Optimizing Network Provisioning through Cooperation

Harsha Sharma, Parth Thakkar, Sagar Bharadwaj, Ranjita Bhagwan, Venkata N. Padmanabhan, Yogesh Bansal, Vijay Kumar, and Kathleen Voelbel, *Microsoft*

https://www.usenix.org/conference/nsdi22/presentation/sharma

This paper is included in the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation.

April 4–6, 2022 • Renton, WA, USA

978-1-939133-27-4

Optimizing Network Provisioning through Cooperation

Harsha Sharma,[*] *Parth Thakkar,*[*] *Sagar Bharadwaj,*[*] *Ranjita Bhagwan, Venkata N. Padmanabhan,*
Yogesh Bansal, Vijay Kumar, Kathleen Voelbel
Microsoft

Abstract

The rise of cloud-scale services has fueled a huge growth in inter-data center (DC) Wide-Area Network (WAN) traffic. As a result, cloud providers provision large amounts of WAN bandwidth at very high costs. However, the inter-DC traffic is often dominated by *first-party applications*, i.e., applications that are owned and operated by the same entity as the cloud provider. This creates a unique opportunity for the applications and the network to cooperate to optimize network provisioning (which we term as optimizing the *"provisioning plane"*), since the demands placed by dominant first-party applications often *define* the network. Such optimization is distinct from and goes beyond past work focused on the control and data planes (e.g., traffic engineering), in that it helps optimize the provisioning of network capacity and consequently helps reduce cost.

In this paper, we show how cooperation between application and network can optimize network capacity based on knowledge of the application's deadline coupled with network link failure statistics. Using data from a tier-1 cloud provider and a large enterprise collaboration service, we show that our techniques can potentially help provision significantly lower network capacity, with savings ranging from 30% to 45%.

1 Introduction

The growth of cloud-scale mega services [1] has fueled a huge increase in network traffic, not only within data centers (DCs) and on the Internet but, importantly, also on the inter-DC wide-area network (WAN). We observe that the inter-DC WAN traffic for large public cloud providers is dominated by first-party applications and services e.g., the e-commerce service in the case of Amazon, Google search and Gmail in the case of Google, and Bing search and Office 365 in the case of Microsoft [8, 9]. This presents both a challenge and an opportunity.

[*]Equal contribution

[1]We use "services" and "applications" interchangeably.

The challenge is that, as application usage grows, so does the inter-DC WAN traffic. This leads to increased provisioning of inter-DC WAN capacity, often with a multiplicative factor to provide redundancy. Note that large cloud providers typically build and operate their own inter-DC WAN with owned and leased fiber, which is different from a small provider who might use an ISP for such connectivity. The increased availability of bandwidth, and the fact that it is sunk cost (i.e., already paid for and so might as well be utilized), fuels the appetite of existing and new applications. They consume more bandwidth (though there might be back-pressure due to mechanisms such as traffic engineering [7, 8] or internal pricing), causing the network to forecast higher usage for the future, which in turn leads to increased network provisioning and so forth. Such a vicious cycle is quite expensive since inter-DC WAN bandwidth imposes an amortized annual cost of 100s of millions of dollars on a large cloud provider [7].

This situation is quite different from the third-party application setting, wherein market forces of cost and price operating between the consumer of bandwidth (third-party application) and the provider of bandwidth (cloud provider) would tend to keep the process in check. If the provider has excess capacity, they could find new customers to sell it to rather than encouraging existing customers to be profligate in their use of bandwidth. There is an opportunity to do better in the first-party case by leveraging *cooperation* across the network and the applications. Specifically, rich information flow between the application and the network would enable informed network provisioning. We term the task of provisioning network capacity as optimizing the "provisioning plane", to set it apart from the well-established notions of the control and data planes.

We leverage the knowledge of the *application's deadline* to optimize network provisioning. Such an application deadline is quite distinct from the more common notion of delay tolerance, which centers on the latency that the application can tolerate in network communication. Application deadline, on the other hand, arises from the application's ability to pause, or defer, some or all of its activities and the associated communication for an extended period of time, for example, when there is a loss of network capacity because of link failures. If the network capacity is likely to be restored (e.g., by repairing the failed link(s)) within the period of the application deadline, then we can dispense with the provisioning of redundant network capacity for "deferrable" traffic.

If user impact is to be avoided, the deferrable application activity, and the resulting traffic, should be in the background. A good example is load balancing, which might be triggered when a server is at the risk of running "hot" on one or more resources such as CPU, IO (I/O operations) capacity, or storage space, and therefore needs to shed load. Load balancing would typically involve the transfer of a large volume of data (e.g., user mailboxes in the context of an email application). This activity could be paused for a length of time so long as there is enough "headroom" in resources, i.e., none is on the verge of being saturated. Based on past data, we can make a conservative but specific choice of deadline to facilitate network provisioning.

We develop a generic framework for applications to express their demands in a way that reflects the traffic volume, deadlines, and desired probability of satisfaction (i.e., the likelihood that the demand will be met). We also develop a model for link failures and repairs that is informed by historical data. A key aspect of this model is the data-driven discovery of links with correlated failures, say because they share one or more components (e.g., fiber conduit, power supply, etc.) and the recreation of such correlated failures to simulate scenarios that are particularly challenging from the viewpoint of capacity provisioning.

We evaluate the above using data from the WAN of a tier-1 cloud provider, Microsoft, and from a large enterprise collaboration service, M365 Substrate (which we shall hereafter refer to as Substrate), which uses this WAN. We find that application-informed provisioning reduces network capacity by more than 30% in multiple regions.

Note: While we use data from commercial services — Microsoft and Substrate— to drive our analyses here, this data is highly sensitive due to commercial reasons. Therefore, we are not in a position to report metrics such as the traffic volume, network capacity, failure characteristics, etc. in an absolute sense and report relative numbers instead.

1.1 Comparison with Traffic Engineering

Before proceeding, it is useful to compare our work on cooperative provisioning with the well-established prior work on traffic engineering.

Traffic engineering focuses on supporting a specified demand (i.e., source-destination flows, quality of service requirements) on a given network (i.e., the network topology, including link capacities). Both the demand matrix and the network topology are taken as input and traffic engineering looks for ways of supporting the demand at *run time* through techniques such as routing and path selection [8], smoothing to shift traffic peaks into the valleys [7], and using store-and-forward techniques to deal with temporal offsets between traffic peaks in different parts of the network [13]. There is a body of traffic engineering work that specifically considers the problem of satisfying deadlines in the face of new demands, link failures, etc., by employing admission control, online scheduling, and fairness across demands when the deadlines cannot be satisfied [11, 22].

In comparison, our work on cooperative provisioning focuses on the *planning phase* that precedes the creation of the network or the augmentation of its capacity. This requires modeling link failures upfront by simulating the failures of (combinations of) links and making sure that sufficient capacity is provisioned to accommodate the demands even in the face of such failures. In contrast, traffic engineering deals with link failures as these arise and does not entail provisioning capacity. Furthermore, in optimizing the provisioning of network capacity, the ability to defer traffic by pausing workload, for days or weeks, provides us a qualitatively greater flexibility for optimization than the QoS requirements that are typically dealt with in traffic engineering [11, 22]. For instance, the former would allow tiding over link failures without redundant provisioning, while the latter typically would not.

The opportunity to optimize provisioning by counting on the ability to pause certain demands arises because of the first-party setting, which enables cooperation between the network and the applications. Therefore, cooperative provisioning is limited to settings where such cooperation is feasible. Traffic engineering, on the other hand, is applied more broadly (e.g., in ISP networks) and as such cannot assume such cooperation.

2 Application Traffic Demands

The starting point for network capacity provisioning is the demand placed by applications. Since building a network or augmenting the capacity of an existing network would typically take time (several months to more than a year), there is also the need to *forecast* future demand. Forecasting is a well-studied problem and there exist many techniques for it [15, 19]. Thus, in this paper, we do not focus on forecasting and our analysis is based on taking a snapshot of the present demand and using it to perform capacity planning, with and without our optimizations. However we do show in Section 4.3 that our findings carry over even if applied to future demand obtained through forecasting.

2.1 Demand Specification

An application, i, specifies its demands as a set of discrete time series, with the j^{th} demand on behalf of i being specified as $D_{ij} = (t, A, B, V, d, p)$, i.e., a demand for conveying a volume V of traffic from location A to location B, expressed at time

t and with a deadline of d (that is, the demand should be satisfied during $[t, t+d]$), and with the desire that the demand be satisfied with a probability of p, i.e., in the network failure scenarios that cumulatively account for a fraction p of time. (Each of these quantities is set by the application i and should carry the subscript ij, but we drop it for the ease of exposition.) We describe two types of application demands which are relevant to this paper: immediate and deferrable.

