Guaranteeing Deadlines for Inter-Data Center Transfers

Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang

***Abstract*— Inter-data center wide area networks (inter-DC WANs) carry a significant amount of data transfers that require to be completed within certain time periods, or deadlines. However, very little work has been done to guarantee such deadlines. The crux is that the current inter-DC WAN lacks an interface for users to specify their transfer deadlines and a mechanism for provider to ensure the completion while maintaining high WAN utilization. In this paper, we address the problem by introducing a deadline-based network abstraction (DNA) for inter-DC WANs. DNA allows users to explicitly specify the amount of data to be delivered and the deadline by which it has to be completed. The malleability of DNA provides flexibility in resource allocation. Based on this, we develop a system called Amoeba that implements DNA. Our simulations and test bed experiments show that Amoeba, by harnessing DNA's malleability, accommodates 15% more user requests with deadlines, while achieving 60% higher WAN utilization than prior solutions.**

***Index Terms*— Inter-datacenter WAN, deadline, scheduling.**

I. INTRODUCTION

GLOBAL online services and cloud platform providers, such as Google, Microsoft, and Amazon, construct multiple datacenters (DCs) across the world to deliver their services [1], [2]. The wide area network (WAN) that connects these geographically distributed DCs is one of the most critical and expensive infrastructures that costs hundreds of millions of dollars annually [1]. The shared infrastructure provides transit services for tenants. In public clouds (*e.g.*, Amazon AWS), a tenant could be a customer that launches multiple virtual private clouds (VPC) in multiple DCs. In private clouds (*e.g.*, Google and Microsoft's internal DCs), a tenant could be a service team that launches multiple VMs globally.

The inter-DC traffic can be broadly classified into three categories based on time-sensitivity: *interactive* traffic that is most sensitive to delay (*e.g.*, 100ms [1]); *large transfers* which require delivery within certain time periods (*e.g.*, hours); and *background* traffic without strict time requirements [1]–[3]. One key characteristics of inter-DC WAN is that a significant amount of traffic belong to large transfers and have deadlines, either hard or soft [1], [4]. Hard deadline means a transfer is useless to applications once late, whereas soft deadline means the value of a transfer degrades after the deadline, affecting application performance (§III).

Thus, providing deadline guarantees for inter-DC transfers is essential for many applications. To the best of our knowledge, however, no existing mechanism is in place to guarantee the deadlines:

- In private clouds, state-of-the-art inter-DC traffic engineering (TE) techniques do not guarantee deadlines. For example, SWAN [1] and B4 [2] enforce a strict priority among traffic categories, but do not explicitly account for deadlines and thus can cause many transfers to miss their deadlines (§VIII). Tempus [4] maximizes the minimal fraction of delivery of all transfers until deadlines, but does not guarantee the completion of any of them before deadlines (§III).
- In public clouds, the current practice does not even differentiate among different traffic categories. Our measurements of a real inter-DC WAN show that only rate limiting is applied to provide isolation across tenants. In addition, even with the rate limiting, bandwidth varies dramatically across time and DC sites (§II).

This makes it difficult for critical business applications to run on top of the infrastructure. As a result, the inter-DC WAN is under increased pressure to provide service level agreements (SLAs) [5]. The crux is that the current inter-DC WAN lacks both an interface for tenants to specify their transfer deadlines and a mechanism for provider to meet the deadlines. We seek such an interface and a mechanism in this paper. We aim to fully utilize the scarce WAN bandwidth resource to guarantee deadlines for as many transfers as possible.

Existing solutions of intra-DC bandwidth guarantees [6]–[8] cannot be adopted to solve our problem. The reason is that although these pre-determined bandwidth reservation models (either flat [6], [7] or time-varying [8]) can guarantee deadlines by providing minimum bandwidth guarantees,

Manuscript received May 20, 2015; revised November 30, 2015 and May 2, 2016; accepted July 4, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Wierman. Date of publication August 18, 2016; date of current version February 14, 2017. This work was supported in part by Hong Kong RGC under Grants ECS-26200014, GRF-16203715, GRF-613113, and CRF-C703615G; in part by the China 973 Program under Grant 2014CB340303; in part by the Ministry of Education under Project 313035; in part by the ICT R&D Program of MSIP/IITP, Republic of Korea, under Grant 14-911-05-001; and in part by the Basic Science Research Program of NRF within the MSIP, Republic of Korea, under Grant 2013R1A1A1076024. Part of this work was presented in ACM EuroSys'16 [37].

H. Zhang, K. Chen, and W. Bai are with The Hong Kong University of Science and Technology, Hong Kong (e-mail: hzhangan@cse.ust.hk; kaichen@cse.ust.hk; wbaiab@cse.ust.hk).

D. Han is with the Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea (e-mail: dongsuh@ee.kaist.ac.kr).

C. Tian is with Nanjing University, Nanjing 210000, China (e-mail: tianchen@nju.edu.cn).

H. Wang is with the University of Toronto, Toronto, ON M5S, Canada (e-mail: wh.sjtu@gmail.com).

H. Guan is with Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: hbguan@sjtu.edu.cn).

M. Zhang is with Alibaba, Hangzhou 311121, China (e-mail: aegiszhang@gmail.com).

Digital Object Identifier 10.1109/TNET.2016.2594235

they cannot fully utilize the WAN bandwidth due to their inflexibility (§III).

In this paper, we introduce DNA, a Deadline-based Network Abstraction, tailored for inter-DC WANs. DNA allows tenants to explicitly express what they want from the network in terms of the data volume and the deadline by which it must be delivered. Note that DNA allows bandwidth allocation for a single request to change over time as long as the total transfer volume is kept. Such intrinsic malleability enables providers to schedule the scarce WAN bandwidth in a more flexible and efficient way based on network conditions. Providers can now arrange *when* and *how much* data to transfer to achieve better multiplexing and to ensure higher network utilization.

We develop a system, Amoeba, that implements DNA in a scalable manner. Amoeba employs a temporal-spatial allocation algorithm for on-line admission control, and our algorithm strikes a good balance between scalability and optimality: it achieves $30\times$ speedup in terms of allocation time at the expense of sacrificing 3% in performance compared to a global optimal strategy. Amoeba further considers a series of practical design and implementation issues, *e.g.*, how to handle network dynamics and be robust to failures and traffic mispredictions. Finally, we discuss a simple pricing model to encourage tenants to reveal their authentic requirements under Amoeba.

In short, this work makes the following contributions:

- Using measurements of a production Inter-DC WAN and simulations, we reveal that the current Inter-DC WAN is insufficient to guarantee deadline-sensitive Inter-DC transfers.
- We introduce DNA, a deadline-based network abstraction tailored for inter-DC WANs, and develop Amoeba, a system that implements DNA. We deploy Amoeba on a small testbed emulating a 6-site inter-DC WAN, and evaluate our design using testbed experiments as well as large-scale simulations with realistic inter-DC WAN topologies.
- Our evaluation shows that Amoeba accommodates 15% more transfer requests with deadlines guaranteed than state-of-the-art solutions, while achieving 60% higher network utilization. Using a simple pricing model, this directly translates to 40% more revenue for the provider.

II. Measuring an Inter-DC WAN

While prior work [1], [2], [9] describes how TE is done in private clouds, very little is known about how public clouds perform. To get a sense of the quality of service of public clouds, we perform measurements on Amazon AWS intra- and inter-DC networks. We choose 6 DCs to measure: Virginia (US east), Oregon (US west), Ireland (Europe), S.Paulo (South America), Tokyo (Asia) and Sydney (Oceania). In each DC, we choose 3 machine types whose network performance metrics are labeled *low*, *moderate*, and *high*.

Our measurement results show the performance heavily depends on rate limiting, and varies significantly over time and across DCs.

Rate Limiting: We measure the total TCP throughput when increasing the number of TCP flows between each pair of VMs from 1 to 15. To observe the difference between intra- and inter-DC traffic, we vary the VM locations. We first place all VMs in the same DC (Virgina). Then, one VM in each pair is moved to Ireland. Figure 1 shows the aggregate throughput between VMs of the *high* network performance type. Similar patterns are observed in other types. We make two observations (which have been confirmed with Amazon engineers):

- *Per-VM rate limiting:* The bandwidth is capped at the same limit for both Intra-DC and Inter-DC (while different VM types have different rate limits). As shown in the figure, the cap for *high* performance VM type is around 1000Mbps.
- *Additional per-flow rate limiting for Inter-DC transfers:* The results in Figure 1 suggest that inter-DC traffic is rate-limited on a per-flow basis. At the beginning, the total throughput increases almost linearly to the number of flows, but eventually reaches the per-VM rate limit. This is not a consequence of TCP's per-flow fairness because the total throughput stabilizes only after a specific number of TCP flows. We have also verified that the observed per-flow rate limiting is not due to a small receive window. For example, the throughput remains the same when we double the TCP receive buffer.

WAN Performance Variability: Even though strict rate limiting is in place, inter-DC WAN performance significantly varies across DCs and over time. We measure the throughput from VMs in Virginia to VMs in the other five DCs respectively. Table I shows the maximum (stable) throughput over a 5 minute interval for each VM pair. We find that the throughput varies between different DCs by a factor of up to 2.8.

Moreover, we measure the throughput from VMs in Virginia to VMs in the other five DCs every 5 minutes over a total period of 35 hours. For each pair of VMs, we have 420 measurement points. Table II shows the ratio between the 95th percentile value over 5th percentile value over the

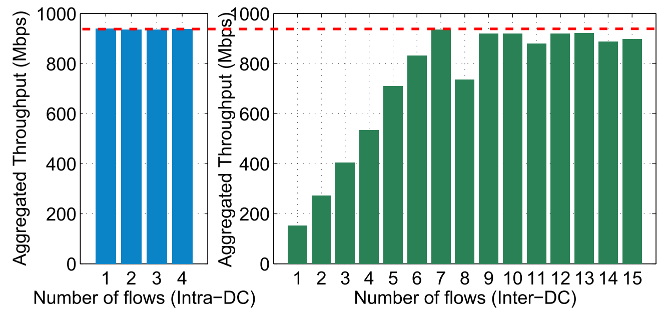


Fig. 1. Aggregate throughput between VMs of high network performance type.

TABLE I
THE THROUGHPUT OF INTER-DC FLOWS MEASURED FROM VIRGINIA (Mbps)

Type \ Region	Oregon	Ireland	S.Paulo	Tokyo	Sydney
Low	61	58	47	27	29
Moderate	180	150	106	82	69
High	296	223	182	126	107

TABLE II
THE INTER-DC WAN PERFORMANCE VARIABILITY ON BANDWIDTH AND TIME (95TH PERCENTILE VS 5TH PERCENTILE RATIO)

Variate \ Region	Oregon	Ireland	S.Paulo	Tokyo	Sydney
Bandwidth ratio	5.05	2.59	2.13	4.01	4.12
Transfer time ratio	2.67	1.43	1.39	1.97	2.41

35 hours, which varies from 2 to 5. The largest variability occurs between Virginia and Oregon. One possible cause of such variability is congestion in inter-DC WAN. However, we do not observe such high variability for intra-DC VM pairs. To further quantify the consequence of inter-DC WAN bandwidth variability, we simply transfer 1GB data between each VM pair at different time and measure the variations in completion time. The measurement in Table II shows that the variation can be as large as 2.67×. This suggests that it is difficult to ensure timely data delivery for traffic with deadline.

