Traffic Engineering in Software Defined Networks

Sugam Agarwal [1] Murali Kodialam T. V. Lakshman

Bell Labs,Alcatel-Lucent
Holmdel, NJ, USA
{muralik, lakshman@alcatel-lucent.com}

***Abstract*—Software Defined Networking is a new networking paradigm that separates the network control plane from the packet forwarding plane and provides applications with an abstracted centralized view of the distributed network state. A logically centralized controller that has a global network view is responsible for all the control decisions and it communicates with the network-wide distributed forwarding elements via standardized interfaces. Google recently announced [5] that it is using a Software Defined Network (SDN) to interconnect its data centers due to the ease, efficiency and flexibility in performing traffic engineering functions. It expects the SDN architecture to result in better network capacity utilization and improved delay and loss performance. The contribution of this paper is on the effective use of SDNs for traffic engineering especially when SDNs are incrementally introduced into an existing network. In particular, we show how to leverage the centralized controller to get significant improvements in network utilization as well as to reduce packet losses and delays. We show that these improvements are possible even in cases where there is only a partial deployment of SDN capability in a network. We formulate the SDN controller's optimization problem for traffic engineering with partial deployment and develop fast Fully Polynomial Time Approximation Schemes (FPTAS) for solving these problems. We show, by both analysis and ns-2 simulations, the performance gains that are achievable using these algorithms even with an incrementally deployed SDN.**

I. Introduction

Software Defined Networking (SDN) is a new networking paradigm [1], [13], [14] that separates the control and forwarding planes in a network. This functional separation and the implementation of control plane functions on separate centralized platforms has been of much research interest due to various expected operational benefits. Separating interdomain routing from individual routers and using a logically centralized Routing Control System was proposed in [2], [4] as a means to make routing systems more manageable, less complex and be accommodative of new service needs. The SoftRouter architecture [12] proposed the disaggregation of routers into packet forwarding elements, with open standardized interfaces, that are controlled by centralized control plane servers. This was proposed as a means for quicker introduction of network functions such as traffic engineering, new VPN features, etc. and also to achieve various other cost, operational benefits. Re-architecting the control plane into a dissemination plane and a decision plane was proposed in [10]. The disseminaton plane reliably disributes information to the network elements and the decision plane makes all decisons that affect the network. This re-architecture makes it easier for the decision plane to have a global view of network state and permits operators to express desired network behavior more easily.

Common to all the above approaches is that an SDN comprises of two main components:

- **SDN Controller (SDN-C):** The controller is a logically centralized function [3], [8]. A network is typically controlled by one or a few controllers. The controllers determines the forwarding path for each flow in the network.
- **SDN Forwarding Element (SDN-FE):** The SDN-FEs constitute the network data plane. The logic for forwarding the packets is determined by the SDN controller and is implemented in the forwarding table at the SDN-FE.

Openflow [14] is a standardized interface that can be used by the controller to communicate with the forwarding element. When a flow is initiated in the network the following actions are taken: (i) The first packet of the flow is sent by the SDN-FE to the SDN controller, (ii) The forwarding path for the flow is computed by the SDN controller (SDN-C), (iii) The SDN-C sends the appropriate entries to install in the forwarding tables at each SDN-FE on the path from the source to the destination, (iv) All subsequent packets in the flow are forwarded in the data plane and do not need any control plane action.

The SDN controller is responsible for path selection and therefore all policy information resides at the controller. It is also possible to implement centralized or partially centralized traffic management function at the SDN controller. For instance, Google has built a SDN with Openflow routers to interconnect its data centers (G-Scale) [5]. Google is expecting an improvement in network utilization of 20-30% as well as improved delay and loss performance [5] as a result of using an SDN.

The traffic engineering problem that we consider is motivated by scenarios where SDNs are incrementally deployed in an existing network. In such a network, not all the traffic is controlled by a single SDN controller. There may be multiple controllers for different parts of the network and also some parts of the network may use existing network routing. The key question then is whether it is still possible to do effective traffic engineering when all the traffic in the network cannot be controlled centrally by a single SDN controller.

In this paper, we consider traffic engineering in the case where a SDN controller controls only a few SDN forwarding elements in the network. The rest of the network does hop-by-hop routing using a standard routing protocol like OSPF.

[1] Work done while at Bell Labs

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

The objective of the paper is to develop a SDN deployment scheme that can adaptively and dynamically manage traffic in a network to accommodate different traffic patterns. Our main contributions in this paper are the following:

- To our knowledge, this is the first paper to address network performance issues in an incrementally deployed SDN.
- We formulate the SDN controller's optimization problem and outline a Fully Polynomial Time Approximation Scheme (FPTAS) to solve the controllers problem.
- We show by analysis and simulatons the considerable gains in delay and loss performance even when a limited number of SDN forwarding elements are deployed in the network.
- Given a fixed number of SDN-FEs we outline an algorithm for determining the placement of these forwarding elements.

Due to space limitations, we do not show extensions to the approach in this paper to the problem where the network is partitioned between multiple SDN controllers.

II. System Description

We consider a network where a centralized SDN controller computes the forwarding table for a set of SDN forwarding elements. We assume that the SDN forwarding elements are a subset of the nodes in the network. The rest of the nodes in the network run some standard hop-by-hop routing protocol like OSPF. We assume that the SDN-C peers with the network and gathers link-state information. The SDN-FEs forward packets and the logic for computing the routing table at the SDN-FEs resides at the centralized SDN -C. In addition to forwarding packets, the SDN-FEs do some simple traffic measurement which they forward to the controller. The controller uses this traffic information along with information disseminated in the network by OSPF-TE to dynamically change the routing tables at the SDN-FEs in order to adapt to changing traffic conditions. The controller also exploits the fact that it can make co-ordinated changes across the different SDN-FEs that it controls to manage changing traffic effectively. The regular nodes in the network, also referred to as the *non-SDN-FEs*, are standard routers using the usual hop-by-hop forwarding mechanism. This hybrid network scenario with traditional networks intermixed with SDNs is also considered in [11]. Our objectives are considerably different from the objectives in [11]. We assume that no changes are made to the non-SDN-FEs and in fact they can be unaware of the existence of the SDN-FEs in the network.

