BDS+: An Inter-Datacenter Data Replication System with Dynamic Bandwidth Separation

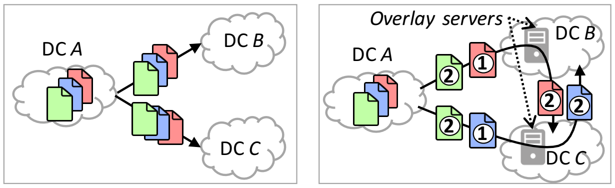
Yuchao Zhang*✉, *Member, IEEE,* Xiaohui Nie*, *Member, IEEE,* Junchen Jiang, *Member, IEEE,*
Wendong Wang, *Senior Member, IEEE,* Ke Xu, *Senior Member, IEEE,*
Youjian Zhao, *Senior Member, IEEE,* Martin J. Reed, *Member, IEEE,*
Kai Chen, *Senior Member, IEEE,* Haiyang Wang, *Member, IEEE,* and Guang Yao

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 rich inter-DC overlay paths that exist in geo-distributed DCs, as well as the remaining bandwidth reserved for online traffic under fixed bandwidth separation scheme. To take advantage of these opportunities, we present *BDS+*, a near-optimal network system for large-scale inter-DC data replication. BDS+ is an application-level multicast overlay network with 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. Furthermore, in each overlay path, it leverages dynamic bandwidth separation to make use of the remaining available bandwidth reserved for online traffic. By constantly estimating online traffic demand and rescheduling bulk-data transfers accordingly, BDS+ can further speed up the massive data multicast. Through a pilot deployment in one of the largest online service providers and large-scale real-trace simulations, we show that BDS+ can achieve 3-5× speedup over the provider's existing system and several well-known overlay routing baselines of static bandwidth separation. Moreover, dynamic bandwidth separation can further reduce the completion time of bulk data transfers by 1.2 to 1.3 times.

Index Terms—Overlay Network, Data Replication, Centralized Control, Dynamic Bandwidth Separation.

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 [2], [3]. 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

Fig. 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*→*B*→*C* and *A* →*C*→*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., [4], [5], [2], [6], [7], [8]), 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

- *Yuchao Zhang and Wendong Wang are with the School of Computer Science (National Pilot Software Engineering School) at the Beijing University of Posts and Telecommunications. Wendong Wang is also with the State Key Laboratory of Networking and Switching Technology at the Beijing University of Posts and Telecommunications. Beijing, China. E-mail: yczhang@bupt.edu.cn, wdwang@bupt.edu.cn*
- *Xiaohui Nie, Ke Xu and Youjian Zhao are with Tsinghua University, Beijing, China. Ke Xu is also with BNRist and Peng Cheng Laboratory. E-mail: niexiaohui@mail.tsinghua.edu.cn, xuke@tsinghua.edu.cn, zhaoyoujian@tsinghua.edu.cn*
- *Junchen Jiang is with the Department of Computer Science at The University of Chicago, Chicago, US. E-mail: junchenj@uchicago.edu*
- *Martin J. Reed is with the School of Computer Science and Electronic Engineering at the University of Essex, Colchester, UK. E-mail: mjreed@essex.ac.uk*
- *Kai Chen is with Hong Kong University of Science and Technology, Hong Kong, China. E-mail: kaichen@cse.ust.hk*
- *Haiyang Wang is with the Department of Computer Science at the University of Minnesota at Duluth, US. E-mail: haiyang@d.umn.edu*
- *Guang Yao is with Baidu Company, Beijing, China. E-mail: yaoguang@baidu.com*

This paper significantly extends BDS [1] by adding dynamic bandwidth separation.
**Yuchao Zhang and Xiaohui Nie contribute equally to this paper.*
✉*Corresponding Author: Yuchao Zhang, yczhang@bupt.edu.cn*

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 first introduces *BDS+*, an *application-level centralized near-optimal network system*, which splits data into fine-grained units, and sends them in parallel via bottleneck-disjoint overlay paths with dynamic bandwidth sharing. 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], [10], [11], [12]), 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 [13], [14], [15], which leads to suboptimal decisions for lack of global information, or restricts itself to strictly structured (e.g., layered) topologies [16], 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 [17], 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).

In practice, there is always a fixed upper bound of available bandwidth for bulk-data multicast, because multicast overlay network shares the same inter-DC WAN with online latency-sensitive traffic. Existing solutions always reserve a fixed amount of bandwidth for the latency-sensitive traffic, according to its peak value. This guarantees the strict bandwidth separation, but the side affect is the waste of bandwidth, especially when the online traffic is in its valley. To further improve link utilization, BDS+ implements dynamic bandwidth separation that can predict online traffic and reschedule bulk-data transfer. In other words, BDS+ achieves dynamic bandwidth separation between bulk-data multicast and online traffic to further speed up data transfer.

We have implemented a prototype and integrated it in Baidu. We first 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× speedup over Baidu's existing solution named Gingko, and it can eliminate the incidents of excessive bandwidth consumption by bulk-data transfers. Using micro-benchmarking, we 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 [1]. We then use trace-driven simulations to evaluate BDS+ with dynamic bandwidth separation, the results show that: BDS+ further speeds up the bulk data transfer by 1.2 to 1.3 times in the network where online and offline services are mixed deployed. Our contributions are summarized as followed:

- Present the characteristics of Baidu's workload of inter-DC bulk-data multicast, which motivates 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).
- Making dynamic bandwidth separation to further improve link utilization in the network where online and offline services are mixed deployed. (§3,5).
- Demonstrating the practical benefits of BDS+ by a real-world pilot deployment and large-scale simulations in Baidu (§6,7).

2 Motivation of BDS+ design

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, and by leveraging dynamic bandwidth separation (§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). We conclude all these observations, which 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 (§2.4).

2.1 Baidu's inter-DC multicast workload

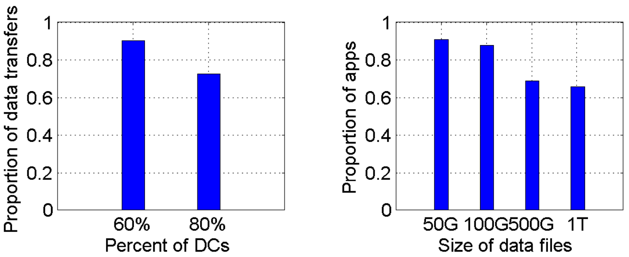
Share of inter-DC multicast traffic: Table 1 shows inter-DC multicast (replicating data from one DC to multiple DCs) as a fraction of all inter-DC traffic [2]. We see that inter-DC multicast dominates Baidu's overall inter-DC traffic (91.13%), as well as the traffic of individual application types (89.2 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.*

[1] As the existing solutions are with fixed bandwidth separation, so in these series of experiments, we use BDS+ without dynamic bandwidth separation (named BDS) as comparation, while BDS+ with dynamic bandwidth separation is evaluated separately.

[2] 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.

Type of application	% of multicast traffic
All applications	91.13%
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.

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

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.*

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 [18]. 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.

