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

Yuchao Zhang
HKUST

Junchen Jiang
The University of Chicago

Ke Xu
Tsinghua University

Xiaohui Nie
Tsinghua University

Martin J. Reed
University of Essex

Haiyang Wang
University of Minnesota at Duluth

Guang Yao
Baidu

Miao Zhang
Baidu

Kai Chen
HKUST

ABSTRACT

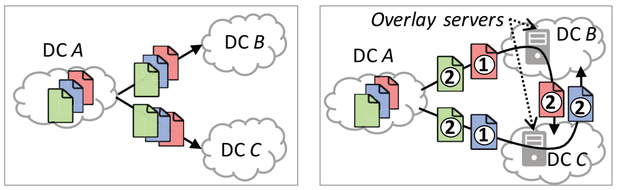
Many important cloud services require replicating massive data from one datacenter (DC) to multiple DCs. While the performance of pair-wise inter-DC data transfers has been much improved, prior solutions are insufficient to optimize bulk-data multicast, as they fail to explore the capability of servers to store-and-forward data, as well as the rich inter-DC overlay paths that exist in geo-distributed DCs. To take advantage of these opportunities, we present *BDS*, an application-level multicast overlay network for large-scale inter-DC data replication. 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 the 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EuroSys '18, April 23–26, 2018, Porto, Portugal
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5584-1/18/04…$15.00
https://doi.org/10.1145/3190508.3190519

ACM Reference Format:
Yuchao Zhang, Junchen Jiang, Ke Xu, Xiaohui Nie, Martin J. Reed, Haiyang Wang, Guang Yao, Miao Zhang, and Kai Chen. 2018. BDS: A Centralized Near-Optimal Overlay Network for Inter-Datacenter Data Replication. In *EuroSys '18: Thirteenth EuroSys Conference 2018, April 23–26, 2018, Porto, Portugal.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3190508.3190519

1 INTRODUCTION

For large-scale online service providers, 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 [27, 58]. As more DCs are deployed globally and bulk data are exploding, inter-DC traffic then needs to be replicated in a frequent and efficient manner.



(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 $A \to B \to C$ and $A \to C \to B$ 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., [22, 24, 27, 44, 51, 57]), 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 incomplete, 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, 29, 34, 48]), building one for inter-DC multicast traffic poses two challenges. First, as each DC has tens of thousands of servers, the resulting large number 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 [23, 26, 41], which leads to suboptimal decisions for lack of global information, or restricts itself to strictly structured (e.g., layered) topologies [37], which fails to leverage all possible overlay paths. Second, even a small increase in the delay of latency-sensitive traffic can cause significant revenue loss [56], so the bandwidth usage of inter-DC bulk-data multicasts must be tightly controlled to avoid negative impact on other latency-sensitive traffic.

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. (2) 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; this is four orders of magnitude faster than solving routing and scheduling jointly when considering the workload of a large online service provider (e.g., sending 10^5 data blocks simultaneously along 10^4 disjoint overlay paths).

We have implemented a prototype of BDS and integrated it in Baidu. We deployed BDS in 10 DCs and ran a pilot study on 500 TB of data transfer for 7 days (about 71 TB per day). Our real-world experiments show that BDS achieves 3-5$\times$ speedup over Baidu's existing solution named Gingko, and it can eliminate the incidents of excessive bandwidth consumption by bulk-data transfers. Using trace-driven simulation and micro-benchmarking, 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 followed:

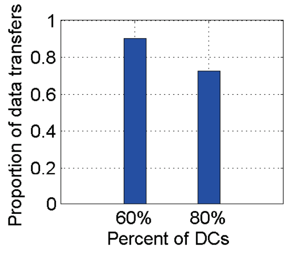
- Characterizing Baidu's workload of inter-DC bulk-data multicast to motivate the need of application-level multicast overlay networks (§2).
- Presenting BDS, an application-level multicast overlay network that achieves near-optimal flow completion time by a centralized control architecture (§3,4,5).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

2 A CASE FOR APPLICATION-LEVEL INTER-DC MULTICAST OVERLAYS

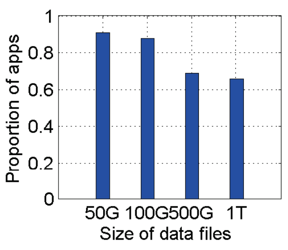
We start by providing a case for an application-level multicast overlay network. We first characterize the inter-DC multicast workload in Baidu, a global-scale online service provider (§2.1). We then show the opportunities of improving multicast performance by leveraging disjoint application-level overlay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

Type of application	% of multicast traffic
All applications	91.13% [1]
Blog articles	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) Proportion of multicast transfers destined to percent of DCs.



(b) Proportion 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.

2.1 Baidu's 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 all 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 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 share common destinations. Figure 2a sketches the distribution of the percentage of Baidu's DCs to which multicast transfers are destined. We see that 90% of multicast transfers are destined to at least 60% of the DCs, and 70% are destined to over 80% of the DCs. Moreover, we found a great diversity in the source DCs and the sets of destination DCs (not shown here). These observations suggest that it is untenable to pre-configure all possible multicast requests; instead, *we need a system to automatically route and schedule any given inter-DC multicast transfers.*

[1]The overall multicast traffic share is estimated using the traffic that 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 from other DCs.

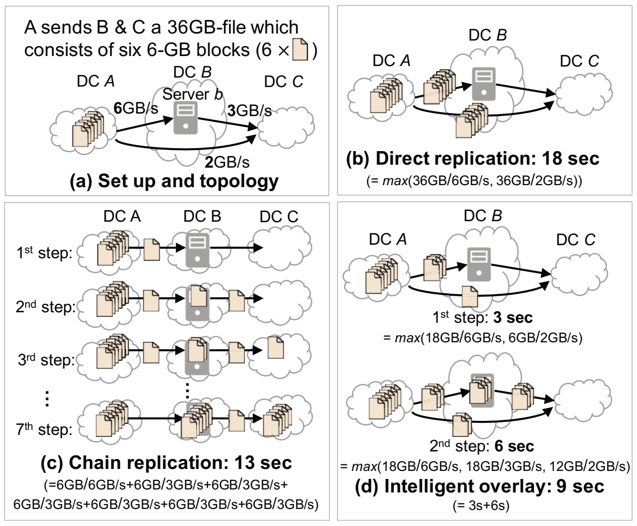

A sends B & C a 36GB-file which consists of six 6-GB blocks (6 ×)
DC A
DC B
Server b
DC C
6GB/s
3GB/s
2GB/s
(a) Set up and topology
(b) Direct replication: 18 sec
(= max(36GB/6GB/s, 36GB/2GB/s))
1st step:
2nd step:
3rd step:
7th step:
(c) Chain replication: 13 sec
(=6GB/6GB/s+6GB/3GB/s+6GB/3GB/s+6GB/3GB/s+6GB/3GB/s+6GB/3GB/s+6GB/3GB/s)
1st step: 3 sec
= max(18GB/6GB/s, 6GB/2GB/s)
2nd step: 6 sec
= max(18GB/6GB/s, 18GB/3GB/s, 12GB/2GB/s)
(d) Intelligent overlay: 9 sec
(= 3s+6s)

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).