An example of a network with SDN-FEs and controller is shown in Figure 1. SDN-FEs $2, 9, 14$ are controlled externally. We will use this network to illustrate some of the concepts that we outline in the rest of the paper. We assume that all the links in the network are bidirectional and all link weights are set to one. We now describe the SDN-FEs and the SDN-C in some more detail. The description is at a fairly high level especially for the SDN-C. The actual algorithm that are run at the controller is outlined in Section 3.

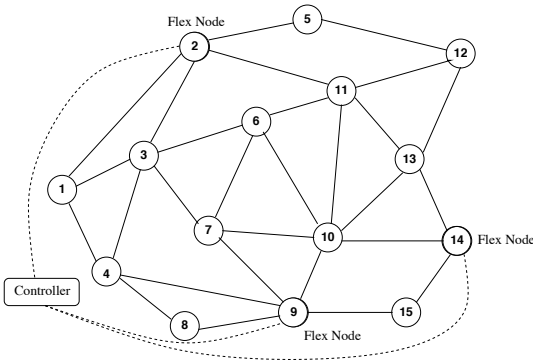

Fig. 1. SDN-FEs and SDN Controller

A. SDN Forwarding Element

The SDN-FEs perform the following functions:

Forwarding: The SDN-FEs act basically as forwarding elements. The routing table at the SDN-FEs is however computed by the SDN-C. We assume that the SDN-FEs can handle multiple next hops for a given destination. If there are multiple next hops for a given destination, then the SDN-FEs can split traffic to the destination in a pre-specified manner across the multiple next hops.

It is relatively easy for the controller to compute multiple next hops and load the routing table to the SDN-FEs [12]. There are several ways of splitting traffic on multiple next hops [15] while ensuring that a given flow is not split across multiple next hops. Some of these approaches need extra measurements and it is relatively easy to extend the current approach to obtain the extra information. Therefore in the rest of the paper, we assume the the SDN-FEs can split traffic across multiple next hops for a given destination. We also assume that the SDN-FEs can perform traffic measurements using techniques such as those in [15].

Measurement: The routing table at the SDN-FEs is modified slightly, compared to a standard routing table, in order to aid traffic measurement at the SDN-FEs. A schematic representation showing the difference between a standard routing table and the routing table at the SDN-FEs is shown in Figure 2. Note that there is an extra column for the node in the network that can reach the destination IP address. When a packet is processed by the SDN-FE it does a longest prefix match on the destination IP address to determine the next hop. It also increments a counter corresponding to the destination node by the packet length. This is done in order to determine the amount of traffic between the SDN-FE and all other nodes in the network. Consider the routing table at node 2. Assume that node 15 (IP address 45.67.2.5) announces reachability to the subnet 135.23\16. Let node 11 with interface IP address 43.2.34.7, be the next hop on the shortest path from node 2 to node 15. A portion of the routing table at node 2 is shown in Figure 2. The column corresponding to the traffic tracks the number of bytes routed from node 2 to node 15 for the destination prefix 135.23\16. It is easy for the SDN-FE to also compute the total traffic sent from node 2 to node 15.

Prefix	Node	Next Hop	Traffic
135.23/16	45.67.2.5	43.2.34.7	

Fig. 2. Enhanced Routing Table at the SDN-FE

B. SDN Controller

The SDN-C has all the routing logic and it coordinates the routing of all the SDN-FEs in order to achieve good network performance. The controller does the following functions:

Peering: The SDN-C peers with the other nodes in the network exchanging link weights and other topology information using OSPF-TE. (See [9] for an example.) Note that in OSPF-TE, the nodes also exchange available bandwidth information on the links in the network. Therefore the controller knows the current OSPF weights as well as the amount of traffic flow on each link (averaged over some time period).

Route Computation: The controller is responsible for computing the routing table for all the SDN-FEs in the network. It computes these routing tables taking into consideration the routing done by non-SDN-FEs (based on OSPF link weights), the traffic at the links (derived from OSPF-TE information or) and the current traffic pattern (inferred from the measurements at the SDN-FEs). The algorithm for computing the routing table for the SDN-FEs has to ensure that routing will be along *loop-free paths* while minimizing the congestion in the network. We will describe the SDN controller's problem formulation and solution technique in Sections 3 and 4.

III. The SDN Controller's Problem

We now describe the problem that the SDN controller has to solve in detail. Assume that the network comprises of a set of nodes N interconnected by a set of directed links E. We assume that there are n nodes and m links in the network. Let $C \subseteq N$ denote the set of SDN-FEs and $D = N \setminus C$ denote the non-SDN-FEs. Let $w(e)$ and $c(e)$ denote the OSPF link weight and capacity respectively of a link $e \in E$. We use $f(e)$ to represent the traffic flow on link e. The flow on all links $e \in E$ is available to the controller from OSPF-TE. We use T_{sd} to represent the traffic rate from node $s \in N$ to some other node $d \in N$ and W_{ud} to represent the total amount of traffic for destination $d \in N$ that either originates or passes through SDN-FE $u \in C$. Note that in general $W_{ud} \geq T_{ud}$. SDN-FE u can measure W_{ud} for all destinations d using the enhanced routing table described in Section 2. The value of T_{sd} for all node pairs (s, d) will not be known to the controller. Each node computes the shortest path to all other nodes in the network. The routing table at node $u \in N$ comprises of the next hop on the shortest path to each node in the network. We use $NH(u, d)$ to denote the next hop node for destination d at node u. In other words, $NH(u, d)$ is the first node on the shortest path from u to d. In the remainder of this paper, we assume that the next hop is unique for all the non-SDN-FEs, i.e, $NH(u, d)$ has only one element for all $u \in D$. We make this assumption purely for the ease of exposition. The techniques in this paper extend directly to the case where there alternate shortest paths between two nodes and traffic is split between these two paths as in Equal Cost Multipaths. Note that while $NH(u, d)$ is computed based on shortest paths for all nodes $u \in D$, $NH(u, d)$ can be set arbitrarily when $u \in C$ as long as there are no routing loops.