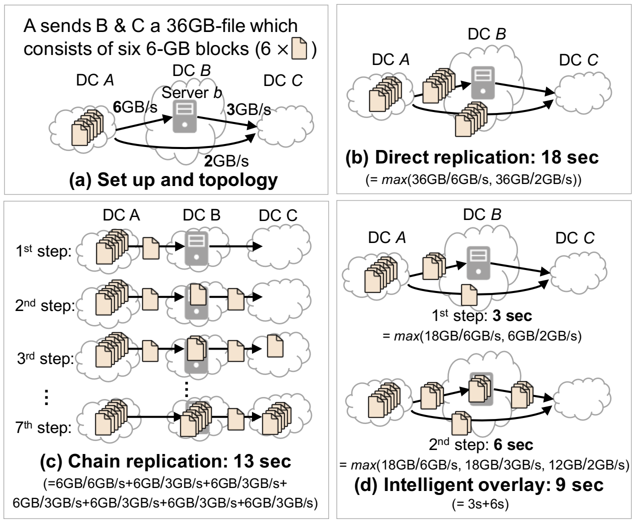

A sends B & C a 36GB-file which consists of six 6-GB blocks (6 ×)
DC A
DC B
Server b
DC C
6GB/s
3GB/s
2GB/s
(a) Set up and topology
DC A
DC B
DC C
(b) Direct replication: 18 sec
(= max(36GB/6GB/s, 36GB/2GB/s))
DC A
DC B
DC C
1st step:
2nd step:
3rd step:
7th step:
(c) Chain replication: 13 sec
(=6GB/6GB/s+6GB/3GB/s+6GB/3GB/s+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

Fig. 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).**

The basic idea of an application-level overlay network is to distribute traffic along *bottleneck-disjoint* overlay paths [19], 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 (*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

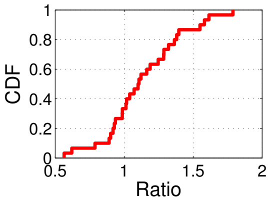


Fig. 4: **There is a significant performance variance among the inter-DC overlay paths in our network, indicating that most pairs of overlay paths are bottleneck disjoint.**

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.

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 bulk data transfers, we still see negative impacts on latency-sensitive traffic by bursty arrivals of bulk-data multicast requests, and inefficiency on bulk-data transfer when latency-sensitive traffic is in its valley. Figure 5 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. Also, at 4:00-5:00am in the first day, near 50% of the bandwidth was being wasted. These cases show that, an algorithm with dynamical interactions with latency-sensitive traffic would be more reasonable and efficient.

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 [11], [20], [21]). 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 $\frac{30\times1024}{640\times20Mbps\times60s/min} = 41$ minutes. But as shown in Figure 6, 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., [15]), 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.

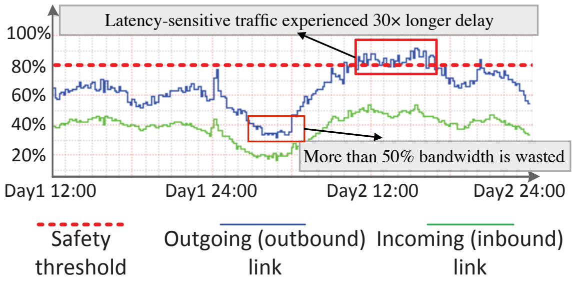


Fig. 5: **The utilization of the inter-DC link in two days: The traffic valley on the 1st day results in nearly 50% bandwidth waste. Inter-DC bulk data transfer on the 2nd day caused severe interference on latency-sensitive traffic.**

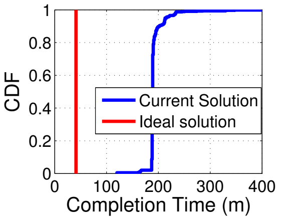


Fig. 6: **The CDF of the actual flow completion time at different servers in the destination DCs, compared with that of the ideal solution.**

Limitation 2: High computation overhead. To obtain a global view and achieve optimal scheduling protocols, existing centralized protocols suffer from high computation overhead. Most formulations are super-linear, so the computational overhead of centralized protocols always grows exponentially, making them intractable in practice.

Limitation 3: Fixed bandwidth separation. As shown in Figure 5, a fixed separation of link bandwidth would result in both excessive utilization and underutilization. Ideally, if we can make full use of the available bandwidth left by online traffic in real time, then the link utilization would be more stable. In this particular example, about 18.75% bandwidth was wasted in those two days (while still caused excessive utilization case).

2.4 Key observations

The key observations from this section are following:

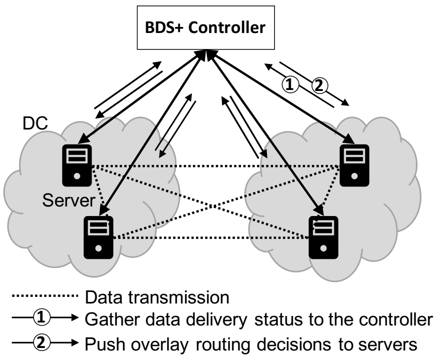

BDS+ Controller
DC
Server
① ②
Data transmission
① Gather data delivery status to the controller
② Push overlay routing decisions to servers

Fig. 7: **The centralized design of BDS+.**

- *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*.
- *Dynamic bandwidth separation* can be helpful to improve link utilization by making full use of the remaining bandwidth of online services.

3 System Overview

To optimize inter-DC multicasts on overlay network with dynamical separation with latency-sensitive traffic, we present *BDS+*, a *fully centralized* near-optimal network system with dynamic bandwidth separation for data inter-DC multicast. Before presenting the details, we first highlight the intuitions behind the design choices, and the challenges behind its realisation.

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., [11], [14], [15], [22]), 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 with other recent works that centralize the management of large-scale distributed systems, e.g., [23]. 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, the system will fall back to a decentralized control scheme to ensure graceful performance degradation to local adaptation (§6.3).

Our centralized design is driven by several empirical observations:

1. *Large decision space:* The sheer number of inter-DC overlay paths (which grow exponentially with the increasing number 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. *Flexible traffic control:* BDS+ can enforce bandwidth allocation by setting limit rates in each data transfer, while each server can use Linux Traffic Control (tc) to enforce the limit on the teal bandwidth usage. This allows BDS+ to leverage flexible dynamic bandwidth separation. Once any network changes are detected, BDS+ could easily adjust bandwidth for each data transfer by controlling the sending rate at all servers in a centralized fashion (no matter to reserve more bandwidth when online traffic burst, or to reduce transfer rate when online traffic is in valley). (§6.4).

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 (§7.2.4).

The key to realizing dynamic bandwidth separation: Dynamic bandwidth separation raises two requirements, one is to reserve enough bandwidth for latency-sensitive online traffic so as to avoid negative impacts on these services, and the other is to make full use of the residual bandwidth so as to reduce the completion time of bulk data transfer. With the traditional strict safety threshold and decentralized protocols, it could be impossible to make efficient bandwidth usage in the dynamic and mixed deployed network (§7.3).

The potential under different topologies: The potential of BDS+ comes from two aspects, one is the dynamic bandwidth separation, and the other is the bottleneck disjoint overlay path. While dynamic bandwidth separation could improve link utilization no matter in any network topology, the overlay path brings performance improvement under most network topologies, but not all. The premise for this algorithm to work is that there are bottleneck disjoint overlay paths under such network topology. For example, on the topologies such as *ring, star, tiered, full mesh, partial mesh*, BDS+ could work well because there are potential bottleneck disjoint overlay paths under such topologies. However, under the simple point-to-point network topology, BDS+ could not bring performance improvement because there are no alternative paths. Generally speaking, the topology of large-scale

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.

inter-DC network is more complex than the point-to-point network topology, so BDS+ could bring performance improvement in most cases.

The following two sections will present how BDS+ works.

4 Near-Optimal Application-level Overlay Network

The core of BDS+ is a centralized decision-making algorithm that periodically updates overlay routing decisions at scale and 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+ splits 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[3].).

