A Connectionless Approach to Intra-domain Traffic Engineering

Hema Tahilramani Kaur, Shivkumar Kalyanaraman, Shifalika Kanwar, Andreas Weiss,
Niharika Mateti, Biplab Sikdar

***Abstract*—**

We propose a new hash-based approach for connectionless intra-domain traffic engineering (TE) in the Internet. The main idea is to capture an intra-domain path as a 32-bit hash of globally known quantities such as router Ids and link weights in the packet header. This hash, called the "PathID" allows explicit source-directed routing *without signaling* or *high per-packet overhead*, while enabling an *incremental upgrade strategy* for common intra-domain routing protocols like OSPF and IS-IS. Another key advantage of the framework (similar to that achievable in signaled frameworks like MPLS) is the de-coupling of the multi-path computation problem from the problem of mapping traffic to the paths. The proposed scheme essentially trades off computational complexity at upgraded multi-path capable routers to avoid signaling. We analyze these tradeoffs and propose concrete solutions to manage the complexity. Simulation and Linux/Zebra based implementation experiments have been used to demonstrate various aspects of the proposed framework. We demonstrate that a variety of polynomial complexity route computation algorithms, combined with selective upgrades of routers can offer substantial traffic engineering capabilities in the network.

***Index Terms*— Traffic Engineering, MPLS, Multi-path Routing, OSPF**

Methods Keywords – **System design, Simulations, Experimentation with real networks/Testbeds**

I. Introduction

Traffic Engineering (TE) deals with the task of mapping traffic flows to the routes setup in an existing physical topology to meet the network and user performance objectives. A desirable traffic engineering solution must provide network operators a precise control over the traffic flows within their routing domains. This enables them to provide new services by appropriately managing the traffic. Multi Protocol Label Switching (MPLS) is an example of a *connection-oriented* or *signaled* TE approach. MPLS allows explicit setup of label switched paths (LSPs), and arbitrary flexibility in mapping traffic to the available LSPs.

In contrast, current work in the area *connectionless* intra-domain TE uses a *parametric approach* where the routing algorithm only provides a single shortest path (or multiple paths in case of Equal Cost Multi Path (ECMP)) between any node pair, but the link weights are optimized to achieve TE objectives. In this case, the problem of traffic mapping to paths is *coupled* with the problem of route calculation. The parametric approach will hence lead to a route change for any desired change in traffic mapping (or change in the demand matrix). This approach would rule out the possibility of source-controlled traffic mapping on finer time-scales, and would lead to control traffic overhead (LSA re-advertisement) for every change in link weight.

In this paper, we propose a new *connectionless* approach to TE that allows explicit source control over the route taken by a packet through the network. Our approach *decouples* the way traffic is mapped to available paths from the problems of route computation and packet forwarding. Moreover the approach is connectionless, (i.e. it does not need signaling) and can be very effective even in a *partially upgraded* network. We show that the new capability requires a forwarding algorithm upgrade and route computation upgrade at selected routers; and the route computation tradeoff is manageable. In summary, the goal of this paper is to emulate a critical subset of signaled TE capabilities (e.g. MPLS-like) within a partially upgraded connectionless network, with manageable tradeoffs.

A. Classification of Intra-domain Traffic Engineering Approaches

We classify the current intra-domain TE approaches into three groups- a) connectionless parametric approach; b) connection-oriented approach; c) connectionless multi-path approach.

The authors are with the Department of ECSE, Rensselaer Polytechnic Institute, Troy, NY-12180. E-mail: {hema, shivkuma, kanwas}@networks.ecse.rpi.edu, {weissa2, matetn, sikdab}@rpi.edu

The project was supported in part by DARPA Contract F30602-00-2-0537 NSF Grant ANI9819112 and an Intel grant

1) The Parametric Approach: In this approach, the TE problem of mapping traffic on physical links is translated to the problem of using a routing that achieves the desired traffic mapping. The desired routing is obtained by using appropriate OSPF link weights. This is the most common approach to achieve traffic engineering in OSPF networks. The idea of optimizing OSPF link weights for the prevailing traffic conditions was proposed in [5], [16]. The traffic is assumed to be quasi-stationary and link weights are optimized according to the projected traffic demands for certain times of the day. However, finding optimal link weights for a given traffic demand under constraints is a NP-Hard problem [6]. Fortz and Thorup use a local search algorithm to find a good link-weight setting [5], [6]. In [9], we formulate the minimization of packet loss rate in the network as the optimization objective and use an efficient *global search* approach called Recursive Random Search (RRS) [26] to find good link weights.

OSPF-ECMP [8], [22] allows traffic to be split equally among the multiple next hops for paths with equal weights to the destination. OSPF-OMP [24] uses ECMP, but instead of depending upon link weight assignments, samples and floods per-link traffic load information using opaque Link State Advertisements (LSAs). The information is used to change local load-splitting decisions at other upgraded nodes. Though OSPF-ECMP and OSPF-OMP based traffic mapping schemes are different in character from link-weight settings, they involve a degree of coupling between route computation and traffic mapping to these routes.

2) The Connection-oriented Approach: The connection-oriented approach (e.g. MPLS, ATM and frame-relay) is a more direct approach to intra-domain TE. In this approach, paths are established with optional resource reservation and packets may be independently mapped to available paths at the source. The explicit-path selection and setup allows an attractive range of TE capabilities. The clean separation between routing (i.e. control-plane) and forwarding (i.e data-plane) allows the deployment of a single forwarding algorithm (e.g. label-switching). New services can be provided by simply changing the way packets are mapped to routes. The main limitation of the connection-oriented approach is the need for a signaling protocol which requires all the routers along the path to be upgraded and is hard to extend to multiple routing areas within a domain or across multiple domains.

3) The Multi-Path Approach: The multi-path approach aims to exploit the resources of the underlying physical network by providing multiple paths between any source and destination. Multi-path routing has a potential to aggregate bandwidth on various paths, allowing a network to support data transfer rates higher than what is possible with any single path [21]. Prior work has extended intra-domain routing algorithms (both RIP and OSPF) for multi-path support [15], [25]. Narvaez et al [15] propose to find loop-free multi-paths only by looking at concatenating the shortest paths of their neighbors with their link to the neighbors (i.e. depth first search with a depth of 1). Chen et al and Vutukury et al [1], [25] propose more general multi-path computations, but their schemes require the co-operation and upgrade of *all* the routers in the network. Chen et al's work presents a general concept of suffix-matched path identifier to allow multi-path computation using distributed computation, but they use local labels to realize the path. Note that the local labels in Chen et al's framework and MPLS labels only have a *local* significance, and hence require a signaling protocol to map a global path specification to locally assigned labels at each node.

