PACE: Policy-Aware Application Cloud Embedding

Li Erran Li[⋆] Vahid Liaghat[◦] Hongze Zhao[∧] MohammadTaghi Hajiaghayi[◦‡]
Dan Li[∧] Gordon Wilfong[⋆] Y. Richard Yang[†] Chuanxiong Guo[⊕]
Bell Labs[⋆] Microsoft Research Asia[⊕] Tsinghua University[∧] Yale University[†] University of Maryland[◦]
AT&T Research[‡]

***Abstract*—The emergence of new capabilities such as virtualization and elastic (private or public) cloud computing infrastructures has made it possible to deploy multiple applications, on demand, on the same cloud infrastructure. A major challenge to achieve this possibility, however, is that modern applications are typically distributed, structured systems that include not only computational and storage entities, but also policy entities (e.g., load balancers, firewalls, intrusion prevention boxes). Deploying applications on a cloud infrastructure without the policy entities may introduce substantial policy violations and/or security holes. In this paper, we present *PACE*: the first systematic framework for Policy-Aware Application Cloud Embedding. We precisely define the policy-aware, cloud application embedding problem, study its complexity and introduce simple, efficient, online primal-dual algorithms to embed applications in cloud data centers. We conduct evaluations using data from a real, large campus network and a realistic data center topology to evaluate the feasibility and performance of PACE. We show that deployment in a cloud without considering in-network policies may lead to a large number of policy violations (*e.g.*, using tree routing as a way to enforce in-network policies may observe up to 91% policy violations). We also show that our embedding algorithms are very efficient by comparing with a good online fractional embedding algorithm.**

I. Introduction

Cloud data centers supporting virtualization and elastic resources offer many advantages over traditional approaches, where a dedicated infrastructure is constructed for each application [6]. For example, in cloud data centers, applications can be rapidly deployed, migrated, or scaled, on demand, on the cloud infrastructures, improving infrastructure efficiency.

A major challenge to the cloud data center approach, however, is that modern applications are not simple sets of virtual machines (VM). Rather, they are distributed, structured systems that also include policy entities such as load balancers, application accelerators, firewalls, and intrusion prevention boxes. Hence, designing cloud data centers without a strong support for policy entities may introduce substantial policy violations and/or security holes.

The importance of introducing policy support has been widely recognized in the industry. For example, VMWare introduces the capability of associating (software) policies with a VM. Recently VMWare also introduced a capability called Virtual Service Domain [7], which allows a group of virtual machines to be protected by a virtual appliance. All traffic entering or leaving the group of virtual machines will be sent to that particular virtual appliance for policy verification. Data center switches (*e.g.*, Voltaire Vantage 6024 switch [23]) have also started to introduce features to support port policy migration when a VM attached to the port migrates.

A limitation of the aforementioned approaches, however, is that they are limited to enforce policy only at the VM last hop. We call such policies *end-point policies*. On the other hand, for many networks, some policies are best enforced or can be enforced only in the network. For example, due to performance requirements, some policy middleboxes need hardware acceleration and thus are available only at certain network locations with the hardware; some desirable features are available only from certain systems that are available only at certain network locations; some policy boxes (*e.g.*, intrusion prevention boxes) perform better when they can observe traffic involving multiple endpoints. We refer to such policies as *application-wide, in-network policies*.

In this paper, we conduct the first systemic study of enforcing application-wide, in-network policies in cloud data centers. Our framework, called *PACE*, complements existing capabilities on end-point policies to build a complete solution framework for cloud application deployment.

We make several contributions.

- First, we precisely characterize application requirements in cloud data centers (Section II). We introduce concepts such as the flow security graph to rigorously capture the requirements.
- Second, we study the complexity of enforcing cloud application flow security graph during cloud application deployment (Section III). We refer to this problem as the *cloud application embedding problem*. We show that the complexity of the embedding problem depends on the enforcing mechanisms. In particular, we show that using traditional tree-based routing, the problem is NP-complete. This points out a serious challenge in using the traditional network infrastructure for supporting cloud application deployment.
- Third, using feasible mechanisms (*e.g.*, source routing, OpenFlow [19], or p-switches [15]), we present fast, effective, primal-dual algorithms for cloud application embedding (Section IV). Our algorithms consider policy and realistic constraints such as on bandwidth and reliability.
- Fourth, we study the policy entity placement problem (Section V). We give a negative result on the performance of application request agnostic placement (oblivious placement). We also give a placement algorithm based on the knowledge of application request distributions.
- Fifth, we conduct evaluations on the importance of policy-aware cloud application embedding as well as the performance of our algorithms (Section VI). Using real applications from a large real campus network and several realistic data center topologies, we show that policy-agnostic deployment may lead to a large number of policy violations (*e.g.*, using tree routing as a way to enforce in-network

978-1-4673-5946-7/13/$31.00 ©2013 IEEE

policies may observe up to 91% policy violations). We also demonstrate the effectiveness of our algorithms.

II. Encoding Cloud Application Policies

Many modern applications are designed as structured distributed systems to achieve scalability and security, based on concepts such as load balanacers and DMZ.

We encode the requirements of a given application i (denoted as $\texttt{App}_\texttt{i}$) with a tuple $(\texttt{FSG}_\texttt{i}, \texttt{Config}_\texttt{i})$. The first component is called the *flow security graph* (FSG), which encodes the logical entities and the information flow among the entities of the application. *An FSG is much richer than a virtual network*. Nodes can represent virtual machines, middleboxes, virtual routers, etc. An FSG is annotated with demands on computing resources, middleboxes, network bandwidth, and reliability, etc. The second component represents the configuration state of application entities. For example, for a firewall, the configuration state is the set of firewall rules; for a virtual machine, the configuration state consists of the set of allowed services, etc.

We now give more details on an FSG. We represent entities implementing an application as nodes in a graph. For example, in a two-tiered application, each tier or middlebox is represented as a node in the graph. In addition, Internet clients as a group are represented as a single node. Edges represent direct communication between nodes. Nodes and edges are annotated with labels that encode application requirements on computing resources, network bandwidth and reliability.

- Network bandwidth requirements: an application may require certain bandwidth between pairs of logical entities (e.g., between tier 1 and tier 2). Bandwidth requirements are encoded as edge demands on the flow security graph.
- Computing resource requirements: various logical entities may need computing resources. For example, in a two-tiered application, each tier requires a certain number of virtual machines. This is encoded as node demand.
- Reliability requirements: an application may want its logical entities be placed in more than one "fault domain". A fault domain is a unit that has correlated failures of a certain type. A machine, rack or "pod" can a fault domain. A customer may demand that its applications be placed in two fault domains. To encode reliability constraint, we annotate each logical entity with its required number of fault domains.