III. BACKGROUND AND MOTIVATION

Deadlines: The nature of many DC applications has imposed hard or soft deadlines to a large amount of inter-DC WAN traffic [1]–[3]. Deadline is important for inter-DC transfers. The main reason is that the total demand for inter-DC transfers typically far exceeds the available capacity. Many online services and applications like search, email, cloud storage etc., want geo-replication to improve performance (closer to users) and reliability (robustness against single-DC failure). Given this, cloud providers set different data replication SLAs (or deadlines) for different applications based on factors such as their delay tolerance and price (paid by customers).

Typical data transfer sizes between DCs range from tens of terabytes to petabytes; deadlines range from an hour to a couple of days [4]. For example, a web search application must update and propagate a new index once every 24 hours across DCs. A web document application must geo-replicate user data once every 2 hours to ensure that only the changes in the most recent 2 hours could be lost due to single DC outage. A key characteristic of such transfers is that they are elastic to bandwidth allocation as long as they complete before the deadlines. Missing deadlines will violate the application SLAs and greatly degrades application performance.

However, state-of-the-art solutions and the current practice, such as rate limiting, TE [1], [2], [4], and network virtualization approaches [6]–[8], are all insufficient when handling deadline-based Inter-DC transfers.

Public Inter-DC Rate Limiting Does Not Respect Deadlines: Rate limiting provides isolation among flows, but it is far from deadline guarantee. Even with rate limiting, the inter-DC transfer time is highly variable, as shown in our AWS measurements. Meeting deadlines requires fine-grained service differentiation. However, the current practice does not differentiate among different traffic classes.

Private Inter-DC TE Techniques Do Not Guarantee Deadlines: SWAN and B4 take a TE approach to improve the inter-DC WAN network utilization. They consider traffic characteristics and priorities (e.g., interactive > elastic > background) to enhance application performance. However, such prioritization is too coarse-grained and does not guarantee any specific transfer deadlines. Because there exists no interface for tenants to specify their transfer deadlines, and the provider has no way to honor them. In our evaluation (§VIII-D) we find that a large portion of transfers will miss their deadlines in SWAN.

Tempus [4] is deadline-aware and promises each request a maximal fraction of transfer before deadline without guaranteeing the completion, especially when demand exceeds the network capacity. However, for many applications, partial data transfer is useless as the applications move forward up on the completion of last byte of the last flow. As a result, this paper focuses on how to fully utilize the WAN bandwidth to guarantee the completion of as many transfers as possible before deadlines.

Applying Solutions for Intra-DC to Inter-DC Are Insufficient to Ensure High WAN Utilization: The bandwidth guarantee provided by virtual network abstractions [6]–[8], such as the hose model, supports transfer deadlines by guaranteeing minimum bandwidth. However, when applied to inter-DC WAN, they are insufficient to fully utilize the WAN bandwidth. The reason is that these pre-determined bandwidth reservation models (either static [6], [7] or time-varying [8]) are less flexible than the deadline based reservation. They provide fixed bandwidth guarantees over time while our design focuses on guaranteeing the total transfer volume given a deadline. Their models place a more stringent requirement at the admission time, while our model is more flexible because the bandwidth allocation can change over time as long as the total volume is delivered within the time limit. In our evaluation (§VIII-C), we find that pre-determined bandwidth reservations under-utilize the WAN resources, leaving many transfer requests unsatisfied.

IV. DEADLINE-BASED ABSTRACTION

The overarching goal of our work is to seek a user-provider interface and a mechanism to fully utilize the expensive WAN bandwidth to meet deadlines for as many transfers as possible. Realizing this needs an abstraction satisfying two objectives:

1) *Expressive specification:* The abstraction must allow tenants to easily express their deadline requirements in an explicit fashion to ensure application-level SLAs [10].
2) *Provider flexibility:* The abstraction must provide flexibility in provider's resource allocation. Leveraging its flexibility, the provider is then able to maximize the utilization of the expensive inter-DC WAN, and at the same time accommodate as many deadline transfers as possible.

To this end, we present DNA, an explicit deadline-based network abstraction, that allows the tenants to directly express their transfer deadlines.

Transfer: A transfer represents a tenant's data delivery demand from a source DC to a destination DC. Note that this captures a tenant-level aggregate demand between a pair of DCs. Scheduling individual flows within a tenant is handled by tenants, which is not the fcous of this paper. A transfer, T, is specified as a tuple $\{src, dst, Q, ts, td_1, td_2\}$, where src and

dst are the source and destination DCs, Q is the data volume, ts is the starting time, and(td_1, td_2) captures the deadline, either hard or soft. Specifically, td_1 represents the completion time before which the transfer suffers no utility loss, and after td_1, the utility degrades gradually to 0 at time td_2. Note that, if $td_1 = td_2$, it indicates a hard deadline. A similar model has been adopted in Tempus [4] as well.

Request: A tenant may have multiple co-related transfer demands across many DCs. For example, when running MapReduce as a single geo-distributed operation across DCs [11], multiple shuffle transfers from several mappers to a reducer are barrier-synchronized, and the completion of a single transfer does not improve the job completion time. To this end, DNA allows tenants to specify such a demand by submitting a request $\mathbf{R} = \{T_1, \cdots, T_n\}$, where T_is are transfers with a same deadline requirement $[td_1, td_2]$. The provider accommodates all transfers of a request in an atomic fashion.

V. AMOEBA

In this section, we introduce Amoeba, a system that implements DNA. We set up the following goals for Amoeba.

- **High WAN utilization & acceptance rate:** The system must fully utilize inter-DC WAN bandwidth to maximize the acceptance rate of tenant requests with deadlines, which is also the chief design goal of this paper.
- **Ensure coexistence:** The system must work with all types of traffic. Interactive traffic must be delivered without any delay, while background traffic is served in a best-effort manner.
- **Handle dynamics:** The system must be able to handle network dynamics and be robust to failures and mispredictions in interactive traffic. Temporal variations in interactive traffic demand and network failures are the major sources of dynamic events that the system must deal with.
- **Scalability & deployability:** The admission control decision must be made in near real-time upon request. The enforcement of delivery schedule must also be done in a scalable fashion to support many transfers and to scale up to tens of DCs. For practical deployment, the system must not require modification to existing network devices.

A. System Overview

In general, Amoeba implements a two-level bandwidth sharing policy. First, priority classes are enforced (*i.e.*, interactive > deadline transfers > background) and bandwidth is allocated in strict precedence across these classes. Second, within the deadline transfer class, bandwidth is scheduled to meet the deadlines of the transfer requests.

Figure 2 illustrates the system architecture of Amoeba, which contains a logically centralized *controller* and *site brokers*. The central controller is the core of Amoeba and orchestrates all network activities.[1] To be fault-tolerant, the controller is replicated across multiple DCs, and one of them is elected as master using distributed consensus protocols such as Paxos [13]. The controller maintains global information about the network bandwidth and interactive traffic demand, and performs the spatial-temporal resource allocation (§V-B) for deadline transfers using residual bandwidth left over by interactive traffic. A site broker, located in each DC, is a local representative. It predicts and reports interactive traffic demand for a local DC to the central controller periodically, and coordinates the bandwidth enforcement to realize the decision made by the controller. Note that the admission control discretizes time into timeslot for bandwidth allocation. Meaning, bandwidth allocation is fixed within a timeslot, but can vary across different timeslots. This way, timeslot also represents the maximum time that a newly arriving request has to wait before starting to transmit. We set the timeslot to 3-5 minutes in our implementation to achieve a reasonable tradeoff between performance and overhead, similar to SWAN [1].

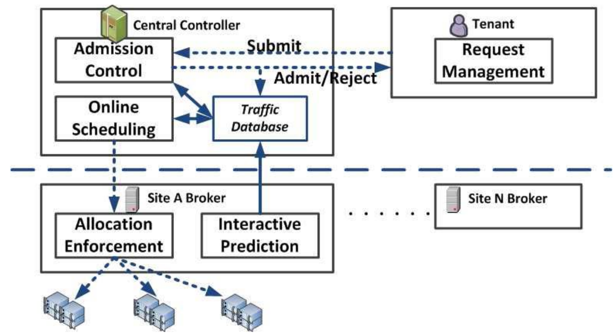


Fig. 2. System model.

Amoeba works as follows. When a new request arrives, the controller quickly determines if the request can be admitted in an online fashion (§V-B.1). The design of our spatial-temporal resource allocation also considers handling practical system factors, such as mispredictions (§V-B.2) and failures (§V-B.3). For each accepted request, before the beginning of each timeslot, the controller will inform the site brokers of the actual bandwidth allocated to each request and the corresponding path. The site brokers, in turn, enforce this via host/hypervisor-level rate limiting and explicit routing path control (§V-B.4).

B. Spatial-Temporal Allocation

1) Admission Control: Similar to bandwidth guarantee services provided in intra-DC networks [6]–[8], the admission control of Amoeba is performed in a first-come first-served (FCFS) manner.[2] To achieve high WAN utilization and high acceptance rate of deadline transfers, the admission control algorithm tries best to accept each incoming request without preempting the already admitted requests. To make effective online admission decisions, the key to our admission control algorithm is to balance scalability and optimality. On

[1] The control latency introduced by the centralized control is acceptable for large transfers, and therefore centralized resource allocation is widely adopted for large transfers in inter-DC WAN recently [1], [4], [9], [12]. Amoeba follows this trend.

[2] As a result, users should submit requests as soon as they are clear about their demand, in order to maximize the possibility of getting accepted.

one hand, the algorithm can be fast if we simply assume all the previous request schedules are fixed and perform allocation on the new request with the residual bandwidth. However, this is sub-optimal. As we will show in §VIII-F, Amoeba can bring 7%-12% performance improvement over such a solution. On the other hand, the algorithm can be optimal[3] if for any incoming new request, all existing requests, together with the new one, are rescheduled. However, although polynomial time solvable, it is still time-consuming. As we will show in §VIII-E, such an algorithm takes tens of seconds per allocation, and the time cost increases dramatically as flow arrival rate increases. Furthermore, we note that the *all-or-nothing* nature of guaranteeing transfer completion in Amoeba makes it hard to optimize as it cannot be captured with a linear constraint. Thus, the optimization framework developed for fractional allocation in Tempus [4] cannot be directly adopted for Amoeba.

Our algorithm seeks a tradeoff between scalability and optimality. We briefly summarize the high-level idea of our algorithm here and defer the details to §VI.