Our proposed framework allows source-based multi-path routing using a "PathID". The use of a *globally significant path hash* allows multi-path capabilities *without signaling* (i.e. in a connectionless manner) even in a *partially upgraded* network. Even though MPLS has gained popularity in some large ISPs, a significant fraction of ISPs [14] favor using OSPF/IS-IS to enable TE without migrating to a fully-MPLS upgraded network. This is due to the widespread deployment and operational experience available with OSPF/IS-IS. Our approach extends the OSPF/IS-IS to allow TE capabilities even in partially upgraded networks.

The key contributions in this paper can be summarized as follows.

- A new connectionless framework that allows packets to be forwarded along explicit path specified by the source without using signaling and in a partially upgraded network scenario. The proposed approach also provides a clean separation between routing (i.e. control-plane) and forwarding (i.e. data-plane), that does not induce routing instability even for fine-grained traffic mapping changes.
- The proposed framework trades off local computational complexity to avoid signaling. We quantify this tradeoff and propose various ways to manage the computation and space complexity. In particular, we propose polynomial-time complexity algorithms for computing available paths in a partially upgraded network.
- We demonstrate mapping of the proposed scheme to an OSPF implementation. Simulation and Linux/Zebra implementation experiments are used

to validate the framework and illustrate its performance.

This paper is organized as follows. Section II describes concepts such as PathID (i.e. hash), packet forwarding paradigm and a basic path computation algorithm using the proposed connectionless approach and sets up the fundamental tradeoffs in the framework. We quantify the computation tradeoffs by analytically deriving, the number of paths available to a router for both the full and partially upgraded case (Appendix). We propose different mechanisms to manage the complexity tradeoff in Section III. In particular, we allow arbitrary choice of polynomial-complexity route computation algorithms at each upgraded router; and show that a common validation phase can ensure that only consistent and valid routes are chosen. We illustrate this concept by using a k-shortest path computation algorithm. The value of k can be chosen to manage both the computational and space overheads. Section IV summarizes the extensions in OSPF to implement the proposed scheme. We have implemented the proposed scheme in Linux using the OSPF implementation in Zebra version 0.92a and MIT's Click Modular Router package [10] for the forwarding plane. In Section V we illustrate the operation of the framework using Linux-based implementation and simulations using SSFNet. Finally, conclusions and directions for future work are presented in Section VI.

II. Proposed Connectionless Approach

A key idea in this paper is the use of a hash of the path in a fixed-size *routing header* which is carried by the packet. The hash is based upon the sequence of router ip-addresses (or router ids) and link weights in the path (that are globally known due to link-state nature of OSPF and IS-IS) unlike local identifiers, i.e. labels, in MPLS.

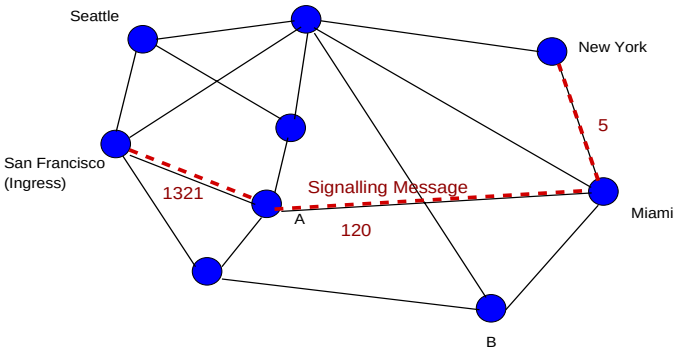

Fig. 1. MPLS Path Set-up

To illustrate the key idea, consider an MPLS LSP from "San Francisco" to "New York" in Figure 1. The ingress router (San Francisco) chooses an explicit path and invokes a signaling protocol (e.g. RSVP-TE or CR-LDP) to setup the LSP. The path setup involves local assignment of *labels* and setting up a input-output label and port mapping in the label switch table. The sequence of label swaps along the path may be imagined as a mapping between local labels at switches to the global path specification which is carried in the signaling protocol. The path itself may be computed in any out-of-band manner or specified manually by the operator. To forward a packet, the ingress router prepends the MPLS header to the IP packet, and specifies the MPLS label of the first hop in the MPLS header. The MPLS label is matched at each hop to decide the output port, and the incoming label is swapped with an outgoing label from the matching table entry. Figure 2 shows an example of label changes as a packet is forwarded along the LSP.

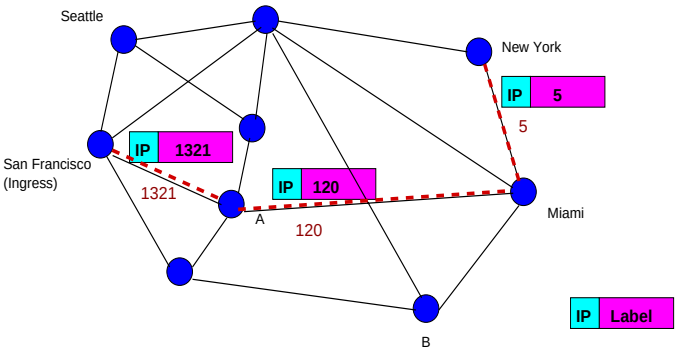

Fig. 2. MPLS Packet Forwarding

Figure 3 shows the packet forwarding and pathID swapping using our proposed connectionless approach. The ingress computes the paths to the destination and forwards packets on these paths by setting appropriate pathID in the *routing header*. In this example, for simplicity, we assume that all the routers are upgraded and pathID is the sum of the weights associated with the links along the path. These assumptions are made in this example for ease of explanation. A routing header is prepended to the incoming IP packet at the ingress router. The pathID field in the routing header is set to 36 for a packet to be routed on the path (SFO-A-Miami-NYC). Note that 36=9+27, i.e. sum of link weights A-Miami and Miami-NYC. At the router A, the pathID of the incoming packet is matched and the packet is forwarded on link A-Miami and the pathID field is set to 27. Figure 3 shows how a multi-path capable ingress utilizes the two paths and achieves forwarding on these paths.