We represent the virtual machine demand and fault domain demand as a tuple. Any two nodes are connected by at least one path in FSG. If there are multiple paths and an application prefers a specific path, the application has to annotate the FSG with the chosen path.

As an example, the flow security graph of a two-tier application is shown in Figure 1. Tier 1 and 2 are represented as nodes u_1 and u_2 respectively. Middleboxes are firewall F_i, load balancer LB_i, intrusion prevention system IPS_i, $i = 1,2$. We do not need to represent the switches in our flow security graph. The tuple $(50,1)$ means that it needs 50 virtual machines and one fault domain. The Internet clients are denoted as u_e with zero demands on virtual machines and fault domains. The application needs one copy of each middlebox. Each middlebox requires one fault domain. There are no bandwidth demands on links since it is a best effort service.

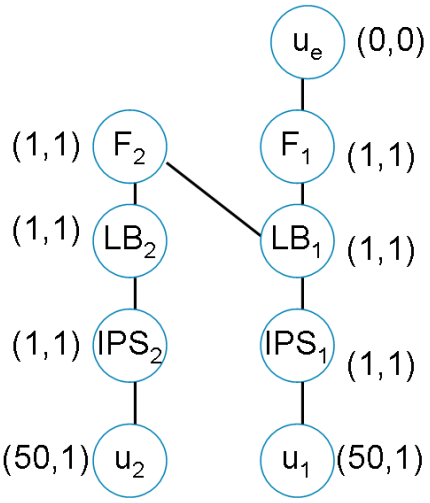

Fig. 1. Flow security graph of a two-tier application.

III. Feasibility of Enforcing Policies

We first conduct a feasibility study. We are given a cloud data center with servers (hosts several VMs) and middleboxes. The middleboxes can be hardware based or software based. Most hardware-based middleboxes are virtualized and can support multiple tenants. Their placements are fixed. Then the key feasibility issue for cloud provider is that given the policies of a cloud application represented by an FSG, decide the existence of a cloud data center routing to satisfy the FSG.

Our results are that the existence problem is NP-hard for tree routing. For Openflow [19], a solution may not exist. However, if one exists, we can decide in polynomial time as long as the number of middleboxes in each path segment is a constant. For pswitch [15] or source routing, a solution always exists. The key intuition here is that different flows have different policies. Constraining all of them to use the underlying tree-based routing leads to infeasibility.

To quantify the hardness of the problem, we look at a specific class of FSG where there is a set of application terminal nodes, each application terminal node requires a corresponding middlebox, and all of the middleboxes connect to a shared root. We assume general link layer network topology.

A. Policy enforcement mechanisms

Different applications do not share Scope. Thus, applications must be isolated. There is no sharing of forwarding and routing table, no communication between VMs of different applications, no sharing of middlbox instances (typically a physical middleboxes have many virtual instances). Besides isolation between applications, different components of an application must also be isolated. In our example application, tier 1 and tier 2 are on separate VLANs to enforce isolation.

To enforce FSG embedding, cloud providers need mechanisms to manipulate routing so that packets can go through middleboxes specified by the policy. We consider five types of link layer routing support: (1) Layer 2 routing uses traditional

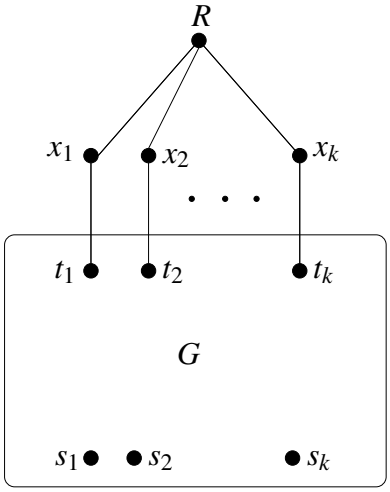

Fig. 2. NP-hardness construction.

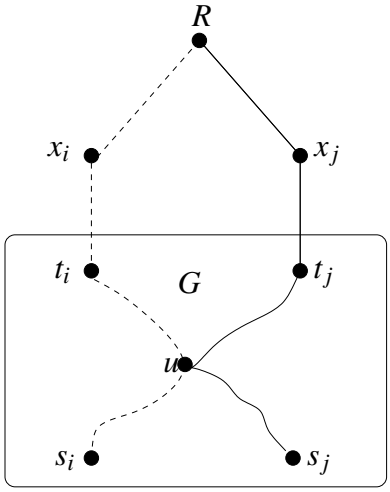

Fig. 3. A node other than R in both P_i and P_j implies a cycle.

spanning trees; (2) Recent layer 2 shortest path routing proposed by IEEE SPB or IETF TRILL; (3) Openflow based layer 2 routing; (4) source routing; and (5) pswitch-based routing where all middleboxes are placed off the network path. We will study a unified solution for (1) and (2). The case of Openflow, pswitch and source routing admits simple solutions because of their ability to control routing. We focus on a unified case of (1) and (2).

B. Feasibility using Tree Routing

We want the policy to be implemented by the spanning tree protocol. Thus, we want a tree. Our NP complete proof looks at a simpler problem. We are given a graph $G = (V,E)$, a distinguished root node R, a set of source nodes $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$ and a set of middleboxes $X = \{x_1, x_2, \ldots, x_k\} \subseteq V$. The problem is to determine if there is a path P_i between s_i and R such that $x_i \in P_i$ for each i, $1 \leq i \leq k$, where $\bigcup_{1 \leq i \leq k} P_i$ forms a tree. We call this problem the *k-paths-tree problem*. This problem encodes a special case of our FSG policy feasibility problem. R can represent the gateway to the Internet (clients need to go through the gateway to get web service from each application). Each source node s_i together with the middlebox x_i, and R forms an embedding of a FSG representing a web application.

Theorem 1: The k-paths-tree problem is NP-complete.

Proof: Note that checking if a set of paths $P_1, P_2, \ldots, P_k$ satisfies the given instance of the k-paths-tree problem can trivially be done in polynomial time. Hence the k-paths-tree problem is in NP.

