BDS: A Centralized Near-Optimal Overlay Network for Inter-Datacenter Data Replication

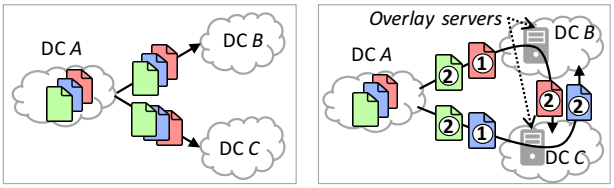
Yuchao Zhang[1], Junchen Jiang[2], Ke Xu[3], Xiaohui Nie[3],
Martin J. Reed[4], Haiyang Wang[5], Guang Yao[6], Miao Zhang[6], Kai Chen[1],
[1] HKUST [2] The University of Chicago [3] Tsinghua University
[4] University of Essex [5] University of Minnesota at Duluth [6] Baidu
zyc@ust.hk, junchenj@uchicago.edu, xuke@tsinghua.edu.cn,
nxh15@mails.tsinghua.edu.cn, mjreed@essex.ac.uk, haiyang@d.umn.edu,
{yaoguang,zhangmiao02}@baidu.com, kaichen@cse.ust.hk

Abstract

Many important cloud services require replicating massive data from one datacenter (DC) to many DCs. While the performance of pair-wise inter-DC data transfers has been greatly improved, existing solutions are insufficient to optimize bulk-data multicast, as they fail to harness capability of servers to store-and-forward data, as well as the rich inter-DC overlay paths that exist in geo-distributed DCs. To harness these opportunities, we present *BDS*, an application-level multicast overlay network, which sends inter-DC multicast traffic via many inter-DC overlay paths in a near-optimal manner. At the core of BDS is a *fully centralized* architecture, allowing a central controller to maintain an up-to-date global view of data delivery status of intermediate servers, in order to fully utilize the available overlay paths. To quickly react to network dynamics and workload churns, BDS speeds up the control algorithm by decoupling it into selection of overlay paths and scheduling of data transfers, each can be optimized efficiently. This enables BDS to update overlay routing decisions in near real-time (e.g., every other second) at a scale of multicasting hundreds of TB data over tens of thousands of overlay paths. A pilot deployment in one of the largest online service providers shows that BDS can achieve 3-5× speedup over the provider's existing system and several well-known overlay routing baselines.

1. Introduction

For large-scale online services, such as Google, Facebook, and Baidu, an important data communication pattern is *inter-DC multicast* of bulk data–replicating massive amounts of data (e.g., user logs, web search indexes, photo sharing, blog posts) from one DC to multiple DCs in geo-distributed locations. Our study on the workload of Baidu shows that inter-DC multicast already amounts to 91% of inter-DC traffic (§2), which corroborates the traffic pattern of other large-scale online service providers [26, 55]. As more DCs are deployed globally and data sizes exploding, inter-DC traffic will continue to grow rapidly.



(a) Direct replication over pair-wise inter-DC WANs

(b) Replication leveraging overlay paths

Figure 1: **A simple network topology illustrating how overlay paths reduce inter-DC multicast completion time. Assume that the WAN link between any two DCs is 1GB/s, and that A wants to send 3GB data to B and C. Sending data from A to B and C separately takes 3 seconds (a), but using overlay paths $A \rightarrow B \rightarrow C$ and $A \rightarrow C \rightarrow B$ simultaneously takes only 2 seconds (b). The circled numbers show the order for each data piece is sent.**

While there have been tremendous efforts towards better inter-DC network performance (e.g., [21, 23, 26, 44, 50]), the focus has been improving the performance of the wide area network (WAN) path between each pair of DCs. These WAN-centric approaches, however, are imcomplete, as they fail to leverage the rich application-level overlay paths across geo-distributed DCs, as well as the capability of servers to store-and-forward data. As illustrated in Figure 1, the performance of inter-DC multicast could be substantially improved by sending data in parallel via multiple overlay servers acting as intermediate points to circumvent slow WAN paths and performance bottlenecks in DC networks. It is important to notice that these overlay paths should be *bottleneck-disjoint*; that is, they do not share common

bottleneck links (e.g., $A \rightarrow B \rightarrow C$ and $A \rightarrow C \rightarrow B$ in Figure 1). and that such bottleneck-disjoint overlay paths are available in abundance in geo-distributed DCs.

This paper presents *BDS*, an *application-level multicast overlay network*, which splits data into fine-grained units, and sends them in parallel via bottleneck-disjoint overlay paths. These paths are selected dynamically in response to changes in network conditions and the data delivery status of each server. Note that BDS selects application-level overlay paths, and is therefore complementary to network-layer optimization of WAN performance. While application-level multicast overlays have been applied in other contexts (e.g., [9, 28, 33, 48]), building one for inter-DC multicast traffic poses two challenges. First, as each DC has tens of thousands of servers, the resulting sheer amount of possible overlay paths makes it unwieldy to update overlay routing decisions at scale in real time. Prior work either relies on local reactive decisions by individual servers [22, 25, 41], which leads to suboptimal decisions for lack of global information, or restricts itself to strictly structured (e.g., layered) topologies [36], which fails to leverage all possible overlay pathes. Second, even a small increase in the delay of latency-sensitive traffic can cause significant revenue loss, so the bandwidth usage of inter-DC bulk-data multicasts must be tightly controlled to avoid negative impact on other latency-sensitive traffic [1] .

To address these challenges, BDS fully *centralizes* the scheduling and routing of inter-DC multicast. Contrary to the intuition that servers must retain certain local decision-making to achieve desirable scalability and responsiveness to network dynamics, BDS's centralized design is built on two empirical observations (§3). (1) While it is hard to make centralized decisions in real time, most multicast data transfers last for at least tens of seconds, and thus can tolerate slightly delayed decisions in exchange for near-optimal routing and scheduling based on a global view. (b) Centrally coordinated sending rate allocation is amenable to minimizing the interference between inter-DC multicast traffic and latency-sensitive traffic.

The key to making BDS practical is how to update the overlay network in near real-time (within a few seconds) in response to performance churns and dynamic arrivals of requests. BDS achieves this by *decoupling* its centralized control into two optimization problems, scheduling of data transfers, and overlay routing of individual data transfers. Such decoupling attains provable optimality, and at the same time, allows BDS to update overlay network routing and scheduling in a fraction of second, four orders of magnitude faster than solving routing and scheduling jointly with the workload of a large online service provider (e.g., sending 10^5s of data blocks simultaneously along 10^4s of disjoint overlay paths).

We have implemented a prototype of BDS and integrated it in Baidu, one of the largest search service providers. We deployed BDS in 10 DCs and ran a pilot study on 500 TB data transfer for 7 days (about 71 TB per day). Our real-world experiments show that BDS achieves 3-5× speedup over Baidu's existing solution, and it can eliminate the incidents of excessive bandwidth consumption by bulk-data transfers. Using trace-driven simulation and microbanchmarking, we also show that BDS outperforms techniques widely used in CDNs, that BDS can handle the workload of Baidu's inter-DC multicast traffic with one general-purpose server, and that BDS can handle various failure scenarios.

Our contributions are summarized as following:

- Characterizing Baidu's workload of inter-DC bulk-data multicast to motivate the need of application-level multicast overlay networks (§2).
- Presenting BDS, an application-level multicast overlay network that achieves near-optimal flow completion time by a centralized control architecture (§3,4,5).
- Demonstrating the practical benefits of BDS by a real-world pilot deployment in a large online service provider (§6).

2. A case for application-level inter-DC multicast overlay

To provide a case for an application-level multicast overlay network, we first characterize the inter-DC multicast workload in Baidu, a global-scale online service provider (§2.1). We then show the potential improvement of multicast performance by leveraging disjoint application-level overlay paths available in geo-distributed DCs (§2.2). Finally, we examine Baidu's current solution of inter-DC multicast, and draw lessons from real-world incidents to inform the design of BDS (§2.3). The findings are based on a dataset collected from Baidu's inter-DC WANs over a duration of 7 days. The dataset comprises of about 1265 multicast transfers among over 30 geo-distributed DCs.

2.1 Characterizing inter-DC multicast workload

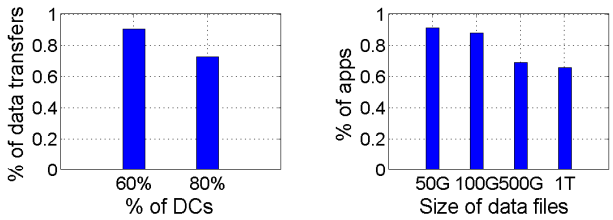
Share of inter-DC multicast traffic: Table 1 shows inter-DC multicast (replicating data from one DC to multiple DCs) as a fraction of inter-DC traffic. We see that inter-DC multicast dominates Baidu's overall inter-DC traffic (91.13%), as well as the traffic of individual application types (89.2 to 99.1%). *The fact that inter-DC multicast traffic amounts to a dominating share of inter-DC traffic*

[1] Despite priority-based queuing at the ingress/egress points of each DC, the inter-DC traffic can still negatively impact latency-sensitive traffic performance, as they share the WAN links not managed by Baidu. This problem is so profound that some CDNs have built dedicated infrastructure as their inter-DC WANs, but in many cases, such as in Baidu's, DCs are connected through public ISPs

[2] The overall multicast traffic share is estimated by that of the traffic goes through one randomly sampled DC, because we do not have access to information of all inter-DC traffic. But this number is consistent with what we observe on other DCs.

