BDS: Near-Optimal Multicast Overlay for Inter-Datacenter WANs

PaperID: 192

Abstract

Distributing bulk data across datacenters (DCs) in a timely and cost-effective manner is critical to large-scale online service providers. While recent research has significantly improved the WAN performance between DCs, we argue that a multicast overlay network that optimally schedules and delivers data in a way that fully exploits available overlay paths is essential to achieving desirable performance. Drawing on the experience of a large online service providers, we observe two requirements of an inter-DC multicast overlay network: (1) overlay routing and scheduling needs to be driven by an up-to-date global view of data delivery status at all servers, and (2) it must prevent delay of latency-sensitive traffic caused by bulk data transfers.

This paper presents BDS, a near-optimal inter-DC multicast overlay network that distributing bulk data. At the core of BDS are two design choices. First, decision making in BDS is fully centralized; the BDS controller directly orchestrates servers to split, reorder, and deliver data dynamically along overlay paths, in order to circumvent inter-/intra-DC bottlenecks. Second, BDS enforces a dynamic separation of bandwidth allocated for bulk data transfers and latency-sensitive traffic. While both design choices introduce costs in performance (i.e., BDS is unable to update decisions in real time or achieve full link utilization), our design philosophy is that these costs are outweighed by the benefits of centralized optimization driven by a global view. Using a pilot deployment of BDS in one of the largest search engine service providers, we show that BDS achieves 3-5$\times$ speedup over the provider's existing systems and the techniques widely used in today's content delivery networks.

1 Introduction

Global-scale online services, such as Google, Facebook, and Baidu, transfer massive amounts of data between geo-distributed datacenters (DCs). A common communication pattern is *bulk-data multicast*, which replicates data (typically several to 100s GB), such as user logs, search engine indexes, and databases, to any subset of DCs. However, multicasting bulk data in a timely and cost-effective manner has introduced substantial challenges, as data sizes continue to explode and more DCs are deployed to reach a global footprint; e.g., data Google has seen a XXX-fold increase in XXX years in the amount of data that needs to be distributed to all DCs [**?**], and this number would even multiply with an increasing number of DCs. These trends have been a key driving force behind recent efforts to optimize the performance of inter-DC WANs [**?**, **?**, **?**, **?**, **?**, **?**],

In this paper, we argue that these solutions are incomplete, since they fall short of harnessing servers' capability to store-and-forward data, and could create hot-spots and bottlenecks in DC networks. Figure **??** shows

Drawing an analogy to classic multicast overlay networks, we argue that online service providers need an *application-level overlay multicast network* that optimally schedules and delivers data in a way that fully utilizes available application-level overlay paths between DCs.

While multicast overlay networks have been studied extensively in settings of peer-to-peer (P2P) streaming [**?**, **?**] and content delivery networks (CDNs) [**?**, **?**], it remains unclear whether existing approaches apply to the scale of large online service providers like Google and Baidu. Drawing on the operational experience from Anon[1], a large online service providers, we see two key requirements of an inter-DC multicast overlay network. First, because these DCs have more servers and thus exponentially more overlay paths than CDNs or P2Ps, it is untenable to exploit all overlay paths by traditional approaches, such as decentralized or hybrid path selection (e.g., [**?**, **?**]), or structured topologies (e.g., [**?**]). Second, these WANs are shared by latency-sensitive traffic and bulk background traffic. Because any small increase in delay of latency-sensitive traffic can cause substantial revenue losses, there is a strong need to prevent interference on latency-sensitive traffic, even at the expense of slightly lower link utilization.

This paper presents *BDS*, a near-optimal inter-DC

[1] Anonymized for double-blind reviewing

multicast overlay network, which minimizes data distribution delay from one DC to any subset DCs by dynamically splitting, reordering, and deliver data via overlay paths selected from all server-level paths between the source DC and destination DCs. BDS focuses on distributing background bulk data (e.g., XXX), which is by several orders of magnitudes larger than latency-sensitive user data. At the core of BDS are two design choices.

