Bridging the Gap between Peak and Average Loads on Science Networks

Sam Nickolay[a], Eun-Sung Jung[b,*], Rajkumar Kettimuthu[c], Ian Foster[a,c]

[a]*Department of Computer Science, University of Chicago, Chicago, Illinois, USA*
[b]*Hongik University, South Korea*
[c]*Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, Illinois, USA*

Abstract

Backbone networks are typically overprovisioned in order to support peak loads. Research and education networks (RENs), for example, are often designed to operate at 20–30% of capacity. Thus, Internet2 upgrades its backbone interconnects when the weekly 95th-percentile load is reliably above 30% of link capacity, and analysis of ESnet traffic between major laboratories shows a substantial gap between peak and average utilization. As science data volumes increase exponentially, it is unclear whether this overprovisioning trend can continue into the future. Even if overprovisioning is possible, it may not be the most cost-effective (and desirable) approach going forward. Under the current mode of free access to RENs, traffic at peak load may include both flows that need to be transferred in near-real time—for example, for computation and instrument monitoring and steering—and flows that are less time-critical, for example, archival and storage replication operations. Thus, peak load does not necessarily indicate the capacity that is absolutely required at that moment. We thus examine how data transfers are impacted when the average network load is increased while the network capacity is kept at the current levels. We also classify data transfers into on-demand (time-critical) and best-effort (less time-critical) and study the impact on both classes for different proportions of both the number of on-demand transfers and amount of bandwidth allocated for on-demand transfers. For our study, we use real transfer logs from production GridFTP servers to do simulation-based experiments as well as real experiments on a testbed. We find that when the transfer load is doubled and the network capacity is fixed at the current level, the gap between peak and average throughput decreases by an average of 18% in the simulation experiments and 16% in the testbed experiments, and the average slowdown experienced by the data transfers is under 1.5×. Furthermore, when transfers are classified as on-demand or best-effort, on-demand transfers experience almost no slowdown and the mean slowdown experienced by best-effort transfers is under 2× in the simulation experiments and under 1.2× in the testbed experiments.

Keywords: Network utilization, Data transfer scheduling, Network planning

1. Introduction

Scientific applications in various domains such as high-energy physics, cosmology, genomics, etc., generate large data sets that need to be transported over the network for a variety of reasons. In order to support these applications, federal agencies in different countries fund organizations to build and operate high-speed research and education networks. These networks are typically overprovisioned so that all science users continue to have adequate network resources even at times of peak load. Thus, they are underutilized most of the time. Research and education networks (RENs), such as Internet2, have a policy of operating their networks at light loads (25–30%) to allow the networks to absorb surges in traffic [1]. Other RENs such as ESnet and GEANT are also engineered to operate at similar average load levels.

Several recent studies project that, given predictions of science traffic's exponential growth, it may not be feasible to continue such overprovisioning [2, 3, 4]. Recent reports on science network requirements note that different transfers have different needs [5, 6, 7]. Some transfers, such as those in which remote analysis result of one experiment is used to guide the next, are time-critical and have tight constraints. In contrast, some transfers, such as certain data replication, backup, and archiving use cases, have more flexibility and may only need to be completed within a window several times longer than the transfer time under average load. Under the current mode of free access to RENs, the traffic at peak load may include a combination of different types of transfers including some that are less time-critical. We argue that measures to spread the load and keep the peak under control are important, and use simulations based on real traffic traces to quantify the benefits that may be gained from such measures.

The two main contributions of this study are as follows:

- We characterize how data transfers are impacted when the gap between peak and average throughput is re-

[*]Corresponding author
Email addresses: samnickolay@uchicago.edu (Sam Nickolay), ejung@hongik.ac.kr (Eun-Sung Jung), kettimut@anl.gov (Rajkumar Kettimuthu), foster@anl.gov (Ian Foster)

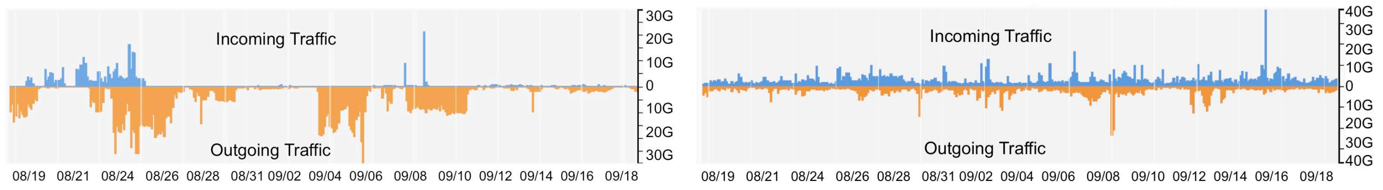


Figure 1: WAN traffic patterns at two HPC facilities, both for the period Friday 19 Aug 2016 to Sunday 18 Sep 2016. (Source: my.es.net)

duced by increasing the average network load while keeping network capacity at the current levels.

- We determine how these impacts change when data transfers are classified as either on-demand (ones that are time-critical) or best-effort (ones that are less time-critical), with on-demand transfers getting a relatively larger share of the bandwidth.

These results suggest that the utilization of research networks can be increased significantly with modest or no impact on the performance of deadline-sensitive data transfers, simply by identifying deadline-sensitive transfers as such.

We use transfer logs from production GridFTP servers for our study. Our results indicate that when load is doubled with network capacity fixed at the current level, the gap between peak and average load decreases by an average of 18% and the average slowdown experienced by data transfers is still less than 1.5×. For the logs in which the peak load is 5× or more than the average load, average slowdown experienced by the data transfers is under 1.1×. Under the same scenario of doubled network load with the same network capacity, when the transfers are classified into on-demand and best-effort, on-demand transfers experience almost no slowdown and the slowdown experienced by best-effort transfers is under 2×, even when 50% of transfers were treated as on-demand and on-demand transfers are given a reasonably higher share (70%) of the bandwidth. For the logs in which the peak load is 5× or more than the average load, the average slowdown experienced by best-effort data transfers is under 1.2×.

The rest of the paper is organized as follows. Section 2 introduces the two characteristics of contemporary networks and science workloads that motivate our research. Section 3 presents the problem our research addresses and the metrics we use to evaluate our results. Section 4 defines the algorithm we developed for our research. Section 5 evaluates the results from our experiments. Section 6 describes the testbed experiments and their results. Section 7 discusses how this research can be generalized. Section 8 examines related work. Section 9 discusses the conclusions from our research.

2. Background and Motivation

Our research is motivated by the following two characteristics of contemporary networks and science workloads.

2.1. Big Gap between Peak and Average Network Load

Figure 1, which shows the wide-area network traffic over a one-month period for two HPC facilities, illustrates the substantial gap between average and peak loads that is common on RENs.

We also obtained logs from the anonymized usage statistics that Globus GridFTP servers send to a usage collector. These logs include transfer size, start time, and end time. We collected the logs of the four servers that transferred the most bytes in a one month period. For each of those servers, we then picked the log for the day in that month in which it transferred the most bytes. Figure 2 shows the aggregate throughput over that 24-hour period for each server. Once again, we see substantial difference between mean and peak throughputs.

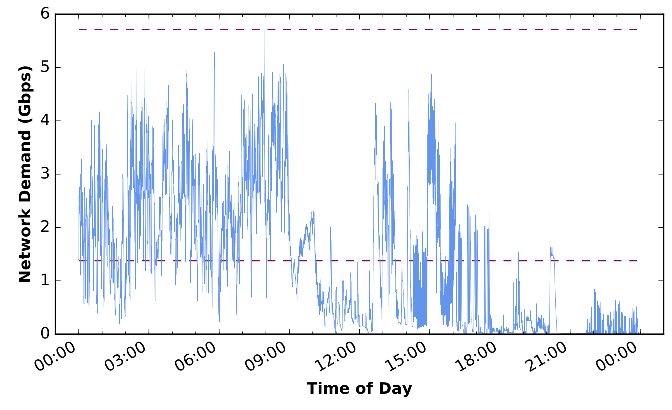
2.2. Some Transfers can Tolerate Delays

While certain transfer requests are time-sensitive, others can tolerate delays. For example, science communities replicate large quantities for performance [8, 9], fault tolerance [10], and/or preservation [11]. Such replication activities are often cited as a common relatively time-insensitive data transfer use case [5, 6, 7], as when a multi-TB dataset needs to be copied to a remote site overnight. Because subsequent processing involves manual steps, there is no advantage in completing the transfer earlier. Replicating 100 TB within a month is another use case cited. For these use cases, transfer times can vary by at least an order of magnitude without compromising science goals.