Sizes of inter-DC multicast transfers: Finally, Figure 2b outlines the distribution of data size of inter-DC multicast. We see that for over 60% multicast transfers, the file sizes 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, these transfers are not transient but persistent, typically lasting for at least tens of seconds. Therefore, *any scheme that optimizes multicast traffic must dynamically adapt to any performance variation during a data transfer.* On the flip side, such temporal persistence also implies that *multicast traffic can tolerate a small amount of delay caused by a centralized control mechanism, such as BDS* (§3).

These observations together motivate the need for a systematic approach to optimizing inter-DC multicast performance.

2.2 Potentials of inter-DC application-level overlay

It is known that, generally, multicast can be delivered using application-level overlays [13]. Here, we show that inter-DC multicast completion time (defined by the time until each destination DC has a full copy of the data) can be greatly reduced by an application-level overlay network. Note that an application-level overlay does not require any network-level support, so it is complementary to prior work on WAN optimization.

The basic idea of an application-level overlay network is to distribute traffic along *bottleneck-disjoint* overlay paths [15], i.e., the two paths do not share a common bottleneck link or intermediate server. In the context of inter-DC transfers, two overlay paths either traverse different sequences of DCs

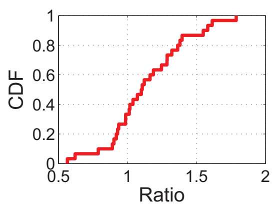


Figure 4: There is a significant performance variance among the inter-DC overlay paths in our network, indicating that most pairs of overlay paths are bottleneck disjoint.

(*Type I*), or traverse different sequences of servers of the same sequence of DCs (*Type II*), or some combination of the two. Next, we use examples to show bottleneck-disjoint overlay paths can arise in both types of overlay paths and how they improve inter-DC multicast performance.

Examples of bottleneck-disjoint overlay paths: In Figure 1, we have already seen how two Type I overlay paths ($A \to B \to C$ and $A \to C \to B$) are bottleneck-disjoint, and how it improves the performance of inter-DC multicast. Figure 3 shows an example of Type II bottleneck-disjoint overlay paths (traversing the same sequence of DCs but different sequence of servers). Suppose we need to replicate 36GB data from DC A to B and C via two bottleneck-disjoint paths: (1) $A \to C$: from A through B to C using IP-layer WAN routing with 2GB/s capacity, or (2) $A \to b \to C$: 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 six 6GB-blocks. We consider three strategies. (1) *Direct replication*: if A sends data directly to B and C via 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 in our network performance dataset, since it does not have per-hop bandwidth information of each multicast transfer. Instead, we observe that if two overlay paths have different end-to-end throughput at the same time, they should be bottleneck-disjoint. We show one example of bottleneck-disjoint overlay paths 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). If $\frac{BW_{A \to C}}{BW_{A \to b \to C}} \neq 1$, they are bottleneck-disjoint (BW_p denotes the throughput of path p). 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 more than 95% pairs of $A \to b \to C$ and $A \to C$ have different end-to-end throughput, i.e., they are bottleneck disjoint.

2.3 Limitations of existing solutions

Realizing and demonstrating the potential improvement of an application-level overlay network has some complications. As a first order approximation, we can simply borrow existing techniques from multicast overlay networks in other contexts. But the operational experience of Baidu shows two limitations of this approach that will be described below.

Existing solutions of Baidu: To meet the need of rapid growth of inter-DC data replication, Baidu has deployed Gingko, an application-level overlay network a few years ago. Despite years of refinement, Gingko is 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, 55]). The basic idea is that when multiple DCs request a data file from a 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 suffers from suboptimal scheduling and routing decisions. 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 (in the same magnitude of bandwidth assigned to each bulk-data transfer in production traffic). This 30GB file was evenly stored across all these 640 servers. Ideally, if the servers select the best source for all blocks, the 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 took 195 minutes (4.75× the optimal completion time) to receive data, and 5% of servers even waited for over 250 minutes. The key reason for this problem is that individual servers only see a subset of available data sources (i.e., servers who have already downloaded part of a file), and thus cannot leverage all available overlay paths to maximize the throughput. Such suboptimal performance could occur even if the overlay network is only partially decentralized (e.g., [23]), where even if each server does have a global view, local adaptations by individual servers would still create potential hotspots and congestion on overlay paths.

Limitation 2: Interaction 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

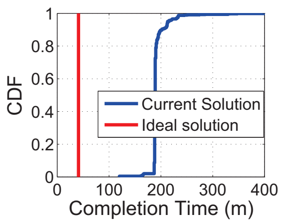


Figure 5: The CDF of the actual flow completion time at different servers in the destination DCs, compared with that of the ideal solution.

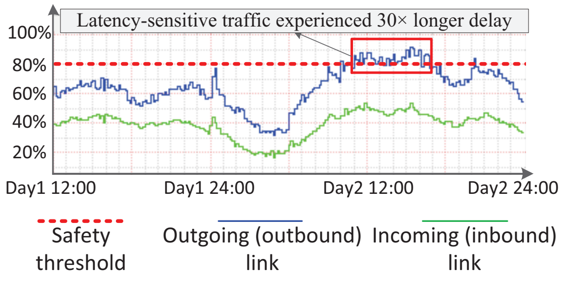


Figure 6: The utilization of the inter-DC link in two days. Inter-DC bulk data transfer on the 2nd day caused severe interference on latency-sensitive traffic.

bulk data transfers, we still see negative impacts on latency-sensitive traffic by bursty arrivals of bulk-data multicast requests. Figure 6 shows the bandwidth utilization of an inter-DC link in two days during which a 6-hour long bulk data transfer started at 11:00pm on the second day. The blue line denotes the outgoing bandwidth, and the green line denotes the incoming bandwidth. We can see that the bulk data transfer caused excessive link utilization (i.e., exceeding the safety threshold of 80%), and as a result, the latency-sensitive online traffic experienced over 30× delay inflation.

