TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 06/10 pp51-64
Volume 19, Number 1, February 2014

On Peer-Assisted Data Dissemination in Data Center Networks: Analysis and Implementation

Yaxiong Zhao*, Jie Wu, and Cong Liu

Abstract: Data Center Networks (DCNs) are the fundamental infrastructure for cloud computing. Driven by the massive parallel computing tasks in cloud computing, one-to-many data dissemination becomes one of the most important traffic patterns in DCNs. Many architectures and protocols are proposed to meet this demand. However, these proposals either require complicated configurations on switches and servers, or cannot deliver an optimal performance. In this paper, we propose the *peer-assisted data dissemination* for DCNs. This approach utilizes the rich physical connections with high bandwidths and mutli-path connections, to facilitate efficient one-to-many data dissemination. We prove that an optimal P2P data dissemination schedule exists for FatTree, a specially-designed DCN architecture. We then present a theoretical analysis of this algorithm in the general multi-rooted tree topology, a widely-used DCN architecture. Additionally, we explore the performance of an intuitive line structure for data dissemination. Our analysis and experimental results prove that this simple structure is able to produce a comparable performance to the optimal algorithm. Since DCN applications heavily rely on virtualization to achieve optimal resource sharing, we present a general implementation method for the proposed algorithms, which aims to mitigate the impact of the potentially-high churn rate of the virtual machines.

Key words: data center networks; cloud computing; P2P; scheduling; peer-assisted data dissemination

1 Introduction

Cloud computing has become an transformational industry trend. Data Center Networks (DCNs), as the fundamental infrastructures of cloud computing, are gaining tremendous interest both in industrial and academic research. Many important and interesting networking problems are studied. Among all of these topics, we notice that one is frequently mentioned or directly considered in a large body of literature, that is, the support of efficient *one-to-many data dissemination* in DCNs. One-to-many data dissemination refers to delivering a bulk of data from a single source to multiple, and potentially a large amount of receivers. This problem is important because: (1) this traffic pattern is common in many important DCN applications, like software update[1] and data shuffle of massively-parallel computing[2]; (2) historically, supporting such traffic efficiently is hard, as is embodied in the Internet multicast problem. These two driving factors are discussed in various recent works[3-6].

There are proposals that solve the problem through an *architectural approach*, i.e., design new DCN architectures that provide better multicast support. For instance, in BCube[7], DCell[6], and CamCube[8], the authors explicitly demonstrated that their architecture admits natively-supported efficient multicast. Coupling with the careful protocol designs, efficient multicast is

• Yaxiong Zhao is with Google Inc., Mountain View, CA 94043, USA. This work was done while the author was with Temple University, Philadelphia, PA 19122, USA. E-mail: yaxiongzhao@google.com.
• Jie Wu is with Temple University, Philadelphia, PA 19122, USA. E-mail: jiewu@temple.edu.
• Cong Liu is with Sun Yat-Sen University, Guangzhou 510275, China. E-mail: gzcong@gmail.com.
* To whom correspondence should be addressed.
Manuscript received: 2013-12-13; accepted: 2013-12-20

achieved, as shown in the prototype implementation and simulation studies of these papers. However, we notice that such systems put routing tasks on commodity servers, which results in extended processing delay[6, 7]. What is worse is that, the configuration of such systems is more complicated than the conventional DCNs. Although an automatic network configuration algorithm is proposed[9], its effectiveness has not yet been validated in large-scale deployment.

Another line of research is the *software approach*, which aims to alter the Internet multicast protocol and make it work better in DCNs[3]. This method provides heuristics based on the rich multi-path connections in DCNs. It needs to alter the protocol stack implementation that resides inside the Operating System (OS) kernel. This approach lacks support from the existing industry standard, and the reliability of the approach has not been verified.

It is difficult for these two approaches to provide efficient one-to-many data dissemination in a cost-effective manner. We thus consider leaving the existing standards intact and employing application layer overlay technologies. However, the application layer overlay technology has only been used in wide area networks. Its performance in a highly performance-demanding environment, like DCN, is still in question. To make such a design a viable choice for DCNs, we must achieve the following goals, ranked on the basis of priority: (1) high performance, (2) high fault-tolerance, and (3) low complexity.

In this paper, we avoid the architectural approach and focus on widely-used DCN architectures: *FatTree* and *Multi-Rooted Tree* (MRT)[10]. We propose that the *peer-assisted transmission* can be used for one-to-many data dissemination in DCNs. This technique is able to utilize the rich connections in DCNs to provide fast data delivery. Figure 1 illustrates this idea. In the figure, the source distributes the data to different servers, and the servers exchange data between each other to accelerate the dissemination speed. This

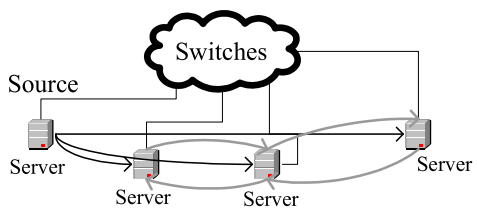


Fig. 1 The one-to-many communication scenario in a DCN.

approach is efficient. As an example, a recent industrial experience[1] of using BitTorrent[11] to do software updates in a large DCN (thousands of servers) indicates that, although BitTorrent is designed for wide area networks and misses many key properties of DCNs, it achieves a speed-up of more than 75× compared with the conventional Client/Server approach.

In this paper, we conduct theoretical analyses and show that an optimal solution exists for FatTree, a specially-designed DCN topology. Additionally, the fact that most DCNs do not follow the particular design makes its use limited in practice. We then slack the constraints on the network topology and focus on the MRT topology. We show that in the single-chunk case, there is an optimal scheduling that can be found in polynomial time. We then present a method to apply the optimal algorithm in MRTs by utilizing the structural property of MRTs.

However, this solution may not scale well in practice, since it requires a central scheduler that has a global view of all peers. Inspired by an existing industrial initiative[1], we explore the effectiveness of a simple line structure, i.e., a line of peers that are connected sequentially. In the first look, this design restricts the potential of P2P transmission and cannot achieve optimal throughput. However, in real world applications, software failures, instead of the optimality of the algorithm, are more likely to kill the performance.

Our contributions in this paper are as follows:

- We conduct thorough theoretical analyses on peer-assisted data dissemination for DCNs. We give an optimal scheduling method for the FatTree topology. We also present a method for adapting the optimal algorithm to work for general MRTs.
- We propose a simple line-structure-based data dissemination scheme. We analyze several structure manipulation primitives for this structure.
- We give the detailed design and implementation techniques for applying the proposed algorithms. We also evaluate the performance of the proposed algorithm by using simulation studies.

2 Related Work

Many novel data center network topologies and architectures have been designed[6, 7, 10, 12, 13]. In Ref. [10], the authors promote using the FatTree[14] structure as the DCN topology. FatTree uses only homogeneous Commodity-Off-The-Shelf (COTS) switches and delivers non-blocking bisection bandwidth

between all servers. FatTree can be seen as a special instance of MRT. Both are the targeted topologies of this paper. Server-centric DCNs[6-8] equip every server with multiple Network Interfaces Cards (NICs) and let them do routing. Such architectures still use switches, but they enable more flexible and complex communication patterns than conventional architectures do. One-to-many communication is made practical in these architectures. A serious disadvantage is that the routing and forwarding performance of servers are no match to COTS switches. In practice, implementing such a system is much more difficult.