Since OSPF and IS-IS are link-state based algorithms, each router has a complete map of the network. Therefore, a router can *locally* compute the paths that are available to

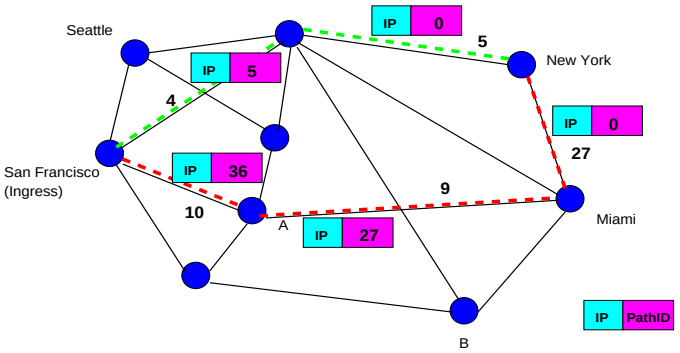

Fig. 3. Packet forwarding and pathID swapping in the proposed scheme

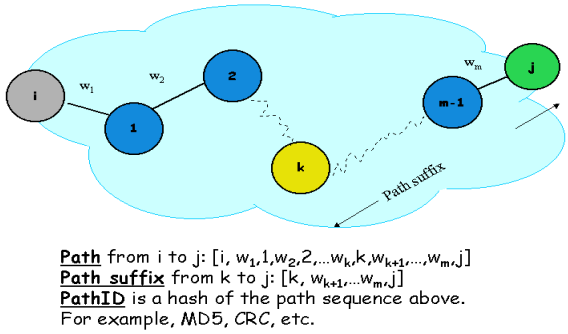

Fig. 4. Path, Path Suffix and PathID Concepts

each destination. It then uses a well known hash function to create the corresponding forwarding table entries. No signaling or connection set-up is used for establishing the path; the approach is connectionless. The connectionless global-hash idea also allows an interesting extension: even in the case of a *partially upgraded* network, the available paths can be computed as described in subsequent sections. Observe that the fundamental tradeoff in our approach is local route computation and space complexity incurred at upgraded routers to avoid signaling.

A. PathID Concept

Consider a network modeled as a graph $G = (V, E)$ where V is the set of vertices or nodes and E is the set of edges or links in the network. Let N denote the number of nodes in the network, i.e. the cardinality of the set V. Each link $(i, j) \in E$ has a weight or cost associated with it, denoted by $w_{i,j}$. Consider a path $p_{i,j}$ from node i to node j, which passes through nodes $i, 1, 2, ..., m-1, j$ and links of weights $w_{i,1}, w_{1,2}, ..., w_{m-1,j}$. This path can be represented as a sequence: $[i, w_{i,1}, 1, w_{1,2}, 2, ..., w_{m-1,j}, j]$. This path sequence can be represented by a *hash* of its elements. If this hash leads to a unique hash value for a unique input sequence with a high probability, we can use this hash to represent a path concisely. A path identifier, in short PathID, is defined as a hash of the sequence of node and/or link identifiers (we use link weights as link identifiers). These concepts are illustrated in Figure 4.

In the case of intra-domain routing (e.g. OSPF or IS-IS), the node IDs (i.e. router IDs) and link weights are known at all routers. Hashing such a sequence of *globally known* quantities allows us to avoid signaling because each upgraded router on the path can unambiguously interpret the hash with a very high likelihood. Recall that one purpose of signaling in ATM and MPLS is to map global IDs (addresses, path specifications) to local IDs (labels). Since we obtain pathID from the global IDs (router IDs and link weights), *signaling is not necessary for path selection*.

The choice of the hash function is dictated by the need to minimize the collision probability which directly affects the uniqueness and utility of the hash. A simple hash of the path sequence may be obtained by using the sum or XOR function. The advantage of using a simple hash is that pathID computation is very simple and fast. On the other hand, it may lead to non-unique pathIDs for different paths with a high probability. We therefore propose to use a 128-bit MD5 hash of the nodeIDs along the path, followed by a 32-bit CRC of the 128 bit MD5 hash to result in a 32-bit hash field. We use the notation (MD5 + CRC-32) hash to represent the above two-step hashing process. This hash in conjunction with the destination address (j) is used to forward a packet at the intermediate routers. If the sequence of node IDs along the path is unique (and assuming for simplicity that adjacent nodes do not have multiple links), then by the properties of the MD5 and CRC-32 hash functions, the tuple = [j, PathID] is very highly likely to be unique.

A quick survey of the collision probabilities for the MD5 hash leads us to the conclusion that MD5 is a robust hash with a collision probability of less than 1 in 100 million. For the CRC-32, the probability for collision can be approximated by $\frac{128+32}{2^{31}}$ which can be evaluated to about 1 in 8 million.

B. Packet Forwarding

The forwarding table entries of the existing OSPF/IS-IS routers is of the form **[destination prefix, outgoing interface]**, and a longest-prefix-match IP lookup procedure is used. At upgraded routers, we propose to include an "incoming pathID" and an "outgoing PathID" fields in the routing table entry to enable explicit forwarding along multiple paths. Hence, the upgraded routers will have the forwarding table entries of the form **[destination**

prefix, incoming pathID, outgoing interface, outgoing pathID]. The "incoming pathID" field represents the hash of the explicit path starting from the current router to the destination prefix. The "outgoing pathID" field is the hash of the path from the *next upgraded router* on the explicit path specified by "incoming pathID" to the destination. As mentioned earlier, the pathID field is stored in a new routing header in IP packets. Incoming packets at a router already have a pathID specification in their routing header or just the destination IP address.

An upgraded router first matches the destination IP address using the longest prefix match followed by an *exact match* of the pathID for that destination. If matched, the incoming pathID in the packet is replaced by the outgoing pathID, and the packet is sent to the outgoing interface. Observe that this procedure is a *hybrid* of IP's longest prefix match and MPLS's label swapping, but using the globally known pathID instead of labels which does not need signaling. If an exact match is not found (i.e. errant hash value in packet), then the hash value in the packet is set to zero, and the packet is sent on the default path (i.e. shortest path in OSPF/IS-IS). The hash value may also be set to zero if the next-hop is the destination itself, or there are no upgraded routers in the path specified by the incoming pathID. A non-upgraded router simply ignores the pathID field and forwards the packet on the shortest path. Such an extension to the forwarding tables will lead to increased space requirements at the router. In general, if there are K paths on an average to any destination then the forwarding table now has $(K-1)N$ more entries in the forwarding table. Also, each routing table entry has 64 bits more than the normal case.

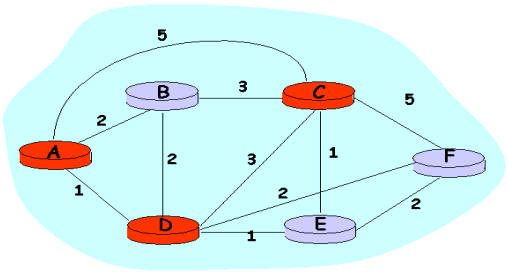

Fig. 5. Multi-Path Forwarding with Partial Upgrades

Figure 5 illustrates the forwarding procedure in a partially upgraded network. Nodes A, C and D in the figure are multi-path capable (MPC). Lets say that node A is the originating node for a packet destined to node F. The shortest path from intermediate node B to node F is B-D-F and path A-B-C-F is not available for forwarding because node B is a non-upgraded node. However, paths such as A-B-D-C-F, A-D-E-F, A-D-C-E-F are available. If the path A-B-D-E-F is chosen, then the PathID of an incoming packet will be Hash(A-B-D-E-F). A sets the PathID field to Hash(D-E-F), i.e. the hash of the path suffix from the next MPC router to destination. Node B forwards the packet on its shortest-path (i.e. to D). Node D sets the PathID to zero, because there is no MPC router on the path to F.

