2009-68

The Virtual Network Scheduling Problem for Heterogeneous Network Emulation Testbeds

Authors: Charlie Wiseman, Jonathan Turner

Corresponding Author: wiseman@wustl.edu

Abstract: Network testbeds such as Emulab and the Open Network Laboratory use virtualization to enable users to define end user virtual networks within a shared substrate. This involves mapping users' virtual network nodes onto distinct substrate components and mapping virtual network links onto substrate paths. The mappings guarantee that different users' activities can not interfere with one another. The problem of mapping virtual networks onto a shared substrate is a variant of the general graph embedding problem, long known to be NP-hard. In this paper, we focus on a more general version of the problem that supports advance scheduling of virtual network mappings. We experimentally study the performance of heuristic testbed schedulers in the context of the Open Network Laboratory. Our algorithms incorporate Mixed Integer Programs to optimally solve key subproblems, are fast enough to respond to reservation requests in under one second, and rarely reject requests needlessly.

Type of Report: Other

The Virtual Network Scheduling Problem for Heterogeneous Network Emulation Testbeds

Charlie Wiseman, Jonathan Turner
Applied Research Laboratory
Washington University in St. Louis
{wiseman,jon.turner}@wustl.edu

***Abstract*—Network testbeds such as Emulab and the Open Network Laboratory use virtualization to enable users to define end user *virtual networks* within a shared *substrate*. This involves mapping users' virtual network nodes onto distinct substrate components and mapping virtual network links onto substrate paths. The mappings guarantee that different users' activities can not interfere with one another. The problem of mapping virtual networks onto a shared substrate is a variant of the general graph embedding problem, long known to be NP-hard. In this paper, we focus on a more general version of the problem that supports advance scheduling of virtual network mappings. We experimentally study the performance of heuristic testbed schedulers in the context of the Open Network Laboratory. Our algorithms incorporate Mixed Integer Programs to optimally solve key subproblems, are fast enough to respond to reservation requests in under one second, and rarely reject requests needlessly.**

I. Introduction

Network emulation testbeds provide a safe, controlled environment for researchers and educators to conduct experiments across a wide range of emulated network configurations. Most emulation testbeds have two key characteristics that increase their utility. First, they support multiple concurrent experiments so that many users can use the testbed simultaneously. Second, they guarantee that each active experiment is isolated from all others, giving the appearance of a dedicated, non-shared infrastructure to each experiment. This is accomplished with a collection of switches and routers that indirectly connect all of the resources in the testbed and an associated testbed *scheduler*.

The scheduler takes a virtual network request and attempts to find a mapping from the virtual network onto the available physical resources. If a mapping is found, the physical resources in that mapping are allocated to the user and the physical network is configured (with VLANs) to emulate the virtual topology. The scheduler must ensure that each physical node is mapped to only one virtual node across all virtual networks (virtualized resource instances can be treated as separate physical nodes). Moreover, there must be enough capacity in the infrastructure switches and routers to support every virtual link without causing interference with other virtual links. Any mapping that meets these requirements could be returned by the scheduler. There are two testbed design choices that affect the associated testbed scheduler.

One of these design choices is how the resources are allocated to users. That is, the resources are either given on-demand or reserved in advance. In the former case, the scheduler looks strictly at the physical resources that are not in use by any other experiment, as in standard admission control. This is how most emulation testbeds operate. The latter case is somewhat more complicated. The scheduler must keep a time line of *reservations* that determines what resources are available at any given time. In addition to the virtual network to be emulated, user requests include a period of time when that virtual network should be active. When a new request is made, all previously accepted reservations that overlap are considered. Any reservations whose start time has not come (i.e., are not yet active) could potentially be remapped to a new set of physical resources. Clearly, maintaining a schedule of network mappings is a generalization of pure on-demand admission control.

The second testbed design choice is whether or not the testbed nodes are *typed*. In most testbeds, homogeneous PCs are the only user-allocatable resource, meaning that all physical nodes can be treated more or less equally. Other testbeds support heterogeneous resources. This might include many different PC configurations or other networking technologies altogether, such as programmable routers and reconfigurable hardware. In this case, every virtual and physical node is assigned a type that reflects these diverse resource possibilities. Naturally, finding mappings with typed nodes is a more general problem than with non-typed nodes, which results in more complicated schedulers.

Our work in this area is driven by the Open Network Laboratory (ONL) [1][2] testbed, which is an emulation testbed focused on providing users with heterogeneous, highly configurable resources. As such, ONL schedulers must support typed nodes. ONL also uses the reservation model, which further complicates the design of ONL schedulers.

Scheduling virtual networks in testbeds is a variant of the general network embedding problem, which is known to be NP-hard. As such, our attention is focused on heuristic schedulers. In particular, we present a new class of schedulers for ONL that use Mixed Integer Programs (MIPs) to optimally solve key subproblems by incorporating knowledge of the physical network topology of the testbed. These new schedulers are evaluated and results presented for response times to new requests and for the probability of rejecting requests. There are a large number of factors which affect the performance, including the size and shape of the virtual and

physical topologies, reservation durations, and user flexibility. Another major contribution of this work, then, is providing a new way to characterize the work load for testbed schedulers to assist in understanding the limitations and bottlenecks of different approaches.

The rest of the paper is organized as follows. Section II covers related work. Section III gives the formal problem statement for building schedules of mappings between virtual and physical networks in emulation testbeds. Section IV describes the new schedulers for our testbed. Performance results for the schedulers are presented in Section V. Section VI discusses the results and their implications for future work. Finally, section VII concludes the paper.

II. Related Work

Emulab [3] is one of the most popular emulation testbeds. It has been widely used for research and can often be found as part of the experimental evaluation in papers at top networking conferences. The testbed provides access to a large number of heterogeneous PCs which are used to emulate many different types of networks. The Emulab scheduler, *assign*, is based on simulated annealing [4]. It has support for typed nodes in order to take advantage of the different PC classes available in the testbed, but does not support future resource reservations. Emulab does, however, allow users to put their rejected experiment requests into a scheduling queue that *assign* periodically reevaluates. In this case, the scheduler is still only performing admission control. Recent work [5] suggests that Emulab is working on support for a more general resource reservation model. The Emulab software has also been used in a number of other testbeds, including DETER [6] and WAIL [7].