Type of application	% of multicast traffic
All applications	91.13% [2]
Blog posts	91.0%
Search indexing	89.2%
Offline file sharing	98.18%
Forum posts	98.08%
Other DB sync-ups	99.1%

Table 1: **Inter-DC multicast (replicating data from one DC to many DCs) dominantes Baidu's inter-DC traffic.**



(a) Percent of multicast transfers destined to percent of DCs.

(b) Percent of multicast transfers larger than certain threshold.

Figure 2: **Inter-DC multicasts (a) are destined to a significant fraction of DCs and (b) have large data sizes.**

highlights the importance of optimizing the performance of inter-DC multicast.

Where are inter-DC multicasts destined? Next, we want to know if these transfers are destined to a large fraction (or just a handful) of DCs, and whether they have common destinations. Figure 2a sketches the distribution of the percentage of Baidu's DCs to which multicast transfers are destined. We can see that 90% multicasts are destined to over 60% DCs, and 70% are destined to over 80% DCs. Moreover, we found a great diversity in source DCs and the sets of destination DCs (not shown here). These observations suggest that *it is untenable to pre-configure the data transfer for each possible multicast request; instead, we need a scheme that automatically routes and schedules data transfers for any given source and destination DCs.*

Sizes of inter-DC multicast transfers: Finally, Figure 2b outlines the distribution of data size of inter-DC multicast. We see that over 60% multicast data files are over 1TB (and 90% are over 50GB). Given that the total WAN bandwidth assigned to each multicast is on the order of several Gb/s, this suggests that the transfers are not transient but typically lasting for a duration on the timescales of at least tens of seconds instead. Therefore, *any scheme that optimizes multicast traffic must adapt to network dynamics.* On the flip side, such temporal persistence also indicates that *multicast traffic might tolerate certain delay when adapting to changes in network conditions*, which inspires BDS's centralized design (§3).

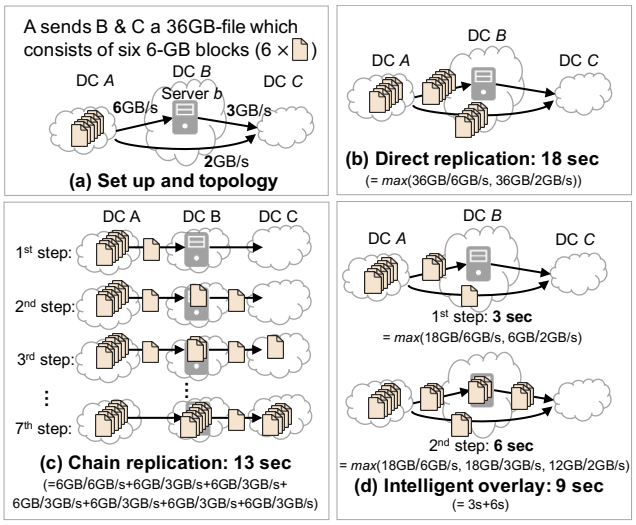


Figure 3: **An illustrative example comparing the performance of an intelligent application-level overlay (d) with that of baselines: naive application-level overlay (c) and no overlay (b).**

These observations, as we'll see later, together motivate the need for a systematic approach to optimizing inter-DC multicast.

2.2 Potentials of inter-DC application-level overlay

It is known that multicast can benefit from application-level overlays [14]. Here we show that inter-DC multicast completion time (defined as the duration of data transfers until each destination DC has a full copy of the data) can be greatly improved by an application-level overlay network. [3]

The basic idea of application-level overlay networks is to spread traffic along *bottleneck-disjoint* overlay paths. In the context of inter-DC overlays, two overlay paths either traverse different sequences of DCs (*Type I*), or traverse the same DCs but different sequences of servers (*Type II*), or some combinations of the two types. Next, we use examples to show these bottleneck-disjoint overlay paths can be used to improve inter-DC multicast performance.

Examples of bottleneck-disjoint overlay paths: Recall that Figure 1 illustrates the benefits of using Type I bottleneck-disjoint overlay paths (i.e., going through different sequences of DCs), $A \rightarrow B \rightarrow C$ and $A \rightarrow C \rightarrow B$. Figure 3 shows an example of Type II bottleneck-disjoint overlay paths (traversing the same sequence of DCs but different servers). Suppose we need to replicate 36GB data from DC A to B and C via two bottleneck-disjoint paths: (1) $A \rightarrow C$: from A through B to C using IP-layer WAN routing with 2GB/s capacity, or (2) $A \rightarrow b \rightarrow C$: from A to a server b in B with 6GB/s capacity and b to C with 3GB/s capacity. The data is split into 6 6GB-blocks. We consider three strategies. (1) *Direct replication*: if A sends data directly to B and C via

[3] An application-level overlay is complementary to prior work that focused on optimizing WAN.

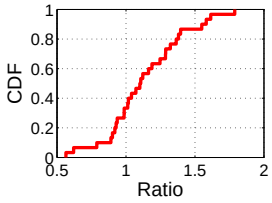


Figure 4: **Even among inter-DC paths of the same DC sequence, there is significant performance variance. The figure shows the ratio between the available bandwidth from A to C through WAN ($BW_{A\to C}$) and that from A to C through b ($BW_{A\to b\to C}$), across all possible b.**

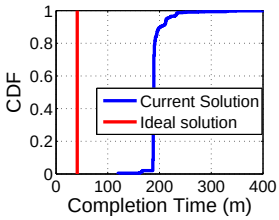


Figure 5: **The CDF of the actual flow completion time at different servers in the destination DCs, compared with that of the ideal solution.**

WAN paths (Figure 3(b)), the completion time is 18 seconds. (2) *Simple chain replication*: a naive use of application-level overlay paths is to send blocks through server b acting as a store-and-relay point (Figure 3(c)), and the completion time is 13 seconds (27% less than without overlay). (3) *Intelligent multicast overlay*: Figure 3(d) further improves the performance by selectively sending blocks along the two paths simultaneously, which completes in 9 seconds (30% less than chain replication, and 50% less than direct replication).

Bottleneck-disjoint overlay paths in the wild: It is hard to identify all bottleneck-disjoint overlay paths, since the dataset only has end-to-end throughput between servers. Instead, we show empirically study one example of bottleneck-disjoint overlay path in the wild, which consists of two overlay paths $A\to b\to C$ and $A\to C$, where the WAN routing from DC A to DC C goes through DC B, and b is a server in B. These two paths are topologically identical to Figure 3. Note if two overlay paths have different end-to-end bandwidth (i.e., $\frac{BW_{A\to C}}{BW_{A\to b\to C}} \neq 1$) at the same time, they are bottleneck-disjoint. Figure 4 shows the distribution of $\frac{BW_{A\to C}}{BW_{A\to b\to C}}$ among all possible values of A, b, and C in the dataset. We can see that $\frac{BW_{A\to C}}{BW_{A\to b\to C}}$ is below 0.8 or over 1.3 for about 30% overlay pairs. These pairs are widely available, and we can create a new bottleneck-disjoint path by using any server b as an overlay point.

2.3 Limitations of existing solutions

However, realizing the potential improvement of application-level overlay network foe inter-DC multicast is easier said than done. At a first glance, we may simply borrow existing techniques from multicast overlay networks in other contexts. But the operational experience of Baidu indicates two limitations of this approach.

Existing solutions of Baidu: To meet the need of rapid growth of inter-DC data replication, Baidu has deployed an application-level overlay network a few years ago. Despite years of refinement, the overlay network is still based on a receiver-driven decentralized overlay multicast protocol, which resembles what was used in other overlay networks (such as CDNs and overlay-based live video streaming [9, 47, 54]). When multiple DCs send the request of a data file to the source DC, the requested data would flow back through multiple stages of intermediate servers, where the selection of senders in each stage is driven by the receivers of the next stage in a decentralized fashion.

Limitation 1: Inefficient local adaptation. The existing decentralized protocol lacks the global view and thus may stuck with suboptimal scheduling and routing. To show this, we sent a 30GB file from one DC to two destination DCs in Baidu's network. Each DC had 640 servers, each with 20Mbps upload and download bandwidth. This 30GB file was evenly stored in all these 640 servers. Ideally, if the servers select the best source for all blocks, the ideal completion time will be $\frac{30\times 1024}{640\times 20Mbps\times 60s/min} = 41$ minutes. But as shown in Figure 5, servers in the destination DCs on average received data in 195 minutes (4.75 times of the optimal completion time), and 5% of them even took over 250 minutes. The key reason is that individual servers only see a subset of possible data sources (i.e., servers who have already downloaded part of a file), and thus cannot leverage all available overlay path to maximize the throughput. Such suboptimal performance could occur even if the overlay network is only partially decentralized where each server does have a global view (e.g., [22]), since each server still makes local adaptations which creates potential hotspots and congestion on overlay paths.

Limitation 2: Interference with latency-sensitive traffic. The existing multicast overlay network shares the same inter-DC WAN with latency-sensitive traffic. Despite using standard QoS techniques and giving the lowest priority to bulk data transfers, we still see negative impacts on latency-sensitive traffic by bursty arrivals of bulk-data multicast requests. We monitored the bandwidth utilization of an inter-DC link in two days during which there was a bulk data transfer at 11:00 on the 2nd day lasting for 6 hours. Figure 6 shows the total utilization of an inter-DC link over time. The blue line denotes the outgoing (outbound) bandwidth and the green line denotes the ingoing (inbound) bandwidth. From this figure, we can see that the bulk data transfer caused excessive link utilization (i.e., exceeding the safety threshold of 80%) due to lack of global coordination. As a result, the latency-sensitive online traffic experienced over 30× longer delay.