Immediate: Such traffic tends to be in the critical path of user latency, so there is little temporal flexibility. The deadline for such a demand is "now", so the demand can be expressed as a data rate to be supported between a specified source and destination. Hence we can simplify the formulation of the demand as $D_i = (t, A, B, r, p)$, where we subsume V and d by the rate $r = V/d$. Note that if the network were provisioned to support the desired rate r for this demand continually, we would be able to transfer volume V within a deadline d.

Deferrable: The traffic demand arising from asynchronous activity tends to be temporally flexible, i.e., deferrable, with a long deadline. A good example is traffic arising from distributed load balancing, wherein resources on a server (e.g., CPU, IO capacity, storage space) grow in terms of utilization (i.e., start becoming "hot"), necessitating the rebalancing of the workload and entailing the transfer of related data (e.g., a user's mailbox in the context of an email service or folder in the case of a storage service). The urgency of the load balancing activity and hence the deadline of the resulting traffic demand would depend on how much headroom there is on the server resources that are heating up.

Deferrable demand would be specified with a deadline d that reflects the degree of temporal flexibility. For instance, in the case of load balancing triggered workload, there might be enough headroom in the server resources (i.e., none is close to being saturated) to allow d to be set to days or even weeks. Note that the demand can tolerate such latency (e.g., by slowing, pausing or turning off the application components responsible for the demand); however, once the application component is resumed and it actually starts its data transfers, these would be completed in a much shorter and typical "network timescale" (e.g., within seconds or minutes or hours, depending on the size of the transfer). Also, just because a demand can be temporarily paused, it does not mean that it can be forgotten. So, the network would be provisioned with sufficient capacity to allow "catch-up" on the deferred demand once it is resumed. Processes that perform periodic tasks on data such as garbage collection, compliance checks, and workload analytics can also cause asynchronous traffic demands.

3 Cooperative Provisioning

In this section, we describe how the network, with the application's cooperation, can provision the network efficiently. We first provide some background on network provisioning and how it is currently done. Next, we discuss two specific opportunities for cooperative provisioning: (a) deriving a smoothed demand signal based on explicit application input, and (b) provisioning network redundancy in a way that is cognizant of the demand deadlines and the repair time of links. Finally, we describe a mathematical framework which, using constraint optimization, ensures appropriate provisioning of network capacity, including redundant capacity, to satisfy all demands. We dub this framework Approv, short for "Application-informed Provisioning".

3.1 Background on Capacity Provisioning

We sketch how capacity provisioning is done by the cloud provider, Microsoft. Although we do not have information from other providers, we believe that the process outlined here is general and not provider-specific.

Periodically, e.g., every few months, the cloud operator forecasts demands between each datacenter (DC) pair and subsequently provisions network capacity to satisfy all demand forecasts. Since the demands arise from third-party applications (which the operator has no real leverage over) as well as first-party ones, the network typically works with just the actual traffic originated by applications (i.e., an implicit signal) rather than an explicit expression of application traffic demand (e.g., as in Section 2.1). Such an implicit signal is derived from application traffic categorized into tiers of service, with each tier corresponding to a different priority level [7, 8].

Based on this implicit demand signal derived from actual traffic, the operator derives the peak or 95%ile (P95) traffic level and uses this to forecast the traffic level, say months or even more than a year into the future. In the absence of any additional context from the applications sourcing the traffic, the operator has no choice but to work with the implicit demand signal, no matter how spiky it is (i.e., how high the peaks are).

With the demand so determined, the operator proceeds to simulate the failure of various links and combinations of links. A simulator is used to route each demand over possibly multiple paths chosen based on the latency and the available bandwidth. If the demands cannot be satisfied using the available bandwidth, the operator would augment capacity in the network (i.e., add redundant capacity) so that the demands are satisfied even in the face of such failures. Specifically, the capacity of a link is augmented if its utilization during simulation rises above a high-water mark and it is reduced if the utilization drops below a low-water mark. At the end of this iterative process of simulation, the operator would arrive at the *network capacity build out plan*, specifying the number and capacities of the links needed.

There are a couple of details worth noting. First, in general, the operator would start with an existing network and then determine the capacity augmentation and link decommissioning needed based on the demand forecast. However, to enable fair comparisons in this paper, we provision the network from scratch, i.e., starting with links of zero capacity, which helps avoid the need to carry the baggage of past provisioning done without the benefit of Approv. Second, for simplicity, we work with the specified topology (where each link starts out with zero capacity) and only compute the capacity needed on the links that are part of the topology. While the Approv framework is general enough to accommodate the creation of new links, the ability to create a link between two locations would, in general, be constrained by practical considerations that would have to be provided as additional input.

3.2 Leveraging Application Cooperation

We can leverage the application's cooperation — specifically that of first-party applications that have nothing to hide from the network — to improve the network capacity provisioning process outlined above in two ways.

3.2.1 Explicit Demand Signal to Aid Smoothing

First, rather than just work with the implicit signal of actual traffic sent, the operator can take advantage of application demands expressed explicitly per the framework presented in Section 2.1. Such explicit knowledge would enable the operator to calculate the smoothed demand.

For example, even if the peak (or P95) rate of traffic sent by an application was 1Gbps, knowledge of the explicit application demand (including the deadline d from Section 2.1) might allow the operator to determine that the application's traffic could have been smoothed down to a peak rate of 0.8 Gbps. Therefore, instead of basing its forecast, and consequently the actual capacity provisioning computation, on the 1Gbps peak, the operator could work with the smoothed 0.8 Gbps peak, thereby "rightsizing" the capacity. In the case of a deferrable demand, expressing the demand in terms of the volume V and deadline d would enable further rightsizing of capacity compared to just smoothing down the demand expressed as a rate r. The reason is that much or all of a demand in terms of V and d could be fit within the valleys and headroom of the immediate demands (Figure 2), obviating the need for any additional provisioning.

3.2.2 Rightsizing Redundancy for Deferrable Demands

Second, knowledge of the deadlines enables the operator to "rightsize" the redundancy too. The intuition is that if an application demand has a longer deadline than the time to recover a failed link, then during capacity provisioning (and during the subsequent operation of the provisioned network), we can assume that part or all of such demands are simply paused or deferred until the failed link has been repaired. This would avoid the need for provisioning full redundancy for such deferrable demands.

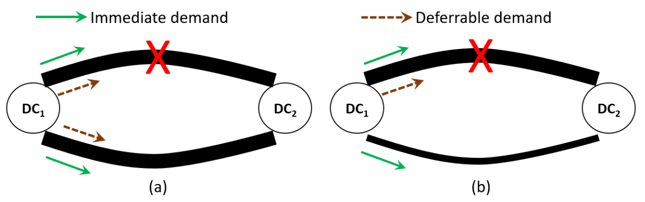


Figure 1: Rightsizing of redundancy provisioning: (a) shows the default provisioning of redundancy with no distinction between immediate and deferrable demands. Upon failure of the primary link at the top, the backup link at the bottom has the capacity to carry the full load of traffic). (b) shows how the provisioning of redundancy reduces by confining it to the immediate demand. The thinner backup link is sized to carry just immediate demand, although the overall provisioning would be sized to accommodate the "catch-up" of the deferred demand upon repair.

A simple example illustrates the savings made possible by such informed redundancy provisioning. Consider a network comprising just two nodes, A and B. Consider two cases:

Case I: The application's implicit demand is 1 GB/day.

Case II: The application explicitly specifies a demand of $V = 30$ GB with a deadline of $d = 30$ days, which also works out to 1 GB/day.

Say the cloud operator knows that the $A-B$ link could be down for up to 10% of the time, i.e., for up to 3 days in the month. Therefore, to support the implicit demand in Case I, the operator would have to provision at least one additional link of 1 GB/day capacity (and possibly more depending on the likelihood of concurrent failures of multiple links). Therefore, the operator would have to provision a total of at least $1+1=2$ GB/day capacity, and possibly more, on the $A-B$ path.

On the other hand, in Case II, explicit knowledge of the deadline would enable the operator to determine that the demand could be paused, or deferred, temporarily to "tide over" the network link's downtime. Of course, once the link has been restored, there would be the need to "catch up" on the deferred demand. Still, given the up to 10% downtime of the link, the operator can get away with provisioning a single $A-B$ link of capacity 1.11 GB/day (computed as 1/(1-0.1)), which would be significantly lower than the (at least) 2 GB/day in Case I above. (In this illustrative example, we ignore the quantization of bandwidth, i.e., the minimum step size for allocation.)