C. Path Computation Issues

In link-state routing, each router has a complete map of the network in the form of link-state database. Using this link-state database and knowledge of upgraded routers, every router can locally compute the paths that are available to each destination. The knowledge of upgraded routers can be obtained by the upgraded multi-path capable (MPC) routers setting a bit (referred to as the MPC-bit) in their link state advertisements (LSAs). Non-upgraded routers merely ignore this bit, whereas the upgraded routers use it to infer available multiple paths. Using this model we first develop a simple (but computationally complex) algorithm to compute *all* paths to a destination under the constraint that a known subset of nodes in the network have been upgraded to support multi-path routing.

The algorithm (Algorithm 1) at upgraded node i uses the network map (graph) and first runs an all-pairs shortest path computation, i.e., the Floyd-Warshall Algorithm [3]. For any chosen node k and destination j, the Floyd-Warshall algorithm sets up the next-hop node l in the shortest path in node k's routing table. Given these routing tables, a depth-first search (DFS) rooted at node i is done to discover multiple paths from i to each destination j. A per-node variable $visited\_nodes$ is used, within each DFS pass, to mark the nodes *visited* by the DFS algorithm. A node is included in the path is it is not *visited* already, thus ensuring loop-free paths. At node k, the algorithm considers *all* neighbors if k is an upgraded node, otherwise, it only considers the next-hop node on the shortest path from k to the destination. If the chosen next-hop node of k has not been *visited* earlier, this node is appended to the path. The procedure is now repeated using k's next-hop node as the source. Once the DFS is complete at node k, then the $visited\_nodes[k]$ is reset to zero. With minor extensions, Algorithm 1 can be used to compute the hash for each path simultaneously.

The computational complexity of simple sequential Floyd-Warshall implementation is $O(N^3)$ where N is the number of nodes in the network. Alternatively, one

Algorithm 1 Algorithm for computing all paths between a source and destination with only some nodes supporting multi-path forwarding

```
adjacency_matrix[i][j]=link_weight if ∃ link from i to j, else -1
partial_paths is the sum of link weights on the path, it is initialized to zero
partial_paths_nexthop is the next-hop node on the path
no_paths denotes the number of path currently being traversed, at the end of the procedure it will denote the number of paths found between a source and destination
N denotes the number of nodes in the network
The array visited_nodes marks a node if it has appeared in the currently traversed path. It is initialized to zero
procedure ComputePartialPaths(src, dst, no_paths, partial_paths, partial_paths_nexthop, level)
begin
visited_nodes[src] ← 1
if src is a multi_path node then
  for i = 1 to N do
    save current value of partial_paths
    if (∃ link from src to i) && (visited_nodes[i]==0) then
      if level == 0 then
        partial_paths_nexthop[no_paths] ← i;
      end if
      partial_paths[no_paths]+=adjacency_matrix[src][i]
      if i==dst then
        partial_paths_nexthop[no_paths+1] ←
                        partial_paths_nexthop[no_paths]
        no_paths ++
      else
        ComputePartialPaths(i,dst,no_paths,
          partial_paths,partial_paths_nexthop,level+1)
      end if
    end if
  end for
else
  /* shortest_paths and shortest_paths_nexthop is computed by calling AllShortestPaths() */
  i ← shortest_paths_nexthop[src][dst]
  if visited_nodes[i]==0 then
    partial_paths[no_paths] += adjacency_matrix[src][i];
    if level == 0 then
      partial_paths_nexthop[no_paths] ← i;
    end if
    if i==dst then
      partial_paths_nexthop[no_paths+1] ←
                        partial_paths_nexthop[no_paths]
      no_paths ++
      ComputePartialPaths(i,dst,no_paths,partial_paths,
                partial_paths_nexthop, level+1)
    end if
  end if
end if
visited_nodes[src] ← 0
end
```

may run Dijkstra *(N-u)* times where u is the number of multi-path capable or upgraded nodes. The Dijkstra's algorithm with adjacency lists has complexity of $O(Elog(N))$, where E is the number of edges or links in the network. So varying over $N-u$ source nodes gives a complexity of $O((N-u)Elog(N))$. Unfortunately, the above complexity is dominated by the complexity of DFS which is given by b^m where b is the branching factor, that is 1 for a non-upgraded node and equal to degree for an upgraded node. m is the depth we hit in a DFS which can be equal to $N-1$ in the worst case. This results in an overall NP-complete algorithm which presents a need to find solutions to manage the computational complexity. The reduction in computational complexity will also lead to the reduction of space complexity. Appendix provides an average analysis of the space complexity, i.e., the number of paths computed using the above approach which is also large for the fully upgraded case.

III. Managing the Computational Complexity

We propose two methods of managing the computational and space complexity:

a) Leave the route computation algorithm (Algorithm 1) unchanged, but we divide routers into two types: Passively Multi-Path Capable (P-MPC) and Multi-Path Capable (MPC). The former class of routers will not advertise themselves as MPC to other routers (i.e. they will not set the MPC bit in their LSAs), even though they are aware of the presence of other MPC routers and can compute the available multi-paths through other MPC routers. This essentially makes Algorithm 1 see a smaller effective number of MPC routers and hence a smaller DFS computational complexity.

b) Change the route computation algorithm to find a *subset* of all available valid paths through the partially upgraded network. Several algorithms of polynomial complexity (in space and time) are available in this realm. Given this, we develop a new polynomial complexity path validation algorithm which can be applied to any set of paths computed by the above algorithms, provided it knows what algorithm is being used at each MPC router. This validation phase hence allows each MPC node to use heterogeneous route computation algorithms to manage the overall space-time tradeoff. An LSA extension is needed for this approach to capture the parameters and the type of the algorithm used at an MPC router. This is discussed in Section IV.

These approaches are discussed further in the following subsections. We present Linux implementation and SSFNet simulation results in Section V to illustrate the simplicity and viability of these two methodologies.

A. Passively Multi-Path Capable Routers

The notion of a Passively Multi-Path Capable (P-MPC) router serves three purposes: control of computation complexity, defining the notion of "sources" in a network, allowing for targeted upgrades and targeted set of multi-paths for a given traffic engineering goal.

First, as discussed earlier, the computation complexity of the DFS is dependent upon the branching factors it encounters. If fewer routers are MPC, then DFS traversals will encounter smaller average branching factors, and hence control the actual computational complexity seen in practice. The total number of paths computed (space tradeoff) is also dramatically reduced in this process.