1) When a request arrives, we quickly find out a schedule with completion time t' as early as possible, assuming all previous decisions are fixed. We refer to this step as adaptive scheduling (AS, §VI-A). AS essentially tries to use residual network capacity to quickly accept a request by solving a min-cost flow problem (§VI-A). For a request, if it can be satisfied at this step (i.e., $t' \le td_1$), go to step 3; otherwise, go to step 2.
2) We try to reduce the completion time t' by rescheduling bandwidth schedules of previously accepted requests without violating their deadlines. Opportunistic rescheduling (OR, §VI-B) is designed to select a small subset of previous schedules that are most relevant to the current request and performs a cost-effective joint rescheduling. This increases the chance of reducing t' while being computationally more efficient than considering all previous requests. After performing OR, if we can at least accommodate it as a soft deadline request (i.e., $t' \le td_2$), go to step 3; otherwise, go to step 4.
3) Accept the request with a guaranteed transfer time of $t^* = max\{td_1, t'\}$. If the original request is a soft-deadline request, this step transforms it to a hard-deadline request with t^* as the guaranteed deadline. Given t^*, the central controller calculates an initial bandwidth schedule that meets this deadline (§VI-C). This initial schedule is subject to changes when handling future requests, mispredictions, and failures.
4) Reject the request. Note that a rejected request can be submitted again later (probably with a looser deadline requirement).

Our evaluations in §VIII show that our algorithm strikes a good balance between scalability and optimality. Amoeba achieves 30× speedup at the cost of sacrificing only 3% in performance compared to a global optimal strategy.

2) Handling Mispredictions: According to our experiences with production DCs and prior work [1], interactive traffic takes only a small portion of the overall Inter-DC traffic, e.g., $5\% - 15\%$. While the interactive traffic demand is bursty and highly diurnal, the average volume over a 5 minute window is relatively stable and can be largely predicted [1]. However, misprediction is inevitable. Without proper handling, it may degrade the quality of service. In particular, extra interactive traffic can preempt the bandwidth allocated to deadline transfers as interactive traffic has higher priority, which may cause accepted requests to miss their deadlines.

We address this problem by setting aside different headrooms for different timeslots proportional to how far away the timeslot is. This is motivated by our observation that the degree of misprediction may be large for a timeslot far into the future, but gradually becomes more accurate as it comes closer. For example, for the next timeslot, the headroom can be just 5% of the predicted interactive traffic, whereas for a timeslot an hour later, the headroom can be set to 15% of the predicted interactive traffic. Through this approach, we can safely accept requests for future timeslots. As time advances, the overprovisioned headroom of a timeslot can be released for accepting new requests or speeding up existing requests. To prevent resources from being wasted, we periodically run an algorithm similar to OR at the beginning of each timeslot and move allocation towards the current timeslot opportunistically.

Furthermore, interactive traffic may surge inside a timeslot. In Amoeba, the site broker is in charge of this. The basic idea is that interactive traffic can borrow bandwidth from deadline transfers whenever needed, and return in the future. More specifically, the site broker maintains a record of the interactive "debt" of each destination for each bandwidth allocation cycle (*i.e.*, 10 seconds). It keeps monitoring the interactive traffic fluctuation: if the headroom cannot absorb the interactive fluctuation to a destination, it dynamically decreases bandwidth from user request with the same destination and farthest deadline; the debts are paid back when the interactive traffic becomes lower than the headroom. Note that such debts can be transferred between timeslots so that even large interactive surges can be handled.

In addition, Tenant's demand specification can be inaccurate. Amoeba simply handles this as follows. For an over-estimated request, the over-estimated part can be reclaimed once reported, and the tenant will be charged partially for this part. For an under-estimated request, the additional demand will be treated as a new request for allocation. If the new request cannot be satisfied, the tenant will be informed and it is up to the tenant whether the transfer should continue. If not, the tenant only pays for the transferred amount.

3) Handling Failures: In Amoeba, link/switch failures can be detected and communicated to the controller by the site brokers according to the framework introduced in [1]. However, when failures happen, Amoeba may not be able to satisfy all the requests that have been accepted. In this case, Amoeba has to remove some accepted requests. However, obtaining an optimal solution (either minimizes throughput loss or minimizes the number of removed requests, while guaranteeing the deadlines of requests that are not removed) is NP-hard (shown in Appendix). Instead, we

[3]In the sense that it maximizes the possibility of accepting the request.

perform online rescheduling similar to our admission control. First, we remove all the requests that pass through the failed link, and set the residual bandwidth as the available bandwidth after failure. Then, we treat these removed requests as new requests (with their residual transfer volume) and perform admission control one by one according to their arrival times. Moreover, failures of the central controller and site broker are handled in a similar way as in [1].

4) Allocation Enforcement: We briefly introduce how Amoeba enforces the rate and path allocation decided by the controller.

Rate Enforcement: In practice, any distributed rate limiting solutions [14] can be applied to translate aggregate tenant-level allocations into flow-level allocations for practical enforcement. In our implementation, the end hosts perform per-flow rate limiting and the site brokers ensure that the sum of individual flow rates does not exceed the aggregate tenant-level allocation.[4] In addition, we note that the allocation distribution mechanism in BwE [15] system can be applied here for more fine-grained flow-level bandwidth enforcement.

Routing Enforcement: Many approaches such as source routing [16], MPLS [17], and OpenFlow [18] can enforce explicit path control. However, source routing is usually not supported by the data center switch hardware; while MPLS needs a signaling protocol, i.e., Label Distribution Protocol, to establish the path, which is typically used only for traffic engineering in core networks instead of application-level or flow-level path control. OpenFlow can establish fine-grained routing paths by installing flow entries in the OpenFlow switches via the controller, however the generic OpenFlow TCAM rules in commodity switches are limited to a small number, typically 1–4K [1]. To overcome this limitation, recent solutions such as SWAN [1] dynamically change the set of paths available in the inter-DC network at different times through dynamic flow table configurations, which introduce non-trivial implementation overhead and performance degradation. In Amoeba, we leverage XPath [19], a simple, scalable and readily-deployable way to implement explicit path control. We elaborate the enforcement module and its implementation in §VII.

C. Pricing Model

We discuss a simple pricing model to encourage tenants to reveal their authentic transfer requirements to the provider, *i.e.*, class, volume, and deadlines.

Encouraging true class declaration can be done by simply setting a "higher price for better service" policy. Interactive traffic is assigned the highest priority, thus deserves the highest unit price (price per GB) p_{int}. Background traffic receives a best-effort service, and should be charged at the lowest unit price p_{bck}. Deadline transfer lies in-between, and its unit price $p_{ddl}(\cdot)$ varies depending on both volume and deadline. To distinguish different classes, we simply set $p_{int} \geq p_{ddl}(\cdot) \geq \alpha * p_{int}$ and $\beta * p_{ddl}(\cdot) \geq p_{bck}$,[5] where $\alpha, \beta \in (0, 1)$ can be flexibly adjusted according to the supply-demand relationship.

Encouraging true volume declaration is also simple. For under-claimed requests, the extra volume beyond requested is handled as background traffic in a best-effort manner; For over-claimed requests, the unused bandwidth can be exploited by background traffic, but the tenant should pay for the entire volume claimed.

Encouraging true deadline declaration is necessary: consider two requests transferring the same amount of data from site A to site B with deadlines of 2 timeslots and 20 timeslots respectively; although they transfer the same volume, the pressure they exert to the network is different. Thus, the charging of the deadline traffic should depend on both volume Q and deadline guaranteed t^*. For this, we can use the expected bandwidth $\overline{B} = Q/t^*$ as the criteria for charging, i.e., $p_{ddl}(\cdot)$ should be a non-decreasing function of $\overline{B}$. Note that users may reduce their costs by splitting their requests into smaller chunks and use the same deadline for all chunks. However, it is risky to do so because some chunks may be rejected. Moreover, a lower bound on the smallest chunk size can be set in order to regulate user requests.

Moreover, such a pricing model also helps to substantiate the benefit brought by our deadline guarantee service. In our evaluation (§VIII-F), we propose two simple pricing functions based on the above discussion, and evaluate the corresponding provider revenue gain over the fixed bandwidth abstraction (§VIII-C). The results show that the 60% higher network throughput brought by Amoeba directly translates to around 30% higher provider revenue.

VI. Algorithm Details

We elaborate the algorithm in §V.

A. Adaptive Scheduling (AS)

AS tries to embed a new request $\mathbf{R}$ into the WAN substrate along two dimensions, time and space, without changing the bandwidth schedules of existing requests. To do so, we keep track of the residual bandwidth on each link, and denote the residual bandwidth of link l at time t as $R^l(t)$. To determine the feasibility and routing paths, we solve a min-cost flow problem on a temporally expanded flow graph.

Creating the Expanded Flow Graph: First, we construct a flow graph G by creating a virtual node for each DC at every timeslot, as shown in Figure 3. Each virtual node $DC_{n,t}$ represents a DC n at timeslot t. In each timeslot, these virtual nodes are connected to each other just as they were in the original inter-DC topology. Each link l between two virtual nodes of the same timeslot t is assigned a capacity of $c(l) = R^l(t)$.

Second, we add a pair of super nodes S_i and D_i for each transfer T_i in $\mathbf{R}$, and connect S_i to all source DCs $DC_{j,t}$ with timeslot t inside T_i's possible transmission period $[ts, td_2]$. We then connect the destination DCs to D_i in a similar way.

Finally, we add a source node S and a sink node D in the graph, and connect all S_is and D_is to S and D respectively. The link capacity $c(l)$ of each link is set to Q_i. Such time

[4] An alternative way is to rate limit the aggregate tenant-level allocation on switches. However, the number of transfers that can be rate limited is bounded by the number of policers on the switch [8];

[5] We use "$\geq$" as $p_{ddl}(\cdot)$ varies and we only restrict the upper/lower bound.

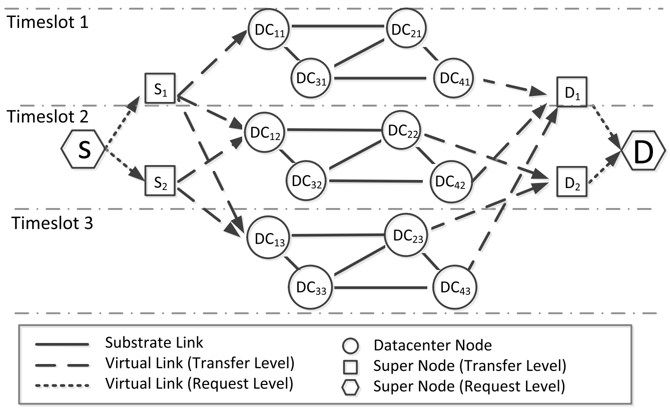


Fig. 3. Creating temporally expanded flow graph for adaptive scheduling over multiple timeslots.

expansion can be regarded as a variation of the technique introduced in [20].

Figure 3 shows an example to expand a request $\mathbf{R}$ over 3 timeslots, where $\mathbf{R}$ includes two transfers $T_1 = \{DC_1, DC_4,\ Q_1, ts = 1, td_1 = 2, td_2 = 3\}$ and $T_2 = \{DC_1, DC_2, Q_2, ts = 2, td_1 = 2, td_2 = 3\}$. T_1 is expanded over timeslot 1 to 3, and T_2 is expended over timeslot 2 to 3.

Finding the Shortest Possible Completion Time: Given G, we approximate the minimal completion time by assigning different weights to edges in the flow graph, and then solving the corresponding min-cost flow problem (problem formulation in Algorithm 1). More specifically, for each edge l from S_i to the source $DC_{j,t}$, we assign weight $w(l) = 2^{t-ts}$. Through this way, the solution to Algorithm 1 tends to pack more flows in the earlier timeslots in order to minimize the cost. The first constraint requires that the aggregated flow rate on each link does not exceed the link capacity. And the third constraint requires that for each transfer, the aggregated flow rate over different paths and timeslots should be equal to the transfer volume. Note that we use only k-shortest paths between each source-destination DC pairs as input of Algorithm 1. This reduces the number of desired paths we need to pre-install in the switches.[6] In our simulation in G-scale topology, we find that $k = 10$ already results in negligible loss on both request acceptance rate and throughput.

