了分布式文件系统 CapFS。CapFS 由 Client（客户端）、OSS（对象存储服务器）和 MDS（元数据存储服务器）三部分组成，其中每个部分分别可以构建成集群。Client 提供兼容 POSIX 接口的用户访问接口，OSS 存储所有对象数据，MDS 存储与管理所有元数据。可以看出，MDS 是存储系统中的关键节点，如果攻击者想要破坏存储系统以使得其不能对外提供服务，那么其可以优先地攻击 CapFS 中的 MDS，从而快速地使得系统瘫痪。例如，攻击者可以从网络中分析请求模式，猜测出哪个是 MDS 节点；或者先控制某个 Client 节点，然后获得 MDS 节点的 IP 地址，之后再发动 DDoS 攻击。通过使用匿名通信方案，分布式文件系统中可以实现内部结点之间的匿名通信，从而保护系统中的一些关键节点（例如 MDS，OSD）不被攻击者或者恶意用户锁定并发起攻击。

为此，本节利用 MIC 实现了基于匿名通信的分布式存储系统 CapFS-M。首先，使用 MIC 的客户端接口来替换 TI-RPC 中的 SOCKET 接口，从而实现基于 MIC 的 RPC-M。然后，利用 RPC-M 替换原始的 RPC，实现基于 MIC 的分布式存储系统 CapFS-M。让 CapFS 上的所有节点之间的通信都使用 MIC 来实现，从而使得节点之间互相匿名，其架构如图4-7所示。在系统启动时，需要给每个节点分配一个别名，例如 $mds0$，$osd0$，$client0$ 等。每个节点启动时需要向 MIC 控制器发送注册请求，成为匿名的接收

① https://sourceforge.net/projects/libtirpc/

端。当一个节点需要主动地向另外一个节点发起请求时，则通过别名进行通信。例如，当一个 Client 需要向一个 MDS 发起元数据请求，那么 Client 直接通过其别名 $mds0$ 进行通信，而无法获取该节点的真实地址。同样地，$mds0$ 也无法获得 $client$ 的真实地址。由于 MIC 控制器知道每个节点的身份信息，因此可以在控制器上进行访问控制。

通过实现匿名的分布式存储系统，可以有效地抵御以下两方面的安全威胁：

（1）网络流量分析。MIC 可以有效地提升抗流量分析攻击，从而可以有效地提升攻击者在网络环节中通过分析流量模式来定位目标节点（如 MDS 节点和存储敏感数据的 OSD 节点）的难度。

（2）恶意客户端。若攻击者已经控制了存储系统中某个客户端，那么它可以很容易地获得存储系统中关键节点的位置信息。同时，通过客户端可以定位一些敏感数据的位置（保存该数据的存储服务器的位置）。考虑以下情况，假设恶意的 Client 并不具备访问敏感数据的权限，但是企图窃取这些数据。在一些开源的分布式存储系统中，Client 可以随意地获得或者计算出某一个对象所在存储服务器信息。例如 Ceph 系统中，任何一个节点都可以利用 Crush 算法计算出一个对象所在的服务器位置。那么，即使客户端不能直接访问敏感数据，也可以定位到该数据所在的服务器，然后通过其它方式访问该服务器以窃取敏感数据。在章节4.1的 Target 公司信用卡泄露的例子中即为类似的场景。攻击者在获得目标节点的 IP 地址后，可以通过一系列的手段绕过防火墙与其它网络安全设施，最终窃取目标节点上的数据。MIC 正是可以破坏其攻击链中重要的“定位目标节点 IP 地址”环节，从而提升分布式存储系统的安全性。

4.7 性能评价

测试平台在虚拟网络环境 Mininet[65] 上搭建。硬件平台为一台服务器，具体配置为：Intel(R) Xeon(R) E5-2620@ 2.00GHz CPU，32GB 内存。安装 Ubuntu 12.04.5 LTS 操作系统、Mininet 2.2.0[①]模拟软件和 Openvswitch 2.4.0[66] 虚拟交换机软件。SDN 控

① http://mininet.org/

制器使用 RYU 3.17①。网络拓扑为 k=4 的 Fat-Tree[67] 拓扑结构，包涵二十个 4 端口交换机和 16 个主机。

分别对比测试了 MIC 与 Tor、TCP 和 SSL 的路由建立时延、传输时延和传输带宽。MIC-TCP 和 MIC-SSL 分别是基于 TCP 与 SSL 实现的 MIC 版本。测试过程中使用 TCP 和 SSL 作为基准。

4.7.1 路由建立时延

本节对比测试了 MIC、Tor、TCP 和 SSL 的路由建立时延。对于 MIC，测试了发起端完成“MIC_connect”操作所需的时间。该过程包括发起端向拟态控制器 MC 发送请求报文，MC 完成路由计算并安装到相应的交换机，返回入口地址给发起端，发起端再与接收端建立 TCP 连接。MIC 建立过程中的请求报文使用 OpenSSL②中的 AES③函数进行加密和解密。对于 Tor、TCP 和 SSL，测试了发送端完成“connect”操作所需的时间，其中进行 Tor 测试时使用“torsocks”命令。通过修改 Tor 源代码中的“DEFAULT_ROUTE_LEN”可以测试不同的路由长度时的结果。

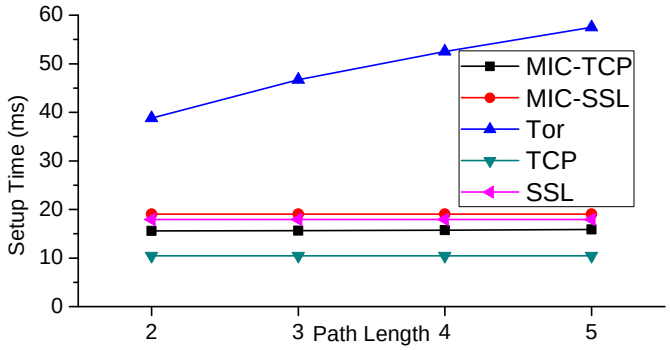


图 4-8 MIC、Tor、TCP 与 SSL 的路由建立时间

图4-8所示为在不同的路由长度时各方案的路由建立时延。对于 Tor，路由长度（Path Length）为流量被转发的次数。而对于 MIC，路由长度为报文地址被变换的次数。TCP 和 SSL 作为基准测试，路由长度对其无意义。与所预期的结果一致，MIC 的建立时延优于 Tor。这是由于 MIC 比 Tor 具有更加轻量化的操作和更短的传输路径。

① http://osrg.github.io/ryu
② https://www.openssl.org/
③ https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Tor 为基于覆盖网络（overlay）的方案，其传输路径会随着路由长度的增加而增加，因此其建立时延也会随着路由长度的增加而明显增长。而 MIC 为网内（in-network）匿名方案，实际传输路径并不会随路由长度而明显增长。相比于基准的 TCP 和 SSL，由于 MIC 需要额外地向拟态控制器 MC 发送建立请求，因为需要引入少量的开销。

4.7.2 传输时延与带宽

在会话建立之后，对比测试了 MIC 与 Tor、TCP 和 SSL 的传输时延与带宽。在时延测试中，测量了发送端向接收端发送 10 字节的数据，接收端接收到数据后向发送端回复 10 字节的数据所需的时间。每个测试重复 10 次取平均值。