We illustrate some of the ideas outlined in the above paragraph using Figure 3. We assume that all link weights are one and the solid links represent the shortest path tree to node 13. This is the tree that will result if the SDN-FEs also use the standard shortest path computation. Recall that nodes $2, 9, 14$ are the SDN-FEs. Note that $NH(6, 13) = 10$, $NH(1, 13) = 2$ and so on. The dotted lines in the network that show the alternate links possible at the SDN-FEs. For example, node 2 can split the traffic to node 13 along two different next hops one going to node 5 and the other to node 11.

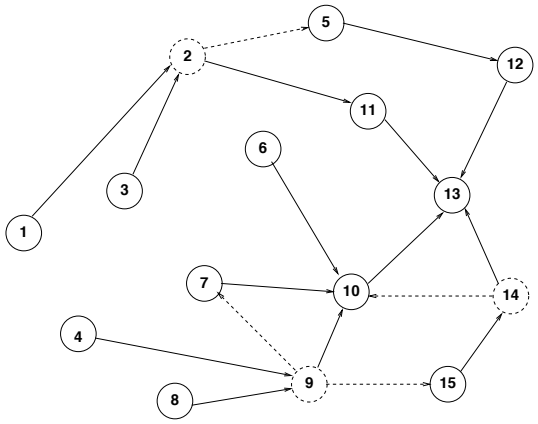

Fig. 3. Shortest Path Tree to Node 13

Definition 1: Given a set of SDN-FE nodes C, a path $s = u_0, u_1, u_2, \ldots u_k = d$ from a source node s to a destination node d will be termed **feasible** if for $j = 1, 2, \ldots, k$, $(u_{j-1}, u_j) \in E$ and

$$u_j = NH(u_{j-1}, d) \text{ if } u_{j-1} \in D \ .$$

A feasible path where $u_0, u_1, \ldots, u_k$ are distinct is called an **admissible path**. Let $\mathcal{P}_{sd}$ denote the set of admissible paths between s and d.

From the definition, note that a path is feasible if the next hop to a given destination for all the non-SDN-FEs is given by the shortest path algorithm. Further a feasible path is admissible only if it is loopless. Therefore we have to ensure that all the traffic between s and d has to be routed on $P \in \mathcal{P}_{sd}$.

For example, in Figure 3, $3-2-5-12-13$ is an admissible path from 3 to 13. Note that this is not the shortest path which is $3-2-11-13$. The path $3-6-11-13$ is not admissible since the next hop for node 3 which is a non-SDN-FE has to be the next hop on the shortest path, which is node 2.

Definition 2: Given shortest path routing at the non-SDN-FEs, traffic that goes from source to destination without transiting through a SDN-FE will be referred to as **uncontrollable traffic**. If the source of a packet is a SDN-FE, or if it passes through at least one SDN-FE before it reaches its destination then this traffic will be called **controllable traffic**.

In other words, controllable traffic comprises of packets that pass through at least one SDN-FE if the packets are routed using standard OSPF. There is at least an opportunity at the SDN-FEs to manipulate the path of controllable traffic. For example the traffic from 6 to 13 is routed by OSPF along $6-10-13$, and since neither 6 nor 10 are SDN-FEs, traffic from 6 to 13 is not controllable. In contrast, traffic from node 8 to 13 passes through node 9 which is a SDN-FE and hence this traffic is controllable.

Definition 3: We say that a SDN-FE $u \in C$ injects a packet if

- Node u is on the OSPF routing path for the packet.
- The packet passes through u before it passes through any other SDN-FE.

The traffic that is injected by SDN-FE $u \in C$ to some destination node $d \in N$ will be denoted by I_{ud}.

Therefore, for all controllable traffic there is a unique SDN-FE that injects this traffic. Note that the SDN-FE may or may not be the source of the traffic that it injects. We illustrate these ideas in Figure 4. In this figure, the number next to the node represents the traffic rate from that node to node 13. For example, the traffic from node 1 to node 13 ($T_{1,13}$) is 3 units. By Definition 3, note that the traffic from 3 to 13 will be injected by the SDN-FE 2. If the values of T_{sd} are known for all source-destination pairs (s, d), then the value of I_{ud} can be computed as follows: Remove the links going out of the SDN-FEs and let OSPF route all demands until it reaches the SDN-FEs or the destination. The traffic that accumulates at the SDN-FEs is the traffic that is injected by the SDN-FE. For example in Figure 3, $I_{2,13} = 9$, $I_{9,13} = 13$ and $I_{14,13} = 5$. As stated earlier, the values of T_{sd} are not known to the SDN-C. The only measurements available at the SDN-C are the values of W_{ud} which is the traffic for destination d that passes through node $u \in C$.