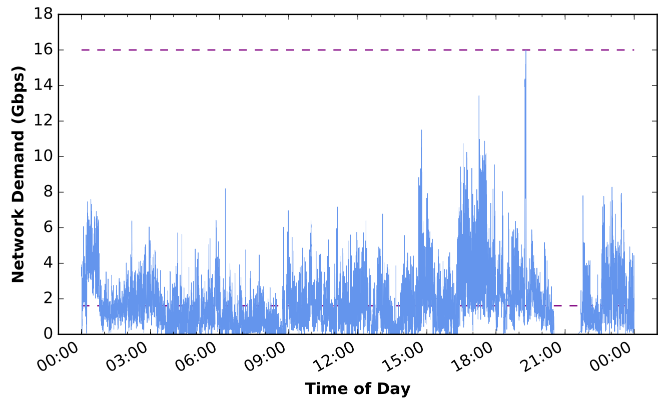
3. Problem Statement and Goals

Current RENs experience highly variable load patterns and offer no differentiation in the service provided to different applications. These networks are overprovisioned (see Definition 1 in Section 3.2) to absorb traffic surges and to ensure that no flow is delayed due to aggregate load exceeding capacity. Consequently, the average usage is limited to X%, where X is usually small (as low as 10% in some cases). Thus the gap between peak and average aggregate throughput on these networks is often high.

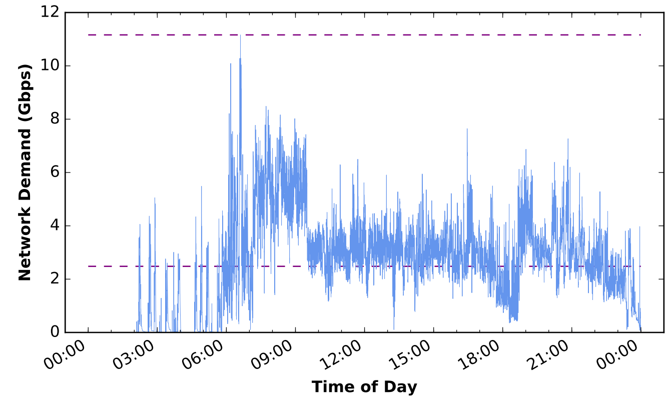
If the service provided by a REN could be differentiated so that traffic requiring instant service (on-demand) was treated differently from traffic that can be delayed by a certain amount (best-effort), then we could reduce the gap between peak and average throughput (e.g., by increasing the average usage to Y%) on these networks without any flows being delayed beyond their service level. The question then is under what circumstances is this true and how do X and Y relate to the service levels chosen for the best-effort and on-demand traffic classes. This is the problem we study in this work.



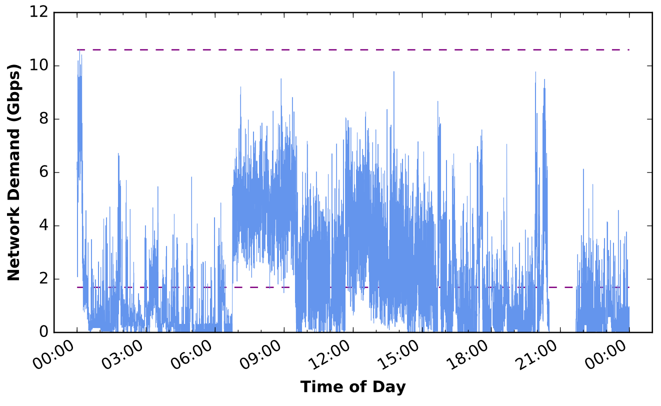
(a) *trace*$_1$



(b) *trace*$_2$



(c) *trace*$_3$



(d) *trace*$_4$

Figure 2: Network demand for *trace*$_1$, *trace*$_2$, *trace*$_3$, and *trace*$_4$. The two dashed lines on each trace indicate the peak and mean throughputs for that trace during the 24-hour period. The peak is $\sim 4\times$, $8\times$, $5\times$, and $6\times$ the mean for *trace*$_1$, *trace*$_2$, *trace*$_3$, and *trace*$_4$, respectively.

3.1. Background of the Problem

Our target networks are wide-area networks composed of many components such as switches and data transfer nodes (DTNs). The networks may be either circuit-switched networks, in which paths are reserved ahead of data transfers, or packet-switched networks, in which packets from multiple data transfers can share network links. In reality, most networks used by data transfer users such as scientists are packet-switched networks, except for special networks such as ESnet [12] and Internet2 [13] which support circuits. However, we assume that both kinds of networks offer ways (e.g., tc [14] Linux) to perform traffic engineering on each flow.

3.1.1. Network Traffic Model

In general, network traffic or data transfers can be categorized into multiple classes ranging from near-real time to best-effort, depending on the importance of response time. In this paper, we consider just two classes, on-demand (OD) data transfer and best-effort (BE) data transfer. OD data transfers have higher priority than BE data transfers since we assume that OD data transfers have stricter timing constraints than BE data transfers. However, our data transfer model does not have explicit timing constraints other than data transfer type. Thus, one transfer can be described by a five-tuple (*source*, *destination*, *transfer type*, *data size*, *requested rate*) where *source* is a data sender, *destination* is a data receiver, *transfer type* is either OD or BE, *data size* is size of data to send (in bytes), and *requested rate* is the estimated throughput of the data transfer (in bits/s) when there is no network contention. We define requested load as the total requested rate of all transfers of the same type at a certain point of time.

3.1.2. Traffic Management/Engineering

Another important assumption in this paper is that a single centralized controller is responsible for admission control and scheduling of all the data transfer requests. The single centralized controller thus has a single data transfer request queue and sequentially selects the next data transfer to schedule/perform. In order to properly schedule/perform the next data transfer, the controller will check the available bandwidth of the path for the data transfer. The available bandwidths of links in the networks for which the controller is responsible are assumed to be monitored periodically by using some mechanisms such as SNMP. Without any congestion or failures, the path taken for a certain data transfer remains the same throughout the data transfer even in packet-switched networks.

In our scheduling framework, all transfers have two rates associated with them: *current* and *requested*. The current rate is the bandwidth currently allocated to the transfer by a centralized controller. The requested rate is the estimated throughput provided in the data transfer request. For our simulations, we set the requested rate to the transfer rate in the original trace.

The scheduling framework operates under the assumption that we can selectively limit the bandwidth allocated to an individual transfer at any given time. It also assumes that a transfer's requested rate represents the transfer throughput when there is no limiting external load and so the requested rate is the highest bandwidth that the transfer could achieve. Thus, a transfer's current rate can never be higher than its requested rate.

3.2. Goals and Performance Metrics

The goals of our work are two-fold. First, we want to reduce the gap between peak and average link utilization on paths so as to enable higher bandwidth appliations and/or avoid the need for expensive network upgrades, e.g., 100 Gbps to 1 Tbps. Second, we want to ensure that OD data transfers can be initiated immediately, without little or no delay, while BE data transfers suffer only reasonable amount of delays.

We thus define two performance metrics: 1) slowdown and 2) gap between peak and average rate. *Slowdown*, a concept commonly used in parallel job scheduling [15], is generally defined to be $\frac{runtime-with-new-scheduling}{runtime-with-old-schehduling}$ where *runtime* includes the wait time between the transfer's arrival time and scheduled time. However, *Slowdown* may become so large for a small data transfer with a large wait time that it can distort the overall *slowdown* average. Thus, *bounded slowdown* is introduced to bound such cases to some extent using the *max* operation, as follows, where *Bound* is a user-defined threshold used to limit the influence of short transfers on slowdown.

$$BS = \frac{max(\text{Simulation Runtime}, \text{Bound})}{max(\text{Trace Runtime}, \text{Bound})} \tag{1}$$

Since we do simulations with real data traces, we make several tweaks in the original slowdown equation. *Runtime* in the numerator is replaced by *simulation runtime*, which is the transfer's runtime in the simulation. *Runtime* in the denominator is replaced by *trace runtime*, which is the transfer's runtime in the original data transfer trace. For our simulations, *bound* is set to be equal to the scheduling interval used for scheduling the transfers, which is 1 second.

For the simulations, we actually use a total of four different bounded slowdown metrics.