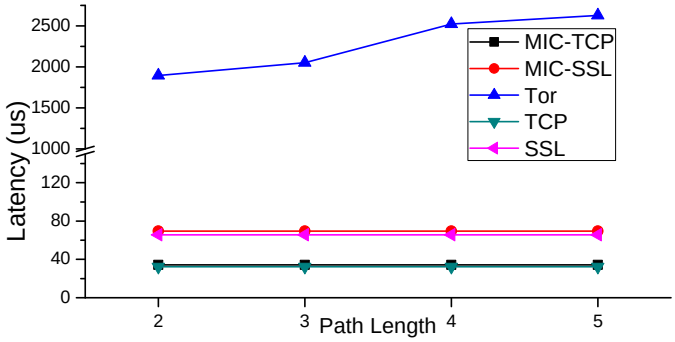


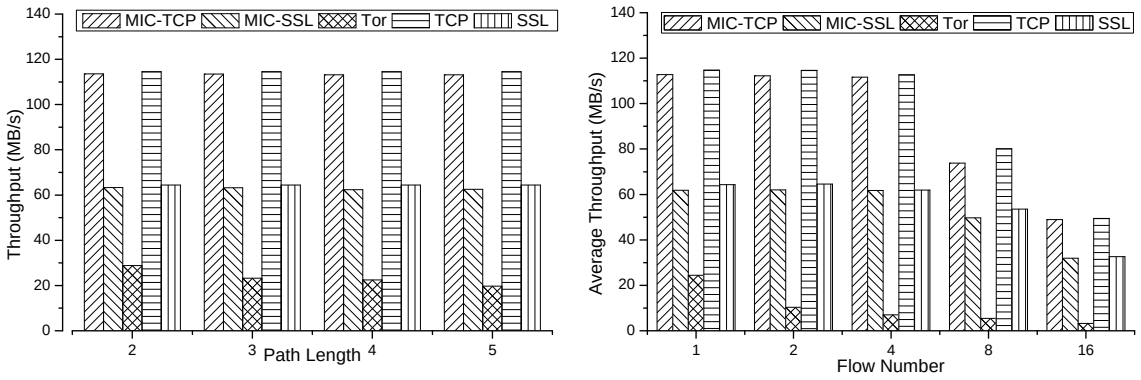
图 4-9 MIC、Tor、TCP 与 SSL 的传输时延

图4-9所示为时延测试的结果。从结果中可以看出，MIC（MIC-TCP 和 MIC-SSL）的时延明显低于 Tor，并且 MIC-TCP 和 MIC-SSL 的时延分别与 TCP 和 SSL 相当。相比于 Tor，MIC 具有更少的加密/解密操作和更短的传输路径（链路与主机协议栈），因此可以达到更低的时延。相比于 TCP（或者 SSL），MIC 只需在交换机加引入更多的“Actions”（修改报文头部的操作），其带来的时延开销约为 5% 左右。

在带宽测试中，使用 Iperf[①]测试 Tor 和 TCP 的带宽，并使用修改的 Iperf 测试 MIC 和 SSL 的传输带宽。每个测试传输 1G 数据，重复测试 10 次取平均值。首先测试了单个流在不同的路径长度下的传输带宽，然后测试了多个流同时测试时的平均传输带宽（其中路径长度为 3）。

图4-10所示为带宽测试的结果。不出所料，由于具有更加轻量化的设计，MIC 比

① https://iperf.fr/



(a) 单个流的传输带宽　　(b) 多个流的平均带宽

图 4-10 MIC、Tor、TCP 与 SSL 的传输带宽

Tor 具更高的传输带宽。另外，由于 Tor 为基于覆盖网络（overlay-based）的方案，完成一个数据传输需要占用大量冗余的计算和网络资源，因此，随着流数的增加，Tor 会更快地使网络资源利用率达到饱和，从而造成拥塞。而 MIC 并不需要引入额外的传输路径，因此可以达到与 TCP 或者 SSL 相当的传输性能。

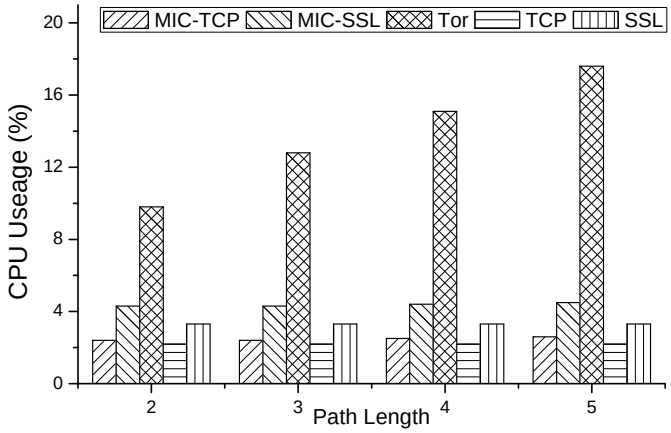
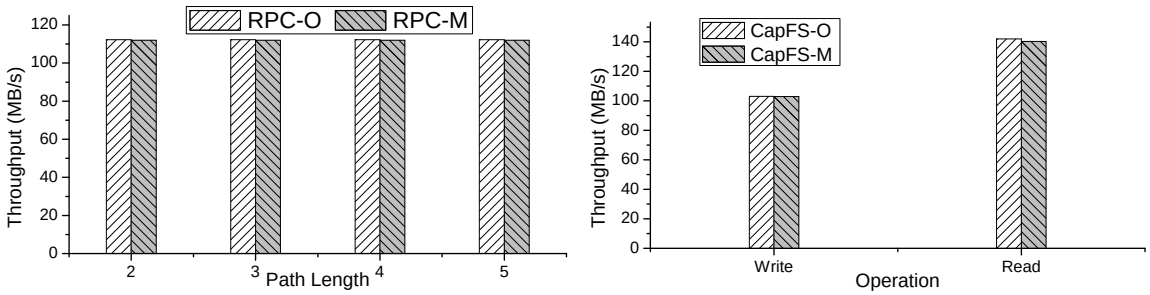


图 4-11 单个流带宽测试时的 CPU 利用率

另外，还对比测试了在进行单个带宽测试时各个方案的 CPU 利用率，结果如图4-11所示。从结果中可以看出，MIC 相比于 TCP（或者 SSL）会引入少量的额外 CPU 开销。这主要是由于 MIC 需要在交换机中进行更多的操作。然而 Tor 因为需要进行大量的额外传输与中间操作（如加密与解密操作），从而会引入巨大的 CPU 开销。

4.7.3 应用测试

本节测试了原始的 RPC（RPC-O）与基于 MIC 的 RPC（RPC-M），以及原始的分布式文件系统 CapFS-O 和基于 MIC 的 CapFS-M 之间的性能。首先，测试了 RPC 的带宽。测试过程中，发送端向接收端传输 1GB 数据，每次过程调用传输 1MB 数据。图4-12(a) 显示了 RPC-O 与 RPC-M 的带宽对比的结果。从测试结果中可以看出，RPC-M 相比于 RPC-O 引入的带宽开销为 0.2%，基本上可以忽略不计。