2.4 Key observations

The key observations from this section are following:

- *Inter-DC multicasts* amount to a substantial 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 rely on local adaptation can have *suboptimal performance* and *negative impact on online traffic*.

3 OVERVIEW OF BDS

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

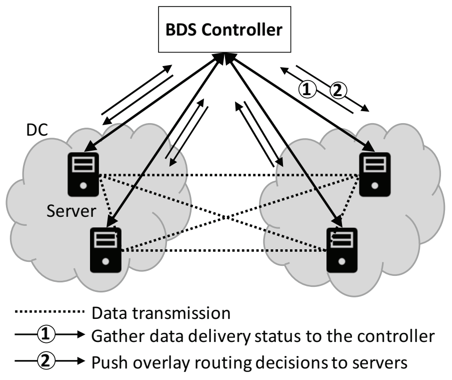

BDS Controller
DC
Server
①
②
Data transmission
Gather data delivery status to the controller
Push overlay routing decisions to servers

Figure 7: The centralized design of BDS.

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, 23, 35, 41]), despite the resulting suboptimal performance due to lack of global view or orchestration. In contrast, BDS takes an explicit stance that it is practical to fully centralize the control of wide-area overlay networks and still achieve near-optimal performance in the setting of inter-DC multicasts. The design of BDS coincides other recents works of centralizing management of large-scale distributed systems, e.g., [20] At a high level, BDS uses a centralized controller that periodically pulls information (e.g., data delivery status) 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 fall back to a decentralized control scheme to ensure graceful performance degradation to local adaptation (§5.3).

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

1. *Large decision space:* The sheer number of inter-DC overlay paths (which grow exponentially with more servers acting as overlay nodes) makes it difficult for individual servers to explore all available overlay paths based only on local measurements. In contrast, we could significantly improve overlay multicast performance by maintaining a global view of data delivery status of all servers, and dynamically balancing the availability of various data blocks, which turns out to be critical to achieving near-optimal performance (§4.3).
2. *Large data size:* Unlike latency-sensitive traffic which lasts on timescales of several to 10s of milliseconds, inter-DC multicasts last on much coarser timescales. Therefore, BDS can tolerate a short delay (of a few seconds) in order to get better routing decisions from a centralized

controller which maintains a global view of data delivery and is capable of orchestrating all overlay servers.

3. *Strict traffic isolation:* As observed in §2.3, it is vital that inter-DC multicasts avoid hotspots and excessive bandwidth usage that negatively impact the latency of delay-sensitive traffic, but it is difficult to prevent such situations without any coordination across overlay servers. In contrast, it is simpler to allocate bandwidth of each data transfer by controlling the sending rate at 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, in which the control logic running locally in each server can be stateless and triggered only on arrivals of new data units or control messages.

The key to realizing centralized control: In essence, the design of BDS performs a trade-off between incurring a small update delay in return for the near-optimal decisions brought by a centralized system. Thus, the key to striking such a favorable balance is a near-optimal yet efficient overlay routing algorithm that can update decisions in near realtime. At a first glance, this is indeed intractable. For the workload at a scale of Baidu, the centralized overlay routing algorithm must pick the next hops for 10^5 of data blocks from 10^4 servers. This operates at a scale that could grow exponentially when we consider the growth in the number of possible overlay paths that go through these servers and with finer grained block partitioning. With the standard routing formulation and linear programming solvers, it could be completely unrealistic to make near-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

The core of BDS is a centralized decision-making algorithm that periodically updates overlay routing decisions at scale in near real-time. 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 use to send each data block (§4.4), each of which can be solved efficiently and near-optimally.

4.1 Basic formulation

We begin by formulating the problem of overlay traffic engineering. Table 2 summarizes the key notations.

The overlay traffic engineering in BDS operates at a fine granularity, both spatially and temporally. To exploit the many overlay paths between the source and destination DCs, BDS split each data file into multiple data blocks (e.g., 2MB). To cope with changes of network conditions and arrivals of requests, BDS updates the decisions of overlay traffic engineering every ΔT (by default, 3 seconds[2].).

Variables	Meaning
$\mathbb{B}$	Set of blocks of all tasks
b	A block
$\rho(b)$	The size of block b
$\mathbb{P}_{s,s'}$	Set of paths between a source and destination pair
p	A particular path
l	A link on a path
$c(l)$	Capacity of link l
ΔT	A scheduling cycle
T_k	The k-th update cycle
$w_{b,s}^{(T_k)}$	Binary: if s is chosen as destination server for b at T_k
$R_{up}(s)$	Upload capacity of server s
$R_{down}(s)$	Download capacity of server s
$f_{b,p}^{(T_k)}$	Bandwidth allocated to send b on path p at T_k

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

Now, the problem of multicast overlay routing can be formulated as following:

Input: BDS takes as input the following parameters: the set of all data blocks $\mathbb{B}$, each block b with size $\rho(b)$, the set of paths from server s' to s, $\mathbb{P}_{s',s}$, the update cycle interval ΔT, and for each server s the upload (resp. download) capacity $R_{up}(s)$ (resp. $R_{down}(s)$). Note that each path p consists of several links l, each defined by a pair of servers or routers. We use $c(l)$ to denote the capacity of a link l.

Output: For each cycle T_k, block b, server s, and path $p \in \mathbb{P}_{s',s}$ destined to s, BDS returns as output a 2-tuple $\langle w_{b,s}^{(T_k)}, f_{b,p}^{(T_k)} \rangle$, in which $w_{b,s}^{(T_k)}$ denotes whether server s is selected as the destination server of block b in T_k, $f_{b,p}^{(T_k)}$ denotes how much bandwidth is allocated to send block b on path p in T_k, and $f_{b,p}^{(T_k)} = 0$ denotes path p is not selected to send block b in T_k.