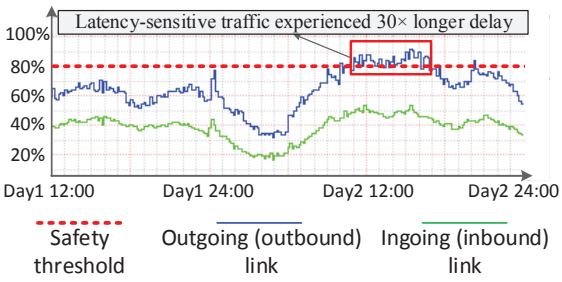


Figure 6: **The utilization of the inter-DC link in two days. Inter-DC bulk data transfer on the 2nd day caused severe interference on latency-sensitive traffic.**

2.4 Key observations

The key observations from this section are:

- *Inter-DC multicasts* amount to a great fraction of inter-DC traffic, have a great variability in source-destination, and typically last for at least tens of seconds.
- *Bottleneck-disjoint overlay paths* are widely available between geo-distributed DCs.
- Existing solutions that use local adaptation can have *suboptimal performance* and *negative impact on online traffic*, due to lack of a global view and coordination.

3. Overview of BDS

To optimize inter-DC multicasts while minimizing interference with latency-sensitive traffic, we present *BDS*, a near-optimal application-level overlay network for inter-DC multicasts. Before presenting BDS in details, we first highlight the intuitions behind its key design choice, and the challenges to make it practical.

Centralized control: Conventional wisdom on wide-area overlay networks has relied, to some extent, on *local* adaptation of individual nodes (or relay servers) to achieve desirable scalability and responsiveness to network dynamics (e.g., [9, 22, 41]), despit of the suboptimal performance due to lack of global view and coordination. Recent work (e.g., [34]), however, shows the feasibility of combining local adaptation with a centralized logic operating on coarse timescales. In contrast, BDS takes an explicit stance that fully centralized control is practical and can achieve near-optimal performance in the setting of inter-DC multicasts. At a high level, BDS uses a centralized controller that periodically pulls information from all servers, updates the decisions regarding overlay routing, and pushes them to agents running locally on servers (Figure 7). Note that when the controller fails or is unreachable, BDS will still fall back to a decentralized control scheme to ensure graceful performance degradation to local adaptation.

BDS's centralized design is driven by several empirical observations:

1. *Large decision space:* The sheer numbers of inter-DC overlay paths (which grow exponentially with more servers acting as overlay nodes) make it difficult for servers to explore all available overlay paths based only on local measurements. Instead, we could significantly improve overlay multicast performance by maintaining a global view of data delivery status of all servers and dynamically balance the availability of various data blocks, which is critical to achieving near-optimal performance (§4.3).
2. *Large data size:* Unlike latency-sensitive traffic (e.g., online partition-aggregation) which lasts on timescales of several to 10s of milliseconds, inter-DC multicasts are large in volume and last on much coarser timescales. Thus, being highly responsive to transient network dynamics is a necessary concern. Therefore, BDS can tolerate a short delay (of a few seconds) to query a centralized controller, which maintains a global view of data delivery and could make optimal decisions.
3. *Strict traffic isolation:* As observed in §2.3, it is vital that inter-DC multicasts avoid hotspots and excessive bandwidth usage that can increase the latency of delay-sensitive traffic, but it is difficult to prevent such situations without any coordination across overlay servers. In contrast, it is practical to determine the bandwidth allocation and periodically update it to all servers in a centralized fashion (§5).
4. *Lower engineering complexity:* Conceptually, the centralized architecture moves the control complexity to the centralized controller, making BDS amenable to a simpler implementation, where the control logic running locally in each server can be stateless and triggered only on arrivals of new data units or control messages.

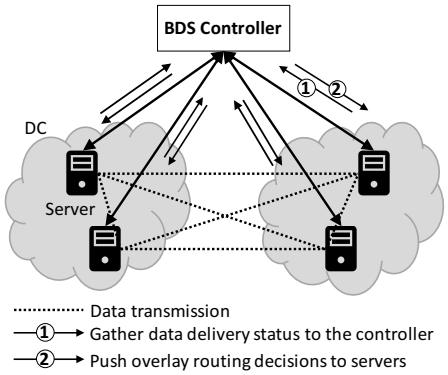


Figure 7: **The centralized design of BDS.**

The key to realizing centralized control: In essence, BDS trades updating decisions on slightly coarse timescales for the potential of making optimal decisions in a centralized fashion. Thus, the key to striking such a favorable balance is a near-optimal and efficient overlay routing algorithm that can be updated on near-realtime timescales. At a first glance, this is indeed intractable: the centralized overlay routing algorithm must pick the next hops from 10^4s of servers for 10^5s of blocks, a scale that could grow exponentially when we consider all possible overlay paths that go through these servers and finer-grained block partitions. Using standard

Variables	Meaning
$\mathbb{A}$	Set of all source and destination pairs
$\mathbb{B}$	Set of blocks of all tasks
$\mathbb{P}$	Set of all possible paths in $\mathbb{A}$
$\mathbb{B}_n$	Set of blocks on server n
$\overrightarrow{o_{\mathbb{B}}}$	Transmission sequence of $\mathbb{B}$
$B_{i,j}$	Block i in Task j
$R_{up}(n)/R_{down}(n)$	Upload/download rate limit of server n
$p_{(s,d)}$	A path between a source and destination pair
$l(p_{(s,d)})$	A link in path $p_{(s,d)}$
$c(l)$	Capacity of link l
$s_{B_{i,j}}$	Data source of $B_{i,j}$
$f_{B_{i,j},p}$	Transmission rate of $B_{i,j}$ on path p_λ
$I_{B_{i,j},p}$	0 or 1: whether p is selected for $B_{i,j}$

Table 2: **Notations used in BDS's decision-making logic.**

routing formulation and solver, it could be completely unrealistic to make optimal solutions by exploring such a large decision space §6.3.4. The next section will present how BDS addresses this challenge.

4. Near-Optimal and Efficient Decision-Making Logic

At the core of BDS is a centralized decision-making algorithm that periodically updates overlay routing decisions at scale in near real-time at the timescales of several seconds. BDS strikes a favorable tradeoff between solution optimality and near real-time updates by *decoupling* the control logic into two steps (§4.2): overlay scheduling, i.e., which data blocks to be sent (§4.3), and routing, i.e., which paths to send each data block (§4.4), each of which can be solved efficiently and near-optimally with proved guarantees (§4.3, 4.4).

4.1 Basic formulation

Under fixed network capacity: We start with the basic formulation of the overlay control under the assumption of fixed network capacity and fixed demand of multicast requests. We will later extend it to dynamic network performance and traffic demands. Table 2 summarizes the key variables and parameters. The problem of multicast overlay routing can be formulated as follows:

(1) *Input.* Each multicast transmission, referred to as a *task* T, is defined by a source DC, a set of destination DCs, and a data file chopped into a list of fixed-sized data *blocks*[4]. The ith block of the jth task is denoted as $B_{i,j}$. Besides traffic demand, the inputs also include link capacity $c(l_{u,v})$, and server upload (and download) rate limit $R_{up}(n)$ (and $R_{down}(n)$).

(2) *Output.* A four-tuple for each server n. $\langle \overrightarrow{o_{\mathbb{B}_n}}, s^*_{\mathbb{B}_n}, p^*_{(s^*_{\mathbb{B}_n},d)}, f^*_{\mathbb{B}_n,p^*_{(s^*_{\mathbb{B}_n},d)}} \rangle$, which denotes the block transmission sequence, the optimal source, overlay path for each block, and the optimal allocated bandwidth on this path, respectively.

(3) *Constraints.* The formulation is subject to constraints similar to those of maximum concurrent flow (MCF) problem [18].

- The allocated bandwidth on path $p_{(s,d)}$ should be the minimum of three parameters: the capacity of links that are in the path $c(l(p_{(s,d)}))$, source server upload rate $R_{up}(s)$, and destination server download rate $R_{down}(d)$.

$$f_{B_{i,j},p_{(s,d)}} \leq min\{c(l(p_{(s,d)})), R_{up}(s), R_{down}(d)\}, \quad \text{for } \forall B_{i,j}, p_{(s,d)} \tag{1}$$

- The summed allocated bandwidth of a link should be no more than its capacity $c(l)$.

$$c(l(p_{(s,d)})) \geq \sum_{p_{(s,d)} \in \mathbb{P}} \sum_{B_{i,j} \in \mathbb{B}} f_{B_{i,j},p_{(s,d)}} \cdot I_{B_{i,j},p_{(s,d)}}, \text{for } \forall l(p_{(s,d)}) \tag{2}$$

- The product of the allocated bandwidth and schedule cycle ΔT should be no less than block size $\mathbb{S}(B_{i,j})$.

$$\mathbb{S}(B_{i,j}) \leq \sum_{p_{(s,d)} \in \mathbb{P}} \sum_{B_{i,j} \in \mathbb{B}} f_{B_{i,j},p_{(s,d)}} \cdot I_{B_{i,j},p_{(s,d)}} \cdot \Delta T, \text{for } \forall B_{i,j} \tag{3}$$

- Only one path will be chosen for a particular block.

$$\sum_{p_{(s,d)} \in \mathbb{P}} I_{B_{i,j},p_{(s,d)}} = 1, \text{for } \forall p_{(s,d)} \tag{4}$$

(4) *Objective.* To speed up the bulk data distribution, BDS aims at maximizing the sum of allocated bandwidth for all the blocks over all the paths.

$$max \sum_{p_{(s,d)} \in \mathbb{P}} \sum_{B_{i,j} \in \mathbb{B}} w(B_{i,j}) \cdot f_{B_{i,j},p_{(s,d)}} \cdot I_{B_{i,j},p_{(s,d)}} \tag{5}$$

where $w(B_{i,j})$ is the weight of $B_{i,j}$.