The authors in Ref. [15] studied the *topology-aware P2P file distribution problem* on general topology. The problem is essentially a generalized version of our problem and is NP-hard. The most important difference is that they assume non-equal link capacities, which cause non-equal upload capacities and an extremely complicated combinatorial structure for deciding the upload capacity of chunk transfer. The approximate solution proposed in the paper is too complicated to be used in practice.

The scheduling of P2P data dissemination in an overlay network with non-equal upload capacities was studied in Ref. [16]. Our theoretical analysis on the optimal solution for data dissemination in the FatTree topology is based on the work in Ref. [17], which studies the optimal scheduling of P2P file dissemination in a complete graph. Murder[1] is a P2P software updating tool implemented by Twitter Inc. Murder uses BitTorrent[11] for P2P transmission. Interesting enough, the transmission pattern between peers using Murder converges to a line structure, as indicated in the public presentation[1].

As far as we know, Ref. [5] is the only work that studies the same problem as ours. The authors in Ref. [5] discussed several questions concerning the efficiency of the BitTorrent-like protocol in DCNs and proposed potential improvements. They have given a randomized algorithm based on the analysis given in Ref. [17], which is asymptotically optimal in the FatTree topology. However, they have not considered general multi-rooted tree topologies.

3 Problem Settings

3.1 General data center network topology

Modern DCNs[18] employ the MRT topology. We give an illustration of an example of such a DCN in Fig. 2. In an MRT topology, *servers* are interconnected by 3 levels of switches. A fixed number of servers are packaged into a *rack* and are interconnected by a *Top of Rack* (ToR) switch, which delivers non-blocking bandwidth between the servers. Multiple racks are interconnected by connecting their ToR switches to an *Edge of Row* (EoR) switch, which forms a *row*. A rack can connect with other EoR switches to improve fault-tolerance. Every EoR switch of all rows is connected with multiple *Core* switches, which forms a connected network of all servers.

The *downlinks* of the switches connect with servers or switches at a lower level, whereas the *uplinks* connect with a higher level. In current industry practice, ToR switches have 1 Gbps links connecting to servers, and EoR switches have 10 Gbps links connecting to ToR switches and Core switches. We assume that all links are *fully duplex*, meaning that traffic moving in both directions on every link can reach the maximal bandwidth, which is called the *capacity* of the link. The capacities of the uplinks are at least as large as the downlinks.

We have not found formal definitions of MRT in literature. Since most DCNs use 3 levels of switches, we fixed the depth of MRTs to 3, which has 3 levels of inner node and 1 level of leaf nodes. The *oversubscription* of a switch is the ratio of the aggregate bandwidth of the downlinks to the uplinks. MRT is formed by connecting multiple *Recursive MRTs* (RMRTs) with multiple switches. RMRT is a recursively defined structure. RMRT$(1, i, o, f)$ is a singly-rooted tree, formed by connecting f servers with a single switch. Figure 3 shows a sample. An RMRT(k, i, o, f) is constructed by connecting o RMRT$(k-1, i, o, f)$. Each root of the i subtrees exposes f uplinks to connect with f upper-level switches. MRT(k, i, o, f) is formed by connecting all

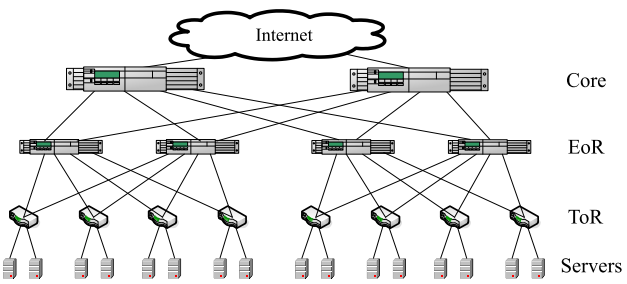


Fig. 2 The abstract view of a multi-rooted tree DCN topology.

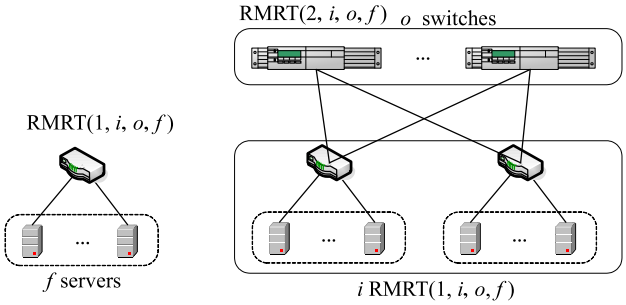


Fig. 3 The construction of RMRT.

of the uplinks of the roots of an arbitrary number of RMRT($k-1, i, o, f$). The number of root switches is determined in such a way where the number of their links is equal to the total number of the uplinks of the roots of the RMRT($k-1, i, o, f$)s. In this paper, we fixed the k of the MRT(k, i, o, f) to 3, since most DCNs that are used nowadays employ a 3-level switch network.

The *oversubscription* of a switch is defined as the ratio of the aggregate bandwidth of its uplinks to downlinks. For example, the oversubscription of an EoR switch is $(o \times B_u)/(f \times B_d)$. Note that a switch that does not have uplinks has no oversubscription. So the root switches do not have this definition in MRT. The oversubscription of the network is defined by the multiplication of the oversubscription of the switches on the successive levels. For example, the oversubscription of the network in Fig. 2 is the product of the oversubscriptions of the ToR and EoR switches. In order to achieve an oversubscription of 1, all switches in the network need to ensure that their oversubscriptions are 1.

3.2 FatTree

FatTree[10] can be seen as a special case of MRT, which is adapted from Ref. [14]. Like the MRTs that we introduced previously, we restrict the switches to 3 levels for the FatTree. We give an example of FatTree in Fig. 4. A key feature of FatTree is that it uses only one type of switch in the entire network, usually the cheap COTS switch. The links connecting the switches in a FatTree all have identical bandwidth. ToR and EoR switches use an equal number of uplinks and downlinks to deliver non-blocking bandwidth between all servers. With k-port switches, FatTree can form a $k/2$-ary 3-tree (3 levels of switches), with $5k^2/4$ switches and $k^3/4$ servers. Another important property of FatTree is that its oversubscription is 1. We will not

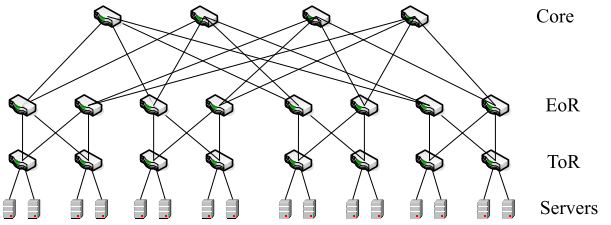


Fig. 4 The illustration of the FatTree topology. Note that all switches are homogeneous.

go into details here for the sake of space. More details can be found in Ref. [10].