There is a related embedding problem in overlay testbeds, such as PlanetLab [8]. PlanetLab nodes are PCs connected directly to the Internet, and users choose specifically which nodes they want to use as part of their experiment. Many users select nodes in an ad-hoc manner, but there has been some effort to allow users to make more informed decisions. Services like SWORD [9] gather real-time data about resources available on nodes (e.g., CPU and memory utilization) and the network paths between pairs of nodes (e.g., path capacity and latency). Given that information, users specify node and path constraints for their desired experiment, and standard constraint satisfaction techniques are used to find a mapping [10]. Of course, there are no guarantees that resources remain at the current utilization levels after they are chosen for an experiment.

Similar ideas have also been extended for use in network virtualization research. In this case, users specify full virtual network topologies that are embedded into a well-known and provisioned substrate network. Although the mapping problem is generally the same as in emulation testbeds, the goals are somewhat different. It is usually assumed that the virtual network can be embedded in multiple ways, and schedulers attempt to find the "best" solution. One approach is to use constraint satisfaction, as above, to try to find a minimum (monetary) cost embedding [11]. Other approaches include standard optimization techniques which focus on balancing load across nodes and along network paths [12] and some work on redesigning the substrate to make the embedding problem simpler [13]. These ideas also carry over to other areas such as embedding routings in wireless sensor networks [14].

III. Testbed Scheduling

At the core of building schedules of testbed resources is finding mappings from user-defined virtual networks to a fixed physical testbed network. Not surprisingly, it is easiest to reason about network mappings using network graph structures and algorithms. Indeed, many existing graph problems are related to the testbed mapping problem, including multi-commodity flow and subgraph isomorphism. Note that the general testbed mapping problem is known to be NP-hard by reducing to the multiway separator problem [15].

User virtual networks and the physical testbed network are represented as undirected graphs. The physical testbed topology will be referred to as the *testbed* graph and be denoted $T = (V^T, E^T)$. A virtual user topology will be referred to as a *user* graph and be denoted $U = (V^U, E^U)$. Edges have capacities equal to network link capacities, denoted by $cap(e)$ for all e in E^T or E^U. Each vertex has an associated type which corresponds to a particular kind of resource in the testbed (e.g., a router or PC), denoted by $type(v)$ for all v in V^T or V^U.

The testbed graph has one special vertex type used to represent the switches and routers that indirectly connect all of the other nodes in the testbed. These *infrastructure* vertices are hidden from users and are never directly part of any user graph. Instead, they are used to form paths through T that correspond to edges in U. An example is shown in Figure 1, where the dashed line in each graph shows one such mapping from an edge in U to a path in T. A complete mapping from U to T is represented by $M = (M^V, M^E)$. $M^V = \{(v_1^U, v_1^T), (v_2^U, v_2^T), \ldots\}$ is a set of vertex mappings, where $v_i^U \in V^U$ and $v_i^T \in V^T$. $M^E = \{(e_1^U, \rho_1^T), (e_2^U, \rho_2^T), \ldots\}$ is the set of edge to path mappings described above, where $e_i^U \in E^U$ and $\rho_i^T = (e_1^T, e_2^T, \ldots, e_k^T), e_i^T \in E^T$, is a path in T. Every mapping must be *consistent* with the associated user graph and testbed graph. This means that every edge and vertex in U is contained in the mapping, and that the endpoints of each edge in U must be mapped to the endpoints of the corresponding path in T:

$$\forall (v^U, w^U) \in E^U,\ (v^U, v^T), (w^U, w^T) \in M^V \iff \quad \text{(III.1)}$$

$$((v^U, w^U), \rho^T) \in M^E \text{where } \rho^T = ((v^T, z_1), \ldots, (z_m, w^T))$$

Recall that users request a reservation for resources in advance. This request is defined as $R = (U, A, l)$, where U is the user graph representing a virtual network, $A = (t_1, t_2)$ is the range of acceptable start times, and l is the length of time they need to run their experiments. If a request is accepted, it can not be moved in time or revoked. It is, however, permitted to remap any future reservations (i.e., those with begin times after the current time) on to a different set of

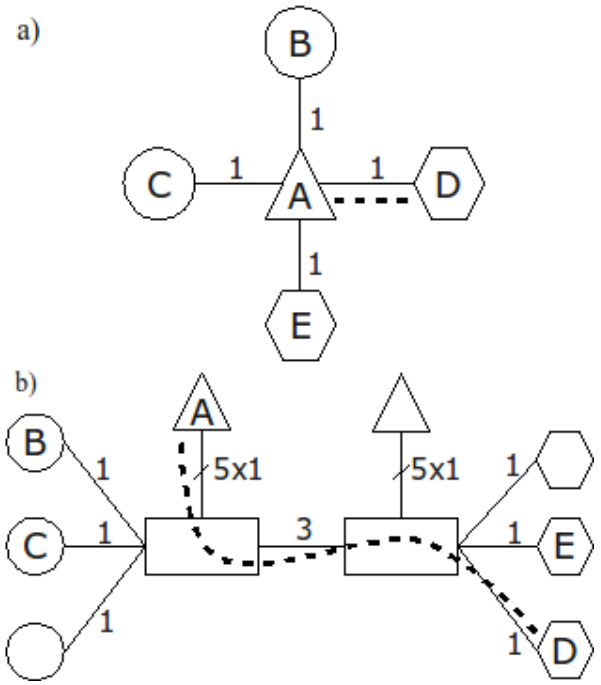

Fig. 1. a) Example user graph, and b) example testbed graph. Different shapes represent different types, with rectangles representing infrastructure nodes. The edge labels are the edge capacities. Node labels show an example mapping from nodes in the user graph to nodes in the testbed graph. The dashed lines show one mapping from an edge in the user graph to the corresponding path in the testbed graph.