Constraints:

- The allocated bandwidth on path p must not exceed the capacity of any link l in p, as well as the upload capacity of the source server $R_{up}(s)$, and the download capacity of the destination server $R_{down}(s')$.

$$f_{b,p}^{(T_k)} \leq min\left(min_{l \in p} c(l), q_{b,s'}^{(T_k)} \cdot R_{up}(s'), w_{b,s}^{(T_k)} \cdot R_{down}(s)\right) \quad \text{for } \forall b, p \in \mathbb{P}_{s',s} \tag{1}$$

where $q_{b,s}^{(T_k)} = 1 - \prod_{i<k}(1 - w_{b,s}^{(T_i)})$ denotes whether server s has ever been selected to be the destination of block b before cycle T_k.

[2]We use a fixed interval of 3 seconds, because it is long enough for BDS to update decisions at a scale of Baidu's workload, and short enough to adapt to typical performance churns without noticeable impact on the completion time of bulk data transfers. More details in §6

- For all the paths, the summed allocated bandwidth of a link should be no more than its capacity $c(l)$.

$$c(l) \geq \sum_{b \in \mathbb{B}} f_{b,p}^{(T_k)}, \text{for } \forall l \in p \tag{2}$$

- All blocks selected to be sent in each cycle must complete their transfers within ΔT, that is,

$$\sum_{b \in \mathbb{B}} w_{b,s}^{(T_k)} \cdot \rho(b) \leq \sum_{p \in \mathbb{P}} \sum_{b \in \mathbb{B}} f_{b,p}^{(T_k)} \cdot \Delta T, \text{for } \forall T_k \tag{3}$$

- Finally, all the blocks must be transmitted at the end of all cycles.

$$\sum_{b \in \mathbb{B}} \rho(b) \leq \sum_{k=1}^{N} \sum_{p \in \mathbb{P}} \sum_{b \in \mathbb{B}} f_{b,p}^{(T_k)} \tag{4}$$

Objective: We want to minimize the number of cycles needed to transfer all data blocks. That is, we return as output the minimum integer N for which the above constraints have a feasible solution.

Unfortunately, this formulation is intractable in practice for two reasons. First, it is super-linear and mixed-integer, so the computational overhead grows exponentially with more potential source servers, and data blocks. Second, to find the minimum integer N, we need to check the feasibility of the problem for different values of N.

4.2 Decoupling scheduling and routing

At a high level, the key insight behind BDS is to decouple the aforementioned formulation into two steps: a *scheduling* step which selects the subset of blocks to be transferred each cycle (i.e., $w_{b,s}^{(T_k)}$), followed by a subsequent *routing* step which picks the path and allocates bandwidth to transfer the selected blocks (i.e., $f_{b,p}^{(T_k)}$).

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 are considered in the subsequent routing step, the searching space is thus significantly reduced. Mathematically, by separating the scheduling step from the problem formulation, the routing step is reduced to a mixed-integer LP problem, which though is not immediately tractable, can be solved with standard techniques. Next, we present each step in more details.

4.3 Scheduling

The scheduling step selects the subset of blocks to be transferred in each cycle, i.e., $w_{b,s}^{(T_k)}$.

The key to do the scheduling (pick the subset of blocks) is to make sure that the subsequent data transmission can be done in the most efficient manner. Inspired by the "rarest-first" strategy in BitTorrent [14] that tries to balance block availability, BDS adopts a simple-yet-efficient way of selecting the data blocks: for each cycle, BDS simply picks the subset of blocks with the least amount of duplicates. In other words, BDS generalizes the rarest-first approach by selecting a set of blocks in each cycle, instead of a copy of a single block.

4.4 Routing

After the scheduling step selects the block set to transfer in each time slot ($w_{b,s}^{(T_k)}$), the routing step decides the paths and allocates bandwidth to transfer the selected blocks (i.e., $f_{b,p}^{(T_k)}$). To minimize the transfer completion time, BDS maximizes the throughput (total data volume transferred) in each cycle T_k.

$$max \sum_{p \in \mathbb{P}} \sum_{b \in \mathbb{B}} f_{b,p}^{(T_k)} \tag{5}$$

This is of course an approximation, since greedily maximixing the throughput in one cycle may lead to suboptimal data availability and lower the maximum achivable throughput in the next cycle. But in practice, we find that this approximation can lead to significant performance improvement over baselines, partly because the scheduling step, described in the last section, automatically balances the availability of blocks, so suboptimal data availability (e.g., starvation of blocks) caused by greedy routing decisions in past cycles happens rarely.

This formulation, together with the constraints from §4.1 is essentially an integer multi-commodity flow (MCF) 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].

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: (1) it groups the blocks with the same source and destination pair to reduce the problem size (detailed in §5.1); and (2) it uses the improved fully polynomial-time approximation schemes (FPTAS) [17] to optimize the dual problem of the original problem and works out an ε-optimal solution. These two strategies further reduces the running time of centralized algorithm.

5 SYSTEM DESIGN

This section presents the system design and implemetation of BDS.

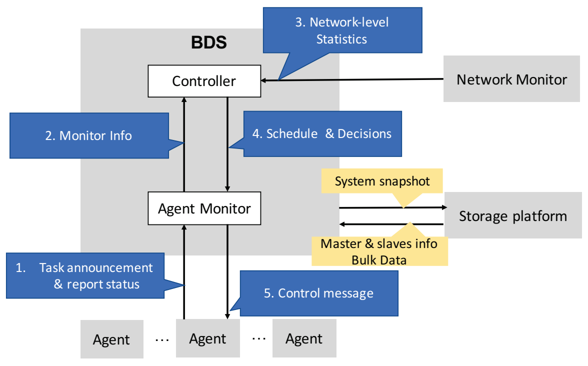


Figure 8: Interfaces of BDS's centralized control.

5.1 Centralized control of BDS

BDS periodically (by default, every three seconds) updates the routing and scheduling decisions in a centralized fashion. Figure 8 outlines the workflow in each three-second cycle.

(1) It starts with the *Agent*, running local on each server, checking the local states, including data block delivery status (which blocks have arrived, and which blocks are outstanding), server availability, and disk failures, etc.
(2) These statistics are then wrapped in a *control message*, and sent to the centralized *BDS Controller* via an efficient messaging layer called *Agent Monitor*.
(3) The BDS Controller also receives network-level statistics (the bandwidth consumption by latency-sensitive traffic and the utilization on each inter-DC link) from a *Network Monitor*.
(4) On receiving the updates from all Agents and the Network Monitor, the BDS Controller runs the centralized decision-making algorithm (§4) to work out the new scheduling and routing decisions, and sends the difference between the new decision and the previous one to the per-server local Agent via the Agent Monitor messaging layer.
(5) Finally, the Agent allocates bandwidth for each data transfer, and carries out the actual data transfers according the Controller's routing and scheduling decisions.