Second, one can safely upgrade a router (e.g. an edge-router) and allow it to have visibility into the available multi-paths in the network, but without adding to the computational complexity of all other MPC routers. Such visibility is useful for the P-MPC router to act as a "source" for the purposes of source-based traffic mapping within the framework. Indeed, one rough definition of a "source" is the first upgraded (MPC or P-MPC) router in the path of a packet that makes the multi-path forwarding and traffic-splitting decisions (by setting pathID and forwarding appropriately) on behalf of the traffic originator (i.e. source host).

Third, since the other MPC routers do not "see" the P-MPC router as multi-path capable, this feature provides a convenient way of specifying that the P-MPC routers do not offer multi-path forwarding services to the network. Routers with specific characteristics (e.g. core versus edge, routers with large degree, large degree in the shortest path tree etc.) could be chosen to be made MPC, and the rest could be P-MPC. This would also localize a majority of traffic mappings to flow through a controlled set of routers. Moreover, if certain specific explicit paths are desired rather than a broad set of multi-paths, the key nodes on such paths could be upgraded to be MPC.

Thus the simple feature of P-MPC or MPC upgrades provides a considerable administrative flexibility in managing the computation and space tradeoffs.

B. Computing a Subset of Available Multi-Paths

Another approach to manage the complexity at a router is by computing and storing only a subset of available paths. A router may keep multiple paths for a subset of destinations or few paths to all the destinations. Any technique such as k-shortest paths, all k-hop paths, DFS with constrained depth, k-disjoint paths, etc. can be used to compute this subset of all available paths. However, this approach introduces a new problem: the actual paths for which forwarding is available in the network depends in an interrelated manner on the route computation algorithms used at each node. For example, if a MPC router keeps only a subset of available paths, it may happen that forwarding along a path does not exist because a downstream MPC router does not keep that path in its forwarding table. We propose a fully distributed method of ensuring consistency between routers and to enable the discovery of all possible valid multi-paths in the network. We refer to this as the *multi-path validation* algorithm.

To illustrate our approach, we use the k-shortest paths algorithm (See for example [4], [12], [19], [23]) to compute *up to* k loop-free routes at MPC routers. If *all* the routers in the network are MPC routers, compute k shortest paths to each destination, and have the same value of k, the packet forwarding will exist for all the computed paths. In other words, the locally computed paths are all valid, i.e., forwarding exists for every locally computed path and no further multi-path validation phase is necessary. However, if only a subset of routers are MPC, and/or different routers use a different value of k (possibly due to different speeds and/or memory capacity), packet forwarding may not be available on all the computed paths, and a validation phase is necessary.

In particular, forwarding along a path computed by router A will not exist if a downstream router B does not install the suffix path (i.e. suffix pathID) in its forwarding table. We define *valid path* as a path for which forwarding exists, i.e., every suffix path of this path is also valid. We propose a polynomial time algorithm to validate the computed paths locally at a router. Note again that the algorithm we propose is general and can be used to validate paths even if any other deterministic path computation scheme is used, so long as the computation also includes the shortest paths to any destination. The validation algorithm assumes that a router has knowledge of the path-computation scheme at other upgraded routers. We discuss how to implement this feature in OSPF/IS-IS in Section IV.

The computation of *up-to k, valid, loop-free shortest paths* is a two-phase algorithm. In first phase, a router *locally* computes k-shortest paths from itself to all other nodes in the network [12], [4], [23], [19]. A *validation computation* is done to find a set of valid paths. The validation algorithm proceeds as follows:

Knowing that other MPC routers also use a k-shortest

path computation algorithm, it computes the k_i-shortest paths for all the upgraded routers in the network with a value of k as advertised by the router. We assume that upgraded routers advertise the path computation scheme and its parameters in their router LSAs. To generalize, note that we can locally compute the paths for other MPC routers according to the path computation scheme advertised by the router. For non-upgraded or P-MPC routers, shortest paths are computed using Dijkstra's algorithm. Now all the computed paths (including those for other MPC routers) are sorted in ascending order of hop count.

Algorithm 2 shows the pseudo-code of the second phase, i.e., the validation part. It uses the simple recursive definition of a valid path, i.e. a path is valid if all its suffix paths are valid. We can break this up into a mathematical induction, where a k-hop path is valid if the corresponding (k-1)-hop suffix path is valid; and prove the base condition, i.e. the 1-hop suffix path to the destination is valid. All the 1-hop paths as computed above (locally and for other MPC routers) are *always valid* because there is a direct link between the relevant router and the destination of the path. Now, observe that any 2-hop path is valid if the corresponding *1-hop path suffix is valid*. For example, the path 1-2-3 is valid if the path 2-3 is valid and so on. So, the algorithm first processes all 1-hop paths in the sorted list of computed paths. Once these are validated (trivially), it sorts this sub-list of one-hop paths. For every 2-hop path, it then searches for the existence of the corresponding one-hop suffix path in the sorted and validated 1-hop path sublist. If the 1-hop path is not found, then the 2-hop path is immediately invalidated and removed from the list. Else the 2-hop path is validated. Once all the 2-hop paths are processed, the 2-hop path sublist is processed. Similarly, when a m-hop path is being processed, the algorithm merely looks up the corresponding (m-1)-hop path suffix in the sorted sublist of validated (m-1)-hop paths. The path is invalidated and removed if the (m-1)-hop path suffix is not found. By mathematical induction, the algorithm terminates and validates all the residual paths in polynomial time complexity.

In a N-node network with u upgraded routers, the complexity of first phase is given $uC(k) + (N - u)C(d)$ where, $C(k)$ denotes the complexity of computing k-shortest paths to all the nodes in the network, $C(d)$ denotes the complexity of Dijkstra's algorithm or any other scheme used to find the shortest paths to all the nodes. If k_i denotes the value of k for upgraded router i, the total number of paths, T, at the end of first phase is equal to $(N-u)+\sum_{i=1}^{i=u} k_i$. The complexity of the validation phase is $O(Tlog(T)\bar{h})$ where, $\bar{h}$ is the average hop count for the paths. The log(T) term arises due to searching for a suffix in the Map (see Algorithm 2). This can be achieved with a lower complexity by using specialized data structures such as hash-maps. In summary, Algorithm 2 is a general validation procedure that can be applied to validate paths computed using *any* deterministic path computation algorithm at MPC routers.