- *Centralized decision-making:* Decision making in BDS is fully centralized; the BDS controller maintains a global view of data delivery status on all servers and makes overlay routing and scheduling decisions every few seconds (by default, 3 seconds). BDS solves a multicommodity problem using a near-optimal algorithm amenable to efficient incremental updates.
- *Dynamic bandwidth separation:* To prevent delay caused by background data on the latency-sensitive user data, BDS continuously monitors the aggregated traffic of latency-sensitive data and enforces a dynamic separation of bandwidth between background data and latency-sensitive data.

While BDS's design choices introduce performance costs (i.e., BDS does not update decisions in real time or achieve full link utilization), our design philosophy is that these costs are outweighed by several benefits. (1) Since delivering bulk data takes tens of seconds to minutes, BDS can tolerate a delay of updating centralized control decisions at coarse timescales of several seconds in exchange for closer-to-optimal decisions made by a centralized controller. (2) That the aggregation of latency-sensitive traffic tends to be stable on timescales of several seconds suggests that it is plausible to eliminate undesired interference on latency-sensitive traffic by clean bandwidth separation while maintaining a high link utilization. (3) Finally, these design choices are amenable to a simple implementation from an engineering perspective, since the logic running on the server side is only triggered by data arrivals or control messages, and thus can be stateless.

We implemented a prototype of BDS and integrated it in Anon, one of the largest search engine service providers. We deployed BDS in ten of Anon's DCs, and ran pilot study on XXX PB data over XXX days. Our real-world experiments show that BDS achieves 3-5$\times$ speedup over Anon's existing solution. Using real trace-driven evaluation and microbanchmarking, we also show that BDS outperforms many techniques used in today's CDNs, and BDS can scale out to the traffic volume of Anon's inter-DC WAN, and tolerate various failure scenarios.

2 Overlay Network for Inter-DC Multicast

[JC: Starting from Section 2, it would be paper outline. Please think each bullet point as a separate paragraph.]

We begin by motivating the need for an inter-DC multicast network, and lessons from operating Anon's inter-DC multicast network.

2.1 The need for inter-DC multicast

- The architecture of Anon's inter-DC WAN. Scale of data centers, data volume, and traffic types
- A substantial fraction of objects need to be multicast. Its share is rising
- There are many overlay paths between two DCs. A substantial fraction of them are not bottlenecked by the inter-DC WAN (this is necessary to justify the use of server-level overlay paths, not just DC-level)

2.2 Anon's inter-DC multicast network

- Explain how to multicast data from one DC to many DCs through overlay paths across Anon's DCs: e.g., data transfer over HTTP?, How to get the public address of a server in another DC?
- Describe the basic idea of Anon's protocol (describe it as a receiver-driven decentralized protocol) We should also stress that this solution has been running for XXX years and has been continuously improved over time.
- Briefly mention other solutions (layered structure, hybrid approach, and why not optimizing pair-wise DC link is not sufficient)

2.3 Lessons from Anon's inter-DC multicast network

- Lesson 1: need for a global view. Without global view, there is a significant room for improvement. Ideally, we need to show a CDF with two lines that one for flow completion time under the optimal overlay decisions, and other showing the completion time of the existing protocol.
- Lesson 2: need for clean traffic separation. (1) Use a figure to show that bulk data transfer can cause significant delay on latency-sensitive traffic, and (2) put some concrete numbers to show such delay can cause significant revenue loss.

3 Overview of BDS

The insights obtained from Anon's operational experience has inspired the design of BDS, a near-optimal inter-DC multicast overlay network. This section starts with BDS key design choices, highlights the design philosophy behind our choices, and provide an overview of the BDS system, which builds on these design choices.

