BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing

Alok Kumar
Nikhil Kasinadhuni
Björn Carlin
Sushant Jain
Enrique Cauich Zermeno
Mihai Amarandei-Stavila
Stephen Stuart
Uday Naik
C. Stephen Gunn
Mathieu Robin
Amin Vahdat
Anand Raghuraman
Jing Ai
Aspi Siganporia

Google Inc.
bwe-sigcomm@google.com

ABSTRACT

WAN bandwidth remains a constrained resource that is economically infeasible to substantially overprovision. Hence, it is important to allocate capacity according to service priority and based on the incremental value of additional allocation. For example, it may be the highest priority for one service to receive 10Gb/s of bandwidth but upon reaching such an allocation, incremental priority may drop sharply favoring allocation to other services. Motivated by the observation that individual flows with fixed priority may not be the ideal basis for bandwidth allocation, we present the design and implementation of Bandwidth Enforcer (BwE), a global, hierarchical bandwidth allocation infrastructure. BwE supports: i) service-level bandwidth allocation following prioritized bandwidth functions where a service can represent an arbitrary collection of flows, ii) independent allocation and delegation policies according to user-defined hierarchy, all accounting for a global view of bandwidth and failure conditions, iii) multi-path forwarding common in traffic-engineered networks, and iv) a central administrative point to override (perhaps faulty) policy during exceptional conditions. BwE has delivered more service-efficient bandwidth utilization and simpler management in production for multiple years.

CCS Concepts

•**Networks** → **Network resources allocation;** *Network management;*

SIGCOMM '15 August 17-21, 2015, London, United Kingdom

ACM ISBN 978-1-4503-3542-3/15/08.

DOI: http://dx.doi.org/10.1145/2785956.2787478

Keywords

Bandwidth Allocation; Wide-Area Networks; Software-Defined Network; Max-Min Fair

1. INTRODUCTION

TCP-based bandwidth allocation to individual flows contending for bandwidth on bottleneck links has served the Internet well for decades. However, this model of bandwidth allocation assumes all flows are of equal priority and that all flows benefit equally from any incremental share of available bandwidth. It implicitly assumes a client-server communication model where a TCP flow captures the communication needs of an application communicating across the Internet.

This paper re-examines bandwidth allocation for an important, emerging trend, distributed computing running across dedicated private WANs in support of cloud computing and service providers. Thousands of simultaneous such applications run across multiple global data centers, with thousands of processes in each data center, each potentially maintaining thousands of individual active connections to remote servers. WAN traffic engineering means that site-pair communication follows different network paths, each with different bottlenecks. Individual services have vastly different bandwidth, latency, and loss requirements.

We present a new WAN bandwidth allocation mechanism supporting distributed computing and data transfer. BwE provides work-conserving bandwidth allocation, hierarchical fairness with flexible policy among competing services, and Service Level Objective (SLO) targets that independently account for bandwidth, latency, and loss.

BwE's key insight is that routers are the wrong place to map policy designs about bandwidth allocation onto per-packet behavior. Routers cannot support the scale and complexity of the necessary mappings, often because the semantics of these mappings cannot be captured in individual packets. Instead, following the End-to-End Argument[28], we push all such mapping to the source host machines. Hosts rate limit their outgoing traffic and mark packets using the DSCP field. Routers use the DSCP marking to determine which

path to use for a packet and which packets to drop when congested. We use global knowledge of network topology and link utilization as input to a hierarchy of bandwidth enforcers, ranging from a global enforcer down to enforcers on each host. Bandwidth allocations and packet marking policy flows down the hierarchy while measures of demand flow up, starting with end hosts. The architecture allows us to decouple the aggregate bandwidth allocated to a flow from the handling of the flow at the routers.

BwE allocates bandwidth to competing applications based on flexible policy configured by *bandwidth functions* capturing application priority and incremental utility from additional bandwidth in different bandwidth regions. BwE supports hierarchical bandwidth allocation and delegation among services while simultaneously accounting for multipath WAN communication. BwE is the principal bandwidth allocation mechanism for one of the largest private WANs and has run in production for multiple years across hundreds of thousands of end points. The systems contributions of our work include:

- Leveraging concepts from Software Defined Networking, we build a unified, hierarchical control plane for bandwidth management extending to all end hosts. In particular, hosts report per-user and per-task demands to the control plane and rate shape a subset of flows.
- We integrate BwE into existing WAN traffic engineering (TE) [17, 11, 12] mechanisms including MPLS Auto-Bandwidth [22] and a custom SDN infrastructure. BwE takes WAN pathing decisions made by a TE service and re-allocates the available site-to-site capacity, split across multiple paths, among competing applications. At the same time, we benefit from the reverse integration: using BwE measures of prioritized application demand as input to TE pathing algorithms (Section 5.3.1).
- We implement hierarchical max-min fair bandwidth allocation to flexibly-defined FlowGroups contending for resources across multiple paths and at different levels of network abstraction. The bandwidth allocation mechanism is both work-conserving and flexible enough to implement a range of network sharing policies.

In sum, BwE delivers a number of compelling advantages. First, it provides isolation among competing services, delivering plentiful capacity in the common case while maintaining required capacity under failure and maintenance scenarios. Capacity available to one service is largely independent of the behavior of other services. Second, administrators have a single point for specifying allocation policy. While pathing, RTT, and capacity can shift substantially, BwE continues to allocate bandwidth according to policy. Finally, BwE enables the WAN to run at higher levels of utilization. By tightly integrating loss-insensitive file transfer protocols running at low priority with BwE, we run many of our WAN links at 90% utilization.

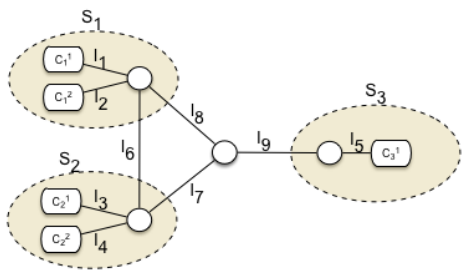


Figure 1: WAN Network Model.

2. BACKGROUND

We begin by describing our WAN environment and highlight the challenges we faced with existing bandwidth allocation mechanisms. Thousands of individual applications and services run across dozens of wide area *sites* each containing multiple *clusters*. Host machines within a cluster share a common LAN. Figure 1 shows an example WAN with sites S_1, S_2 and S_3; C_1^1 and C_1^2 are clusters within site S_1.

We host a combination of interactive web services, e.g. search and web mail, streaming video, batch-style data processing, e.g., MapReduce [13], and large-scale data transfer services, e.g., index copy from one site to another. Cluster management software maps services to hosts independently; we cannot leverage IP address aggregation/prefix to identify a service. However, we can install control software on hosts and leverage a control protocol running outside of routers.

We started with traditional mechanisms for bandwidth allocation such as TCP, QoS and MPLS tunnels. However these proved inadequate for a variety of reasons:

- *Granularity and Scale:* Our network and service capacity planners need to reason with bandwidth allocations at different aggregation levels. For example, a product group may need a specified minimum of site-to-site bandwidth across all services within the product area. In other cases, individual users or services may require a bandwidth guarantee between a specific pair of clusters. We need to scale bandwidth management to thousands of individual services, and product groups across dozens of sites each containing multiple clusters. We need a way to classify and aggregate individual flows into arbitrary groups based on configured policy. TCP fairness is at a 5-tuple flow granularity. On a congested link, an application gets bandwidth proportional to the number of active flows it sends across the links. Our services require guaranteed bandwidth allocation independent of the number of active TCP flows. Router QoS and MPLS tunnels do not scale to the number of service classes we must support and they do not provide sufficient flexibility in allocation policy (see below).
- *Multipath Forwarding:* For efficiency, wide area packet forwarding follows multiple paths through the network, possibly with each path of varying capacity. Routers hash individual service flows to one of the available paths based on packet header content.

Any bandwidth allocation from one site to another must simultaneously account for multiple source/destination paths whereas existing bandwidth allocation mechanisms—TCP, router QoS, MPLS tunnels—focus on different granularity (flows, links, single paths respectively).

- *Flexible and Hierarchical Allocation Policy:* We found simple weighted bandwidth allocation to be inadequate. For example, we may want to give a high priority user a weight of 10.0 until it has been allocated 1 Gb/s, a weight of 1.0 until it is allocated 2 Gb/s and a weight of 0.1 for all bandwidth beyond that. Further, bandwidth allocation should be hierarchical such that bandwidth allocated to a single product group can be subdivided to multiple users, which in turn may be hierarchically allocated to applications, individual hosts and finally flows. Different allocation policies should be available at each level of the hierarchy.

- *Delegation or Attribution:* Applications increasingly leverage computation and communication from a variety of infrastructure services. Consider the case where a service writes data to a storage service, which in turn replicates the content to multiple WAN sites for availability. Since the storage service acts on behalf of thousands of other services, its bandwidth should be charged to the originating user. Bandwidth delegation provides differential treatment across users sharing a service, avoids head of line blocking across traffic for different users, and ensures that the same policies are applied across the network for a user's traffic.