BDS uses two additional optimizations to make the workflow more efficient.

- *Blocks merging*. To reduce the computational scale and achieve more efficient transmissions, BDS merges the blocks with the same source and destination into one subtask. Its benefits are two-fold: (1) it significantly reduces the number of pending blocks in each scheduling cycle, thus reducing the computational cost of the centralized decision-making logic; and (2) it reduces the number of parallel TCP connections between servers, which could otherwise reduce link utilization and degraded performance.
- *Non-blocking update*. To avoid being blocked by the controller's decision-making logic, each local Agent keeps the ongoing data transmissions alive while the Controller runs the centralized decision-making logic. Similarly, the Controller takes this into account by speculating the changes in data delivery status while the decisions are being re-calculated, and using these speculated data delievery status as the input of the centralized logic.

5.2 Dynamic bandwidth separation

To guarantee clean separation of bandwidth between inter-DC bulk-data multicasts and delay-sensitive traffic, BDS Network Monitor monitors the aggregated bandwidth usage of all latency-sensitive flows on each inter-/intra-DC link, and dynamically allocates the bandwidth of each inter-DC multicast transfer. To protect delay-sensitive flows from being negatively affected by bursty bulk-data transfers, BDS uses 80% link utilization as a "safety threshold", i.e., the total bandwidth consumption of bulk-data transfers cannot exceed 80% of the link capacity on any link, and dynamically decides the sending rate of each data transfer.

The key advantage of BDS's dynamic bandwidth separation is that it efficiently allocates bandwidth in a centralized manner, similar [27] in the transport layer. The traditional techniques (e.g., [27]) that gives higher priority to online latency-sensitive traffic can still have bandwidth wastage or performance interference in the presence of dynamic network environments [49]. BDS, in contrast, dynamically monitors the aggregated bandwidth of delay-sensitive applications, and calculates the residual bandwidth to be used by inter-DC multicast while meeting the safety link utilization threshold. Finally, note that BDS optimizes the application-level overlay, and is thus complementary to network-layer techniques that improve the WAN performance and fairness [10, 25, 32, 40].

5.3 Fault tolerance

Next we describe how BDS handles the following failures.

1. *Controller failure:* The BDS controller is replicated [28]: if the master controller fails, another replica will be elected as the new controller. If all controller replicas are not available (e.g., a network partition between DCs and the controllers), 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 failure state (e.g., server crash, disk failure, etc.) to the agent monitor in the next cycle. 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.

5.4 Implementation and deployment

We have implemented BDS, and deployed on 67 servers in 10 of Baidu's geo-distributed DCs. Evaluation in the next section is based on this deployment. BDS is implemented with 3621 lines of golang code [1].

The duplications of the controller are implemented on three different geo-located zookeeper servers. The Agent Monitor uses `HTTP POST` to send controll messages between the controller and servers. BDS uses `wget` to make each data transfer, and enforce bandwidth allocation by setting `--limit-rate` field in each data transfer. 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 can be seamlessly integrated with any inter-DC communication patterns. All the applications need to do is to call the BDS APIs that consist of three steps: (1) provide BDS with the source DC, destination DCs, intermediate servers, and the pointer to the bulk data; (2) install agents on all intermediate servers; (3) and finally, set the start time of the data transfers. Then BDS will start the data distribution at the specified time. We speculate that our implementation should be applicable to other companies' DCs too.

BDS has several parameters that are set either by adminitrators of Baidu, or empirically by evaluation results. These parameters include: the bandwidth reserved for latency-sensitive traffic (20%), the data block size (2MB), and update cycle length (3 seconds).

6 EVALUATION

Using a combination of pilot deployment in Baidu's DCs, trace-driven simulation, and microbenchmarking, we show that:

1. BDS completes inter-DC multicast 3-5× faster than Baidu's existing solutions, as well as 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 achieves close to optimal flow completion time (§6.3).

6.1 Performance improvement over existing solutions

6.1.1 Methodology. **Baselines:** We compare BDS with three existing solutions: Gingko (Baidu's existing decentralized inter-DC multi-cast strategy), Bullet [26], and Akamai's overlay network [9] (a centralized strategy for multicasting live videos).

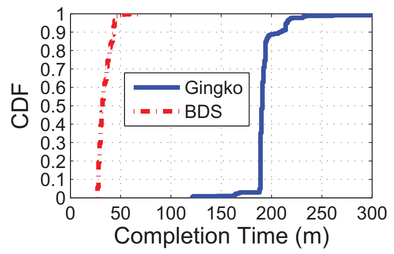
Pilot deployment: We choose several services with different data sizes, and run A/B testing in which we run BDS instead of Baidu's default solution Gingko for the same hours in several randomly chosen days.

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 using the same topology, number of servers, and link capacities as BDS, and replay inter-DC multicast data requests in the same chronological order as in the pilot deployment.

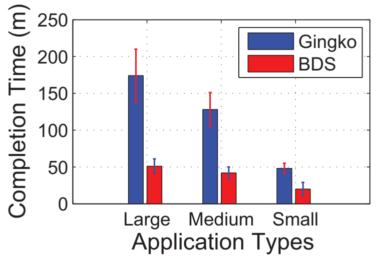
6.1.2 BDS vs. Gingko. We begin by evaluating BDS and Gingko on one service that needs to distribute 70 TB data from one source DC to ten destination DCs. Figure 9a shows the cumulative distribution function (CDF) of the completion time on each destination server. We can see that the median completion time of BDS is 35 minutes, 5× faster than Gingko, where most DCs takes 190 minutes to get the data.

To generalize the finding, we pick three applications whose data volumes are large, medium and small, and compare BDS's and Gingko's mean (and standard deviation) of completion time for each application in Figure 9b. We see that BDS consistently outperforms Gingko, and has less performance variance. We also see that BDS has greater improvement in applications with larger data sizes. Finally, Figure 9c shows the timeseries of the mean completion time of BDS and Gingko in one randomly chosen application, and we see that BDS consistently outperforms Gingko by 4×.