3.2.3 Summary

In general, we would have a mix of "immediate" user-facing demand and "deferrable" background demand, and the former cannot tolerate any delay. Therefore, we might still have to include redundant links. However, by rightsizing the provisioning of redundancy for the deferrable demand, we would be able to reduce the overall capacity provisioned, as illustrated in the simple example in Figure 1. Furthermore, as explained above, specifying deferrable demand in terms of volume and deadline would enable more effective provisioning (perhaps even fitting within the valleys and headroom of the immediate demand) than working with a smoothed rate.

In the remainder of this section, we formalize our framework for cooperative capacity provisioning and also present our method for simulating network link failures and repairs (including the concurrent failure of links that carry shared risks), informed by the history of actual link failures and repairs.

3.3 Framework for Capacity Provisioning

Our Approv framework models network provisioning as a constraint optimization problem using a linear program (LP). As with any network provisioning approach (Section 3.1 provided some background on this), the LP needs to simulate all "likely" link failures, singly or in combination, and ensure that the network has sufficient redundant capacity to fulfill all demands despite such link failures.

A key challenge in our work arises from our richer demand model compared to prior work. State-of-the-art approaches [1] take demand data rates as input (e.g., 1 Gbps from A to B) and then only simulate link failures as point-in-time events, to verify the satisfaction of demand in the face of failures. Since we support demands that specify deadlines (d) that could stretch to days or even weeks, we incorporate a much richer notion of failures, including the actual duration, i.e., the time from the loss of capacity due to the onset of a failure episode until the restoration of capacity upon repair. As we discuss, this approach enables optimizations that are not possible with point-in-time simulation of failures.

Given a time-window (t_s, t_e), We define a *failure scenario* as a set of per-link time-series $\{f_1, f_2, \ldots, f_n\}$, where n is the number of links in the network. f_l is a time series representing the status of link l, i.e., whether it was up (working) or down (failed), over time. f_l is a time series, $f_{l1}, f_{l2}, ..., f_{lt}$ over the full duration of the simulation, where $f_{lt} \in \{0(down), 1(up)\}$ and time is discretized (in steps of 1 hour in our work). To synthesize realistic failure scenarios that extend over the full duration of the simulation, we have built a novel history-based failure model, described in Section 3.4.

The Approv capacity provisioning formulation assumes a fixed network topology, where the nodes are either datacenters in the cloud network or network points of presence. Given such a topology and the application demands the Approv provisioning framework allocates capacity to each link in the topology so as to satisfy all demands, by simulating two functions:

- *Topology and Routing:* Approv incorporates constraints arising from the network topology and the set of valid network-level routes between any two datacenters (DCs). The routes are derived from the large inter-DC WAN deployed by Microsoft.
- *Link Failures:* Approv also incorporates constraints to capture the effect of each failure scenario that is simulated. This helps ensure that the network is provisioned with enough capacity to fulfill all demands even in the face of such failures.

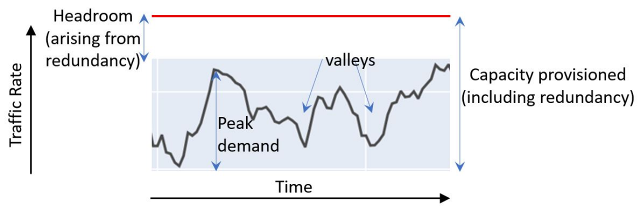


Figure 2: Illustration of the peaks and valleys in the time series of the immediate demand, and of the headroom above the peak, arising from the overall capacity provisioned, inclusive of the redundancy. The valleys and the headroom represent capacity that could be used to accommodate background demand, subject to its deadline.

For ease of exposition, we use a uniform p (i.e., probability of satisfaction) for all demands. Later, in Section 3.3.2, we relax this assumption to generalize the framework to incorporate a demand-specific p.

Given a set of failure scenarios $\mathbf{F}$, we construct a "capacity provisioning LP" with constraints corresponding to each combination of demand and failure scenario. The LP generates a capacity provisioning plan in two steps:

1. *Provisioning of immediate demands:* Such demands have an immediate deadline, so we represent these in terms of the rate r, as discussed in Section 2.1. The demanded rate needs to be supported even in the face of the failure scenario considered. To ensure this, we sweep across time, from the start to the end of the simulation. We ensure that the network is sufficiently provisioned to support the immediate demands at each point in time, which corresponds to a specific combination of link failures.
2. *Provisioning of deferrable demands:* Deferrable demands are expressed in terms of the desired volume, V, of data to be transferred and the corresponding deadline d. Our provisioning framework first checks to see whether (and if so, how much of) such demands could be accommodated in the valleys and the headroom of the capacity provisioned above for the immediate demands (see Figure 2). To the extent the deferrable demand exceeds what can be so accommodated, the framework augments the capacity on one or more links to ensure that the deferrable demands are satisfied too (in addition to the immediate demands). In doing so, we take advantage of the demand smoothing and redundancy rightsizing techniques discussed in Section 3.2.

At the end, we arrive at the network build plan, which gives the capacity to be provisioned on each link in the network. We present the LP formulation and an example in Appendix A.1 and A.2.

3.3.1 Accommodating Probability of Satisfaction (p)

To ensure that the capacity provisioned in the network build plan satisfies the demand with the desired probability p, we use the failure model (discussed next in Section 3.4) to generate a large number of failure scenarios. As discussed in Section 3.4, the generation of failure scenarios is governed by the failure and repair history of each link and group of links. Consequently, the more likely link failure combinations will appear more often in the failure scenarios, just as we would desire. Since we would like the network to be provisioned so as to satisfy the demand with probability p, or equivalently in at least a fraction p of the failure scenarios considered, we sort the $|\mathbf{F}|$ individual failure scenarios in increasing order of the capacity of the build plan generated when each such scenario alone is simulated. To enable the sorting, we consider the impact of each failure scenario on the build plan capacity in isolation instead of the collective impact of a set of failure scenarios, as in the capacity provisioning LP discussed in Section 3.3. We then consider just the first p fraction of the failure scenarios (i.e., the $p.|\mathbf{F}|$ scenarios with the least impact on capacity) and disregard (or cut off) the last $1-p$ fraction of scenarios (i.e., the ones with the greatest impact on capacity). This subset of failure scenarios is then provided as input to the capacity provisioning LP to generate the build plan.

3.3.2 Accommodating Demand-Specific p

In general, each demand D_i could have its own p_i, representing the desired probability of satisfaction for that demand. To accommodate such demand-specific levels of p, we employ an iterative process. We first sort the p_i in increasing order, i.e., going from the least level of assurance sought by a demand to the highest. For ease of exposition, we assume that the sorted order is $p_1, p_2, ..., p_n$. Then, we proceed as follows, where in each subsequent step, the additional capacity provisioned, if any, is over and above that which was already provisioned in the preceding steps. In other words, the capacity provisioned never decreases as we progress through the steps:

1. First, present all demands $(D_1, D_2, ..., D_n)$ to Approv and sort the failure scenarios in increasing order of their *individual* impact on capacity (as outlined at the end of Section 3.3 above). Then, pick p_1 as the cutoff in this sorted list of failure scenarios (i.e., focus on just the first p_1 fraction of the failure scenarios) and run Approv on these, to arrive at the *cumulative* capacity. This would ensure that the network is provisioned to satisfy all demands with probability (at least) p_1.

2. Then, exclude the first demand, D_1, and present the rest $(D_2, ..., D_n)$ to Approv and sort the *remaining* failure scenarios (i.e., excluding the ones already satisfied in step 1 above) in increasing order of capacity impact. Then, pick p_2 as the cutoff in this sorted list of failure scenarios, and proceed as in step 1 above. This would ensure that demands $(D_2, ..., D_n)$ are satisfied with probability (at least) p_2. Note that the provisioning performed in step 1 has already ensured that all demands $(D_1, D_2, ..., D_n)$ are satisfied *concurrently* in the face of the p1 fraction of failure scenarios. It is only for the additional p2 - p1 fraction of failure scenarios (during which D1 need not be satisfied) that some of the capacity provisioned to satisfy D1 in step 1 could potentially be used to satisfy $(D_2, ..., D_n)$. Therefore, this subsequent step (step 2) does not *impinge* on the provisioning done in the earlier step (step 1).

3. Proceed accordingly, progressively raising the bar on the probability of satisfaction while narrowing down the corresponding set of demands considered at each step.

4. The process would conclude when, in the last step, we only consider demand D_n and ensure its satisfaction with probability p_n.

This ensures that the provisioning at the end of the process would satisfy all demands D_i with the corresponding probability p_i.