Dynamic updates: Since any change in network performance or arrival of new request may alter the optimal overlay routing decisions, BDS updates the solutions to the above formulation every cycle of 3 seconds, which is empirically sufficient to achieve near-optimal performance (§6.3.4). Unfortunately, as a mixed-integer LP problem, the original formulation is intractable in practice, because the computational overhead grows exponentially with more potential sources, paths, and data blocks. Specifically, the centralized overlay routing algorithm should pick the data source from 10^4s of servers for 10^5s of blocks, and this scale even grows exponentially when we consider more fine-grained block partitions. Such computational complexity makes it intractable for the controller to explore all the possible paths to make optimal decisions on near-realtime timescales.

[4] Splitting data into fine-grained blocks enables parallel transfer along multiple overlay paths to increase throughput. Too small data blocks increases computational overhead of the routing algorithm. In practice, we found 2MB is a good tradeoff (§6.3.3)

4.2 Decoupling scheduling and routing

At a high level, the key insight that allows BDS to update the overlay control problem is to decouple the aforementioned formulation into two steps: a *scheduling* step which selects a subset of blocks to be transferred ($\overrightarrow{o_{\mathbb{B}}}$), followed by a subsequent *routing* step which determines the residual three tuples: $\langle s^*_{\mathbb{B}}, p^*_{\lambda}, f^*_{\mathbb{B},p^*_{\lambda}} \rangle$, as described in §4.1.

On one hand, such decoupling significantly reduces the computational overhead of the centralized controller. As the scheduling step selects a subset of blocks, and only these selected blocks will be considered in the subsequent routing step, so the searching space is thus significantly reduced.

On the other hand, such decoupling is near-optimal. In the origin formulation, the optimal solution is to decide the optimal block transmission order $\overrightarrow{o^*_{\mathbb{B}}}$ for all blocks, while such decoupling converts this optimal solution into an equivalent one, i.e., to decide the optimal order to transmit the selected blocks in each scheduling cycle (so as to enforce network dynamic updates). Thus, the decoupling introduces little degradation as long as BDS could find the optimal solutions for both scheduling and routing steps in each cycle. Next we'll describe each step in more details.

4.3 Scheduling

The scheduling step selects the subset of blocks to be transferred in each cycle and outputs the transmission order $\overrightarrow{o_{\mathbb{B}}}$, and the objective of this step is to reduce the sheer numbers of potential overlay paths and reduce the searching space of the centralized algorithm.

The key to avoid degradation in reducing the searching space is to make sure that the optimal result is still retained in the reduced searching space. We claim that *the optimal result occurs when availability of all data blocks is balanced*, i.e., all the blocks are duplicated with the same amount of copies. Thus, a simple-yet-efficient way of avoiding degradation when reducing searching space is to balance block availability. We will give a full proof of the above statement in Appendix §9.1. Therefore, in this scheduling step, BDS will firstly pick out the subset of blocks with the least amount of duplicates, to balance the block availability. Such a scheme is similar to BitTorrent's "rarest-first" [15], but BDS selects a set of blocks in each cycle instead of one piece of data, thus can be more efficient than BitTorrent.

4.4 Routing

After the scheduling step selects the block set to transfer ($\overrightarrow{o_{\mathbb{B}}}$), the routing step then decides the residual three tuples, $\langle s^*_{\mathbb{B}}, p^*_{\lambda}, f^*_{\mathbb{B},p^*_{\lambda}} \rangle$, for all blocks.

With the constraints described in §4.1, this formulation in Equation 5 is an integer multi-commodity flow algorithm which is known to be NP-complete [19]. To make this problem tractable in practice, the standard approximation assumes each data file can be infinitesimally split and transferred simultaneously on a set of possible paths between the source and the destination. BDS's actual routing step closely resembles this approximation as BDS also splits data into tens of thousands of fine-grained data blocks (though not infinitesimally), and it can be solved efficiently by standard linear programming (LP) relaxation commonly used in the MCF problem [18, 39]. The key idea is to assume blocks can be infinitesimally split and transferred, so that the original maximization problem (Equation 5) can be re-formulated as transferring at least a fraction α of each task,and the optimal transmission rate of the relaxed problem is $F^* = \alpha f^*$, where f^* is the optimal transmission rate of the original routing problem. Therefore, finding the optimal solution to the original problem becomes simply maximing α instead, and this can be solved in polynomial time [39].

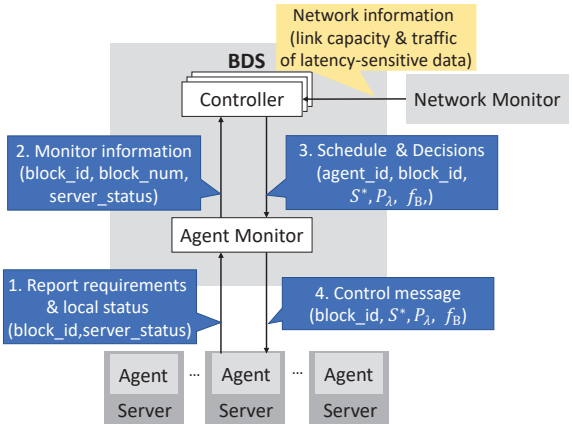


Figure 8: **The system design of BDS.**

However, when splitting tasks infinitesimally, the number of blocks will grow considerably large, and the computing time will be intolerable. BDS adopts two coping strategies: it merges the blocks with the same source and destination pair into one subtask so as to reduce the calculation scale (see blocks merging in §5.2); furthermore, it uses the improved fully polynomial-time approximation schemes (FPTAS) ([17]) to optimize the dual problem of the origin problem and works out an ε-optimal solution (see Appendix §9.2 for the proof of near-optimality of BDS).

5. System Design

This section presents the detailed system design and implemetation of BDS.

5.1 BDS architecture

BDS is built on top of the existing underlying multicast system. The detailed architecture is shown in Figure 8. It consists of four main components: Controller, Agent Monitor, Agent, and Network Monitor. (1) *Controller* runs the centralized scheduling and routing algorithm described in §4. (2) *Agent Monitor* is a messaging layer between *Controller* and *Agents*. (3) *Agent* is a program running on each server, which reports local status to the agent monitor, and handles the specific data transfers according

to the control messages. (4) *Network Monitor* monitors the bandwidth used by latency-sensitive traffic and link utilization.

5.2 Centralized control

The centralized controller of BDS sets a 3-second scheduling cycle, and updates its decisions at the beginning of each cycle. The control interfaces between the controller and agents are two-fold: the *measurement collection* from agents to the controller, and the *control messages* from the controller to agents. The basic workflow in one cycle is following: (1) the local agent on each server checks the block status, records the IDs of blocks that finished downloading in the last cycle, and then reports the information to the agent monitor; (2) Agent Monitor aggregates the information from all the agents, updates the block delivery status and server availability status (e.g., running state, disk state, etc.), and then sends the updated information to the controller; (3) the controller runs the centralized scheduling algorithm, works out the scheduling and routing results, and sends them to the agent monitor; (4) agent monitor then forwards the control messages back to the corresponding local agents; finally, (5) the local agent sets the transmission parameters (e.g., transmission rate) according to the received control message.

There are two optimizations in our implementation.

- *Blocks merging*. To reduce the computational scale and achieve higher-efficient transmissions, BDS merges the blocks with the same source and destination pair into one subtask. This has two advantages: (1) it significantly reduces the number of pending blocks in each scheduling cycle, making calculation on the centralized control algorithm computationally more manageable, (2) it avoids establishing multiple TCP connections between two servers, which could cause performance instability and performance degradation.
- *Non-blocking update*. To avoid being blocked by the controller decision-making, BDS continues the transmissions according to the configurations in the last cycle while waiting for the new configurations. To ensure consistency, the controller estimates the task status at algorithm ending time, and uses that estimated status as inputs of the centralized algorithm.

5.3 Dynamic bandwidth separation

To guarantee dynamic bandwidth separation between inter-DC bulk-data multicasts and delay-sensitive traffic, BDS monitors the aggregated bandwidth usage of all latency-sensitive flows on each inter-/intra-DC link, and dynamically calculates the limit of the total available bandwidth for inter-DC multicasts transfers. To protect delay-sensitive flows under transient traffic bursty, we reserve 20% link capacity to mask network conditions, i.e., the upper bound of the available bandwidth for bulk-data transfers is set to be the difference between 80% of link capacity and the aggregated bandwidth usage of latency-sensitive traffic.

The dynamic bandwidth separation of BDS enjoys two advantages. (1) It uses global coordination to achieve more efficient bandwidth utilization. The traditional priority-based techniques (such as [26]) that set higher priority to online latency-sensitive traffic than background traffic cannot react to the dynamic network environment, resulting in bandwidth wastage or performance interference. While BDS can dynamically monitor the aggregated bandwidth occupied by delay-sensitive applications, and reserves moderate amounts of bandwidth for them. (2) BDS, which optimized the application-level overlay, is complementary to network-layer techniques that improve the WAN performance and fairness [10, 24, 31, 40].