3.3 Peer-assisted data dissemination problem

We use *peer* to refer to the servers that are involved in a peer-assisted data dissemination. We use *node* to refer to any device in the MRT, including servers and switches. Following the terminology of the scheduling literature, we use *makespan* to refer to the minimal time it takes to fully disseminate a file of m chunks from the source to N peers. Our problem in this paper is *to find algorithms that achieve the minimal makespan and to design practical protocols for implementing the algorithms*.

We assume that there is only one server/peer initially holding the data file, which is called the *source*. Since we are interested in P2P dissemination, we assume that a node can not multicast data to other nodes. The only way for a node to transmit data to multiple receivers is to establish multiple connections to every receiver and to transmit the data multiple times, either sequentially or concurrently. The data file is divided into multiple equally-sized chunks. We also ignore the delays of transmitting data from one node to another, which means that the completion time of transmitting data of a given size is only determined by the bandwidth.

These assumptions are abstract ones that are necessary for the theoretical analysis in Section 4. Implementation details of the proposed algorithms are given in Section 7.

4 An Optimal Solution for FatTree

4.1 A lower-bound of the makespan of the P2P data dissemination in MRT

Note that there is a bound to the speed at which the peers in FatTree can upload and download, which is the capacity of the links connecting servers and EoR switches. We denote t as the time used to transfer a chunk at the capacity of these links; the lower bound

of the makespan of transmitting m chunks to N peers is given by

$$T^* \leqslant t \times (m + \lfloor \log_2 N \rfloor) \quad (1)$$

This lower bound is derived from two observations. First, $m-1$ chunks must be transmitted by the source at least once, which takes $(m-1) \times t$ of time. Second, after transmitting $m-1$ chunks, at least one chunk needs to be disseminated to all N peers, which takes at least $\lfloor \log_2 N \rfloor$ more time.

4.2 The optimal scheduling of P2P data dissemination in FatTree

It remains unclear whether or not this lower bound can be achieved in FatTree. In Ref. [15], the authors proved that finding the minimal makespan of P2P data dissemination in general topologies is NP-hard. Fortunately, FatTree admits an optimal P2P data dissemination schedule that achieves the lower bound of Formula (1).

The enabling property is that the *oversubscription* of FatTree networks is 1. The definition of oversubscription is given below.

Definition 1 In a communication system where multiple users share a common resource, oversubscription refers to the ratio of the allocated bandwidth per user to the guaranteed bandwidth per user.

An oversubscription of 1 means that any host can communicate, both inbound and outbound, with any other hosts at the full rate of its interface. Note that the actual achieved throughput may not be as large as the interface bandwidth due to the sharing of the bandwidth at the links connecting servers and the access switches. However, the links between switches will never be the bottleneck.

In Ref. [17], the authors prove that P2P data dissemination in the *uplink-sharing model with identical upload capacity* has a polynomial-time optimal solution. The uplink-sharing model basically has the following requirements:

• The network is a bidirectional complete graph, i.e., any host can send and receive messages from any other hosts.

• The upload bandwidths of all hosts are a constant. Any host can reach this bandwidth as long as it has data to transmit.

• The download bandwidth is at least as big as the upload bandwidth.

An oversubscription of 1 actually guarantees all of these three requirements. A brief analysis verifies this fact:

• A non-zero oversubscription means that all hosts are bidirectionally connected.

• An oversubscription of 1 guarantees that all hosts can send and receive at the full rate of its interface, which is a constant.

• The inbound and outbound interface rates are the same; an oversubscripton of 1 guarantees that the download bandwidth can be as large as the upload bandwidth.

The above analysis indicates that DCNs with an oversubscription of at least 1 have an optimal P2P data dissemination schedule that achieves the lower-bound of Formula (1).

Mundinger et al.[17] also proved that the above optimal result can be achieved by using the *pairwise transmission model*, where any server can upload to and download from at most 1 other server simultaneously. Pairwise transmission makes the scheduling easier to implement in practice. They also provide such an algorithm. The algorithm tries to disseminate distinctive chunks to a maximal number of distinctive peers in minimal time.

Immediately following, any topology with an oversubscription of at least 1 has such an optimal dissemination schedule, as long as the servers have the identical links connected to the switches. We summarize the above analysis in Theorem 1.

Theorem 1 For any topology with an oversubscription of at least 1, there is a deterministic P2P data dissemination schedule that achieves the optimal makespan in Formula (1).

We use "**OPT**" to denote the optimal algorithm introduced above. We divide the algorithm into 3 phases. The pseudocode of the 3 phases is listed in Algorithms 1, 2, and 3. Note that **OPT** is based on the description in Ref. [17], but requires a non-trivial

Algorithm 1 OPT Phase I: Round 1 to n

1: $S \leftarrow$ the source
2: $m \leftarrow$ the number of chunks
3: $N \leftarrow$ the number of peers that need the data
4: $n \leftarrow \lfloor \log_2 N \rfloor$
5: Let $x \in \{1, 2, \cdots, 2^n\}$ such that $N = 2^n - 1 + x$
6: **for all** Round $i \in \{1, 2, \cdots, n\}$ **do**
7: S uploads chunk $\min(i, m)$ to a peer that has no chunk
8: $2^{i-1} - 1$ peers possess a chunk upload it to $2^{i-1} - 1$ peers that have no chunk
9: **end for**

Algorithm 2 OPT Phase II: Round $n+1$

1: **if** $x \leqslant 2^{n-1}$ **then**
2: x peers have chunk 1 upload it to x peers that have no chunk
3: $2^{n-2} - \lfloor x/2 \rfloor$ peers have chunk 1 upload to the peers that have chunk 2
4: $2^{n-2} - \lceil x/2 \rceil$ peers have chunk 1 upload it to the peers that have a chunk in $3, \cdots, m$
5: $2^{n-1} - 1$ peers have a chunk in $2, \cdots, m$ upload it to the peers that have chunk 1
6: **else**
7: 2^{n-1} peers have chunk 1 upload it to the x peers have no chunk
8: $2^{n-1} - \lfloor x/2 \rfloor$ peers have chunk 2 upload it to the peers that have only chunk 1
9: $2^{n-1} - \lceil x/2 \rceil$ peers have a chunk in $3, \cdots, m$ to the peers that have only chunk 1
10: $\lfloor x/2 \rfloor - 2^{n-2}$ peers have chunk 2 upload it to the peers that have no chunk
11: $\lceil 2/2 \rceil - 2^{n-2}$ peers have chunk in $3, \cdots, m$ to the peers that have no chunk
12: **end if**
13: S uploads chunk $\min\{n+1, m\}$ to a peer that has no chunk

Algorithm 3 OPT Phase III: Round $n+2$ to $n+m-1$

for all Round $n+1+j$ in $\{n+2, \cdots, n+m-1\}$ **do**
DEFINE:
$B_{j,j+1}$ {peers possess chunks j and $j+1$}
B_{jp} {peers possess chunk j and another in $\{j+2, \cdots, m\}$}
B_j {peers possess only chunk j}
B_{j+1} {peers possess only chunk $j+1$}
B_p {peers possess only a chunk in $\{j+2, \cdots, m\}$}
END;
$x-1$ peers in B_j upload chunk j to B_{j+1} and B_p
$B_{j,j+1}$ upload chunk $j+1$ to B_{jp}
B_{j+1} upload chunk $j+1$ to $\lfloor x/2 \rfloor$ peers of B_j
B_{jp} upload a chunk in $\{j+2, \cdots, m\}$ to $B_{j,j+1}$
B_p upload a chunk in $\{j+2, \cdots, m\}$ to $\lceil x/2 \rceil - 1$ peers of B_1
S uploads chunk $\min(n+1+j, m)$ to a peer of B_1
end for

translation of the details. **OPT** works in rounds. Each round takes t time, which is the time used to upload a single chunk, as was discussed previously.