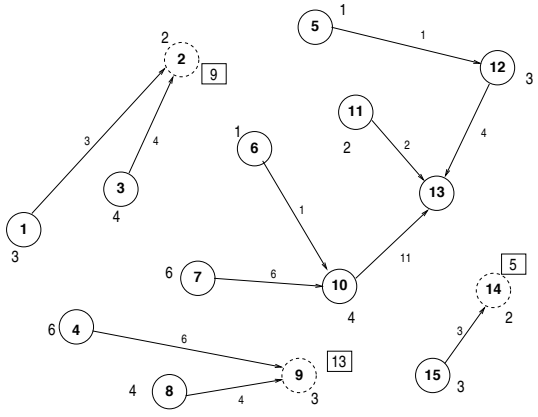

Fig. 4. Independently Routable Traffic at the SDN-FEs

A. Formulation of the SDN-C's Problem

Since the only traffic that we can manipulate is the traffic that passes through the SDN-FEs, we just focus on the traffic injected at the SDN-FEs. Traffic I_{ud} is injected by SDN-FE $u \in C$ that has to reach destination d. It can only do so along one of the admissible paths $P \in \mathcal{P}_{ud}$. Let $g(e)$ denote the uncontrollable flow on link e. (Note that $g(e)$ is easy to computed if the source-destination traffic rates T_{sd} are all known in advance. As stated earlier, this will not be the case in practice. However, we still give the formulation below in order to motivate the actual dynamic routing problem solved by the SDN-C). The objective of the SDN-C is to route the controllable traffic such that delay and packet loss at the links are minimized. The delay and packet loss at the links are increasing functions of the link utilization and therefore we use the link utilization as a surrogate for the delay/losses at the link. One natural objective for the SDN-C is to minimize the maximum utilization of the links in the network. In the formulation, the variables are $x(P)$, which is the flow in path P. Since the number of paths in the network can be exponential in the number of nodes and arcs, the formulation is also exponential. We prefer this path to the more compact node-arc formulation since it lends itself better to the development primal-dual approximation algorithms. The SDN-C solves the following optimization problem:

$$\text{minimize } \theta$$

subject to

$$g(e) + \sum_{P:P\ni e} x(P) \leq \theta\, c(e) \quad \forall e \in E \tag{1}$$

$$\sum_{P\in\mathcal{P}_{ud}} x(P) \geq I_{ud} \quad \forall u \in C \quad d \in N \tag{2}$$

$$x(P) \geq 0 \quad \forall P \tag{3}$$

- The first set of inequalities ensure that the total flow on the link which is the sum of the uncontrollable flow (represented by $g(e)$) and the controllable flow (which is the second sum term on the right hand side) is less than the product of the maximum link utilization (θ) and the capacity of the link ($c(e)$).
- The second set of inequalities ensures that the total injected traffic is routed in the network.
- The third set of inequalities ensures that the flow on any path is non-negative.

The optimum value of θ is the maximum utilization of any link. Note that if the optimum value of $\theta < 1$, then none of the links will be over-utilized. Once the SDN-C solves this optimization problem, it is easy to compute the next hops and the corresponding fractions at all the SDN-FEs for each destination.

In the formulation above, we assumed that values I_{ud} and $g(e)$ are known. In reality both the quantities I_{ud} as well as $g(e)$ have to be computed by the SDN-C based on

the measurements made by the SDN-FEs and the OSPF-TE messages received by the SDN-C.

B. Computing I_{ud} and $g(e)$ Dynamically

The only measured quantities that are available to the SDN-C are

- The link load $f(e)$ for all links $e \in E$ that can be got from OSPF-TE information.
- The quantities W_{ud} for all $u \in C$ for all $d \in N$. This is measured by the SDN-FEs and sent to the SDN-C.

Using these two quantities, the SDN-C has to compute the values of $g(e)$ for all $e \in E$ and I_{ud} for all $u \in C$ for all $d \in N$. We first outline the computation of I_{ud}. Consider a fixed destination node d. The SDN-C knows the current routing to this destination node d. It knows all the next hops for all nodes in D and at all the SDN-FEs it not knows all the next hops for the destination and the traffic split if there are multiple next hops.

Definition 4: Given a destination d and the current routing in the network, the **routing order** of the nodes in C with respect to this destination d is defined as an ordering of the nodes in $C \setminus d$ such that if $u \in C$ appears before $v \in C$ in this list then there is no traffic whose destination is d that is routed from v to u. We denote the routing order for destination node d as $R(d)$ and the fact that u appears before v in $R(d)$ as $u \prec_d v$.

This routing order is well defined for any destination node d since there cannot be any routing loops in the traffic flowing to destination d. (In fact it is possible to order all the nodes in the network not just the nodes in C, but we are interested only in the ordering of the nodes in C.) Assume that the current routing to node 13 is as shown in Figure 5. We only show the portion of the routing that is relevant for the SDN-FEs. In this case note that there is traffic from node 9 that passes through node 14. Therefore $9 \prec_{13} 14$. One routing order is $(2, 9, 14)$. Other orderings are possible but node 14 should appear after node 9 in any ordering.

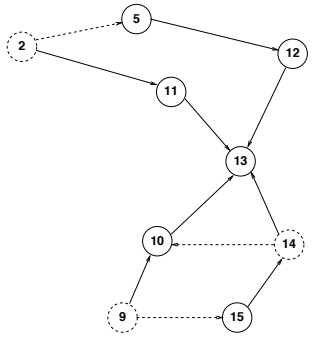

Fig. 5. Illustrating Routing Order

The algorithm to compute the values of I_{ud} is given below.

Computing I_{ud}

For each destination $d \in N$

1. Compute the routing order $R(d)$
2. For the first node u in $R(d)$ set $I_{ud} = W_{ud}$
3. Route one unit of flow from u to d and set $\beta_v(u, d)$ be the fraction of this unit flow that reaches node $v \in C$.
4. For each successive node w in $R(d)$
   Set $I_{wd} = W_{wd} - \sum_{u \prec_d w} \beta_w(u, d) I_{ud}$.
   Route one unit of flow from w to d and compute $\beta_v(w, d)$ for all $v \in C$.