physical resources. As such, the scheduler keeps an ongoing schedule of accepted requests. This schedule is represented by the set $S = \{S_1, S_2, \ldots, S_n\}$, where $S_i = (U_i, M_i, b_i, f_i)$ is one accepted request with user graph U_i, mapping M_i from U_i to T, request begin time b_i, and request finish time f_i. Naturally, the begin time must fall within the user's provided start time range and the reservation must be the correct length:

$$t_1 \leq b_i \leq t_2 \tag{III.2}$$

$$b_i < f_i = b_i + l \tag{III.3}$$

The problem, then, is to find a mapping from a new user graph to the testbed graph that is *feasible* given the current schedule. The schedule needs two properties to remain feasible. First, at any time, each vertex in T is mapped to at most one vertex over all U_i in the schedule:

$$\begin{aligned}\forall v^T \in V^T, (1)\ (v^U, v^T) \notin M_i^V, \forall i, 1 \leq i \leq n, v^U \in V_i^U \text{ or} \\ (2)\ (v^U, v^T) \in M_i^V, \exists i, 1 \leq i \leq n, v^U \in V_i^U \text{ and} \\ (w^U, v^T) \notin M_j^V, \forall i \neq j, w^U \neq v^U \in V_j^U\end{aligned} \tag{III.4}$$

Second, at any time, the sum of all user graph edge capacities that are mapped to each edge in the testbed graph must be less than or equal to that edge's capacity:

$$\forall e^T \in E^T, cap(e^T) \geq \sum_{e \in X_e^T} cap(e), \text{ where} \tag{III.5}$$

$$X_e^T : \{e^U |\ (e^U, \rho^T) \in M_i^E,\ \exists i, 1 \leq i \leq n, \text{and } e^T \in \rho^T\}$$

The formal problem statement follows. Given a testbed graph T with typed vertices and edge capacities, a schedule $S = \{S_1, S_2, \ldots, S_n\}$ of previously accepted reservations, and a new reservation request $R = (U, A, l)$, find a new feasible schedule $S' = S \cup \{(U, M', b, f)\}$ where M' is a consistent mapping from U to T subject to conditions (III.1), (III.2), (III.3), (III.4), and (III.5). If no such mapping exists, leave S unchanged and reject R.

IV. Heuristic Schedulers

Finding mappings from user graphs to the testbed graph is already NP-hard, and our schedulers must add a time component to the problem. Fortunately, testbed graphs in emulation testbeds generally fall into one of only a few basic designs. We use this knowledge to build heuristic schedulers that incorporate the testbed graph structure into Mixed Integer Programs that solve the mapping problem for a single user graph.

The simplest testbed graph structure is a linear series of N infrastructure nodes. Each user-allocatable node is connected to exactly one of these infrastructure nodes, i.e., nodes with multiple edges are connected to the same infrastructure node. This is the structure used in ONL. The actual ONL testbed graph is shown in Figure 2. This testbed graph will be used for the evaluation in the next section. The edge capacities shown are in Gb/s. Note that the labels next to nodes are used to indicate how many of that type of node are connected to the same infrastructure node. For example, the circle in the upper left actually represents four nodes of the same type, each with eight 1 Gb/s links to the left-most infrastructure node.

Although the problem formulation specifically allows accepted reservations to be remapped to a different set of testbed nodes, the schedulers presented here do not do so. Instead, they attempt to build schedules that maximize the probability of accepting future requests. We consider two variations of the same basic scheduler that differ in the heuristic used to achieve this goal. The first, denoted *MINBW*, minimizes usage of bandwidth between infrastructure nodes. The second, denoted *MAXPACK*, extends the first by computing *packing* scores for the user graph across all subsets of infrastructure nodes and considering potential subsets in order of the packing score. These two schedulers are clearly related, but the distinction is made in order to better characterize different approaches to the scheduling problem. Note that both of the following descriptions are applicable only for a linear testbed graph, but similar approaches could work for star and tree topologies.

A. Minimizing Bandwidth

The pseudocode for *MINBW* is shown in Algorithm 1. As described in the previous section, the inputs are the user's virtual network graph U, the range of acceptable start times A, the length of the requested experiment l, the testbed graph T, and the current schedule of accepted reservations S. The first step is to compute the set of potential begin and finish times, P, for this request. Resource availability only changes at existing reservation boundaries, so we need only consider candidate

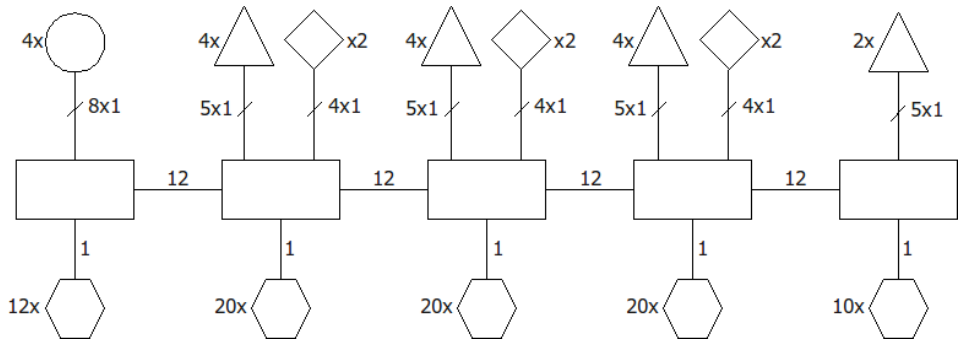

Fig. 2. The ONL testbed graph. Different shapes represent different types, with rectangles representing infrastructure nodes. The edge labels are the edge capacities. Labels next to nodes indicate how many of that type of node are present on the connected infrastructure node.

start times that correspond to the finish time of a existing reservation. For example, if no reservations overlap with the acceptable time range for U, then $computePossibleTimes$ will return only one possible time in P.