After AS, if the completion time $t' \leq td_1$, then the provider will accept the request with deadline td_1 directly. Otherwise, we proceed to OR, where we try to further reduce t' by rescheduling existing accepted requests.

B. Opportunistic Rescheduling (OR)

As mentioned earlier, obtaining an optimal spatio-temporal schedule over *all* existing requests is time-consuming. To achieve a near optimal schedule with affordable computation time, we design a two-step heuristics for OR: local stretching and joint rescheduling.

[6]Moreover, AS is essentially a multicommodity flow problem, and it is equivalent to the min-cost flow presentation (the LP formulation is the same) with path constraint.

Algorithm 1 Min Cost Flow Formulation on the Expended Flow Graph

Input: $\mathbf{R} = \{T_1, T_2, ...T_n\}$: A tenant request with n transfers;

$\mathbf{P_i} = \{p_1, p_2, ...p_k\}$: k-shortest paths between the DC pair of transfer $i \in [1, n]$;

$I_{p_m,l}$: 1 if $l \in p_m$; and 0 otherwise;

$c(l)$: residual link capacity for link l in the expended graph;

Output: Return the latest timeslot t' in the solution of the following problem;

$$minimize \sum_{i,t,p_m} \sum_{l \in E} w(l) \cdot f_{itp_m} \cdot I_{p_m,l}$$

s.t.

$$\forall l, t, \quad \sum_i \sum_{p_m \in \mathbf{P_i}} f_{itp_m} \cdot I_{p_m,l} \leq c(l)$$
$$\forall i, t, p_m, \quad f_{itp_m} \geq 0$$
$$\forall i \in \mathbf{R}, \sum_t \sum_{p_m \in \mathbf{P_i}} f_{itp_m} = Q_i$$

% f_{itp_m}: the allocation to the flow over path $p_m \in \mathbf{P_i}$ in timeslot t for transfer T_i;

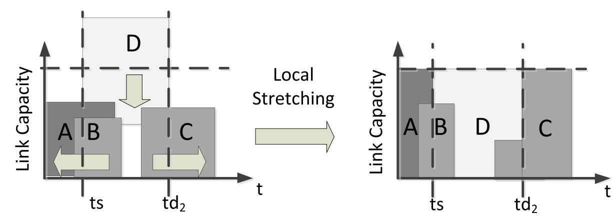


Fig. 4. Local stretching implements OR.

Local Stretching is a simple greedy algorithm. As illustrated in Figure 4, to accommodate request $\mathbf{R}$, we "shift" the bandwidth schedules of previous requests out of $\mathbf{R}$'s time window $[ts, td_2]$. This is performed on every path that $\mathbf{R}$ passes through. By local stretching, we set aside more residual bandwidth to accommodate $\mathbf{R}$. Thus, when performing AS again, it is more likely to reduce t'. If t' is still larger than td_1 after local stretching, we proceed with joint rescheduling.

Window Selection: We define a *stretching window*, $[W_{left}, W_{right}]$, that determines the set of flows involved in the stretching operation. Only the flows with starting time and deadline within this stretching window will be stretched. As shown in Eq. 1, the window is selected to be centered around and larger than the $[ts, td_2]$ of $\mathbf{R}$. The choice of window size β is a trade-off between performance and computational cost, we set $\beta = 2$ in our simulation.

$$\begin{cases} W_{left} = max(t_{current}, ts - \beta(td_2 - ts)) \\ W_{right} = td_2 + \beta(td_2 - ts) \end{cases} \tag{1}$$

Bi-Directional Stretching: For all the flows in the stretching window, forward stretching tries to stretch as much volume from $[ts, td_2]$ to $[td_2, W_{right}]$ as possible. We perform forward stretching on flows one by one in a descending order of their deadlines. And for each flow, the stretching is done by greedily reallocating its bandwidth allocation in $[ts, W_{right}]$ towards later timeslots without changing its path, as we did for flow

C in Figure 4. Backward stretching is then done on each flow in the ascending order of ts, and the procedure is similar to that of forward stretching (flow A and B in Figure 4).

Joint rescheduling is a partial optimization, in which we select some existing requests to do coordinated rescheduling together with the new one, i.e., running AS on all these requests collectively. Note that the time cost of AS is related to the number of requests. Therefore, instead of considering all existing requests, the idea is to find a subset of most relevant requests to reschedule so that the chance of further reducing the completion time t' of the new request maximizes.

Request Selection: For each transfer X_i in $\mathbf{R}$, we define a scoring metric, $S(X_i, Y_j)$, between X_i and an existing accepted transfer Y_j to estimate Y_j's rescheduling effectiveness, i.e., how much Y_j's rescheduling will help in reducing t' of X_i. $S(X_i, Y_j)$ is related to the following two factors:

1) The possibility that Y_j's traffic can be shifted out of X_i's transmission window $[ts, td_2]$. We estimate this as $\frac{td^j - ts^j}{t^+ - t^-}$, where $[ts^j, td^j]$ is the transmission range of Y_j, and $[t^-, t^+]$ is the overlapping time period of X_i and Y_j's transmission time.
2) The amount of Y_j's traffic that goes through X_i's bottleneck link. This is quantified by the amount of X_i and Y_j's traffic that goes through the same link weighted by the link's utilization[7]:

$$\sum_{t=t^-}^{t^+} \sum_{l \in p_m}^{\forall p_m \in \mathbf{P}(Y_j)} (U_l(t) \cdot I_{l,X_i} \cdot I_{l,Y_j} \cdot f_{jtp_m});$$

where $I_{l,X_i}/I_{l,Y_j}$ indicates whether link l is used in X_i/Y_j, and $\mathbf{P}(Y_j)$ is the set of k-shortest paths from src_j to dst_j in Y_j, and $U_l(t)$ is the link utilization of l.

For each transfer $X_i \in \mathbf{R}$, we select a set of n existing requests that have the highest scores, denoted as $H_n(X_i)$. We then define the set of n highly relevant requests with $\mathbf{R}$, $H_n(\mathbf{R})$, as $\cup_{X_i \in \mathbf{R}} H_n(X_i)$.

Partial Optimization: We create a new request $\mathbf{R}' = H_n(\mathbf{R}) \cup \mathbf{R}$. We run AS over $\mathbf{R}'$ on the WAN substrate where the residual link capacities are obtained by removing the requests in $H_n(\mathbf{R})$. Then, AS tries to accommodate all transfers in $\mathbf{R}'$. If it fails, we finally reject $\mathbf{R}$.

C. Bandwidth Schedule

For each accepted request with guaranteed deadline t^*, the controller calculates a bandwidth schedule that meets t^* and updates the residual bandwidth $R^l(t)$ accordingly. Note that a deadline t^* may correspond to many feasible spatial-temporal schedules, and different schedules may have different impacts on the admission control of future requests.

To increase the chance of accepting future requests using AS only (time-efficient), a heuristic is that for each new request, AS should minimize the link utilization across all involved timeslots. That is, we always try to allocate a request with shorter paths. This is realized by assigning each link with a uniform link weight of 1, and then solve the corresponding min-cost problem in Algorithm 1 with an extra constraint $t \leq t^*$. In this case, the bandwidth schedule may not always favor earlier time slots. Therefore, in our implementation, at the beginning of each timeslot if Amoeba detects available bandwidth in the current slot, it runs an OR alike heuristic to pull more traffic from the future timeslots back to the current one, in order to fully utilize the bandwidth.

[7]The intuition of using link utilization as weight is that highly utilized links are most likely to be bottlenecks of reducing the completion time of $\mathbf{R}$. And the score should take that into consideration.

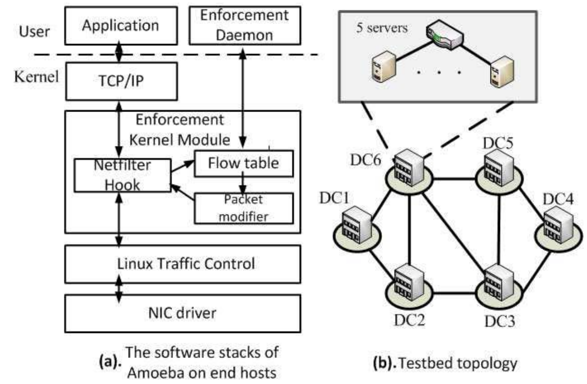


Fig. 5. Implementation & Testbed.

VII. Implementation

Our prototype consists of the controller, site brokers and enforcement modules. We implement our algorithm in §VI for the controller and site brokers. For routing enforcement we use XPath [19] which enables explicit routing path control, and for bandwidth enforcement we leverage the Linux Traffic Control (TC).

Amoeba's enforcement module consists of a kernel module and an enforcement daemon, as shown in Figure 5. The enforcement daemon communicates with the kernel module via iotcl. The enforcement daemon interacts with the site brokers to obtain VM-level rate limits and the corresponding path IDs. It is responsible for managing the flow table, such as inserting, updating or deleting flow marking rules. The kernel module is located between TCP/IP stack and the Linux TC module. The kernel module intercepts all outgoing packets, it modifies the nfmark field of `sk_buff` (netfilter mark, a field which can be used for packet marking) after looking up the flow table, which will be used as the identifier for rate limiting in TC. Meanwhile, it also modifies the destination IP of `sk_buff` into the corresponding path ID in order to enforce the routing path. Then these packets are directed to TC for rate limiting. In virtualized environments, we envision that the kernel module runs in the hypervisor and Dom0 to control all traffic going through physical NICs.

To perform distributed per-flow rate limiting on end hosts, we leverage the Hierarchical Token Bucket (HTB) in TC. We use the two-level HTB: the leaf nodes enforce per-flow rates and the root node classifies outgoing packets to their corresponding leaf nodes based on nfmark field which has been modified by Amoeba kernel module.

For routing enforcement, we perform explicit path control using Xpath as mentioned in §V-B.4. First, the XPath manager (which is integrated into the Amoeba controller) explicitly identifies each desired end-to-end path with a path ID (in the format of a 32-bit IPv4 address), and compresses these IDs into IP LPM (Longest Prefix Match) tables. These routing tables are then pre-installed into the corresponding inter-DC switches, thus no further dynamic reconfiguration is needed. Second, each site broker maintains a path-to-ID mapping table, and translates the routing decision (made by the controller) into the corresponding path IDs for each request. Finally, given the path IDs, we leverage NAT at each sender end-host to translate the raw destination IP into the desired path ID for each packet, thus explicitly specifies the routing path.[8]

To make sure that the overhead of Amoeba's enforcement module is negligible, we measure the extra CPU usage it introduces. We generate more than 900Mbps of traffic with more than 100 flows on a Dell PowerEdge R320 server with 8GB of memory and a quad-core Intel E5-1410 2.8GHz CPU. The extra CPU overhead introduced is around 3% compared with the case that Amoeba's enforcement module is not used. The throughput remains the same in both cases.