5.4 Fault tolerance

Next we describe how BDS handles the following failure types.

1. *Controller failure:* The controller is replicated using standard schemes [27]. If the master controller fails, another replica will be elected as the new controller. If the network partition happens between DCs and the controller or when all controller copies fail, the agents running in servers will fallback to the current decentralized overlay protocol as default to ensure graceful performance degradation.
2. *Server failure:* If the agent on server is still able to work, it will report the abnormal state (e.g., server crash, disk failure, etc.) to the agent monitor. Otherwise, the servers that selected this server as data source would report the unavailability to the agent monitor. In either case, the controller will remove that server from the potential data sources in the next cycle.
3. *Network partition between DCs:* If network partition happens between DCs, the DCs located in the same partition with the controller will work the same as before, while the separated DCs will fallback to the decentralized overlay network.

5.5 Implementation and deployment

We have deployed BDS on Baidu's DCs, which consist of 67 geo-distributed servers in 10 DCs. Evaluation in the next section is based on this deployment.

BDS is implemented with 3621 lines of golang code [1], and it was fully integrated in Baidu's DCs. The duplications of the controller are implemented on three different geo-located zookeeper servers. The data plane bulk data transmissions between controller and agents use TCP, and the control plane decision messages use `HTTP POST`. For specific transmissions, BDS uses `wget` to make data transfer, and enforce bandwidth by `--limit-rate`. The agent running in each server uses Linux Traffic Control (`tc`) to enforce the limit on the total bandwidth usage of inter-DC multicast traffic.

BDS n be seamlessly integrated with applications because BDS's implementation makes no specific requirements on applications. All the applications need to do is to call the public APIs provided by BDS to complete three steps: first, register on BDS, assign the source DC, destination DCs, servers and the bulk data; second, install agents on all those servers; third, set the start time of bulk data transfers. Then BDS will start the data distribution at the specified time. Such simple implementation also makes BDS applicable to other companies' DCs.

6. Evaluation

To evaluate BDS, we integrated our end-to-end prototype in Baidu, and ran a pilot deployment under the above implementation. Using a combination of pilot deployment, trace-driven simulation, and microbenchmarking, we show that:

1. BDS completes inter-DC multicast 3-5× faster than Baidu's existing solution and other baselines used in industry (§6.1).
2. BDS significantly reduces the incidents of interferences between bulk-data multicast traffic and latency-sensitive traffic (§6.2).
3. BDS can scale to the traffic demand of a large online service provider, tolerate various failure scenarios, and is close to the theoretically optimal performance (§6.3).

6.1 Performance improvement over existing solutions

We compare BDS against three existing solutions: Baidu's existing solution (a decentralized strategy which has been evolving for 5 years with end-to-end evaluations), Bullet [25] (a centralized strategy), and Akamai's overlay network [9] for live video streaming multicast.

6.1.1 Methodology

We start with the methodology.

Pilot deployment: We choose several services with different data sizes and run randomized A/B testing in which we randomly choose several days to use BDS instead of the current solution. To make sure the two solutions work in the similar environment, we conduct all the experiments at 02:00 every day in the morning.

Trace-driven simulation: Complementary to the pilot deployment on real traffic, we also use trace-driven simulation to evaluate BDS on a larger scale. We simulate the other two overlay multicast techniques by setting topology, DC scale and configuring servers with the same scale as BDS's DCs, and replay inter-DC multicast data requests in the same chronological order as in the pilot deployment (including the number of requests and the number of nodes).

6.1.2 BDS vs. Baidu's existing solution

We first choose one service with 70 TB aggregated daily bulk data, which is distributed stored in the source DC (with 1000 servers). Each of the other DCs downloads a copy of the data and stores it in the same way with 1000 servers.

To intuitively present the overall improvements by BDS, we draw the cumulative distribution function (CDF) of the completion time in Figure 9a, from which we can see that the completion time under Baidu's existing solution is about 200 minutes while that under BDS is less than 40 minutes, which is 5 times shorter.

For detailed illustration, we further pick three applications whose data volumes are large, medium and small, and compare BDS's and Baidu's mean (stddev) completion time for each application in Figure 9b. These bars show that the benefit of BDS increases along with the bulk data size. For small data volume, BDS outperforms Baidu by 1.6 times, and more than 3 times than Baidu for large-data transfers.

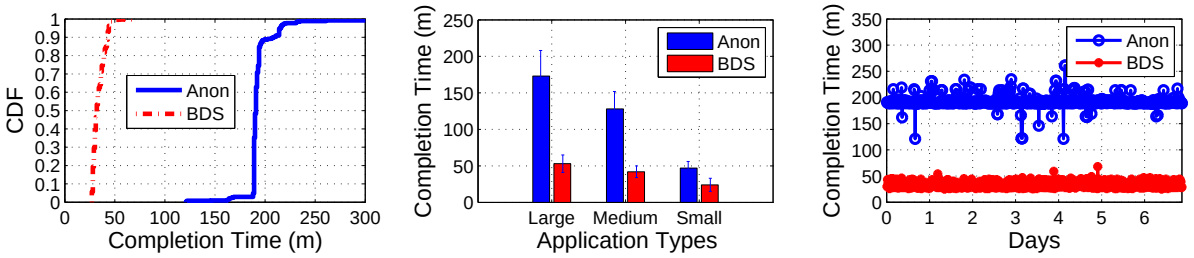
We also repeat the experiments for 7 days to offer more statistical significance and draw the average completion time of both BDS and Baidu in Figure 9c. BDS consistently outperforms Baidu by 4 times, and has less performance variations. is quite stable during these 7 days while Baidu illustrates some volatilities. In general, BDS achieves about 4.5 times shorter completion time than Baidu.

6.1.3 BDS vs. other overlay multicast techniques

In the trace-driven simulations, we use a topology with 13 DCs (one of which acts as the source DC) and set all the parameters the same as the real network, including data file size, data source and destination, paths, link capacities and server upload/download rates. We use two baselines: Bullet [25] is a data dissemination system using an overlay mesh, and Akamai's video streaming system [9] in overlay multicast networks.

We conduct three series of experiments and show the results in Table 3: baseline, large-scale and small bandwidth experiments. In the baseline experiments, the size of the bulk data is 10TB, the number of servers per DC is 100, and the upload and download rate limits are set to 20MBs. In the large-scale experiment, the bulk data is 100TB, server number per DC is 1000. In the small bandwidth experiment, server upload and download rate limit are reduced to 5MBs. We find that BDS achieves 3 times shorter completion time than Bullet and Akamai in the baseline experiments, and more than 4 times shorter completion time in the large-scale and small bandwidth experiments.

Considering the results in large-scale experiments, we found that the completion time of Bullet (and Akamai) increase to more than 3 times compared that in the baseline experiments - from $28m$ (and $25m$)to about $82m$ (and $87m$), while BDS only increases to about 2 times - from $9.41m$ to $20.33m$. This proves BDS's good scalability. Similarly, in the small bandwidth experiments (which is very common when delay-sensitive traffic bursts), the completion time of Bullet and Akamai increases to $171m$ and $138m$, respectively, while that of BDS is just $38.25m$. This result illustrates the high efficiency of BDS under strict bandwidth limitations.



(a) Distribution of completion time. (b) Comparison by application types. (c) Comparison by completion time.

Figure 9: **[BDS vs. Baidu] Results from pilot deployments.**

Solution	Baseline	Large Scale	Rate Limit
Bullet	$28m$	$82m$	$171m$
Akamai	$25m$	$87m$	$138m$
BDS	$9.41m$	$20.33m$	$38.25m$

Table 3: **[BDS vs. Bullet [25], Akamai [9]] Completion time of the three solutions in trace-driven simulation.**

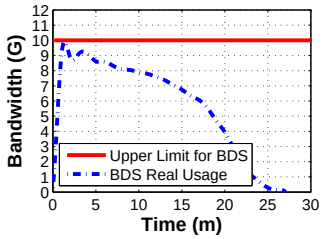


Figure 10: **The effectiveness of bandwidth separation.**

System	Source DC egress link	l_1	l_2	l_3
Baidu	69.82%	53.09%	57.98%	63.01%
BDS	70.55%	62.46%	63.23%	64.24%

Table 4: **Average link utilizations of the source DC egress link and 3 inter-DC links (l_1, l_2, l_3) under Baidu and BDS.**

6.2 Benefits of coordinated bandwidth allocation

To test the effectiveness of bandwidth separation and check whether the bulk data transfers affect latency-sensitive traffic, we set the upper bound of available bandwidth for bulk data transfer to 10 GBs, and monitor the real usage of one inter-DC link. The results are shown in Figure 10, from which we can see the real bandwidth usage stays below 10 GBs throughout the whole transmission process. This shows that BDS can effectively separate bandwidth and thus reduce the incidents of delay on latency-sensitive traffic caused by bulk data transfers.

As BDS implements strict bandwidth separation between latency-sensitive traffic and bulk data transfers while still showing shorter completion time, does it occupy too much bandwidth on inter-DC links? To answer this question, we record the average utilizations of the egress link of the source DC and 3 randomly selected inter-DC links, denoted as l_1, l_2 and l_3. The result in Table 4 shows that link utilizations do not change much with BDS. This is because BDS spreads data transfers over bottleneck-disjoint paths, so it avoids transferring the same data on the same link.

6.3 Micro-benchmarks

This section examines BDS in very details through micro-benchmarks. (1) Scalability of the centralized control. (2) Fault tolerance. (3) Setting of BDS parameters.