- Avg_BS_{OD}: Average bounded slowdown of all OD transfers, defined as follows.

$$Avg\_BS_{\text{OD}} = \left(\sum_{\forall t} (Bounded\ Slowdown)_t \right) /T \tag{2}$$

  where $t \in$ OD transfers in a trace and T is the total number of OD transfers.
- Max_BS_{OD}: Maximum bounded slowdown among all OD transfers, defined as follows:

$$Max\_BS_{\text{OD}} = \max_{\forall t} (Bounded\ Slowdown)_t \tag{3}$$

- Avg_BS_{BE}: Average bounded slowdown of all BE transfers, defined in the same way as Avg_BS_{OD}.
- Max_BS_{BE}: Maximum bounded slowdown for BE transfers, defined in the same way as Max_BS_{OD}.

We also formally define the degree of network overprovisioning under a certain network traffic management method.

Definition 1 (Overprovisioned network). *An overprovisioned network is a network such that its link capacities prevent any potential overload caused by traffic stochastic behavior and traffic shifts.*

4. Scheduling Algorithm

After presenting our formal problem statement and goals, we now present the algorithm we have developed for scheduling OD and BE data transfers to achieve the goals within our simulation framework. Listing 1 gives the algorithm, and Table 1 summarizes our terminology.

Table 1: Summary of terminology used in describing our algorithm

Item	Description
OD transfers	On-demand transfers
BE transfers	Best-effort transfers
transfer.rate	Bandwidth currently allocated to transfer
transfer. requestRate	Bandwidth that transfer has requested as an estimated throughput. In simulation, this is set to the transfer throughput in the original trace.
$requestLoad_{\text{OD}}$	$\sum_{\forall \text{running OD transfers}} transfer.requestRate$
$requestLoad_{\text{BE}}$	$\sum_{\forall \text{running BE transfers}} transfer.requestRate$
$availBW_{\text{OD}}$	Bandwidth available for OD transfers
$availBW_{\text{BE}}$	Bandwidth available for BE transfers
netCap	Maximum bandwidth for the network
$netCap_{\text{OD}}$	Bandwidth allocated for OD transfers
$netCap_{\text{BE}}$	Bandwidth allocated for BE transfers

4.1. Overview of Algorithm

Our scheduling algorithm works with two distinct categories of data transfers, OD and BE. Our scheduling algorithm does not give any implicit priority or scheduling advantage to either type of transfer. Instead it is though explicitly setting the allocated OD bandwidth ($netCap_{\text{OD}}$) and BE bandwidth ($netCap_{\text{BE}}$) values that priority is given to one transfer type over another. For the sake of clarity, we define $netCap_{\text{OD}}$ and $netCap_{\text{BE}}$ in terms of a percentage of the total *netCap*, and since $netCap_{\text{OD}} + netCap_{\text{BE}} = netCap$, the two percentages always sum to 100. In our work, OD transfers are designated as being more time critical than BE transfers. To account for this in our simulations, we allocate OD transfers a $netCap_{\text{OD}}$ that is a higher percentage of the total *netCap* than the percentage of OD transfers in the simulation. For instance,

Listing 1: Scheduling heuristic

```
# 1. Start OD transfers and BE transfers
for transfer in OD-transfer-queue:
  start transfer
for transfer in BE-transfer-queue
  start transfer

# 2. Compute BE transfers available bandwidth
if requestLoad_OD < netCap_OD:
  availBW_BE = netCap - requestLoad_OD
else:
  availBW_BE = netCap_BE

# 3. Update BE transfer rates
for transfer in BE running transfers:
  fairshareBandwidth = transfer.requestRate /
      requestLoad_BE × availBW_BE
  transfer.rate = min(fairshareBandwidth,
      transfer.requestedRate)

# 4. Compute OD transfers available bandwidth
if requestLoad_BE < netCap_BE:
  availBW_OD = netCap - requestLoad_BE
else:
  availBW_OD = netCap_OD

# 5. Update OD transfer rates
for transfer in OD running transfers:
  fairshareBandwidth = transfer.requestRate /
      requestLoad_OD × availBW_OD
    transfer.rate = min(fairshareBandwidth,
        transfer.requestedRate)
```

consider a simulation in which 30% of all transfers are OD and the remaining 70% are BE. By allocating OD transfers a guaranteed bandwidth that is more than 30% of the total available, such as setting $netCap_{\mathrm{OD}} = netCap \times 40\%$ and $netCap_{\mathrm{BE}} = netCap \times 60\%$, we indirectly give OD transfers a higher priority in our scheduling algorithm.

The detailed algorithm is elaborated in Listing 1. The scheduling cycle repeats every n seconds; the two queues `OD-transfer-queue` and `BE-transfer-queue` contain any new transfers that arrive in those n seconds. (In our implementation, $n = 1$.) At the start of each cycle, the scheduler starts any OD and BE transfers that are in `OD-transfer-queue` and `BE-transfer-queue` respectively. Next, the algorithm computes the available bandwidth for BE transfers ($availBW_{\mathrm{BE}}$) by comparing $requestLoad_{\mathrm{OD}}$ with $netCap_{\mathrm{OD}}$. If the requested OD bandwidth is less than the allocated OD bandwidth, then there is extra bandwidth that was allocated for OD transfers but is now available for BE transfers. Thus, $availBW_{\mathrm{BE}}$ is equal to the difference between total bandwidth and the bandwidth requested by OD transfers ($netCap - requestedLoad_{\mathrm{OD}}$). Otherwise, the available BE bandwidth is equal to the allocated BE bandwidth ($netCap_{\mathrm{BE}}$).

Then, for every active BE transfer, the scheduler calculates the transfer's *fairshareBandwidth*:

$$fairshareBandwidth = \frac{transfer.requestRate \times availBW}{requestLoad} \tag{4}$$

where *requestLoad* is the requested bandwidth for all active transfers of the same type as the current transfer, and *availBW* is the bandwidth available to transfers of the same type as the current transfer. As noted above, *transfer.currentRate* is capped by *transfer.requestRate*, and so a transfer's current rate is the minimum of its *fairshareBandwidth* and requested rate.

After updating the transfer rates for all BE transfers, the process is repeated for OD transfers. First, the scheduler computes the available bandwidth for OD transfers ($availBW_{\mathrm{OD}}$). If $requestLoad_{\mathrm{BE}}$ is less than $netCap_{\mathrm{BE}}$, then there is extra BE bandwidth available for OD transfers, and $availBW_{\mathrm{OD}}$ equals the difference between total bandwidth and the bandwidth requested by BE transfers ($netCap - requestLoad_{\mathrm{BE}}$). Otherwise, $availBW_{\mathrm{OD}}$ equals the allocated OD bandwidth ($netCap_{\mathrm{OD}}$). Then, for every active OD transfer, the algorithm computes its *fairshareBandwidth* as defined above, and sets the transfer's current rate to be the minimum of its *fairshareBandwidth* and requested rate.

Our algorithm is focused only on two end points. We assume that the path between two end points does not change over time and that no data transfer between other end points sharingthe links on the path takes place. We leave the development of an algorithm for the general network topology as future work; our goal in this paper is to verify that it is feasible to cap link utilization through well-designed data transfer scheduling.

4.2. Illustrative Example

To understand how the dynamic scheduling algorithm distributes the network bandwidth to OD and BE transfers, consider a simulation in which $netCap_{\mathrm{OD}} = 30\% \times netCap$. In this case, OD transfers are guaranteed access to 30% of the total bandwidth regardless of the bandwidth requested by BE transfers. However, this does not mean that the total OD bandwidth will always be 30%.

For example, if at one point during the simulation, $requestLoad_{\mathrm{OD}} = 15\% \times netCap$ and $requestLoad_{\mathrm{BE}} = 100\% \times netCap$, then OD transfers only use the bandwidth they requested ($15\% \times netCap$), and the remaining 85% of the bandwidth is available for BE transfers.

If at another point in the simulation, the situation is reversed and $requestLoad_{\mathrm{OD}} = 100\% \times netCap$ and $requestLoad_{\mathrm{BE}} = 15\% \times netCap$, then BE transfers use the bandwidth they requested ($15\% \times netCap$), and OD transfers the remaining 85%. In short, each transfer type is guaranteed a certain percentage of the *netCap*. However, if at any point one transfer type requests less bandwidth that it is allocated, any remaining bandwidth becomes available for the other transfer type.