VIII. Evaluation

In this section, we answer five specific questions through extensive evaluations:

- **Does Amoeba provide deadline guarantees for inter-DC transfers in practice?** In §VIII-B, we show that Amoeba guarantees deadlines for all accepted requests, and all flows complete within the scheduled time given by the controller. We also show that Amoeba ensures this while achieving no worse (much better in some cases) network utilization than the state-of-the-art solutions.
- **How does Amoeba compare with existing solutions that provide a fixed minimum bandwidth guarantee?** In §VIII-C, we show that Amoeba achieves up to 60% higher utilization while satisfying up to 15% more requests with deadlines.
- **How does Amoeba compare with existing SDN-based inter-DC TE solutions?** In §VIII-D, we show that Amoeba accommodates 60% more requests with deadlines while achieving similar levels of utilization.
- **How effective is Amoeba and how scalable is it?** In §VIII-E, we show that our heuristics make reasonable tradeoffs. They achieve 30× speedup at the cost of sacrificing only 3% of network utilization compared to a strategy which tries to find an optimal schedule.
- **How do Amoeba's components contribute to performance and computational cost?** In §VIII-F, we show the performance breakdown of each component, present some results on misprediction and failure handling, and analyze the effect of supporting soft deadlines. We also proposed two simple pricing functions and evaluated the corresponding revenue gain brought by Amoeba.

[8]And we translate the path ID back to the raw destination IP at each receiver to avoid confusing the network stack and application.

A. Evaluation Methodology

We evaluate Amoeba with both testbed experiments and simulations. On the testbed, we show the overall performance of Amoeba and also demonstrate that the obtained schedules from our algorithm can be effectively enforced. Through simulations, we unravel the details of Amoeba across different settings, topologies, and workloads.

Testbed Setup: We build a small testbed with 30 servers to emulate an inter-DC WAN with 6 DCs as in Figure 5. Each DC has 5 physical servers and a Pronto 3295 48-port Gigabit Ethernet switch. The switch has installed PicOS 2.0.4 system which supports both Layer2/Layer3 and OpenFlow. Each inter-DC link is emulated using one physical 1G link. The central controller locates in DC 1. We add delays to emulate the WAN environment, and the delays are generated based on the speed of light and geometric distances between randomly selected DC sites in the G-Scale topology [2]. The OS of each server is Debian 6.0 64-bit version with Linux 2.6.32 kernel. Each server has a qual-core Intel E5-1410 2.8GHz CPU, 8G memory, 500GB hard disk with 1G Ethernet NIC. The CPU, memory or hard disk is not a bottleneck in our testbed evaluation. We use iperf to generate TCP flows.

Simulation Setup: We simulate two production inter-DC WANs: (i) G-Scale, Google's inter-DC WAN with 12 DCs and 19 inter-DC links [2], and (ii) IDN, with 40 DCs, each connected to 2-16 other DCs [1]. We assume that each link has a uniform capacity of 160 Gbps. Interactive traffic on each link is randomly generated between 5% and 15% of the link capacity for each timeslot, which is also assumed to be the predicted interactive workload. Based on such predicted workload we leave extra headroom and keep updating the headroom as we discussed in §V-B.2. Each run simulates 150 5-minute timeslots (about 12 hours). We report the average of 5 runs.

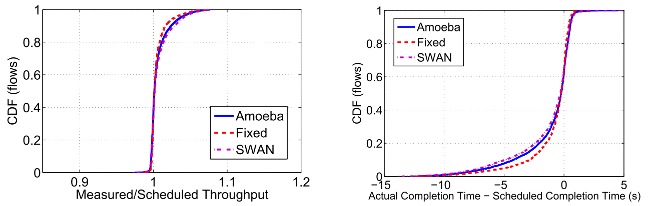
Metrics: We measure three performance metrics: network utilization (i.e., the average link utilization of interactive traffic and all accepted requests), request acceptance/rejection rate, and network throughput.

Workload: The inter-DC deadline traffic demand is generated with the following parameters:

- *Request arrival time* is modeled as a Poisson process with arrival rate λ per timeslot.
- *Deadlines:* The maximum transfer time without utility loss, i.e., $td_1 - ts$, is modeled under exponential distribution with a mean of one hour, and the deadline td_1 can be calculated accordingly. We consider soft deadlines in §VIII-F.
- *Transfer volume:* As transfer volume and deadline are related, we set up its value in a way that $\frac{Transfer\_volume}{Deadline}$ (i.e., average transfer throughput) follows an exponential distribution with a mean of 20 Gbps.
- *Number of transfers per request:* each request contains 1-6 transfers.

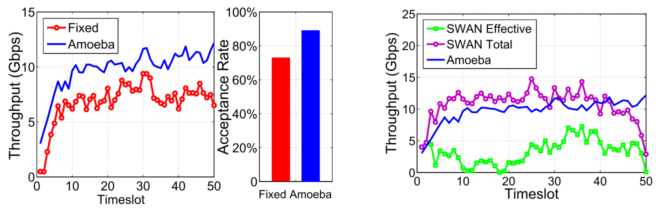
B. Testbed Experiments

We perform experiments on our testbed for a duration of 50 3-minute timeslots (2.5 hours). At any given time,



(a)Throughput deviation (b)Completion time deviation

Fig. 6. Deviation between schedules and testbed results. (a) Throughput deviation. (b) Completion time deviation.



(a) (b)

Fig. 7. Experiment results. (a) Amoeba *vs*. *Fixed*. (b) Amoeba *vs*. SWAN.

the actual traffic per DC-pair is composed of 20 to 200 TCP flows. Our experiment results demonstrate: 1) Amoeba guarantees deadlines by generating effective bandwidth schedules and accurately enforcing the schedules at each timeslot; and 2) Amoeba delivers higher utilization/throughput compared to others, including solutions that provide fixed minimum bandwidth (*Fixed*) and SDN-based TE (SWAN).

To demonstrate that Amoeba performs effective bandwidth schedules and accurate real-time enforcement, we show the difference between the scheduled bandwidth allocation and the throughput actually measured in the experiment in Figure 6 (a). We observe that for more than 95% of requests, the difference is less than 5%. In addition, Figure 6 shows that for the majority of flows, the completion times on the testbed matches their schedules (note one flow lasts at least for one timeslot). Such result indicates that a sub-second level inter-DC delay has a negligible impact on the bandwidth allocation and deadline guarantee of Amoeba, as the schedule is at the granularity of 3-minute timeslot. Furthermore, we note that the throughput measured is slightly higher (and the completion time is slightly smaller) than scheduled. One possible reason is that the completion time measured by iperf is the time to copy data from user space to kernel space at sender side, which is smaller than that from user space of sender side to user space of receiver side (i.e., the actual completion time), especially for short flows.

We further compare Amoeba with two baseline algorithms (*Fixed* and SWAN) in terms of throughput/utilization in Figure 7. Figure 7 (a) shows that Amoeba achieves around 40-50% higher throughput than *Fixed*. This is mainly because Amoeba has the flexible DNA model. The higher utilization translates to higher acceptance rate. As shown in the figure, Amoeba has an acceptance rate of 89%, whereas the acceptance rate for *Fixed* is 72%. Figure 7 (b) shows the results for SWAN versus Amoeba. SWAN achieves a slightly better throughput/utilization than Amoeba. However, SWAN is deadline-agnostic and many flows miss their deadlines despite of the higher total throughput. In terms of the effective throughput (i.e., the throughput of flows that meet their deadlines), SWAN is less than half of Amoeba.

C. Amoeba vs. Fixed Minimum BW Guarantee

We compare Amoeba with *Fixed* using large-scale simulations. We generate requests with randomly selected sources and destinations in both IDN and G-scale topologies. The request arrival rate for IDN is higher because IDN is larger than G-Scale.[9] The minimum bandwidth guarantee in *Fixed* is set to satisfy the deadlines, *i.e.*, $B_{fix} = \frac{Q}{td_1 - ts}$.

Figure 8 (a) and Figure 9 (a) show the rejection rates for IDN and G-scale respectively. It is obvious that Amoeba show much better performance than *Fixed*. In both cases, Amoeba accepts around 15% more tenant requests than *Fixed* consistently. This is because Amoeba resource allocation algorithm fully takes advantage of the malleability provided by the flexible DNA model. In contrast, in *Fixed* the bandwidth reservation is pre-determined and cannot be changed during runtime, and such inflexibility leads to higher rejection rate.

Figure 8 (b)/(c) and Figure 9(b)/(c) show the network utilization and throughput. Due to the same reason as above, Amoeba outperforms *Fixed* in both topologies. In many cases, Amoeba achieves 40%-60% higher network utilization than *Fixed*. In terms of throughput, Amoeba also outperforms *Fixed* by 50%-60% in most cases.

D. Amoeba vs. Current Inter-DC TE

We compare Amoeba with deadline-oblivious TE solutions, such as SWAN [1] and Netstitcher [9], in G-scale topology. We adopt SWAN's allocation algorithm per timeslot with an objective of maximizing the throughput in the current slot. Netstitcher models the data delivery for each request as a minimum transfer time (MTT) problem [9]. We approximate its allocation algorithm for each incoming request. We define request success rate as the percentage of requests that meet deadlines. As Amoeba offers deadline guarantees, the request success rate of Amoeba equals to its request acceptance rate.

Note that we omit the comparison between Amoeba and Tempus [4]. The reason is that Tempus focuses on fairness and maximizes the minimal fraction among all transfers delivered until deadlines, but does not guarantee the completion of any transfer before deadline. When demands exceed network capacity, Tempus always tries to fairly share the limited bandwidth among all requests, leading to a very low or even 0 request success rate.

Figure 10 (a) shows the request success rates for SWAN, Netstitcher, and Amoeba respectively. As the arrival rate increases, the request success rate decreases for all three solutions. However, SWAN and Netstitcher experience a more dramatic drop than Amoeba. This is because SWAN greedily

[9]We set the arrival rate to be at most 8/50 in G-scale/IDN because the network is already saturated under such rate and higher arrival rate will not cause obvious changes in network utilization and throughput.