6.3.1 Scalability

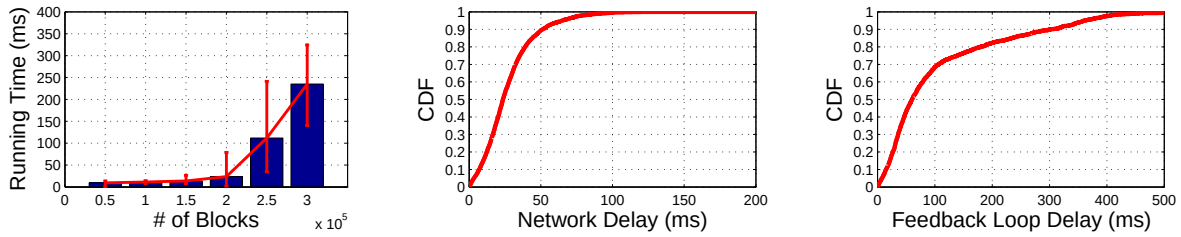
Controller running time: As the controller needs to assign specific transmissions for all blocks, the algorithm running time is naturally related to the number of blocks. We show the relationship of block number and algorithm running time in Figure 11a, from which we can see that the controller running time is still lower than $300ms$ even when there are 3×10^5 blocks. This is the instantaneous peak value of block number when the bulk data size is $10TB$, and the block merging scheme described in §5.2 will further reduce the block number and reduce the algorithm running time.

Network delay: BDS works in inter-DC networks, so the network delay among DCs is a key factor in the algorithm updating process. We record the network delay of 5000 requests and present the CDF in Figure 11b. We can see that 90% of the network delay is below $50ms$ and the average value is about $25ms$, which is less than 1% of the decision updating cycle (3 seconds).

Feedback loop delay: For centralized algorithms, feedback loop delay is essential to algorithm scalability. In BDS, this feedback loop consists of several procedures: status updating from agents to the controller, running of the centralized algorithm, and decision updating from the controller back to agents. We measure the delay of the whole process, show the CDF in Figure 11c, and find that in most cases (over 80%), the feedback loop delay is lower than $200ms$. So we can claim that BDS demonstrates a short enough update delay and is able to scale to even larger systems.

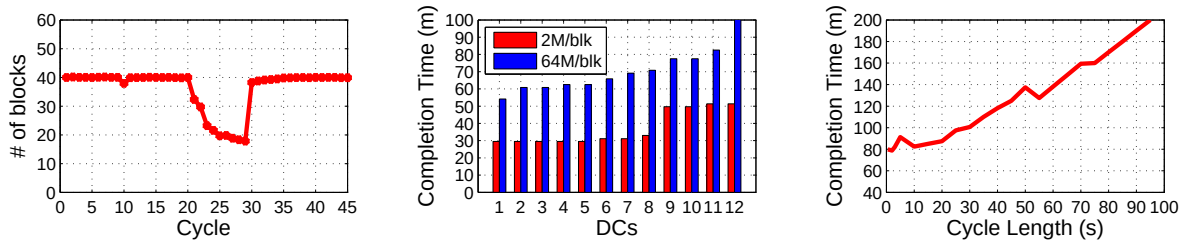
6.3.2 Fault tolerance

§5.4 introduces the working principles under failures. Here we examine the impact of the following failure scenario on the number of downloaded blocks per cycle. During cycles 0 to 9, BDS works as usual, and one agent fails in the 10th cycle. The controller fails in the 20th cycle and recovers in the 30th cycle. Figure 12a shows the average number of downloaded blocks per cycle. We find that the slight



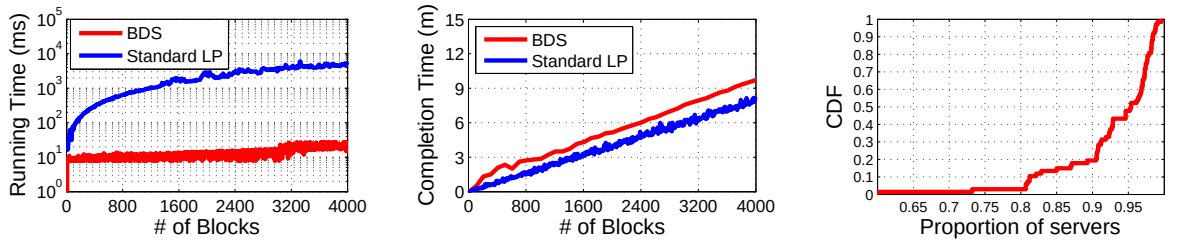
(a) The controller running time. (b) The inter-DC network delay. (c) Feedback loop delay.

Figure 11: **[System scalability] Measurements on (a) controller running time, (b) network delay, (c) Feedback loop delay.**



(a) Average number of downloaded blocks per cycle under failures. (b) Completion time under different block sizes. (c) Completion time under different cycle lengths.

Figure 12: **BDS's (a) fault tolerance, (b) sensitivity to different block sizes, and (c) different cycle lengths.**



(a) The reduction on algorithm running time of BDS over standard LP. (b) The near-optimality of BDS to standard LP in small scale. (c) The proportion of blocks downloaded from the original source.

Figure 13: **[In-depth analysis] on (a) reduction on algorithm running time, (b) near-optimality, and (c) effects of overlay transmission.**

impact of agent failure only lasts for one cycle, and the system recovers in the 11th cycle. When the controller is unavailable, BDS falls back to a default decentralized overlay protocol, resulting in graceful performance degradation. With the recovery of the controller, the performance recovers in the 30th cycle.

6.3.3 Choosing the values of key parameters

Block size: In BDS, the bulk data file is split into blocks and can be transferred on bottleneck-disjoint paths. But this introduces a tradeoff between scheduling efficiency and calculation overhead. We therefore conduct two series of experiments using different block sizes (2MB and 64MB). Figure 12b shows that the completion time in the 2MB/block scenario is 1.5 to 2 times shorter than that in the 64MB/block scenario. This is because smaller block sizes result in closer-to-optimal performance (see Appendix for the proof). However, this optimization introduces longer controller running time, as shown in Figure 11a. We pick block size by balancing two considerations: (1) requirements on the completion time, and (2) controller's operation overhead.

Update cycle lengths: Since any change in network environment may potentially alter the optimal overlay routing decisions, BDS reacts to the changing network conditions by adjusting the routing scheme periodically. To testify the adjustment frequency, we set different cycle lengths from 0.5s to 95s to the same bulk data transfer, and Figure 12c shows the completion time. Smaller cycle lengths result in shorter completion time, but the benefit diminishes when the cycle length is less than 3s. This is because too frequent updates introduce more overhead on: (1) the information collection from agents to the controller, (2) the execution of the centralized algorithm, and (3) the re-establishment of new TCP connection. Thus, considering adjustment granularity and the corresponding overhead, we finally choose 3s as the default cycle length.

6.3.4 In-depth analysis.

Optimization over algorithm running time: BDS decouples scheduling and routing, which can significantly reduce the computational complexity. To clearly show the optimization, we measure the algorithm running time under BDS and the standard linear programming (LP) solution. For the standard LP experiments, we use the *linprog* library on MATLAB [5], set the upper bound of the iteration number (10^6) if the algorithm does not converge, and record the CPU time as a function of the block number. Figure 13a shows that the running time of BDS keeps below 25*ms* while that of standard LP grows quickly to 4*s* with only 4000 blocks. BDS is much faster than off-the-shelf LP solver.

Near-optimality of BDS: To measure the near-optimality, we evaluate the data transfer completion time under the standard LP and BDS: 2 DCs, 4 servers, 20MBs for server upload/download rate. We vary the number of blocks from 1 to 4000, over which the LP solver can't finish in reasonable time. Figure 13b shows that the difference of completion time between BDS and the optimal standard LP is about 15%.

Benefit of disjoint overlay paths: §2.2 reveals the benefits of disjoint paths on application-level overlay networks. To explore the potential benefit, we record the ratio of the number of blocks downloaded from the original source to the total number of blocks, and the CDF is shown in Figure 13c. For about 90% servers, the proportion is less than 20%, which means that more than 80% blocks are downloaded from other DCs on the disjoint paths, demonstrating great potential of multicast overlay network.

In summary, both the prototype pilot deployment and the trace-driven simulations of BDS show 3-5× speedup over existing solutions, with good scalability and reliability, and near-optimal scheduling results.

7. Related Work

Here we discuss some representative work that is related to BDS in three categories.

Overlay Network Control. Overlay network releases great potential to various applications, especially for data transfer applications. The representative networks include Peer-to-Peer (P2P) network and Content Delivery Network (CDN). As a mature distributed application protocol, P2P has already been verified by many applications, such as live streaming systems (CoolStreaming [54], Joost [2], PPStream [4], UUSee [6]), video-on-demand (VoD) applications (OceanStore [3]), distributed hash tables [42] and even nowaday's Bitcoin [16], but it cannot achieve the optimal solution due to the lack of global visibility. CDN distributes services spatially relative to end-users to provide high availability and performance (e.g., to reduce page load time), serving many applications such as multimedia [56] and live streaming [47].