Next, to understand how the fairshare heuristic allocates bandwidth, consider an example where $availBW_{\mathrm{OD}}$

is 5 Mbps and there are four active OD transfers, t_1, t_2, t_3, t_4, with requested transfer rates of 1, 2, 3, and 4 Mbps respectively. Thus *requestLoad*$_{\text{OD}}$ is 10 Mbps. Calculating the *fairshareBandwidth* for each of the OD transfers, we obtain $t_1 = 0.5$ Mbps $(1 \times 5 \div 10)$, $t_2 = 1.0$ Mbps $(2 \times 5 \div 10)$, $t_3 = 1.5$ Mbps $(3 \times 5 \div 10)$, and $t_4 = 2.0$ Mbps $(4 \times 5 \div 10)$. In this case, the current rates of these trnasfers are limited by their *fairshareBandwidth*, since $availBW_{\text{OD}}$ was less than *requestLoad*$_{\text{OD}}$. If $availBW_{\text{OD}}$ was higher, for example 20 Mbps, then their current rates would instead be limited by their requested rates and would be 1, 2, 3, and 4 Mbps, respectively.

5. Simulation Studies

We performed extensive simulations to compare the performance of our algorithm with that indicated in Globus GridFTP traces. We first present and then discuss the results of these simulations.

5.1. Simulation Environment

We constructed a simulation framework that uses the transfers in a trace to run simulations based on a number of user-defined variables. The simulator reads the logs in an online fashion, meaning that future transfer requests are not known a priori. Our framework is a discrete event simulator specifically designed to simulate transfers between two end points based on an inputted trace. It operates under the principle that it schedules and processes transfers based on a user-defined scheduling interval. For our experiments, we set the scheduling interval to 1 second, which we found provided good performance and accuracy.

At the start of each scheduling interval, the simulator updates any transfers from the previous interval and removes any transfers that completed transferring. It then runs the scheduling algorithm described above before moving on to the next interval. Each experiment consists of a 24 hour simulation from 12:00am to 12:00am the following day. Since data traffic patterns and quantities tend to vary over the course of a day, we wanted to ensure we captured a full days worth of transfers.

Our ultimate goal is to minimize the gap between the peak and average throughput for all OD and BE transfers. The mean throughput should remain the same for all of our simulations, because a different mean would imply that a different amount of total data was transferred. Therefore, our proposed scheduling algorithm should be able to minimize the peak throughput, which is limited by the network capacity, without significantly affecting the slowdown of transfers. Although our objective is to keep slowdown values down to a reasonable level for both OD and BE transfers, since OD transfers are more response critical than BE transfers, the threshold for impractical slowdown values is much lower for OD transfers than it is for BE transfers.

5.2. Data Transfer Workload

We evaluated our algorithm with four Globus GridFTP server traces, each of which contains information for all transfers for a particular source-destination end point pair on one day. We chose four traces that have varying peak and mean throughput. The peak throughput for the traces varies from 5.7 to 16.0 Gbps, and the mean throughput varies from 1.4 to 2.5 Gbps; more importantly, all four traces have a mean throughput between 10% and 25% of the peak. We assign each transfer an original transfer throughput, which we refer to as the transfer's *requestedRate*, based on limited information in the traces. We assume that the assigned original throughput is the achievable throughput between two end points when there is no network contention.

5.3. Simulation Variables

We ran our simulation framework with various parameter configurations to assess how our algorithm performs under different workloads. The variables that we adjusted are %-OD-transfers, %-OD-bandwidth, and transfer-load-ratio. **%-OD-transfers** is the percentage of transfers that are labeled as OD transfers and all other transfers (i.e., 100% − %-OD-transfers) are labeled as BE transfers. When the simulator reads in a trace, it randomly splits the transfers into OD and BE categories based on the %-OD-transfers. **%-OD-bandwidth** is the percentage of the total path capacity that is dedicated to OD transfers. **Transfer-load-ratio** is the ratio of the transfer load size used in the simulation compared to the transfer load size used in the original trace. This parameter is used to test the simulation under different intensities of transfer loads. For example, a simulation with a transfer-load-ratio of 2 uses the same transfers as defined in the original trace, but with every transfer having double the transfer size and duration, resulting in a total transfer load that is twice that of the original.

We evaluated four different values for each of %-OD-transfers, %-OD-bandwidth, and transfer-load-ratio, for a total of 64 different configurations for each trace. We simulated with %-OD-transfers $\in \{10\%, 30\%, 50\%, 70\%\}$. For the %-OD-bandwidth, we varied the values based on the %-OD-transfers values in order to make the results more pertinent. In order to give OD transfers a higher priority than BE transfers, as we intended, we wanted %-OD-bandwidth $\geq$ %-OD-transfers. Thus, for each %-OD-transfers value, we define %-OD-bandwidth values equal to %-OD-transfers, 10% and 20% greater than %-OD-transfers, and 100% of the total path bandwidth. For example, when %-OD-transfers = 50%, we evaluated %-OD-bandwidth $\in \{50\%, 60\%, 70\%, 100\%\}$. Finally, we studied transfer-load-ratio $\in \{1.0, 1.5, 2.0, 2.5\}$. Note that although we consider the same transfer-load-ratio values for all traces, the resulting transfer loads are different, since the default $(1.0\times)$ transfer loads vary for each trace. For instance, a transfer load resulting from a $2\times$ transfer load

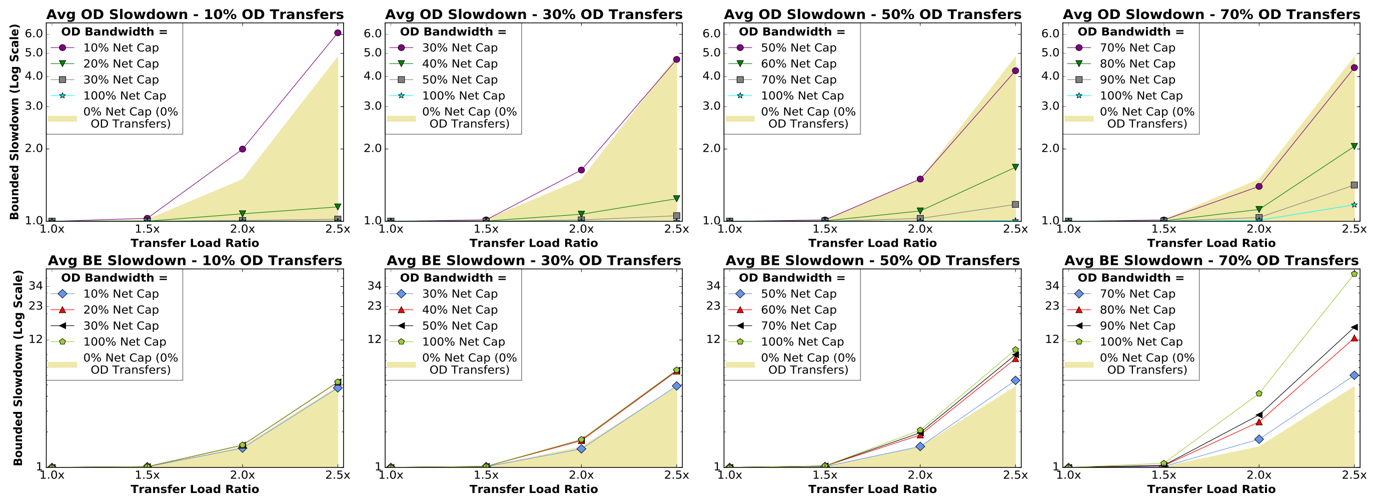


Figure 3: Average bounded slowdown for $trace_1$.

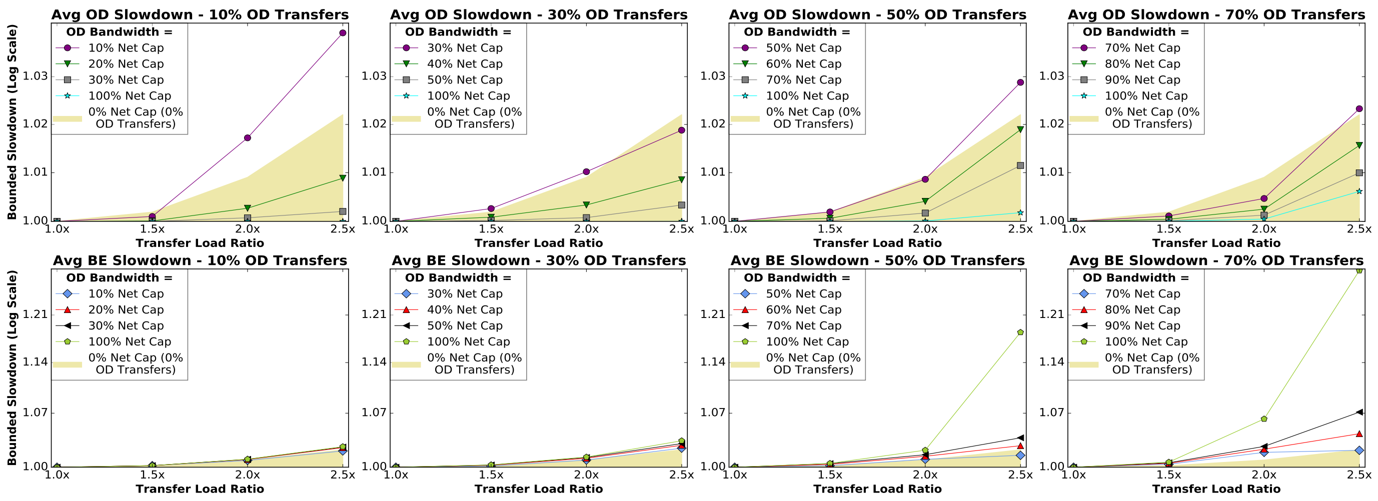


Figure 4: Average bounded slowdown for $trace_2$.

ratio for one trace is different than the transfer load resulting from a 2× transfer load ratio for a different trace.