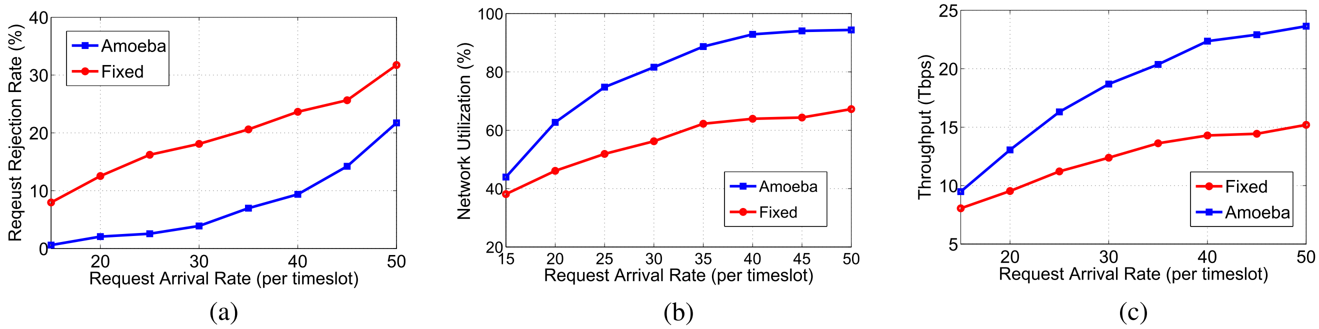


Fig. 8. Amoeba vs. Fixed minimum bandwidth guarantee in IDN topology. (a) Request rejection rate. (b) Network utilization. (c) Throughput

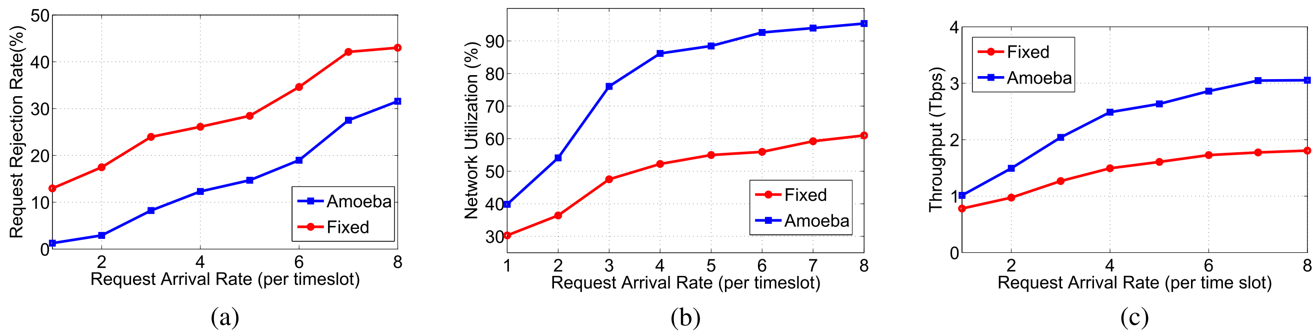


Fig. 9. Amoeba vs. Fixed minimum bandwidth guarantee in G-Scale topology. (a) Request rejection rate. (b) Network utilization. (c) Throughput.

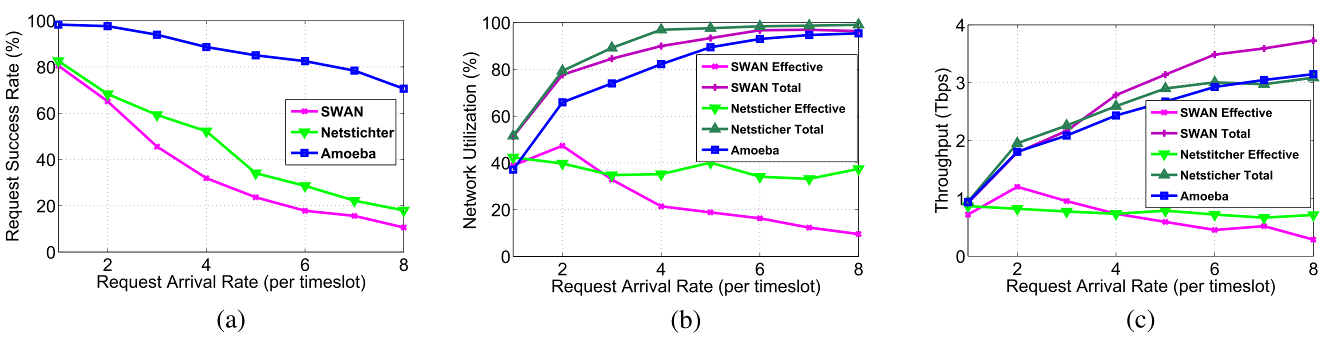


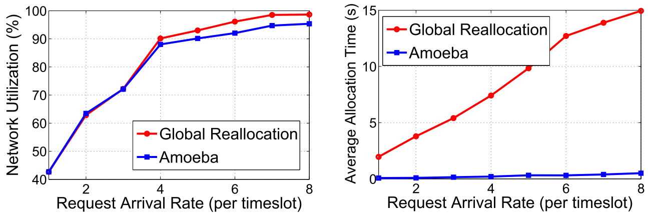
Fig. 10. Amoeba vs. deadline-oblivious TE in G-Scale topology. (a) Request success rate. (b) Network utilization. (c) Throughput.

allocates requests per timeslot without considering the deadlines, while Netstitcher only tries to minimize the transfer time regardless of the deadlines. As a result, as the arrival rate increases, more requests will miss their deadlines. We also find that the request success rate of Netstitcher is higher than that of SWAN. This is because SWAN splits bandwidth among multiple transfers and it is possible that very few of them can meet their deadlines when the number of requests is large. On the other hand, Netstitcher serves requests in a first-come first-served fashion, and thus the first few requests can always meet their deadlines.

Figure 10 (b) and Figure 10 (c) show the network utilization and throughput. In the figures, *total* network utilization refers to the network utilization of all (including partially allocated) requests, and *effective* network utilization only refers to the requests that meet their deadlines. *Total* and *effective* throughput are defined in a similar way. From the figures, we observe that the deadline-agnostic solutions achieve high total network utilization and throughput, but very low effective network utilization and throughput. This result is expected because they do not respect deadlines. In contrast, Amoeba maintains much better effective network utilization, as it has a much higher request success rate by guaranteeing deadlines.

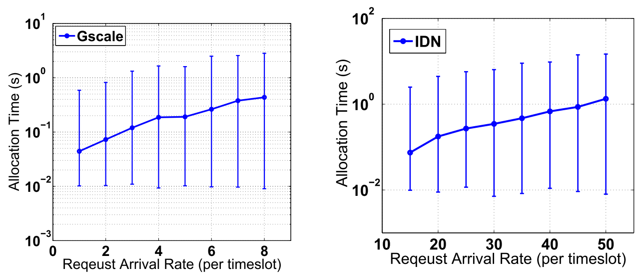
E. Effectiveness and Scalability

Effectiveness: To demonstrate the effectiveness of Amoeba, we compare it with a strawman global optimization algorithm in G-Scale. Whenever a request comes, the global optimization algorithm reallocates all previous requests using a similar formulation as AS. Figure 11 (a) shows the network utilization of Amoeba and the global optimization algorithm (we observed similar results in terms of request acceptance rate and throughput as well). We find that Amoeba performs almost the same as the global optimization algorithm under low arrival rate, and is slightly worse than it (by around 3%)



(a) (b)

Fig. 11. Amoeba vs. strawman global optimization. (a) Network utilization. (b) Average allocation time.



(a) (b)

Fig. 12. The min/mean/max Amoeba allocation time. (a) G-Scale. (b) IDN.

as the arrival rate increases. The reason behind this is as follows. First, when the arrival rate is low, both algorithms are able to accept most of the requests, thus achieving almost the same performance. Second, as the arrival rate increases, there are more requests to handle. Since Amoeba only reallocates a subset of relevant requests when handling the new ones (§VI-B), it becomes less effective than the optimal solution that performs a global reallocation. As a consequence, some requests accepted by the global reallocation may be rejected by Amoeba.

In Figure 11 (b) we can see that Amoeba achieves 30× speedup in terms of average allocation time compared to the global optimization algorithm. Note that the allocation time of the global optimization algorithm in Figure 11 (b), i.e., tens of seconds, might be acceptable for some transfer requests, however Amoeba can achieve comparable performance in a much shorter time. And it is always desirable to have shorter time in admission control for timely decision on user requests. Furthermore, as the global optimization algorithm requires reallocation of all previous allocated requests, the time cost can increase dramatically as the arrival rate increases, which eventually results in unacceptable allocation time under higher arrival rate. In this regard, the time cost of Amoeba increases much slowly as shown in Figure 11 (b).

Scalability: We quantify Amoeba's scalability by measuring the allocation time per request in both G-Scale and IDN. The simulation is performed on a server with 384G memory and 2 quad-core 2.8GHz Xeon CPUs.

As shown in Figure 12, the average allocation time is less than 0.5 second in G-Scale and less than 1.5 seconds in IDN, and the maximal allocation time is only about 6 seconds. Another observation is that the allocation time increases as the arrival rate increases. This is because most requests can be easily and quickly allocated using only AS under low arrival rate. However higher arrival rate brings higher network utilization and higher request rejection rate. As a result, the average allocation time increases since OR is more frequently invoked.

F. Component-Wise Benchmark

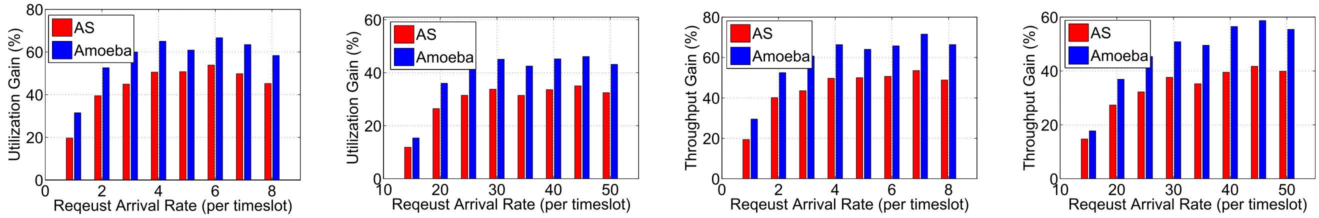
Performance Breakdown: We use simulations to show the benefits of AS and OR individually. Figure 13 shows the gain of AS and Amoeba (AS+OR), and we use *Fixed* as the baseline. From the figure, we can see that AS contributes around 20%-40% performance gain over *Fixed* under most arrival rates, and OR can bring additional 7%-12% performance gain.

We further check the benefits of two operations in OR, i.e., local stretching and joint rescheduling. Figure 14 shows the results. Local stretching brings around 2%-4% utilization gain and 2%-3% improvement in acceptance rate (not shown in the figure), at the cost of increasing allocation time from 0.02 to 0.1 second on average. Joint rescheduling brings 4%-8% utilization gain at the cost of increasing allocation time from 0.1 to 0.4 second on average.

Mispredictions and Failures: To show the benefit brought by Amoeba's evolving headroom (EH) when dealing with mispredictions, we simulate a variant of Amoeba with a fixed headroom (FH). Figure 15 (a) shows that, with EH, Amoeba can set aside more bandwidth for deadline transfers, and thus results in 2% higher throughput. Note that this is not a small number considering that the total interactive traffic only accounts for 5%-15% of the link capacity.

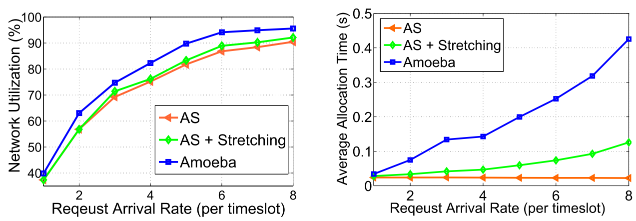
To test Amoeba under link failure, we use G-Scale and randomly fail an inter-DC link. Since an inter-DC link can contain multiple physical fibers [1], we consider two cases: 100% link capacity loss and 50% link capacity loss. We run the failure handling procedure described in §V-B.2 and calculate the survival rate, which is the portion of successfully reallocated requests over all affected requests. Figure 15 (b) shows the result. We observe that almost all the affected requests can be successfully reallocated under low arrival rate. Moreover, Amoeba still achieves over 80% and 90% survival rate respectively under high arrival rate. Figure 15 (b) also shows the time cost of failure handling, which grows as the arrival rate increases. We argue that a time cost of 10s of seconds is acceptable as inter-DC link failure seldom happens and usually takes minutes to days to repair [21].