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 $R_{down}(s')$.

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

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.

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

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

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

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

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

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

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 the increase in 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 solving the scheduling (picking the subset of blocks) is to make sure that the subsequent data transmission can be done

[3] 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 §7

in the most efficient manner. Inspired by the "rarest-first" strategy in BitTorrent [24] 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. The proof of optimality of this algorithm is shown in the Appendix.

In addition, BDS+ also supports setting different priorities for different blocks (i.e., applications) if necessary. For example, we can set higher priority for those blocks from more important (or shorter) applications, making those blocks selected for transmission as early as possible, even if they are not the rarest ones. But in the current version of BDS+, it treats them equally.

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 maximizing 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 [25]. 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 [26], [27].

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 §6.1); and (2) it uses the improved fully polynomial-time approximation schemes (FPTAS) [28] 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 Dynamic Bandwidth Separation

The primary version without dynamic bandwidth separation (BDS) performs well under fixed network separation, but in the mixed deployment situations where online traffic and offline traffic shares the same server I/O, it results in low link utilizations when online traffic reduces. This is because bulk data transfer will never occupy any bandwidth exceeding the fixed threshold even though online traffic is far below the reserved bandwidth (see §2.3).

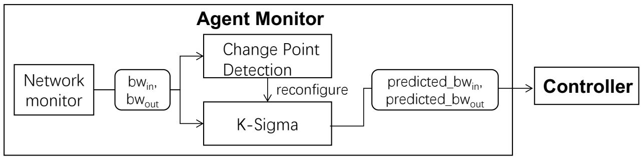


Fig. 8: **Logical diagram of BDS+'s dynamic bandwidth separation.**

So we further present BDS+ with dynamic bandwidth separation, which adjusts the available bandwidth for bulk data transfer in a real-time manner, by continuously predicting online traffic and automatically adjusting the scheduling decisions, so as to fully utilize network bandwidth accordingly. To be specific, BDS+ automatically adjust the scheduling results under different network conditions: if online traffic encounters its peak, BDS+ shirks its occupied bandwidth to avoid congestions, while online traffic encounters its valley, BDS+ aggressively uses more bandwidth to make full use of the residual bandwidth.

To achieve this, BDS+ leverages a customized online traffic prediction algorithm, which identifies the changes of server bandwidth usage, and triggers re-scheduling to adjust bandwidth allocation to the bulk-data transfer. Figure 8 shows the logical diagram of BDS+'s dynamic bandwidth separation. The Network Change Monitor reads the agent observations (bw_{in} and bw_{out}) and executes a customized combination of *k-Sigma* [29] and a change point detection algorithm [30]. *k-Sigma* is responsible to calculate the mean and standard deviation of agent observation, and the change point detection is responsible for detecting abrupt changes by observing historical data, in order to make the Agent Monitor both stable and sensitive.

To integrate to BDS, we make the above online traffic prediction at the beginning of each cycle, and based on the predicted traffic, BDS updates the available link status and then calculates bottleneck disjoint paths. As the online traffic is time-varying, the available bandwidth of all paths are therefore varies along with the online traffic, making the bottleneck disjoint path different in each scheduling cycle. This requires BDS+ to be able to work under different scenarios (varying number of bottleneck disjoint paths), which also proves the generalizability of BDS+.

5.1 Design Logic

To detect online traffic changes and dynamically adjust configurations, there are some basic methods, such as exponentially weighted moving average (EWMA) control scheme, *k-sigma* [31], [32]. Such approaches sometimes result in continual reconfigurations even when the network is (statistically) stationary (since samples may vary in time series). So it encounters a tradeoff when predicting the available bandwidth: When we put more importance to the recent values as a reference (i.e., k is small), there will be an obvious oscillation in the predicted value, which introduces continual but unnecessary reschedules. When we put more importance to the historical values as references (i.e., k is large), the predicted value will not be affected timely when a change point is suddenly detected, making the system insensitive to network changes.

To address the above problem, BDS+ combines *k-sigma* with a change point detection algorithm [30], which can identify

Changes/Adjustments	Scheduling	Routing
Online Traffic ↑	$w_{b,s}^{(T_k)}$ -	$f_{b,p\in\hat{P}}^{(T_k)} \downarrow$
Online Traffic ↓	$w_{b,s}^{(T_k)}$ +	$f_{b,p\in\hat{P}}^{(T_k)} \uparrow$

TABLE 3: Dynamic adjustment in BDS+ according to the online traffic prediction.

abrupt changes of sequential data. Such algorithms offers both online and offline processing methods, while offline methods [33], [34], [35], [36] require the complete data in full time series to generate samples from the posterior distribution over change point locations, online methods [37], [38], [39] can generate an accurate distribution of the next unseen data with only already observed data. In BDS+, we implemented our customized $sliding-k$ algorithm based on [30] (with code can be found in [40]) into the Network Change Monitor. Specifically, we set an upper bound K for the EWMA algorithm, k gradually increases to K when there is no change point, and will be reset to 0 once a change point is detected, and then gradually increases to K again. This improvement makes the $sliding-k$ more stable.

5.2 Integrated to BDS

5.2.1 Online traffic prediction algorithm

During a scheduling cycle ΔT_k in BDS+, Network Change Monitor is continually fed with a series of agent observations of server throughput (bandwidth usage), which is used to predict the available bandwidth in the next scheduling cycle. To get the bandwidth usage, the Network Change Monitor periodically reads the record in process activity monitor on servers. For particular servers, they continuously log processing activities (including server throughput) and send the sampled summed throughput to the Network Change Monitor. In this way, any network changes occurred during the bulk data downloading can be timely detected.

In addition, it should also be noted that BDS+ faces different mixes of delay sensitive traffic and bulk data traffic at every moment. Specifically, online traffic consists of all the real-time traffic from all the online applications (such as online search, shopping transactions, real-time conversations and so on), which is a different mix at every moment, and it is unknown what applications the online traffic come from in the next cycle. At the same time, bulk data consists of the traffic from multiple offline applications (such as blog articles, search index, forum posts, file sharing and so on). Therefore, when BDS+ is running, the scenario it faces in each scheduling cycle is a different mix of online traffic and bulk data transfer traffic.

5.2.2 Dynamic Bandwidth Separation

When a change is detected, the Network Change Monitor signals the change and the updated available bandwidth to the Controller, triggering rescheduling in BDS+ to make bandwidth adjustments in the next scheduling cycle. Shown in Table 3, such adjustment can be two-fold (assume the affected path by the online traffic change is $\hat{P}$):

- When the total link utilization exceeds the pre-configured safety threshold (80% in the example in §2.3), BDS+ shirks the occupied bandwidth for bulk-data transfer in both scheduling and routing steps to avoid congestions: 1. cancel some blocks that were scheduled in the current scheduling cycle ΔT but not yet transferred; 2. reduce the allocated bandwidth $f_{b,p}^{(T_k)}$ for block b on path $p \in \hat{P}$ in T_k.