3.4 Failure Modeling

In this section, we describe our *history-based* failure model that we use to generate realistic failure scenarios to drive the provisioning framework described in Section 3.3. Using a history of link failure characteristics, we build a generative model of link failures that captures characteristics such as the distribution of Time To Recovery (TTR) and the Time Between Failures (TBF) for the individual links, and the correlation of failure and recovery across links.

Dataset: The dataset we use to build the failure model contains detailed link failure data at hourly granularity collected over a period of 13 months for the over 500 links in Microsoft inter-DC WAN. The WAN consists of 17 interconnected regions, such as Asia-Pacific (APAC) and North America (NAM), with each region including multiple datacenters. Each link l in the network is characterized by a discrete *link-status vector* time-series f_l, where f_{lt} is 1 if the link l was up at time t and 0 if the link was in a failed state at time t.

We build a separate model for each region to keep the modeling tractable, since there could be correlations — accidental or otherwise — across arbitrary pairs of links. Accordingly, our failure modeling uses two main steps, *Link Clustering and Characterization* and *Failure Scenario Generation*. We now describe each step.

Cluster no.	Correlation	% links	Cluser no.	Correlation	% links
0	0.81	46.3	1	0.75	7.3
2	0.79	7.3	3	0.79	7.3
4	0.80	4.9	5	0.92	7.3
6	1.0	2.4	7	1.0	2.4
8	1.0	2.4	9	1.0	2.4
10	1.0	4.9	11	0.97	4.9

Table 1: Avg intracluster Pearson correlation coefficient

3.4.1 Link Clustering and Characterization

Link failures are often correlated since links could share components such as a power source, a router, or a fiber cable [16, 20]. To capture such correlations, we cluster links that display similar failure patterns in our data. We run the Complete-linkage agglomerative clustering algorithm [3] using the *1-Pearson correlation coefficient* between the link-status vectors noted above as the distance metric between two links to form such clusters of links. Links which fail at the same time will have correlated failure patterns and hence land in the same cluster (In Section 3.4.2, we explain how we simulate failure of all links in a cluster simultaneously). We evaluated the average of Pearson correlation coefficient between links within each cluster, while sweeping over the count of clusters, and determined that setting the number of clusters to 12 yielded a satisfactory division of links, with the average intra-cluster Pearson coefficient being 0.9.

Table 1 shows average intra-cluster Pearson correlation coefficient for the Asia-Pacific (APAC) region. The high intra-cluster Pearson correlation coefficient underscores the high degree of correlation in the failure pattern of links within the clusters. Note that cluster 0, containing over 46% of the links, comprises links whose failure pattern does not correlate with that of any other link; in fact, these links suffer few failures and so their link status vectors are set to 1 at almost all times. So, this is not considered as a failure cluster during the generation phase (Section 3.4.2).

Since the clusters capture links with similar link-status vectors, we assume that links belonging to a cluster have the same distribution of Time To Repair (TTR) and Time Between Failures (TBF). Accordingly, we estimate a single distribution of TTR and TBF for the links in each cluster. From the link-status vectors in a given cluster, we gather all the link TTRs and estimate the Cumulative Distribution Function (CDF) of TTR by using linear interpolation. We do likewise to estimate the CDF of the TBF. We then use these CDFs to generate plausible failure scenario, i.e., a set of realistic link-status vectors for each link.

3.4.2 Failure Scenario Generation

Our failure scenario generation algorithm uses the estimated CDFs to generate a common link-status vector for each link cluster for a given time-window. This approach assumes that links within a cluster are perfectly correlated: all links fail and are restored at exactly the same time. While simplifying

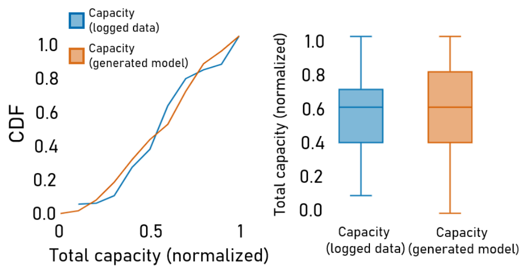


Figure 3: Comparing distribution of total capacity between logged historic data and generated failure scenarios.

the task of generating failure scenarios, this assumption of perfect correlation also ensures that the network provisioned in Section 3.3 is "stress-tested", i.e., made tolerant to a worst-case scenario where a chunk of capacity on correlated links is lost in one fell swoop.

Algorithm 1 Algorithm to generate timeseries of failures.

```
$uf \leftarrow$ uptime fraction
$ttr\_cdf \leftarrow$ CDF of TTRs from past data
$tbf\_cdf \leftarrow$ CDF of TBFs from past data
$timesteps \leftarrow$ no. of timesteps for which we generate timeseries

$timeseries = \emptyset$
Add 1 or 0 to $timeseries$ with probability $uf$ or $1 - uf$ respectively
$t = 1$
while $t \leq timesteps$ do
    if last item in $timeseries = 1$ then
        $tbf$ = Sample a value from $tbf\_cdf$
        Add $tbf$ 1s to $timeseries$ followed by a 0
    else
        $ttr$ = Sample a value from $ttr\_cdf$
        Add $ttr$ 0s to $timeseries$ followed by a 1
    $t$ =Length of $timeseries$
Output: First $timesteps$ elements of $timeseries$
```

We use Algorithm 1 to generate a common link-status vector per-cluster. A value of 1 at *timeseries[t]* denotes that all the links in the cluster are up at timestep t. Similarly, a value of 0 denotes that all the links in the cluster are in the failed state at timestep t. If the latest state of the generated timeseries is 1, we determine how much longer the links in the cluster will stay up by sampling a value, *tbf*, from the TBF distribution. Similarly, if the latest state is 0, we sample *ttr* from the estimated TTR distribution and fail all links in the cluster for the next *ttr* timesteps.

3.4.3 Properties of generated timeseries

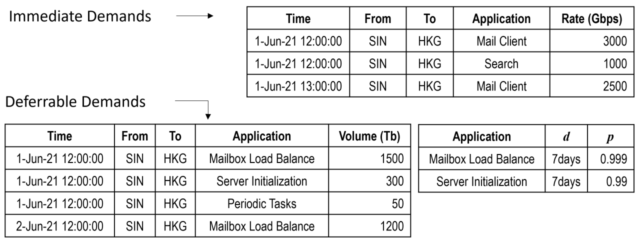
We inspected some properties of the generated link failure timeseries and compare it with those of the actual failures recorded in the history.

Total network capacity available in a region at a given time is the sum of capacities of links that are up at that time. Therefore the total network capacity varies with time. Figure 3 shows the CDF and Box plots of the distribution of total network capacity in a region and compares the distributions that we see in the past data and the generated failure scenarios. 10 failure scenarios each spanning a month were generated from the past 13 months of logged data. The capacities are min-max normalized using the same minimum and maximum normalization factors for both generated and logged capacities. The generated timeseries has a distribution of total capacity that is reasonably close to the distribution we see in the past data, but does not exactly replicate it. In particular, the minimum generated capacity is lower than the minimum logged capacity. This is because, as noted above, when failure scenarios are generated, all links in a cluster are failed together, thereby generating worst case failure scenarios that may not have occurred in the past. This ensures that enough capacity is provisioned for scenarios not present in our logs but are nevertheless possibilities because of failure correlations and so would be prudent to be prepared for.

3.5 Provisioning and Control Interfaces

In this section, we describe the APIs required in the provisioning and control plane between the applications and the network to realize the gains from cooperative provisioning.

Every few months, each application component provides its immediate and deferrable demands between every DC pair as a time-series to the network. Figure 4 provides an example of how applications specify their demands to the network provisioning process. Immediate demands specify an hourly rate r while deferrable demands specify a volume V of traffic that is generated within each day. Additionally, for each application, the deferrable demand includes a deadline d and a probability of satisfaction p.


Immediate Demands
Deferrable Demands

Time	From	To	Application	Rate (Gbps)
1-Jun-21 12:00:00	SIN	HKG	Mail Client	3000
1-Jun-21 12:00:00	SIN	HKG	Search	1000
1-Jun-21 13:00:00	SIN	HKG	Mail Client	2500

Time	From	To	Application	Volume (Tb)
1-Jun-21 12:00:00	SIN	HKG	Mailbox Load Balance	1500
1-Jun-21 12:00:00	SIN	HKG	Server Initialization	300
1-Jun-21 12:00:00	SIN	HKG	Periodic Tasks	50
2-Jun-21 12:00:00	SIN	HKG	Mailbox Load Balance	1200

Application	d	p
Mailbox Load Balance	7days	0.999
Server Initialization	7days	0.99

Figure 4: The specification of immediate demands using rate and deferrable demands using volume.