Effect of Soft Deadlines: So far we assume that all requests have hard deadlines. We now extend hard deadlines to soft deadlines, and we use G-Scale in our simulation. In Figure 16, the red curve shows the performance when all requests are assigned with hard deadlines, and the blue curve shows the performance with soft deadlines (20% extension based on the hard deadlines). We observe that under low request arrival rates, soft deadlines bring better network utilization and throughput. This is because a rejected request with a hard deadline still has a chance to be accepted with a looser soft deadline. However, under high arrival rates, soft deadlines do not always perform better than hard deadlines in terms of throughput as the link capacity are eventually exhausted in both cases.



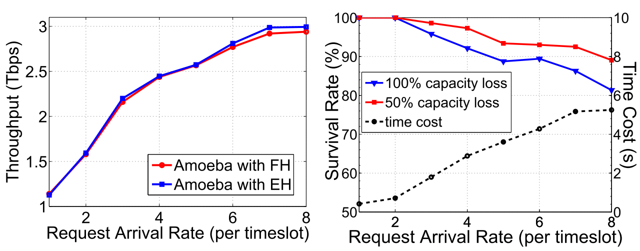
(a) (b) (c) (d)

Fig. 13. The benefit of the two components of Amoeba. (a) Utilization gain (G-Scale). (b) Utilization gain (IDN). (c) Throughput gain (G-Scale). (d) Throughput gain (IDN).



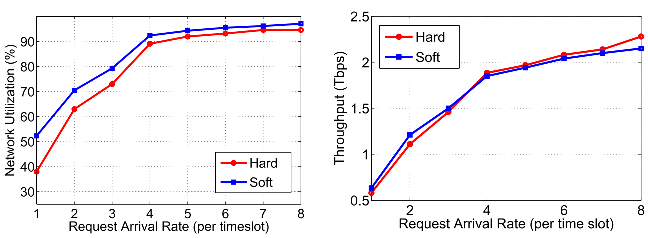
(a) (b)

Fig. 14. The incremental benefit of local stretching and joint rescheduling. (a) Network utilization. (b) Average allocation time.



(a) (b)

Fig. 15. Mispredictions and failures. (a) Benefit of EH. (b) Amoeba under link failure.

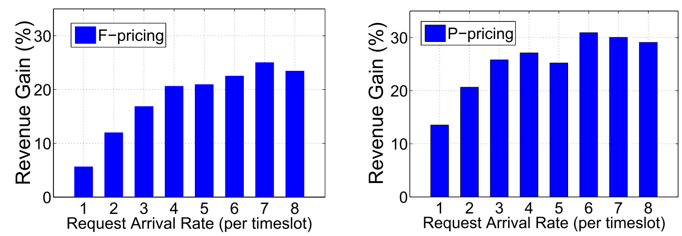


(a) (b)

Fig. 16. Soft-deadline vs. hard deadline. (a) Network utilization. (b) Throughput.

Provider Revenue: We propose two simple pricing functions and evaluate the revenue gain of Amoeba over the fixed bandwidth solution under the G-Scale topology.

- F(ixed)-pricing: We fix the unit price p_{ddl} for all deadline traffic, and set $p_{ddl} = 0.5 * p_{int}$ and $p_{bck} = 0.5 * p_{ddl}$.
- P(ropotional)-pricing: We set $p_{bck} = 0.1 * p_{int}$ and p_{ddl} lies in between. More specifically, we set $p_{ddl} = p_{bck} + min(1, \overline{B}/B_{up}) * (p_{int} - p_{bck})$. This way, p_{ddl} grows proportional to the expected bandwidth $\overline{B}$. Here B_{up} is a constant and we set it to the 5th percentile highest $\overline{B}$ among all requests.



(a) (b)

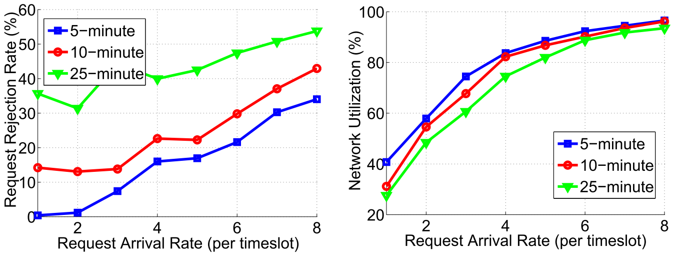
Fig. 17. The revenue gain of Amoeba over *Fixed*. (a) F-pricing function. (b) P-pricing function.

The design simply follows some principles discussed in §V-C, and we note that a more sophisticated modeling can be an interesting future work.

To calculate the provider revenue, we assume that the bandwidth left-over by deadline traffic is fully saturated by background traffic. Also, note that neither background nor interactive traffic is expressed as requests. In order to simulate the throughput of background and interactive traffic based on the corresponding network utilization, we randomly select source and destination for background and interactive traffic with an average hop count of two. Figure 17 (a) and 17 (b) show the revenue gain of Amoeba over *Fixed* with both pricing functions. We see that the revenue gain follows a similar trend as the throughput gain for deadline traffic shown in Figure 13 (c). In addition, we observe that the up to 60% higher network throughput directly translates to up to 25% and 32% higher provider revenue respectively.

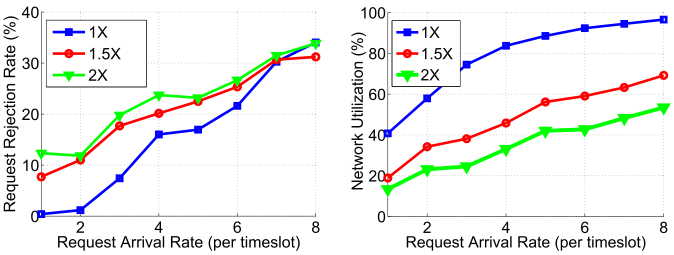
G. Parameter Sensitivity Analysis

Impact of Larger Timeslots: Figure 18 (a) shows the request rejection rate of Amoeba with a time slot of 10 minutes (2×) and 25 minutes (5×) respectively. We see that larger timeslots brings up to 15% points and 37% points increase on request rejection rate. It is intuitive that larger timeslot results in less fine-grained and less flexible bandwidth allocation, thus in general reduces the probability of being accepted for any incoming request. Moreover, as we can see in Figure 18 (b), lower acceptance rate directly results in a lower network utilization under low arrival rate for both 10-minute and 25-minute cases. However, we also observe that the impact



(a) (b)

Fig. 18. Impact of larger timeslots. (a) Request rejection rate. (b) Network utilization.



(a) (b)

Fig. 19. Impact of tighter deadlines. (a) Request rejection rate. (b) Network utilization.

of larger timeslot is less obvious under high arrival rate as the network can be eventually exhausted in all cases.

Impact of Tighter Deadlines: In the following simulation we keep the transfer volume unchanged and tighten the required transfer time (i.e., $td_1 - ts$) by 1.5$\times$ and 2$\times$ respectively. As we can see in Figure 19 (a), a tighter deadline makes it harder for a request to be accepted, thus increasing the request rejection rate by up to 9% and 13% points compared to the default setting. Moreover, in Figure 19 (b) we observe up to 37% points and 50% points utilization loss in the 1.5$\times$ and 2$\times$ cases respectively. One possible reason is that many requests with large volume get rejected with a tighter deadline. More interestingly, note that the rejection to a big request may lead to the acceptance to one or more small requests which arrive afterwards. This may explain why rejection rate is not heavily affected under high arrival rate.

IX. Discussion

We further discuss some extensions to the current Amoeba design.

Utility Function: Different tenants may desire different utility functions describing their utility decrease from td_1 to td_2, and some tenants may be clear about their utility functions, while others may not. To this end, an optional field $U(t)$ can be added to the DNA abstraction for tenants to describe their utility functions, and Amoeba can be extended to account for arbitrary utility functions accordingly.

On the one hand, for tenants who are not so clear about their utility functions (optional field left blank), the admission procedure shown in §V-B.1 can be directly applied, which does not rely on any utility function information. On the other hand, for tenants with some arbitrary utility function $U(t)$, we extend Amoeba to optimize for the tenant's benefit, i.e., the utility minus the payment. More specifically, note that the payment (§V-C) is also a function of the guaranteed transfer time, and thus can be denoted as $P(t)$. Then instead of trying to finish the request as soon as possible, Amoeba returns the guaranteed transfer time t^* which maximizes the tenant's benefit, i.e., $t^* = argmax_{t\in[t',td_2]}(U(t) - P(t))$.

Amoeba Speedup: While Algorithm 1 is an LP and can be solved by using typical LP solvers, the constrains can be normalized into the standard form of mixed packing-covering (MPC) problem [22]. This observation leads to a possible speedup of AS. More specifically, we can calculate the shortest possible completion time (i.e., the output of AS) by performing a binary search to find the minimum t which satisfies all the constrains in Algorithm 1. Note that checking the feasibility of a given t is a standard MPC problem, thus fast approximation algorithms such as those introduced in [4] and [22] can be applied to speed up the calculation.

In addition, Amoeba consists of several components which support incremental deployment as we show in Figure 14. As a result, we can make a tradeoff between allocation performance and scalability: in large inter-DC networks or under high request arrival rates, we can disable parts of Amoeba, sacrificing some performance to achieve acceptable allocation time.

X. Related Work

There are many related works on datacenter traffic optimization or deadline-aware flow scheduling. However none of them can directly solve our problems in Amoeba.

TE for Inter-DC WAN networks has attracted great interest recently. B4 [2] presents Google's inter-DC WAN solution based on the popular software-defined networking technology; its centralized TE drives links to near full utilization. Similarly, SWAN [1] also boosts the utilization of inter-DC WAN by scheduling the service traffic in a centralized manner; it further achieves congestion-free and disruption-free updates. However, they both are deadline-agnostic.

More recently, Tempus [4] has been proposed to maximize the fraction of transfer delivered before deadline. It achieves fairness among all the requests, but does not guarantee the completion of any of them. Relative to Tempus, Amoeba maximizes the number of transfers completed before their deadlines, which is more suitable for applications with hard deadlines. Moreover, the abstraction of Tempus deals with each transfer individually, whereas Amoeba takes the correlation among multiple transfers into consideration.

NetStitcher [9] focuses on using a store-and-forward approach to schedule large scale data transfers between DCs, but without considering any transfer deadlines. In contrast, Amoeba moves one step further by adding a deadline-aware transfer interface into the system so that providers can develop algorithms to better utilize expensive WAN bandwidth to guarantee transfer deadlines.

In the context of Intra-DC networks, there are several bandwidth guaranteed abstractions [6]–[8], [23]–[26]. For example, SecondNet [6] enforces bandwidth reservation between any pair of VMs. Oktopus [7] proposes two virtual cluster (VC) models, one non-oversubscribed and one oversubscribed of bandwidth. Gatekeeper [23] and EyeQ [24] can enforce hose model for congestion-free networks. TIVC [8] tries to catch the time-varying nature of the networking requirement by defining time-interleaved virtual clusters. ElasticSwitch [25] and Trinity [26] achieve bandwidth guarantee and work conservation simultaneously. Our DNA abstraction allows requests to be expressed in terms of the volume of data to be delivered and the deadlines by which they must be delivered. This is more appropriate for deadline-driven inter-DC bulk transfers.