Phase I runs from round 1 to n. The pseudocode is listed in Algorithm 1. Lines 1-5 give the notations used in all of the pseudocode. Lines 7 and 8 let the source and all peers have a chunk to upload to the same number of peers that have no chunk. In lines 7 and 8, each peer determines which peer it should upload to. All uploads are executed concurrently. Note that all peers must select different recipients, which doubles the number of peers that possess a chunk in every round. After Phase I, any chunk $i \in \{1, \cdots, \min(m,n)-1\}$ is possessed by 2^{n-i} peers. The last chunk being uploaded, $\min(m,n)$, is possessed by $2^{n-\min(m,n)+1} - 1$, and in total, $2^n - 1$ peers possess a chunk, and x peers has no chunk.

Phase II runs in round $n+1$. Lines 2-6 and 8-13 are executed depending on x. Again, all uploads are performed concurrently in each round. After Phase II, all peers possess at least one chunk.

After Phase III, all peers will have chunks from 1 to $m-2$; $2^n - x$ peers have also chunks $m-1$ and m, x peers have also chunk $m-1$, and $x-1$ peers have also chunk m. In the last round, S and $x-1$ peers that have chunk m upload chunk m to x peers that have chunk $m-1$, and the x peers having chunk $m-1$ upload to $x-1$ peers having chunk m. The dissemination is completed.

4.3 The limitations of the optimal solution

Most DCN designs do not comply with the pre-requisites of Theorem 1, so the optimal solution is not widely applicable. In Section 4.4, we will present a technique that can be used to readily apply the **OPT** algorithm in MRTs.

Additionally, the algorithm requires a central scheduler that needs to know the statuses of all peers, i.e., what chunks are received by all peers. The source, or a peer, can then select an appropriate peer to disseminate the next chunk by consulting the scheduler. Although the scheduler is straightforward to implement in theory, the actual implementation may potentially be much more complicated. There is a dilemma between scalability and complexity: A scheduler that is implemented as a single instance of service can quickly become a single point of failure and a performance bottleneck when the number of peers grows. However, implementing the scheduler as a distributed service introduces issues like *distributed consensus*, which is a non-trivial task[19,20].

4.4 A distributed approximation algorithm

In implementation, the above optimal algorithm requires a central scheduler to keep track of the received chunks of all peers. Each peer, before sending a chunk, needs to contact the server and obtain the other peer to which it should send the chunk. This requirement adds excessive delay and control overheads. In order to remedy this limitation, a distributed approximation algorithm based on the above algorithm is presented in this section. In the design, we try to avoid letting

all peers collect accurate statuses of all other peers, which will be a faithful distributed implementation of the above optimal algorithm. Instead, we present a method for nodes to estimate the status of all other peers.

By inspecting the optimal algorithm, we found that it follows a sequential delivery strategy, which is explained below:

• Sequential delivery. A peer receives the $(i+1)$-th chunk only after receiving the i-th chunk. A peer sends the $(i+1)$-th chunk only after all peers have received the i-th chunk or the delivery of i-th chunk will be completed by some other peers.

This strategy indicates that if a peer received the i-th chunk, all chunks prior it were already received. The distributed algorithm tries to stick with this rule. A peer that receives the i-th chunk, the largest-numbered chunk, and all its previous chunks is called in the i-th generation. The peers in the i-th generation should send the i-th chunk to the peers in the $(i-1)$-th generation, and request the $(i+1)$-th chunk from the peers in the $(i+1)$-th generation.

The key is to estimate the members of a given generation when sending or requesting chunks. Each chunk transmitted in the network is assigned an *age*, which is the overlay hop count that this chunk being forwarded. The age of a chunk at the source is 0. Each time the chunk is forwarded, its age is incremented by 1. For instance, when the source forwards the first chunk to peer 1, the age of the first chunk becomes 1. After peer 1 forwards the first chunk to another peer, the age of the first chunk becomes 2. The age indicates the time for which the chunk has already been transmitted in the network. According to the optimal algorithm, any chunk will have an age of at most of $\lceil \log_2 N \rceil$ or $\lfloor \log_2 N \rfloor + 1$. The age of a chunk can be used to estimate the number of peers that already received the chunk. Based on the transmission pattern of the optimal algorithm, this number grows exponentially. A chunk having an age of g will be possessed by 2^g peers.

By inspecting the age value, a peer can deduce the peers that have not already obtained the chunk. After obtaining such information, the peer picks multiple candidates to solicit transmission. We apply a hash function to mitigate collisions, which refer to the situations that multiple peers are soliciting transmission to the same peer. Choosing multiple candidates further reduces collisions. The number of the candidates, c, is a tunable parameter, which is set to 3 in this paper. The same number of hash functions, c, is also required.

5 The P2P Data Dissemination in MRTs

5.1 The single chunk case

If the data file is transmitted as a single chunk, then there is an optimal P2P dissemination schedule that can be found in polynomial time. This result is obtained based on a result of Ref. [21]. The problem is formulated to find the *minimal multicast time* of a single message from a single source to a subset of nodes on an arbitrarily connected network.

Formally, given a network modeled as a graph $G(V, E)$, a source, $S \in V$ and a set of nodes, $D \in V \backslash S$, find the minimal time, T^*, to disseminate a message from S to all nodes of D. The message can be transmitted from a node to another node (neighbor or non-neighbor) in a unit of time by using *cut-through* routing, which assumes that routing the message through intermediate nodes does not cause excessive delay. Two concurrent transmissions can happen only if they follow edge-disjoint paths. This is called the *line-network model*. The following theorem is proved in Ref. [21].

Theorem 2 for any network $G = (V, E)$, any node, $S \in V$, and any shortest path routing function on G, one can compute, in polynomial time, a minimal-time multicast schedule from S to all nodes of any set, $D \in G$, which performs in $\lfloor \log_2 |D| \rfloor + 1$ rounds.