Choosing deadlines: In some cases, component owners can systematically calculate deadlines, while in others, it is left to domain experts to use their judgement and specify the deadline. Section 4.1 provides examples from the Substrate service that fall into both categories. To gain more insight into the choice of deadlines, we interviewed the service owners, who are the domain experts. Since the service owners are going from operating in a mode without any deferrable traffic to one where they are willing to defer part of their traffic to enable cost savings, it is understandable that they were conservative in picking appropriate deadline values. Also, in some cases, they prefer the component to have a minimum amount of capacity available at all times. To support this, we simply divide the component's WAN traffic demands into an immediate component and a deferrable component. For instance, for the mailbox load balancing component described in Section 4.1, the domain experts require 10% of traffic to be flagged as immediate.

Forecasting: The network's provisioning process takes these inputs and forecasts future usage. To do this for immediate demands, it first aggregates hourly usage across all applications, and then computes the P95 usage over an extended period such as a day. Using this number across multiple days, it then determines a trend and forecasts future usage. For deferrable demands, the network forecasts daily volume of traffic for an extended period, say months, and inputs these to Approv. Appendix A.1 explains how Approv uses this input.

Run-time Interface: Since cooperative provisioning is done assuming that part of the demand is deferrable, cooperation of both the network and the application at run-time becomes necessary to live up to this assumption. Specifically, the application and the network need to react quickly and appropriately to link failure events. When a link fails, the network informs deferrable application components to either slow their rate of generating traffic or to pause such traffic altogether. We achieve this through a combination of two techniques: one enforced by the network and the other effected through application cooperation.

When traffic generated by a deferrable component is uniquely identifiable through network-visible identifiers, such as IP addresses or five-tuples, the network uses a bandwidth enforcer similar to previous work [12, 14] to apply backpressure on the application. When it detects a link failure, the bandwidth enforcer reduces the allocation to the deferrable component that needs to be slowed down. Substrate's deferrable components (such as load-balancing, database migration, and background cleanup processes) are designed to slow down when the bandwidth enforcer reduces their allocation. As a result the components either reduce the amount of work they do or pause it altogether. When the failed link comes back up, the bandwidth enforcer increases the allocated bandwidth and sends an explicit signal to the application components to resume normal activity.

Often, however, multiple application components use the same network identifier to send traffic and consequently, network bandwidth enforcement cannot isolate traffic from the individual application components to be slowed down or paused. Fortunately, Substrate uses a job scheduler that controls the progress of background and asynchronous tasks. Hence when the network informs the application of a link failure, the application's job scheduler explicitly slows the appropriate deferrable component(s), thereby keeping the application's network traffic in line with the reduced network capacity.

We are in the process of productizing WAN capacity provisioning at Microsoft based on Approv. This work is in collaboration with both the WAN team and the owners of the Substrate service. The implementation comprises Approv-based optimization of the offline provisioning pipeline and run-time adaptation . The latter comprises two parts. The first is bandwidth throttling enforced by the network, when there is loss of capacity due to link failure, to limit the traffic of applications that had marked part of their demand as deferrable during the offline provisioning phase. This ensures the protection of the network from overload during such times of capacity crunch, regardless of application actions. The second is run-time adaptation of the Substrate service, to ensure that the application throttles its activity, and hence traffic, in a manner that avoids user impact, e.g., by pausing deferrable components but not the user-facing ones. For this purpose, we have implemented interface enhancements using approximately 2500 lines of C# code, and are running comprehensive tests to verify that all Substrate's deferrable components are indeed able to pause or slow down when required.

4 Evaluation

In this section, we evaluate the benefits of cooperative provisioning. Our evaluation uses Microsoft inter-DC WAN topology and link failure characteristics, and applies Approv to satisfy demands for Substrate, a large-scale service that supports several collaboration applications such as email, shared file services, and enterprise analytics. Substrate accounts for well over a third of the overall traffic on Microsoft WAN and as such plays a significant role in defining the WAN capacity. Therefore, we believe it is useful to analyze capacity even just in the context of the Substrate service. We first describe our methodology, and then turn to our results.

4.1 Application Demands

Substrate consists of various *components* which perform different logical functions. Many components are user-facing; for instance, a component that responds to a user's REST API calls to read email. These components create immediate traffic demands. Additionally, Substrate implements a number of components that use the WAN extensively to ensure high data availability, reliability, and performance. These mostly run asynchronously and therefore their traffic demands are mostly deferrable. By interviewing component owners, we have determined the following four components whose application demands can be deferred.

Mailbox Load Balancing (MLB): This component is specific to the email application built on Substrate. With time, utilization on some servers (measured in terms of CPU usage, storage or IO capacity) can become disproportionately high. To preempt this from impacting user latency, the load-balancing component periodically schedules mailbox moves from heavily loaded to lightly loaded servers. Such moves can be deferred until a certain deadline. To determine the right deadline, the load-balancing team continuously monitors the free and available resources (i.e., the "headroom") on each server, e.g., the free storage available. Based on the rate at which utilization grows on each server and the available headroom on the various resources, the team calculates a deadline by which the load balancing must complete while still keeping the utilization under control and the user latency unaffected. Currently, this deadline is conservatively calculated as 7 days, which allows for occasional unexpected surges in load .

Periodic Tasks (PT): Substrate requires certain maintenance and analytics tasks to run periodically at a daily, weekly, or monthly cadence. These run in the background and perform two main functions. First, they perform data clean-up. For example, one task permanently deletes data items that are marked as deleted. Another task ensures compliance by performing time-driven deletions as mandated by legislation such as GDPR [2]. Second, they perform analytics tasks that extract useful enterprise-specific information from the data, such as generating weekly reports on how an employee splits their time between meetings and focus time. Based on conversations with owners of these tasks, we determined that these demands are deferrable by up to 1 day.

Server Initialization (SI): Substrate is a rapidly growing cloud service. To support this growth, it periodically adds new servers to its datacenters. This growth happens in bursts and is not gradual: a large number of new servers may become available at a particular datacenter in a particular month, and the next set of new servers may land only much later. The new servers are initialized with data from existing servers that may be in other datacenters, triggering large data transfers over the WAN. This workload is deferrable since such initialization does not have immediate, user-visible consequences. On the flip side however, it has to complete within a reasonable time so that the service can start utilizing the new servers, and thereby support growing demand. Per the operations team, 7 days is a reasonable deadline for the demand arising from the (new) server initialization. Although SI-based demands occur only sporadically, the network provisioning has to explicitly consider this demand since the traffic is bursty with a high peak.

Database Geo-Replication (DG): Substrate uses database-level geo-replication to ensure data availability. Changes to any data item are written to a database, and the database transaction log is replayed at three geo-replicated servers. Hence all components that modify data items generate geo-replication traffic. Replication traffic triggered by user-visible updates such as the creation of a new email is treated as immediate, while the rest is treated as deferrable, with a deadline that is the same as for the component that generates it: 7 days for MLB and SI, and 1 day for PT.

4.2 Evaluation Methodology

We first describe the inputs used to evaluate Approv. Then, we outline the different provisioning approaches we compare Approv with. Finally, we explain the network topology that we provision for and the link failure characteristics Approv uses to simulate failures.

4.2.1 Input Demands

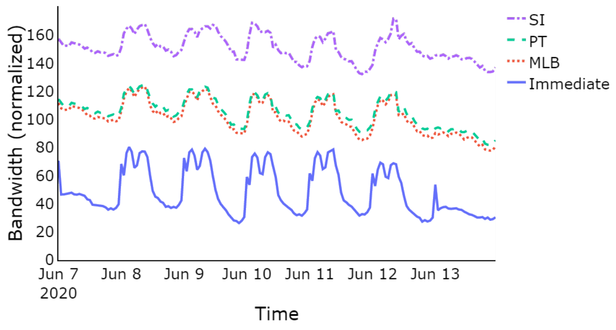


Figure 5: Substrate's network usage over a week in June 2020 on a link from Singapore to Hong Kong, separated into immediate and deferrable (MLB, PT and SI).

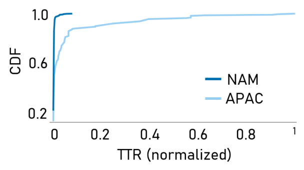
We use one month of network usage data from June 2020 to evaluate Approv. The deferrable demand constitutes 59% of total demand by volume, of which mailbox load balancing accounts for about 36%, periodic tasks for about 3% and server initialization for about 20% (see below for elaboration on SI traffic). Figure 5 shows immediate and deferrable demands (stacked over immediate) from a DC in Singapore (SIN) to a DC in Hong Kong (HKG) over one week in June 2020. The curve for each component includes both the traffic arising from updates to the primary replica and the database geo-replication triggered from updates to the secondary replicas.