Algorithm 2 Algorithm for validating paths at a router with only some nodes supporting multi-path forwarding

```
All paths are stored in a data structure (routing_map).It has
a pair of hopcount and series of nodes which form the path
(PathString,eg 1-2-3) as primary key. Pathid is stored as a
value associated with each key.
MaxHops is the number of hops in the longest path
countHops is used as a counter for iterating from 2 to Max-
Hops
j is used for keeping track of the length of suffix to be
matched
Paths with hop count one are valid
temp_pair is a pair of hopcount and PathString. It is used as
a key to find a particular entry in the map.
routing_map.find(arg) returns true if the map entry corre-
sponding to arg is found
Initialize j ← 2
for countHops 2 to maxHops do
  for all i ∈ map do
    PathString ← path stored in i
    if hop count of i ≥ countHops then
      tempnodes ← last j nodes in PathString
      temp_pair.hopcount ← countHops-1;
      temp_pair.PathString ← tempnodes;
      if routing_map.find(temp_pair) == FALSE then
        delete i
      end if
    end if
  end for
  countHops+=1;
  j+=1;
end for
```

IV. OSPF/IS-IS Extensions

In this section, we summarize the extensions to OSPF/IS-IS to support the proposed scheme. Apart from the extensions discussed in this section, a 32-bit PathID field is required in the packet header which may be implemented as a new *routing options* header. Observe that the proposed per-packet overhead of 32-bits is smaller than a 128-bit IPv6 address, that is considered acceptable per-packet overhead.

The route computation algorithm (Dijkstra's algorithm) at upgraded routers must be replaced with a multi-path computation algorithm, such as DFS with partial upgrades (DFS-PU), k-shortest paths or any other path computation algorithm, and a validation algorithm (Algorithm 2). A

validation algorithm is not needed if the subset of routers which are upgraded use DFS-PU or all routers in the network are upgraded and keep exactly k-shortest paths with same value of k. However, the routers must keep the shortest path as the default path. Incoming packets with erroneous pathID are forwarded on the shortest paths and the pathID field set to zero.

The intra-domain forwarding tables at upgraded routers would have tuples *(Destination prefix, pathID, Next-Hop, out-pathID)*. The OSPF Link State Advertisements (LSAs) [13] can be extended with one bit to indicate whether the router is multi-path capable (MPC); the bit is not set by P-MPC routers. In our Linux/Zebra based implementation, we have used the eighth bit in the *LSA options* field of the router-LSA as the MPC bit. Also, since we allow different upgraded routers to compute paths differently, we need some bits to indicate the choice of route computation algorithm along with its parameters e.g. the value of k in k-shortest paths algorithm. Note that this assumes that a router has same value of k for all destinations. In our implementation, we have added 8-bits after the router type field in the LSA to indicate the value of k. Additional 4-bits can be used to indicate the route computation algorithm at a router.

V. Implementation and Simulation Results

In this section, we illustrate the working of the proposed framework using SSFNet [20] simulations and Linux implementation using MIT's Click Modular Router package [10]. We use the SSFNet simulations to illustrate the framework in larger network topologies. We use the Linux implementation to demonstrate the operation of the proposed scheme in a small topology. We demonstrate both the cases, i.e, when an upgraded router keeps all available paths and when it keeps only k-shortest paths.

A. SSFNet Simulation Results

We demonstrate that even with very few upgraded routers, many paths are available to the upgraded routers i.e. the MPC routers as well as the P-MPC routers. Availability of many paths at an upgraded router allows flexibility in traffic splitting and thus, enable the operator to use the bandwidth available on these paths.

Figure 6 shows a 33-node, 61 link topology where all the links have 1Mbps bandwidth and 0.005ms latency. The network is simulated as a single OSPF area. Table I shows the number of paths available when only two routers, 12 and 19, were upgraded. These results assume that a MPC router stores all the available paths. Table II shows the number of paths available when four routers, 7, 12, 19 and 27, were upgraded. The results show that even with a small number of upgraded routers, many paths are available to the "source" or P-MPC routers.

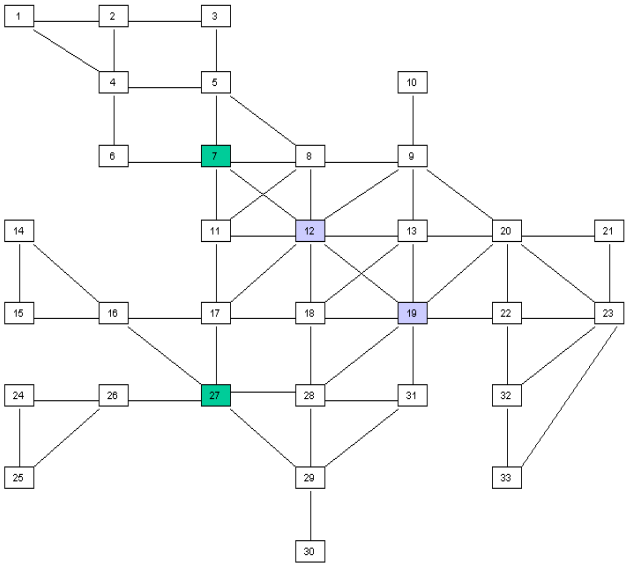

Fig. 6. 33-node topology used for SSFNet simulations

Router	No. of Paths	Avg. no. of paths per dst
12	204	6.4
19	233	7.3

TABLE I
NUMBER OF PATHS AVAILABLE AT UPGRADED ROUTERS 12 AND 19

Router	No. of Paths	Avg. no. of paths per dst
7	484	15.1
12	224	7.0
19	285	8.9
27	519	16.2

TABLE II
NUMBER OF PATHS AVAILABLE AT UPGRADED ROUTERS 7, 12, 19 AND 27

B. Linux Implementation Results

Figure 7 shows the topology of a simple validation experiment conducted on Utah's Emulab [11] testbed with the Linux Zebra version 0.92a of OSPF upgraded with our traffic engineering building blocks. The forwarding plane was implemented in Linux using MIT's Click Modular Router package [10]. Figure 7 also indicates the ip-addresses of various router interfaces and the link weights. The router ID is statically defined to be same as the ip-address of one of the router interfaces. However, for simplicity, we have chosen the smallest ip-address interface as the router ID.

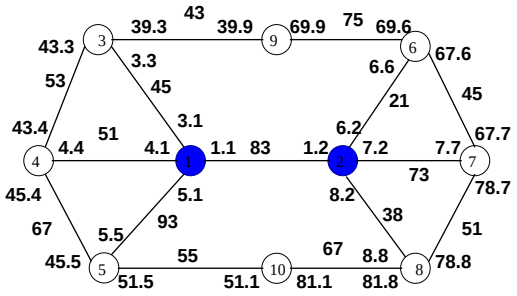

Fig. 7. Experimental Topology on Utah Emulab using Linux Zebra/Click Platforms

1) All Paths with Partial Upgrades: Table III illustrates a partial forwarding table at node 1 (IP address 192.168.1.1) for destination 3 (192.186.3.3). Note that the path string shown in Table III is only for the sake of illustration and is not stored in the actual routing table. The pathIDs are the (MD5 + CRC-32) hashes of the router IDs (i.e. IP addresses of nodes) on the path. For example, the pathID 2084819824 corresponds to a hash of the set of router IDs {192.168.1.1, 192.168.1.2, 192.168.6.6, 192.168.39.9, 192.168.3.3 }. The path suffix ID is the hash of the suffix set formed after omitting 192.168.1.1. If the path goes through other nodes which are not upgraded (e.g. 1-4-3), the suffix path ID is the hash of the suffix path starting from the next upgraded router on the path. In the case of the path 1-4-3, both nodes 4 and 3 are not upgraded, so the suffix path ID is zero.