In an MRT, ignoring the delay on the path between any two servers, any paths on the graph can transmit a single chunk in a unit of time. Thus, the MRT is equivalent to the above network model. With a further analysis, we can relate the above model with the uplink-sharing model given in Section 4. The key to note here is that since there is only 1 chunk, although the line-network model does not allow uplink sharing, the upload speed does not increase since the download speed of any peer does not increase. Since no mater how many peers are uploading to the same downloading peer, they are transferring the same data, which makes the time for a peer to download a constant determined by the bandwidth of its uplinks. There is still a gap: MRT does not allow non-blocking bandwidth between arbitrary pairs of peers. Fortunately, it has been proven that there are always pair-wise link-disjoint paths between arbitrary pairs, as long as the pairs do not share starting or ending node. That is, any peer cannot appear twice as starting or ending node of all the

transmission pairs. Note that the result in Theorem 2 is the same as the lower bound of Formula (1). This is the optimal P2P data dissemination time.

5.2 An approximation solution for the multiple chunks case

We conjecture that the optimal P2P data dissemination scheduling in MRTs is NP-hard. An observation is that the interconnection between switches made finding parallel paths between peers much more complex.

The **OPT** algorithm can be adapted to be used in the general MRT: The servers connected by the same ToR switch (or in the same rack) are grouped together. Figure 5 illustrates this idea. Let one server of the group to be the leader and directly take part in the data dissemination process, whereas all other servers form a pipeline and get data from the leader. The pipeline transmission achieves an optimal completion time since all servers in the same rack have non-blocking bandwidth. Suppose that k servers are put in a rack; it takes $t \times (m+k-2)$ to transmit m chunks from the leader to $k-1$ other servers. In all practical DCNs, the bandwidth of the uplinks is at least as large as the downlinks, so if we force only 1 server to take part in the dissemination, the effective oversubscription of the network becomes 1. We can achieve the lower-bound of Formula (1) for all leaders of all groups.

In this scheme, more time is needed to disseminate all chunks from the leader to all other servers. We only consider the worst case scenario, where there exists at least 1 rack that all its servers are in the session. Firstly, notice that servers form a pipeline; thus the completion time to transmit m chunks is the time it takes to transmit the m-th chunk. After delivering m chunks to all group leaders, it takes $t \times (k-1)$, at most, for the group leader

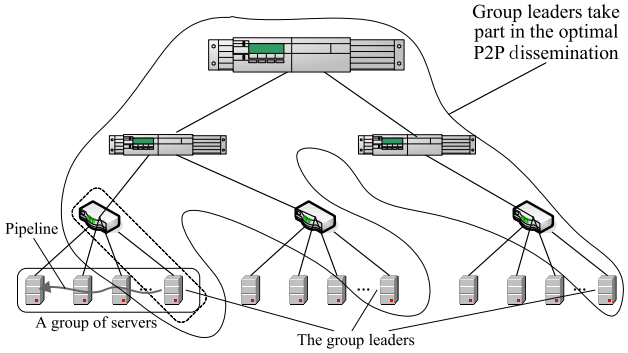


Fig. 5 Grouping together the servers in the same rack. Just the group leaders take part in the optimal P2P data dissemination. A pipeline is formed inside the group to forward chunks.

to send the m-th to all other servers in the rack. The result is in Eq. (2). Since not all servers of a rack usually take part in the dissemination, the completion time would be smaller.

$$T^* = t \times (m + \lfloor \log_2 N' \rfloor) + t \times (k-1) \quad (2)$$

5.3 A fine-grained group leader selection technique

Following the previous discussion, note that the key to applying the **OPT** algorithm is to choose group leaders so that the oversubscription for them to be interconnected is 1. Therefore, we should select the group leaders in such a way that the aggregate traffic they pose to upper-level switches does not exceed the capacity of any uplinks of the ToR switches. Assuming that the aggregate bandwidth of a switch's uplinks is evenly shared by all downlinks, we can calculate the aggregate bandwidth that can be allocated to each rack. In this case, we select multiple servers as group leaders and make sure that the oversubscription is $\leqslant 1$.

Denote the number of leaders of a rack as g_{rack}. Following the approach to derive Eq. (2), we can form g_{rack} pipelines in every rack, and each has $(k-g_{\text{rack}})/g_{\text{rack}}$ servers. The additive term in Eq. (2) becomes $t \times (k-g_{\text{rack}})/g_{\text{rack}}$. The resultant completion time is as follows:

$$T^* = t \times (m + \lfloor \log_2 N \rfloor) + t \times (k-g_{\text{rack}})/g_{\text{rack}} \quad (3)$$

We still use **OPT** to refer to this adapted version when we consider the MRT topology. Furthermore, instead of restricting the number of servers (group leaders) that take part in the **OPT** algorithm, we can lower down the bandwidth allocated to every server. Given an oversubscription value r_{oversub}, each server in a rack gets r_{oversub} of the original bandwidth, which increases the time to upload a single chunk from t to t/r_{oversub}. Substitute t with t/r_{oversub} into Formula (1), we get

$$T^* \leqslant (t/r_{\text{oversub}}) \times (m + \lfloor \log_2 N \rfloor) \quad (4)$$

6 Line Structure for the MRT Topology

We use a line structure as the basis to connect all peers. A line of peers has a few properties that make it a good candidate for high-performance, peer-assisted transmission in DCNs. Figure 6 depicts this fact. A line topology is easy to construct. It can be expanded easily by inserting or concatenating another line. Removing nodes is equally as easy by concatenating the two peering nodes of the line of nodes that need to be removed. The only necessary information is the *source* and the *end* of the line used in the operations.

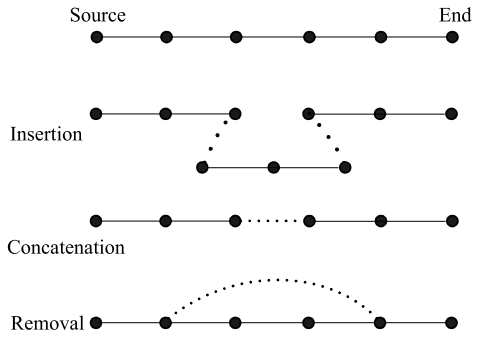


Fig. 6 The line topology and the operations that can be applied.

A line topology is also quite efficient. Following the notations used previously, delivering m chunks to N peers takes

$$T = t \times (N + m - 1) \tag{5}$$

For the conventional client/server transmission (like FTP), which is N times the one-to-one transmission, the completion time is $N \times m \times t$. There is a huge difference between them. In practice, due to a lot of influence from background transmissions, the complexity of maintaining states, and so on, the improvement will be better.

Twitter[1] has used BitTorrent for disseminating software patches to thousands of servers in their data center. They report that the resultant transmission pattern becomes a line, where most servers request and send chunks from and to only one other server. Although extremely simple, the speed is increased 75× compared to the client/server approach they used before.

We would like to eliminate the conflicts between underlying paths. Two paths conflict when they share common directed links. It is obvious that line topology has no conflicts for (multi-rooted) tree-like networks, including FatTree. Two paths conflict if they share at least one common directed link. A conflict-free line can be constructed by sorting all servers based on their positions in the MRT. Align switches from left to right. A server, A, is ranked before another server, B, if A is on the left branch of their *lowest common ancestor*.