We designed BwE to address the challenges and requirements described above around the principle that bandwidth allocation should be extended all the way to end hosts. While historically we have looked to routers with increasingly sophisticated ASICs and control protocols for WAN bandwidth allocation, we argue that this design point has resulted simply from lack of control over end hosts on the part of network service providers. Assuming such access is available, we find that the following functionality can be supported with a hierarchical control infrastructure extending to end hosts: i) mapping WAN communication back to thousands of flow groups, ii) flexibly sub-dividing aggregate bandwidth allocations back to individual flows, iii) accounting for delegation of resource charging from one service to another, and iv) expressing and enforcing flexible max-min bandwidth sharing policies. On the contrary, existing routers must inherently leverage limited information available only in packet headers to map packets to one of a small number of service classes or tunnels.

Figure 2 shows an instance of very high loss in multiple QoS classes during a capacity reduction on our network. TCP congestion control was not effective and the loss remained high until we turned on admission control on hosts.

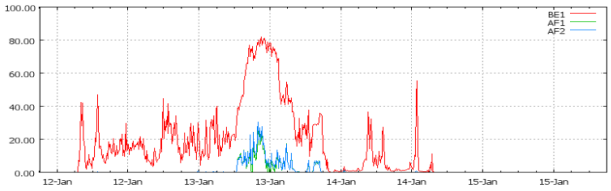


Figure 2: Reduction in TCP packet loss after BwE was deployed. Y-axis denotes packet loss in percentage. Different lines correspond to different QoS classes (BE1 denoting best effort, and AF1/AF2 denoting higher QoS classes.)

3. ABSTRACTIONS AND CONCEPTS

3.1 Traffic Aggregates or FlowGroups

Individual services or users run *jobs* consisting of multiple *tasks*. Each task may contain multiple Linux processes and runs in a Linux container that provides resource accounting, isolation and information about user_name, job_name and task_name. We modified the Linux networking stack to mark the per-packet socket buffer structure to uniquely identify the originating container running the task. This allows BwE to distinguish between traffic from different tasks running on the same host machine.

BwE further classifies task traffic based on destination cluster address. Optionally, tasks use setsockopt() to indicate other classification information, e.g. for *bandwidth delegation*. Delegation allows a task belonging to a shared infrastructure service to attribute its bandwidth to the user whose request caused it to generate traffic. For example, a copy service can delegate bandwidth charges for a specific file transfer to the user requesting the transfer.

For scalability, baseline TCP regulates bandwidth for most application flows. BwE dynamically selects the subset of flows accounting for most of the demand to enforce. Using TCP as the baseline also provides a convenient fallback for bandwidth allocation in the face of a BwE system failure.

BwE allocates bandwidth among *FlowGroups* at various granularities, defined below.

- Task FlowGroup or *task-fg*: <delegating_service, user_name, job_name, task_name, source_cluster, destination_cluster>. This FlowGroup is the finest unit of bandwidth measurement and enforcement.

- Job FlowGroup or *job-fg*: Bandwidth usage across all *task-fg*s belonging to the same job is aggregated into a *job-fg*: <delegating_service, user_name, job_name, source_cluster, destination_cluster>.

- User FlowGroup or *user-fg*: Bandwidth usage across all *job-fg*s belonging to the same user is aggregated into a *user-fg*: <delegating_service, user_name, source_cluster, destination_cluster>.

- Cluster FlowGroup or *cluster-fg*: Bandwidth usage across all *user-fg* belonging to same user_aggregate and belonging to same cluster-pair is combined into a *cluster-fg*: <user_aggregate, source_cluster,

destination_cluster>. The user_aggregate corresponds to an arbitrary grouping of users, typically by business group or product. This mapping is defined in BwE configuration (Section 3.2).

- Site FlowGroup or *site-fg*: Bandwidth usage for *cluster-fg*s belonging to the same site-pair is combined into a *site-fg*: <user_aggregate, source_site, destination_site>.

BwE creates a set of trees of FlowGroups with parent-child relationships starting with *site-fg* at the root to *cluster-fg*, *user-fg*, *job-fg* and eventually *task-fg* at the leaf. We measure bandwidth usage at *task-fg* granularity in the host and aggregate to the *site-fg* level. BwE estimates demand (Section 6.1) for each FlowGroup based on its historical usage. BwE allocates bandwidth to *site-fg*s, which is redistributed down to *task-fg*s and enforced in the host kernel. Beyond rate limiting, the hierarchy can also be used to perform other actions on a flow group such as DSCP remarking. All measurements and rate limiting are done on packet transmit.

BwE policies are defined at *site-fg* and *user-fg* level. Measurement and enforcement happen at *task-fg* level. Other levels are required to scale the system by enabling distributed execution of BwE across multiple machines in Google datacenters.

3.2 Bandwidth Sharing Policies

3.2.1 Requirements

Our WAN (Figure 1) is divided in two levels, the inter-site network and the inter-cluster network. The links in the inter-site network (l_7, l_8 and l_9 in the figure) are the most expensive. Aggregated demands on these links are easier to predict. Hence, our WAN is provisioned at the inter-site network. Product groups (user_aggregates) create bandwidth requirements for each site-pair. For a site-pair, depending on the network capacity and its business priority, each user_aggregate gets approved bandwidth at several allocation levels. Allocation levels are in strict priority order, ex, *Guaranteed* allocation level should be fully satisfied before allocating to *Best-Effort* allocation level. Allocated bandwidth of a user_aggregate for a site-pair is further divided to all its member users.

Even though provisioning and sharing of the inter-site network is the most important, several links not in the inter-site network may also get congested and there is a need to share their bandwidth fairly during congestion. We assign weights to the users that are used to subdivide their user_aggregate's allocated bandwidth in the inter-cluster network. To allow more fine grained control, we allow weights to change based on allocated bandwidth as well as to be overridden to a non-default value for some cluster-pairs.

3.2.2 Configuration

Network administrators configure BwE sharing policies through centralized configuration. BwE configuration specifies a fixed number of strict priority allocation levels, e.g., there may be two levels corresponding to Guaranteed and Best-Effort traffic.

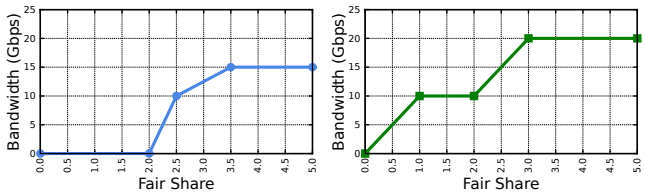
(a) fg_1

Allocation Level	Weight	Bandwidth (Gbps)
Guaranteed	0	0
Best-Effort	20	10
	5	∞

(b) fg_2

Allocation Level	Weight	Bandwidth (Gbps)
Guaranteed	10	10
Best-Effort	10	∞

Table 1: BwE Configuration Example.



(a) fg_1 (b) fg_2

Figure 3: Example Bandwidth Functions.

The BwE configuration maps users to user_aggregates. Mapping from *user-fg* to *site-fg* can be derived from this. The BwE configuration policies describe how *site-fg*s share the network and also describe how *user-fg*s within a *site-fg* share bandwidth allocated to the *site-fg*. For all FlowGroups in a level of hierarchy, the BwE configuration defines: 1) bandwidth for each allocation level and 2) within each allocation level, weight of the FlowGroup that can change based on allocated bandwidth. An example of a BwE configuration for the relative priority for two FlowGroups, fg_1 and fg_2 is shown in Table 1.

3.2.3 Bandwidth Functions

The configured sharing policies are represented inside BwE as *bandwidth functions*[1]. A *bandwidth function* [17] specifies the bandwidth allocation to a FlowGroup as a function of its relative priority on an arbitrary, dimensionless measure of available fair share capacity, which we call *fair share*. *fair share* is an abstract measure and is only used for internal computation by the allocation algorithm. Based on the config, every *site-fg* and *user-fg* is assigned a piece-wise linear monotonic *bandwidth function* (e.g. Figure 3). It is capped at the dynamic estimated demand (Section 6.1) of the FlowGroup. They can also be aggregated to create *bandwidth functions* at the higher levels (Section 3.2.4).

The *fair share* dimension can be partitioned into regions (corresponding to allocation levels in the BwE configuration) of strict priority. Within each region, the slope[2] of a FlowGroup's *bandwidth function* defines its relative priority or weight. Once the bandwidth reaches the maximum approved for the FlowGroup in a region, the *bandwidth function* flattens (0 slope) until the start of the next region. Once the

[1]Bandwidth functions are similar to *utility functions* [8, 6] except that these are derived from static configured policy (Section 3.2) indicating network fair share rather than application-specified utility as a function of allocated bandwidth.

[2]Slope can be a multiple of weight as long as the same multiple is used for all FlowGroups.

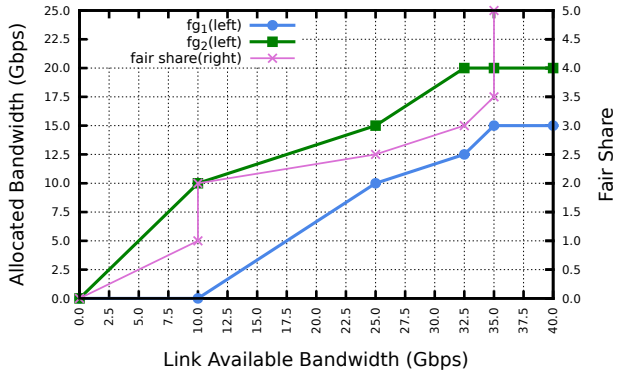


Figure 4: Bandwidth Sharing on a Bottleneck Link.