To show that the k-paths-tree problem is NP-hard, we show a reduction from the k-node-disjoint-paths problem to it. An instance of the k-node-disjoint-paths problem consists of a graph $G = (V,E)$ and k disjoint pairs of nodes (s_i, t_i), $1 \leq i \leq k$. The problem is to decide if there exist k pairwise disjoint paths $P_1, P_2, \ldots, P_k$ where P_i is a path between s_i and t_i for $1 \leq i \leq k$ such that these k paths are pairwise node disjoint. This problem is known to be NP-complete [8].

Consider an instance *NDP* of the k-node-disjoint-paths problem given by the graph $G = (V,E)$ and pairs (s_i, t_i), $1 \leq i \leq k$. Then we construct an instance *PT* of the k-paths-tree problem as follows. Let $G' = (V', E')$ be the graph where $V' = V \cup \{x_1, x_2, \ldots, x_k, t_1, t_2, \ldots, t_k, R\}$ and $E' = E \cup \{t_i x_i : 1 \leq i \leq k\} \cup \{x_i R : 1 \leq i \leq k\}$. See Figure 2 for an illustration of the construction of G'. Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of source nodes and $\{x_1, x_2, \ldots, x_k\}$ be the set of middleboxes. Then we wish to determine if there are paths P_i between s_i and R so that $x_i \in P_i$ for $1 \leq i \leq k$ and where $\bigcup_{1 \leq i \leq k} P_i$ is a tree.

We claim that there is a solution to *NDP* if and only if there is a solution to *PT*. First suppose P_i, $1 \leq i \leq k$ is a solution for *NDP*. Clearly if we define T to be the graph consisting of paths P_i and edges $t_i x_i$ and $x_i R$ for $1 \leq i \leq k$,then T is a tree satisfying the requirement that the path from s_i to R contains x_i. That is, the paths in T from the source nodes to R satisfy *PT*.

Suppose T is a solution to *PT*. Then T is a tree and the path from s_i to R in T passes through x_i for $1 \leq i \leq k$. It can easily be checked that any path from s_i to R containing x_i must contain the edges $t_i x_i$ and $x_i R$. Let P_i be the path in T from s_i to t_i, $1 \leq i \leq k$. We claim that these paths are pairwise disjoint. To show this, consider two such paths P_i and P_j. Suppose by way of contradiction that P_i and P_j are not node disjoint. (See Figure 3.) Let u be the node in $P_i \cap P_j$ such that no other node in the subpath of P_i between u and t_i is in P_j. Then the union of the edges in P_i from u to R with the edges in P_j from u to R, form a cycle in T. But this contradicts the assumption that T is a tree. ■

One may wonder if we limit the class of network topology, the problem may become easier. For example, one class of topologies consists of planar graphs. However, by the construction given in the preceding proof and the fact that k-node-disjoint-paths problem remains NP-complete in planar graphs, we conclude that the k-paths-tree problem also remains NP-complete even when restricted to planar graphs [17].

Another class of network topologies is that of fat trees. Given the increasing trend to adopt fat tree as a topology for data centers, we study feasibility in such a network. We have the following results.

Theorem 2: Even if graph $G = (V,E)$ is a fat tree, the k-paths-tree problem is NP-complete.

Proof: See appendix. ■

C. Feasibility using Openflow

The only restriction in this case is that the path segment has to be a simple path (*i.e.*, it does not contain a cycle). If one exists, we can decide in polynomial time as long as the number of middleboxes in each path segment is a constant.

D. Feasibility using pswitch or source routing

A pswitch can perform hop-by-hop routing and forward traffic matching policy to a middlebox directly. Thus, a solution always exists. The case for source routing is similar as long as middleboxes can be addressed in layer 3.

IV. Deploying enterprise applications in public data centers

Enterprises use a management interface similar to, but richer than Amazon EC2 to submit their application deployment requests (each represented as a flow security graph). There are three steps for the cloud provider to deploy enterprise applications in public data centers. First, the cloud provider needs to realize the flow security graph in the underlying data center network. We refer to this step as flow security graph embedding. The second step for the cloud provider is to configure the entities in the flow security graph. The third step for the cloud provider is policy enforcement. The second step is straightforward and application specific. We have considered policy enforcement mechanisms and feasibility of policy enforcement in Section III. We now only consider FSG embedding. We assume mechanisms such as Openflow, policy switch or source routing are available to enforce specific policies on flows of an application. To reduce routing complexity, and leverage underlying data center routing protocol, we assume that neighbors in a flow security graph are connected by the native routing path in the data center.

A. Flow security graph embedding

For ease of the description of the algorithm, we consider a few simplifying assumptions; all the results carry to the general case. We assume the flow security graph of each application request is a simple path and the fault domain constraint is one, i.e., all resources of an entity must be placed together. We also assume there is one type of middlebox. Application requests must be processed as they arrive in real time. Thus, we need an online algorithm.

1) Problem Formulation: An instance of the *offline FSG embedding problem* is the tuple (G,C,n,R) in which

- The weighted graph $G=(V,E,B)$ represents the network topology where $B:E\rightarrow\mathbb{N}$ shows the bandwidth of the edges;
- The function $C:V\rightarrow\mathbb{N}$ shows the number of compute nodes (i.e. VMs) on each vertex; [1]
- The integer n is the number of available middleboxes to be placed on vertices; and
- R is the sequence of arriving *requests* which are to be allocated on the topology.

Each request r is associated with a *prize* π_r and a *path structure* q_r. The prize of a request shows the benefit of allocating that request which may encode the priority, importance, size, or other considerations. A path structure describes the demands of the request.

An *extended path* of length k in the graph is an ordered sequence of k vertices with simple paths connecting them. Therefore an extended path would be in the form of

$$p=<v_1,p_1,v_2,\ldots,p_{k-1},v_k>$$

where p_i is a path between vertices v_i and v_{i+1}. We note that these paths may be empty, i.e., v_i's are not necessary different. We allocate the requests to the extended paths of the topology with enough resources. Thus we can define the path structure of a request as a sequence of the form

$$q_r=<(r_1,t_1),b_1,(r_2,t_2),b_2,\ldots,b_{k-1},(r_k,t_k)>$$

where k is a small constant [14]; $r_i\in\mathbb{N}$ is the number of resources needed on the i^{th} vertex; t_i is the type of the resource needed on the i^{th} vertex; and b_i is the bandwidth of the connection needed between vertex i and $i+1$. As mentioned before, we assume that t_i may have only two options: either 'middlebox' or 'compute node'. To *allocate* a request on the topology, we have to choose a *valid* extended path. An extended path p is *valid* for the path structure q_r if for $1\leq i\leq k$, the vertex v_i has r_i amounts of either middleboxes (if $t_i=$ *'middlebox'*) or compute nodes (if $t_i=$ *'computenode'*) available and for $1\leq i\leq k-1$ all the edges in the path p_i have b_i amounts of bandwidth available. By allocating r to p, these amounts of resources would be allocated to the request and thus will not be available anymore. We may denote the valid extended paths as *candidate paths*.