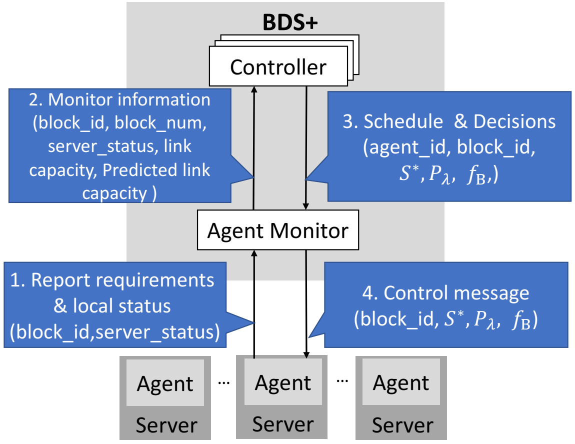

BDS+
Controller
2. Monitor information (block_id, block_num, server_status, link capacity, Predicted link capacity)
3. Schedule & Decisions (agent_id, block_id, $S^*, P_\lambda,\ f_{\mathrm{B}},$)
Agent Monitor
1. Report requirements & local status (block_id,server_status)
4. Control message (block_id, $S^*, P_\lambda,\ f_{\mathrm{B}}$)
Agent
Server
···
Agent
Server
···
Agent
Server

Fig. 9: **Interfaces of BDS+'s centralized control.**

- When online traffic usage encounters its valley, making link utilization fall below the safety threshold, BDS+ aggressively occupies more bandwidth in scheduling and routing steps: 1. transfer some additional blocks that were not scheduled in the current scheduling cycle ΔT; 2. increase the allocated bandwidth $f_{b,p}^{(T_k)}$ for block b on path $p \in \hat{P}$ in T_k, to make full use of the residual bandwidth detected by the online traffic prediction algorithm.

6 SYSTEM DESIGN

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

6.1 Centralized control of BDS+

BDS+ periodically (by default, every three seconds) updates the routing and scheduling decisions in a centralized fashion. Figure 9 outlines the workflow in each three-second cycle.

1) It starts with the *Agent*, running locally 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 an *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 Monitors, 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.

- *Block 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.

6.2 Dynamic bandwidth separation of BDS+

To guarantee dynamic bandwidth separation between inter-DC bulk-data multicasts and delay-sensitive traffic, the BDS+ Network Change Monitor detects any changes of the aggregated bandwidth usage of all latency-sensitive flows on each inter-/intra-DC link, and dynamically allocates the bandwidth for bulk-data multicast transfer accordingly. To protect delay-sensitive flows from being negatively affected by bursty bulk-data transfers, BDS+ is designed to be sensitive to network changes by using a sliding k in the traffic prediction algorithm. In other words, it puts more importance to sudden increases or decreases when online traffic oscillates (to be sensitive), while simultaneously referring to history information when online traffic doesn't change much (to be stable).

BDS+'s dynamic bandwidth separation also takes advantages of the centralized logic of BDS. The traditional techniques (e.g., [2]) that gives higher priority to online latency-sensitive traffic can still have bandwidth wastage or performance interference in the presence of dynamic network environments [41]. BDS+, in contrast, dynamically predicts the bandwidth usage of latency-sensitive applications, and calculates the residual bandwidth that can be allocated to inter-DC multicast. Finally, note that BDS+ optimizes the application-level overlay, thus is complementary to network-layer techniques that improve the WAN performance and fairness [42], [43], [44], [45].

6.3 Fault tolerance

Next we describe how BDS+ handles the following failures.

1. *Controller failure:* The controller is replicated [46]: 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 in a 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 a 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 partitioning happens between DCs, the DCs located in the same partition with the controller will work the same as before, while the DCs in the other partition(s) will fallback to the aforementioned, decentralized overlay network.

6.4 Implementation and deployment

We have implemented BDS+, and deployed it on 67 servers in 10 of Baidu's geo-distributed DCs, with 3621 lines of code in the Go language [47]. Evaluation in the next section is based on this deployment.

The controller was duplicated (for reliability) on three different geo-located zookeeper servers. The Agent Monitor uses `HTTP POST` to send control 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 APIs that consist of three steps: (1) provide 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 administrators 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).

7 Evaluation

Using a combination of pilot deployment in Baidu's DCs, microbenchmarking, and trace-driven simulations, 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 (§7.1).
2. 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 (§7.2).
3. BDS+ can: (1). further complete inter-DC multicast 1.2 to 1.3 times faster with dynamic bandwidth separation, (2). predict the bandwidth utilization of online traffic with about 95% accuracy, (3). increase bandwidth utilization when the online traffic is low, while reducing the bulk data transfer when online traffic bursts, (4). achieve near real-time scheduling with relatively low computational overhead (§7.3).

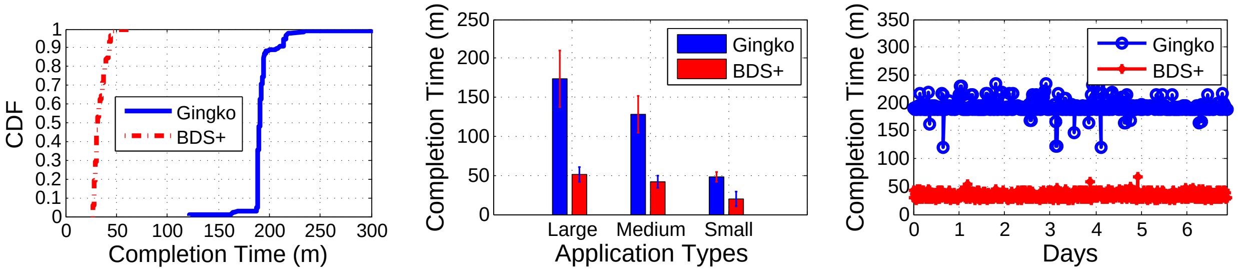
7.1 BDS+ over existing solutions

7.1.1 Methodology

Baselines: We compare BDS+ with three existing solutions: Gingko (Baidu's existing decentralized inter-DC multi-cast strategy), Bullet [13], and Akamai's overlay network [11] (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



(a) Distribution of completion time.

(b) Comparison by application types.

(c) Comparison by completion time.

Fig. 10: **[BDS+ vs. Gingko (Baidu's existing solution)] Results from pilot deployments.**

BDS+ on a larger scale. The simulation is not to reproduce the results of the above pilot deployment, but to provide evaluation results in large-scale scenarios, which is complementary to the pilot deployment. Specifically, 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.

As the existing solutions are all designed under the situation where the available bandwidth is fixed, so in this subsection, we evaluate the basic version of BDS+ with fixed bandwidth separation, to ensure fairness. The additional improvements by BDS+'s dynamic bandwidth separation are shown in §7.3. The whole logic of BDS+ can be summarized as follows: BDS+ first obtains all the paths from the topology of Baidu network, and then conducts the dynamic bandwidth separation by predicting the online traffic on each link. Thus, BDS+ obtains the residual bandwidth of all the paths, and therefore confirms the available links for the subsequent bulk data transfer. On this base, BDS+ runs the scheduling and routing algorithm periodically to find overlay paths for those selected blocks. With the power of the bottleneck disjoint overlay paths, blocks can be transmitted on multiple paths simultaneously, and avoid going through bottleneck links. That's the key to accelerate inter-DC bulk data transfer.

7.1.2 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 10a 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 (70 TB), medium (50 TB) and small (20 TB), and compare BDS+'s and Gingko's mean (and standard deviation) of completion time for each application in Figure 10b. 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. This is because BDS+ adopts "rarest-first" strategy in the scheduling stage, which treats all blocks as the same no matter it belongs to a larger bulk data transmission or a smaller transmission, so there is a strong possibility that blocks from large bulk data transmissions will be scheduled earlier, resulting in greater improvement for those larger bulk data. Finally, Figure 10c 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×.

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

TABLE 4: [BDS+ vs. Bullet [13], Akamai [11]] Completion time of the three solutions in trace-driven simulation.