We did not observe server initialization traffic in June 2020, which is the month for which we have detailed traffic traces. However, to determine the effect it would have on provisioning, we took an estimate of such sporadic traffic, obtained from the concerned team, and overlaid it on top of the total June 2020 traffic. The aggregate traffic traces so synthesized are then provided as input to the provisioning algorithm.

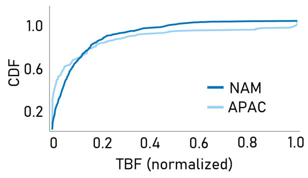
All demands are specified as a time-series in the format shown in Figure 4. Due to the commercial sensitivity of some of this data, as was alluded to in Section 1, we do not spell out the actual values here.

4.2.2 Provisioning Approaches

We quantify the benefits of Approv by comparing it with two alternative approaches:



(a) CDF of link TTRs



(b) CDF of link TBFs

Figure 6: WAN link reliability statistics.

1. *Baseline (BL)*: This is the current state-of-the-art provisioning approach mentioned in Section 3.1. There are several similarities between this baseline approach and the capacity planning scenario described in previous work [1, 21]. All application demands are treated equally. The demands are expressed as rates that need to be satisfied, and are derived by observing the P95 value of rate over the entire dataset. Failures are simulated as point-in-time events during which the demands are to be satisfied, rather than as failure episodes that the demand could tide over.
2. *Smoothing-only (SO):* This approach performs provisioning with differentiated application demands, but only for smoothing of the deferrable demand (expressed as a rate) to fit in the valleys of the immediate demand (see Figure 2) . In this approach , once the deferrable traffic is smoothed into the valleys, we determine the P95 rate over the dataset and perform provisioning using point-in-time failure simulation. This approach allows us to evaluate how just smoothing traffic , which is well studied in the context of traffic engineering [7, 13], can help improve capacity provisioning.

4.2.3 Network Topology and Link Characteristics

Each provisioning approach mentioned above includes failure simulation. Our failure simulation method is driven by the network topology and link failure data from Microsoft WAN. We concentrate on two regions, Asia-Pacific (APAC) and North America (NAM), since they represent two very diverse network topologies. NAM is the largest region in Microsoft WAN in terms of both the number of links and capacity (it has tens of datacenters and hundreds of links), but consists of mostly land-bound links. APAC, on the other hand, while being smaller, includes an extremely diverse set of links, with a combination of land-bound links and several under-sea cables, given the geography of the region. Figure 6 shows the difference in link failure characteristics between APAC and NAM. While the TTR is almost uniformly low for the links in NAM, there is wide variation in the TTR for APAC, because some links (e.g., those on undersea cables) take time to repair. The TBFs are fairly similar across both regions.

We evaluate each provisioning approach with two failure simulation methods:
1. *Replay-based:* We simulate failures by replaying the 16-month link-status vector history from Microsoft available to us, with each month treated as a separate failure scenario,

Failure Method	Region	SO (% savings)			Approv (% savings)		
		MLB	+PT	+SI	MLB	+PT	+SI
Replay	APAC	6.5	6.5	9.7	16.8	17.4	38.1
	NAM	9.8	10.4	10.8	28.0	29.2	30.5
Generated	APAC	5.3	5.3	8.2	24.9	25.2	44.2
	NAM	10.8	10.9	11.1	27.3	29.0	31.8

Table 2: Percentage capacity savings over Baseline for SO and Approv, for both replay and failure generation. We first show savings with only Mailbox Load Balancing (MLB) considered deferrable, then we add on Periodic Tasks (PT). Finally, we add Server Initialization (SI) traffic as estimated by the SI team (shaded columns).

yielding a total of 16 scenarios for each of APAC and NAM.
2. *Model-generated:* We use the failure model described in Section 3.4 to generate a synthetic but realistic set of link-status vectors.

While the replay-based approach enables evaluation independent of our failure generation model, the latter enables the creation of an unbounded number of failure scenarios (we create 10,000 for each of APAC and NAM), in turn allowing us to evaluate the impact of varying the probability of satisfaction, p (from 0.9 to 0.999).

4.3 Results

In this section, we first evaluate how Approv reduces the provisioned network capacity while supporting Substrate's demand, and improves link utilization compared to the baseline. Next, we perform a sensitivity analysis to evaluate how Approv's savings vary with d and p. Finally, we examine and compare forecasting for the baseline and Approv.

Network Capacity Reduction[2]: We first evaluate the percentage reduction in network capacity provisioned using the three approaches. We calculate the network capacity as the sum of all link capacities in the network. Table 2 shows the percentage network capacity reduction for the Asia-Pacific and North America regions, for both methods of simulating failures: replay-based and model-generated. As noted in Section 4.2.1, since the Substrate components that write to data items trigger database geo-replication (DG), we have included the DG demands in those of the respective Substrate components. Since the historical time-series contains 16 months of data, for the generated method, we generate link-status time-series of length 16 months too.

We concentrate on the last row of the table (NAM with failure generation), though the same explanation applies to the other rows as well. Approv, with MLB alone treated as deferrable, reduces provisioned capacity by 27.3%. With the deferrable set expanded to also include PT, which is a smaller workload and with a tighter deadline of only 1 day, the gains

[2]In general, we would also want to consider link-cost-weighted capacity savings, but we defer this to future work.

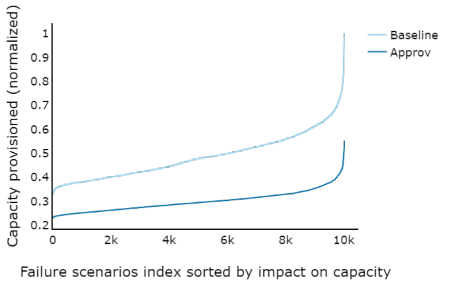


Figure 7: Incremental augments in network capacity for the APAC region for Baseline and Approv.

increase to 29%. Though it is a small increase, there is still value in considering demands like PT as deferrable, since even a small improvement in percentage capacity provisioned yields large absolute gains. Finally, the inclusion of SI as well in the deferrable set adds to the gains, taking it up to 31.8%. The benefits of Approv over SO are also apparent, since SO gives gains of 11.1% whereas Approv's total gains are much higher at 31.8%. Note that both failure simulation methods yield a significant reduction in the capacity to be provisioned, underscoring that the gains are not just specific to our failure model.

Including SI in the deferrable set provided significantly higher gains in APAC than NAM (e.g., an increase of about 20% in capacity savings in APAC compared to just 1-3% in NAM). This is because the number of servers added to APAC and NAM datacenters, and hence the volume of initialization data, was roughly the same in both regions. Since Substrate service is much larger in NAM, the addition contributed less WAN traffic relative to the total in NAM compared to APAC.

Link Utilization Improvement: We now discuss the relative improvement in link utilization due to Approv. We analyze the utilization improvement in APAC network provisioned with Approv using the replay failure scenarios. The relative improvement is computed as 100*(Approv utilization - Baseline utilization)/(Baseline utilization) for each link. Since Approv reduces network capacity significantly, the link utilization improve significantly as well, with an average relative improvement of 42% (note that we compute the utilization improvement on a per-link basis and report the arithmetic average whereas the capacity savings is reported in aggregate).

Sensitivity Analysis: Figure 7 shows 10000 generated failure scenarios for the APAC region on the x-axis, sorted by the total network capacity that is provisioned for each failure scenario considered individually. We use all traffic (Immediate, MLB, PT and SI) in this experiment. The graph shows that Approv almost uniformly provisions about 30% less capacity than Baseline. This shows that irrespective of the exact nature of the failure scenarios, Approv consistently helps reduce the network capacity to be provisioned.

Experiment	p=0.999	p=0.99	p=0.9
Baseline	1.00	0.90	0.75
d=12hrs	0.66	0.61	0.51
d=1day	0.62	0.56	0.47
d=2days	0.61	0.51	0.43
d=4days	0.60	0.51	0.42
d=7days	0.60	0.51	0.42

Table 3: The sensitivity of capacity provisioned to the deadline d (varied across rows) and the probability of satisfaction p (expressed in terms of 9s and varied across columns). We normalize all values by the capacity provisioned for the Baseline approach with $p = 0.999$.

Next, we evaluate how Approv's provisioned network capacity varies as we sweep through a range of settings for the deadline, d, and the probability of satisfaction, p. Table 3 shows the results in terms of the normalized capacity provisioned for the baseline case (the first row) and for Approv (the remaining rows).