There are new protocols designed to meet flow deadlines or minimize flow completion times in intra-DC, e.g., [27]–[32]. They are handling flow deadlines at the millisecond level, and most rely on the short round-trip-time in intra-DC environment for effective congestion or rate control. The problem of scheduling large transfers in inter-DC WAN is fundamentally different, as it operates at a much longer time scale (with a deadline of minutes or hours) and schedules requests which are aggregations of many flows [4].

In the context of Grid networks, some works such as [33] and [34] have studied the problem of deadline-aware data transfer. However, the discrepancy between the Grid networks and the inter-DC WANs makes it hard to directly apply their solutions to our problems in Amoeba. First, they do not consider practical issues in inter-DC WANs such as traffic priorities, mispredictions, traffic dynamics and failures. Second, the modelings and abstractions in [33] and [34] cannot capture the demands of inter-DC applications, e.g., soft deadlines, and barrier-synchronized deadline transfers in a request. Moreover, neither of these works achieves a good balance between scalability and optimality. For example, Chen's scheduling algorithm [33] and the SR module in [34] are ineffective as they assume the previous allocations to be fixed. On the other hand, the RR module proposed in [34] is too time-consuming for real-time allocations because it blindly applies a global optimization.

Deadline-aware scheduling has also been widely considered in real-time systems [35]. However, most deadline-aware real-time system algorithms (*e.g.*, earliest deadline first) are distributed and suboptimal compared to the centralized scheduling. We design Amoeba in a centralized manner to achieve high network utilization. Another related field is background traffic scheduling in distributed applications [36], however they usually do not take deadlines into account.

XI. Conclusion

A large portion of Inter-DC transfers have deadlines, however, currently no mechanism is in place to ensure such deadlines. This paper introduces a deadline-based network abstraction, DNA, as an interface that allows tenants to explicitly express an inter-DC transfer request in terms of the data volume and the deadline by which it must be delivered. Our system, Amoeba, performs on-line admission control and enforcement to implement DNA in a scalable manner. Our evaluation shows that Amoeba effectively accommodates more transfer requests with deadline guarantees, while achieving around 60% higher network throughput than state-of-the-art bandwidth guarantee solutions.

Appendix

NP-Hardness of the Failure Handling Problem: We reduce the all-or-nothing multi-commodity flow (AN-MCF) problem (which is NP-hard [38]) to the failure handling problem.

Proof Sketch: We first present the formulation for both the failure handling and the AN-MCF problem.

Failure Handling Problem: Given the DCN topology with residual bandwidth under failure, and the set of already accepted/allocated requests $\mathbf{R} = \{R_1, R_2, ...R_n\}$. We need to find out $\mathbf{R}'$ — a subset of $\mathbf{R}$, such that all $R_i \in \mathbf{R}'$ can finish their remaining transfer volume within their deadlines. Denote w_i as the weight for request R_i, then the goal is to maximize $\sum_{R_i \in \mathbf{R}'} w_i$, the weighted sum of the remaining requests.

AN-MCF: Given a capacitated undirected graph $G = (V, E, u)$ and a set of k pairs $s_1t_1, s_2t_2, \ldots, s_kt_k$. Each pair has a unit demand. The objective is to find a largest set $\mathbf{S}$ of pairs, such that for every $s_it_i \in \mathbf{S}$ we can send a flow of one unit between s_i and t_i.

We now show that the AN-MCF problem can be reduced to the failure handling problem in general graphs in polynomial time: Given an instance of the AN-MCF problem, we can construct a DCN topology equivalent to G, and for each pair s_it_i, we create a corresponding request R_i with one transfer $T = \{src = s_i, dst = t_i, Q = 1, ts = td_1 = td_2 = 1\}$. Moreover, we set the weight of all request to 1. In such a way, an $\mathbf{S}$ is a solution to the instance of the AN-MCF problem if and only if the corresponding $\mathbf{R}'$ is a solution to the corresponding instance of the failure handling problem, thus the AN-MCF problem can be reduced to the failure handling problem. $\square$

Acknowledgements

Part of this work was presented in ACM EuroSys'16 [37]. The authors thank the anonymous EuroSys and TON reviewers for their constructive comments and Harsha Madhyastha for his detailed feedback and suggestions.

References

[1] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 15–26.

[2] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 3–14.

[3] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via Yahoo! Datasets," in *Proc. INFOCOM*, 2011, pp. 1620–1628.

[4] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula, "Calendaring for wide area networks," in *Proc. ACM SIGCOMM*, 2014, pp. 515–526.

[5] R. Illsley, (2012). *Why DC-DC WAN Optimization Matters*. [Online]. Available: http://www.telecomasia.net/content/why-dc-dc-wan-optimization-matters?page=1

[6] C. Guo *et al.*, "SecondNet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. 6th CoNEXT*, 2010, Art. no. 15.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. SIGCOMM*, 2011, pp. 242–253.

[8] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *Proc. SIGCOMM*, 2012, pp. 199–210.

[9] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with netstitcher," in *Proc. SIGCOMM*, 2011, pp. 74–85.
[10] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bridging the tenant-provider gap in cloud services," in *Proc. ACM SoCC*, 2012, Art. no. 10.
[11] C. Jayalath, J. Stephen, and P. Eugster, "From the cloud to the atmosphere: Running MapReduce across data centers," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 74–87, Jan. 2014.
[12] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proc. ACM SIGCOMM*, 2014, pp. 527–538.
[13] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, May 1998. [Online]. Available: http://doi.acm.org/10.1145/279227.279229
[14] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 337–348, 2007.
[15] A. Kumar *et al.*, "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *Proc. ACM SIGCOMM*, 2015, pp. 1–14.
[16] *Source Routing*. accessed on Dec. 2013, [Online]. Available: http://en.wikipedia.org/wiki/Sourcerouting
[17] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," RFC 3031, 2001.
[18] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.
[19] S. Hu *et al.*, "Explicit path control in commodity data centers: Design and applications," in *Proc. 12th NSDI*, 2015, pp. 15–28.
[20] L. R. Ford, Jr., and D. R. Fulkerson, "Constructing maximal dynamic flows from static flows," *Oper. Res.*, vol. 6, no. 3, pp. 419–433, 1958.
[21] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 350–361, Aug. 2011.
[22] N. E. Young, "Sequential and parallel algorithms for mixed packing and covering," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, Oct. 2001, pp. 538–546.
[23] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proc. 3rd WIOV*, 2011, p. 6.
[24] V. Jeyakumar *et al.*, "EyeQ: Practical network performance isolation at the edge," in *Proc. 10th USENIX NSDI*, 2013, pp. 297–312.
[25] L. Popa *et al.*, "ElasticSwitch: Practical work-conserving bandwidth guarantees for cloud computing," in *Proc. ACM SIGCOMM*, 2013, pp. 351–362.
[26] S. Hu *et al.*, "Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud," in *Proc. INFOCOM*, 2016, pp. 1–9.
[27] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 50–61, Aug. 2011.
[28] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter tcp (D2TCP)," in *Proc. ACM SIGCOMM*, 2012, pp. 115–126.
[29] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 127–138, 2012.
[30] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 139–150, 2012.
[31] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," in *Proc. SIGCOMM*, 2013, pp. 435–446.
[32] W. Bai *et al.*, "Information-agnostic flow scheduling for commodity data centers," in *Proc. 12th NSDI*, 2015, pp. 455–468.
[33] B. B. Chen and P. V.-B. Primet, "Scheduling deadline-constrained bulk data transfers to minimize network congestion," in *Proc. 7th CCGRID*, vol. 7. 2007, pp. 410–417.
[34] K. Rajah, S. Ranka, and Y. Xia, "Advance reservations and scheduling for bulk transfers in research networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 11, pp. 1682–1697, Nov. 2009.
[35] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo, *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*, vol. 460. New York, NY, USA: Springer, 2012.
[36] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP nice: A mechanism for background transfers," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 329–343, 2002.
[37] H. Zhang *et al.*, "Guaranteeing deadlines for inter-datacenter transfers," in *Proc. 10th EuroSys*, 2015, Art. no. 20.
[38] C. Chekuri, S. Khanna, and F. B. Shepherd, "The all-or-nothing multicommodity flow problem," in *Proc. 36th Annu. ACM Symp. Theory Comput.*, 2004, pp. 156–165.

Hong Zhang received the B.S. and M.S. degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China, in 2010 and 2013, respectively. He is currently pursuing the Ph.D. degree in computer science with The Hong Kong University of Science and Technology. His current research interests are in the area of data center networks.

Kai Chen received the Ph.D. degree in computer science from Northwestern University, Evanston, IL, USA, in 2012. He is an Assistant 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.

Wei Bai received the B.E. degree in information security from Shanghai Jiao Tong University China in 2013. He is currently pursuing the Ph.D. degree in computer science with The Hong Kong University of Science and Technology. His current research interests are in the area of data center networks.

Dongsu Han received the B.S. degree in computer science from the Korea Advanced Institute of Science (KAIST) in 2003, and the Ph.D. degree in computer science from Carnegie Mellon University in 2012. He is currently an Assistant Professor with the School of Electrical Engineering and Graduate School of Information Security, KAIST. He is interested in networking, distributed systems, and network/system security.

Chen Tian received the B.S., M.S., and Ph.D. degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China, in 2000, 2003, and 2008, respectively. He was an Associate Professor with the School of Electronics Information and Communications, Huazhong University of Science and Technology. He is an Associate Professor with the State Key Laboratory for Novel Software Technology, Nanjing University, China. From 2012 to 2013, he was a Post-Doctoral Researcher with the Department of Computer Science, Yale University. His research interests include data center networks, network function virtualization, distributed systems, Internet streaming, and urban computing.

Hao Wang received the B.E. and M.E. degrees from Shanghai Jiao Tong University in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree with the University of Toronto. During his master's studies, he was a Research Assistant with The Hong Kong University of Science and Technology for 14 months. His research interests include datacenter networking, distributed computing, and software defined networking.

Haibing Guan received the Ph.D. degree in artificial intelligence from Tongji University, Shanghai, China, in 1999. He is currently a Professor with the School of Electronic, Information and Electronic Engineering, Shanghai Jiao Tong University, Shanghai, and the Director of the Shanghai Key Laboratory of Scalable Computing and Systems. His research interests include distributed computing, network security, network storage, green IT, and cloud computing.

Ming Zhang was a Senior Researcher with Microsoft Research Redmond for over ten years, during which time he delivered multiple key technologies that power the massive cloud networks of Microsoft Azure. He is a Senior Director of Alibaba Infrastructure Service, where he leads the development of automation systems that keep Alibabas global datacenter networks running reliably, efficiently, and at scale. He holds 16 U.S. patents and seven patents files and publishes many seminal research papers in top systems and networking conferences, including the ACM SIGCOMM, OSDI, and NSDI. His research was featured in influential media outlets, such as BBC, CNN, MIT Tech Review, and The Verge.