Given an instance (G,C,n,R), the algorithm has two phases. In the first phase, the algorithm should define a function $F:V\rightarrow\mathbb{N}$ which shows the number of middleboxes to be placed on each vertex. We note that the total number of middleboxes is n, i.e., $\sum_{v\in V}F(v)=n$. In the second phase, the algorithm should choose a subset of the requests and allocate them to the candidate paths. An *allocation scheme* is the selection of the subset of the requests to be allocated with their corresponding allocated candidate paths. The *payoff* of an allocation scheme is the total prize of the allocated requests.

The *online FSG embedding problem* is an online version of the offline FSG embedding problem, in the sense that the sequence of requests is revealed to the algorithm in an online manner, however we know the number of requests in advance (which might be large). The algorithm should decide about the placement of the middleboxes (i.e. $F(v)$) before receiving the requests. Upon receiving a request the algorithm should either allocate a candidate path to the request or discard it right away; this decision is not revocable. The goal is to maximize the total prize of the allocated requests.

We can formulate the offline version as the following integer programming optimization (IP) problem. Let P_r be the set of all possible candidate paths for request r in the topology. We note that since the size of a request is constant and neighbors in a FSG are connected by the native routing path in G, then the number of possible candidate paths is polynomial. This might still be a large number. However, we typically want nodes (VMs or middleboxes) in an application to be

[1] In a fat tree, the compute nodes are only on the leaves.

allocated close together, e.g. within a fault domain (if the FSG of an application has no requirements for more than one fault domain). This can drastically reduce the number of candidate paths to a manageable level. The LP variable $\mathbf{y}_{rp}$ indicates the situation where the candidate path $p \in P_r$ is allocated to the request $r \in R$. The values c_{rpv}, f_{rpv}, b_{rpe} are independent of LP variables, showing the required (i) computer nodes on vertex v, (ii) middleboxes on vertex v and (iii) bandwidth on edge e, if the path p was to be allocated to the request r.

$$\text{Maximize.} \quad \sum_r \sum_p \pi_r \mathbf{y}_{rp} \qquad \text{(FSG)}$$

$$\forall r \in R: \quad \sum_p \mathbf{y}_{rp} \le 1 \qquad \text{(FSG.A)}$$

$$\forall v \in V: \quad \sum_r \sum_p \mathbf{y}_{rp} c_{rpv} \le C(v) \qquad \text{(FSG.B)}$$

$$\forall v \in V: \quad \sum_r \sum_p \mathbf{y}_{rp} f_{rpv} \le \mathbf{F}(v) \qquad \text{(FSG.C)}$$

$$\forall e \in E: \quad \sum_r \sum_p \mathbf{y}_{rp} b_{rpe} \le B(e) \qquad \text{(FSG.D)}$$

$$\sum_v \mathbf{F}(v) \le n \qquad \text{(FSG.E)}$$

$$\mathbf{y}_{rp} \in \{0,1\}$$

$$\mathbf{F}(v) \in \mathbb{N}$$

In the maximization LP, the set of constraints

- FSG.A insure that each request can be assigned at most once;
- FSG.B insure that the number of required compute nodes on each vertex cannot exceed the number of available compute nodes on that vertex;
- FSG.C insure that the number of required middleboxes on each vertex cannot exceed the number of available middleboxes on that vertex;
- FSG.D insure that the allocated requests cannot exceed the bandwidth limit on each edge.

In the FSG embedding problem we should decide about the placement of middleboxes before receiving the requests. For any placement there could be a sequence of requests where none of the requests can be allocated but the same sequence can be allocated in a different placement. Thus we have to decide about the placement of middleboxes by finding the best possible placement for a set of samples for the requests. In the rest of this section we assume that we know the placement in advance and try to give an online allocation scheme with a near optimum competitive ratio. We call this problem *FSG Path Allocation Problem*.

2) FSG Path Allocation Problem: If we re-write the FSG linear programming using a specific middlebox placement, then all our constraints would be in the form of a packing constraint[2]. Formally, we can consider the following relaxed LP for the offline problem:

$$\text{Maximize.} \quad \sum_r \sum_p \pi_r \mathbf{y}_{rp}$$

$$\forall i \in [|R|+|V|+|V|+|E|]: \sum_r \sum_p \mathbf{y}_{rp} q(i,r,p) \le u(i) \quad \text{(FSG.A,B,C,D)}$$

$$\mathbf{y}_{rp} \ge 0$$

[2]A packing constraint denotes a linear combination of variables with positive coefficients which has to be be smaller than a positive constant.

where all the previous constraints are re-written in a general form. Indeed one can simplify the LP even more. We write a new LP by replacing $\mathbf{y}_{rp}$ with $\frac{\mathbf{z}_{rp}}{\pi_r}$ and $q(i,r,p)$ with $\pi_r a(i,r,p)$. Thus we get the following LP and its dual. One can easily verify that there is a 1-to-1 relation between the feasible solutions of the two LPs which preserves the objective value. Therefore any α-competitive algorithm which uses the modified LP is also an α-competitive algorithm for our problem.

$$\text{Maximize.} \quad \sum_r \sum_p \mathbf{z}_{rp}$$

$$\forall i \in [|R|+|V|+|V|+|E|]: \sum_r \sum_p \mathbf{z}_{rp} a(i,r,p) \le u(i) \quad \text{(FSG.A,B,C,D)}$$

$$\mathbf{z}_{rp} \ge 0$$

$$\text{Minimize.} \quad \sum_i u(i) \mathbf{x}_i$$

$$\forall r \in R \text{ and } p \in P_r: \sum_i a(i,r,p) \mathbf{x}_i \ge 1$$

$$\mathbf{x}_i \ge 0$$

In the online version of the FSG Path Allocation Problem, we have the same LP but the variables (along with their set of coefficients) are revealed to us in an online fashion. As usual, the performance of an online algorithm on a given input is defined to be the ratio between the maximum (offline) payoff of any solution (i.e., allocation scheme) and the payoff of the solution given by the algorithm. The maximum ratio, taken over all input sequences, is defined to be the competitive ratio of the algorithm. If we accept any real value for the variables, then we get the *general online fractional covering problem*. Buchbinder and Naor [4] give an algorithm that gets the desired competitive ratio $B > 0$ and produces a solution that does not violate the ith packing constraint by more than a factor of $\frac{2\lg\left(1+\frac{Na_i(max)}{a_i(min)}\right)}{B}$ where N is the total number of constraints and $a_i(max)$ and $a_i(min)$ are the maximum and minimum (non-zero) coefficients of the constraint. They also gave a tight lower bounds on any online algorithm for the problem, proving that their scheme is optimal for any $B > 0$.