Once the values of I_{ud} are known for all $u \in C$ for all $d \in N$, we use this to compute the values of the $g(e)$ which is the uncontrollable traffic that flows on link $e \in E$. This is done as follows We inject one unit of flow at node $u \in C$ for destination $d \in N$ and computes $\alpha_e(u, d)$ which is the fraction of this unit flow that is routed on link e. Since we know I_{ud} for $u \in C$, we can compute

$$g(e) = f(e) - \sum_{u \in C} \sum_{d} \alpha_e(u, d) I_{ud} \quad \forall e \in E.$$

The SDN-C now knows the values of I_{ud} for all nodes $u \in C$ for all $d \in N$ as well as the values of $g(e)$ for all $e \in E$.

C. Formulating the Dynamic Routing Problem

The SDN-C routes traffic to minimize the maximum utilization of the links in the network. An equivalent problem that is more convenient to solve is to keep the capacities of the link fixed but scale the injected traffic so that it still fits in the network. This problem is the following:

$$\text{maximize } \lambda$$

subject to

$$\sum_{P: P \ni e} x(P) \leq c(e) - g(e) = b(e) \quad \forall e \in E \tag{4}$$

$$\sum_{P \in \mathcal{P}_{ud}} x(P) \geq \lambda I_{ud} \quad \forall u \in C \quad d \in N \tag{5}$$

$$x(P) \geq 0 \quad \forall P \tag{6}$$

If the optimal $\lambda > 1$ then the current traffic can be routed at the SDN-FEs while ensuring that all link utilizations are less than one. Note that the optimal solution to this scaling problem is the inverse of the optimal solution to the min-max utilization problem. In spite of the fact that the problem has an exponential number of variables, we can solve the problem to any desired level of accuracy using a primal-dual algorithm.

In order to write the dual linear program to the dynamic routing problem shown above, we associate dual variables $l(e)$ with each link capacity constraint (4) and z_{ud} for the

demand constraints (5). The dual can now be written as

$$\text{minimize} \sum_{e \in E} b(e)\, l(e)$$

subject to

$$\sum_{e \in P} l(e) \quad \geq \quad z_{ud} \quad \forall P \in \mathcal{P}_{ud} \;\; \forall u \in C \;\; \forall d \quad (7)$$

$$\sum_{u \in C} \sum_{d \in N} I_{ud}\, z_{ud} \quad \geq \quad 1 \quad (8)$$

$$l(e) \quad \geq \quad 0 \quad \forall e \in E. \quad (9)$$

Assume that we set $l(e)$ to be the weight of link $e \in E$. (We use the term weight instead of cost in order to avoid confusion with the OSPF link costs). Note from the first set of constraints z_{ud} is the lightest path from u to d. (Again we use the term lightest path to avoid confusion with the shortest path using OSPF costs). Let L_{ud} denote the lightest path from u to d using the link weights $l(e)$ on link e. The dual can now be re-written as

$$\text{minimize} \sum_{e \in E} b(e)\, l(e)$$

subject to

$$\sum_{u \in C} \sum_{d \in N} I_{ud}\, L_{ud} \quad \geq \quad 1 \quad (10)$$

$$l(e) \quad \geq \quad 0 \quad \forall e \in E. \quad (11)$$

In other words, given any non-negative set of link weights $l(e)$, note that $\frac{\sum_{e \in E} b(e) l(e)}{\sum_{u \in C} \sum_{d \in N} I_{ud} L_{ud}}$ is an upper bound on the dynamic routing problem. We now outline the solution of the dynamic traffic management problem.

D. Solving the Dynamic Routing Problem

We use a Fully Polynomial Time Approximation Scheme (FPTAS) to solve the dynamic routing problem at the SDN-C. The reason for solving the problem as an FPTAS instead of a standard linear programming problem is that the FPTAS is very simple to implement and runs significantly faster than a general linear programming solver especially on medium and large sized problems. An FPTAS provides the following performance guarantees: for any $\epsilon > 0$, the solution has objective function value within $(1+\epsilon)$-factor of the optimal, and the running time is at most a polynomial function of the network size and $1/\epsilon$. The FPTAS in our case is a primal dual algorithm.

The primal dual algorithm for our problem works as follows: The algorithm first computes a value δ that is a function of the desired accuracy level ϵ, the number of nodes n and the number of arcs m. The dual weight of each edge $e \in E$ is initialized to $l(e) = \frac{\delta}{b(e)}$. The primal dual algorithm operates in phases where in each primal phase flow is routed to a given destination from all SDN-FEs along the lightest permissible path (using the dual vector $l(e)$ as the link weight). Once each SDN-FE u has shipped a flow of I_{ud} to destination d, the (dual) weights of the arcs on which flow has been sent is incremented. This process of augmenting flow and updating the dual lengths is repeated until the problem is dual feasible.

The algorithm is given in more detail below:

Algorithm COMPUTE THROUGHPUT:

$D_L \leftarrow 0$
$l(e) \leftarrow \delta / b(e)\;\; e \in E$
$R_{sd} \leftarrow 0\;\; \forall (s,d)$
while $D_L < 1$ **do**

for each destination $d \in N$
  $d'(u) = I_{ud}\;\; \forall u \in C$
  while $D_L < 1$ and $d'(u) > 0$ for some $u \in C$ **do**
    P_{ud} : Shortest admissible path using l,
      $\forall u$ with $d'(u) > 0$
    $c = \min_{e \in \cup_s P_{sd}} b(e)$
    $\rho(e)$ is the utilization of $e \in E$
    $\rho = \max\{1, \max_{e \in \cup_u P_{ud}} \rho(e)\}$
    $f(u) = \min\{d'(u), c\}\;\; \forall u$
    Route $\frac{f(u)}{\rho}$ flow from each u to d.
    $d'(u) = d'(u) - \frac{f(u)}{\rho}$
    $R_{ud} = R_{ud} + \frac{f(u)}{\rho}$
    $l(e) = l(e)\,(1 + \epsilon \rho(e))$
    Recompute $D_L = \sum_{e \in E} b(e) l(e)$
  end while
end for

end while

$\lambda = \min \frac{R_{ud}}{T_{ud}}$

Output λ

The next result gives the running time of the algorithm and the proof of this result is almost identical to the one in Karakosatas [7].