Next, *MINBW* attempts to find a mapping from U to T for each time in P. Recall that these schedulers do not attempt to remap any previous reservations. As such, $findOverlappingReservations$ is used to get the set of reservations that intersect (b, f), and then all resources from the overlapping reservations are removed from T to form T'. Here, subtraction means that all testbed nodes from reservations in O are removed from V^T and all capacity used along edges in O is subtracted from the edge capacities in E^T. $enoughNodesAvailable$ is called next to check that there are at least as many nodes of each type in T' as in U. If there are not, then there is no reason to continue. In fact, even a scheduler that attempts to remap other reservations can never accept U at this point because there are not enough nodes of each type left. This will be used in the evaluation section to give bounds on the rejection rate of the scheduler.

If there are enough nodes of each type in T', then $findMapping$ is called to try to find a mapping from U to T'. If a valid mapping is found, then that mapping is added to the schedule. Otherwise the next candidate time is tested, until either a valid mapping is found or there are no more times to try.

Algorithm 1 MINBW(U, A, l, T, S)

```
P ⇐ computePossibleTimes(A, l, S)
for all (b, f) ∈ P do
  O ⇐ findOverlappingReservations(b, f, S)
  T' ⇐ T − O
  if enoughNodesAvailable(U, T') then
    M ⇐ findMapping(U, T')
    if validMapping(M) then
      S ⇐ S ∪ (U, M, b, f)
      return TRUE
    end if
  end if
end for
return FALSE
```

$findMapping$ is the Mixed Integer Program that is the core of our schedulers. The basic idea is to directly incorporate the structure of the testbed graph into the MIP and then have the MIP compute a mapping from the user graph to the testbed graph directly. For all of our schedulers, $findMapping$ uses an objective function that minimizes the bandwidth used on the edges between infrastructure nodes. The $findMapping$ pseudocode is shown in Mixed Integer Program 1.

There are a number of constants used in the MIP formulation that are derived from U and T. N is the number of infrastructure nodes in the testbed graph. N=5 for the ONL topology. Recall that we are considering linear testbed graphs, where the N infrastructure nodes are connected in a line by N–1 edges. These edges are referred to as *infrastructure edges*. e_i^T is the edge from infrastructure node i to infrastructure node i+1. The infrastructure nodes are thus ordered so that infrastructure nodes i and i+1 are adjacent in the linear topology. L is the number of node types in U. For example, in Figure 1a, there are three different node types, so L=3. The MIP will have to ensure that the number of each type mapped to individual infrastructure nodes does not exceed the number available on that infrastructure node. α_j, then, is the set of nodes of type j in U, and β_{ij} is the number of nodes of type j available on infrastructure node i.

There are only two sets of variables in the MIP. First, $I_i(u)$ is a binary variable that is used to indicate which infrastructure node each $u \in V^U$ is mapped to. That is, $I_i(u) = 1$ if and only if u is mapped to infrastructure node i, and $I_i(u) = 0$ otherwise. Second, $x_i(e)$ is a positive, real-valued variable representing the bandwidth used between infrastructure node i and infrastructure node i+1 due to edge $e \in E^U$. For example, the highlighted edge to path mapping in Figure 1 would cause $x_1(e = (A, D)) = 1$, but $x_1(e = (A, B)) = 0$. Note that even though the $x_i(e)$ variables are allowed to be real-valued, the MIP formulation will force every $x_i(e)$ to be either zero or the edge capacity. Bandwidth used between infrastructure nodes is then minimized by setting the objective function to minimize the sum of all $x_i(e)$ values.

There are four sets of constraints which govern the MIP. The first set ensures that the total bandwidth mapped to each infrastructure edge is less than the available capacity on that

TABLE I
THE VALUES OF $x_i(e)$, $e=(u,v)$, GIVEN THE MAPPING OF u AND v TO INFRASTRUCTURE NODES.

$x_i(e=(u,v))$	$I_k(u)$=1, $\exists k \leq i$	$I_k(u)$=1, $\exists k > i$
$I_k(v)$=1, $\exists k < i+1$	0	$cap(e)$
$I_k(v)$=1, $\exists k \geq i+1$	$-cap(e)$	0

edge. The second set ensures that every node in U is mapped to exactly one infrastructure node. The third set uses the α_j and β_{ij} constants to ensure that the number of each type of user node mapped to each infrastructure node is less than the number available on that node, as discussed above. All of these constraints are reflecting the conditions described in the previous section. The last set of constraints, on the other hand, ties the MIP variables together.

Each constraint from this last set is used to set the value of the associated $x_i(e)$ variable based on the mapping of e=(u,v)'s endpoints to infrastructure nodes. For each value of i, the constraint ensures that $x_i(e)$=$cap(e)$ if and only if e is mapped to a path in the testbed graph that uses infrastructure edge i, and $x_i(e)=0$ otherwise. The mapping of u and v determines that path. Consider partitioning the testbed graph along infrastructure edge i such that all infrastructure nodes 1,...,i are on the "left" and infrastructure nodes i+1,...,N are on the "right". Then $x_i(e)$=$cap(e)$ if u and v are mapped to different sides of this partition. Recall that $I_k(u)$=1 indicates that user node u is mapped to infrastructure node k. So, $\sum_{k=1}^{i} I_k(u)$ is 1 when u is on the left of infrastructure edge i and 0 when it is on the right. The same is true for v. There are thus four cases for the values of $I_k(u)$ and $I_k(v)$ that determine the value assigned to $x_i(e)$, as shown in Table I. If both endpoints are mapped to the same side of the partition, then the total of both sums in the constraint is 1, leading to $x_i(e)$=0. Otherwise, the end points are mapped to different sides of the partition, leading to either $x_i(e)$=$cap(e)$ or $x_i(e)$=$-cap(e)$. Although it is not shown in the formulation for clarity, we use the standard MIP technique of creating two variables $x_i^+(e)$ and $x_i^-(e)$ that are used in place of $x_i(e)$, where $x_i^+(e)-x_i^-(e) = x_i(e)$ and $x_i^+(e)+x_i^-(e) = |x_i(e)|$. Note that this formulation assumes that the infrastructure nodes are each internally non-blocking. If that is not the case, then additional constraints could be added to ensure the forwarding capacity of infrastructure nodes is not exceeded.