Using the algorithm given in [4] with $B = 2\lg(1+NT_{max})$ we can give a $O(\lg(|R|+|V|+|E|)+\lg(T_{max}))$-competitive algorithm which do not violate any constraint; where $N = |R|+2|V|+|E|$ and $T_{max} = \max\{\frac{c_{max}}{c_{min}}, \frac{f_{max}}{f_{min}}, \frac{b_{max}}{b_{min}}\}$, i.e., the maximum of required compute nodes ratio, required middlebox ratio and required bandwidth ratio. Since $\lg(T_{max})$ can be considered a small constant in the context of this paper, the algorithm gives a $O(\lg(|R|+|V|+|E|))$-competitive *fractional* solution for the online path allocation problem. For the sake of completeness we present the algorithm of [4]. We initialize all the variables $\mathbf{x}_i$ and $\mathbf{z}_{rp}$ to zero. Upon receiving the request r, we will define a variable $\mathbf{z}_{rp}$ for every possible path p (or we can improve the performance by using a heuristic algorithm to generate only a set of suitable paths for the request). For each path p we compute the coefficients of $\mathbf{z}_{rp}$ in the constraints and run Algorithm 1 given in the figure.

Theorem 1 in [4] proves that this algorithm gives a B-competitive fractional solution for the problem. However, we

Algorithm 1 Online Fractional FSG Path Allocation Problem

$\mathbf{z}_{rp} \leftarrow 0$.
For each $\mathbf{x}(i)$: $a_i(max) \leftarrow \max_{(r',p')\in R\times P} a(i,r',p')$.
while $\sum_i a(i,r,p)\mathbf{x}(i) < 1$ **do**
  Increase $\mathbf{z}_{rp}$ continuously.
  Increase each variable $\mathbf{x}(i)$ by the following increment function:
  $\mathbf{x}(i) \leftarrow \max\{\mathbf{x}(i), \frac{1}{Na_i(max)}\left[\exp\left(\frac{B}{2u(i)}\sum_{(r',p')\in R\times P} a(i,r',p')\mathbf{z}_{r'p'}\right)-1\right]\}$

should either discard a request or allocate the whole request to one path and thus we need an integral solution. We can show that no online integral algorithm can always guarantee $o(|R|)$-competitiveness. Therefore we have to rely on the competitive ratio of the fractional solution and use a randomized rounding method to obtain the integral solution based on the fractional solution. First we compute the vector $\mathbf{y}$ corresponding to the current solution $\mathbf{z}$, i.e., for all $r \in R$ and $p \in P_r$ we set $\mathbf{y}_{rp} = \frac{\mathbf{z}_{rp}}{w_{rp}}$. For the request r, consider $p_1,\ldots,p_m$ as an ordering of the possible paths. Since $\sum_i \mathbf{y}_{rp_i}$ is always smaller than or equal to one, we can consider the values of $\mathbf{y}_{rp_i}$ for $1 \le i \le m$ as the distribution of request r over all the paths. Thus we like to choose the path p_i with probability $\mathbf{y}_{rp_i}$ and discard r with probability $1-\sum_i \mathbf{y}_{rp_i}$. We pick a random number $x \in [0,1)$ and we choose path p_k for r, iff $\sum_{i=1}^{k-1}\mathbf{y}_{rp_i} \le x < \sum_{i=1}^{k}\mathbf{y}_{rp_i}$; or we discard r if $x \ge \sum_{i=1}^{m}\mathbf{y}_{rp_i}$. If a path was chosen, we assign it to r unless this assignment together with previous integral decisions violate some of the constraints, which in that case we discard the request r. We will set the variables according to this decision and continue to the next request.

V. Middlebox Placement

Most recent data center architecture designs [11], [10], [20] have ignored middleboxes. One exception is the recent Cisco multi-tenant data center architecture [5] which advocates placement at the aggregation layer in a three-layer topology. Due to cost, middleboxes can not be attached to every switch. The question is how should they be placed to maximize data center performance (ability to host maximal applications requiring in-network policy enforcement)?

Insights on placement can shape the design of future policy-aware data center network architecture. Ideally, we would like to address this question independent of specific application request sequences. However, we show that there is no good guarantees for oblivious middlebox placement problem. Therefore, we are obliged to consider request distributions in our solution.

A. Negative results on request oblivious placement

One can show that similar to many other online problems[3] no online algorithm can always guarantee a $o(|R|)$-competitive solution to the Online (Integral) FSG Path Allocation Problem when adversary has full control over the input.

Theorem 3: There is no $o(|R|)$-competitive algorithm for the Online (Integral) Path Allocation Problem.

[3]For example the famous secretary problem

Proof. Consider an online algorithm and a simple path topology with $M > |R|$ compute nodes at its vertices. The adversary will give a request which can be satisfied but will use all the resources (i.e. the first request is just the topology itself). If the algorithm does not assign the first request, then the adversary will send only impossible requests as the remaining $|R|-1$ requests and thus the algorithm will not satisfy any of the requests. However if the algorithm assigns the first request, then the adversary will send $|R|-1$ small requests which they only need one compute node. Thus the competitive ratio in this scenario would be $|R|-1$. This shows that no online algorithm can be $o(|R|)$-competitive. □

B. Placement algorithm

Assume that we have a topology $< G,C,n >$ and a set of sample sequences of requests $R_1,\ldots,R_K$. Solving the relaxation of FSG for a set of requests R_i would give us a fractional solution for middleboxes positions, i.e. the function F_i. The function F_i gives us a distribution π_i of n middleboxes over the set of vertices, i.e., $\pi_i(v) = F_i(v)/n$. However, π_i is only optimal for one set of requests, but we need a placement that could give a maximum average solution for different scenarios. Therefore we combine all LP-relaxations of different samples, i.e., we duplicate the topology for any sample sequence, but will use the same $F(v)$ set of variables for all of these topologies. The LP-relaxation is given below.