6.1 The joining and leaving process

The OPT algorithm: Joining and leaving a line is straightforward. The tracker records all of the nodes that have joined since the beginning. Each time a new peer requests to join, the tracker assigns an appropriate predecessor node to it. The joining peer then contacts the predecessor. The predecessor records the peer as its downstream node, and lets the peer connect to its previous downstream node. This process is depicted in the insertion operation of Fig. 6. When leaving, the peer contacts its upstream and downstream peers and informs them that it will be leaving the line. The upstream peer then connects with the downstream peer to form a new line. This process is depicted in the removal operation of Fig. 6.

6.2 Line structure manipulation for scalable dissemination

The length of a line structure grows linearly with the peer count. In order to reduce delivery time, a long line is split into two shorter lines and these shorter lines are spliced into the source to make a new dissemination tree. This process is illustrated in Figs. 7 and 8. This process can be recursively performed on sub-trees.

7 Design and Implementation

7.1 The basic interaction scenario

We introduce an overview of the design by going through a basic interaction process of a complete transmission. We first define the concepts of the various components, entities, and processes of the system. We then give the details of a complete running process of a transmission.

A de-facto practice of cloud computing is virtualization, i.e., installing multiple virtual machines on any physical machine. By virtualization, virtual machines greatly outnumber physical

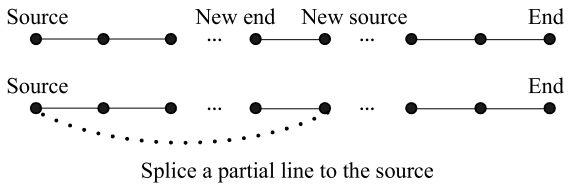


Fig. 7 Splicing a partial line to the source.

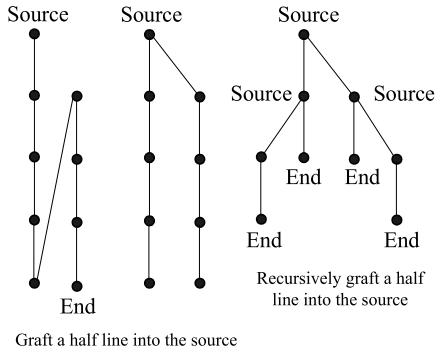


Fig. 8 Splicing a half line to the source to reduce line length.

machines. Provisioning one-to-many communication for every virtual machine is an extremely challenging task. It is doable, but will certainly be too costly to be practical, as many have stated in Ref. [20]. We install a daemon process on every physical server that is running in the hosting OS. This daemon process is called *agent*. Every virtual machine can receive data by asking its agent to join the connection and then will copy the data to the virtual machine using a specific service provided by the *hypervisor/virtualization manager*. This makes virtual machine migration simple to handle, which will be discussed in detail later.

Our system works by weaving agents into an efficient, scalable, and fault-tolerant one-to-many transmission structure. For the sake of clear presentation, we may use *peer* or *node* to refer to agents if not specified otherwise.

We assume that there is a single peer initially holding some bulk data which needs to be disseminated to multiple peers. This peer is called the *source*. As we discussed above, it may likely be the virtual machines behind the peers that are sending or receiving the data.

We borrow a few designs from BitTorrent. The source compiles a *torrent* file containing chunking information about the data. There is a *tracker* in the network, which is responsible for all of the operations that require a global view of the system. We require the tracker to run on a physical machine. For any transmission session, the source designates a tracker; other peers can request the torrent file and the information necessary for joining the network from the tracker. We require trackers to have a designated DNS name, so that we can replicate and replace them for reliability and load-balancing.

When contacting a tracker, a peer also gets a list of IP addresses of peers (including the source). Based on the technique we designed, which will be discussed in the next section, the peer finds the appropriate position in the system. It then requests and uploads data from and to its directed connected peers.

A peer may leave the system because of an unexpected crash, job completion, or it simply no longer needs the data. In either case, those peers that directly connect with them need to find new peers in order to stay connected. This is done by contacting the tracker to obtain an appropriate peer to connect with.

7.2 In-order transmission

As depicted in Eq. (5), the completion time of a line transmission is determined by the number of chunks of a data file. When the chunk number approaches infinity, the completion time converges to the time it takes to complete a single transmission. In this case, the whole line topology is abstracted as a pipeline of bit-stream. However, in practice, we do not split the data file in too many chunks since chunking introduces additional overhead, like meta-information, error-handling, connection establishment, and tear-down costs.

In a line topology, when comparing two nodes, the one with a smaller hop-count to the source is called the *up-stream* node, the other is the *down-stream* node. If the line topology is fixed, and the source is the only node that has the complete data file in the beginning, then it is clear that any down-stream nodes will not have more chunks than the up-stream nodes.

However, if the topology constantly changes, this property may be violated. Specifically, if we require each node to join an existing line based on its location in the base-line configuration, a late-coming node may join a transmission where a lot of chunks have already been transmitted. When the new node joins the line, its immediate up-stream node still goes on with the previous transmission to guarantee that the further along down-stream nodes can get new chunks. The new node can get updates from its up-stream node after it completes the transmission to the down-stream nodes.

8 Simulation Study

8.1 Simulation settings

We use FatTree and MRT as topologies for the simulation studies of the three algorithms. We use **OPT**, **Line**, and **CS** to represent the optimal algorithm for FatTree, the line-structure-based protocol, and the conventional client/server approach, respectively. Details of the simulation settings are given below.

Methodology: We wrote a customized simulator with C++[22]. Packet-level granularity is implemented. This is necessary for evaluating the overhead and the delay of the completion time caused by protocol interactions. However, transport-layer abstraction is not implemented due to the excessive complexity. We assume that all packets can be forwarded reliably, but with 90% of the available bandwidth of the path (determined by the bottleneck of the path). This assumption is based on the measurement studies in Ref. [23], which state that a TCP connection can

sustain a throughput of approximately 90% of the link capacity. We deem this is as sufficient for modeling the scaling properties of the algorithms in a comparative study. In all simulations, a data file is split into 1000 1-MB chunks. All peers are selected uniformly from all servers in the network.

Routing: We use load-balanced routing. All flows are spread evenly on all available paths. A routing path is established before a chunk transmission session begins. The path can change between different chunk transmission sessions, but not during them. In a set of simulations, background traffic is injected. The parameters are modeled according to Ref. [4] with scaled-down traffic, which corresponds to the fact that the application we consider is usually performed when the network load is low. All packets are 1500 bytes.

Scheduling: We assume that the central scheduler can smoothly handle all of the queries from any number of peers. We acknowledge that this may not hold in practice, but the system in Ref. [1] demonstrates that a modified BitTorrent tracker can handle thousands of servers, which makes this an acceptable assumption for a comparison study.

Topology: We use a FatTree composed of 48-port 1 Gbps switches, which contains 27 648 servers. For MRT, 64-port 1 Gbps switches are used for ToR switches, which support 20 480 servers. We vary the oversubscription in $\{1/2, 1/4, 1/8\}$. The connections between the ToR and EoR, and the EoR and Core switches are made accordingly.