Outgoing I/f	Path	PathID	PathSuffixID
192.168.1.1	1-2-6-9-3	2084819824	664104731
192.168.3.1	1-3	599270449	0
192.168.4.1	1-4-3	4183108560	0
192.168.5.1	1-5-4-3	1365378675	0

TABLE III
PARTIAL ROUTING TABLE AT 192.168.1.1 FOR DESTINATION 192.186.3.3

2) k-Shortest Paths with Partial Upgrades: In this section we illustrate, using the Linux implementation, the case when the upgraded routers compute up-to k-shortest paths. We consider the case when different upgraded routers used a different value of k.

Consider the 10-node topology shown in Figure 7. This topology was setup in the Emulab network. We assume that the routers 192.168.1.1 and 192.168.1.2 are upgraded with k equal to 3 and 2 respectively. The results are presented to verify the correctness of the "validation phase" (Algorithm 2). Tables IV, V show respectively part of the routing tables at 198.168.1.1 for destinations 198.168.6.6 and 198.168.8.8 respectively. Tables VI, VII show the corresponding entries at router 198.168.2.2. For destination 198.168.6.6 the router 198.168.1.1 finds 3 paths, all of which are valid as two paths have next-hop 198.168.2.2 and router 198.168.2.2 keeps 2 shortest paths. For destination 198.168.8.8, the router 198.168.1.1 computes 3-paths, 1-2-8, 1-2-6-7-8, 1-2-7-8. The path 1-2-7-8 is invalidated in the "validation phase" as router 198.168.2.2 only keeps 2 paths (2-8, 2-6-7-8). Note that the *Path* string is shown in tables IV-VII for the purpose of explanation.

Path	PathID	Next-hop	out-PathID
1-2-6	1989316858	192.168.1.2	3491782861
1-2-7-6	656924081	192.168.1.2	3645081405
1-3-9-6	534784006	192.168.3.3	0

TABLE IV
PART OF ROUTING TABLE AT 192.168.1.1 FOR DESTINATION 192.186.6.6

Path	PathID	Next-hop	out-PathID
1-2-8	3654096761	192.168.1.2	1973392862
1-2-7-6-8	1777786090	192.168.1.2	2123671348

TABLE V
PART OF ROUTING TABLE AT 192.168.1.1 FOR DESTINATION 192.186.8.8

Path	PathID	Next-hop	out-PathID
2-6	1973392862	0.0.0.0	1973392862
2-7-6	2123671348	192.168.7.7	2123671348

TABLE VI
PART OF ROUTING TABLE AT 192.168.2.2 FOR DESTINATION 192.186.6.6

Path	PathID	Next-hop	out-PathID
2-8	3491782861	0.0.0.0	0
2-6-7-8	3645081405	192.168.6.6	0

TABLE VII
PART OF ROUTING TABLE AT 192.168.2.2 FOR DESTINATION 192.186.8.8

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed a connectionless approach to intra-domain traffic engineering. The proposed approach allows routing along explicit paths without requiring a signaling protocol and with only partial network upgrades. In particular, we propose to use a hash of the path in a fixed-size *routing header* which is carried by the packet. The main idea is to use a *global path identifier* (pathID) that is based on ip-addresses and/or link weights (globally known due to link-state nature of OSPF and IS-IS) instead of local identifiers, i.e. labels, in MPLS. This

allows different routers to compute pathIDs consistently without the use of a signaling algorithm. The forwarding is based on longest destination prefix match followed by an exact pathID match. We have developed algorithms to compute paths available at a router under partial upgrade assumptions. The Linux implementation and simulations illustrate different aspects of the proposed scheme.

Future work includes the demonstration of a range of practical traffic engineering applications using our framework. The impact of using heterogeneous route computation algorithms on the TE capabilities needs further research. We also plan to map interesting applications like interactive multimedia and large-file-transfer to statistically multiplex the network using the multi-paths to get a larger multiplexing gain out of the network. In this paper, our focus was on a single-area OSPF routing domain. But the work can be easily extended to multiple areas, if we assume that the PathID is re-initialized at area boundaries. Our ongoing work extends the framework to inter-domain TE and multi-exit routing in autonomous systems by considering AS numbers or exit ASBRs as node-identifiers. We are also beginning to work with key representatives in the IETF and IRTF to consider large-scale trials and potential standardization of this framework.

REFERENCES

[1] J. Chen, P.Druschel, D.Subramanian, "An Efficient Multipath Forwarding Method," in *INFOCOM'98*, March, 1998.

[2] D. Coppersmith and and S. Winograd, " Matrix Multiplication via Arithmetic Progression," *ACM Symp. on Theory of Computing (STOC)*, 1987, pp.1-6.

[3] T.H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "*Introduction to Algorithms*", The MIT Press, McGrawHill Book Company, Second Edition, 2001.

[4] D. Eppstein, "Finding the k shortest Paths," Proceedings of 35th IEEE Symposium on Foundations on Computer Science, pp. 154-165, 1994.

[5] B. Fortz, M. Thorup, "Internet Traffic Engineering by Optimizing OSPF Weights, in *Proceedings of the INFOCOM 2000*, pp. 519-528, 2000.

[6] B. Fortz, M. Thorup, "Increasing Internet Capacity Using Local Search," *Preprint*, 2000.

[7] B. Fortz, M. Thorup, "Optimizing OSPF/IS-IS Weights in a Changing World," *IEEE Journal on Selected Areas in Communications*, Vol. 20 No. 4, 2002.

[8] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," *IETF RFC 2992*, 2000.

[9] Hema T. Kaur, T. Ye, S. Kalyanaraman, K. S. Vastola, "Minimizing Packet Loss by Optimizing OSPF Weights using Online Simulation," Preprint, available from http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers-rpi.html, 2002.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, Vol. 18, No. 3, August 2000, pages 263-297.

[11] J. Lepreau, "The Utah Emulab Network Testbed," http://www.emulab.net/

[12] Ernesto Q. Vieira Martins, Marta Margarida B. Pascoal and Jos Luis E. Santos, "The K shortest loopless paths problem," *Research Report, CISUC*, July 1998, available from http://www.mat.uc.pt/ eqvm/cientificos/investigacao/r_papers.html.