$$\text{Maximize.} \quad \sum_{k\in[K]}\sum_{r\in R_k}\sum_{p\in P_k}\mathbf{y}_{rp} \tag{OPL}$$

$$\forall k\in[K]\ \&\ \forall r\in R_k: \quad \sum_{p\in P_k}\mathbf{y}_{rp} \le 1 \tag{OPL.A}$$

$$\forall k\in[K]\ \&\ \forall v\in V_k: \quad \sum_{r\in R_k}\sum_{p\in P_k}\mathbf{y}_{rp}c_{rpv} \le C(v) \tag{OPL.B}$$

$$\forall k\in[K]\ \&\ \forall v\in V_k: \quad \sum_{r\in R_k}\sum_{p\in P_k}\mathbf{y}_{rp}f_{rpv} \le \mathbf{F}(v) \tag{OPL.C}$$

$$\forall k\in[K]\ \&\ \forall e\in E_k: \quad \sum_{r\in R_k}\sum_{p\in P_k}\mathbf{y}_{rp}b_{rpe} \le B(e) \tag{OPL.D}$$

$$\forall k\in[K]: \quad \sum_{v\in V_k}\mathbf{F}(v) \le n \tag{OPL.E}$$

$$\mathbf{y}_{rp} \in \{0,1\}$$
$$\mathbf{F}(v) \in \mathbb{N}$$

By solving the LP-relaxation for all samples together, we can get the distribution Γ over the vertices which gives the optimum average solution for all the samples. Finally, we assign $\lfloor\Gamma(v)n\rfloor$ middleboxes to every vertex v and assign the remaining middleboxes one by one to the vertices with highest $\Gamma(v)$. After finding the proper placement, we run the online algorithm in Section IV-A2 to give a solution for the online stream of requests.

VI. Evaluation

When enterprises deploy their applications in remote public data centers, policies can be enforced either in remote data centers or back in the base enterprise networks (base networks for short). We first evaluate the cost and the feasibility of enforcing policy constraints in base networks. We show that it may not be feasible if the enterprise can only use tree-based

routing to enforce policies. It also incurs substantial cost in terms of path length.

We then consider enforcing policies locally in remote data center. We evaluate our embedding algorithm, and impacts of different middlebox placements.

A. Methodology

An enterprise network topology and application policies: We obtain router, middlebox and switch configuration files of a campus network with more than 50 routers and more than 1000 switches.

We first extract the route distribution graph, and layer-three topology using a tool in [2]. We insert the link-layer topology into the layer-three topology due to the fact that switch configurations are not adequate.

We then infer the middlebox traversal policy based on the topology properties and the route distribution graph. We examine the possible paths between two endpoints (represented as two subnets or two VLANs). From the path, we determine the middleboxes traversed and store this sequence as the waypoints for this particular path.

For scope, if both endpoints are in the same VLAN, then the scope is all nodes in the broadcast domain. If they are not in the same VLAN, we use all reachable nodes (based on route distribution graph and ACLs) in the security zone as the scope. We compare the tree routing case and the pswitch case. We pick 7 layer-two networks as the application network topologies to migrate to a remote data center. We assume the remote data center has no middleboxes. We evaluate the two mechanisms: tree routing and pswitch. We leave the evaluation of Openflow mechanism to future work.

Policy-aware data center network topology: We use simulation to study the performance of our embedding algorithms. We use three typical data center structures Bcube [11], Dcell [13], and fat-tree [1]. We use $Bcube_3$ which has 2187 leaf nodes and 4 levels of 3-port mini-switches. Each leaf node has 300 VMs. Each switch has 150 middlebox instances. Each link has a bandwidth of 1Gbps. We use $Dcell(32,1)$ which has 1056 leaf nodes. Each leaf node has 500 VMs, and each switch has 1000 middlebox instances, each link has bandwidth 1Gbps. We use fat-tree with $k = 24$. This gives us 3456 leaf nodes. Each link node has 400 VMs. We use the Google cluster dataset [9] for request size distribution. This dataset gives a normalized job size distribution extracted from Google product workloads. The distribution shows more than 51% jobs are the smallest. 20% middle sized jobs use 65% of the total resources. This distribution is used for both the VM demand and middlebox demand of application embedding requests. The average number of VMs and middleboxes per request is 50 and 60 respectively. The average bandwidth demand per request is 350Mbps which is generated from a Poisson distribution. We use Amazon EC2 pricing schemes for VMs ($0.048/h per VM) and bandwidth (0.01$/GB). Middleboxes are assumed to be 4 times more expensive than VMs.

B. Results on enforcing policy in base networks

Feasibility: We first study the feasibility of satisfying in-network policies. Figure 4 shows the results of policy violation using tree routing. We see that tree routing may not always find a feasible solution. In 5 out of 7 applications, tree routing fails to find a feasible solution. The number of policy violations (varies from 0 to 91%) depends on the set of policies associated with the L2 networks. Using pswitch we find feasible solutions in all cases.

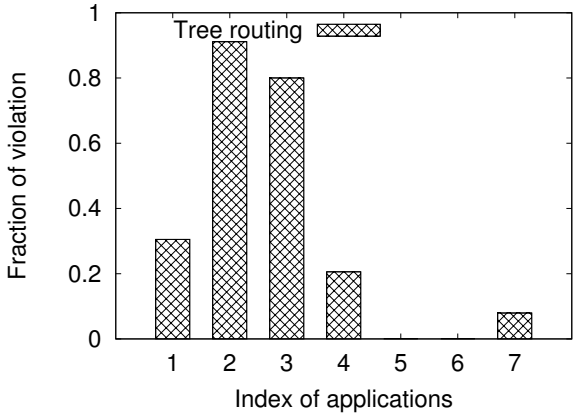

Fig. 4. Fraction of paths with policy violations in the case of tree-routing.

Costs: In the next experiment, we evaluate the cost of enforcing policies by considering the average path length for communication between points in relocated layer-two networks and other endpoints in the network. We only look at source-destination pairs whose policies are satisfied. The path length is defined as the number of network devices a packet must traverse from source to destination (*i.e.*, network hops).

Figure 5 shows the trade-off of the two mechanisms in terms of path length. In the cases (applications indexed 5 and 6) that tree routing is feasible, both mechanisms have the same average path length and so both cases use simple path (no cycles). In the cases (applications indexed 2 and 3) that tree routing has the largest percentage of violations (91% and 80%), the path lengths differ the most. That is, the path length of the pswitch case is 7.6 and 6.5 hops longer than that of the tree routing case. This is due to the fact that the algorithm for the pswitch case finds non-simple paths to satisfy policies. This is not possible in the tree routing case. For the intermediate cases (applications indexed 1 and 4) where policy violation percentage is 31% and 21%, the path length of the pswitch case is only 2.4 and 1.5 hops longer than that of the tree routing case. This means that most of the policies can be satisfied by simple paths.