Our first design choice is the fully centralized control. For large online service providers like Anon, there are considerably large number of servers and exponentially more overlay paths, so it is hard to find out the optimal one for any decentralized solutions with only local information. BDS, however is able to make near-optimal overlay routing and scheduling decisions with a centralized decision-making scheme. Furthermore, the embedded controller is lightweight in terms of CPU, bandwidth consumed, and thus enjoys good scalability over WANs.

The second design choice is dynamic bandwidth separation. To prevent delay caused on the latency-sensitive user data, BDS separates background bulk data transfer from latency-sensitive traffic. Specifically, BDS maintains the information of all links and monitors the aggregated traffic from all latency-sensitive data, thus can dynamically calculate the residual bandwidth that can be allocated for background bulk data transfer. This clean bandwidth separation can efficiently prevent interference on latency-sensitive traffic.

The above two design choices may introduce performance costs, i.e., fully centralized control makes BDS unable to update decisions in real time, because the BDS controller works in a centralized manner and has to collect information from geo-distributed DCs, and this will naturally introduce some communication latency to the decision making process. Besides, the clean separation on bandwidth will possibly result in low link utilization due to the ever changing link utilization and the coarse-grained scheduling decisions.

Fortunately, the above costs are outweighed by the benefits. (1) bulk data transfer usually takes tens of seconds to minutes, so it can tolerates a delay at coarse timescale of several seconds in exchange for near-optimal scheduling decisions. (2) the aggregation of latency-sensitive data is stable on timescales of several seconds, therefore, it is plausible for the dynamic bandwidth separation to prevent interference on the latency-sensitive data while still maintaining high link utilizations. (3) the resulting system is amenable to a simpler implementation because the decision-making logic running on the controller side does not need to maintain data status or to deliver complex control messages, thus can be stateless and lightweight.

The key technical challenge here is how to make optimal overlay scheduling and routing decisions at the scale tens of thousands of objects and tens of thousands of servers in near real time. To achieve desirable performance in a multicast overlay network, fully exploiting all the available overlay paths is essential, but it is untenable to go through all the potential servers and exponentially more paths by traditional approaches.

Variables	Meaning
$\mathbb{A}$	Set of all (s,d) pair
$\mathbb{B}$	Set of blocks of all tasks
$B_{i,j}$	Block i in Task j
$c(l_{u,v})$	Capacity of link $l_{u,v}$
$Path(s,d)$	Set of all potential paths in $\mathbb{A}$
$f_{B_{i,j},p_\lambda}$	Allocated bandwidth for $B_{i,j}$ on path p_λ
$I_{B_{i,j},p_\lambda}$	0 or 1: whether p_λ is selected for $B_{i,j}$

Table 1: Variables in BDS.

4 Near-Optimal and Efficient Decision-Making Logic

At a high level, BDS optimizes the data distribution performance by splitting data into fine-grained blocks so as to exploiting all available server-level overlay paths, and possible reordering of blocks to speed up the process. In a general case, it is indeed intractable to solve the problem in near real-time, but BDS can find a near-optimal solution for our problem scale in several seconds by using applying two approximations: (1) separating the problem of data scheduling and overlay routing, and (2) using standard linear-programming relaxation to solve them efficiently.

4.1 Problem formulation

To formulate the data distribution problem over inter-DC WANs and design the decision-making logic for the centralized controller, we should first clarify the following aspects (Table 1 summarizes some key variables and parameters):

(1) **Input.** The number of DCs m, the set of all blocks $\mathbb{B}$, the optional source and destination pairs (s,d), all potential paths between s and d $Path(s,d)$, link capacity of p_λ $c(p_\lambda)$, the upload/download rate of server n $R_{up}(n)/R_{down}(n)$.

(2) **Output.** The optimal data source for any block s^*, the optimal path for the block p^*, and the allocated bandwidth for $B_{i,j}$ on path p^* $f^*_{B_{i,j},p^*}$.