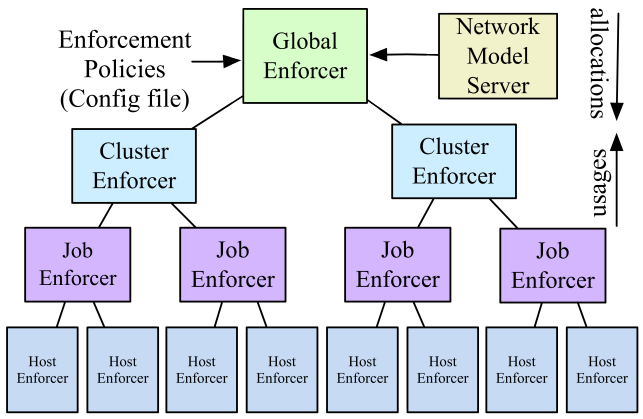


Figure 5: BwE Architecture.

bandwidth function reaches the FlowGroup's estimated demand, it becomes flat from that point for all the following regions.

Figure 3 shows example *bandwidth functions* for two FlowGroups, fg_1 and fg_2, based on BwE configuration as defined in Table 1. There are two regions of *fair share*: Guaranteed (0-2) and Best-Effort (2-∞). The endpoints for each region are system-level constants defined in BwE configuration. BwE's estimated demand of fg_1 is 15Gbps and hence, its *bandwidth function* flattens past that point. Similarly, fg_2's estimated demand is 20Gbps.

We present a scenario where fg_1 and fg_2 are sharing one constrained link in the network. The goal of the BwE algorithm is to allocate the bandwidth of the constrained link such the following constraints are satisfied: 1) fg_1 and fg_2 get maximum possible but equal *fair share*, and 2) sum of their allocated bandwidth corresponding to the allocated *fair share* is less than or equal to the available bandwidth of the link. Figure 4 shows the output of the BwE allocation algorithm (Section 5.3) with varying link's available bandwidth shown on the x-axis. The allocated *fair share* to the FlowGroups is shown on the right y-axis and the corresponding bandwidth allocated to the FlowGroups is shown on the left y-axis. Note that the constraints above are always satisfied at each snapshot of link's available bandwidth. One can verify using this graph that the prioritization as defined by Table 1 is respected.

One of BwE's principal responsibilities is to dynamically determine the level of contention for a particular resource (bandwidth) and to then assign the resource to all competing FlowGroups based on current contention. Higher values of *fair share* indicate lower levels of resource contention and correspondingly higher levels of bandwidth that can potentially be assigned to a FlowGroup. Actual consumption is capped by current FlowGroup estimated demand, making the allocation work-conserving (do not waste any available bandwidth if there is demand).

The objective of BwE is the max-min fair [6] allocation of *fair share* to competing *site-fg*s and then the max-min fair allocation of *fair share* to *user-fg*s within a *site-fg*. For each *user-fg*, maximize the utilization of the allocated bandwidth to the *user-fg* by subdividing it to the lower levels of hierarchy (*job-fg*s and *task-fg*s) equally (no weights) based on their estimated demands.

3.2.4 *Bandwidth Function Aggregation*

*Bandwidth Function*s can be aggregated from one FlowGroup level to another higher level. We require such aggregation when input configuration defines a *bandwidth function* at a finer granularity, but the BwE algorithm runs over coarser granularity FlowGroups. For example, BwE's input configuration provides *bandwidth function* at *user-fg* level, while BwE (Section 5.1) runs across *cluster-fg*s. In this case, we aggregate *user-fg*s *bandwidth function*s to create a *cluster-fg bandwidth function*. We create aggregated *bandwidth functions* for a FlowGroup by adding bandwidth value for each value of *fair share* for all its children.

4. SYSTEM DESIGN

BwE consists of a hierarchy of components that aggregate network usage statistics and enforce bandwidth allocations. BwE obtains topology and other network state from a network model server and bandwidth sharing policies from an administrator-specified configuration. Figure 5 shows the functional components in BwE.

4.1 Host Enforcer

At the lowest level of the BwE hierarchy, the Host Enforcer runs as a user space daemon on end hosts. Every five seconds, it reports bandwidth usage of local application's *task-fg*s to the Job Enforcer. In response, it receives bandwidth allocations for its *task-fg*s from the Job Enforcer. The Host Enforcer collects measurements and enforces bandwidth allocations using the HTB (Hierarchical Token Bucket) queuing discipline in Linux.

4.2 Job Enforcer

Job Enforcers aggregate usages from *task-fg*s to *job-fg*s and report *job-fg*s' usages every 10 seconds to the Cluster Enforcer. In response, the Job Enforcer receives *job-fg*s' bandwidth allocations from the Cluster Enforcer. The Job Enforcer ensures that for each *job-fg*, bandwidth usage does not ex-

ceed its assigned allocation. To do so, it redistributes the assigned *job-fg* allocation among the constituent *task-fg*s using the WaterFill algorithm (Section 5.4).

4.3 Cluster Enforcer

The Cluster Enforcer manages two levels of FlowGroup aggregation - *user-fg*s to *job-fg*s and *cluster-fg*s to *user-fg*s. It aggregates usages from *job-fg*s to *user-fg*s and computes *user-fg*s' *bandwidth function*s based on input from a configuration file. It aggregates the *user-fg*s' *bandwidth function*s (capped at their estimated demand) to *cluster-fg* bandwidth functions (Section 3.2.4), reporting them every 15 seconds to the Global Enforcer. In response, the Cluster Enforcer receives *cluster-fg*s' bandwidth allocations, which it redistributes among *user-fg*s and subsequently to *job-fg*s (Section 5.4).

4.4 Network Model Server

The Network Model Server builds the abstract network model for BwE. Network information is collected by standard monitoring mechanisms (such as SNMP). Freshness is critical since paths change dynamically. BwE targets getting an update every 30 seconds. The consistency of the model is verified using independent mechanisms such as traceroute.

4.5 Global Enforcer

The Global Enforcer sits at the top of the Bandwidth Enforcer hierarchy. It divides available bandwidth capacity on the network between different clusters. The Global Enforcer takes the following inputs: i) *bandwidth function*s from the Cluster Enforcers summarizing priority across all users at *cluster-fg* level, ii) global configuration describing the sharing policies at *site-fg* level, and iii) network topology, link capacity, link utilization and drop statistics from the network model server. A small fraction of flows going over a link may not be under BwE control. To handle this, for every link we also compute *dark bandwidth*. This is the amount of traffic going over the link which BwE is unaware of. This may be due to packet header overhead (particularly tunneling in network routers) or various failure conditions where BwE has incomplete information. Dark bandwidth is the smoothed value of (actual link usage - BwE reported link usage), and link allocatable capacity is (link capacity - dark bandwidth). BwE reported link usage is computed by taking the set of flows (and their current usage) reported to the Global Enforcer by Cluster Enforcers, and mapping them to the paths and links for those flows. Given these inputs, the Global Enforcer runs hierarchical *MPFA* (Section 5.3) to compute *cluster-fg*s' bandwidth allocations and sends these allocations to Cluster Enforcers.

5. BWE ALLOCATION ALGORITHM

One of the challenges we faced was defining the optimization objective for bandwidth allocation to individual flows. First, we did not wish to allocate bandwidth among competing 5-tuple flows but rather to competing FlowGroups. Second, services do not compete for bandwidth at a single bottleneck link because services communicate from multiple clusters to multiple other clusters, with each cluster pair utilizing multiple paths. Hence, the bandwidth allocation must simultaneously account for multiple potential bottlenecks.

Here, we present an adaptation of the traditional max-min fairness objective for FlowGroups sharing a bottleneck link to multipath cluster-to-cluster communication. We designed a centralized *MultiPath Fair Allocation (MPFA)* algorithm to determine global max-min fairness. We present a simpler version of the problem with a single layer of FlowGroups (Section 5.2) and then extend it to multiple layers of FlowGroups with different network abstractions in hierarchical *MPFA* (Section 5.5).

5.1 Inputs and Outputs

Inputs to the BwE algorithm are *task-fg*s' demands, *bandwidth function*s of *user-fg*s and *site-fg*s and network paths for *cluster-fg*s and *site-fg*s. We aggregate *task-fg*s' demands all the way up to *site-fg*s and aggregate *user-fg*s' *bandwidth function*s to *cluster-fg*s' *bandwidth function*s (Section 3.2.4). We run global hierarchical *MPFA* (Section 5.5) on *site-fg*s and *cluster-fg*s that results in *cluster-fg*s' allocations. Then, we distribute *cluster-fg*s' allocations to *task-fg*s (Section 5.4), which are enforced at the hosts.

5.2 MPFA Problem

Inputs for *MPFA* are:

1. Set of n FlowGroups, $\mathcal{F} = \{f_i,\ \forall i \mid 1 \le i \le n\}$ where FlowGroups are defined in Section 3.1. Each f_i has an associated *bandwidth function* (Section 3.2.3), B_{f_i}. B_{f_i} maps *fair share* to bandwidth for f_i. If f_i is allocated *fair share* of s, then it should be allocated bandwidth equal to $B_{f_i}(s)$.
2. Set of m links, $\mathcal{L} = \{l_k,\ \forall k \mid 1 \le k \le m\}$. Each link l_k has an associated allocatable capacity c_{l_k}.
3. Set of n_{f_i} paths for each f_i. Each path, $p_j^{f_i}$, has an associated weight, $w_j^{f_i}$, where $1 \le j \le n_{f_i}$ and for each f_i, $\sum_{1 \le j \le n_{f_i}} w_j^{f_i} = 1$. Each path, $p_j^{f_i}$, is a set of links, i.e, $p_j^{f_i} \subseteq \mathcal{L}$.