(a) RPC-O 与 RPC-M 的带宽对比　　(b) CapFS-O 与 CapFS-M 的读写带宽对比

图 4-12 基于 MIC 的应用性能测试

其次，测试了分布式文件系统 CapFS 的读写性能。由于 MIC 的原型实现是在用户态实现的，因此测试过程中使用基于 FUSE(用户态文件系统，Filesystem in Userspace)[①]的用户态 CapFS 版本。由于 CapFS 采用的 FUSE 框架在 Mininet 的主机节点上挂载失败，因此采用 KVM(Kernel Virtual Machine)[②]虚拟机来搭建测试环境。在该测试平台中，宿主机为一台安装有 Ubuntu 12.04 的主机，使用 KVM 安装了三台虚拟机，分别配置为 MDS、OSS、Client 节点。虚拟机之间使用 Openvswitch 连接。使用文件系统测试程序 Iozone[③]分别测试了 CapFS-O（原始的 CapFS 版本）与 CapFS-M（基于 MIC 的 CapFS 版本）的读写带宽，测试文件大小为 512M，测试结果如图4-12(b) 所示。从结果中可能看出，相比于 CapFS-O，CapFS-M 引入的读写带宽开销小于 1%。总体上来看，从上述测试中可以得出，MIC 具有极小的性能开销（小于 1%），同时具有很好的应用性，可以很方便地应用于现有的分布式存储系统中。

① https://en.wikipedia.org/wiki/Filesystem_in_Userspace
② http://www.linux-kvm.org/page/Main_Page
③ http://www.iozone.org/

4.7.4 扩展性分析

为了最大化拟态流的匿名性，每个流的转发路由应当尽量地随机，以增加攻击者破解的难度。为此，MIC 不能利用 SDN 中一项重要的技术，“通配路由”（wildcard rule），来优化转发效率，减少网络中流表的数量。因此 MIC 将可能会在交换机上产生大量的流表。为了缓解这个问题，可以将拟态流尽量均衡地分布到各个路径上，从而使得每个交换机上的拟态流的数目均衡。最简单的方案就是在拟态流生成的时候，随机地选取等价路径中的一条。下面测试了在不同的网络规模下（Fat-Tree 拓扑），每个交换机上流表的数量，如表4.2所示。由于当前 SDN 交换机中流表的容量一般 1500 左右，从结果中可以得出，MIC 可以在大规模的数据中心环境下（具有 27648 主机，每个主机 10 条流）正常工作。

表 4.2 在不同规模网络环境下每个交换机上流表的数量

拓扑规模	主机数	每个主机上流的数量					
		平均值			最大值		
		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

由算法4.1可以得出，MIC 的路由生成的时间复杂度为 $O(|F|)$，其中 $|F|$ 为一个 MIC 内的拟态流的个数。默认地，每个 MIC 具有一个拟态流。并且在实际部署中，单个通道内的拟态流一般不会超过 10 个。下面测试了拟态控制器生成一个通道需要的时间，如表4.3所示。生成一个通道的时候包括计算路由与安装流表的时间。值得注意的是，安装流表的时间与 SDN 控制器的实现有关，而与 MIC 的设计无关。从测试结果中可以看出，单个控制器可以处理 2000 条流。现有的实现是在 Ryu 上完成的，

而其目前只能够支持单线程处理。相信通过优化控制器软件的实现（例如采用多线程）之后，单个控制器节点每秒能够处理更多的流。

表 4.3 不同拟态流数量时的通道生成时间

拟态流数量	1	2	3	4	5
计算时间 (ms)	0.5	0.603	0.813	0.937	1.053
安装时间 (ms)	4.887	9.595	14.384	19.113	23.716

由于 MIC 采用逻辑集中控制的架构，那么自然会面临控制器单点瓶颈的问题。幸运的是，SDN 架构的扩展性可以通过多控制器的方案[102] 来解决。而 MIC 可以很轻松地部署在多控制器上。只要保证每个通道具有唯一的 ID，MIC 的冲突避免机制就可以保证其的正确性。例如，可以为每个控制器分配一个独立的 ID 空间，那么就可以保证所有流不会发生冲突，并且每个控制器可以独立地生成拟态流。

4.8 本章小结

随着云计算技术的发展，越来越多的应用向云数据中心迁移的同时将越来越多的安全威胁引入到数据中心内部。分布式存储系统作为数据中心内的应用，面临着越来越多来自网络与恶意用户的威胁。通过保护分布式存储系统各个节点的身份信息（如 IP 地址）可以有效地提升系统的安全性。为此，本章提出基于匿名通信的分布式存储系统，使得系统内各个节点之间的通信彼此匿名。然而，现有的匿名通信方案都是针对互联网环境设计，在提供匿名性的同时需要引入大量的开销，从而无法满足数据中心内对低延时、高带宽的需求。为此，本章针对数据中心环境，提出一种基于全局路由冲突避免机制的网内匿名通信方案 MIC。与传统的基于覆盖网络（overlay）的匿名方案不同，MIC 采用网内（in-network）匿名的架构，即在交换机上修改报文的源/目标地址，达到隐藏通信参与者的目的。因此，MIC 可以在极小的开销下实现传输的匿名性。利用 MIC 在分布式存储系统 CapFS 上实现了基于匿名通信的存储系统 CapFS-M。测试结果表明，CapFS-M 相比于原始的存储系统 CapFS-O 引入的开销小于 1%。

5 全文总结与展望

数据规模的爆炸式增长，对分布式存储系统提出了更高的需求。新型存储介质（例如 SSD，NVM 等）加入到分布式存储系统中使得存储节点内的 I/O 处理速度高于跨存储节点之间网络 I/O 的处理速度。因此网络传输将逐渐成为阻碍分布式存储系统性能提升的瓶颈。为了降低构建成本，分布式存储系统往往部署在廉价的通用服务器上。为了补足廉价服务器可能造成的频繁的失效，分布式存储系统通常采用多副本机制来提升数据存储的可靠性。然而在存储与维护数据的多个副本时，需要引入额外的存储和数据传输开销。在存储节点失效时，进行数据恢复的过程也会产生大量的网络传输开销。如何提升网络传输的效率从而提升存储系统整体性能是需要研究的重要课题。另一方面，随着存储系统规模的扩张，遭受到来自网络与恶意用户攻击的概率也随之增加。如何提升分布式存储系统抵御攻击者发起的系统破坏、数据窃取等攻击的能力也是值得研究的课题。

软件定义网络技术的出现与发展使得网络传输更加灵活高效，也为提升分布式存储系统的网络传输效率与安全性带来了新的思路。通过利用 SDN 全局视图、中心化控制与可编程能力，可以使得网络的路由计算与流量调度更加灵活高效，以满足不同的网络应用需求。

5.1 主要成果贡献

本文针对提升分布式存储系统网络传输效率与安全性等方面开展了相关的研究，具体的研究工作的主要成果贡献如下：

1、针对分布式存储系统多副本机制性能问题，提出一种拥塞感知的可靠组播方案 MCTCP 和组播多副本机制。MCTCP 针对分布式存储系统多副本场景进行了专门设计，考虑了该场景下小组模式、要求数据传输可靠、发送端发起、要求高效率与鲁棒性等特点。其通过在 TCP 上扩展实现可靠的一对多模式的数据传输，并且利用 SDN 的全局视图对每个组播会话的转发树 MST 根据网络链路状态进行实时调整来达到拥