7.1.3 BDS+ vs. other overlay multicast techniques

Table 4 compares BDS+ with two other baselines, Bullet and Akamai's overlay network, using trace-driven simulation. In the simulation, we set the inter-DC bandwidth to the range from $5TB$ to $25TB$, which is scaled down proportionally according to the real network. We show the results in three setups. In the baseline evaluations, we send 1TB 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 10TB 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 the findings in §7.1.2 that BDS+ has greater improvement when data sizes are large.

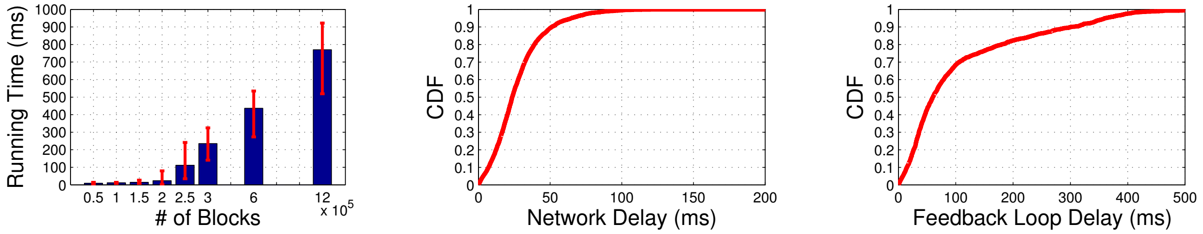
7.2 Micro-benchmarks

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.

7.2.1 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

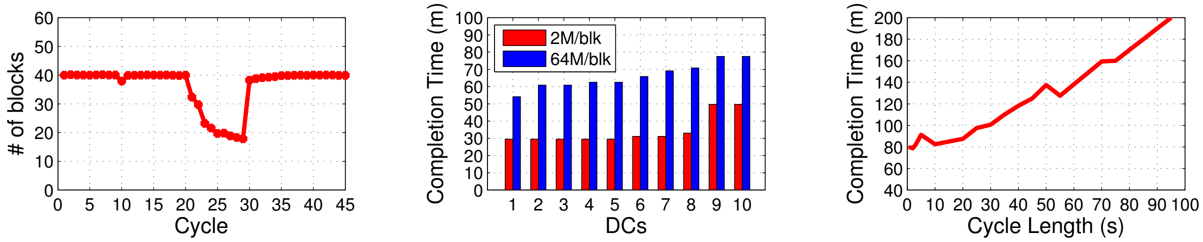


(a) The controller running time.

(b) The inter-DC network delay.

(c) Feedback loop delay.

Fig. 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.

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

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.

7.2.2 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.

7.2.3 Choosing the values of key parameters

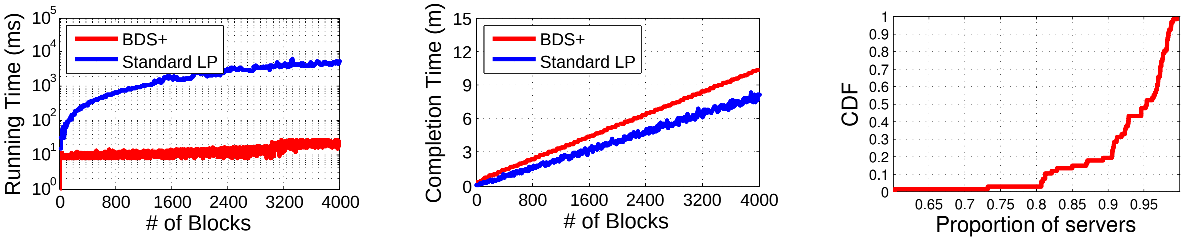
Block size: In BDS+, the bulk data file is split into blocks and can be transferred on bottleneck-disjoint paths. But this introduces a tradeoff between scheduling efficiency and calculation overhead. We therefore conduct two series of experiments using different block sizes (2MB and 64MB). Figure 12b 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.

7.2.4 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 [48], 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 25ms while that of standard LP grows quickly to 4s 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+.



(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.

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

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.

7.3 BDS+'s dynamic bandwidth separation

As existing solutions reserve fixed amount of bandwidth for online traffic according to the peak value (e.g., 20%), while real traces show that online traffic rarely reaches that peak and is far below that in most cases. Thus, BDS+ leverages dynamic bandwidth separation between online traffic and offline traffic, allowing offline traffic (bulk data transfer) to use more bandwidth when online traffic is below the threshold. BDS+ achieves this by designing an online traffic prediction algorithm, and this section shows the results of improved performance by dynamic bandwidth separation. For easy description, we name the basic version with fixed bandwidth separation BDS, while the version with dynamic bandwidth separation BDS+.

In the following experiments, we send 1TB data from one DC to 11 DCs, each has 100 servers, and the upload and download link capacities are set to be 20MBs, same as the previous experiments. The online traffic is set according to the cluster trace (machine_usage) from Ali [49].

7.3.1 Further improvements over BDS.

Completion time: We start the bulk data transfer at 23:00 on 27, Jan, 2019. Figure 14a shows the CDF of the completion time on each destination server. We can see that the average completion time of BDS+ is 150ms, while that under BDS is more than 200ms.

Improvements over BDS: To make the results more general, we further conduct a series of experiments during different time periods, in other words, once per 30 minutes. We compare the completion time of BDS and BDS+, and show the results in Figure 14b. We can see that the improvements of BDS+ changes with time, specifically, the improvements during midnight is much higher than that during the day, especially at 05:30, when online traffic is at its valley. These results show that BDS+ can make full use of the idle bandwidth that is not used by online traffic. Overall, the CDF of improvements is shown in Figure 14c, which means that BDS+ can bring at least 17.8% improvement in about 86% cases.

7.3.2 BDS+'s prediction algorithm.

The improvement of BDS+ mainly comes from the prediction of online traffic, so in this subsection, we evaluate the accuracy of the prediction algorithm, and then analyze the overhead incurred in achieving such improvements.

Algorithm accuracy: The online traffic is set according to the cluster trace (machine_usage) from Ali [49], the real residual bandwidth (the difference between server I/O and online traffic) is shown in black in Figure 15a, where the predicted value is shown in red (after normalization to 100). As we can see that the red line is smooth and quite close to the real bandwidth, indicating that BDS+ can predict online traffic precisely. The exact statistics are shown in Figure 15b, which indicates that the accuracy of about 99% predictions is greater than 92%. Only in 1.6% cases, BDS+ shows a little bit aggressiveness by giving a little bit higher predicted value (where the x-axis is below zero). Taken together, BDS+ can not only increases bandwidth utilization when online traffic is in valley, but also reduces the incidents of interferences caused by bulk-data transfer.

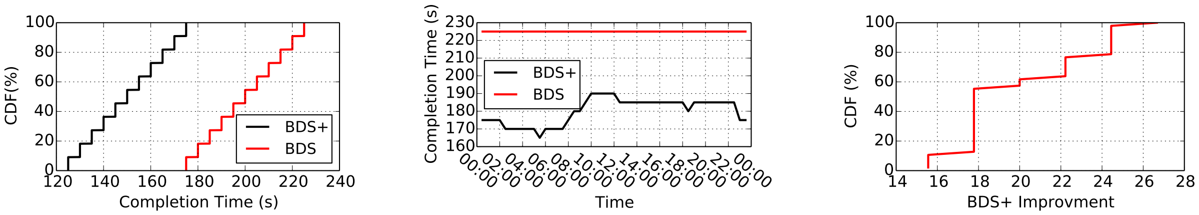
Algorithm overhead: Although BDS+ bring performance improvements by making full use of the residual bandwidth, it introduces some overhead by introducing an additional algorithm. So here we evaluate the additional time spent on making predictions on online traffic. Figure 15c shows the running time during a complete bulk data transfer. We can see that BDS+ takes less than 20ms to make predictions in more than 97% cases. What's more, this overhead does not increase with system scale, because the prediction on each server is independent of each other and thus can be executed simultaneously.