We define the fraction of f_i that traverse l_k as $FR(f_i, l_k)$. This is calculated as the sum of weights, $w_j^{f_i}$, for all paths, $p_j^{f_i}$, for the FlowGroup, f_i, such that $l_k \in p_j^{f_i}$.

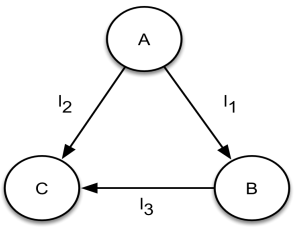
$$FR(f_i, l_k) = \sum_{1 \le j \le n_{f_i} | l_k \in p_j^{f_i}} w_j^{f_i}$$

The output of *MPFA* is the max-min *fair share* allocation s_{f_i} to each FlowGroup, f_i, such that ascending sorted $(s_{f_1}, s_{f_2}, \ldots, s_{f_n})$ is maximized in lexicographical order. Such maximization is subject to the constraint of satisfying capacity constraints for all links, l_k.

$$\sum_{\forall f_i} FR(f_i, l_k) \times B_{f_i}(s_{f_i}) \le c_{l_k}$$

5.3 MPFA Algorithm

The *MPFA* algorithm (Algorithm 1) can be described in the following high-level steps:



Flow Group	Src	Dst	Paths
f_1	A	C	l_2: 0.75 $l_1 \rightarrow l_3$: 0.25
f_2	A	C	l_2: 1.00
f_3	A	B	l_1: 1.00

Figure 6: MPFA Example.

1. For each link, l_k, calculate the link's *bandwidth function*, B_{l_k}, by aggregating *bandwidth functions* of all non-frozen[3] FlowGroups, f_i, in appropriate fractions, $FR(f_i, l_k)$. B_{l_k} maps *fair share*, s, to allocation on the link l_k when all FlowGroups traversing l_k are allocated the *fair share* of s.

2. Find the bottleneck *fair share*, $s^b_{l_k}$, for each remaining (not bottlenecked yet) link, l_k, by finding the *fair share* corresponding to its capacity, c_{l_k} in the link's *bandwidth function*, B_{l_k}. Since *bandwidth function* is a piece-wise linear monotonic function, finding *fair share* for a given capacity can be achieved by a binary search of the interesting points (points where the slope of the function changes).

3. The link, l_b, with the minimum bottleneck *fair share*, s^{min} is the next bottleneck. If the minimum bottleneck *fair share* equals ∞, then terminate.

4. Mark the link, l_b, as a bottleneck link. Freeze all FlowGroups, f_i, with non-zero fraction, $FR(f_i, l_b)$, on l_b. Frozen FlowGroups are not considered to find further bottleneck links. Subtract frozen FlowGroups' *bandwidth functions* beyond the bottleneck *fair share*,s^{min} from all remaining links.

5. If any link is not a bottleneck, continue to step 2.

Figure 6 shows an example of the allocation algorithm with three links, l_1, l_2 and l_3, with capacity 13, 13 and 4 respectively. Assume all bandwidth numbers are in Gbps for this example. There are three FlowGroups: 1) f_1 takes two paths (l_2 and $l_1 \rightarrow l_3$) with weights 0.75 and 0.25 respectively, 2) f_2 takes one path (l_2), and 3) f_3 taking one path (l_1). All FlowGroups have demand of 18. Assume f_1, f_2 and f_3 have weights of 1, 2 and 3 respectively, corresponding *bandwidth functions* of the FlowGroups are: $B_{f_1}(s) = \min(18, s)$, $B_{f_2}(s) = \min(18, 2s)$ and $B_{f_3}(s) = \min(18, 3s)$.

Based on that paths, fraction of FlowGroups(f_i) traversing Links(l_k) are: $FR(f_1, l_1) = 0.25$, $FR(f_1, l_2) = 0.75$, $FR(f_1, l_3) = 0.25$, $FR(f_2, l_2) = 1$ and $FR(f_3, l_1) = 1$.

We calculate *bandwidth function* for links as:

$$B_{l_1}(s) = \begin{pmatrix} 0.25(\min(18, s)) \\ +\min(18, 3s) \end{pmatrix} = \begin{cases} 3.25s & : 0 \le s < 6 \\ 0.25s + 18 & : 6 \le s < 18 \\ 22.5 & : s \ge 18 \end{cases}$$

[3]A frozen FlowGroup is a FlowGroup that is already bottlenecked at a link and does not participate in the *MPFA* algorithm run any further.

Input:
FlowGroups, $\mathcal{F} : \{f_i, \forall i \mid 1 \le i \le n\}$;
Links, $\mathcal{L} : \{l_k, \forall k \mid 1 \le k \le m\}$;
Allocatable capacities for $\forall l_k$: $\{c_{l_k}, \forall k \mid 1 \le k \le m\}$;
bandwidth function for $f_i : B_{f_i}$;
// $\forall f_i, \forall l_k$, Fraction of f_i traversing link, l_k
Function, $FR(f_i, l_k)$: Output is a fraction ≤ 1;

Output:
Allocated fair share for $\forall f_i : \{s_{f_i}, \forall i \mid 1 \le i \le n\}$;

Bottleneck Links , $\mathcal{L}^b \leftarrow \varnothing$;
Frozen FlowGroups, $\mathcal{F}^f \leftarrow \varnothing$;
foreach f_i **do** $s_{f_i} \leftarrow \infty$;
// Calculate *bandwidth function* for each l_k
foreach l_k **do** $\forall s, B_{l_k}(s) \leftarrow \sum_{\forall f_i} FR(f_i, l_k) \times B_{f_i}(s)$;
while $(\exists l_k \mid l_k \notin \mathcal{L}^b) \wedge (\exists f_i \mid f_i \notin \mathcal{F}^f)$ **do**
  Bottleneck link, $l_b \leftarrow null$;
  Min Bottleneck fair share, $s^{min} \leftarrow \infty$;
  foreach $l_k \notin \mathcal{L}^b$ **do**
    Find $s^b_{l_k} \mid c_{l_k} = B_{l_k}(s^b_{l_k})$;
    if $s^b_{l_k} < s^{min}$ **then** $s^{min} \leftarrow s^b_{l_k}$; $l_b \leftarrow l_k$;
  if $l_b \neq null$ **then** Add l_b to $\mathcal{L}^b$;
  else break;
  // Freeze f_i taking the bottleneck link, l_b
  foreach $f_i \mid FR(f_i, l_b) > 0 \wedge f_i \notin \mathcal{F}^f$ **do**
    Add f_i to $\mathcal{F}^f$; $s_{f_i} \leftarrow s^{min}$;
    // Remove allocated bandwidth from B_{f_i}
    $\forall s, B_{f_i}(s) \leftarrow \max(0, B_{f_i}(s) - B_{f_i}(s^{min}))$;
    // Subtract B_{f_i} from B_{l_k} for all its links
    foreach $l_k \mid FR(f_i, l_k) > 0 \wedge l_k \notin \mathcal{L}^b$ **do**
      $\forall s, B_{l_k}(s) \leftarrow B_{l_k}(s) - FR(f_i, l_k) \times B_{f_i}(s)$;

Algorithm 1: MPFA Algorithm

$$B_{l_2}(s) = \begin{pmatrix} 0.75(\min(18, s)) \\ +\min(18, 2s) \end{pmatrix} = \begin{cases} 2.75s & : 0 \le s < 9 \\ 0.75s + 18 & : 9 \le s < 18 \\ 31.5 & : s \ge 18 \end{cases}$$

$$B_{l_3}(s) = \quad 0.25(\min(18, s)) = \begin{cases} 0.25s & : 0 \le s < 18 \\ 4.5 & : s \ge 18 \end{cases}$$

Next, we find bottleneck *fair share*, $s^b_{l_k}$ for each link, l_k, such that $B_{l_k}(s^b_{l_k}) = c_{l_k}$. This results in $s^b_{l_1} = 4$, $s^b_{l_2} \approx 4.72$, $s^b_{l_3} = 16$. This makes l_1 the bottleneck link and freezes both f_1 and f_3 at *fair share* of 4. l_1 will not further participate in *MPFA*. Since f_1 is frozen at *fair share* of 4, B_{l_2} and B_{l_3} need to be updated to not account for B_{f_1} beyond *fair share* of 4. The updated functions are:

$$B_{l_2}(s) = \begin{cases} 2.75s & : 0 \le s < 4 \\ 2s + 3 & : 4 \le s < 9 \\ 21 & : s \ge 9 \end{cases}$$

$$B_{l_3}(s) = \begin{cases} 0.25s & : 0 \le s < 4 \\ 1 & : s \ge 4 \end{cases}$$

We recalculate $s^b_{l_2}$ and $s^b_{l_3}$ based on the new values for B_{l_2} and B_{l_3}. This results in $s^b_{l_2} = 5$ and $s^b_{l_3} = \infty$. l_2 is the next bottleneck with *fair share* of 5. f_2 is now frozen at the *fair share* of

5. Since all FlowGroups are frozen, MPFA terminates. The final allocation to (f_1, f_2, f_3) in *fair share* is (4, 5, 4), translating to (4Gbps, 10Gbps, 12Gbps) using the corresponding *bandwidth functions*. This allocation fills bottleneck links, l_1 and l_2 completely and *fair share* allocation (4, 5, 4) is max-min fair with the given pathing constraints. No FlowGroup's allocation can be increased without penalizing other FlowGroups with lower or equal *fair share*.