We see that the capacity to be provisioned decreases as the probability of satisfaction p decreases and also as the deadline d increases. Both of these are as expected since the less "demanding" a demand is, the greater the opportunity Approv has to accommodate the demand without increasing the capacity to be provisioned. However, we also see that the capacity tends to flatten out as the deadline increases, with a diminishing benefit to relaxing the deadline. The reason is that Approv is able to effectively utilize the large amount of headroom in the network arising from the redundant capacity provisioned to support the immediate demand. Note that in the absence of failures, this redundant capacity is not needed for serving the immediate demand and so is available for Approv to accommodate the deferrable demand, even with a shorter deadline.

Table 3 also points to an interesting tradeoff between p and d. For instance, the normalized capacity (0.61) with $p = 0.99$ and $d = 12$hrs is roughly the same as it is (0.6) when p is made more demanding ($p = 0.999$) but d is relaxed to $d = 7$days.

Forecast: Finally, we evaluate the effect of forecasting error on Approv. We used 3 months of traffic to forecast 10 days of traffic using the Holt-Winter method of forecasting [6] with a seasonality of 7 days. To ensure that the network is able to accommodate sudden, unforeseen peaks in traffic, network forecasts typically use a high confidence interval (CI) envelope, hence, we use a confidence interval of 95% in our experiment. First, we forecast overall traffic, without differentiating immediate and deferrable traffic, as is the current state of the art. Next, we forecast immediate and deferrable traffic separately, as is required by Approv.

For overall demand, the forecast overestimation error for the high end of the 95% CI is 12%. When we separately forecast the deferrable demand, the forecast overestimation error is 11%. In other words, the forecasting is about as accurate for the deferrable part of the traffic as it is for the overall traffic. Therefore, Approv could as well use the forecast demand as input as it can the current snapshot of the demand (as we do in this paper, using the June 2020 snapshot). Furthermore, given the large amount of headroom that is created by provisioning of immediate traffic (shown in Table 3), we believe Approv has an overall low sensitivity to forecasting errors as well. For these reasons, we believe Approv will remain effective even when used with demand forecasts.

5 Discussion

In this paper, we have used the applications' deadline and desired probability of satisfaction information to provision the network optimally. One challenge that arises from this is incentivizing applications to specify these requirements, which convey their flexibility, appropriately. If the applications are too conservative in specifying their requirements, that would result in reduced capacity savings, whereas if the applications specify their demands loosely, then that might result in an incorrect forecast and hence inadequate network provisioning. We believe that this opens up the possibility of devising pricing mechanisms to encourage applications to specify their demands appropriately, thereby facilitating the cooperative provisioning of network capacity. Another challenge is in extending cooperative provisioning to third-party applications. Unlike with first-party applications, where we assume an implicit trust between the applications and the network that enables free sharing of information up and down the stack, in the third-party setting, we would need to rethink this approach and consider how cooperation can be effected in the absence of such trust.

6 Related Work

In this section, we position our work in relation to previous efforts in traffic engineering, network provisioning and failure characterization.

Traffic Engineering: As discussed in Section 1.1, traffic engineering employs routing and scheduling strategies to satisfy the possibly differentiated demands of applications on a given network. B4 [8] and SWAN [7] use routing algorithms that allow flows to specify different priority levels. Tempus [11] and Amoeba [22] allow applications to set deadlines for traffic, the former uses constraint optimization to satisfy deadlines while the latter uses a graph-based algorithm. Failure-aware traffic engineering has also been explored. FFC [18] splits traffic over multiple paths to be resilient to failures. Song et al. [17] proposed an availability-aware traffic engineering algorithm for optical networks that we believe can be generalized to WANs. NetStitcher [13] uses store-and-forward techniques to deal with temporal offsets between traffic peaks in different parts of the network.

Pretium [10] uses dynamic pricing to route third-party application traffic while handling link failures.

It is important to recognize that traffic engineering and network provisioning are fundamentally different problems. Traffic engineering addresses the short-term, i.e., run-time, problem of how networks route traffic while assuming given link capacities provided as input. Provisioning, on the other hand, addresses the longer-term problem of determining how link capacities should be provisioned in the first place, possibly several months or even over a year into the future, so as to satisfy the communication requirements of applications. In doing so, the provisioning framework (such as Approv in this paper) would invoke traffic engineering techniques under the hood and iteratively, to check if the demand can be satisfied by the network as provisioned and in the face of one or more failure scenarios. If traffic engineering is unable to route the demand, the capacity provisioned would be augmented.

Network Provisioning: We now discuss previous work that directly addresses the problem of network provisioning. Robust network validation [1] proposes a generic optimization framework to determine worst-case network performance across multiple scenarios. They use this framework to model the network provisioning problem which determines the right augments to link capacity so as to handle multiple failures. Liu et al. [21] propose an optimization framework that also solves the network provisioning problem given a set of failure scenarios. Both efforts address network provisioning, however, without using first-party context, and therefore consider all application traffic to be equivalent. Moreover, since they do not inherently consider deferrable demands, they use a simple failure model that only simulates instantaneous, point-in-time failures, with no notion of the temporal dynamics of link failures, i.e., a link failing at a certain time but then being restored at a later time. The baseline provisioning approach we have used in Section 4 is derived from these techniques.

Failure Characterisation: Turner et al. explore and characterize network failures in the CENIC network and provide a methodology to reconstruct these failures [20]. This is similar to the replay-based model we use in Section 4. Shaikh et al. [16] similarly measure link status over time using OSPF link-state advertisements in a large enterprise network. Gill et al. [5] characterize network failures within a datacenter, not on the WAN. Unlike our failure modeling methodology, these efforts analyze historical failures and do not propose generative failure models based on this history. Previous work [4] analyzes optical layer outages in a large backbone and shows that Q-drop events are predictive of upcoming failures. We believe that augmenting our failure model with information from the optical layer would be an interesting future direction.

7 Conclusion

We have described how cooperation, cutting across applications and the network, can significantly lower capacity provisioned in inter-DC WANs in a first-party setting. Using co-operative provisioning, we show how we can provision the WAN more carefully while ensuring that application demands are accommodated with the desired satisfaction probability.

Acknowledgements. We thank our shepherd, John Wilkes, and the anonymous reviewers for their feedback. We are particularly grateful to John for his meticulous shepherding of our paper. We also thank Neha Agrawal, Matt Calder, Wengjie Chen, Christina Chou, Pavel Egorov, Luis Irun-Briz, Jim Kleewein, Angus Leeming, Shijuan Lu, Otavio Pereira, Saravanakumar Rajmohan, Nadia Razek, Xiaoshi Sha, Jianke Tang, Jiaojian Wang, and Paul Wang for their help and input during the course of this work.

References

[1] Y. Chang, S. Rao, and M. Tawarmalani. Robust validation of network designs under uncertain demands and failures. In *USENIX NSDI*, pages 347–362, 2017.

[2] European Commission. 2018 reform of EU data protection rules. https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf. Accessed Sep 12, 2021.

[3] B. S. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster analysis 5th Edition*. John Wiley, 2011.

[4] M. Ghobadi and R. Mahajan. Optical layer failures in a large backbone. In *ACM IMC*, page 461–467, 2016.

[5] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *ACM SIGCOMM*, 2011.

[6] C. C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004.

[7] 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*, page 15–26, 2013.

[8] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, K. N. B., C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, and A. Vahdat. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN. In *ACM SIGCOMM*, page 74–87, 2018.

[9] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu,

J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM*, page 3–14, 2013.

[10] V. Jalaparti, I. Bliznets, S. Kandula, B. Lucier, and I. Menache. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In *ACM SIGCOMM*, pages 73–86, 2016.

[11] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula. Calendaring for wide area networks. In *ACM SIGCOMM*, pages 515–526, 2014.

[12] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat. BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In *ACM SIGCOMM*, page 1–14, 2015.

[13] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with netstitcher. In *ACM SIGCOMM*, page 74–85, 2011.

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

[15] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[16] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb. A case study of OSPF behavior in a large enterprise network. In *ACM SIGCOMM Workshop on Internet Measurment*, page 217–230, 2002.

[17] L. Song, J. Zhang, and B. Mukherjee. Dynamic provisioning with availability guarantee for differentiated services in survivable mesh networks. *IEEE Journal on Selected Areas in Communications*, 25(3):35–43, 2007.

[18] M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford. Network architecture for joint failure recovery and traffic engineering. *SIGMETRICS Perform. Eval. Rev.*, 39(1):97–108, June 2011.

[19] S. Taylor and B. Letham. Forecasting at scale. PeerJ Preprints 5:e3190v2 https://doi.org/10.7287/peerj.preprints.3190v2, 2020. [Accessed March 10, 2021].