If $findMapping$ finds a solution to the MIP, then the values of $I_i(u)$ contain a valid mapping from U to T, which is then returned to *MINBW*. By minimizing the bandwidth across infrastructure nodes, $findMapping$ is attempting to conserve as much of a scarce resource as possible for future requests. If, however, the infrastructure edge capacities are high, then it is possible that there could be more contention over node usage.

Mixed Integer Program 1 findMapping(U,T)

Let N be the number of infrastructure nodes in T
Let e_i^T, $1 \leq i \leq N-1$, be the edge connecting infrastructure nodes i and $i+1$
Let L be the number of distinct node types in U
Let α_j, $1 \leq j \leq L$, be the set of nodes of type j in U
Let β_{ij}, $1 \leq j \leq L$, $1 \leq i \leq N$, be the number of nodes of type j on infrastructure node i

Variables: $\forall u \in V^U$ and $\forall i, 1 \leq i \leq N$, $\quad I_i(u) \in \{0,1\}$
$\forall e \in E^U$ and $\forall i, 1 \leq i \leq N-1$, $\quad x_i(e) \geq 0$

Objective: $\min \sum_{e \in E^U} \sum_{i=1}^{N-1} x_i(e)$

s.t. $\forall i, 1 \leq i \leq N-1$, $\sum_{e \in E^U} x_i(e) \leq cap(e_i^T)$

$\forall u \in V^U$, $\sum_{i=1}^{N} I_i(u) = 1$

$\forall j, 1 \leq j \leq L$, and $\forall i, 1 \leq i \leq N$, $\sum_{u \in \alpha_j} I_i(u) \leq \beta_{ij}$

$\forall e=(u,v) \in E^U$, and $\forall i, 1 \leq i \leq N-1$,

$$x_i(e) + cap(e) \left(\sum_{k=1}^{i} I_k(u) + \sum_{k=i+1}^{N} I_k(v) \right) = cap(e)$$

B. Maximizing Packing

The second scheduler, *MAXPACK*, is derived from *MINBW* and shares the same basic structure. *MAXPACK* uses a modified heuristic based on node packing to try to increase the probability that future requests can be accepted. As in *MINBW*, *MAXPACK* does not attempt to remap any existing reservations.

The pseudocode for *MAXPACK* is given in Algorithm 2. Everything is the same as in *MINBW* until after $enoughNodesAvailable$ is called. If there are enough nodes of each type available in T', then $findValidSubsets$ computes the set Q that contains all *subsets* of T' that have at least as many nodes of each type as are in U. Here, a subset of T' is any contiguous set of infrastructure nodes and all of their connected non-infrastructure nodes. If there are N infrastructure nodes, then a subset would be defined as all infrastructure nodes k_1 to k_2, where $1 \leq k_1 \leq k_2 \leq N$, and the non-infrastructure nodes connected to them. For every subset in Q, a packing score is computed and then Q is reordered using the packing scores. Each subset is passed to $findMapping$ in order of best packing score first until either a valid mapping is found or there are no more subsets. The $findMapping$ method is unchanged from the version used in *MINBW*.

There are many possibilities for computing the packing score. The goal is to map the user graph onto as few infrastructure nodes as possible, thereby increasing the number

of resources available on each other infrastructure node. As mentioned above, this is related to minimizing bandwidth usage across infrastructure nodes, but the results can be different. Using fewer infrastructure nodes could actually increase the bandwidth usage between infrastructure nodes in the subset, but it does decrease the usage (to zero) between infrastructure nodes in the subset and those not in the subset. For this initial work, we use a simple packing score. First, if T_i' has exactly the same number of nodes of each type as U, then T_i' has the best packing score possible. In other words, if U could be packed on to T_i' such that T_i' has no unmapped nodes afterward, then that is the best possible packing. Otherwise, subsets with more unmapped nodes have a higher score than those with less unmapped nodes. That is, the score is simply the number of unmapped nodes in the subset. This simple scoring scheme treats nodes of different types equally, but it could easily be extended to give more or less weight to certain types depending on their relative availabilities, historical usage, etc.

Algorithm 2 MAXPACK(U, A, l, T, S)

```
P ⇐ computePossibleTimes(A, l, S)
for all (b, f) ∈ P do
  O ⇐ findOverlappingReservations(b, f, S)
  T' ⇐ T − O
  if enoughNodesAvailable(U, T') then
    Q ⇐ findValidSubsets(T')
    for all T'_i ∈ Q do
      Z_i ⇐ computePackingScore(U, T'_i)
    end for
    Q ⇐ reorderSubsetsByPackingScore(Q, Z)
    for all T'_i ∈ Q do
      M ⇐ findMapping(U, T'_i)
      if validMapping(M) then
        S ⇐ S ∪ (U, M, b, f)
        return TRUE
      end if
    end for
  end if
end for
return FALSE
```

V. Evaluation

Characterizing testbed scheduler performance is difficult due to the many parameters that can affect the results. In this section, we identify a small number of key parameters to study, and we provide a framework to generate a series of users requests that put a target average load on the scheduler. Results are given for response times to new requests and for the request rejection percentages of the two schedulers.

There are three broad categories of parameters to consider: the testbed topology, user graph topologies, and length and start time flexibility of requests.

The ONL testbed graph shown in Figure 2 is used for these evaluations since the schedulers being evaluated are for use in ONL. Indeed, *MAXPACK* is the current ONL scheduler. We do not vary anything about the base testbed topology, but we do parameterize the capacity of the infrastructure edges, denoted *IEC*. The capacity of these edges is often the limiting factor which leads to rejecting requests. All of the infrastructure nodes in ONL are 48 port, VLAN-capable Ethernet switches that are connected together with vendor-specific 12 Gb/s stacking connections, so *IEC* is usually set to 12. *IEC* values of 3, 6, 9, and ∞ are also used in the evaluation.