塞感知的高鲁棒性的路由转发。由于 MCTCP 在主机端协议与路由两个方面上都具有很高的效率，从而能够达到比传统的可靠组播传输方案更高的性能。利用 MCTCP 在分布式存储系统 HDFS 上实现组播多副本方案，从而优化了分布式存储系统的多副本写操作的性能。相比于 HDFS 原始的链式多副本方案，组播多副本方案可以有效地减少网络冗余报文，提升系统的写操作的性能。该研究成果发表在会议 IEEE/ACM IWQoS'16[103] 和期刊 Journal of Network and Computer Applications (JNCA)[104]。

2、针对分布式存储系统网络资源浪费问题，提出一种动态的网络流量调度方案 MAX，通过提升存储系统中空闲网络带宽资源的利用率来提升存储系统的性能。MAX 主要针对部署两个网络的分布式存储系统场景。由于两个网络（前端与后端网络）之间彼此隔离，因此两个网络之间的空闲网络资源无法共享，从而导致网络资源的浪费。通过在存储节点上加入一个网络调度层，可以在不修改存储系统代码实现的前提下实现跨前端与后端网络之间的流量调度。具体而言，由于前端网络中用户的请求流量不能够到达后端网络，因此，MAX 只能提升系统后台操作的性能，例如恢复与重均衡操作等。MAX 会在前端网络存在空闲带宽时，将后端网络中的部分流量调度到前端从而提升恢复与重均衡的性能。然而，在利用前端网络空闲带宽的时候，需要保证后端网络流量不会影响前端用户请求的性能。通过给前端网络流量设置更高的优先级可以在避免互相影响的前提下充分利用空闲网络带宽。该研究成果已投稿会议 IEEE INFOCOM'18。

3、针对分布式存储系统的安全性问题，提出一种基于全局路由冲突避免机制的匿名通信方案 MIC 和匿名分布式存储系统，进而提升分布式存储系统的安全性。MIC 的主要目标是在引入少量性能开销的情况下实现匿名通信和具有一定的抗流量分析攻击的能力。为了达到该目标，MIC 在网络交换机上进行报文头部的修改，从而隐藏报文真实的发送端与接收端。这种基于网内（in-network）的方案比传统的基于覆盖网络（overlay）的匿名通信方案更加轻量化，从而具有更小的开销。该方案面临着两个技术问题：第一，在网络中修改报文头部信息可能会导致路由冲突。第二，如何在不修改交换机实现的前提下提升抗流量分析攻击的能力。对于第一个问题，通过设计合理的地址变换算法来避免路由冲突，即将不同的流可以携带的地址的集合映射到不重合的空间上。对于第二个问题，通过采用多个子流和局部多播的机制来

提升抗流量分析攻击的能力。利用 MIC 实现匿名的分布式存储系统，使得系统中每个节点之间的彼此匿名，从而可以有效地提升攻击者定位系统关键节点的难度，提升分布式存储系统的安全性。该研究成果发表在会议 ICPP'16[105] 和期刊 IEEE/ACM Transactions on Networking (ToN)[106]。

5.2 研究展望

随着越来越多的应用被迁移到云数据中心内，存储系统需要存储与处理来自不同应用和用户的数据。在多租户的情况下，具有不同需求的租户或者应用将数据存储在统一的存储系统上。因此，保证应用端到端的服务质量需求以及提供灵活可编程的服务能力将是分布式存储系统面临的挑战。同时，提升分布式存储系统的安全性，防止恶意用户嗅探、分析或者窃取数据，破坏存储系统依然是需要研究的重大问题。

针对上述问题，下一步的研究工作将集中在以下几个方面：

1、存储流在网络层的服务质量保证。软件定义存储已经成为当前存储的发展趋势，这要求存储系统具有可编程的策略实施能力。现在的存储系统 I/O 路径冗长而复杂，因此难以实施端到端的策略，例如带宽保证和路由控制。IOFlow[107] 针对这个问题进行了研究，提出了一种类似于软件定义网络的中心化架构，通过控制 I/O 路径中的每个阶段（stage）来实现端到端的控制。其在实现中将网络当作一个抽象层，而并未考虑网络内部的复杂性。因此，需要研究存储流在网络层的服务质量保证技术，从而实现完整的存储端到端策略的实施能力。

2、存储透明的网络层优化。在多应用的环境下，不同类型的存储请求将会在网络层上相互影响，从而导致整体性能较差。例如，IOPS 敏感型（如 Web Server、OLTP 等）与带宽敏感型（如 File Server、备份等）的应用使用统一的存储系统时，其请求将在网络层相互干扰。两种类型的存储 I/O 请求将会在网络中产生两类流量：短流与长流。其中短流的长度很短，将会在网络交换机中被长流阻塞，从而导致很大的时延，最终减少存储请求的 IOPS。而应用一旦在存储系统上部署之后通常不希望再次修改其应用的实现来适应新的存储功能。在不修改存储系统实现的前提下，通过在网络层中智能地对流量分类，并进行相应的调度，如优先级调度（短流优先）等，从

而优化存储系统的整体性能。

3、用户透明的数据放置。很多分布式存储系统为了提升扩展性、避免元数据查找瓶颈，采用哈希函数的方式来进行数据放置，例如 Ceph 采用 CRUSH 算法等。这种方案允许任意节点通过对象名计算出该对象存放的位置，同时允许任意客户端节点直接访问存储节点。然而，该方案在提升了存储系统性能的同时也增加了安全风险。恶意用户可以计算出目标数据所在的存储节点，然而再通过其它攻击方式进行数据窃取或者破坏。为了解决这个问题，可以将数据放置的功能迁移到网络中去。通过在网络交换机上安装相应的转发逻辑即可实现相应对象或者文件的访问，而客户端节点不需要知道对象所在的具体位置，进而可以提升存储系统的安全性。

致　　谢

五年前迷茫地开始了博士学习生涯。五年多的博士学习让我倍感充实、成长良多。从茫然无知的小白，到如今初窥门径的博士毕业生，过程历经辛酸，也充满欢喜。经历过寻找研究方向时的夜夜失眠，发现自己工作与别人冲突时的沮丧气馁，看到论文被拒时的失望悲伤；也经历过寻得思路时的踌躇满志，测试得到满意结果时的得意洋洋，收到论文接收通知时的欣喜若狂。尽管没有取得值得骄傲的成果，但是博士阶段的难忘经历也值得骄傲。五年多的时间里，我收获了名师的指导、真挚的友情、甜蜜的爱情、温暖的亲情，这是我一生中最重要的财富。

感谢我的导师冯丹教授的指导。冯老师以全面系统的专业知识和仰之弥高钻之弥坚的学术水平来指导我如何做科研。她广阔的视野、犀利的视角和对研究前沿的敏感嗅觉成为了我在博士阶段内不断前进的明灯。更加重要的是，冯老师也经常教导我及实验室的所有同学要注重自身德行的培养。太上有立德，其次有立功。只有提高了自己的德行，以后才能在这个社会上做一个有用的人。感谢冯老师在整个博士阶段对我科研与生活上的支持与资助。在实验室中能够用上配置先进的实验平台、参加国际学术会议时得到全额资助以及可以非常“土豪”地每周打羽毛球都让我时刻铭记冯老师的恩情。在此论文完成之际，谨向冯老师致以最衷心的感谢和诚挚的祝福。