In summary of all the above experiments, both the prototype pilot deployment and the trace-driven simulations of BDS+ with fixed bandwidth separation show 3-5× speedup over existing solutions, with good scalability, reliability, and near-optimal scheduling results. While BDS+ with dynamic bandwidth separation further brings 1.2 to 1.3 times improvement, thus working harmoniously with time-varying online traffic.

8 Related Work

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

Overlay Network Control. Overlay networks realize great potential for 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 [21], Joost [50],

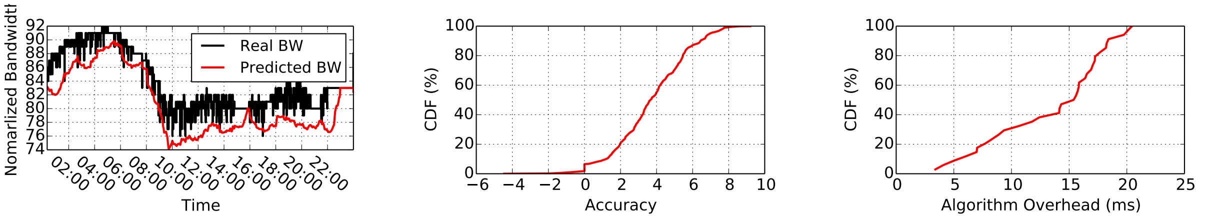


(a) CDF of server completion time.

(b) Comparison at different periods.

(c) CDF of improvement.

Fig. 14: **[BDS+ vs. BDS] Further improvements from BDS.**



(a) Available bandwidth and the predicted value.

(b) CDF of accuracy of the online traffic prediction algorithm.

(c) Running time of the prediction algorithm.

Fig. 15: **[BDS+'s prediction algorithm] Evaluations on: (a) predicted value, (b) algorithm accuracy, (c) running time.**

PPStream [51], UUSee [52]), video-on-demand (VoD) applications (OceanStore [53]), distributed hash tables [54] and more recently Bitcoin [55] and routing [56]. 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 [57] and live streaming [20].

We briefly introduce the two baselines in the evaluation section: (1) Bullet [13], which enables geo-distributed nodes to self-organize into an overlay mesh. Specifically, each node uses RanSub [58] 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 [11], 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 [59], PDQ [60], CONGA [61], DCQCN [62] and TIMELY [63] are all classical protocols showing clear improvements in transmission efficiency. Some congestion control protocols like the credit-based ExpressPass [64] and load balancing protocols like Hermes [65] could further reduce flow completion time by improving rate control. On this basis, the recent proposed Numfabric [66] and Domino [67] further explore the potential of centralized TCP on speeding up data transfer and improving DC throughput. To some extend, co-flow scheduling [68], [69] 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 [42], [43], [44], [45], [70], [71], [72] have illustrated the challenges of scalability, heterogeneity etc., especially on inter-DC level. The representative TE systems include Google's B4 [5] and Microsoft's SWAN [6]. B4 adopts SDN [73] and OpenFlow [74], [75] 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. In recent years, there are also some new research work on inter-DC multicast, for example, [76], [77] propose a tree selection technique called QuickCast, which reduces the centralized computation overhead by cutting the large forwarding tree into multiple smaller ones. As comparison, BDS+ decouples the whole algorithm into scheduling and routing stages. Further, some deadline-aware algorithms like [78], [79] are also emerging, but in our scenario, the data need to be transferred are large amount of bulk data, so we treat small block the same priority, except some special cases (as explained in Section 4.3).

Bandwidth preemption. Resource over-subscription or under-subscription is a common problem in DCs or WANs, and it often leads to unreasonable utilization in clusters, cloud, and data center environments. There have been many efforts that try to schedule more ad-hoc jobs on the premise that the QoS of critical jobs can be guaranteed. One of the most representative schemes is preemption. To eliminate sharing-induced unpredictability, [80] leverages the notion of recurring reservation, which isolates periodic tasks from the sharing noisiness. [81] also proposes a reservation-based scheduling scheme, which delivers resource allocations to both production jobs and best-effort jobs to improve cluster utilization. This work shares the similar problem with BDS+, but it builds upon Hadoop/YARN, which means preemption will happen when critical workloads increase, while BDS+ eliminates the possibility of preemption by dynamically predicting online traffic and make reservation. Flowtime proposed in [82] is also a scheduling framework for both deadline-aware workflows and ad-hoc jobs, and formulates the optimization into a linear program (LP) problem by decomposing workflows into direct acyclic graph (DAG). The difference between BDS+ and this work is the working scenario, in the bulk data transfer problem, online traffic has no deadlines but must be scheduled in real time, while the bulk data (equal to ad-hoc jobs) has deadline which is relatively not so strict. So all these existing solutions can not be applied into Baidu directly.

Network change detection. Detecting network changes is quite important not only in traffic prediction problems, but also in many other applications, such as abnormality detection, network monitoring, and security. There are two basic but mature methods that are widely used, the exponentially weighted moving average (EWMA) control scheme [31], [32] and the change point detection algorithm [30]. EWMA usually gives higher weights to recent observations while gives decreased weights in geometric progression to the previous observations, when predicting the next value. Although EWMA describes a graphical procedure for generating geometric moving averages smoothly, it faces an essential sensitivity problem, in other words, it can not identify abrupt changes. In contrast, change point detection algorithms could exactly solve this problem, in both online [37], [38], [39] and offline [33], [34], [35], [36] manner. BDS+ combines these two methods by designing a sliding observation window, which makes BDS+'s prediction algorithm both stable and sensitive.

Overall, an application-level multicast overlay network with dynamic bandwidth separation 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.

9 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 with dynamic bandwidth separation that substantially improves the performance of inter-DC bulk-data multicast. BDS+ not only 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, but also shows further improvements by dynamically separating online and offline traffic instead of a fixed boundary. We believe that the insight of multicast overlay network in BDS+, to speed up the execution of a centralized algorithm, together with the inspiration of dynamic bandwidth prediction, can be generalized to other control platforms where the decision-making logic strikes a favorable balance between insurance and efficiency.

Acknowledgments

The work of Yuchao Zhang is supported in part by the National Key Research and Development Program of China (2019YFB1802603), The National Natural Science Foundation of China (NSFC) Youth Science Foundation (61802024), and The Fundamental Research Funds for The Central Universities under Grant (2482020RC36). The work of Ke Xu is supported in part by the Science and Technology Innovation Project (2020KJ010501), China National Funds for Distinguished Young Scientists (61825204), and Beijing Outstanding Young Scientist Program (BJJWZYJH01201910003011).

References

[1] Y. Zhang, J. Jiang, K. Xu, X. Nie, M. J. Reed, H. Wang, G. Yao, M. Zhang, and K. Chen, "Bds: a centralized near-optimal overlay network for inter-datacenter data replication," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 10.

[2] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila *et al.*, "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *ACM SIGCOMM*, 2015, pp. 1–14.

[3] Y. Zhang, K. Xu, G. Yao, M. Zhang, and X. Nie, "Piebridge: A cross-dr scale large data transmission scheduling system," in *ACM SIGCOMM*. ACM, 2016, pp. 553–554.