[20] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage. California fault lines: Understanding the causes and impact of network failures. In *ACM SIGCOMM*, 2010.

[21] Yu Liu, D. Tipper, and P. Siripongwutikorn. Approximating optimal spare capacity allocation by successive survivable routing. In *IEEE INFOCOM 2001*, pages 699–708 vol.2, 2001.

[22] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang. Guaranteeing deadlines for inter-data center transfers. *IEEE/ACM Transactions on Networking*, 25(1):579–595, 2016.

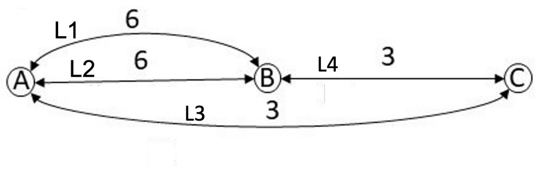


Figure 8: Toy Example

A Appendix

A.1 LP formulation

Objective Function: Minimize weighted sum of capacity augments where weight could be link cost, latency, etc.

$$\text{Minimize} \sum_{l} w_l \cdot X_l \tag{1}$$

Subject to the constraints:

A. Demand constraints at sources and destinations:

$$\sum_{t_i \le t \le t_i + d_i} \sum_{l \in OL_{S_i} \cap L_{R_i}} v_{t,l}^{f,i} = V_i \qquad \forall f \in F, i \in D \tag{2}$$

$$\sum_{t_i \le t \le t_i + d_i} \sum_{l \in IL_{T_i} \cap L_{R_i}} v_{t,l}^{f,i} = V_i \qquad \forall f \in F, i \in D \tag{3}$$

Constraint (2) ensures that for any deferrable demand, i, the total volume of traffic emerging from its source node, S_i, over all time steps within its deadline is equal to the actual demand volume V_i. Constraint (3), on the other hand, ensures that the total volume traffic arriving at the destination node over all time steps within its deadline is also equal to actual demand volume V_i. In computing the total volume in constraint (2), we only consider the subset of links emanating from the source, S_i, that also lie on a valid path between S_i and the destination, T_i, and likewise in constraint (3).

B. Network flow constraint at each node, n:

$$\sum_{i \in D} \sum_{l \in IL_n} v_{t,l}^{f,i} + \sum_{\forall i | S_i = n} V_i = \sum_{i \in D} \sum_{l \in OL_n} v_{t,l}^{f,i} + \sum_{\forall i | T_i = n} V_i \qquad \forall f \in F, n \in N, t \in T \tag{4}$$

This is a network flow constraint to ensure that, in each time slice, the sum of the volume coming into the node and volume generated at that node is equal to the sum of volume going out of that node and volume sinking into that node.

C. Capacity constraint at each link, l:

$$(E_l + X_l) \cdot u_{t,l} \ge I_{t,l}^{f} + \sum_{i \in D} v_{t,l}^{f,i} \qquad \forall l \in L, f \in F, t \in T \tag{5}$$

This constraint ensures that the total volume of traffic that a link is able to carry during a time step (computed as a product of the total capacity provisioned on that link to accommodate the immediate and deferrable demands and the uptime of the link) should be greater than or equal to the sum of immediate and deferrable demands volume routed via that link during that time step.

Note that all of the above constraints apply during each failure scenario, f, so there is a set of such constraints corresponding to each such failure scenario.

A.2 Example of Cooperative Provisioning

We use a toy example network (Figure 8) with 3 datacenters and 2 immediate and 2 deferrable demands to illustrate steps 1 and 2 from Section 3.3. Table 4 shows the TTR and TBF of each link. For fair comparison with the baseline, we simulate 2-link simultaneous failures, in which links L1 and L3, and links L2 and L3 can fail together (the high TTR of 15 days for link L3 makes it more likely that its failures would overlap with those of the other links), along with all single link failures. Table 5 shows the input in terms of the immediate and deferrable traffic demands. Immediate demands R1 and R2 require a peak rate of 3 GB/day but sends 2 GB/day on average, hence sending total 60 GB in a period of 30 days.

1. Baseline :
Table 6 shows point-in-time failure scenarios simulated as part of the baseline provisioning described in the Section 4. Consider the point-in-time failure scenario 2 in Table 6, where links L1 and L3 fail together, hence the required rate between A and B is 6 GB/day to satisfy demands R1 and R3 and required rate between A and C is 9 GB/day to satisfy demands R2 and R4. And as L2 and L4 constitute the only available path, we need to allocate at least 15 GB/day capacity on L2 and at least 9 GB/day capacity on L4, hence capacity augments of 9 GB/day and 6 GB/day are required on L2 and L4, respectively. Provisioning for all such failure scenarios results in a total augment of 30 GB/day. With all demands expressed as peak rates that must be satisfied even when there are failures, point-in-time failure scenarios are effectively equivalent to a complete failure of a link over the entire 30-day period . Such a conservative approach is unnecessary for deferrable demands provisioning, as discussed next.
2. Approv:
Step I : We do peak-rate based point-in-time failure scenario provisioning for immediate demands which is similar to baseline. Considering the same point-in-time failure scenarios as in the baseline provisioning, we find that the topology in the

Failure Scenario	X1	X2	X3	X4
No failure	0	0	3	0
L1 and L3 fails	0	9	0	6
L2 and L3 fails	9	0	0	6
L4 fails	0	0	3	0
Final Augments	9	9	6	6

Table 6: Baseline provisioning with point-in-time failure scenarios

Failure Scenario	X1	X2	X3	X4
No failure	0	0	3	2
L1: [15, 17], L3:[15, 30]	0	0	0	5
L2: [15, 20], L3:[15,30]	0	0	0	5
L4: [15, 20]	0	0	5	0
Final Augments	0	0	1.67	5

Table 7: Volume based LP provisioning with timeseries failure scenarios

Link	TTR	TBF
L1	2	30
L2	5	60
L3	15	30
L4	5	60

Table 4: Link TTRs

Figure 8 is enough to satisfy the immediate demands, R1 and R2.

Step II: In this step, we provision for deferrable demands R3 and R4 on top of the Step 1 topology, while working with time series failure scenarios, as discussed in Section 3.4. Note that we have only shown a few failure scenarios (which cause the highest augments) due to lack of space but the final augments are based on simulating plenty of such failure scenarios.

Consider the failure scenario 2 in which L1 fails at day 15 and comes back up at day 17 and L3 fails at day 15 and comes back up at day 30. We don't need to provision extra capacity for demand R3 because if we send more volume of R3 between day 0 and day 15, we would not need to send any traffic between day 15 and day 17. On the other hand, since the demand R4 starts at day 15 and we need to send 90 GB in a period of 15 days, which coincides with the downtime of link L3, we would need to provision capacity on an alternate path to accommodate demand R4 in the face of such a link failure. Specifically, if we consider link L4 on the alternative path, 15 GB out of the total R4 demand of 90 GB could be accommodated in the valleys of immediate demand R2. That would leave 75 GB of R4 to be satisfied, necessitating 75/15 = 5 GB/day of extra capacity on link L4 to satisfy deferrable demand R4, alongside immediate demand R2 flowing over the same link, L4. Note that since we solve one LP optimally across all failure scenarios, the final capacity augments yielded by our LP is not necessarily the same as the maximum capacity augment required on each link across the individual failure scenarios.

Request	Type	Peak Rate	Start Time	Deadline	Volume
R1: A->B	Immediate	3 GB/day	-	-	60 GB (30 days)
R2: A->C	Immediate	3 GB/day	-	-	60 GB (30 days)
R3: A->B	Deferrable	-	0th day	30 days	90 GB
R4: A->C	Deferrable	-	15th day	15 days	90 GB

Table 5: Immediate and Deferrable demands

Some observations on Approv provisioning:

1. Baseline provisioning requires 30 GB/day total capacity augments whereas Approv requires only 6.67 GB/day total capacity augments.

2. For temporally overlapping demands, Approv will find optimal allocation. For example, deferrable demand R3 will send more bytes in the first 15 days to utilize the network well and will leave enough capacity on links L1 and L2 for sending R4 traffic between day 15 and day 30.

3. In our example, deferrable demands could be routed via either link L3 or link L4, but Approv returns higher augments on link L4 because it has lower TTR value. In effect, Approv favors augmenting low TTR and high TBF links, i.e., high-availability links.

4. As our objective function is to minimize total weighted capacity augments, Approv favors augmenting shorter paths.

5. With our volume-based LP formulation, we do not need to explicitly smooth our input demands, since Approv fills deferrable traffic in the immediate traffic valleys implicitly as part of its optimization.