The various types available in ONL affect how user graphs are generated. In particular, there are two classes of nodes: backbone nodes and edge nodes. In Figure 2, the circles, triangles, and diamonds represent types that are used as backbone nodes, and the hexagons represent edge nodes. In the case of ONL, the circles and triangles are two types of programmable router, the diamonds are NetFPGAs [16], and the hexagons are PCs. There are enough PCs that the number of PCs is very rarely the limiting factor when considering a new request.

User graph topologies are generated using two parameters. The first parameter is the *backbone size*, *BBS*, which is the number of backbone nodes in the graph. The *BBS* backbone nodes are chosen uniformly at random from among all nodes in the testbed graph. This ensures that no user graph uses more of any type than are available in the testbed and balances the number of different types in proportion to the number available. The chosen backbone nodes are then connected randomly to form a tree. *BBS* values will vary from 2 to 8. The second parameter is the *average backbone degree*, *ABD*, which is used to determine the final shape of the user graph. Edges are added randomly between backbone nodes until the number of edges divided by the number of backbone nodes is at least *ABD*. *ABD* values will range from 0.0 to 2.5, where *ABD*=0.0 guarantees that no edges will be added beyond the initial tree. Finally, all unused network interfaces in the user graph are connected to edge nodes.

The next parameter is the user's start time *flexibility*, F. The flexibility is defined as the length of the start time range, A=(t_1, t_2), divided by the reservation length, l. That is, F= $(t_2$–$t_1)/l$, where t_1, t_2, and l are given in the same units. F can range from 0 to 3. Ideally, larger values of F will lead to lower rejection probabilities.

The final parameter is O, which is the order of request start times with respect to request arrivals. Of course, the scheduler operates in an on-line fashion and thus has no control over the request order, but it is certainly true than many scheduling problems have solutions that are greatly impacted by the ordering of input events. In the testbed scheduling context, requests are naturally in rough order of increasing start time. We explore three orderings: random, increasing start time, and decreasing start time.

All of these parameters can ultimately affect the performance of a scheduler, so a simple framework is needed to understand the contributions of the different parameters. We define the testbed scheduling *load*, L, to achieve this. Ideally,

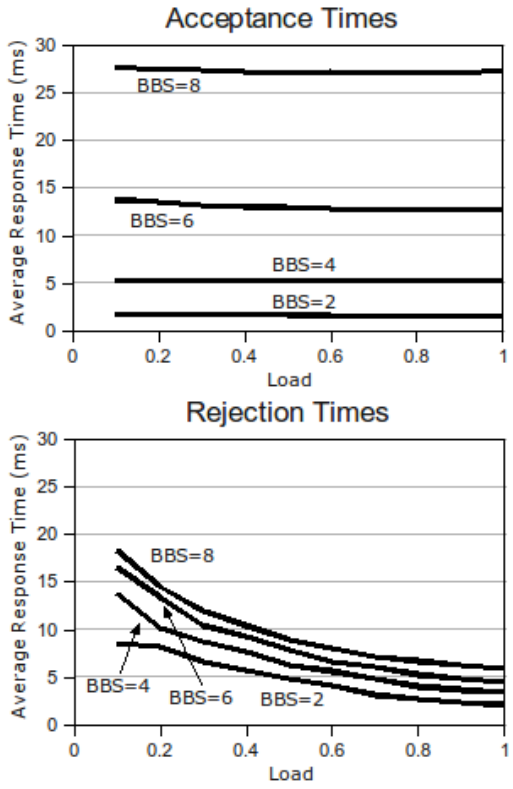

Fig. 3. *MAXPACK* response times for different backbone sizes.

the average load reflects the average percentage of resources used in the testbed. This is complicated by having different node types, as the percentage of each type currently in use could be substantially different, and different types could be the limiting factor at different times. Rather than use a definition of load that attempts to incorporate these nuances, we use a simple probabilistic model to generate a sequence of requests that has the desired load. Given requests with a particular value of *BBS*, average lengths of l, B total backbone nodes in the testbed graph, and a desired load of L, we compute the average time between request start times, τ, as $\tau = ((\text{BBS}/B) * l)/L$. That is, if the average reservation has length l and uses $BBS/B\%$ of the testbed resources, then a new reservation request should start every τ time units to achieve an average load of L. All of the following experiments use this structure to generate 10000 requests with a particular value of *BBS* and L. The average reservation duration has little impact because it is only used to compute τ such that the desired load is seen across all 10000 requests. The intervals between successive start times are randomly generated from a geometric distribution with mean τ.

Experiments were conducted that varied all of the parameters discussed above. Each experiment ran one of the schedulers with all of the parameters fixed. The scheduler processed all 10000 requests and the response times where recorded for each request. The rejection percentage for the experiment was also recorded. All of the results presented are for a load that varies from $L = 0.1$ to $L = 1.0$. Each chart shows the results for different values of one parameter, while the others parameters remained fixed. Unless otherwise noted, the following values are used as the default fixed values for each parameter: *IEC* $= 12$, *BBS* $= 2$, *ABD* $= 1.5$, $F = 0$, and $O =$ random.

A. Response Time

Figure 3 shows average response time results for *MAXPACK* as the backbone size is varied from 2 to 8. First, note the response times are all under 30 ms. This is typical over the entire evaluation. Moreover, the maximum response time over all experiments conducted for this paper was 453 ms. This occurred using *MINBW* with *BBS*$= 8$ and a load of 0.9. In that experiment, the average response time was 29 ms with a standard deviation of 19 ms. In a normal testbed usage scenario, this means that the maximum response time even including network transit times between the user and the testbed will be under one second. These results were generated using a quad-core processor running at 2.4GHz with 4GB of memory. All execution in the simulation is single-threaded.