8.2 Completion time

We give, in Figs. 9-11, the completion time of three algorithms in the FatTree network that was specified in the last section. OPT, Line, and CS refer to the **OPT** algorithm, line structure, and the client/server approach

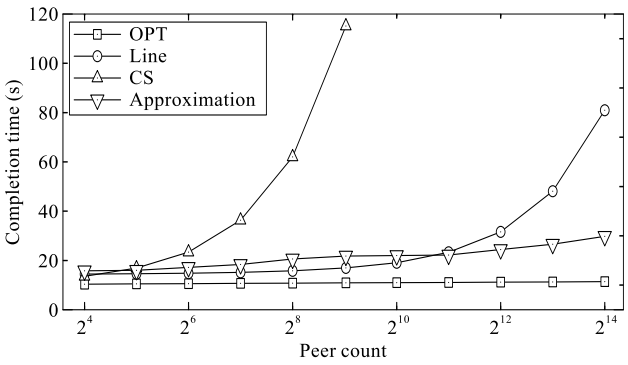


Fig. 9 The completion time of three algorithms in a FatTree network with 100 chunks and a varying numbers of peers.

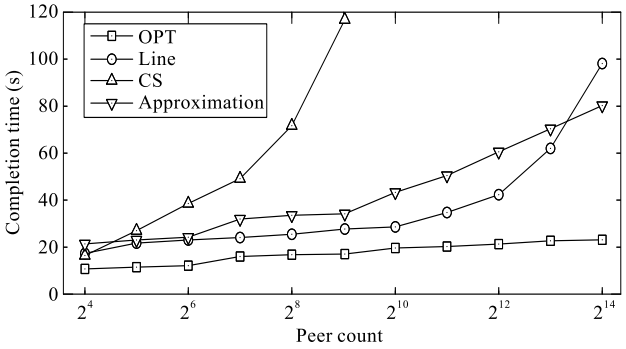


Fig. 10 The completion time of three algorithms in a FatTree network with background traffic.

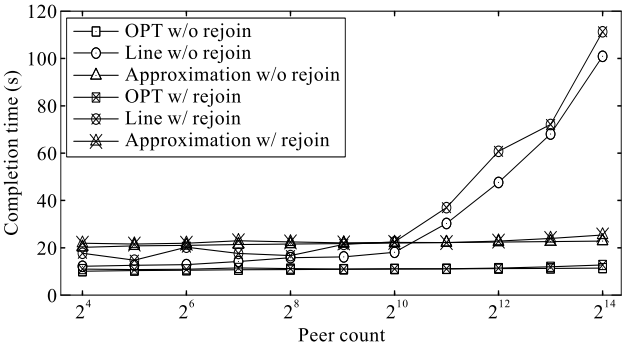


Fig. 11 The completion time of three algorithms in a FatTree network with random server failure.

respectively.

Figure 9 shows the completion time in the FatTree network. The network has no background traffic. The results demonstrate the analytical results. A chunk can be uploaded in approximately 0.01 ms. Uploading 1000 chunks takes 10 s, which accounts for the majority of the upload time for **OPT**. **Approximation** performs closely to **OPT**. Note that **OPT** significantly outperforms **Line** only when there are more than 2^{10} peers. Part of the results for **CS** are not included.

Figure 10 shows the completion time in the FatTree network with background traffic. 10% of the upload/download cpacities of the servers are allocated for background traffic. All paths connecting the server pairs go through the Core switches. 1% of the server pairs are assigned with a heavy flow with a bandwidth of 80% of the servers' upload/download capcities. **Approximation** has a close performance and scaling behavior to the **OPT**. **OPT** and **Line** are more sensitive to the insufficient link bandwidths than **CS**.

Figure 11 shows the completion time in the FatTree network with random server failure. Two types of recovery methods are used: (1) without rejoin, and (2) with rejoin. 10% of the peers will fail after the data

dissemination begins. At each round, t time or t/r time for MRTs, a server is picked uniformly to fail and leave the network. In the case of without rejoin, the server will just leave the network; whereas in rejoin, the server that failed in the last round rejoined the dissemination but lost all previously received data. In both cases, after 10% of the servers are selected we stop picking. Without rejoin, all algorithms perform better since less transmissions are actually needed.

Figures 12-14 show the completion time of the

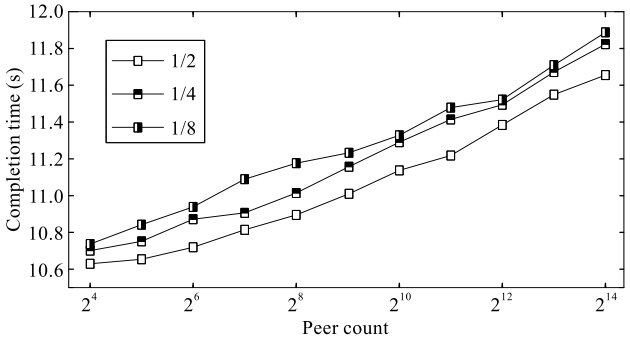


Fig. 12 The completion time of the OPT algorithm in an MRT network with varying oversubscriptions.

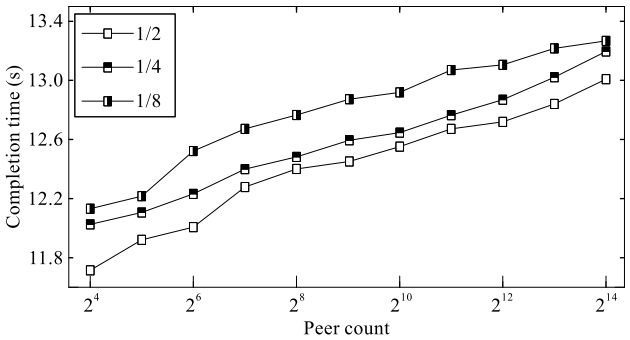


Fig. 13 The completion time of the OPT algorithm in an MRT network with background traffic.

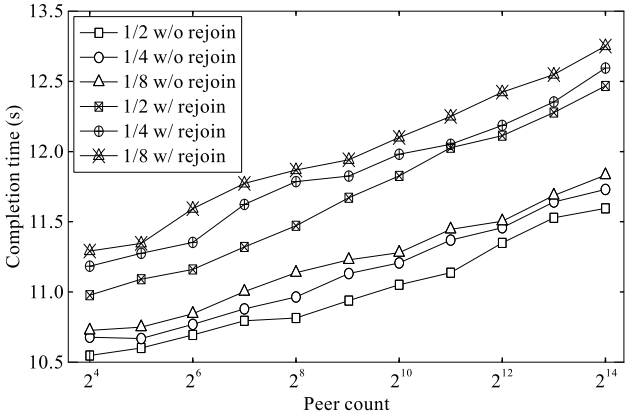


Fig. 14 The completion time of the OPT algorithm in an MRT network with random server failure.

OPT algorithm in the MRT network with different oversubscriptions. Since the **Line** and **CS** behave the same in FatTree and MRT, we omit their results. Our adapted **OPT** algorithm works very well in MRT, which only slightly increases the completion time compared with the FatTree network.