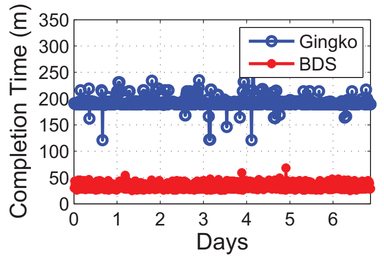
6.1.3 BDS vs. other overlay multicast techniques. Table 3 compares BDS with two other baselines, Bullet and Akamai's overlay network, using trace-driven simulation. We show the results in three setups. In the baseline evaluations, we send 10TB data from one DC to 11 DCs, each has 100 servers, and the upload and download link capacities are set to be 20MBs. In the large-scale evaluations, we send 100TB data between the same DCs, each with 1000 servers. In the rate-limited evaluations, the setup is the same as that in the baseline experiments except the server upload and download rate limit set to be 5MBs. We see that BDS achieves 3× shorter completion time than Bullet and Akamai in the baseline setup, and over 4× shorter completion time in the large-scale and small bandwidth setups, which corroborates



(a) Distribution of completion time.



(b) Comparison by application types.



(c) Comparison by completion time.

Figure 9: [BDS vs. Gingko (Baidu's existing solution)] 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 [26], Akamai [9]] Completion time of the three solutions in trace-driven simulation.

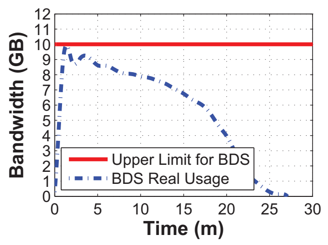


Figure 10: The effectiveness of bandwidth separation.

the findings in §6.1.2 that BDS has greater improvement when data sizes are large.

6.2 Benefits of coordinated bandwidth allocation

To reduce the incidences of negative interference between bulk-data multicast traffic and latency-sensitive traffic, Baidu sets a bandwidth limit of bulk data transfers on each link by the difference between the link capacity and the historical peak bandwidth of latency-sensitive traffic. Naturally, we can minimize the conflicts, if BDS can keep the bulk-data multicast traffic within the bandwidth limit. To test BDS's effectiveness at maintaining differentiation between bulk-data multicast traffic and latency-sensitive traffic, we set an 10GB/s bandwidth limit to bulk data transfers. Figure 10 shows the actual bandwidth usage of BDS on one inter-DC link. We can see that in BDS the actual bandwidth used by bulk data is always below 10 GB/s.

6.3 Micro-benchmarks

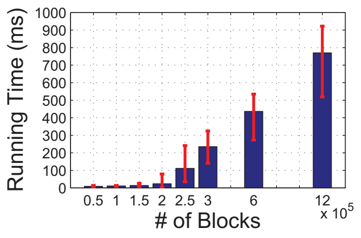
Next, we use micro-benchmarking to evaluate BDS along three metrics: (1) scalability of the centralized control; (2) fault tolerance; and (3) optimality of BDS parameters.

6.3.1 Scalability. **Controller running time:** As the controller needs to decide the scheduling and routing of each data block, the running time of the control logic naturally scales with the number of blocks. Figure 11a shows the running time as a function of the total number of blocks. We can see that the centralized BDS controller can update the scheduling and routing decision within 800ms with 10^6 blocks. To put this number into perspective, in Baidu's DCs, the maximum number of simultaneous outstanding data blocks is around 3×10^5, for which BDS can finish updating the decisions within 300ms.

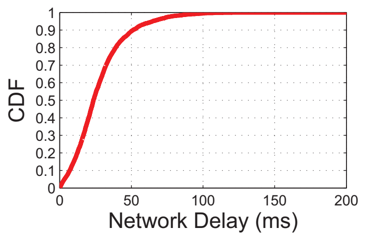
Network delay: BDS works in inter-DC networks, so the network delay among DCs is a key factor in the algorithm updating process. We recorded the network delay of 5000 requests and present the CDF in Figure 11b. We can see that 90% of the network delays are below $50ms$ and the average value is about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

Feedback loop delay: For centralized algorithms, a small feedback loop delay is essential for algorithmic 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, as shown in the CDF of Figure 11c, and find that in most cases (over 80%), the feedback loop delay is lower than $200ms$. So we claim that BDS demonstrates a short enough latency and is able to scale to even larger systems.

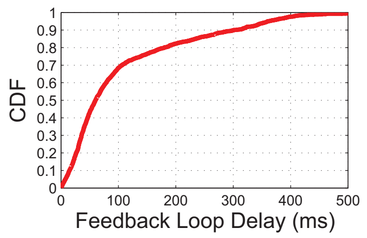
6.3.2 Fault tolerance. Here we examine the impact of the following failure scenario on the number of downloaded blocks per cycle. During cycles 0 to 9, BDS works as usual, and one agent fails in the 10th cycle. The controller fails in the 20th cycle and recovers in the 30th cycle. Figure 12a shows the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



(a) The controller running time.

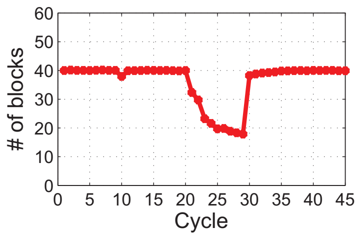


(b) The inter-DC network delay.

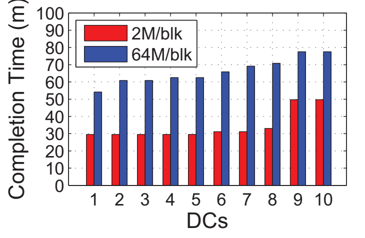


(c) Feedback loop delay.

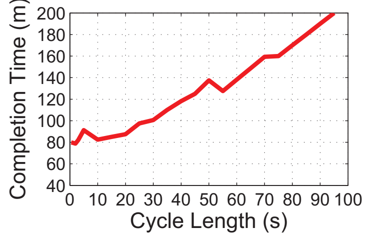
Figure 11: [System scalability] Measurements on (a) controller running time, (b) network delay, (c) Feedback loop delay.



(a) Average number of downloaded blocks per cycle under failures.

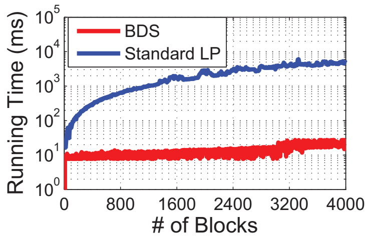


(b) Completion time under different block sizes.