(3) **Constraints.** The link capacity constraint takes effect on any arbitrary path p_λ: the summed allocated

bandwidth on this path should be no more than its capacity $c(p_\lambda)$. The data size constraint takes effect on blocks: the sum of allocated bandwidth should be no less than its size $\mathbb{S}(B_{i,j})$. The bandwidth constraint takes effect on allocations: the allocated bandwidth on path p_λ should be the minimum of three parameters: link capacity $c(p_\lambda)$, the upload rate of source node $R_{up}(s)$ and the download rate of destination node $R_{down}(d)$.

(4) **Objective function.** To speed up the bulk data distribution over inter-DC WANs, BDS aims at maximizing the allocated weighted bandwidth for all the blocks that have been selected in the scheduling stage, by means of making optimal routing.

The key insight underlying the above BDS's formulation is the separation of data scheduling and overlay routing, and there are two main benefits of this separation. The first one is to reduce the computational complexity on the centralized controller side. The objective of the separated scheduling stage is to select a subset of blocks, and only these selected blocks will be routed and transferred in the following routing stage. So the separated data scheduling could eliminate the unnecessary overlay path explorations for the overwhelming majority of the unselected blocks. The second benefit is to speed up the overall data distribution. This advantage comes from the customized block selection scheme that picks out a specific subset of blocks so as to reduce the overall completion time.

4.2 Scheduling

We define each complete duplicate transmission of bulk data as one "task", and each DC will launch at least two tasks due to replication strategy. All the tasks wait to be scheduled together in the pending queue. As the size of a task is extremely large (tens of TBs to PB), BDS splits it into fine-grained blocks (several MB) and makes scheduling and routing in the form of blocks. The objective of this data scheduling procedure is to pick out a subset of blocks that should be transferred first.

Assume the origin bulk data in the source DC is split into n blocks, and there are m DCs in the WAN and each launches two transmission tasks. Different scheduling strategies will lead to different intermediate transmission states and finally result in different completion time. Take two intermediate states as examples: 1) All of the n blocks has k duplicates; 2) Some of these n blocks have $k1$ ($k1 < k$) duplicates and other blocks have $k2$ ($k2 > k$) duplicates. Let t_1 denote the completion time of case 1 and t_2 denote that of case 2, we have: $t_2 > t_1$.

Proof:

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

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

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

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

$$t_1 = \frac{\mathbb{S}}{min\{c(l), \frac{R_{up} \times k}{2m-k}, R_{down}\}} \tag{1}$$

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

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

Therefore, in case 2, for those blocks with $k1$ ($k1 < k$) duplicates, the completion time t_{k1} will be larger than t_1, i.e., $t_2 \geq t_{k1} > t_1$.

So in the scheduling stage, BDS will firstly pick out the subset of blocks with the least downloaded duplicates, so as to reduce the overall completion time.

For efficient selection, BDS keeps a counter c_i in the controller for each block and update it once receiving finish notifications from receivers. The scheduling stage always gives priority to the smallest c_i. For efficient processing, BDS keeps all the counters c_i in a doubly linked list in an ascending order of their values. For each download, the controller selects the top item in the list (the smallest value) to be downloaded. The controller listens and serves an HTTP port, once receiving a transmission completion signal from receivers, it updates the corresponding block's counter value and adjusts its position in the linked list for further processing.

4.3 Routing

After the scheduling stage, BDS routes and transfers these selected blocks in this stage. To be specific, the output includes the data source s^*, transmission path p^* and allocated bandwidth $f_{B_{i,j},p_\lambda}$ for each block $B_{i,j}$.

To speed up data distribution, BDS aims at maximizing the allocated weighted bandwidth for all the selected blocks, so the formulation of the objective can be described as:

$$max \sum_{(s,d)\in\mathbb{A}} \sum_{B_{i,j}\in\mathbb{B}} \sum_{p_\lambda \in Path(s,d)} w(B_{i,j}) \cdot f_{B_{i,j},p_\lambda} \cdot I_{B_{i,j},p_\lambda} \tag{3}$$

where $w(B_{i,j}) = \frac{pr_j}{2^{D_j - t}}$ is the weight of $B_{i,j}$, similar to [8], pr_j is the priority of Task j, D_j is the deadline and t is the current time, so 2^{D_j-t} could represent the urgency. $I_{B_{i,j},p_\lambda}$ denotes whether p_λ is selected for $B_{i,j}$. Note that there are multiple potential data sources for each block in the multicast overlay network, so the objective of routing is to select the most efficient data source and assign intermediate paths to all blocks, and then calculate the bandwidth allocation on those selected paths.