5.3.1 Interaction with Traffic Engineering (TE)

The BwE algorithm takes paths and their weights as input. A separate system, TE [17, 11, 12], is responsible for finding optimal pathing that improves BwE allocation. Both BwE and TE are trying to optimize network throughput in a fair way and input flows are known in advance. However, the key difference is that in BwE problem formulation, paths and their weights are input constraints, where-as for TE [17, 11, 12], paths and their weights are output. In our network, we treat TE and BwE as independent problems.

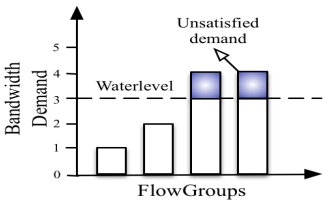
TE has more degrees of freedom and hence can achieve higher fairness. In the above example, the final allocation can be more max-min fair if f_1 only uses the path l_2. In this case, *MPFA* will allocate *fair share* to flow groups ≈ (4.33, 4.33, 4,33) with corresponding bandwidth of (4.33Gbps, 8.66Gbps, 13Gbps). Hence, a good traffic engineering solution results in better (more max-min fair) BwE allocations.

We run TE [17] and BwE independently because they work at different time-scales and different topology granularity. Since TE is more complex, we aggregate topology to site-level where-as for BwE, we are able to run at a more granular cluster-level topology. TE re-optimizes network less often because changing network paths may result in packet reordering, transient loss [16] and resulting routing changes may add significant load to network routers. Separation of TE and BwE also gives us operational flexibility. The fact that both systems have the same higher level objective function helps ensure that their decisions are aligned and efficient. Even though in our network we run these independently the possibility of having a single system to do both can not be ruled out in future.

5.4 Allocation Distribution

MPFA allocates bandwidth to the highest level of aggregations, *site-fg*s. This allocation needs to be distributed to lower levels of aggregation. Distribution of allocation from *cluster-fg* to lower levels is simpler since the network abstraction does not change and the set of paths remains the same during de-aggregation. We describe such distributions in this section. The distribution from *site-fg* to *cluster-fg* is more complex since the network abstraction changes from site-level to cluster-level (Figure 1), requiring an extension of *MPFA* to Hierarchical *MPFA* (Section 5.5) to allocate bandwidth directly to *cluster-fg*s while honoring fairness and network abstractions at *site-fg* and *cluster-fg* level.

To distribute allocation from a *cluster-fg* to *user-fg*s, we calculate the aggregated *bandwidth function*s for the *cluster-fg*s (Section 3.2.4) and determine the *fair share*, s^u, corresponding to the *cluster-fg*'s bandwidth allocation. We use s^u to look up the bandwidth allocation for each *user-fg* using its *bandwidth function*.



Total Resource = 9
Waterlevel = 3

Demand	Allocation
1	1
2	2
4	3
4	3

Figure 7: Allocation Using WaterFill.

Bandwidth distribution from a *user-fg* to *job-fg*s and from a *job-fg* to *task-fg*s is simple max-min fair allocation of one resource to several competing FlowGroups using a *WaterFill* as shown in Figure 7. WaterFill calculates the water level corresponding to the maximum allocation to any FlowGroup.

The allocation to each child FlowGroup is $min(demand, waterlevel)$. If there is excess bandwidth still remaining after running the WaterFill, it is divided among the FlowGroups as bonus bandwidth. Since some (or a majority) of the FlowGroups will not use the bonus assigned to them, the bonus is over-allocated by a configurable scaling factor.

5.5 Hierarchical MPFA

Next, we describe hierarchical *MPFA*, which reconciles the complexity between *site-fg* and *cluster-fg* level allocation. The fairness goal is to allocate max-min *fair share* to *site-fg* respecting bandwidth functions and simultaneously observing inter-site and intra-site topological constraints (Figure 1). Because not all *cluster-fg*s within a *site-fg* share the same WAN paths, individual *cluster-fg*s within a *site-fg* may bottleneck on different intra-site links.

We motivate hierarchical fairness using an example based on Figure 1. All links have 100Gbps capacity, except l_1 (5Gbps) and l_9 (40Gbps). There are two *site-fg*s, sf_1 from S_1 to S_3 and sf_2 from S_2 to S_3. sf_1 consists of *cluster-fg*s: cf_1 from C_1^1 to C_3^1 and cf_2 from C_1^2 to C_3^1. sf_2 consists of a *cluster-fg*: cf_3 from C_2^1 to C_3^1. All *site-fg*s have equal weights and for each *site-fg*, all its member *cluster-fg*s have equal weights. cf_1 and cf_3 have 100Gbps of demand while cf_2 has a 5Gbps demand. If we run *MPFA* naively on *site-fg*s, then sf_1 and sf_2 will be allocated 20Gbps each due to the bottleneck link, l_9. However, when we further subdivide sf_1's 20Gbps among cf_1 and cf_2, cf_1 only receives 5Gbps due to the bottleneck link l_1 while cf_2 only has demand of 5Gbps. cf_3 receives all of sf_2's 20Gbps allocation.

With this naive approach, the final total allocation on l_9 is 30Gbps wasting 10Gbps, where cf_3 could have used the extra 10Gbps. Allocation at the site level must account for independent bottlenecks in the topology one layer down. Hence, we present an efficient hierarchical MPFA to allocate max-min fair bandwidth among *site-fg*s while accounting for cluster-level topology and fairness among *cluster-fg*s.

The goals of hierarchical *MPFA* are:

- Ensure max-min fairness of *fair share* across *site-fg* based on *site-fg*s' *bandwidth function*s.
- Within a *site-fg*, ensure max-min fairness of *fair share* across *cluster-fg*s using *cluster-fg*s' *bandwidth function*s.
- The algorithm should be work-conserving.
- The algorithm should not over-allocate any link in the network, hence, should enforce capacity constraints of intra-site and inter-site links.

For hierarchical *MPFA*, we must run *MPFA* on all *cluster-fg*s to ensure that bottleneck links are fully utilized and enforced. To do so, we must create effective *bandwidth function*s for *cluster-fg*s such that the fairness among *site-fg*s and fairness within a *site-fg* are honored.

We enhance *MPFA* in the following way. In addition to *bandwidth function*, B_{cf_i}, for *cluster-fg*, cf_i, we further consider the *bandwidth function*, B_{sf_x} for *site-fg*, sf_x. Using $\forall i, B_{cf_i}$ and $\forall x, B_{sf_x}$, we derive the effective *bandwidth function*, $B^e_{cf_i}$, for cf_i.

We create $B^e_{cf_i}$ by transforming B_{cf_i} along the *fair share* dimension while preserving the relative priorities of cf_i with respect to each other. We call bandwidth values of different cf_i as equivalent if they map to the same *fair share* based on their respective *bandwidth function*s. To preserve relative priorities of $\forall cf_i \in sf_x$, the set of equivalent bandwidth values should be identical before and after the *bandwidth function*s transformation. Any transformation applied in *fair share* should preserve this property as long as the same transformation is applied to all $cf_i \in sf_x$. Allocated bandwidth to each cf_i on a given available capacity (e.g. Figure 4) should be unchanged due to such transformation. In addition, we must find a transformation such that when all $cf_i \in sf_x$ use their effective (transformed) *bandwidth function*s, $B^e_{cf_i}$, they can together exactly replace sf_x. This means that when $B^e_{cf_i}$ are added together, it equals B_{sf_x}. $\forall s, \sum_{\forall i | cf_i \in sf_x} B^e_{cf_i}(s) = B_{sf_x}(s)$.

The steps to create $B^e_{cf_i}$ are:

1. For each *site-fg*, sf_x, create aggregated *bandwidth function*, $B^a_{sf_x}$ (Section 3.2.4):

$$\forall s, B^a_{sf_x}(s) = \sum_{\forall cf_i \in sf_x} B_{cf_i}(s)$$

2. Find a transformation function of *fair share* from $B^a_{sf_x}$ to B_{sf_x}. The transformation function, T_x is defined as:

$$T_x(s) = \bar{s} \mid B^a_{sf_x}(s) = B_{sf_x}(\bar{s})$$

   Note that since *bandwidth function* is piece-wise linear monotonic function, just find $T_x(s)$ for values for interesting points (where slope changes in either $B^a_{sf_x}$ or B_{sf_x}).