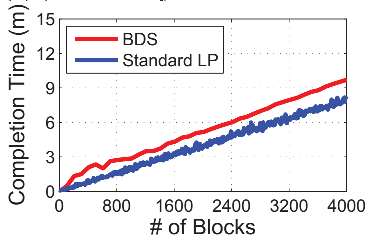


(c) Completion time under different cycle lengths.

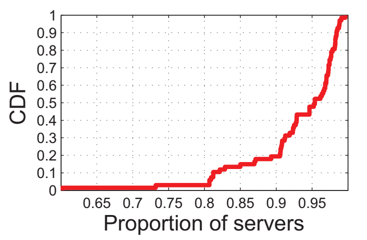
Figure 12: BDS's (a) fault tolerance, (b) sensitivity to different block sizes, and (c) different cycle lengths.



(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.

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. However, this optimization introduces a longer controller running time, as shown in Figure 11a. We pick block size by balancing two considerations: (1) constraints on the completion time, and (2) the controller's operational 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 test the adjustment frequency, we set different cycle lengths from 0.5s to 95s for 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 updating too frequently introduces greater 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 connections. 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 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 an 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 cannot finish in a reasonable time. Figure 13b shows the near-optimality of BDS.

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% of servers, the proportion is less than 20%, which means that more than 80% blocks are downloaded from other DCs on the disjoint paths, demonstrating the great potential of a multicast overlay network.

In summary, both the prototype pilot deployment and the trace-driven simulations of BDS show 3-5$\times$ 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 networks release great potential to various applications, especially for data transfer applications. The representative networks include Peer-to-Peer (P2P) networks and Content Delivery Networks (CDNs). The P2P architecture has already been verified by many applications, such as live streaming systems (CoolStreaming [55], Joost [2], PPStream [4], UUSee [6]), video-on-demand (VoD) applications (OceanStore [3]), distributed hash tables [42] and more recently Bitcoin [16]. But, self-organizing systems based on P2P principles suffer from long convergence times. 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 [59] and live streaming [47].

We briefly introduce the two baselines in the evaluation section: (1) Bullet [26], which enables geo-distributed nodes to self-organize into an 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 its 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 successfully explores a larger search space through a finer grained allocation without the limitation of three coarse grained layers. Second, the receiving sequence of data must be sequential in Akamai because it is designed for a live streaming application. But there is no such requirements in BDS, and the side effect is that BDS has to decide the optimal transmission order as additional work.

Data Transfer and Rate Control. Rate control of transport protocols at the DC-level plays an important role in data transmission. DCTCP [8], PDQ [21], CONGA [7], DCQCN [60] and TIMELY [33] are all classical protocols showing clear improvements in transmission efficiency. Some congestion control protocols like the credit-based ExpressPass [11] and load balancing protocols like Hermes [53] could further reduce flow completion time by improving rate control. On this basis, the recent proposed Numfabric [36] 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 [12, 52] has some similarities to the multicast overlay scheduling, in terms of data parallelism. But that work focuses on flow-level problems while BDS is designed at the application-level.

Centralized Traffic Engineering. Traffic engineering (TE) has long been a hot research topic, and many existing studies [10, 25, 32, 40, 45, 50, 54] have illustrated the challenges of scalability, heterogeneity etc., especially on inter-DC level. The representative TE systems include Google's B4 [24] and Microsoft's SWAN [22]. B4 adopts SDN [30] and OpenFlow [31, 38] 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 for data transfer in inter-DC WANs. Applications like user logs, search engine indexes and databases would greatly benefit from bulk-data multicast. Furthermore, such benefits are orthogonal to prior WAN optimizations, further improving inter-DC application 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 inter-DC bulk-data 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 there are significant benefits achieved by making slightly delayed decisions based on a global view and that this outweighs the cost of centralization. We believe that the insight of decoupling scheduling and routing, to speed up the execution of a centralized algorithm can be generalized to inspire other centralized control platforms where centralized decision-making strikes a favorable balance between costs and benefits.

ACKNOWLEDGMENTS

This work is supported in part by the HK RGC ECS-26200014, CRF-C703615G, HKUST PDF fund, the China 973 Program (2014CB340300), the National Natural Foundation of China (61472212), EU Marie Curie Actions CROWN (FP7-PEOPLE-2013-IRSES-610524). We would like to thank our shepherd, Paolo Romano, and the anonymous EuroSys reviewers for their valuable feedback.

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 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), pp. 63–74.
[9] Andreev, K., Maggs, B. M., Meyerson, A., and Sitaraman, R. K. Designing Overlay Multicast Networks For Streaming. *SPAA* (2013), 149–158.
[10] Chen, Y., Alspaugh, S., and Katz, R. H. Design insights for MapReduce from diverse production workloads. Tech. rep., CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE, 2012.
[11] Cho, I., Jang, K. H., and Han, D. Credit-Scheduled Delay-Bounded Congestion Control for Datacenters. In *ACM SIGCOMM* (2017), pp. 239–252.
[12] Chowdhury, MosharafStoica, and Eecs, I. Coflow: An Application Layer Abstraction for Cluster Networking. *Hotnets* (2012).
[13] Chu, Y.-h., Rao, S. G., and Zhang, H. A case for end system multicast. In *ACM SIGMETRICS* (2000), vol. 28, ACM, pp. 1–12.
[14] Cohen, B. Incentives build robustness in bittorrent. *Proc P Economics Workshop* (2003), 1–1.
[15] Datta, A. K., and Sen, R. K. 1-approximation algorithm for bottleneck disjoint path matching. *Information processing letters 55*, 1 (1995), 41–44.
[16] Eyal, I., Gencer, A. E., Sirer, E. G., and Van Renesse, R. Bitcoin-NG: A scalable blockchain protocol. In *NSDI* (2016).
[17] Fleischer, L. K. Approximating fractional multicommodity flow independent of the number of commodities. *SIDMA* (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] Gog, I., Schwarzkopf, M., Gleave, A., Watson, R. N. M., and Hand, S. Firmament: Fast, Centralized Cluster Scheduling at Scale. In *OSDI* (Savannah, GA, 2016), USENIX Association, pp. 99–115.
[21] Hong, C. Y., Caesar, M., and Godfrey, P. B. Finishing flows quickly with preemptive scheduling. pp. 127–138.
[22] 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* (2013), pp. 15–26.
[23] 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. *SIGCOMM* (2014), 187–198.
[24] 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* (2013), vol. 43, pp. 3–14.
[25] Kavulya, S., Tan, J., Gandhi, R., and Narasimhan, P. An analysis of traces from a production mapreduce cluster. In *CCGrid* (2010), IEEE, pp. 94–103.
[26] Kostić, D., Rodriguez, A., Albrecht, J., and Vahdat, A. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SOSP* (2003), vol. 37, ACM, pp. 282–297.
[27] 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 *ACM SIGCOMM* (2015), pp. 1–14.
[28] Lamport, L. The part-time parliament. *ACM TOCS 16*, 2 (1998), 133–169.
[29] Liebeherr, J., Nahas, M., and Si, W. Application-layer multicasting with Delaunay triangulation overlays. *IEEE JSAC 200*, 8 (2002), 1472–1488.
[30] McKeown, N. Software-defined networking. *INFOCOM keynote talk 17*, 2 (2009), 30–32.
[31] 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 38*, 2 (2008), 69–74.
[32] Mishra, A. K., Hellerstein, J. L., Cirne, W., and Das, C. R. Towards characterizing cloud backend workloads: insights from google compute clusters. *ACM SIGMETRICS PER 37*, 4 (2010), 34–41.
[33] 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 SIGCOMM* (2015), pp. 537–550.
[34] Mokhtarian, K., and Jacobsen, H. A. Minimum-delay multicast algorithms for mesh overlays. *IEEE/ACM TON 23*, 3 (2015), 973–986.
[35] 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* (2014), vol. 44, ACM, pp. 343–344.