Theorem 1: Set

$$\delta = \frac{1}{(1+n\epsilon)^{\frac{1-\epsilon}{\epsilon}}} \left(\frac{1-\epsilon}{m}\right)^{\frac{1}{\epsilon}}$$

then the running time of the primal-dual algorithm is

$$O(\epsilon^{-2} m^2 \log^{O(1)} m)$$

where m is the number of edges in the graph.

Some Remarks:

1. The algorithm follows in the same vein as Karakostas [7]. The correctness of the algorithm as well as the running time analysis is identical to the results in the paper and is therefore omitted. There are however some key differences in the implementation of the algorithm.

2. Unlike the Karakostas [7] paper, the computation is organized on a destination basis rather than the source. This is critically important for us since we can compute the routing from all SDN-FEs to any destination using a single shortest path computation. Since the routing at the non-SDN-FEs is based on the destination, we cannot organize the computation from each source.
3. In each iteration, we have to compute the lightest admissible path from all SDN-FEs to a given destination. We show how this is done in Figures 6 7. In Figure 6, we give the OSPF shortest path tree to node 13 along with all links incident to the SDN-FEs. The numbers next to the links represents the dual weights (not OSPF costs). All other links also have dual weights but we do not show them here. Assume that we are now computing the lightest admissible paths to node 13 from all the SDN-FEs. $(2, 9, 14)$. Since the admissible paths have to use the OSPF shortest path at all the non-SDN-FEs, we can now reduce the graph to remove all the non SDN-FE nodes. We introduce an arc from each SDN-FE to node 13 for each path from that SDN-FE to node 13. This is shown in Figure 7. Note that the arc with 0.8 from node 2 to node 13 represents the path $2 - 11 - 13$ and the arc with weight 0.4 from node 2 to node 13 is the path $2 - 5 - 12 - 13$. Once this new graph is formed, the lightest arc from each SDN-FE to the destination represents the lightest path. The reduced graph is shown explicitly only for illustrative purposes and is only computed implicitly during the primal dual algorithm.
4. Since we are maximizing the throughput of the network, there exists an optimal solution to the problem where the routing has no loops. This is due to the fact that loops increase network utilization without increasing throughput. However, there may be alternate optimal solutions with loops. The primal-dual algorithm typically finds the loopless optimal solution. To guard against loops in the optimal solution, we post process the optimal solution using a breadth-first search to detect and eliminate loops. The running time of this post processor is dominated by the running time of the primal dual step.

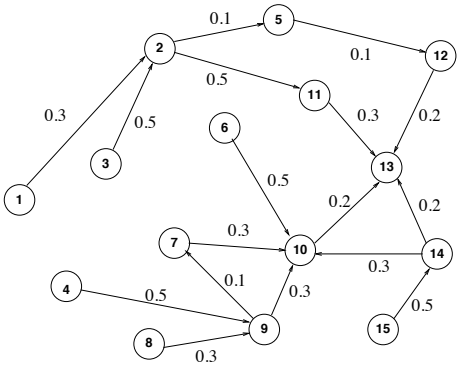

Fig. 6. Part of the Graph with Dual Weights

IV. Selecting the Location of SDN-FEs

Given a network topology, the first decision is to pick the set of SDN-FE locations in the network. Once these SDN-

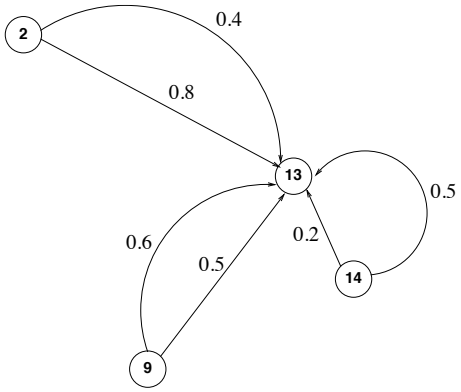

Fig. 7. Reduced Graph for Lightest Path to Node 13

FE locations have been determined, the SDN-C solves the dynamic routing problem periodically to determine the routing of traffic at the SDN-FEs based on the traffic pattern in the network. Theoretically any system with SDN-FEs will outperform a system without SDN-FEs if all the information available to the SDN-C is accurate. In practice the improvement in performance in a system with SDN-FEs depends on the location of the SDN-FEs. The optimum selection of nodes will depend on the traffic matrix which will not be known accurately ahead of time. There are two options. The first is to pick the nodes independent of the traffic matrix and the second approach is to use some estimate of the traffic pattern to pick the SDN-FEs. We attempted both these approaches but we only outline the second approach in this paper. We assume that we know the number of SDN-FEs in the network and we are given a tentative traffic matrix T where T_{sd} is the traffic between nodes $s \in N$ and $d \in N$. The actual traffic can, and in general will, deviate from this traffic matrix. Assume that we have to locate h SDN-FEs. We use T to decide where these h SDN-FEs are placed in the network.