We briefly introduce the two baselines in the evaluation section: (1) Bullet [25], which enables geo-distributed nodes to self-organize into a overlay mesh. Specifically, each node uses RanSub [43] to distribute summary ticket information to other nodes and receive disjoint data from its sending peers. The main difference between BDS and Bullet lies in the control scheme, i.e., BDS is a centralized method that has a global view of data delivery states, while Bullet is a decentralized scheme and each node makes decision locally. (2) Akamai designs a 3-layer overlay network for delivering live streams [9], where a source forwards its streams to reflectors, and reflectors send outgoing streams to stage sinks. There are two main differences between Akamai and BDS. First, Akamai adopts a 3-layer topology where edge servers receive data from their parent reflectors, while BDS explores bigger searching space without layer limitation. Second, the receiving sequence of data must be sequential in Akamai because it serves for the live streaming application. But there is no such requirements in BDS, and the side effect is that BDS has to calculate the optimal order $\overrightarrow{o_B}$ as additional work.

Data Transfer and Rate Control. Rate control of transport protocols in DC-level plays an important role on data transmissions. DCTCP [8], PDQ [20], CONGA [7], DCQCN [57] and TIMELY [32] are all classical protocols showing obvious improvements on transmission efficiency. Besides, some congestion control protocols like the credit-based ExpressPass [12] and load balancing protocols like Hermes [52] could further reduce flow completion time by making rate control. On this basis, the recent proposed Numfabric [35] and Domino [46] further explore the potential of centralized TCP on speeding up data transfer and improving DC throughput. To some extend, co-flow scheduling [13, 51] is a little bit similar to the multicast overlay scheduling, in terms of data parallelism. But they focus on flow-level problems while BDS is designed on application-level.

Centralized Traffic Engineering. Traffic engineering (TE) has long been a hot research topic, and many existing studies [10, 24, 31, 40, 45, 49, 53] have illustrated the challenges of scalability, heterogeneity etc., especially on inter-DC level. The representative TE systems include Google's B4 [23] and Microsoft's SWAN [21]. B4 adopts SDN [29] and OpenFlow [30, 37] to manage individual switches and deploy customized strategies on the paths. SWAN is another online traffic engineering platform, which achieves high network utilization with its software-driven WAN.

Overall, an application-level multicast overlay network is essential to data transfer in inter-DC WANs. Applications like user logs, search engine indexes and databases would benefit much from bulk-data multicast. Furthermore, such benefits are orthogonal to prior WAN optimizations, further improving inter-DC applications' performance.

8. Conclusion

Inter-DC multicast is critical to the performance of global-scale online service providers, but prior efforts that focus on optimizing WAN performance are insufficient. This paper presents BDS, an application-level multicast overlay network that substantially improves the performance of bulk-data inter-DC multicast. BDS demonstrates the feasibility and practical benefits of a fully centralized multicast overlay network that selects overlay paths and schedules data transfers in a near-optimal yet efficient manner. The key insight underlying BDS's centralized design is that the benefits of making slightly delayed decisions based on a global view outweigh the cost of centralization, such as gathering information to the controller. We believe that the insight that informs our design choice can be generalized to inspire centralized control platforms in other settings where centralized decision-making strikes a favorable balance between costs and benefits.

References

[1] The go programming language. https://golang.org.

[2] Joost. http://www.joost.com/.

[3] Oceanstore. http://oceanstore.cs.berkeley.edu/.

[4] Ppstream. http://www.ppstream.com/.

[5] Solve linear programming problems - matlab linprog. https://cn.mathworks.com/help/optim/ug/linprog.html?s_tid=srchtitle.

[6] Uusee. http://www.uusee.com/.

[7] ALIZADEH, M., EDSALL, T., DHARMAPURIKAR, S., VAIDYANATHAN, R., CHU, K., FINGERHUT, A., LAM, V. T., MATUS, F., PAN, R., AND YADAV, N. Conga: distributed congestion-aware load balancing for datacenters. In *ACM Conference on SIGCOMM* (2014), pp. 503–514.

[8] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data center tcp (dctcp). In *ACM SIGCOMM 2010 Conference* (2010), pp. 63–74.

[9] ANDREEV, K., MAGGS, B. M., MEYERSON, A., AND SITARAMAN, R. K. Designing overlay multicast networks for streaming. *Spaa Proceedings of the Fifteenth Annual Acm Symposium on Parallel Algorithms and Architectures* (2013), 149–158.

[10] CHEN, Y., ALSPAUGH, S., AND KATZ, R. H. Design insights for mapreduce from diverse production workloads. Tech. rep., DTIC Document, 2012.

[11] CHERNOFF, H. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statist. 23*, 4 (12 1952), 493–507.

[12] CHO, I., JANG, K. H., AND HAN, D. Credit-scheduled delay-bounded congestion control for datacenters. In *Conference of the ACM Special Interest Group on Data Communication* (2017), pp. 239–252.

[13] CHOWDHURY, MOSHARAFSTOICA, AND EECS, I. Coflow: An application layer abstraction for cluster networking. *Acm Hotnets* (2012).

[14] CHU, Y.-H., RAO, S. G., AND ZHANG, H. A case for end system multicast (keynote address). In *ACM SIGMETRICS Performance Evaluation Review* (2000), vol. 28, ACM, pp. 1–12.

[15] COHEN, B. Incentives build robustness in bittorrent. *Proc P Economics Workshop* (2003), 1–1.

[16] EYAL, I., GENCER, A. E., SIRER, E. G., AND VAN RENESSE, R. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (2016), pp. 45–59.

[17] FLEISCHER, L. K. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics 13*, 4 (2000), 505–520.

[18] GARG, N., AND KOENEMANN, J. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing 37*, 2 (2007), 630–652.

[19] GARG, N., VAZIRANI, V. V., AND YANNAKAKIS, M. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica 18*, 1 (1997), 3–20.

[20] HONG, C. Y., CAESAR, M., AND GODFREY, P. B. Finishing flows quickly with preemptive scheduling. pp. 127–138.

[21] HONG, C.-Y., KANDULA, S., MAHAJAN, R., ZHANG, M., GILL, V., NANDURI, M., AND WATTENHOFER, R. Achieving high utilization with software-driven wan. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 15–26.

[22] HUANG, T. Y., JOHARI, R., MCKEOWN, N., TRUNNELL, M., AND WATSON, M. A buffer-based approach to rate adaptation: evidence from a large video streaming service. *Acm Sigcomm Computer Communication Review 44*, 4 (2014), 187–198.

[23] JAIN, S., KUMAR, A., MANDAL, S., ONG, J., POUTIEVSKI, L., SINGH, A., VENKATA, S., WANDERER, J., ZHOU, J., ZHU, M., ET AL. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 3–14.

[24] KAVULYA, S., TAN, J., GANDHI, R., AND NARASIMHAN, P. An analysis of traces from a production mapreduce cluster. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010* (2010), IEEE, pp. 94–103.

[25] KOSTIĆ, D., RODRIGUEZ, A., ALBRECHT, J., AND VAHDAT, A. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SIGOPS Operating Systems Review* (2003), vol. 37, ACM, pp. 282–297.

[26] KUMAR, A., JAIN, S., NAIK, U., RAGHURAMAN, A., KASINADHUNI, N., ZERMENO, E. C., GUNN, C. S., AI, J., CARLIN, B., AMARANDEI-STAVILA, M., ET AL. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), ACM, pp. 1–14.

[27] LAMPORT, L. The part-time parliament. *ACM Transactions on Computer Systems (TOCS) 16*, 2 (1998), 133–169.

[28] LIEBEHERR, J., NAHAS, M., AND SI, W. Application-layer multicasting with delaunay triangulation overlays. *IEEE Journal on Selected Areas in Communications 200*, 8 (2002), 1472–1488.

[29] MCKEOWN, N. Software-defined networking. *INFOCOM keynote talk 17*, 2 (2009), 30–32.

[30] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[31] MISHRA, A. K., HELLERSTEIN, J. L., CIRNE, W., AND DAS, C. R. Towards characterizing cloud backend workloads: insights from google compute clusters. *ACM SIGMETRICS Performance Evaluation Review 37*, 4 (2010), 34–41.

[32] MITTAL, R., LAM, V. T., DUKKIPATI, N., BLEM, E., WASSEL, H., GHOBADI, M., VAHDAT, A., WANG, Y., WETHERALL, D., AND ZATS, D. Timely: Rtt-based congestion control for the datacenter. In *ACM Conference on Special Interest Group on Data Communication* (2015), pp. 537–550.

[33] MOKHTARIAN, K., AND JACOBSEN, H. A. Minimum-delay multicast algorithms for mesh overlays. *IEEE/ACM Transactions on Networking 23*, 3 (2015), 973–986.

[34] MUKERJEE, M. K., HONG, J., JIANG, J., NAYLOR, D., HAN, D., SESHAN, S., AND ZHANG, H. Enabling near real-time central control for live video delivery in cdns. In *ACM SIGCOMM Computer Communication Review* (2014), vol. 44, ACM, pp. 343–344.

[35] NAGARAJ, K., BHARADIA, D., MAO, H., CHINCHALI, S., ALIZADEH, M., AND KATTI, S. Numfabric: Fast and flexible bandwidth allocation in datacenters. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference* (2016), ACM, pp. 188–201.

[36] NYGREN, E., SITARAMAN, R. K., AND SUN, J. *The Akamai network: a platform for high-performance internet applications*. ACM, 2010.

[37] OPENFLOW. Openflow specification. http://archive.openflow.org/wp/documents.

[38] RAGHAVAN, P., AND THOMPSON, C. D. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica 7*, 4 (1987), 365–374.

[39] REED, M. J. Traffic engineering for information-centric networks. In *2012 IEEE International Conference on Communications (ICC)* (2012), IEEE, pp. 2660–2665.

[40] REISS, C., TUMANOV, A., GANGER, G. R., KATZ, R. H., AND KOZUCH, M. A. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing* (2012), ACM, p. 7.