After simulating each configuration, we calculated the four performance metrics defined in Section 3.2, that is, Avg_BS_{OD}, Max_BS_{OD}, $Avg\_\mathrm{BS}_{\mathrm{BE}}$, and Max_BS_{BE}.

5.4. Experimental Results

We illustrate our results with a series of figures, which contains the average and maximum bounded slowdown results for $trace_1$, $trace_2$, $trace_3$, and $trace_4$. Due to space constraints we only include maximum slowdown graphs for $trace_1$, $trace_2$; however, the trends in the maximum slowdown graphs for $trace_1$, $trace_2$ were comparable.

In Figures 3 through 6, the top and bottom row show the average OD and average BE bounded slowdowns, respectively. The four subplots in each row are for simulation runs with different %-OD-transfers ranging from 10% to 70%. Then, each individual subplot contains four separate lines, one for each different %-OD-bandwidth value. Finally each %-OD-bandwidth line within a subplot has four datapoints, one for each transfer load ratio. As a result, the x-axes, indicating the transfer load ratios, are the same for all subplots in every traces, and the y-axes, which illustrate bounded slowdown using a logarithmic scale, are shared between the columns within a row.

To compare the performance of our scheduling heuristic, we also ran simulations without our scheduling heuristic as baselines. These baseline simulations have 100% BE transfers and 0%-OD-transfers, and BE and OD bandwidths are 100% and 0% respectively. For each baseline experiment, we computed the performance metrics defined in Section 3.2, and represent these metric values in the figures with a yellow shaded region on each of the plots. Almost all of the OD slowdown plot lines fall inside the shaded region, while the BE slowdown plot lines are above the shaded region, meaning that our algorithm performed better for OD slowdown, but worse for BE slowdown compared to the baseline experiments. Thus, our scheduling algorithm offers improved performance for OD transfer, but it comes at the expense of BE transfer performance.

OD bandwidth is negatively correlated with OD slowdown and positively correlated with BE slowdown, meaning that higher OD bandwidth results in lower OD slowdown but higher BE slowdown. In our experiments, as long as the bandwidth allocated to BE transfers is comparable to the proportion of BE jobs, such as when there

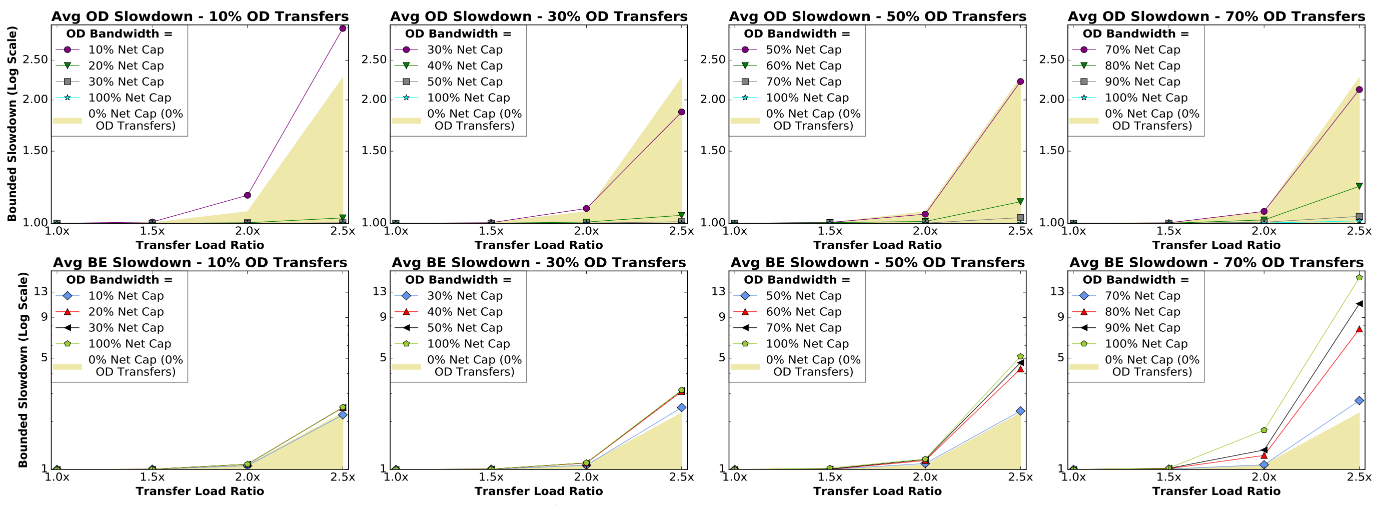


Figure 5: Average bounded slowdown for *trace*$_3$.

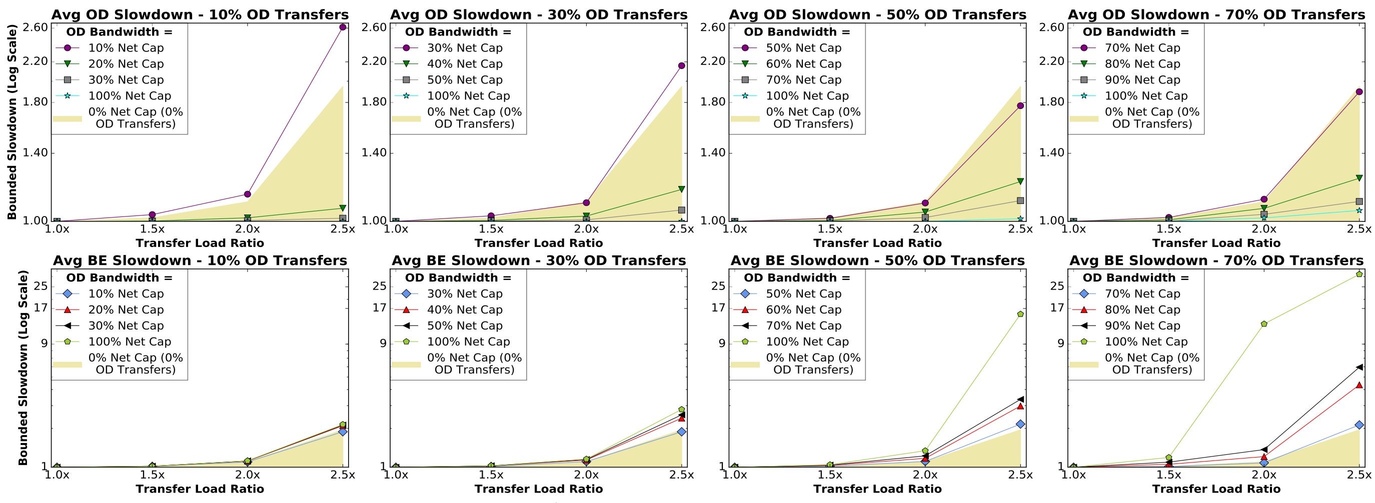


Figure 6: Average bounded slowdown for *trace*$_4$.

are 50% BE transfers and they are allocated 30% of the total bandwidth, the average BE slowdown is less than 2.0, even when the transfer load is doubled. Conversely, in simulations in which OD transfers are allocated 100% of the bandwidth, and thus BE transfers are only given bandwidth when it became available, OD transfers have zero increase in slowdown, but BE transfers have significantly higher slowdown. Thus, it is necessary to consider the requirements and relative importance of OD and BE transfers before choosing parameter values. If the goal is to minimize OD slowdown, it makes sense to allocate 100% of the bandwidth to OD transfers; however, if both OD and BE slowdowns are important, it is necessary to strike a more equal balance between OD and BE transfer bandwidths. Based on our experiments, a %-OD-bandwidth equal to the % of OD transfers + 10% or 20% results in a reasonable balance between OD and BE slowdowns.