The top chart on Figure 3 shows the average response time for accepted requests, and the bottom chart shows the same for rejected requests. There are two important trends to notice. First, the response time increases as the request size increases. The acceptance times suggest that the increase might be exponential. This is consistent with a MIP-based scheduler whose size (number of variables and number of constraints) increases as the user graph size increases. The second trend is a decrease in response times as the load increases, particularly for rejections. This follows because there are fewer resources left in the testbed graph when the load is higher. Fewer resources leads to fewer candidate times for the request, and thus the MIP is called fewer times. This is particularly true for *MAXPACK* where there are also fewer candidate subsets to try. The results for *MINBW* are not shown, but they are similar overall. The only notable difference is that the rejection times are lower (roughly half) because the MIP is called fewer times.

B. Rejection Rate

The second set of results concerns the rejection rate of requests. Two different rejection rates are used to reflect different aspects of the problem. The first is the *absolute rejection rate*, which is the percentage of requests rejected out of the 10000 requests in each experiment. The absolute rejection rate is useful in understanding the scheduler's overall performance from a user's perspective, i.e., how often their requests get rejected. The second rate is the *scheduler rejection rate*. Recall that the schedulers ensure that there are enough nodes of each type available before ever trying to find a mapping. Let R_n be the number of requests that do not proceed past this check for any of the candidate times. No scheduler can ever accept such a request because there simply not enough nodes of the correct types available. Let R_s be the number of requests rejected otherwise, i.e., those requests for which the scheduler attempts to find a mapping at least once. The scheduler rejection

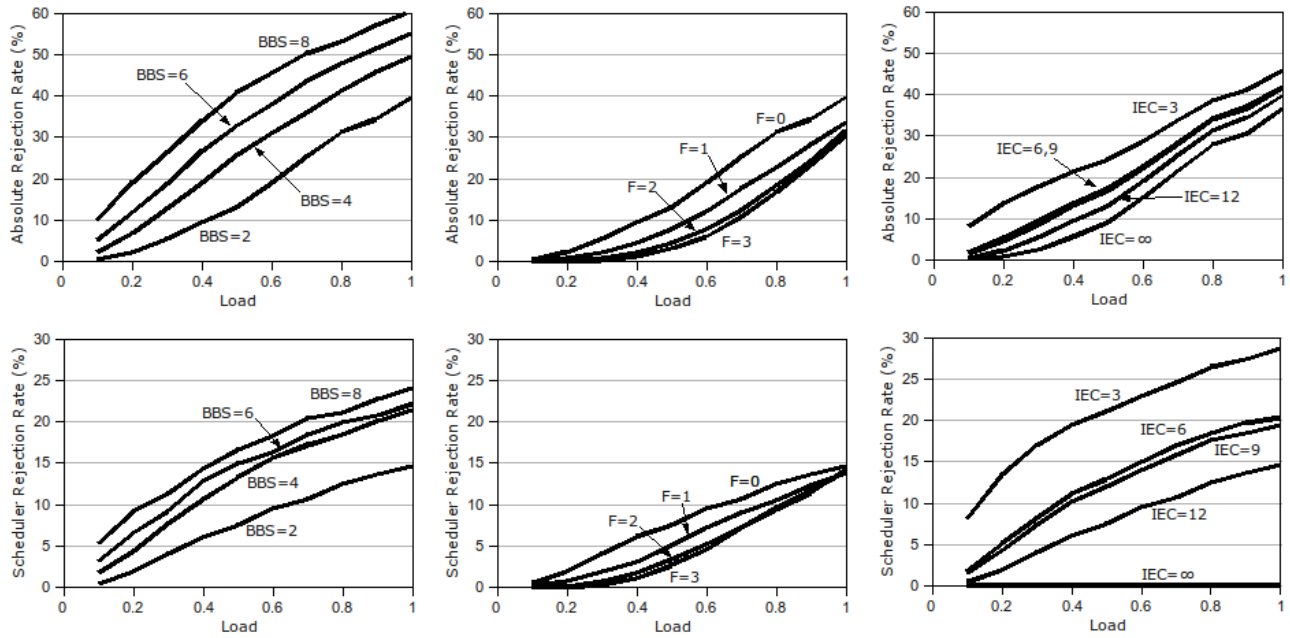

Fig. 4. Rejection rates for *MAXPACK*.

rate, then, is defined as $R_s/(10000 - R_n)$. Better schedulers will have smaller scheduler rejection rates, but even optimal schedulers may still have non-zero scheduler rejection rates if there is not enough capacity on the infrastructure edges.

Figure 4 shows both rejection rates for a set of *MAXPACK* experiments. The left column of charts shows the rejection rates where *BBS* is varied from 2 to 8, the center column shows rejection rates where F is varied from 0 to 3, and the right column shows rejection rates where *IEC* is varied from 3 to ∞. In each column, the top chart shows absolute rejection rates, and the bottom chart shows scheduler rejection rates. Experiments were also conducted that varied *ABD* and O but are omitted here. Different values of *ABD* have little impact on the rejection rates. Requests ordered by start time, increasing or decreasing, do have lower rejection rates than for random orderings, with a maximum absolute difference of around 10% under high load. Results were also gathered for *MINBW*, but the rates are all similar to within a few percent for every experiment.

Overall, varying the *BBS* parameter has the largest impact on absolute rejection rate. As the number of backbone nodes in each request increases, it becomes more likely that overlapping requests will overuse at least one of the node types. This is confirmed by observing that the differences in scheduler rejection rate for different values of *BBS* are much smaller than the differences in the absolute rate.

The scheduler rejection rate is affected most dramatically by varying the *IEC* parameter. First note that *IEC*$=\infty$ removes any bandwidth limitations from the testbed graph which results in a zero scheduler rejection rate, as expected. Across all loads, the scheduler rejection rate is much greater for smaller values of *IEC*. It is interesting to note the rates are very close for *IEC*$=6$ and *IEC*$=9$, but the gap increases as *IEC* decreases and as it increases away from that point. This suggests that a transition in behavior occurs somewhere between values of 6 and 9. If so, it is likely that this is a transition from being fundamentally constrained by the available infrastructure edge bandwidth to being constrained by the performance of a heuristic scheduler. The other possibility is that it is an artifact of the testbed topology that has the only 4 nodes of one type connected to the same infrastructure node. This was done because those 4 nodes are generally only used together in practice and not in combination with the other backbone node types.