[41] REPANTIS, T., SMITH, S., SMITH, S., AND WEIN, J. Scaling a monitoring infrastructure for the akamai network. *Acm Sigops Operating Systems Review 44*, 3 (2010), 20–26.

[42] RHEA, S., GODFREY, B., KARP, B., KUBIATOWICZ, J., RATNASAMY, S., SHENKER, S., STOICA, I., AND YU, H. Opendht: a public dht service and its uses. In *ACM SIGCOMM Computer Communication Review* (2005), vol. 35, ACM, pp. 73–84.

[43] RODRIGUEZ, A., ALBRECHT, J., BHIRUD, A., AND VAHDAT, A. Using random subsets to build scalable network services. In *Conference on Usenix Symposium on Internet Technologies and Systems* (2003), pp. 19–19.

[44] SAVAGE, S., COLLINS, A., HOFFMAN, E., SNELL, J., AND ANDERSON, T. The end-to-end effects of internet path selection. pp. 289–299.

[45] SHARMA, B., CHUDNOVSKY, V., HELLERSTEIN, J. L., RIFAAT, R., AND DAS, C. R. Modeling and synthesizing task placement constraints in google compute clusters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (2011), ACM, p. 3.

[46] SIVARAMAN, A., CHEUNG, A., BUDIU, M., KIM, C., ALIZADEH, M., BALAKRISHNAN, H., VARGHESE, G., MCKEOWN, N., AND LICKING, S. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference* (2016), ACM, pp. 15–28.

[47] SRIPANIDKULCHAI, K., MAGGS, B., AND ZHANG, H. An analysis of live streaming workloads on the internet. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement* (2004), ACM, pp. 41–54.

[48] WANG, F., XIONG, Y., AND LIU, J. mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast. In *International Conference on Distributed Computing Systems* (2007), p. 49.

[49] WILKES, J. More google cluster data. http://googleresearch.blogspot.com/2011/11/, 2011.

[50] ZHANG, H., CHEN, K., BAI, W., HAN, D., TIAN, C., WANG, H., GUAN, H., AND ZHANG, M. Guaranteeing deadlines for inter-datacenter transfers. In *Proceedings of the Tenth European Conference on Computer Systems* (2015), ACM, p. 20.

[51] ZHANG, H., CHEN, L., YI, B., CHEN, K., GENG, Y., AND GENG, Y. Coda: Toward automatically identifying and scheduling coflows in the dark. In *Conference on ACM SIGCOMM 2016 Conference* (2016), pp. 160–173.

[52] ZHANG, H., ZHANG, J., BAI, W., CHEN, K., AND CHOWDHURY, M. Resilient datacenter load balancing in the wild. In *the Conference of the ACM Special Interest Group* (2017), pp. 253–266.

[53] ZHANG, Q., HELLERSTEIN, J. L., AND BOUTABA, R. Characterizing task usage shapes in google's compute clusters. In *Large Scale Distributed Systems and Middleware Workshop (LADIS'11)* (2011).

[54] ZHANG, X., LIU, J., LI, B., AND YUM, Y.-S. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* (2005), vol. 3, IEEE, pp. 2102–2111.

[55] ZHANG, Y., XU, K., YAO, G., ZHANG, M., AND NIE, X. Piebridge: A cross-dr scale large data transmission scheduling system. In *Proceedings of the ACM SIGCOMM 2016 Conference* (2016), ACM, pp. 553–554.

[56] ZHU, W., LUO, C., WANG, J., AND LI, S. Multimedia cloud computing. *IEEE Signal Processing Magazine 28*, 3 (2011), 59–69.

[57] ZHU, Y., ERAN, H., FIRESTONE, D., GUO, C., LIPSHTEYN, M., LIRON, Y., PADHYE, J., RAINDEL, S., YAHIA, M. H., AND ZHANG, M. Congestion control for large-scale rdma deployments. *Acm Sigcomm Computer Communication Review 45*, 5 (2015), 523–536.

9. Appendix

9.1 Proof 1: The optimal result is yielded from balance availability of blocks

Assume the origin data in the source DC is split into n blocks, and there are $2m$ destination DCs in the WAN. Different scheduling strategies output different block subsets and lead to different intermediate transmission states, finally resulting in different completion time. Take two intermediate states as examples: 1) All of the n blocks are balanced duplicated (i.e., each with k duplicates); 2) Blocks are imbalanced duplicated (some have $k1$ duplicates and some have $k2$ duplicates, $(k1 < k < k2)$). Let t_1 denote the completion time of case 1 and t_2 denote that of case 2. We then try to calculate the completion time.

According to the duplication situations, we classify the m DCs into two sets:

$N_a = \{DC_i | DC_i \text{ has already downloaded block } a\}$

$N'_a = \{DC_i | DC_i \text{ hasn't downloaded block } a\}$

where $|N_a| + |N'_a| = 2m$. In case 1, all the n blocks have k duplicates, so $|N_i| = k$ and $|N'_i| = 2m - k$. These $2m - k$ transmissions have k optional data sources to download this block. Let R_{up}/R_{down} denote the upload/download rate limits of each server, then we can calculate t_1 by Equation 6:

$$t_1 = \frac{\mathbb{S}}{min\{c(l), \frac{R_{up} \times k}{2m-k}, R_{down}\}} \tag{6}$$

where $\mathbb{S}$ denotes the size of the remaining data, and $c(l) > R_{up}/2m$ (server upload bandwidth is the bottleneck). This equation is a monotonically decreasing function of k according to the following calculus.

$$\begin{aligned} t'(k) &= \frac{d(t(k))}{d(k)} = \frac{d(\frac{\mathbb{S} \times (2m-k)}{R_{up} \times k})}{d(k)} \\ &= \frac{d(\frac{2m \times \mathbb{S}}{R_{up} \times k})}{d(k)} = -\frac{2m\mathbb{S}}{R_{up} \times k^2} \\ &< 0 \end{aligned} \tag{7}$$

Therefore, in case 2, for those blocks with $k1$ $(k1 < k)$ duplicates, the completion time t_{k1} will be larger than t_1, i.e., $t_2 \geq t_{k1} > t_1$. Therefore, we can conclude that the optimal result is yielded from the balance availability of all data blocks.

9.2 Proof 2: Near Optimality of BDS

Here we present the proof of rounding the integer solution.

BDS distributes each task in an optimal manner given by the maximum concurrent flow (MCF) solution [39]. However, the MCF solution is a linear relaxation of the original integer problem and in practice the solution requires that flows are routed as integers of the block size flow. There are a number of strategies using the output of the MCF for the integer path allocation, however, none of them will be strictly optimal as the original problem is NP-complete. A common strategy is to use *randomized-rounding*. Randomized-rounding was first analyzed in detail for some relaxation approaches by Raghavan and Thompson [38] and a similar approach is followed here for the specific case of the MCF relaxation for which they do not provide a solution. A flow through an arbitrary edge e is denoted as $f_e = \sum_{e \in p_\lambda} F_{B_{i,j},p_\lambda} \ \forall \ p_\lambda$ where F^* is the optimum value in the MCF problem that is equivalent to f^* in the original problem. f_e is the optimal MCF flow through edge e that in the integer solution is approximated from the flow of multiple blocks each of flow rate b. To perform the randomized rounding, first, each of the *fractional flows* $F_{B_{i,j},p_\lambda}$ is truncated to an *integer flow* that is constructed from a number of *block flows*. Each integer flow is denoted as $\lfloor F_{B_{i,j},p_\lambda} \rfloor = kb \leq F_{B_{i,j},p_\lambda}$ where k is strictly the largest positive integer, or zero, to meet the constraint. Then each integer flow is either kept at kb or *rounded* up with probability $(F_{B_{i,j},p_\lambda} - \lfloor F_{B_{i,j},p_\lambda} \rfloor)/b$ to $(k+1)b$. This rounding probability means that the expected value of the integer flows is f_e, the solution of the optimal linear MFC. The problem is then to determine the likelihood that this rounding will cause the capacity constraint on an edge e to be exceeded. To aid the discussion, a *residual flow* $f'_e = f_e - \sum_{e \in p_\lambda} \lfloor F_{B_{i,j},p_\lambda} \rfloor$ is used to describe the flow that needs to be added to the truncated flows to obtain the optimal solution obtained from MFC.

As each flow rounding is an independent random variable, we can determine the likelihood of exceeding the capacity constraint by applying the following well-known Chernoff bound [11]:

$$\Pr(X > (1+\delta)\mu) \leq e^{\frac{2\delta^2\mu^2}{n(b-a)^2}} \tag{8}$$

where $X = X_1 + \ldots X_n$ is the sum of independent, bounded, random variables $a \leq X_i \leq b$ and $\mu = E(X)$. This bound describes the probability that a summation of random variables exceeds the expected value of the sum by a factor of δ. The random variables are the rounded flows which have an expected summation $\mu = f'_e$. Consider an edge, e, that has its capacity fully used by the linear MFC solution. Using the Chernoff bound from Equation 8, note that $a = 0$ and the

highest value of the rounded flow added is one block with flow b, we find that the integer flow is beyond optimal and the residual flow is bounded by:

$$\delta \leq \frac{b}{r}\sqrt{\frac{n}{2}\ln\varepsilon} \tag{9}$$

with probability $1-\varepsilon$. It should be noted that as the size of the blocks becomes small compared to the overall flow f_e, the probability that the linear flow is exceeded by a factor δ becomes exponentially small due to the exponential form of the Chernoff bound.