As shown in Figures 7 and 8, in our baseline experiments when 100% of transfers are BE, the maximum OD slowdown is under 12.0 even under a 2× load. Note that we present the results for only *trace*$_1$ and *trace*$_2$ as the results of *trace*$_3$ and *trace*$_4$ are similar to that of *trace*$_1$. When the transfers are categorized into OD and BE, the maximum slowdown for BE becomes significantly higher when BE transfers are only given the leftover bandwidth (OD Bandwidth = 100% of available bandwidth) and there are 50% or more OD transfers. However, when OD transfers are only allowed to use 10% or 20% more than their proportion, the maximum slowdown for OD transfers is under 5.0, and the maximum slowdown for BE transfers is under 18.0.

Although there is a correlation between transfer load ratios and bounded slowdown values (higher transfer load ratios, results in higher bounded slowdown values), our results indicate that it is possible to increase the transfer load without significantly affecting transfer slowdown. For example, in the baseline experiments, when the transfer load is doubled and the network capacity fixed at the current level, the averaged slowdown for the transfers is under 1.5×, and under our algorithm with 50%-OD-transfers and 50% of BE transfers, OD transfers experience negligible average slowdown and BE transfer average slowdown is under 2×.

Furthermore, simulations from the different traces resulted in a wide range of slowdown values. In particular, the average slowdowns for *trace*$_1$ are significantly higher

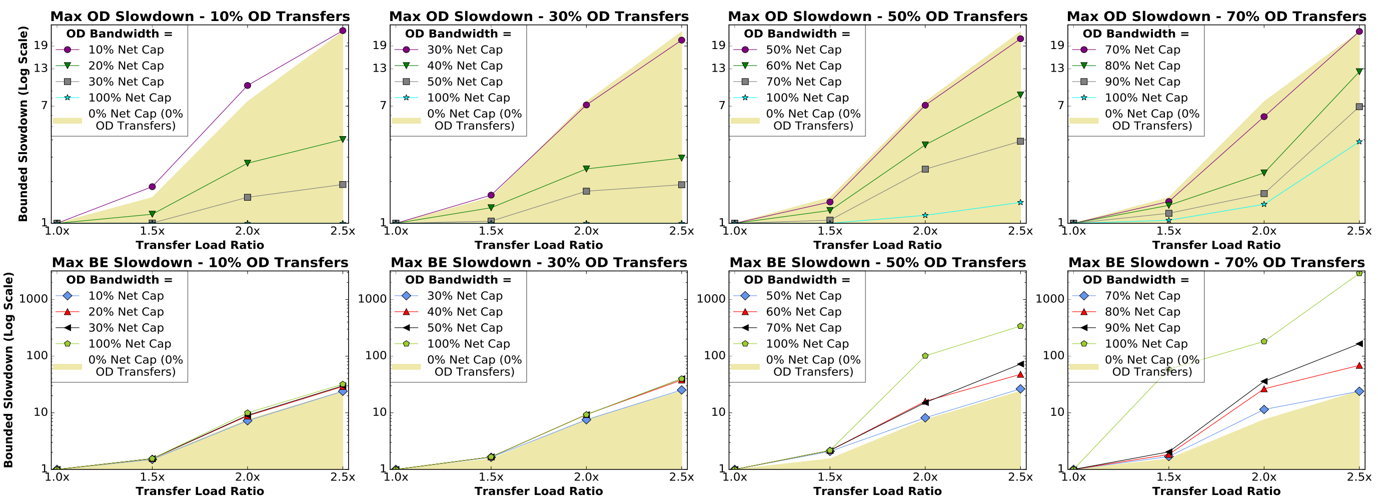


Figure 7: Maximum bounded slowdown for $trace_1$.

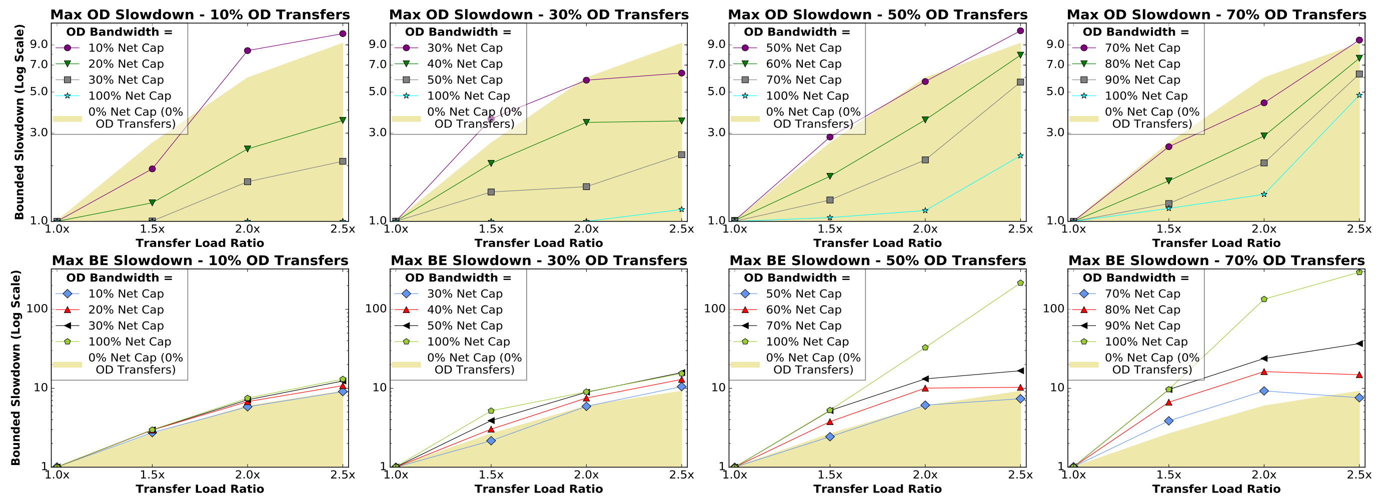


Figure 8: Maximum bounded slowdown for $trace_2$.

than for other traces which is likely the result of $trace_1$ having the highest mean to peak ratio (0.24). Thus, proportionally it has the least amount of additional bandwidth available for extra network traffic. This hypothesis is supported by $trace_2$, which has significantly lower slowdowns than the other traces and also has the lowest mean to peak ratio (0.10), meaning that it relatively has the most bandwidth available for additional network traffic.

Finally, although not shown in the slowdown figures, the linear correlation between transfer load and average throughput means that as the transfer load ratio increases, the average throughput for each trace also increases proportionally. Thus, with a 2× transfer load, the average throughout doubles, and the gap between peak and average throughput for $trace_1$, $trace_2$, $trace_3$, and $trace_4$ decreases by 24%, 10%, 22%, and 16% respectively. Combining this with the bounded slowdown results discussed above indicates that by increasing the transfer load and differentiating transfers in OD and BE categories it is possible to decrease the gap between peak and average throughout without overly sacrificing transfer performance.

5.5. Coefficient of Variation

As shown in Figure 2 there is a large gap between the mean and peak demand in the traces used in our experiments, which is common for science networks. Of the four network demand graphs in Figure 2, the bottom two plots ($trace_3$ and $trace_4$) are relatively similar in terms of mean and peak demand; however, comparing these two traces raises an interesting inconsistency. Although $trace_3$ has a higher mean demand, peak demand, and mean-to-peak ratio than $trace_4$ (Mean: 2.5 Gbps vs. 1.7 Gbps; peak: 11.2 Gbps vs. 10.6 Gbps; mean-to-peak ratio: 0.22 vs. 0.16), in our experiments $trace_3$ had lower OD and BE slowdown values. Comparing the network demand graphs for the two traces in Figure 2 show noticeable differences; while that for $trace_3$ is relatively centered around the mean, that for $trace_4$ is thicker and more jagged, indicating rapid and frequent variations in network demand.