The mentioned three constraints can then be formulated as follows:

Link capacity constraint:

$$c(p_\lambda) \geq \sum_{(s,d)\in\mathbb{A}} \sum_{B_{i,j}\in\mathbb{B}} f_{B_{i,j},p_\lambda} \cdot I_{B_{i,j},p_\lambda} \quad \forall p_\lambda \in Path(s,d) \tag{4}$$

Data size constraint:

$$\mathbb{S}(B_{i,j}) \leq \sum_{(s,d)\in\mathbb{A}} \sum_{p_\lambda \in Path(s,d)} f_{B_{i,j},p_\lambda} \cdot I_{B_{i,j},p_\lambda} \cdot \Delta T \quad \forall B_{i,j} \in \mathbb{B} \tag{5}$$

Bandwidth constraint:

$$f_{B_{i,j},p_\lambda} \leq min\{c(p_\lambda), R_{up}(s), R_{down}(d)\} \quad \forall p_\lambda \in Path(s,d) \tag{6}$$

Besides, there is another limitation on path selection: $\sum_{p_\lambda \in Path(s,d)} I_{B_{i,j},p_\lambda} = 1$, which means only one path will be chosen for a particular block.

The integer program (IP) is a multi-commodity flow algorithm which is known to be NP-complete [5] due to the fact that they are integer flows, and there is no known algorithm to find an optimal solution. To make this problem solvable, we look into it from a different perspective. As the size of a task is dozens of TBs to PBs, while each block is just about several MBs, we can approximate tasks although they are infinitesimally split and can be transferred to a set of possible paths between the source DC and the destination DC. So it is possible to solve this IP problem by a linear programming (LP) relaxation [4, 7], and the relaxed problem aims at transferring a fraction of each transmission. However, the number of blocks will thus grow considerably large when splitting tasks infinitesimally, and this will lead to intolerable computing time on the controller side. There are two coping strategies on this problem: on one hand, BDS has a merge scheme before each transmission cycle, and this step merges blocks with the same (s,d) pair into one subtask so as to reduce task number; on the other hand, BDS adopts the improved fully polynomial-time approximation schemes (FPTAS) by Fleischer [3] to work out an ε-optimal solution with $\alpha' \geq \alpha_\varepsilon \geq \alpha'(1-\varepsilon)^{-3}$. This algorithm optimizes the dual problem of the relaxed LP problem by proceeding in phases and iterations.