As expected, the flexibility does decrease the rejection rates. The most benefit is gained near loads of 50% because that is where the scheduler is able to use the flexibility to fit the request in other available times. Most requests are accepted for lower loads regardless of the flexibility, and resource contention is too high for flexibility to help when the load is higher. Also note the quickly diminishing returns for flexibility beyond 2. In ONL, users can refer to charts that show how many nodes of each type are currently reserved for future times of up to two weeks in advance. This allows users to choose times when they more likely to have their request accepted.

VI. Discussion

Our MIP-based approach works reasonably well given the parameters explored in the previous section. More importantly, the *MAXPACK* scheduler has been used in ONL for the past several months and our experience has been positive. This is due in large part to the usage patterns seen in the testbed. During the summer months, the average load is small (less

than 30%), and most of that load is from research projects which tend to use larger, more diverse topologies. On the other hand, the average load is higher (closer to 50%) during the fall and spring semesters when ONL is used mostly by students in networking courses. Course topologies are, however, generally much smaller. As seen in the evaluation, the scheduler can handle many smaller topologies or a few larger topologies.

Scaling up to handle higher loads is not trivial. Increasing the testbed graph size does not impact our performance directly, but having more resources available allows users to build larger virtual networks. Rejection rates will be largely unaffected, assuming that user graphs increase proportionally to the testbed graph. Response times, however, could grow to be unacceptably large because the number of variables and constraints in the MIP increases linearly with the user graph size. If there are N infrastructure nodes in the testbed graph, L distinct node types in the user graph, $|V|$ nodes in the user graph, and $|E|$ edges in the user graph, then there are O($N|V| + (N-1)|E|$) variables and O($(N-1) + |V| + LN + (N-1)|E|$) constraints. Larger user graphs thus have proportionally larger MIP formulations, which leads to a potentially exponential increase in response time.

In the future, we plan to explore how our approach works with different testbed graph topologies. Modifying our formulation for star-based testbed topologies is relatively simple. Slightly more general tree-based topologies could potentially be supported as well. There is a fundamental limitation to our approach, however. Using indicator variables to store the mapping of user graph nodes to infrastructure nodes in the testbed allows us to easily compute the bandwidth along all the associated paths directly. This assumes that there is only one path between any two nodes in the testbed graph. If there are multiple possible paths, as in a ring topology, then the current MIP formulation has no way to choose between them. A different approach would be needed to support such topologies.

Another alternative to consider is to allow different network interfaces on a node to be connected to different infrastructure nodes. This could result in less bandwidth along infrastructure edges, but would require a new MIP formulation where each network interface on each user node is assigned to a particular infrastructure node. The trade-off is that this would substantially increase the number of variables in the MIP and thus could lead to higher response times.

It would also be interesting to explore how approaches such as ours could be used in other testbeds. For example, the GENI [17] initiative could result in a very large scale testbed with many diverse types of resources. Normal experiments in such a testbed are likely to be much more long-term, lasting days or weeks rather than a few hours. In that case, response times to a scheduling request could be allowed to increase as well, meaning that our approach might be viable there as well.

VII. Conclusion

Allowing users in emulation testbeds to reserve resources in advance is clearly a useful feature for anyone who has used an on-demand system and had to wait until enough resources became available to run their experiment. It does, however, substantially complicate the problem by adding a scheduling component to an already difficult mapping problem. We have presented a new class of heuristic, MIP-based schedulers that attempt to solve this problem. Our approach keeps response times low and accepts a large percentage of reasonable requests. One of these schedulers is already being used quite effectively in the ONL testbed.

References

[1] J. DeHart, F. Kuhns, J. Parwatikar, J. Turner, C. Wiseman, and K. Wong, "The open network laboratory," in *SIGCSE '06: Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2006, pp. 107–111.

[2] ONL, "Open network laboratory website. http://onl.wustl.edu." [Online]. Available: www.onl.wustl.edu

[3] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*. New York, NY, USA: ACM, 2002, pp. 255–270.

[4] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 65–81, 2003.

[5] R. Ricci, D. Oppenheimer, J. Lepreau, and A. Vahdat, "Lessons from resource allocators for large-scale multiuser testbeds," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 25–32, 2006.

[6] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Design, deployment, and use of the deter testbed," in *DETER: Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–1.

[7] WAIL, "Wisconsin advanced internet laboratory website. http://www.schooner.wail.wisc.edu/." [Online]. Available: http://www.schooner.wail.wisc.edu/

[8] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the internet," in *Proceedings of HotNets–I*, Princeton, New Jersey, October 2002.

[9] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Distributed resource discovery on planetlab with sword," in *WORLDS '04: Proceedings of the First Workshop on Real, Large Distributed Systems*, December 2004.

[10] J. Considine, J. W. Byers, and K. Meyer-Patel, "A constraint satisfaction approach to testbed embedding services," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 137–142, 2004.

[11] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," Washington University, Tech. Rep., June 2006.

[12] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proc. 25th IEEE International Conference on Computer Communications INFOCOM 2006*, Apr. 2006, pp. 1–12.

[13] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, 2008.

[14] J. Newsome and D. Song, "Gem: Graph embedding for routing and data-centric storage in sensor networks without geographic information," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2003, pp. 76–88.

[15] D. G. Andersen, "Theoretical approaches to node assignment," 2002. [Online]. Available: http://www.cs.cmu.edu/ dga/papers/andersen-assign-abstract.html

[16] J. Naous, G. Gibb, S. Bolouki, and N. McKeown, "Netfpga: reusable router architecture for experimental research," in *PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*. New York, NY, USA: ACM, 2008, pp. 1–7.

[17] GENI, "Global environment for network innovations website. http://www.geni.net." [Online]. Available: www.geni.net