To quantify the difference in network demand variation between the traces, we computed concurrency and throughput coefficient of variation metrics. To do so, we divided the 24-hour simulation period into 1 minute intervals, and computed the number of transfers that occurred during each of the intervals (concurrency intervals), and the total transfer throughput during each of the intervals (throughput intervals). Then, to calculate the coefficient of variation metrics, we used the following equations:

$$\text{Coefficient of Variation}_{concurrency} = \frac{Std.Dev.(\text{Concurrency Intervals})}{mean(\text{Concurrency Intervals})} \tag{5}$$

$$\text{Coefficient of Variation}_{Throughput} = \frac{Std.Dev.(\text{Throughput Intervals})}{mean(\text{Throughput Intervals})} \tag{6}$$

Concurrency and throughput coefficient of variations for $trace_3$ were 0.67 and 0.69 respectively, and concurrency and throughput coefficient of variations for $trace_4$ were 0.87 and 1.07 respectively. We suspect that as the network demand increases, the number of concurrent or overlapping transfers also increases, which raises slowdown values since there are more transfers competing for bandwidth. However, the inverse is not necessarily true, since if the network demand drops below the available network bandwidth, all transfers will have a slowdown of 1, which is the lowest possible slowdown value, and so even if the network demand continues to decrease, slowdown values will remain at 1. Since $trace_4$ has a higher variation in concurrency and throughput, it has more moments of low and high network demand in comparison to $trace_3$, which has a network demand that remains more consistently around the mean. The higher variation and frequent moments of high network demand result in higher slowdown values and therefore worse performance in $trace_4$ than in $trace_3$, even though the mean to peak ratio is higher in $trace_3$.

6. Testbed Experiments

To substantiate the results from the simulation experiments, we performed experiments with our scheduling algorithm over a small physical testbed. In this section, we present the design and implementation of these experiments, and discuss the results and how they corroborate the simulation results from Section 5.

6.1. Testbed Environment

Our testbed comprised a small cluster of CentOS7 virtual machine (VM) instances on the Chameleon Cloud testbed [16], connected via a virtual private network. This virtual cluster comprises 10 nodes: one *server node*; four *source nodes* that transfer data to the server, and five *destination nodes* that receive data from the server. We assign every node a total throughput bandwidth of 1 Gbps (0.5 Gbps inbound, 0.5 Gbps outbound) because we found this to be a throughput that every node could reliably achieve. Since each transfer has the server node as either its source or destination, the server node's inbound and outbound bandwidth limits are the bottlenecks in the network.

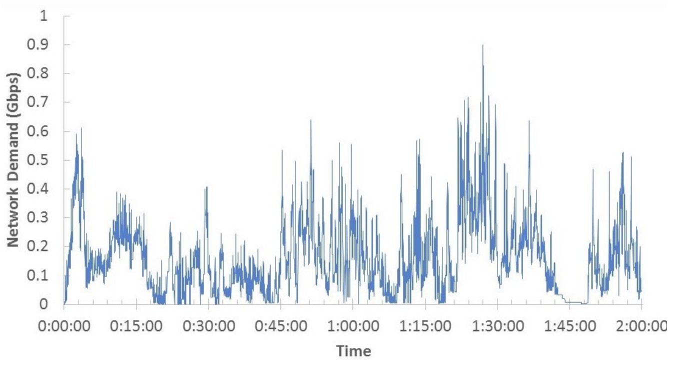


Figure 9: Network demand for $trace_5$

At the start of each experiment, the source and server nodes receive a list of transfers from a trace, with each transfer being defined by a five-tuple (*destination node*, *start time*, *transfer type*, *data size*, *requested rate*). Over the course of the experiment, each node sends each of its transfers to its specified destination node based on the transfer's parameters, and each node manages the allocated bandwidth for all of its active transfers, based on the scheduling algorithm described in Section 4.

6.2. Experiment Workload and Parameters

For the experiments, we used $trace_5$, a log file derived from $trace_2$ described above but spanning two hours instead of 24 as in the original. To make $trace_5$, we divided $trace_2$ into sequential chunks of 12 transfers, randomly chose one transfer from each chunk, and then scaled all of the chosen transfers' start times to fit within a two-hour time period instead of the 24-hour time period in $trace_2$. Thus, a transfer that started at 12:00 in $trace_2$ (i.e., at 50% through $trace_2$'s 24 hours), would start at 1:00 in $trace_5$ (i.e., at 50% through $trace_5$), but it would have the same duration and throughput as in $trace_2$. Since $trace_5$ had 1/12 of the duration and number of transfers as $trace_2$, we were able to conduct shorter experiments than our simulations, while still maintaining the dynamic pattern of network demand shown in $trace_2$.

Finally, to make $trace_5$ fit correctly on our testbed network, we scaled the throughput of all of the transfers in the log by 1/10—the ratio between the maximum network bandwidth in $trace_5$ and the maximum bandwidth available at the server node in our experiments. Figure 9 shows the aggregate throughput for $trace_5$. Since the original trace indicated whether a transfer was incoming or outgoing but not its source or destination (for incoming and outgoing transfers, respectively), we assigned a source node chosen at random to each incoming transfer, and a destination node chosen at random to each outgoing transfers. In addition, for each experiment we randomly partitioned the transfers into OD and BE categories, based on the %-OD-transfers parameter for that experiment.

We tested three different transfer load ratios $\in \{1.0\times, 1.5\times, 2.0\times\}$ on our experiment environment, and for each transfer load ratio, we performed one baseline trial and one experimental trial for a total of $3 \times 2 = 6$ different experiments. In the baseline trials, we treated all transfers as BE transfers. In the experimental trials, we set %-OD-transfers to 30% and %-OD-bandwidth to 40%, which is one of the parameter configurations used in the simulations above. After running each experiment, we calculated the four performance metrics defined in Section 3.2: Avg_BS_{OD}, Max_BS_{OD}, Avg_BS_{BE}, and Max_BS_{BE}.

6.3. Experiment Results

Figure 10 shows the average and maximum bounded slowdown results from our six experiments. In each graph, the x-axes give the transfer load ratios used and the y-axes the observed bounded slowdown values. The dashed orange lines indicate slowdown for BE transfers, the solid blue lines the slowdowns for OD transfers, and the yellow shaded region the baseline slowdowns over all transfers.

We see in both figures some of the same patterns from the simulation results presented in Section 5. There is a correlation between transfer load ratios and slowdown values, i.e., as the transfer load ratio increases, the slowdown values increase for both the baseline and the experimental trials. Furthermore, the experimental scheduling algorithm performs better for OD slowdown but worse for BE slowdown when compared to the baseline experiments, as indicated by the solid blue line being inside the yellow shaded region and the dashed orange line being above it.

More specifically, in the experiment with a 2.0× transfer load ratio, the average on-demand slowdown was only 1.06, which is half the value seen for the baseline under that load. Meanwhile, the average best-effort slowdown value increased, but only modestly (to 1.17), and, though not indicated in the slowdown figures, the gap between the peak and average throughput decreased by 16%. Thus, the results from these experiments corroborate our simulation results from the simulations. When the transfer load is doubled, differentiating the transfers into on-demand and best-effort and giving priority to on-demand reduces the peak and average throughput gap while still keeping average on-demand and best-effort transfer slowdown values under control.

7. Discussion

One question that may arise is whether the algorithm can control the slowdown level of best-effort transfers. We expect the tolerable slowdown level for best-effort transfers to be highly application specific. Thus, our goal was not to implement an algorithm that uses this slowdown level as a parameter, but rather to develop an algorithm that explored what the resulting slowdown values were, given a variety of parameter configurations that could correspond to a number of potential applications. We leave the development of an algorithm that takes the maximum acceptable best-effort slowdown as a parameter as future work. Furthermore, since there is a clear trade-off between slowdown values for on-demand transfers and best-effort transfers based on the % of the bandwidth assigned to each class, simply adding in a parameter for best-effort transfer slowdown could drastically affect on-demand transfer slowdown in extremely high load scenarios. Thus, it might be more pertinent to add a parameter for the ideal on-demand to best-effort transfer slowdown ratio under different load conditions. Such an algorithm is also left as future work.

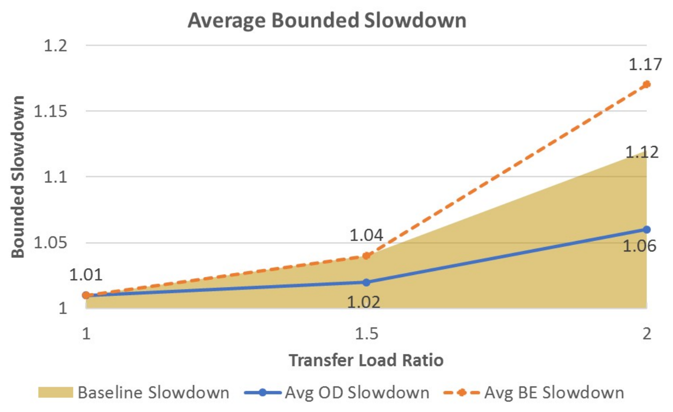
Another question is how to implement the proposed algorithm in the production systems where no log file is available for future transfers, since in both the simulation and the testbed experiments, we use the log file to define an individual transfer's requested rate and then use the summation of the transfers' requested rates to allocate bandwidth. In real systems, we rely on TCP for fairsharing of bandwidth amongst jobs of the same type. Consider an example of 30% OD jobs and +10% bandwidth for OD jobs. When a new OD job arrives, we cap the maximum aggregate bandwidth for BE jobs to 60% of the maximum capacity. We keep monitoring the OD jobs bandwidth usage; if they do not use all 40% allocated, we increase the maximum cap for BE jobs dynamically. Then, when a new OD job arrives, we reset the maximum cap for BE jobs to 60% and keep monitoring the bandwidth usage and adjust this cap as needed. Note that if BE jobs do not use all 60% allocated, OD jobs should automatically be able to use that bandwidth since the 60% is a maximum cap for BE jobs (maximum aggregate bandwidth for all BE jobs).