[4] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The end-to-end effects of Internet path selection," in *ACM SIGCOMM*, vol. 29, no. 4, 1999, pp. 289–299.

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM*, vol. 43, no. 4, 2013, pp. 3–14.

[6] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *ACM SIGCOMM*, 2013, pp. 15–26.

[7] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang, "Guaranteeing deadlines for inter-datacenter transfers," in *EuroSys*. ACM, 2015, p. 20.

[8] Y. Zhang, K. Xu, H. Wang, Q. Li, T. Li, and X. Cao, "Going fast and fair: Latency optimization for cloud-based service chains," *IEEE Network*, 2017.

[9] J. Liebeherr, M. Nahas, and W. Si, "Application-layer multicasting with Delaunay triangulation overlays," *IEEE JSAC*, vol. 200, no. 8, pp. 1472–1488, 2002.

[10] F. Wang, Y. Xiong, and J. Liu, "mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast," in *ICDCS*, 2007, p. 49.

[11] K. Andreev, B. M. Maggs, A. Meyerson, and R. K. Sitaraman, "Designing Overlay Multicast Networks For Streaming," *SPAA*, pp. 149–158, 2013.

[12] K. Mokhtarian and H. A. Jacobsen, "Minimum-delay multicast algorithms for mesh overlays," *IEEE/ACM TON*, vol. 23, no. 3, pp. 973–986, 2015.

[13] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *ACM SOSP*, vol. 37, no. 5. ACM, 2003, pp. 282–297.

[14] T. Repantis, S. Smith, S. Smith, and J. Wein, "Scaling a monitoring infrastructure for the akamai network," *Acm Sigops Operating Systems Review*, vol. 44, no. 3, pp. 20–26, 2010.

[15] T. Y. Huang, R. Johari, N. Mckeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: evidence from a large video streaming service," *SIGCOMM*, pp. 187–198, 2014.

[16] E. Nygren, R. K. Sitaraman, and J. Sun, *The Akamai network: a platform for high-performance internet applications*. ACM, 2010.

[17] Y. Zhang, Y. Li, K. Xu, D. Wang, M. Li, X. Cao, and Q. Liang, "A communication-aware container re-distribution approach for high performance vnfs," in *IEEE ICDCS 2017*. IEEE, 2017, pp. 1555–1564.

[18] Y.-h. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *ACM SIGMETRICS*, vol. 28, no. 1. ACM, 2000, pp. 1–12.

[19] A. K. Datta and R. K. Sen, "1-approximation algorithm for bottleneck disjoint path matching," *Information processing letters*, vol. 55, no. 1, pp. 41–44, 1995.

[20] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *IMC*. ACM, 2004, pp. 41–54.

[21] X. Zhang, J. Liu, B. Li, and Y.-S. Yum, "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming," in *INFOCOM*, vol. 3. IEEE, 2005, pp. 2102–2111.

[22] M. K. Mukerjee, J. Hong, J. Jiang, D. Naylor, D. Han, S. Seshan, and H. Zhang, "Enabling near real-time central control for live video delivery in cdns," in *ACM SIGCOMM*, vol. 44, no. 4. ACM, 2014, pp. 343–344.

[23] I. Gog, M. Schwarzkopf, A. Gleave, R. N. M. Watson, and S. Hand, "Firmament: Fast, Centralized Cluster Scheduling at Scale," in *OSDI*. Savannah, GA: USENIX Association, 2016, pp. 99–115. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gog

[24] B. Cohen, "Incentives build robustness in bittorrent," *Proc P Economics Workshop*, pp. 1–1, 2003.

[25] N. Garg, V. V. Vazirani, and M. Yannakakis, "Primal-dual approximation algorithms for integral flow and multicut in trees," *Algorithmica*, vol. 18, no. 1, pp. 3–20, 1997.

[26] N. Garg and J. Koenemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 630–652, 2007.

[27] M. J. Reed, "Traffic engineering for information-centric networks," in *IEEE ICC*, 2012, pp. 2660–2665.

[28] L. K. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," *SIDMA*, pp. 505–520, 2000.

[29] Friedrich and Pukelsheim, "The three sigma rule," *The American Statistician*, vol. 48, no. 2, pp. 88–91, 1994. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00031305.1994.10476030

[30] R. P. Adams and D. J. MacKay, "Bayesian online changepoint detection," *arXiv preprint arXiv:0710.3742*, 2007.

[31] S. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 1, no. 3, pp. 239–250, 1959.

[32] J. M. Lucas and M. S. Saccucci, "Exponentially weighted moving average control schemes: properties and enhancements," *Technometrics*, vol. 32, no. 1, pp. 1–12, 1990.

[33] A. Smith, "A bayesian approach to inference about a change-point in a sequence of random variables," *Biometrika*, vol. 62, no. 2, pp. 407–416, 1975.

[34] D. Stephens, "Bayesian retrospective multiple-changepoint identification," *Applied Statistics*, pp. 159–178, 1994.

[35] D. Barry and J. A. Hartigan, "A bayesian analysis for change point problems," *Journal of the American Statistical Association*, vol. 88, no. 421, pp. 309–319, 1993.

[36] P. J. Green, "Reversible jump markov chain monte carlo computation and bayesian model determination," *Biometrika*, vol. 82, no. 4, pp. 711–732, 1995.

[37] E. Page, "A test for a change in a parameter occurring at an unknown point," *Biometrika*, vol. 42, no. 3/4, pp. 523–527, 1955.

[38] F. Desobry, M. Davy, and C. Doncarli, "An online kernel change detection algorithm," *IEEE Transactions on Signal Processing*, vol. 53, no. 8, pp. 2961–2974, 2005.

[39] G. Lorden *et al.*, "Procedures for reacting to a change in distribution," *The Annals of Mathematical Statistics*, vol. 42, no. 6, pp. 1897–1908, 1971.

[40] "Bayesian changepoint detection." https://github.com/dtolpin/bocd.

[41] H. Wang, T. Li, R. Shea, X. Ma, F. Wang, J. Liu, and K. Xu, "Toward cloud-based distributed interactive applications: Measurement, modeling, and analysis," *IEEE/ACM ToN*, 2017.

[42] Y. Chen, S. Alspaugh, and R. H. Katz, "Design insights for MapReduce from diverse production workloads," CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE, Tech. Rep., 2012.

[43] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *CCGrid*. IEEE, 2010, pp. 94–103.

[44] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from google compute clusters," *ACM SIGMETRICS PER*, vol. 37, no. 4, pp. 34–41, 2010.

[45] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *SoCC*. ACM, 2012, p. 7.

[46] L. Lamport, "The part-time parliament," *ACM TOCS*, vol. 16, no. 2, pp. 133–169, 1998.

[47] "The go programming language," https://golang.org.

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

[49] "cluster-trace-v2018 from ali," https://github.com/alibaba/clusterdata/blob/v2018/cluster-trace-v2018/trace_2018.md.

[50] "Joost," http://www.joost.com/.

[51] "Ppstream," http://www.ppstream.com/.

[52] "Uusee," http://www.uusee.com/.

[53] "Oceanstore," http://oceanstore.cs.berkeley.edu/.

[54] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "Opendht: a public dht service and its uses," in *ACM SIGCOMM*, vol. 35, 2005, pp. 73–84.

[55] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-NG: A scalable blockchain protocol," in *NSDI*, 2016.

[56] C. Chen, Y. Tock, and S. Girdzijauskas, "Beaconvey: Co-design of overlay and routing for topic-based publish/subscribe on small-world networks," in *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*. ACM, 2018, pp. 64–75.