感谢王芳教授对我的谆谆教诲。王老师是我在实验室里的小导师，悉心指导我日常的工作科研。博士期间每个小论文都是与王老师讨论交流，并得到了全面专业的指导与改进意见。除了科研，王老师也带领我完成了两年多的项目工作，全面丰富地提升了我自身的能力。做项目过程中，虽然我当时内心是拒绝的，但是现在回头再想，这些项目的经验是我为以后工作积累的巨大财富。王老师深厚的学术造诣、严谨的治学精神和一丝不苟的工作作风深深地感染着我。从她身上我可以感受到一个学者的严谨和务实，这些都让我获益菲浅，并且将终生受用无穷。

感谢华宇教授对我在论文写作与修改上的专业指导。正是由于得到了华老师专业的指导让我在论文写作与投稿过程中避开了很多盲区，才能够更快地写出符合要

求的论文以及找到合适的会议与期刊。感谢实验室谭支鹏老师，总是幽默风趣和蔼可亲，并耐心细致地帮我解答问题。感谢刘景宁、曾令仿、施展、童薇、秦磊华、谢雨来等老师在我科研生活上提供的帮助与支持。感谢熊双莲、王子惠琦老师对我在日常事务、出差报销等方面的耐心帮助，让我可以免于处理大量流程上的事务，专心进行项目科研工作。

感谢博士师兄万勇在我刚进入实验室的时候给我细心的指导，引我进入科研的大门。感谢博士师兄明亮在实验室对我的指导，以及毕业后经常回来请我们吃饭，交流工业界新的技术、知识。感谢 Cappella 组师兄孙海峰、肖飞、吴森、邓继旭、郑超、杜鑫等在实验室对我的指导与帮助。感谢同窗好友刘昌芳，经常一起坐火车回家。感谢与我同一届的研究生同学祖文强、吴辉静、夏江、尹忱承、刘芬、赵玉京、田昊、刘玉、徐鹏、汪凯、苏臻等，和大家一起上课、学习、开会、吃饭、聚餐、郊游的日子很难忘。伴友同乐慰吾心，愿能长久共醉魂。感谢与我一起奋战过的博士李楚、苏楠、周俊、朱春节、程永利、苏毅、谢燕文、解为斌等，因为和大家一起，读博路上的艰辛少了许多。感谢博士师弟史庆宇和刘家豪，宇博帮我完成了许多论文实现上的工作，一起交流也非常有收获；豪博是我的室友，经常带我来实验室。感谢系统组的所有同学，他们为实验室营造了良好的科研环境。还有，每周最开心的事情就是周六打羽毛球，感谢夏文、周炜、李楚、苏毅、徐杰、史庆宇等球友，和你们打球非常开心。感谢一同毕业的黄方亭和张宇成博士，毕业过程中给我提供了大量帮助。

感谢我的妻子魏萍多年来的不离不弃，与你在华科相识是我读博过程中最大的收获。感谢你在梧桐树下尴尬的表白之后说了“好”，感谢你面对一无所有的我的求婚时说了“愿意”。感谢你为了我放弃了来之不易的工作岗位。感谢你包容我这个不懂浪漫的工科生。感谢与你相遇。另外，特别感谢岳父岳母对我的关爱与支持。

感谢我的父母多年来对我学业的支持。这么多年来你们一直默默地支持我，没有追问我毕业时间，没有给我任何压力，只有默默的付出。你们的支持是我最坚实的后盾，让我得以全身心投入学业。焉得谖草，言树之背，养育之恩，无以回报，你们永远健康快乐是我最大的心愿。

感谢所有对我给予帮助的人。

2017 年秋于华中科技大学

参考文献

[1] 覃灵军. 基于对象的主动存储关键技术研究: [博士学位论文]. 湖北武汉: 华中科技大学, 2006

[2] 冯丹, 史伟, 覃灵军. 基于对象存储系统的对象文件系统设计. 华中科技大学学报: 自然科学版, 2006, 34(12):47–49

[3] Blumer A D. The parallel virtual file system[PhD Dissertation]. Clemson, South Carolina, US: Clemson University, 1994

[4] Haddad I F. PVFS: A Parallel Virtual File System for Linux Clusters. Linux J., 2000, 2000(80es)

[5] Schwan P, et al. Lustre: Building a file system for 1000-node clusters. in: Proceedings of the 2003 Linux symposium, Ottawa, Ontario, CA, July 23–26, 2003, 380–386

[6] Ghemawat S, Gobioff H, Leung S T. The Google File System. in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA: ACM, October 19-22, 2003, 29–43

[7] Shvachko K, Kuang H, Radia S, et al. The Hadoop Distributed File System. in: Proceedings of Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Villiage, NV, USA: IEEE Computer Society, May 3-7, 2010, 1–10

[8] Weil S A, Brandt S A, Miller E L, et al. Ceph: A Scalable, High-performance Distributed File System. in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, Seattle, Washington: USENIX Association, November 6-8, 2006, 307–320

[9] Weil S A, Brandt S A, Miller E L, et al. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. in: Proceedings of the ACM/IEEE Conference on Supercomputing, Tampa, Florida: ACM, November 11-17, 2006, 1–12

[10] Weil S A, Leung A W, Brandt S A, et al. RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters. in: Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing, Reno, Nevada: ACM, November 11, 2007, 35–44

[11] Lu Y, Shu J, Chen Y, et al. Octopus: an RDMA-enabled Distributed Persistent Memory File System. in: Proceedings of 2017 USENIX Annual Technical Conference (USENIX ATC 17), Santa Clara, CA: USENIX Association, July 12–14, 2017, 773–785

[12] Kreutz D, Ramos F M V, Ver ssimo P E, et al. Software-Defined Networking: A Comprehensive Survey. the IEEE, January, 2015, 103(1):14–76

[13] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling Innovation in Campus Networks. SIGCOMM Comput. Commun. Rev., April, 2008, 38(2):69–74

[14] 左青云, 陈鸣, 赵广松. 基于 OpenFlow 的 SDN 技术研究. 软件学报, 2013, 24(5):1078–1097

[15] 李丹, 陈贵海, 任丰原. 数据中心网络的研究进展与趋势. 计算机学报, 2014, 37(2):259–274

[16] Heller B, Seetharaman S, Mahadevan P, et al. ElasticTree: Saving Energy in Data Center Networks. in: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, San Jose, California: USENIX Association, April 28-30, 2010, 17–31

[17] Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera: Dynamic Flow Scheduling for Data Center Networks. in: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, San Jose, California: USENIX Association, April 28-30, 2010, 1–15

[18] Benson T, Anand A, Akella A, et al. MicroTE: Fine Grained Traffic Engineering for Data Centers. in: Proceedings of the Seventh Conference on Emerging Networking EXperiments and Technologies, Tokyo, Japan: ACM, December 6-9, 2011, 1–12

[19] Wang R, Butnariu D, Rexford J. OpenFlow-based Server Load Balancing Gone Wild. in: Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Boston, MA: USENIX Association, March 29, 2011, 1–6

[20] Handigol N, Seetharaman S, Flajslik M, et al. Plug-n-Serve: Load-balancing web traffic using OpenFlow. ACM Sigcomm Demo, August 17-21, 2009, 4(5):6–7