3. For each $cf_i \in sf_x$, apply T_x on *fair share* dimension of B_{cf_i} to get $B^e_{cf_i}$.

$$B^e_{cf_i}(T_x(s)) = B_{cf_i}(s)$$

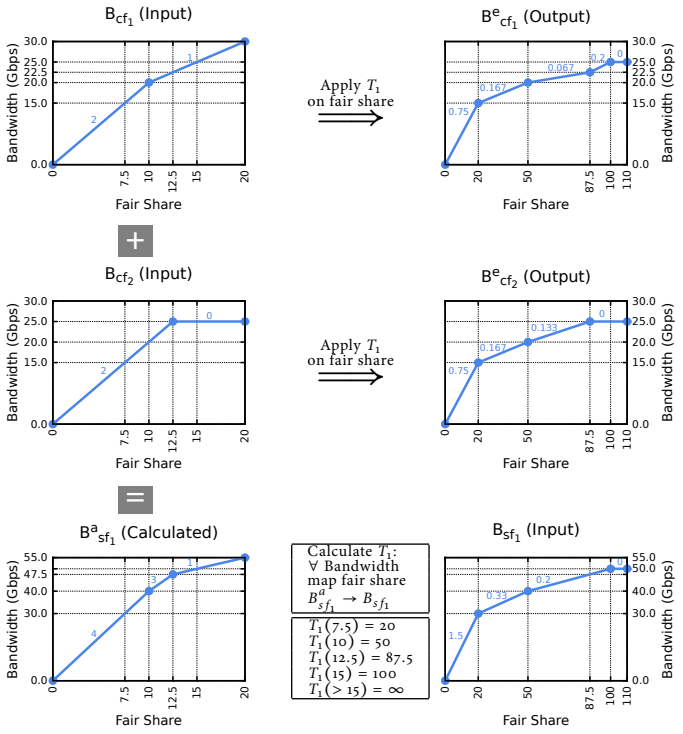


Figure 8: Bandwidth Function Transformation Example

Again, just applying the transformation at the interesting points (points where the slope of the function changes) is sufficient.

An example of creating effective *bandwidth function* is shown in Figure 8. *MPFA* algorithm as described in Section 5.3 is run over *cluster-fg*s as FlowGroups with their effective *bandwidth function*s to achieve hierarchical fairness.

When we run hierarchical *MPFA* in the topology shown in Figure 1, the allocation to cf_3 increases to 30Gbps, fully using bottleneck link l_9. However, if cf_2 has higher demand (say 100Gbps), then it will not receive benefit of cf_1 being bottlenecked early and sf_1 will not receive its full fair share of 20Gbps. To resolve this, we rerun the *bandwidth function* transformation for a *site-fg* when any of its member *cluster-fg*s is frozen due to an intra-site bottleneck link.

6. SYSTEM IMPLEMENTATION

This section describes various insights, design and implementation considerations that made BwE a practical and useful system.

6.1 Demand Estimation

Estimating demand correctly is important for fair allocation and high network utilization. Estimated demand should be greater than current usage to allow each FlowGroup to ramp its bandwidth use. But high estimated demand (compared to usage) of a high priority FlowGroup can waste bandwidth. In our experience, asking users to estimate their demand is untenable because user estimates are wildly inaccurate. Hence, BwE employs actual, near real-time measure-

ments of application usage to estimate demand. BwE estimates FlowGroup demand by maintaining usage history: $Demand = \max(\max_{\Delta t}(usage) \times scale,\ min\_demand)$

We take the peak of a FlowGroup's usage across Δt time interval, multiply it with a factor $scale > 1$ and take the max with min_demand. Without the concept of min_demand, small flows (few Kbps) would ramp to their real demand too slowly. Empirically, we found that $\Delta t = 120s$, $scale = 1.1$ and $min\_demand = 10Mbps$ works well for *user-fg* for our network applications. We use different values of min_demand at different levels of the hierarchy.

6.2 WaterFill Allocation For Bursty Flows

The demands used in our WaterFill algorithm (Section 5.4) are based on peak historical usage and different child FlowGroups can peak at different times. This results in demand over-estimation and subsequently the WaterFill allocations can be too conservative. To account for burstiness and the resulting statistical multiplexing, we estimate a *burstiness factor* (≥ 1) for each FlowGroup based on its demand and sum of its children's demand:

$$burstiness\ factor = \frac{\sum_{\forall children} estimated\ demand}{parent's\ estimated\ demand}$$

Since estimated demand is based on peak historical usage (Section 6.1), the *burstiness factor* of a FlowGroup is a measure of sum of peak usages of children divided by peak of sum of usages of the children. We multiply a FlowGroup's allocation by its *burstiness factor* before running the WaterFill. This allows its children to burst as long as they are not bursting together. If a FlowGroup's children burst at uncoordinated times, then the *burstiness factor* is high, otherwise the value will be close to 1.

6.3 Fair Allocation for Satisfied FlowGroups

A *Satisfied FlowGroup* is one whose demand is less than or equal to its allocation. Initially, we throttled each satisfied FlowGroup strictly to its estimated demand. However we found that latency sensitive applications could not ramp fast enough to their fair share on a congested link. We next eliminated throttling allocations for all satisfied FlowGroups. However, this lead to oscillations in system behavior as a FlowGroup switched between throttled and unthrottled each time its usage increased.

Our current approach is to assign satisfied FlowGroups a stable allocation that reflects the *fair share at infinite demand*. This allocation is a FlowGroup's allocation if its demand grew to infinity while demand for other FlowGroups remained the same. When a high priority satisfied FlowGroup's usage increases, it will ramp almost immediately to its *fair share*. Other low-priority FlowGroups will be throttled at the next iteration of the BwE control loop. This implies that the capacity of a constrained link is oversubscribed and can result in transient loss if a FlowGroup's usage suddenly increases.

The naive approach for implementing *user-fg* allocation involves running our global allocation algorithm multiple times for each FlowGroup, assigning infinite demand to the target *user-fg* without modifying the demand of other *user-fg*s. Because multiple such runs across all FlowGroups does not scale, we run one instance of the global algorithm and pass to Cluster Enforcers the *bandwidth function* for the most constrained link for each FlowGroup. Assuming the most constrained link does not change, the Cluster Enforcer can efficiently calculate allocation for a FlowGroup with ∞ demand in the constrained link, assuming it becomes the bottleneck.

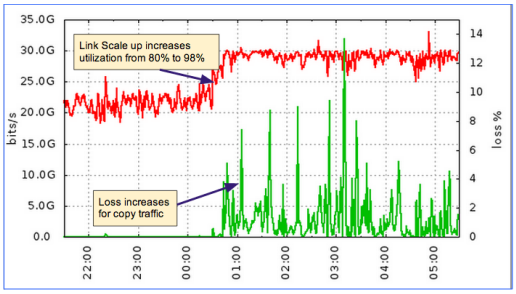


Figure 9: Improving Network Utilization

6.4 Improving Network Utilization

BwE allows network administrators to increase link utilization by deploying high throughput NETBLT [10]-like protocols for copy traffic. BwE is responsible for determining the flow transmission rate for these protocols. We mark packets for such copy flows with low priority DSCP values so that they absorb most of the transient network packet loss. To ensure that the system achieves high utilization (>90%) without affecting latency/loss sensitive flows such as web and video traffic, the BwE Global Enforcer supports two rounds of allocation.

- In the first round, link capacities are set conservatively (for example at 90% of actual capacity). All traffic types are allowed to participate in this round of allocation.
- In the second round, the Global Enforcer allocates only copy traffic, but it scales up the links aggressively, e.g., to 105% of link capacity.
- We also adjust link scaling factors depending on loss on the link. If a link shows loss for higher QoS classes, we reduce the scaling factor. This allows us to better achieve a balance between loss and utilization on a link.

Figure 9 shows link utilization increasing from 80% to 98% as we adjust the link capacity. The corresponding loss for copy traffic also increases to an average 2% loss with no increases in loss for loss-sensitive traffic.

6.5 Redundancy and Failure Handling

For scale and fault tolerance, we run multiple replicas at each level of the BwE hierarchy. There are N live and M cold standby Job Enforcers in each cluster. Hosts report all *task-fg*s belonging to the same *job-fg* to the same Job Enforcer, sharding different *job-fg*s across Job Enforcers by hashing <user name, job name, destination cluster, traffic_type>.

Cluster Enforcers run as master/hot standby pairs. Job Enforcers report all information to both. Both instances independently run the allocation algorithm and return bandwidth allocations to Job Enforcers. The Job Enforcers enforce the bandwidth allocations received from the master. If the master is unreachable Job Enforcers switch to the allocations received from the standby. We employ a similar redundancy approach between Global Enforcers and Cluster Enforcers.

Communication between BwE components is high priority and is not enforced. However, there can be edge scenarios where BwE components are unable to communicate with each other. Some examples are: BwE job failures (e.g. binaries go into a crash loop) causing hosts to stop receiving updated bandwidth allocations, network routing failures preventing Cluster Enforcers from receiving allocation from the Global Enforcers, or the network model becoming stale.

The general strategy for handling these failures is that we continue to use last known state (bandwidth allocations or capacity) for several minutes. For longer/sustained failures, in most cases we eliminate allocations and rely on QoS and TCP congestion management. For some traffic patterns such as copy-traffic we set a low static allocation. We have found this design pattern of defense by falling back to sub-optimal but still operable baseline systems invaluable to building robust network infrastructure.

7. EVALUATION

7.1 Micro-benchmarks on Test Jobs

We begin with some micro-benchmarks of the live BwE system to establish its baseline behavior. Figure 12(a) demonstrates BwE fairness across users running different number of TCP connections. Two users send traffic across a network configured to have 100Mbps of available capacity between the source/destination clusters. User1 has two connections and a weight of one. We vary the number of connections for User2 (shown on the x-axis) and its BwE assigned weight. The graph shows the throughput ratio is equivalent to the users weight ratio independent of the number of competing TCP flows.