Definition 5: The throughput of a traffic matrix T over a set of SDN-FEs C is defined as the largest scalar λ such that λT can be routed in the network along permissible paths. We denote the throughput value by $\lambda(T, C)$.
Note that if $C = \emptyset$ corresponds to the case where there are no SDN-FEs and the network behaves as an OSPF network. The case where $C = N$ corresponds to the case where all the nodes in the network are SDN-FEs. From the definition, it is easy to see

$$\lambda(T, \emptyset) \leq \lambda(T, C) \leq \lambda(T, N) \quad \forall T, C.$$

Also note that $\lambda(T, N)$ can be computed by solving a standard maximum concurrent flow problem on the network.

Therefore given T and f, the objective then is to determine

$$\max_{C:|C|=h} \lambda(T, C).$$

It can be shown that this problem is NP-hard. We use a incremental greedy approach to solve this problem. We start off assuming that the set of SDN-FEs is empty. In each step of the algorithm, we add the node that gives the biggest throughput increase to the existing set. This process is repeated until we have h SDN-FEs in the network. Since

the values of T_{sd} are known, the throughput problem can be written as

$$\text{maximize } \lambda$$

subject to

$$\sum_{P:P\ni e} x(P) \leq c(e) \quad \forall e \in E \tag{12}$$

$$\sum_{P\in\mathcal{P}_{sd}} x(P) \geq \lambda T_{sd} \quad \forall s \in N \quad d \in N \tag{13}$$

$$x(P) \geq 0 \quad \forall P \tag{14}$$

Since the problem is structurally the same as the SDN-Cs problem, we can use a similar primal-dual algorithm for solving this problem.

V. Experimental Results

We ran two groups of experiments to check the effectiveness of the algorithm using the following three topologies: (i) The 15 node topology shown in Figure 1, (ii) The Exodus (Europe) topology from ROCKETFUEL. This topology has 22 nodes and 74 links, (iii) The Abovenet topology from ROCKETFUEL. This topology has 22 nodes and 84 links.

For the 15 node topology all the link capacities were assumed to be equal and the link weights were assumed to be one. For the ROCKETFUEL topologies, the link weights are given and the link capacities are assumed to be the inverse of the link costs. (We have consolidated multiple links between two nodes into a single link in the experiments). We performed two classes of experiments on all three topologies. The first is the static performance measurement and the second is the ns-2 simulation of the new routing scheme.

A. Static Performance Measurement

These experiments were performed to compute the expected performance improvement due to the dynamic routing algorithm if we are given a traffic matrix. This is also used to pick the set of SDN-FEs in the network. In all the plots for static performance measurement, we plot the normalized throughput. The normalized throughput for a given set of SDN-FEs C is defined as

$$\frac{\lambda(T,C)}{\lambda(T,N)}.$$

This ratio is always less than one. Note that for OSPF routing with no SDN-FEs the value of $C = \emptyset$. We performed two sets of experiments.

- **Normalized Throughput versus Number of SDN-FEs**
  For all three topologies, we plot the normalized throughput as we increase the number of SDN-FEs. The initial value when the number of SDN-FEs is zero corresponds to OSPF routing. Note the sharp increase in the normalized throughput initially. This seems to be the reason for the good performance of the algorithm even when there are a few SDN-FEs. For all subsequent experiments, the number of SDN-FEs for the 15 node topology is set to three and for the ROCKETFUEL topologies is set to four.

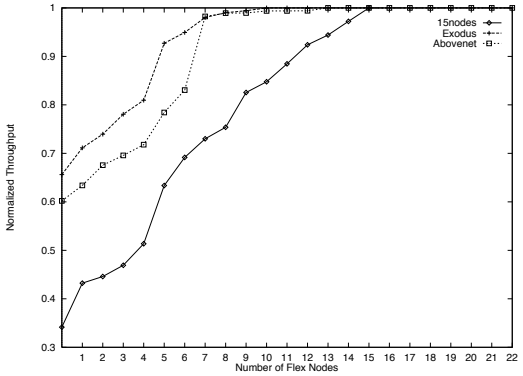

Fig. 8. Effect of Number of SDN-FEs

- **Robustness of the Choice of SDN-FEs**
  In the second set of experiments, we tested the sensitivity of the performance with respect to the real traffic matrix (as opposed to the traffic matrix that is used to pick the SDN-FEs). In other words, since the location of the SDN-FEs is fixed assuming some traffic matrix, we wanted to see how well this choice performed if the traffic matrix is completely different. Therefore, in the second set of experiments, we fixed the SDN-FEs for the Exodus topology to four and fixed their location based on some estimated traffic matrix. We then chose 20 random traffic matrices to check the performance improvement that we get with the different traffic matrices with the same SDN-FE locations. Again we plot the normalized throughput for each of the twenty experiments for both OSPF and SDN routing. Note that the normalized throughput of SDN routing is significantly better than OSPF for all the experiments.

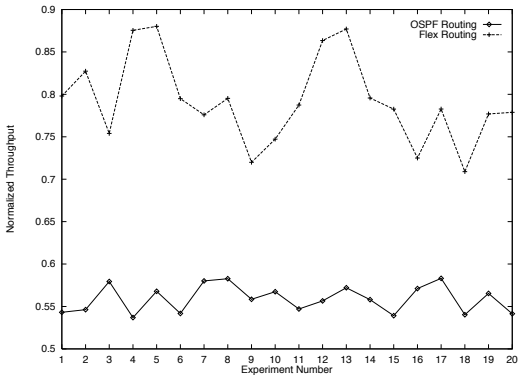

Fig. 9. Performance for Different Traffic Matrices

B. ns-2 Simulation Experiments

Here the objective is to measure link delays and losses in the network under shortest path routing and routing with SDN-FEs. The link state routing protocol was modified to allow SDN-FEs whose routing table was computed centrally using the primal-dual algorithm. For each of the three networks several traffic patterns were generated with each source sending