4.4 Rounding the Integer Solution

BDS distributes each task in an optimal manner given by the maximum concurrent flow (MCF) solution [7]. However, the MCF solution is a linear relaxation of the original integer problem and in practice the solution requires that flows are routed as integers of the block size flow. There are a number of strategies using the output of the MCF for the integer path allocation, however, none of them will be strictly optimal as the original problem is NP-complete. A common strategy is to use *randomized-rounding*. Randomized-rounding was first analyzed in detail for some relaxation approaches by Raghavan and Thompson [6] and a similar approach is followed here for the specific case of the MCF relaxation for which they do not provide a solution. A flow through an arbitrary edge, e is denoted, as $f_e = \sum_{e\in p_\lambda} F_{B_{i,j},p_\lambda} \; \forall \; p_\lambda$ where F^* is the optimum value in the MCF problem that is equivalent to f^* in the original problem. f_e is the optimal MCF flow through edge e that in the integer solution is approximated from the flow of multiple blocks each of flow rate b. To perform the randomized rounding, first, each of the *fractional flows* $F_{B_{i,j},p_\lambda}$ is truncated to an *integer flow* that is constructed from a number of *block flows*, each integer flow is denoted as $\lfloor F_{B_{i,j},p_\lambda} \rfloor = kb \leq F_{B_{i,j},p_\lambda}$ where k is strictly the largest positive integer, or zero, to meet the constraint. Then each integer flow is either kept at kb or *rounded* up with probability $(F_{B_{i,j},p_\lambda} - \lfloor F_{B_{i,j},p_\lambda} \rfloor)/b$ to $(k+1)b$. This rounding probability means that the expected value of the integer flows is f_e, the solution of the optimal linear MFC. The problem is then to determine the likelihood that this rounding will cause the capacity constraint on an edge e to be exceeded. To aid the discussion, a *residual flow* $f'_e = f_e - \sum_{e\in p_\lambda} \lfloor F_{B_{i,j},p_\lambda} \rfloor$ is used to describe the flow that needs to be added to the truncated flows to obtain the optimal solution obtained from MFC.

As each flow rounding is an independent random variable, we can determine the likelihood of exceeding the capacity constraint by applying the following well-known Chernoff bound [2]:

$$\Pr(X > (1+\delta)\mu) \leq e^{\frac{2\delta^2\mu^2}{n(b-a)^2}} \tag{7}$$

where $X = X_1 + \ldots X_n$ is the sum of independent, bounded, random variables $a \leq X_i \leq b$ and $\mu = E(X)$. This bound describes the probability that a summation of random variables exceeds the expected value of the sum by

a factor of δ. In the problem considered here, the random variables are the rounded flows which have an expected summation $\mu = f'_e$. Consider an edge, e, that has its capacity fully used by the linear MFC solution. Using the Chernoff bound from Equation (7), note that $a = 0$ and the highest value of the rounded flow added is one block with flow b then we find that the factor that the integer flow is beyond optimal, and the residual flow is bounded by:

$$\delta \leq \frac{b}{r}\sqrt{\frac{n}{2}\ln\varepsilon} \tag{8}$$

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

5 System Design

Next, we present the system design of BDS to realize the two design choices presented in Section3.

5.1 BDS architecture

BDS reuses the same underlying multicast system described in Section2.2. This section should give a graph to show the key components (controller, local agents, traffic monitoring service), as well as their interfaces.

5.2 Centralized control

- Start with the basic workflow of each 3-second cycle: (1) how local agent collects delivery status, (2) send messages to the controller, (3) controller runs the algorithm, (4) control message to each local agent, and (5) how local agent enforce decision.
- Fault tolerance: what if a server is not available (or straggling), what if one controller instance is not available, what if there is network partition between DCs or between DCs and the controller.
- Explain two optimizations:
  - Merging blocks
  - Non-blocking update

5.3 Dynamic bandwidth separation

- First, how to get real-time aggregated size of latency-sensitive traffic.
- Second, how to calculate the bandwidth cap for background bulk traffic
- Finally, how to enforce the bandwidth cap.

6 Implementation and Deployment

We have deployed BDS on Anon's DCs, which consist of dozens of IDCs located in three major regions in China. BDS is implemented with 3621 lines of golang code [1], and can be fully integrated in Anon's DCs.

Figure 1 shows the detailed implementation of BDS. It consists of five components: *Controller*, *Agent Monitor*, *Agent*, *Storage Platform*, *Network Monitor*. 1) *Controller* is a scheduler which executes the customized scheduling algorithm. 2) *Agent Monitor* delivers messages between the *Controller* and *Agent*. 3) *Agent* is a module deployed in each machines. It announces the bulk data that need to be downloaded to *Agent Monitor*. 4) *Network Monitor* records the each link capacity and the traffic of latency-sensitive data, which is the basic input of the scheduling algorithm. 5) *Storage platform* records the system snapshot, including master and slave status, etc. The *Controller* is replicated three times for fault tolerance. Once the master controller fails, the *Storage Platform* will announce one slave becomes the master.