C. Results on application embedding and middlebox placements

Topology	Accepted Requests	BW Usage	VM Usage	Middlebox Usage	First Reject	Revenue($)
FatT I	5562	41.1%	38.2%	70.2%	146	92837
FatT II	5442	38.8%	37.4%	68.2%	179	84727
FatT III	5412	43.9%	38.1%	66.1%	201	83019
Bcube	5910	37.1%	58.1%	59.9%	229	93475
DCell	4329	46.1%	51.3%	53.1%	148	75738

TABLE I
PERFORMANCE OF FRACTIONAL FSG EMBEDDING ALGORITHM UNDER DIFFERENT PLACEMENT STRATEGY

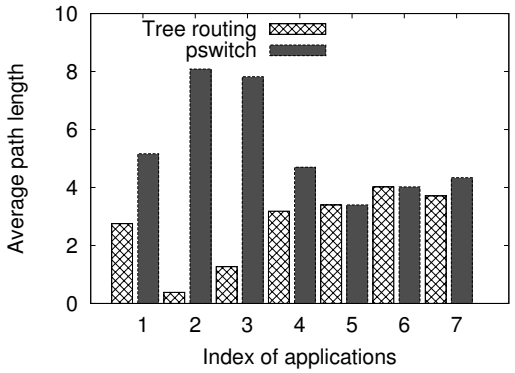

Fig. 5. A comparison of the average path length for tree routing and pswitch.

Topology	Accepted Requests	BW Usage	VM Usage	Middlebox Usage	First Reject	Revenue($)
FatT I	5281	38.8%	33.1%	67.2%	184	77261
FatT II	4991	38.3%	33.2%	65.9%	204	73332
FatT III	4865	49.1%	36.2%	66.4%	155	72918
Bcube	5113	37.2%	79.6%	58.6%	223	82616
DCell	3661	56.9%	47.2%	50.2%	190	61294

TABLE II
PERFORMANCE OF INTEGRAL FSG EMBEDDING ALGORITHM UNDER DIFFERENT PLACEMENT STRATEGIES.

We investigate three middlebox placement strategies for fat-tree: (1) FatT I: all are attached to aggregation switches, each has 3000 instances; (2) FatT II: half attached to aggregation switches, half attached to access switches, each switch has 1500 middlebox instances; (2) FatT III: all are attached to access switches, each has 3000 middlebox instances. For Bcube and Dcube, we place middleboxes uniformly, each switch has 150, and 1000 middlebox instances respectively.

We compare our fractional solution with our integral solution. Our performance metrics are total accepted number of application requests overall and total accepted requests before the first rejection, the resource utilization in terms of VMs, middleboxes, and network bandwidth. Utilization is defined as the total usage by all admitted requests divided by the total resource. Our results are shown in Table I and II. As expected, the fractional solution typically performs better in terms of total accepted requests and total revenue. However, the difference in terms of total accepted requests are smaller than 16% in all cases. This shows that our integral solution is very close to optimal.

In terms of impact of middlebox placement strategies, we see that placing at the aggregation switches performs the best. The reason is that many requests can not be localized to a single rack, if VMs of one rack have to go to anther rack's access switch for middlebox service, then the path is much longer than just go to a middlebox in the aggregation switch.

VII. Related work

With the exception of [16], previous studies on policy enforcement in public data centers or enterprise networks have focused on end-point policies [7], [23] and access control policies [14], [21], [24], [22], [3], [18], [2]. In [16], we have focused on policy preserving network extension. It does not consider application request embedding. Previous work on application embedding in public data centers [12] does not consider policy enforcement.

VIII. Conclusion and Future Work

We conduct the first study on satisfying application-wide, in-network policies, and other realistic requirements such as bandwidth and reliability. we precisely encode application requirements using flow security graph, middlebox configuration states and policy specification. We characterize feasibility of policy enforcement based on enforcement mechanisms. We then propose an effective online algorithm to map enterprise application onto public data center topology. Our study motivates the need of flexible policy enforcement mechanisms such as Openflow and pswitch, and the design of policy-aware data center network architecture.

There are many avenues for future work. We would like to consider designing policy-aware data center network architecture. We are also planning to conduct experiments in a cloud computing testbed which connects a corporate data center with Amazon Virtual Private Cloud.

IX. Acknowledgement

H. Zhao and D. Li are partly supported by 973 Program of China under Grants 2012CB315803, and NSFC program under Grants No. 61170291 and 61161140454. We appreciate the help of A. Voellmy, M.F. Nowlan, and R. Beebee.

References

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of ACM SIGCOMM*, pages 63–74, New York, NY, USA, 2008. ACM.

[2] Theophilus Benson, Aditya Akella, and David Maltz. Mining policies from enterprise network configuration. In *IMC*, 2009.

[3] Theophilus Benson, Aditya Akella, and David Maltz. Unraveling the complexity of network management. In *Proceedings of USENIX NSDI*, pages 335–348, Berkeley, CA, USA, 2009. USENIX Association.

[4] N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.

[5] Cisco. Cisco virtualized multi-tenant data center, version 2.0 compact pod design guide. http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/VMDC/2.0/design_guide/vmdcCPoDDesign20.pdf.

[6] Cisco Inc. Data center interconnect: Layer 2 extension between remote data centers.

[7] Cisco Inc and Vmware Inc. Virtual networking features of the vmware vnetwork distributed switch and cisco nexus 1000v switches. available at http://www.vmware.com/files/pdf/technology/cisco_vmware_virtualizing_the_datacenter.pdf, 2009.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, CA, 1979.

[9] Google. Google cluster data. http://code.google.com/p/googleclusterdata/.

[10] Albert Greenberg, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, Dave Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of ACM SIGCOMM*, August 2009.

[11] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of ACM SIGCOMM*, pages 63–74, New York, NY, USA, 2009. ACM.

[12] Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of Co-NEXT*, pages 15:1–15:12, New York, NY, USA, 2010. ACM.