Next we show how quickly BwE can enforce bandwidth allocations with and without the infinite demand feature (Section 6.3). In this scenario there are 2 users on a simulated 100 Mbps link. Initially, User1 has weight of 3 and User2 has weight of 1. At 120s, we change the weight of User2 to 12. In Figure 12(b), where the infinite demand feature is disabled, we observe that BwE converges at 580s. In Figure 12(c), where infinite demand feature is enabled, we observe it converges at 160s. This demonstrates BwE can enforce bandwidth allocations and converge in intervals of tens of seconds. This delay is reasonable for our production WAN network since large bandwidth consumers are primarily copy traffic.

7.2 System Scale

A significant challenge for BwE deployment is the system's sheer scale. Apart from organic growth to flows and network scale other reasons that affect system scale were supporting

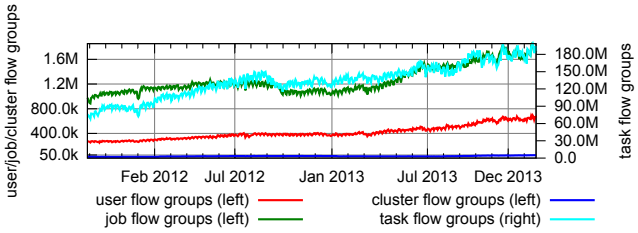


Figure 10: FlowGroup Counts

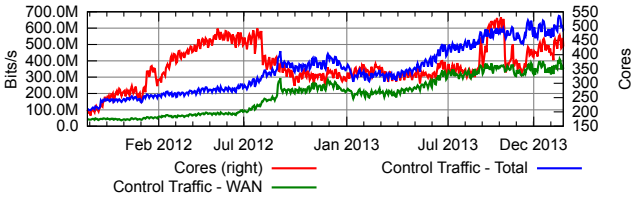


Figure 11: Resource overhead (Control System)

fine-grained bandwidth allocation and decreasing reporting interval for enforcement frequency.

Figure 10 shows growth in FlowGroups over time. As expected, as we go up in the hierarchy the number of FlowGroups drops significantly, allowing us to scale global components. Figure 11 shows the amount of resources used by our distributed deployment (excepting per-host overhead). It also shows the communication overhead of the control plane. We can conclude that the overall cost is very small relative to enforcing traffic generated by hundreds of thousands of cores using terabits/sec of bandwidth capacity.

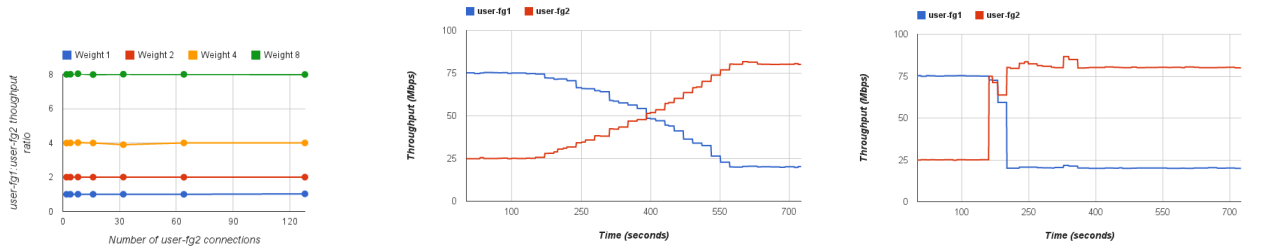
Table 2 shows the number of FlowGroups on a congested link at one point in time relative to all outgoing flow groups from a major cluster enforcer. It gives an idea of overall scale in terms of the number of competing entities. There are portions of the network with millions of competing FlowGroups. Table 3 shows our algorithm run time at various levels in the hierarchy. We show max and average (across multiple instances) for each level except global. Overall, our goal is to enforce large flows in a few minutes, which we are able to achieve. The table also shows that the frequency of collecting and distributing data is a major contributing factor to reaction time.

	site	cluster	user	job	task
One Congested Link	336	3.0k	40k	400k	12714k
Largest Cluster Enforcer	55	3.5k	63k	165k	15660k
Avg across cluster enforcers	53	1.5k	22k	60k	6496k
Global	1594	47.4k	682k	1825k	194088k

Table 2: Number of *-*fg*s (at various levels in BwE) for a congested link and for a large cluster enforcer.

	Algo Run-time		Algo	Reporting
	Max(s)	Mean(s)	Interval(s)	Interval(s)
Global Enforcer	3	-	10	10
Cluster Enforcer	.16	.15	4	10
Job Enforcer	<0.01	<0.01	4	5

Table 3: Algorithm run time and feedback cycle in seconds. Algorithm interval is how frequently algorithm is invoked and Reporting interval is the duration between two reports from the children in BwE hierarchy.



(a) BwE Fairness (b) Allocation Capped at Demand (c) Allocation not Capped at Demand

Figure 12: BwE compliance

	Min	Max	Mean
Number of *user-fg*	3%	11%	7%
Number of *job-fg*	3%	10%	6%
Usage	94%	99%	97%

Table 4: Percentage of FlowGroups enforced.

BwE tracks about 500k *user-fg*s and millions of *job-fg*s, but only enforces a small fraction of these. Processing for unenforced FlowGroups is lightweight at higher levels of the BwE hierarchy allowing the system to scale. Table 4 shows the fraction of enforced flows and the fraction of total usage they represent. BwE only enforces 10% of the flows but these flows account for 94% of the traffic. We also found that for congested links, those with more than 80% utilization, less than 1% of the utilization belonged to unenforced flows. This indicates BwE is able to focus its work on the subset of flows that most contribute to utilization and any congestion.

We introduced a number of system optimizations to address growth along the following dimensions: 1) Flow Groups: Organic growth and increase in specificity (for e.g., delegation). For example, the overall design has been serving us through a growth of 10x from BwE's inception (20M to 200M). 2) Paths: Traffic Engineering introduced new paths in the system. 3) Links: Organic network Growth 4) Reporting frequency: there is a balance between enforcement accuracy and resource overhead. 5) Bottleneck links: The number of bottleneck links affects the overall run time of the algorithm on the Global Enforcer.

8. DISCUSSION AND FUTURE WORK

BwE requires accurate network modeling since it is a key input to the allocation problem. This is challenging in an environment where devices fail often and new technologies are being introduced rapidly. In many cases, we lack standard APIs to expose network topology, routing and pathing information. With Software Defined Networking, we hope to see improvements in this area. Another challenge is that for scalability, we often combine a symmetric full mesh topology into a single abstract link. This assumption however breaks during network failures and handling these edge cases continues to be a challenge.

Our FlowGroup abstraction is limited in that it allows users sending from multiple source/destination cluster pairs over the same bottleneck link to have an advantage. We are exploring abstractions where we can provide fair share to all users across all bottleneck links irrespective of the number and placement of communicating cluster pairs. Other areas of research include improving the reaction time of the control system while scaling to a large number of FlowGroups, providing fairness at longer timescales (hours or days) and including flow deadlines in the objective function for fairness.

8.1 Broader Applicability

Our work is motivated by the observation that per-flow bandwidth allocation is no longer the ideal abstraction for emerging WAN use cases. We have seen substantial benefit of BwE to applications for WAN distributed computing and believe that it is also applicable to a number of emerging application classes in the broader Internet. For example, video streaming services for collaboration or entertainment increasingly dominate WAN communication. These applications have well-understood bandwidth requirements with step functions in additional utility from incremental bandwidth allocation. Consider that a 480P video stream may receive no incremental benefit from an additional 100Kbps of bandwidth; only sufficient additional bandwidth to enable 720P streaming is useful. Finally, homes and businesses are trending toward multiple simultaneous video streams with known relative priority and incremental bandwidth utility, all sharing a single bottleneck with known capacity.

Next, consider the move toward an Internet of Things [29] where hundreds of devices in a home or business may have varying wide-area communication requirements. These applications may range from home automation, to security, health monitoring, to backup. For instance, home security may have moderate bandwidth requirements but be of the highest priority. Remote backup may have substantial, sustained bandwidth requirements. However, the backup does not have a hard deadline and is largely insensitive to packet loss. Investigating BwE-based mechanisms for fair allocation based on an understanding of relative application utility in response to additional bandwidth is an interesting area of future work.

9. RELATED WORK

This paper focuses on allocating bandwidth among users in emerging multi-datacenter WAN environments. Given the generality of the problem, we necessarily build on a rich body

of related efforts, including utility functions [8], weighted fair sharing [2, 11, 12, 8] and host-based admission control [9]. We extend existing Utility max-min approaches [8] for multi-path routing and hierarchical fairness.

Weighted queuing is one common bandwidth allocation paradigm. While a good starting point, we find weights are insufficient for delivering user guarantees. Relative to weighted queuing, we focus on rate limiting based on demand estimation. BwE control is centralized and protocol agnostic, i.e., general to TCP and UDP. This is in contrast to DRL [25], which solves the problem via distributed rate control for TCP while not accounting for network capacity explicitly.

Netshare [20] and Seawall [30] also use weighted bandwidth allocation mechanisms. Seawall in particular achieves per-link proportional fairness. We have found max-min fairness to be more practical because it provides better isolation.