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

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

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

[39] REED, M. J. Traffic engineering for information-centric networks. In *IEEE ICC* (2012), 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 *SoCC* (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* (2005), vol. 35, pp. 73–84.

[43] RODRIGUEZ, A., ALBRECHT, J., BHIRUD, A., AND VAHDAT, A. Using random subsets to build scalable network services. In *USITS* (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. In *ACM SIGCOMM* (1999), vol. 29, 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 *SoCC* (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 *ACM SIGCOMM* (2016), pp. 15–28.

[47] SRIPANIDKULCHAI, K., MAGGS, B., AND ZHANG, H. An analysis of live streaming workloads on the internet. In *IMC* (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 *ICDCS* (2007), p. 49.

[49] WANG, H., LI, T., SHEA, R., MA, X., WANG, F., LIU, J., AND XU, K. Toward cloud-based distributed interactive applications: Measurement, modeling, and analysis. *IEEE/ACM ToN* (2017).

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

[51] ZHANG, H., CHEN, K., BAI, W., HAN, D., TIAN, C., WANG, H., GUAN, H., AND ZHANG, M. Guaranteeing deadlines for inter-datacenter transfers. In *EuroSys* (2015), ACM, p. 20.

[52] ZHANG, H., CHEN, L., YI, B., CHEN, K., GENG, Y., AND GENG, Y. CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark. In *ACM SIGCOMM* (2016), pp. 160–173.

[53] ZHANG, H., ZHANG, J., BAI, W., CHEN, K., AND CHOWDHURY, M. Resilient Datacenter Load Balancing in the Wild. In *ACM SIGCOMM* (2017), pp. 253–266.

[54] ZHANG, Q., HELLERSTEIN, J. L., AND BOUTABA, R. Characterizing task usage shapes in google's compute clusters. In *LADIS* (2011).

[55] 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 *INFOCOM* (2005), vol. 3, IEEE, pp. 2102–2111.

[56] ZHANG, Y., LI, Y., XU, K., WANG, D., LI, M., CAO, X., AND LIANG, Q. A communication-aware container re-distribution approach for high performance vnfs. In *IEEE ICDCS 2017* (2017), IEEE, pp. 1555–1564.

[57] ZHANG, Y., XU, K., WANG, H., LI, Q., LI, T., AND CAO, X. Going fast and fair: Latency optimization for cloud-based service chains. *IEEE Network* (2017).

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

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

[60] 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 45*, 5 (2015), 523–536.

9 APPENDIX

Suppose we want to send N data blocks to m destination DCs. Without loss of generality, we consider two cases:

- **A (Balanced):** Each of the N blocks has k duplicas;
- **B (Imbalanced):** Half blocks have k_1 duplicas each, and the other half have k_2 duplicas each, and $k_1 < k_2, (k_1 + k_2)/2 = k$.

Note that $m > k$, since otherwise, the multicast is already complete. Next, we prove that in a simplified setting, BDS's completion time in A is strictly less than B.

To simplify the calculation of BDS's completion time, we now make a few assumptions (which are not critical to our conclusion): (1) all servers have the same upload (resp. download) bandwidth R_{up} (resp. R_{down}), (2) no two duplicas share the same source (resp. destination) server, so the upload (resp. download) bandwidth of each block is R_{up} (resp. R_{down}). Now we can write the completion time in the two cases as following:

$$\begin{aligned} t_A &= \frac{V}{min\{c(l), \frac{kR_{up}}{m-k}, \frac{kR_{down}}{m-k}\}} \\ t_B &= \frac{V}{min\{c(l), \frac{k_1R_{up}}{m-k_1}, \frac{k_2R_{up}}{m-k_2}, \frac{k_1R_{down}}{m-k_1}, \frac{k_2R_{down}}{m-k_2}\}} \end{aligned} \tag{6}$$

where V denotes the total size of the untransmitted blocks, $V = N(m-k)\rho(b) = \frac{N}{2}(m-k_1)\rho(b) + \frac{N}{2}(m-k_2)\rho(b)$. In the production system of Baidu, the inter-DC link capacity $c(l)$ is several orders of magnitudes higher than upload/download capacity of a single server, so we can safely exclude $c(l)$ from the denominator in the equations. Finally, if we denote $min\{R_{up}, R_{down}\} = R$, then $t_A = \frac{(m-k)V}{kR}$ and $t_B = \frac{(m-k_1)V}{k_1R}$.

We can show that $\frac{(m-k)V}{kR}$ is a monotonically decreasing function of k:

$$\frac{d}{dk}\frac{(m-k)V}{kR} = \frac{d}{dk}\frac{(m-k)^2N\rho(b)}{kR} = \frac{N\rho(b)}{R}(1 - \frac{m^2}{k^2}) < 0 \tag{7}$$

Now, since $k > k_1$, we have $t_A < t_B$.