8. Related Work

Several studies have developed methods of using traffic engineering to improve network performance and efficiency, including locally optimal intradomain traffic engineering [17], globally optimal software driven WAN [18], and differentiating file transfers based on priority level [19]. In particular, studies have explored ways to reduce the adverse efforts of high-rate, bursty traffic associated with large scientific data transfers. Yan et al. [20] used traffic-engineered paths and isolated queues to accomplish this. Our research differentiates itself from previous work by addressing the gap between peak and mean network loads.

Other research has shown that dynamic link width management can be effective for reducing energy costs without negatively affecting sensitive applications involving on-demand transfers [21]. However, this approach involves overprovisioning and building a network to meet the peak demand, and then turning off network links when not in use. In contrast, our approach minimizes the need to overprovision the network by using network traffic scheduling to reduce the gap between peak and mean throughput.

Our research has applications outside of managing traffic on science networks, especially situations that involve a large disparity between the peak and average demand for some resource. For example, electric grids are known to exhibit high variability in demand and overall low efficiency. Several studies have examined how different poli-



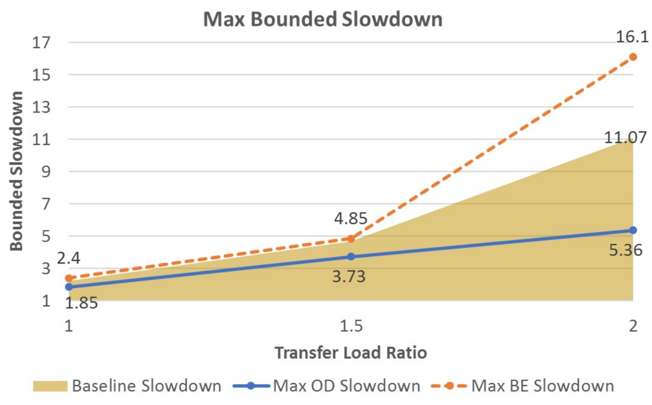


Figure 10: Average (left) and maximum (right) bounded slowdown for $trace_5$.

cies and pricing models can help reduce the peak-to-mean ratio in smart electrical grids [22]. Public cloud computing is another active area of research related to improved resource allocation without overprovisioning, and studies have suggested a number of price models that can be used to reduce peak demand effectively and fairly [23, 24].

9. Conclusions

We presented a study to motivate measures to reduce the substantial gap between peak and average loads in research and education networks. Using logs from production GridFTP servers, we simulated high data transfer loads by keeping the network capacity at current levels while increasing the demands made on the network by individual transfers. We showed that doubling the load while maintaining the current network capacity can reduce the gap between peak and average throughput by an average of 18% while resulting in a minimal impact on data transfer performance, suggesting that there is room to increase load on current networks. We also showed that when the transfers are categorized into on-demand and best-effort, with preferential treatment for on-demand transfers, the impact on the transfers that really need on-demand service can be made negligible, while at the same time keeping the impact on best-effort transfers minimal, under 2.0× on average when the transfer load is doubled.

Acknowledgments

This work was supported in part by the U.S. Department of Energy under contract number DEAC02-06CH11357.

References

[1] "Internet2 IP backbone capacity augment practice," http://www.internet2.edu/policies/ip-backbone-capacity-augment-practice/.

[2] G. Bell and M. Ernst, "HEP Community Summer Study 2013 Computing Frontier: Networking," in *Snowmass'2013*.

[3] K. Bergman, V. Chan, D. Kilper, I. Monga, G. Porter, and K. Rauschenbach, "Scaling terabit networks: Breaking through capacity barriers and lowering cost with new architectures and technologies," *NSF Workshop*, 2013.

[4] "ESnet Strategic Plan," http://www.es.net/assets/Uploads/ESnet-Strategic-Plan-March-2-2013.pdf.

[5] "Biological and Environmental Research Network Requirements, Nov. 2012," http://www.es.net/assets/pubs_presos/BER-Net-Req-Review-2012-Final-Report.pdf.

[6] "Basic Energy Sciences Network Requirements Workshop, September 2010 - Final Report," http://www.es.net/assets/Uploads/BES-Net-Req-Workshop-2010-Final-Report.pdf.

[7] "High Energy & Nuclear Physics Net. Req. Review, Aug. 2013," http://www.es.net/assets/Papers-and-Publications/HEP-NP-Net-Req-2013-Final-Report.pdf.

[8] "LIGO Data Replicator," http://www.lsc-group.phys.uwm.edu/LDR/.

[9] D. Williams *et al.*, "Earth System Grid: Enabling access to multi-model climate simulation data," *Bulletin of American Meteorological Society*, vol. 90, no. 2, 2009.

[10] "Belle-II Experiment Network Requirements, Oct. 2012," http://www.osti.gov/scitech/servlets/purl/1171367.

[11] S. Habib, V. Morozov, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, T. Peterka, J. Insley, D. Daniel, P. Fasel, N. Frontiere, and Z. Lukić, "The universe at extreme scale: Multi-petaflop sky simulation on the BG/Q," in *International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 4:1–4:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389002

[12] I. Monga, C. Guok, W. E. Johnston, and B. Tierney, "Hybrid networks: Lessons learned and future challenges based on ESnet4 experience," *IEEE Communications Magazine*, vol. 49, no. 5, pp. 114–121, May 2011.

[13] Internet2, "Layer 2 services," http://www.internet2.edu/products-services/advanced-networking/layer-2-services/, accessed: 2016-05-04.

[14] "Linux traffic control," http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html, accessed: 2017-03-31.

[15] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," in *Job Scheduling Strategies for Parallel Processing*, ser. IPPS '97. London, UK, UK: Springer-Verlag, 1997, pp. 1–34. [Online]. Available: http://dl.acm.org/citation.cfm?id=646378.689517

[16] J. Mambretti, J. Chen, and F. Yeh, "Next Generation Clouds, the Chameleon Cloud Testbed, and Software Defined Networking (SDN)," in *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, Oct. 2015, pp. 73–79.

[17] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional IP routing protocols," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 118–124, Oct. 2002.

[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-driven WAN," in *ACM SIGCOMM Conference*. New York, NY, USA: ACM, 2013, pp. 15–26. [Online].

Available: http://doi.acm.org/10.1145/2486001.2486012

[19] R. Kettimuthu, G. Agrawal, P. Sadayappan, and I. Foster, “Differentiated Scheduling of Response-Critical and Best-Effort Wide-Area Data Transfers,” in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 1113–1122.

[20] Z. Yan, C. Tracy, M. Veeraraghavan, T. Jin, and Z. Liu, “A network management system for handling scientific data flows,” *Journal of Network and Systems Management*, vol. 24, no. 1, pp. 1–33, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10922-014-9336-2

[21] K. Kant, “Power control of high speed network interconnects in data centers,” in *INFOCOM Workshops 2009, IEEE*, April 2009, pp. 1–6.

[22] H. K. Nguyen, J. B. Song, and Z. Han, “Demand side management to reduce peak-to-average ratio using game theory in smart grid,” in *IEEE Conference on Computer Communications Workshops*, March 2012, pp. 91–96.

[23] N. Nasiriani, C. Wang, G. Kesidis, B. Urgaonkar, L. Y. Chen, and R. Birke, “On fair attribution of costs under peak-based pricing to cloud tenants,” in *23rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Oct 2015, pp. 51–60.

[24] M. Mattess, C. Vecchiola, and R. Buyya, “Managing peak loads by leasing cloud infrastructure services from a spot market,” in *12th IEEE International Conference on High Performance Computing and Communications (HPCC)*, Sept 2010, pp. 180–188.