Gatekeeper [26] also employs host-based hierarchical token buckets to share bandwidth among data center tenants, emphasizing work-conserving allocation. Gatekeeper however assumes a simplified topology for every tenant. BwE considers complex topologies, multi-path forwarding, centralized bandwidth allocation, and a range of flexible bandwidth allocation mechanisms. Secondnet [15] provides pair-wise bandwidth guarantees but requires accurate user-provided demands and is not work conserving.

Oktopus [4] proposes a datacenter tree topology with specified edge capacity. While suitable for datacenters, it is not a natural fit for the WAN where user demands vary based on source-destination pairs. The BwE abstraction is more fine-grained, with associated implementation and configuration challenges. Oktopus also ties the problem of bandwidth allocation with VM placement. Our work however does not affect computation placement but rather takes the source of demand as fixed. We believe there are opportunities to apply such joint optimization to BwE. Datacenter bandwidth sharing efforts such as ElasticSwitch [24], FairCloud [23] and EyeQ [18] focus on a hose model for tenants. EyeQ uses ECN to detect congestion at the edge and assumes a congestion free core. In contrast, our flow-wise bandwidth sharing model allows aggregation across users and is explicitly focused on a congested core.

Recent efforts such as SWAN [16], and B4 [17] are closely related but focus on the network and routing infrastructure to effectively scale and utilize emerging WAN environments. In particular, they focus on employing Software Defined Networking constructs for controlling and efficiently scaling the network. Our work is complementary and focuses on enforcing policies given an existing multipath routing configuration. Jointly optimizing network routing, and bandwidth allocation is an area for future investigation. TEMPUS [19] focuses on optimizing network utilization by accounting for deadlines for long-lived flows . Our bandwidth sharing model applies to non-long-lived flows as well and does not require deadlines to be known ahead of time. Flow deadlines open up possibility of further optimization (for example, by smoothing bandwidth allocation over a longer period of time) and that remains an area for future work for us.

Work in RSVP, Differentiated Services and Traffic Engineering [14, 27, 7, 22, 21, 1, 5] overlaps in terms of goals. However, these approaches are network centric, assuming that host control is not possible. In some sense, we take the opposite approach, considering an orthogonal hierarchical control infrastructure that leverages host-based demand measurement and enforcement.

Congestion Manager [3] is an inspiration for our work on BwE, enabling a range of flexible bandwidth allocation policies to individual flows based on an understanding of application requirements. However, Congestion Manager still manages bandwidth at the granularity of individual hosts, whereas we focus on the infrastructure and algorithms for bandwidth allocation in a large-scale distributed computing WAN environment.

10. CONCLUSIONS

In this paper, we present Bandwidth Enforcer (BwE), our mechanism for WAN bandwidth allocation. BwE allocates bandwidth to competing applications based on flexible policy configured by *bandwidth functions*. BwE supports hierarchical bandwidth allocation and delegation among services while simultaneously accounting for multi-path WAN communication.

Based on multiple years of production experience, we summarize a number of important benefits to our WAN. First, BwE provides isolation among competing services, delivering plentiful capacity in the common case while maintaining required capacity under failure and maintenance scenarios. Second, we provide a single point for specifying allocation policy to administrators. While pathing, RTT, and capacity can shift substantially, BwE continues to allocate bandwidth according to policy. Finally, BwE enables the WAN to run at higher levels of utilization than before. By tightly integrating new loss-insensitive file transfer protocols running at low priority with BwE, we run many of our WAN links at 90% utilization.

Acknowledgements

Many teams within Google collaborated towards the success of the BwE project. We would like to acknowledge the BwE development, test and operations groups including Aaron Racine, Alex Docauer, Alex Perry, Anand Kanagala, Andrew McGregor, Angus Lees, Deepak Nulu, Dmitri Nikulin, Eric Yan, Jon Zolla, Kai Song, Kirill Mendelev, Mahesh Kallahalla, Matthew Class, Michael Frumkin, Michael O'Reilly, Ming Xu, Mukta Gupta, Nan Hua, Nikhil Panpalia, Phong Chuong, Rajiv Ranjan, Richard Alimi, Sankalp Singh, Subbaiah Venkata, Vijay Chandramohan, Xijie Zeng, Yuanbo Zhu, Aamer Mahmood, Ben Treynor, Bikash Koley and Urs Hölzle for their significant contributions to the project. We would also like to thank our shepherd, Srikanth Kandula, and the anonymous SIGCOMM reviewers for their useful feedback.

11. REFERENCES

[1] Wikipedia: Differentiated services. http://en.wikipedia.org/wiki/Differentiated_services.

[2] Allalouf, M., and Shavitt, Y. Centralized and Distributed Algorithms for Routing and Weighted Max-Min Fair Bandwidth Allocation. *IEEE/ACM Trans. Networking 16*, 5 (2008), 1015–1024.

[3] Balakrishnan, H., Rahul, H. S., and Seshan, S. An integrated congestion management architecture for internet hosts. In *In Proc. ACM SIGCOMM* (1999), pp. 175–187.

[4] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. Towards predictable datacenter networks. In *SIGCOMM* (2011).

[5] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. An Architecture for Differentiated Service. RFC 2475 (Informational), December 1998. Updated by RFC 3260.

[6] Boudec, J.-Y. Rate adaptation, congestion control and fairness: A tutorial, 2000.

[7] Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), September 1997. Updated by RFCs 2750, 3936, 4495.

[8] Cao, Z., and Zegura, E. W. Utility max-min: An application-oriented bandwidth allocation scheme. In *INFOCOM* (1999).

[9] Choi, B.-K., and Bettati, R. Endpoint admission control: network based approach. In *Distributed Computing Systems, 2001. 21st International Conference on.* (Apr 2001), pp. 227–235.

[10] Clark, D. D., Lambert, M. L., and Zhang, L. Netblt: A high throughput transport protocol. In *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology* (New York, NY, USA, 1988), SIGCOMM '87, ACM, pp. 353–359.

[11] Danna, E., Hassidim, A., Kaplan, H., Kumar, A., Mansour, Y., Raz, D., and Segalov, M. Upward Max Min Fairness. In *INFOCOM* (2012), pp. 837–845.

[12] Danna, E., Mandal, S., and Singh, A. A Practical Algorithm for Balancing the Max-min Fairness and Throughput Objectives in Traffic Engineering. In *Proc. INFOCOM* (March 2012), pp. 846–854.

[13] Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. *Commun. ACM 51*, 1 (January 2008), 107–113.

[14] Fortz, B., Rexford, J., and Thorup, M. Traffic Engineering with Traditional IP Routing Protocols. *IEEE Communications Magazine 40* (2002), 118–124.

[15] Guo, C., Lu, G., Wang, H. J., Yang, S., Kong, C., Sun, P., Wu, W., and Zhang, Y. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *CoNEXT* (2010).

[16] Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., and Wattenhofer, R. Have Your Network and Use It Fully Too: Achieving High Utilization in Inter-Datacenter WANs. In *Proc. SIGCOMM* (August 2013).

[17] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A. B4: Experience with a Globally-Deployed Software Defined WAN. In *Proceedings of the ACM SIGCOMM 2013* (2013), ACM, pp. 3–14.

[18] Jeyakumar, V., Alizadeh, M., Mazieres, D., Prabhakar, B., Kim, C., and Greenberg, A. Eyeq: Practical network performance isolation at the edge. In *Proc. of NSDI* (2013), USENIX Association, pp. 297–312.

[19] Kandula, S., Menache, I., Schwartz, R., and Babbula, S. R. Calendaring for wide area networks. In *Proc. SIGCOMM* (August 2014).

[20] Lam, T., Radhakrishnan, S., Vahdat, A., and Varghese, G. NetShare: Virtualizing data center networks across services. Tech. rep., 2010.

[21] Minei, I., and Lucek, J. *MPLS-Enabled Applications: Emerging Developments and New Technologies*. Wiley Series on Communications Networking & Distributed Systems. Wiley, 2008.

[22] Osborne, E., and Simha, A. *Traffic Engineering with Mpls (Paperback)*. Networking Technology Series. Cisco Press, 2002.

[23] Popa, L., Krishnamurthy, A., Ratnasamy, S., and Stoica, I. Faircloud: Sharing the network in cloud computing. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2011), HotNets-X, ACM, pp. 22:1–22:6.

[24] Popa, L., Yalagandula, P., Banerjee, S., Mogul, J. C., Turner, Y., and Santos, J. R. Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 351–362.

[25] Raghavan, B., Vishwanath, K., Ramabhadran, S., Yocum, K., and Snoeren, A. C. Cloud control with distributed rate limiting. In *In SIGCOMM* (2007).

[26] Rodrigues, H., Santos, J., Turner, Y., Soares, P., and Guedes, D. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *Workshop on I/O Virtualization* (2011).

[27] Roughan, M., Thorup, M., and Zhang, Y. Traffic Engineering with Estimated Traffic Matrices. In *Proc. IMC* (2003), pp. 248–258.

[28] Saltzer, J. H., Reed, D. P., and Clark, D. D. End-to-end arguments in system design. *ACM Trans. Comput. Syst. 2*, 4 (November 1984), 277–288.

[29] Sarma, S., Brock, D. L., and Ashton, K. The networked physical world–proposals for engineering the next generation of computing, commerce & automatic identification. *White Paper, Auto-ID Center, MIT. Designed b y Foxner. www. foxner. com* (2000).

[30] Shieh, A., Kandula, S., Greenberg, A., Kim, C., and Saha, B. Sharing the data center network. In *NSDI* (2011).