	Max Loss SDN Routing	Max Loss OSPF
15node EXP 1	0.000	415.000
15node EXP 2	0.000	391.000
15node EXP 3	0.000	320.000
Exodus EXP 1	0.000	140.000
Exodus EXP 2	0.000	105.000
Exodus EXP 3	2.000	106.000
Abovenet EXP 1	1.000	449.000
Abovenet EXP 2	52.000	555.000
Abovenet EXP 3	0.000	552.000

TABLE I
COMPARISON OF MAXIMUM LOSS OVER ALL LINKS

	Mean Loss SDN Routing	Mean Loss OSPF
15node EXP 1	0.000	13.543
15node EXP 2	0.000	11.894
15node EXP 3	0.000	13.228
Exodus EXP 1	0.000	4.906
Exodus EXP 2	0.000	4.693
Exodus EXP 3	0.040	4.106
Abovenet EXP 1	0.012	8.200
Abovenet EXP 2	0.612	11.552
Abovenet EXP 3	0.000	6.670

TABLE II
COMPARISON OF MEAN LOSS OVER ALL LINKS

CBR UDP traffic with randomly chosen rates. Different seeds were used to generate random traffic patterns. For the 15 node topology we used 3 SDN-FEs and for the 22 node topologies we used 4 SDN-FEs. The remainder of the nodes used standard shortest path forwarding. Each simulation was run for 10 seconds. We ran several experiments on all three topologies. We show three typical results for all three topologies. Tables I and II show the maximum number of packets lost over all the links and the mean number of packets lost per link respectively over the 10 second simulation period. Note that SDN Routing does significantly better than than standard OSPF routing. The same is true for the maximum delay (Table III) and mean delay (Table IV). The difference in performance is accentuated if the simulation is run for longer time periods.

VI. CONCLUSION

Incremental SDN deployment where SDNs co-exist with traditional networks is an important deployment scenario that needs to be considered. This is of particular importance for large networks where complete greenfield deployment is difficult. We have shown how improved network performance can be achieved with an incremental deployment of a SDN in an existing network. Deploying even a few strategically placed SDN-FEs in the network can lead to improved network performance. The scheme does not involve making any protocol changes at the remaining nodes in the network which route traffic in a shortest path hop-by-hop manner. Initial performance measurements as well as ns-2 simulations show that the method can significantly improve overall network throughput while providing better delay and loss performance in the network.

	Max Delay SDN Routing	Max Delay OSPF
15node EXP 1	0.119	0.283
15node EXP 2	0.153	0.285
15node EXP 3	0.060	0.277
Exodus EXP 1	0.913	1.645
Exodus EXP 2	0.757	1.732
Exodus EXP 3	0.717	1.712
Abovenet EXP 1	0.228	0.242
Abovenet EXP 2	0.288	0.396
Abovenet EXP3	0.297	0.496

TABLE III
COMPARISON OF MAXIMUM DELAY OVER ALL LINKS

	Mean Delay SDN Routing	Mean Delay OSPF
15node EXP 1	0.003	0.011
15node EXP 2	0.005	0.010
15node EXP 3	0.002	0.015
Exodus EXP 1	0.070	0.106
Exodus EXP 2	0.060	0.106
Exodus EXP 3	0.053	0.100
Abovenet EXP 1	0.010	0.021
Abovenet EXP 2	0.014	0.029
Abovenet EXP3	0.013	0.025

TABLE IV
COMPARISON OF MEAN DELAY OVER ALL LINKS

REFERENCES

[1] M.Casado, M.Freedman, J.Petit, J.Luo, N. McKeown, S.Shenker, "Ethane: Taking Control of the Enterprise", *ACM SIGCOMM CCR*, 37(4):1-12, 2007
[2] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe, "Design and Implementation of a Routing Control Platform", *Networked Systems Design and Implementation*, May 2005.
[3] N.Gude, T.Koponen, J.Petit, B.Pfaff, M.Casado, N. McKeown, "Nox: Towards a Network Operating System", *ACM SIGCOMM CCR*, July, 2008
[4] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, K. van der Merwe, "The Case for Separating Routing from Routers", *FDNA 2004*.
[5] U. Holzle, " Opening Address: 2012 Open Network Summit", April 2012.
[6] "Network Development and Deployment Initiative (NDDI)", *http://www.internet2.edu/network/ose/*.
[7] G. Karakostas, "Faster Approximation Schemes for Fractional Multicommodity Flow Problems", *Presented at ACM-SIAM SODA 2002*.
[8] T. Koponen et. al., "Onix: A Distributed Control Platform for Large Scale Production Networks", *OSDI 2010*, October, 2010.
[9] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Correa, S. C. de Lucena, M. F. Magalhaes "Virtual Routers as a Service: the RouteFlow approach leveraging Software-Defined Networks", *CFI 2011*.
[10] J. Rexford et. al., "Network-Wide Decision Making: Toward a Wafer-Thin Control Plane", *HotNets-III*, November 2004.
[11] C. Rothernberg, C. N. A. Correa, R. Raszuk "Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking", *ACM-SIGCOMM HotSDN Workshop, 2012*
[12] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, T. Woo, "The SoftRouter Architecture", *Proceeding of Hotnets 2004*, November 2004.
[13] N.McKeown, T. Anderson, H.Balakrishnan, G.Parulkar, L.Peterson, J.Rexford, S.Shenker, J.Turner, "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM CCR*, April, 2008.
[14] The Openflow Switch, openflowswitch.org
[15] A. Sridharan, R. Guerin, C. Diot, "Achieving Near-Optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks", *IEEE/ACM Transactions on Networking*, V. 13, No.2, April 2005.
[16] B. Fortz, M. Thorup, "Optimizing OSPF/IS-IS Weights in a Changing World", *IEEE Journal on Selected Areas in Communications*, February 2002.