[13] J. Moy, "OSPF Version 2," *IETF RFC 2328*, April 1998.

[14] North American Network Operators Group (NANOG) attendees, *Private communication*, June 2002.

[15] P. Narvaez, K. Y. Siu, "Efficient Algorithms for Multi-Path Link State Routing," *ISCOM'99*, Kaohsiung, Taiwan, 1999.

[16] K.G. Ramakrishnan, M.A. Rodrigues, "Optimal Routing in Shortest Path Data Networks," *Bell Labs Technical Journal*, January-June 2001.

[17] F. Roman, "Shortest Path Problem is Not Harder than Matrix Multiplication," *Information Processing Letters*, Vol. 11, 1980, 134-136.

[18] E. Rosen et al, "Multi-Protocol Label Switching Architecture," *IETF RFC 3031*, January 2001.

[19] R. Shier, "On Algorithms for Finding the k Shortest Paths in a Network," *Networks*, vol. 9, pp. 195-214, 1979.

[20] Scalable Simulation Framework (SSF) Network Models, available from *http://www.ssfnet.org*.

[21] H. Suzuki and F. A. Tobagi, "Fat bandwidth reservation scheme with multi-link and multi-path routing in ATM networks," In *Proceedings of IEEE INFOCOM*, 1992.

[22] D. Thaler and C. Hopps, "Multipath Issues in Unicast and Multicast Next-Hop Selection," *IETF RFC 2991*, 2000.

[23] D. M. Topkis, "A k shortest path algorithm for adaptive routing in communications networks," *IEEE Transactions on Communications*, Vol. 36, 1988.

[24] C. Villamizar, "OSPF Optimized Multipath (OSPF-OMP)," *Expired Internet Draft*, 1999. Available from http://www.ietf.org/proceedings/99mar/I-D/draft-ietf-ospf-omp-02.txt

[25] S. Vutukury and J.J. Garcia-Luna-Aceves, " A Simple Approximation to Minimum-Delay Routing," *SIGCOMM '99*, September, 1999.

[26] T. Ye, S. Kalyanaraman, "A Recursive Random Search Algorithm for Optimization of Network Protocol Parameters," Technical report, ECSE Department, Rensselaer Polytechnic Institute, 2001.

APPENDIX

Consider an arbitrary graph $G = (V, E)$, where V is the set or vertices or nodes in the graph and E is the set of edges denoted by (i, j) where $i, j \in V$. Let N denote the number of nodes in the graph, i.e., the cardinality of V. We characterize the graph by a probability mass function of the node out-degree, i.e. the probability that a node has k neighbors is given by $p(k)$, $1 \leq k \leq N-1$. Let $d(i)$ denote the out-degree of a node i. We also assume that a node is equally likely to have a link to any of the other $n-1$ nodes. Now, consider any two arbitrary nodes s and m. Given node s has i neighbors, the probability that a link exists between s and m is given by $i/(N-1)$. In other words,

$$P[(s, m) \in E | d(s) = i] = \frac{i}{N-1} \qquad (1)$$

Hence, the probability that a link exists between any two nodes s and m is given by

$$q = \sum_{i=1}^{N-1} \frac{i}{N-1} p(i) \tag{2}$$

Then, the probability that any k hop path exists in this arbitrary network can be obtained by considering a fully-connected network. In a fully-connected network,

$$\text{No. of } k \text{ hop paths} = \begin{cases} 1 & \text{for } k = 1 \\ \prod_{i=2}^{k}(N-i) & 2 \le k \le N-1 \end{cases} \tag{3}$$

A k hop path exists in this arbitrary network if each of the k links exist in the arbitrary network i.e. with probability q^k. The expected number of k hop paths, H(k), in the random graph between any two nodes s and d is given by

$$H(k) = \begin{cases} q & \text{for } k = 1 \\ q^k \prod_{i=2}^{k}(N-i) & 2 \le k \le N-1 \end{cases} \tag{4}$$

The expected number of paths from s to d is given by

$$S(N) = q + \sum_{k=2}^{N-1} \left(q^k \prod_{i=2}^{k} (N-i) \right) \tag{5}$$

To validate (5), we generated random graphs with different values of q and used DFS to find total number of paths between a source and destination. Average number of paths was computed by averaging the number of paths over all source and destinations. A total of 1000 topologies were generated an average number of paths computed. Table VIII compares average number of paths as obtained using (5) and from DFS for a 10-node network for different values of q. The results show that our model is a very good aproximation estimating the expected number of available paths.

q	Analysis	DFS	Relative Error
0.20	2.85	2.65	0.075
0.28	15.00	14.62	0.026
0.36	65.70	67.24	0.023

TABLE VIII

COMPARISON OF NUMBER OF PATHS FOUND USING ANALYSIS AND DFS FOR 10-NODE NETWORKS WITH ALL UPGRADED NODES

A. Average number of paths with partial upgrades

We now consider the case when only a subset of the routers in the network are upgraded to support multi-paths. Let the probability that an arbitrary router in the network is upgraded be given by p. As before, the probability that a link between any two nodes exists, q is again given by Equation 2. The number of paths between any source destination pair which now exist depends on the number of upgraded routers in the path and the exact position of these routers. For example, if the source node is upgraded, it has a choice of $N-2$ nodes for the next hop, assuming of course that links exists between then. However, in case it is not upgraded, the number of next hop choices is 1. This argument can also be extended to the other nodes in the path. Also, we not that the status of the penultimate router and the destination do not matter since the last hop constitutes a direct connection between them. For a N node network, the expected number of k hop paths, $H(k)$, is then given by

$$H(k) = \begin{cases} q & \text{for } k = 1 \\ q^k \Big[(1-p)^{k-1} + \\ p^{k-1} \prod_{j=2}^{k}(N-j) + \\ \sum_{i=1}^{k-1} p^{k-1-i}(1-p)^i S(N,k,i) \Big] & 2 \le k \le N-1 \end{cases} \tag{6}$$

where

$$S(N,k,i) = \sum_{j_1=2}^{k} \sum_{j_2=j_1+1}^{k} \cdots \sum_{j_i=j_{i-1}+1}^{k} \prod_{\substack{m=2 \\ m \ne j_1,\cdots,j_i}}^{k} (N-m) \tag{7}$$

In Eqn. (6), the term $(1-p)^{k-1}$ denotes the case where the first $k-1$ routers are not upgraded, in which case there is only one k-hop path, which exists with probability q^k. The second term in that expression considers the case where all the first $k-1$ routers are upgraded in which case this becomes to the fully upgraded scenario. The term $S(N,k,i)$ counts the number of k hop paths when there are i non-upgraded routers in the path for any given source destination pair in the N node network.