[21] Liu H H, Wu X, Zhang M, et al. zUpdate: Updating Data Center Networks with Zero Loss. in: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, Hong Kong, China: ACM, August 12-16, 2013, 411–422

[22] Jain S, Kumar A, Mandal S, et al. B4: Experience with a Globally-deployed Software Defined Wan. in: Proceedings of the ACM SIGCOMM Conference, Hong Kong, China: ACM, August 12-16, 2013, 3–14

[23] Hong C Y, Kandula S, Mahajan R, et al. Achieving High Utilization with Software-driven WAN. in: Proceedings of the ACM SIGCOMM Conference, Hong Kong, China: ACM, August 12-16, 2013, 15–26

[24] Stabler G, Rosen A, Goasguen S, et al. Elastic IP and Security Groups Implementation Using OpenFlow. in: Proceedings of the 6th International Workshop on Virtualization Technologies in Distributed Computing Date, Delft, The Netherlands: ACM, June 18, 2012, 53–60

[25] Wang K, Qi Y, Yang B, et al. LiveSec: Towards Effective Security Management in Large-Scale Production Networks. in: Proceedings of 2012 32nd International Conference on Distributed Computing Systems Workshops, Macau, China, June 18-21, 2012, 451-460

[26] Braga R, Mota E, Passito A. Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. in: Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks, Washington, DC, USA: IEEE Computer Society, October 10-14, 2010, 408–415

[27] Giotis K, Androulidakis G, Maglaris V. Leveraging SDN for Efficient Anomaly Detection and Mitigation on Legacy Networks. in: Proceedings of 2014 Third European Workshop on Software Defined Networks, London, UK, September 1-3, 2014, 85-90

[28] Kumari W, McPherson D R. Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF). RFC 5635, August, 2009. https://rfc-editor.org/rfc/rfc5635.txt

[29] Jafarian J H, Al-Shaer E, Duan Q. Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking. in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, Helsinki, Finland: ACM, August 13, 2012, 127–132

[30] Shin S, Gu G. CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). in: Proceedings of ICNP, Austin, TX, USA, October 30 - November 2, 2012, 1-6

[31] Matias J, Garay J, Mendiola A, et al. FlowNAC: Flow-based Network Access Control. in: Proceedings of 2014 Third European Workshop on Software Defined Networks, London, UK, September 1-3, 2014, 79-84

[32] Cully B, Wires J, Meyer D, et al. Strata: Scalable High-performance Storage on Virtualized Non-volatile Memory. in: Proceedings of the 12th USENIX Conference on File and Storage Technologies, Santa Clara, CA: USENIX Association, February 17-20, 2014, 17–31

[33] Wires J, Warfield A. Mirador: An Active Control Plane for Datacenter Storage. in: Proceedings of 15th USENIX Conference on File and Storage Technologies (FAST 17), Santa Clara, CA: USENIX Association, February 27 - March 02, 2017, 213–228

[34] Shiraki O, Nakagawa Y, Hyoudou K, et al. Managing storage flows with SDN approach in I/O converged networks. in: Proceedings of IEEE Globecom Workshops (GC Wkshps), Atlanta, GA, USA, December 9-13, 2013, 890-895

[35] SDN Storage Demo Synopsis: Plexxi and SolidFire. https://www.sdxcentral.com/resources/sdn-demofriday/2013-demofriday-archives/plexxi-solidfire-sdn-storage-with-qos/, 2013

[36] Rizvi S, Li X, Wong B, et al. Mayflower: Improving Distributed Filesystem Performance Through SDN/Filesystem Co-Design. in: Proceedings of IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, June 27-30, 2016, 384-394

[37] Sun P, Wen Y, Duong T N B, et al. MetaFlow: a Scalable Metadata Lookup Service for Distributed File Systems in Data Centers. IEEE Transactions on Big Data, September 21, 2016, PP(99):1–14

[38] Hua Y, Liu X, Feng D. Smart In-network Deduplication for Storage-aware SDN. in: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, Hong Kong, China: ACM, August 12-16, 2013, 509–510

[39] Hua Y, He W, Liu X, et al. SmartEye: Real-time and efficient cloud image sharing for disaster environments. in: Proceedings of 2015 IEEE Conference on Computer Communications (INFOCOM), Kowloon, Hong Kong, April 26-May 1, 2015, 1616-1624

[40] Chien C. Method and system for transmission path optimization in a storage area network, December 8, 2016. https://www.google.com.hk/patents/US20160357457. US Patent App. 15/140,511

[41] PUTTAGUNTA K, Rao S, MOHAN R. Protocol agnostic storage access in a software defined network topology, September 7, 2015. https://www.google.com.hk/patents/WO2015065476A1?cl=en. WO Patent App. PCT/US2013/068,073

[42] Narayanan R, Xu K. Systems and methods to improve read/write performance in object storage applications, November 10, 2016. https://www.google.com.hk/patents/US20160330281. US Patent App. 14/706,803

[43] Xia Y, Ng T S E, Sun X S. Blast: Accelerating high-performance data analytics applications by optical multicast. in: Proceedings of 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, April 26-May 1, 2015, 1930-1938

[44] Benson T, Akella A, Maltz D A. Network Traffic Characteristics of Data Centers in the Wild. in: Proceedings of IMC, New York, USA: ACM, November 1-30, 2010, 267–280

[45] Speakman T, Crowcroft J, Gemmell J, et al. PGM Reliable Transport Protocol Specification, 2001. RFC3208

[46] Rizzo L. Pgmcc: A TCP-friendly Single-rate Multicast Congestion Control Scheme. in: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Stockholm, Sweden: ACM, August 28-September 1, 2000, 17–28

[47] Lehman L, Garland S, Tennenhouse D. Active reliable multicast. in: Proceedings of INFOCOM, San Francisco, CA, USA, March 29-April 2, 1998, 581-589

[48] Adamson B, Bormann C, Handley M, et al. NACK-Oriented Reliable Multicast (NORM) Transport Protocol, 2009. rfc5740

[49] Liang S, Cheriton D. TCP-SMO: extending TCP to support medium-scale multicast applications. in: Proceedings of INFOCOM, New York, NY, USA, June 23-27, 2002, 1356-1365

[50] Floyd S, Jacobson V, Liu C G, et al. A reliable multicast framework for light-weight sessions and application level framing. IEEE/ACM Transactions on Networking, December, 1997, 5(6):784–803

[51] Paul S, Sabnani K K, Lin J C H, et al. Reliable multicast transport protocol (RMTP). Selected Areas in Communications, IEEE Journal on, April, 1997, 15(3):407–421

[52] Yavatkar R, Griffoen J, Sudan M. A Reliable Dissemination Protocol for Interactive Collaborative Applications. in: Proceedings of the Third ACM International Conference on Multimedia, San Francisco, California, USA: ACM, November 5-9, 1995, 333–344

[53] Li D, Xu M, Zhao M, et al. RDCM: Reliable data center multicast. in: Proceedings of Proceedings IEEE INFOCOM, Shanghai, China, April 10-15, 2011, 56-60

[54] Estrin D, Farinacci D, Helmy A, et al. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification, 1997. RFC2117

[55] Cain B, Deering D S E, Fenner B, et al. Internet Group Management Protocol, Version 3. IETF RFC 3376, 2015