[57] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 59–69, 2011.

[58] A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat, "Using random subsets to build scalable network services," in *USITS*, 2003, pp. 19–19.

[59] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *ACM SIGCOMM*, 2010, pp. 63–74.

[60] C. Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," 2012, pp. 127–138.

[61] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, and N. Yadav, "CONGA: distributed congestion-aware load balancing for datacenters," in *ACM SIGCOMM*, 2014, pp. 503–514.

[62] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion Control for Large-Scale RDMA Deployments," *ACM SIGCOMM*, vol. 45, no. 5, pp. 523–536, 2015.

[63] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," in *ACM SIGCOMM*, 2015, pp. 537–550.

[64] I. Cho, K. H. Jang, and D. Han, "Credit-Scheduled Delay-Bounded Congestion Control for Datacenters," in *ACM SIGCOMM*, 2017, pp. 239–252.

[65] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient Datacenter Load Balancing in the Wild," in *ACM SIGCOMM*, 2017, pp. 253–266.

[66] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti, "Numfabric: Fast and flexible bandwidth allocation in datacenters," in *ACM SIGCOMM*, 2016, pp. 188–201.

[67] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, "Packet transactions: High-level programming for line-rate switches," in *ACM SIGCOMM*, 2016, pp. 15–28.

[68] Chowdhury, MosharafStoica, and I. Eecs, "Coflow: An Application Layer Abstraction for Cluster Networking," *Hotnets*, 2012.

[69] H. Zhang, L. Chen, B. Yi, K. Chen, Y. Geng, and Y. Geng, "CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark," in *ACM SIGCOMM*, 2016, pp. 160–173.

[70] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, "Modeling and synthesizing task placement constraints in google compute clusters," in *SoCC*. ACM, 2011, p. 3.

[71] J. Wilkes, "More google cluster data," http://googleresearch.blogspot.com/2011/11/, 2011.

[72] Q. Zhang, J. L. Hellerstein, and R. Boutaba, "Characterizing task usage shapes in google's compute clusters," in *LADIS*, 2011.

[73] N. McKeown, "Software-defined networking," *INFOCOM keynote talk*, vol. 17, no. 2, pp. 30–32, 2009.

[74] OpenFlow, "Openflow specification," http://archive.openflow.org/wp/documents.

[75] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM*, vol. 38, no. 2, pp. 69–74, 2008.

[76] M. Noormohammadpour, C. S. Raghavendra, S. Rao, and S. Kandula, "Dccast: Efficient point to multipoint transfers across datacenters," in *9th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, 2017.

[77] M. Noormohammadpour, C. S. Raghavendra, S. Kandula, and S. Rao, "Quickcast: Fast and efficient inter-datacenter transfers using forwarding tree cohorts," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 225–233.

[78] L. Luo, K.-T. Foerster, S. Schmid, and H. Yu, "Dartree: deadline-aware multicast transfers in reconfigurable wide-area networks," in *Proceedings of the International Symposium on Quality of Service*. ACM, 2019, p. 28.

[79] L. Luo, Y. Kong, M. Noormohammadpour, Z. Ye, G. Sun, H. Yu, and B. Li, "Deadline-aware fast one-to-many bulk transfers over inter-datacenter networks," *IEEE Transactions on Cloud Computing*, 2019.

[80] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, Í. Goiri, S. Krishnan, J. Kulkarni *et al.*, "Morpheus: Towards automated slos for enterprise clusters," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 117–134.

[81] C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao, "Reservation-based scheduling: If you're late don't blame us!" in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–14.

[82] Z. Hu, B. Li, C. Chen, and X. Ke, "Flowtime: Dynamic scheduling of deadline-aware workflows and ad-hoc jobs," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 929–938.

10 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 $t_B = \frac{(m-k_1)V}{k_1R}$.

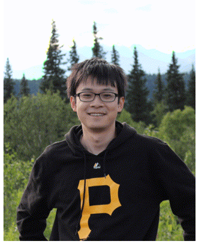
We can show that $\frac{(m-k)V}{kR}$ is a monotonically decreasing function of k:

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

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

Yuchao Zhang received her PhD degree from Computer Science Department at Tsinghua University in 2017. Before that, she received B.S. degree in Computer Science and Technology at Jilin University in 2012. Her research interests include large scale datacenter network (DCN), content delivery network (CDN), data-driven network (DDN) and edge computing (EC). She is currently an Associate Professor in the Beijing University of Posts and Telecommunications.

Xiaohui Nie received B.S. from Computer Science Department, Jilin University, ChangChun, China, in 2013, and received PHD degree from Computer Science Department at Tsinghua University in 2019. His current research interests include AIOps, intelligent TCP, service monitoring and management. He is currently a postdoc in Tsinghua University.



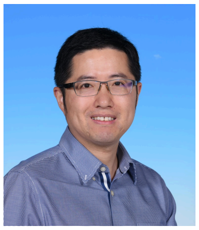
Junchen Jiang received his PhD degree from Computer Science Department at Carnegie Mellon University in 2017. Before that, he received Bachelor degree in Computer Science from Yao Class at Tsinghua in 2011. He is currently an Assistant Professor in the University of Chicago.

Wendong Wang (M'05) received his B.E. and M.E. degrees both from the Beijing University of Posts and Telecommunications, China, in 1985 and 1991, respectively, where he is currently a Full Professor in State Key Laboratory of Networking and Switching Technology. He has published over 200 of papers in various journals and conference proceedings. His current research interests are the next generation network architecture, network resources management and QoS, and mobile Internet. He is a member of IEEE.

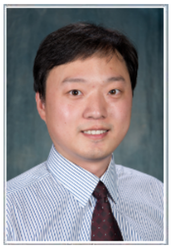
Ke Xu (M'02-SM'09) received his Ph.D. from the Department of Computer Science and Technology of Tsinghua University, where he serves as full professor. He serves as Associate Editor of IEEE Internet of Things Journal and has guest edited several special issues in IEEE and Springer Journals. His research interests include next generation Internet, P2P systems, Internet of Things, network virtualization, and network economics. He is a member of ACM.

Youjian Zhao received the B.S. degree from Tsinghua University in 1991, the M.S. degree from the Shenyang Institute of Computing Technology, Chinese Academy of Sciences, in 1995, and the Ph.D. degree in computer science from Northeastern University, China, in 1999. He is currently a Professor with the CS Department, Tsinghua University. His research mainly focuses on high-speed Internet architecture, switching and routing, and high-speed network equipment.

Martin J. Reed is a Senior Lecturer in the School of Computer Science and Electronic Engineering at the University of Essex, Colchester, UK. His main research interests are the fields of Computer Networks, Network Control, Multimedia transmission over networks and Network Security. His teaching duties are also in these areas.



Kai Chen received the Ph.D. degree in computer science from Northwestern University, Evanston, IL, USA, in 2012. He is an Associate Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong. His research interest includes networked systems design and implementation, data center networks, and cloud computing.



Haiyang Wang is an Associate professor in the Department of Computer Science at the University of Minnesota Duluth, MN, USA. He received his Master's degree from Tsinghua University and Tongji University in 2007, and PhD from Simon Fraser University, Canada, in 2013. His research interests are in networking, in particular, cloud computing, big data, socialized content sharing, multimedia communications, peer-to-peer networks, and distributed computing.

Guang Yao is a senior R&D Engineer in Baidu . He received his Ph. D. degree from the Department of Computer Science, Tsinghua University, in 2012. His main research interests include source address verification, spoofing IP traceability, and traffic control.