8.3 Control overhead

We measure the number of control packets sent for each algorithm. The results are shown in Fig. 15. The **OPT** algorithm uses much more control packets since all peers need to communicate with the scheduler to get the peer for the next round. The **Line** and **CS** algorithms only need to inform the peers once before the transmission. This large overhead will certainly cause issues in real-world applications. It is therefore necessary to explore distributed scheduling in future research.

9 Conclusions

In this paper, we study peer-assisted data dissemination in DCNs. Our work is motivated by the inability of the new DCN architecture and the network-layer multicast, and is inspired by an existing engineering practice of using P2P technology in data center software update. We restrict our attention to MRT topologies and FatTree—a special case of MRT. Based on an existing theoretical work of P2P scheduling, we prove that an optimal solution exists for FatTree, and the solution achieves a parametrized additive factor of the lower bound in general MRTs. Inspired by engineering practices, we propose a line-structure-based pipelining P2P transmission algorithm. We also analyze the methods and techniques for implementing these algorithms. The performances of the proposed algorithms are evaluated using packet-level simulations. The results demonstrate

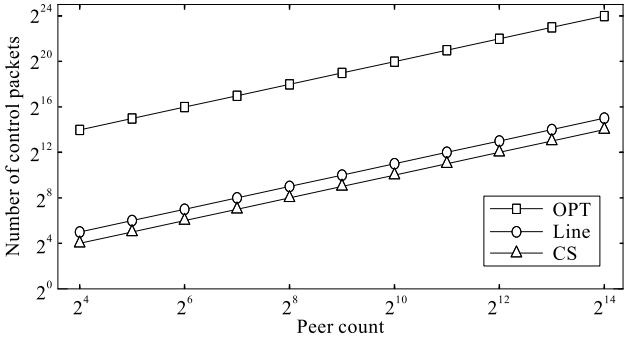


Fig. 15 The overhead of the three algorithms.

the viability of the peer-assisted data dissemination approach.

Acknowledgements

This research was supported in part by the Natural Science Foundation of USA (Nos. ECCS 1128209, CNS 10655444, CCF 1028167, CNS 0948184, and CCF 0830289).

References

[1] Murder: Fast datacenter code deploys using bittorrent, http://engineering.twitter.com/2010/07/murder-fast-data-center-codedeploys.html, 2010.

[2] J. Dean and S. Ghemawat, Mapreduce: Simplified data processing on large clusters, in *Proc. of OSDI '04*, Berkeley, CA, USA, 2004.

[3] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, and Y. Tock, Dr. multicast: Rx for data center communication scalability, in *Proc. Of LADIS '08*, New York, NY, USA, 2008.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang, Understanding data center traffic characteristics, *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 92-99, 2010.

[5] S. James and P. Crowley, Fast content distribution on datacenter networks, in *Proc. of ANCS'11*, 2011.

[6] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, Dcell: A scalable and fault-tolerant network structure for data centers, in *Proc. of SIGCOMM '08*, New York, NY, USA, 2008.

[7] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, Bcube: A high performance, server-centric network architecture for modular data centers, *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 63-74, 2009.

[8] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, Symbiotic routing in future data centers, *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 51-62, 2010.

[9] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu, Generic and automatic address configuration for data center networks, *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 39-50, 2010.

[10] M. Al-Fares, A. Loukissas, and A. Vahdat, A scalable, commodity data center network architecture, in *Proc. of SIGCOMM '08*, New York, NY, USA, 2008.

[11] "BitTorrent," http://www.bittorrent.com/, 2013.

[12] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, Portland: A scalable fault-tolerant layer 2 data center network fabric, *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 39-50, 2009.

[13] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, Vl2: A scalable and flexible data center network, *Commun. ACM*, vol. 54, pp. 95-104, 2011.

[14] C. E. Leiserson, Fat-trees: Universal networks for hardware-efficient supercomputing, *IEEE Trans. Comput.*, vol. 34, pp. 892-901, 1985.

[15] S. T. O'Neil, A. Chaudhary, D. Z. Chen, and H. Wang, The topology aware file distribution problem, in *Proc. of COCOON'11*, Springer-Verlag, 2011.

[16] C. Killian, M. Vrable, A. C. Snoeren, A. Vahdat, and J. Pasquale, Brief announcement: The overlay network content distribution problem, in *Proc. of PODC '05*, New York, NY, USA, 2005.

[17] J. Mundinger, R. Weber, and G. Weiss, Optimal scheduling of peer-topeer file dissemination, *J. of Scheduling*, vol. 11, pp. 105-120, 2008.

[18] Cisco data center infrastructure 2.5 design guide, http://www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration 09186a008073377d.pdf, 2013.

[19] M. Burrows, The chubby lock service for loosely-coupled distributed systems, in *Proc. of OSDI '06*, Berkeley, CA, USA, 2006.

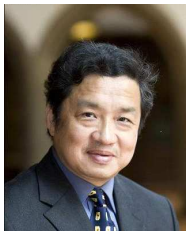
[20] T. D. Chandra, R. Griesemer, and J. Redstone, Paxos made live: An engineering perspective, in *Proc. of PODC '07*, New York, NY, USA, 2007.

[21] J. Cohen, P. Fraigniaud, J.-C. Konig, and A. Raspaud, Broadcasting and multicasting in cut-through routed networks, in *Proc. of the 11th International Parallel Processing Symposium*, 1997.

[22] B. Stroustrup, *The C++ Programming Language*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., 2000.

[23] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, Measurement and analysis of tcp throughput collapse in cluster-based storage systems, in *Proc. of FAST '08*, Berkeley, CA, USA, 2008.

Yaxiong Zhao is a software engineer at Google's Platforms Networking group. He was with Amazon Web Services before joining Google, working on Amazon Kinesis, the first hosted stream data processing service in the world. He received his PhD degree from Temple University, USA. His research interests are in distributed systems, cloud computing, and wireless networks. He was the founding chair of the First International Workshop on Resource Management of Cloud Computing, USA. His current focus is on software defined networking and data center networking for cloud computing.



Jie Wu is the chair and a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University, USA. Prior to joining Temple University, he was a program director at the National Science Foundation and a distinguished professor at Florida Atlantic University. He received his PhD degree from Florida Atlantic University in 1989. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Computers, IEEE Transactions on Service Computing, and Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair/chair for IEEE MASS 2006 and IEEE IPDPS 2008 and program co-chair for IEEE INFOCOM 2011. Currently, he is serving as general chair for IEEE ICDCS 2013 and ACM MobiHoc 2014, and program chair for CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Cong Liu is currently an assistant professor at Sun Yat-sen (Zhongshan) University, China. He received his PhD degree from Florida Atlantic University, USA in 2009. Before that he received his MS degree in computer software & theory from Sun Yat-sen (Zhongshan) University, China, in 2005. He received his BS degree in micro-electronics from South China University of Technology in 2002. His main research interests include routing in Delay Tolerant Networks (DTNs) routing, geometric routing in Mobile Ad hoc Networks (MANETs), deep packet inspection, transaction processing, and rough set theory.