The basic workflow of BDS can be described as follows: All the *Agent* sends its task and status to *Agent Monitor* by HTTP POST, then *Agent Monitor* aggregates all information for *Controller* to perform scheduling. Once scheduling is finished. *Agent Monitor* send the control message to all the *Agent* by HTTP POST, then *Agent* uses *wget* tools to download bulk data. After one scheduling cycle (XXX), the above procedure will performed again.

Note that only *Agent* is deployed in each IDC's machines, the other components is deployed in centralized clusters, so BDS can be easily deployed to other companies' DCs.

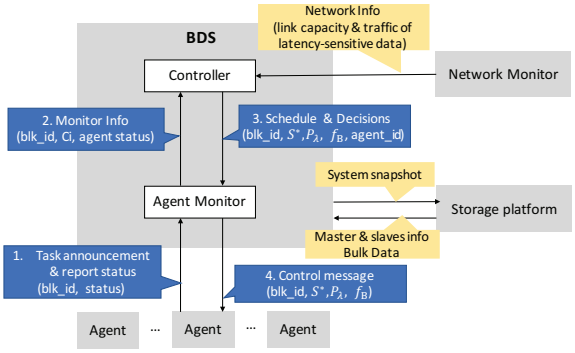

Figure 1: The implementation of BDS.

7 Evaluation

To evaluate BDS, we integrated our end-to-end prototype in Anon, and ran a pilot deployment across XXX

servers in XXX DCs to serve XXX PT data over a period of XXX days. The evaluation results of the pilot deployment, together with trace-driven simulation and microbenchmarks, show that:

1. BDS achieves 3-5× speedup over Anon's existing solution, as well as other industry-standard solutions.

2. BDS can update decisions every 3 seconds over a WAN of the same size as Google, is lightweight in terms of CPU, bandwidth consumed, and can gracefully degrate to decentralized protocols.

7.1 Benefits of centralized control

- BDS vs. Anon's existing solution (pilot deployment)
  - Overall improvement: A CDF with two lines to show the aggregated flow completion time
  - By applications: Pick three applications whose data volumes are large, medium, and small respetively. Draw a bar chart of three pairs of bars, each representing BDS's and Anon's mean (stddev) flow completion time for an application.
  - By time: Timeseris of two lines, each representing BDS's and Anon's mean (stddev) of flow completion time.
- BDS vs. other overlay multicast techniques
  - Begin with the methodology of trace-driven simulation.
  - Briefly explain these techniques: Akamai (3-layer), Bullet (full mesh)
  - Show a CDF that has three lines, representing BDS, Akamai, and Bullet.

7.2 Benefits of dynamic bandwidth separation

- Draw a graph (what graph can you get on this?) to show with bandwidth separation, BDS can reduce the incidents of delay on latency-sensitive traffic caused by bulk data transfers.
- Draw a graph (what graph can you get on this?) to show the link utilization does not change much with BDS or with Anon.

7.3 Micro-benchmarks

- Scalability of centralized control:
  - Y: Bandwidth consumption, vs. X: # of objects
  - Y: Controller CPU usage, vs. X: # of objects
  - Y: Update delay vs. X: # of objects
  - Bar-chart to decompose update delay into collecting updates, running algorithm, and updating local agents
- In-depth analysis:
  - A graph to show the tradeoff caused by different update cycles (why 3 seconds is a good tradeoff?)
  - Reduction on algorithm running time due to the approximation algorithm.
  - Maybe another graph from the current 6.3?
- Fault tolerance:
  - Y: flow completion time, vs. X: time. Create a toy topology to send objects. The experiment begins with no failure. At time t1, one server is not available, and the graph should show Y only has performance degradation for less than 3 seconds; At time t2, the controller is not available, and the local agent should automatically revert to decentralized local control.

8 Related Work

9 Discussion

10 Conclusion

References

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

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

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

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

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

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

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

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