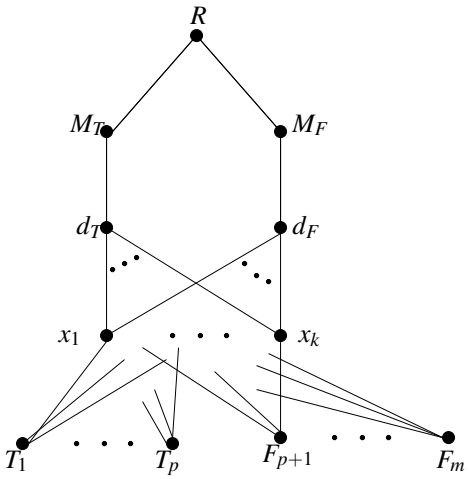

Fig. 6. Construction for 3-SAT reduction.

[13] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of ACM SIGCOMM*, pages 75–86, New York, NY, USA, 2008. ACM.

[14] Mohammad Hajjat, Xin Sun, Yu-Wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai, and Mohit Tawarmalani. Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud. In *Proceedings of ACM SIGCOMM*, 2010.

[15] D. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. *CCR*, 2008.

[16] Li Erran Li, Michael F. Nowlan, Yang Richard Yang, and Ming Zhang. Mosaic: Policy homomorphic network extension. In *Proceedings of the ACM Large-Scale Distributed Systems and Middleware (LADIS)*, 2010.

[17] J. F. Lynch. The equivalence of theorem proving and the interconnection problem. *ACM SIGDA Newsletter*, 5(3):31–36, September 1975.

[18] Ankur Kumar Nayak, Alex Reimers, Nick Feamster, and Russ Clark. Resonance: dynamic access control for enterprise networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking (WREN)*, pages 11–18, New York, NY, USA, 2009. ACM.

[19] Openflow. The openflow switch specification. http://OpenFlowSwitch.org.

[20] Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM*, August 2009.

[21] Lucian Popa, Minlan Yu, Steven Y. Ko, Sylvia Ratnasamy, and Ion Stoica. Cloudpolice: taking access control out of the network. In *Proceedings of the Ninth ACM SIGCOMM Hotnets Workshop*, pages 7:1–7:6, New York, NY, USA, 2010. ACM.

[22] Yu-Wei Eric Sung, Sanjay G. Rao, Geoffrey G. Xie, and David A. Maltz. Towards systematic design of enterprise networks. In *Proceedings of ACM CoNEXT*, pages 1–12, New York, NY, USA, 2008. ACM.

[23] Voltaire Inc. Voltaire vantage 6024 switch. available at http://www.voltaire.com/Products/Ethernet/voltaire_vantage_6024, 2010.

[24] Geoffrey G. Xie, Jibin Zhan, David A. Maltz, Hui Zhang, A. Greenberg, Gisli Hjalmtysson, and Jennifer Rexford. On static reachability analysis of ip networks. In *Proceedings IEEE INFOCOM*, volume 3, pages 2170–2183 vol. 3, March 2005.

X. Proof of NP-hardness of Feasibility for Fat Tree Topology

Proof. For tree based routing, we show that the decision problem is hard even if the network has a fat tree topology.

We reduce the NP-complete problem MONOTONE 3-SAT [8] to our problem. From an instance of MONOTONE 3-SAT with variables x_j, $1 \le j \le n$, and clauses T_i, $1 \le i \le p$, with unnegated variables and clauses F_i, $p+1 \le i \le k$, consisting of negated variables we construct an instance of the k-paths-tree problem as illustrated in Figure 6.

More formally, we define an instance of the k-paths-tree problem as follows. Let $T_i = x_{i,1} \vee x_{i,2} \vee x_{i,3}$ and $F_i = \bar{x}_{i,1} \vee \bar{x}_{i,2} \vee \bar{x}_{i,3}$. Define

$$V = \{R, M_T, M_F, d_T, d_F, x_1, x_2, \ldots, x_n,$$
$$T_1, T_2, \ldots, T_p, F_{p+1}, F_{p+2}, \ldots, F_k\}.$$

Let $E_j = \{x_j d_T, x_j d_F\}$, $1 \le j \le n$. Finally, for $1 \le i \le p$ define $E'_i = \{T_i x_{i,1}, T_i x_{i,2}, T_i x_{i,3}\}$ and for $p+1 \le i \le k$ define $E'_i = \{F_i \bar{x}_{i,1}, F_i \bar{x}_{i,2}, F_i \bar{x}_{i,3}\}$. Then let

$$E = \{RM_T, RM_F, M_T d_T, M_F d_F\} \bigcup_{j=1}^{n} E_j \bigcup_{i=1}^{p} E'_i.$$

In the k-paths-tree instance let $G = (V, E)$ be the graph. We define the source nodes s_i as $s_i = T_i$ if $1 \le i \le p$ and $s_i = F_i$ if $p+1 \le i \le k$. The middle nodes are defined as $m_i = M_T$ if $1 \le i \le p$ and $m_i = M_F$ if $p+1 \le i \le k$.

Suppose there is a solution to the MONOTONE 3-SAT instance. Consider one such solution S. We will construct a solution P to the k-paths-tree instance. Put the edges $M_T R$ and $M_F R$ into the set P. For every T_i there is some $x_{i,j}$ that is true in S and so choose one such $x_{i,j}$ and put the edges $T_i x_{i,j}$ and $x_{i,j} M_T$ into the set P. Similarly for every F_i there is some $x_{i,j}$ that is false in S and so we choose one such $x_{i,j}$ and put the edges $F_i x_{i,j}$ and $x_{i,j} M_F$ into the set P. It can be easily checked that P forms a tree and the path from each source node to R goes through its corresponding middle node. That is, P is a solution to the k-paths-tree instance.

Now suppose we have a solution P to the k-paths-tree instance. Consider any sources T_i and F_h. The path from T_i to R in P is called a *true path* and must be of the form $T_i x_{i,j} d_T M_T R$ and the path from F_h to R in P is called a *false path* and must be of the form $F_h x_{h,s} d_F M_F R$ and it must be that $x_{i,j} \neq x_{h,s}$ since otherwise $x_{i,j} d_T M_T R M_F d_F x_{h,s}$ would be a cycle in P contradicting the fact that P is a tree. Therefore it makes sense to set the variable $x_{i,j}$ to True if there is a true path through it in P and otherwise set it to False. Then since P is a solution to the k-paths-tree instance, there must be a path from every T_i and F_i to R in P and hence for every clause T_i there is a variable $x_{i,j}$ that has been set to True and for every clause F_h there is a variable $x_{h,s}$ set to False. That is we have a solution to the MONOTONE 3-SAT instance. □