[56] Iyer A, Kumar P, Mann V. Avalanche: Data center Multicast using software defined networking. in: Proceedings of 2014 Sixth International Conference on Communication Systems and Networks (COMSNETS), Bangalore, India, January 6-10, 2014, 1-8

[57] Yang Y, Qin Z, Li X, et al. OFM: A Novel Multicast Mechanism Based on OpenFlow. Advances in Information Sciences and Service Sciences, 2012, 4(9):278–286

[58] Marcondes C A C, Santos T P C, Godoy A P, et al. CastFlow: Clean-slate multicast approach using in-advance path processing in programmable networks. in: Proceedings of 2012 IEEE Symposium on Computers and Communications (ISCC), Cappadocia, Turkey, July 1-4, 2012, 94-101

[59] Ge J, Shen H, Yuepeng E, et al. An OpenFlow-Based Dynamic Path Adjustment Algorithm for Multicast Spanning Trees. in: Proceedings of Security and Privacy in Computing and Communications (TrustCom), Melbourne, VIC, Australia, July 16-18, 2013, 1478-1483

[60] Shen S H, Huang L H, Yang D N, et al. Reliable multicast routing for software-defined networks. in: Proceedings of 2015 IEEE Conference on Computer Communications (INFOCOM), Kowloon, Hong Kong, April 26-May 1, 2015, 181-189

[61] Dijkstra E W. A Note on Two Problems in Connexion with Graphs. Numer. Math., December, 1959, 1(1):269–271

[62] Floyd R W. Algorithm 97: Shortest Path. Commun. ACM, June, 1962, 5(6):345–349

[63] Takahashi H, Matsuyama A. An Approximate Solution for the Steiner Problem in Graphs. Math.Japonica, 1980, 24(6):573–577

[64] Wang Z, Crowcroft J. Quality-of-service routing for supporting multimedia applications. IEEE Journal on Selected Areas in Communications, September, 1996, 14(7):1228–1234

[65] Handigol N, Heller B, Jeyakumar V, et al. Reproducible Network Experiments Using Container-based Emulation. in: Proceedings of CoNEXT, Nice, France: ACM, December 10-13, 2012, 253–264

[66] Pfaff B, Pettit J, Koponen T, et al. The Design and Implementation of Open vSwitch. in: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, Oakland, CA: USENIX Association, May 4-6, 2015, 117–130

[67] Al-Fares M, Loukissas A, Vahdat A. A Scalable, Commodity Data Center Network Architecture. in: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, Seattle, WA, USA: ACM, August 17-22, 2008, 63–74

[68] Alizadeh M, Edsall T. On the Data Path Performance of Leaf-Spine Datacenter Fabrics. in: Proceedings of the 2013 IEEE 21st Annual Symposium on High-Performance Interconnects, San Jose, CA, USA: IEEE Computer Society, August 21-23, 2013, 71–74

[69] Alizadeh M, Greenberg A, Maltz D A, et al. Data Center TCP (DCTCP). in: Proceedings of the ACM SIGCOMM 2010 Conference, New Delhi, India: ACM, August 30-September 3, 2010, 63–74

[70] Greenberg A, Hamilton J R, Jain N, et al. VL2: A Scalable and Flexible Data Center Network. in: Proceedings of SIGCOMM, Barcelona, Spain: ACM, August 16-21, 2009, 51–62

[71] Arnold J. OpenStack Swift: Using, Administering, and Developing for Swift Object Storage, 1st ed. Sebastopol, CA: O'Reilly Media, Inc., 2014

[72] OpenFlow Switch Specification Version 1.3.1. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf, 2012

[73] Rosen E, Viswanathan A, Callon R. Multiprotocol Label Switching Architecture. RFC 3031, 2001

[74] Moller U, Cottrell L. Mixmaster Protocol – Version 2. Draft, January 2000. http://www.eskimo.com/ rowdenw/crypt/Mix/draft-moeller-mixmaster2-protocol-00.txt

[75] Danezis G, Dingledine R, Mathewson N. Mixminion: design of a type III anonymous remailer protocol. in: Proceedings of Security and Privacy, Berkeley, CA, USA, May 11-14, 2003, 2-15

[76] Reiter M K, Rubin A D. Crowds: Anonymity for Web Transactions. ACM Transactions on Information and System Security (TISSEC), November, 1998, 1(1):66–92

[77] Dingledine R, Mathewson N, Syverson P. Tor: The Second-generation Onion Router. in: Proceedings of USENIX Security Symposium, San Diego, CA: USENIX Association, August 9-13, 2004, 1–17

[78] Wolinsky D I, Corrigan-Gibbs H, Ford B, et al. Dissent in Numbers: Making Strong Anonymity Scale. in: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, Hollywood, CA, USA: USENIX Association, October 8-10, 2012, 179–192

[79] The Untold Story of the Target Attack Step by Step. https://aroundcyber.files.wordpress.com/2014/09/aorato-target-report.pdf, 2014

[80] Guo C, Lu G, Li D, et al. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. in: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, Barcelona, Spain: ACM, August 16-21, 2009, 63–74

[81] Kirch J. Virtual machine security guidelines Version 1.0. in: Proceedings of The center for Internet Security, 2007

[82] Ristenpart T, Tromer E, Shacham H, et al. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. in: Proceedings of the 16th ACM Conference on Computer and Communications Security, Chicago, Illinois, USA: ACM, November 9-13, 2009, 199–212

[83] Chaum D L. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Commun. ACM, February, 1981, 24(2):84–90

[84] Chaum D. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. J. Cryptol., March, 1988, 1(1):65–75

[85] Furukawa J, Sako K. An Efficient Scheme for Proving a Shuffle. in: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, London, UK, UK: Springer-Verlag, August 19-23, 2001, 368–387

[86] Neff C A. A Verifiable Secret Shuffle and Its Application to e-Voting. in: Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, PA, USA: ACM, November 5-8, 2001, 116–125

[87] Gulcu C, Tsudik G. Mixing Email with Babel. in: Proceedings of the 1996 Symposium on Network and Distributed System Security (SNDSS '96), Washington, DC, USA: IEEE Computer Society, February 22-23, 1996, 2–16

[88] Goldschlag D M, Reed M G, Syverson P F. Hiding Routing Information. in: Proceedings of the International Workshop on Information Hiding, London, UK, UK: Springer-Verlag, May 30-June 01, 1996, 137–150

[89] 刘鑫. 基于 Tor 网络的匿名通信研究: [博士学位论文]. 上海: 华东师范大学, 2011

[90] 王伟平, 陈建二, 陈松乔. 匿名通信中短距离优先分组重路由方法的研究. 软件学报, 2004, 15(4):561–570

[91] Rennhard M, Plattner B. Introducing MorphMix: Peer-to-peer Based Anonymous Internet Usage with Collusion Detection. in: Proceedings of WPES, Washington, DC: ACM, November 21-21, 2002, 91–102

[92] Freedman M J, Morris R. Tarzan: A Peer-to-peer Anonymizing Network Layer. in: Proceedings of CCS, Washington, DC, USA: ACM, November 18-22, 2002, 193–206

[93] Le Blond S, Choffnes D, Zhou W, et al. Towards Efficient Traffic-analysis Resistant Anonymity Networks. in: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, Hong Kong, China: ACM, August 12-16, 2013, 303–314

[94] Le Blond S, Choffnes D, Caldwell W, et al. Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems. in: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, London, United Kingdom: ACM, August 17-21, 2015, 639–652

[95] Levine B N, Shields C. Hordes: A Multicast Based Protocol for Anonymity. Journal of Computer Security, September, 2002, 10(3):213–240

[96] Goel S, Robson M, Polte M, et al. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003

[97] Sherwood R, Bhattacharjee B, Srinivasan A. P5: A Protocol for Scalable Anonymous Communication. J. Comput. Secur., December, 2005, 13(6):839–876

[98] Katti S, Cohen J, Katabi D. Information Slicing: Anonymity Using Unreliable Overlays. in: Proceedings of NSDI, Cambridge, MA: USENIX Association, April 11-13, 2007, 1–14

[99] Hsiao H C, Kim T H J, Perrig A, et al. LAP: Lightweight Anonymity and Privacy. in: Proceedings of the 2012 IEEE Symposium on Security and Privacy, Washington, DC, USA: IEEE Computer Society, May 20-25, 2012, 506–520

[100] Birrell A D, Nelson B J. Implementing Remote Procedure Calls. ACM Transactions on Computer Systems (TOCS), February, 1984, 2(1):39–59

[101] Thurlow R. RPC: Remote Procedure Call Protocol Specification Version 2. RFC 5531, 2009

[102] Berde P, Gerola M, Hart J, et al. ONOS: Towards an Open, Distributed SDN OS. in: Proceedings of HotSDN, Chicago, Illinois, USA: ACM, August 22, 2014, 1–6

[103] Zhu T, Wang F, Hua Y, et al. MCTCP: Congestion-aware and robust multicast TCP in Software-Defined networks. in: Proceedings of 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), Beijing, China, June 20-21, 2016, 1-10

[104] Zhu T, Feng D, Wang F, et al. A congestion-aware and robust multicast protocol in SDN-based data center networks. Journal of Network and Computer Applications, 2017, 95(Supplement C):105 – 117

[105] Zhu T, Feng D, Hua Y, et al. MIC: An Efficient Anonymous Communication System in Data Center Networks. in: Proceedings of 2016 45th International Conference on Parallel Processing (ICPP), Philadelphia, PA USA, August 16-19, 2016, 11-20

[106] Zhu T, Feng D, Wang F, et al. Efficient Anonymous Communication in SDN-Based Data Center Networks. IEEE/ACM Transactions on Networking, 2017, 25(6):1–14

[107] Thereska E, Ballani H, O'Shea G, et al. IOFlow: A Software-defined Storage Architecture. in: Proceedings of Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, New York, NY, USA: ACM, November 03-06, 2013, 182–196

附录 1　攻读博士学位期间发表的学术论文目录

[1] **Tingwei Zhu**, Dan Feng, Fang Wang, Yu Hua, Qingyu Shi, Jiahao Liu, Yongli Cheng, Yong Wan. Efficient Anonymous Communication in SDN-based Data Center Networks. IEEE/ACM Transactions on Networking **(ToN)**, Volume 25, Issue 6, 2017: 1-14 （CCF A 类期刊，SCI 检索）

[2] **Tingwei Zhu**, Dan Feng, Yu Hua, Fang Wang, Qingyu Shi, Jiahao Liu. MIC: An Efficient Anonymous Communication System in Data Center Networks. Proceeding of the 45th International Conference on Parallel Processing (**ICPP'16**), Philadelphia, PA USA, August 16-19, 2016: 11-20 （CCF B 类会议，录用率：21.1%）

[3] **Tingwei Zhu**, Fang Wang, Yu Hua, Dan Feng, Yong Wan, Qingyu Shi, Yanwen Xie. MCTCP: Congestion-aware and robust multicast TCP in Software-Defined networks. Proceeding of IEEE/ACM International Symposium on Quality of Service (**IWQoS'16**), Beijing, China, June 20-21, 2016: 1-10 （CCF B 类会议，录用率：20.6%）

[4] **Tingwei Zhu**, Dan Feng, Fang Wang, Yu Hua, Qingyu Shi, Yanwen Xie, Yong Wan, A congestion-aware and robust multicast protocol in SDN-based data center networks, Journal of Network and Computer Applications (**JNCA**), Volume 95, 2017: 105-117（CCF C 类期刊，SCI 检索）

[5] Yong Wan, Dan Feng, Fang Wang, and **Tingwei Zhu**. A Dedicated Serialization Scheme in Homogeneous Cluster RPC Communication, International Conference on Grid and Pervasive Computing (GPC'13). May, 2013: 403-412（EI 索引）

处于在审的论文：

[6] **Tingwei Zhu**, Dan Feng, Fang Wang, Yu Hua, Yi Su, Qingyu Shi. Optimizing the Recovery and Rebalancing Performance of Cloud Storage Systems using MAX. （投稿于 IEEE INFOCOM'18）

附录 2　攻读博士学位期间申请的发明专利和著作权

[1] 王芳, 冯丹,朱挺炜, 万勇. 一种同构环境下的 RPC 数据传输方法及系统. 中国国家发明专利, 授权公告号: CN104135496B, 授权公告日期: 2017.08.18

[2] 王芳, 冯丹,朱挺炜, 史庆宇, 万勇. 一种基于 SDN 环境的数据的可靠组播传输方法. 中国国家发明专利, 申请号: 2014107873117. 申请日期: 2014.12.17

[3] 王芳, 冯丹,朱挺炜, 史庆宇, 刘家豪, 万进. 一种基于 SDN 环境的匿名通信方法. 中国国家发明专利, 申请号: 2016103451967. 申请日期: 2016.05.23

[4] 王芳, 冯丹,朱挺炜, 史庆宇, 刘家豪, 万进. 基于 SDN 环境的匿名通信软件[简称：MIC]1.0. 中国国家软件著作权, 登记号: 2016SR223458, 登记日期: 2016.08.17

[5] 王芳, 冯丹, 万勇,朱挺炜. 一种分区间的 RPC 超时值自适应调整方法. 中国国家发明专利, 授权公告号: CN104348639B. 授权公告日期: 2017.08.25

[6] 王芳, 冯丹, 史庆宇,朱挺炜, 万勇. 一种 SDN 环境下的源端可控组播数据传输方法及系统. 中国国家发明专利, 申请号: 2015102457707. 申请日期: 2015.05.14

[7] 王芳, 冯丹, 刘芬, 田昊, 朱挺炜, 祖文强. 一种分布式文件系统客户端元数据缓存优化方法. 中国国家发明专利, 授权公告号: CN104113587B, 授权公告日期: 2017.08.01

[8] 王芳, 冯丹, 明亮, 付秋雷, 万勇,朱挺炜. 用户态下基于 RDMA 协议的远程过程调用实现方法. 中国国家发明专利, 授权公告号: CN102546612B, 授权公告日期: 2015.07.08

附录 3　攻读博士学位期间参与的科研项目

[1] 面向复杂应用环境的数据存储系统理论与技术基础研究. 国家重点基础研究发展计划 (973 计划). 项目编号: 2011CB302301. 2011-2015. (已结题)

[2] 存储 N+2 SDS 关键技术合作项目. 企业合作项目. 2014. (已结题)