An Efficient Multipath Forwarding Method

Johnny Chen Peter Druschel Devika Subramanian
Department of Computer Science
Rice University
Houston, TX 77005
Email: {jc,druschel,devika}@cs.rice.edu
Phone: (713) 527-4664, Fax: (713) 285-5930

Abstract— **We motivate and formally define dynamic multipath routing and present the problem of packet forwarding in the multipath routing context. We demonstrate that for multipath sets that are *suffix matched*, forwarding can be efficiently implemented with (1) a per packet overhead of a small, fixed-length path identifier, and (2) router space overhead linear in K, the number of alternate paths between a source and a destination. We derive multipath forwarding schemes for suffix matched path sets computed by both de-centralized (link-state) and distributed (distance-vector) routing algorithms. We also prove that (1) distributed multipath routing algorithms compute suffix matched multipath sets, and (2) for the criterion of ranked k-shortest paths, de-centralized routing algorithms also yield suffix matched multipath sets.**

Keywords— **Routing, Multipath, Forwarding.**

I. Introduction

Multipath routing aims to exploit the connectivity of the underlying physical networks by providing multiple paths between source-destination pairs. Unlike in single shortest path routing, the originating node has a choice at any given time of potentially more than one path to a particular destination. In a connectionless network, we can have per-packet multipath routing where an originating node sends individual packets to a given destination on different paths. In a connection-oriented network, we can have per-connection multipath routing where an originating node uses different paths for different connections to the same destination and packets on each connection travel the same path.

Multipath routing has the potential to aggregate bandwidth, allowing a network to support data transfer rates higher than what is possible with any one path [22]. In connection-oriented networks with QoS guarantees, they reduce blocking probabilities [12]. Multipath routing algorithms are able to multiplex connectionless traffic, thereby proactively reducing routing oscillation and alleviating congestion [3, 10]. Finally, the use of independent paths can increase the reliability of packet delivery [12].

There are two challenges in designing adaptive multipath routing algorithms. The first is devising efficient distributed or de-centralized algorithms for computing good multipath sets between pairs of nodes in the network. The second challenge is the forwarding of packets. Since each node has more than one path to a given destination, packet forwarding procedures at each node need to correctly and efficiently forward packets along their intended paths. Our chief technical contribution in this paper is an efficient solution to multipath forwarding. We demonstrate that for an interesting class of multipath sets, called *suffix matched* path sets, we can use a fixed length path identifier per packet and have router space requirements proportional to the number of paths calculated.

The traditional use of multipath routing (e.g., in the OSPF protocol) has been limited to the exploitation of a very special class of multipath sets, i.e., paths of equal cost. The multipath routing algorithms we consider in this paper allow more general multipath sets and are computed dynamically to adapt to changes in the network. As an example, consider a multipath routing method that maintains at each node the three best paths (not necessarily of equal cost) between that node and each available destination in the network. The originating node in such a network can decide whether to send a packet or a set up a virtual connection to a particular destination along its 1st, 2nd or the 3rd best path.

The paper is organized as follows. In the next section, we formally define adaptive multipath routing and the associated data packet forwarding problem. We then present the *suffix matched* class of multipath sets, a class for which the forwarding problem can be solved efficiently. In Section II, we describe the basis of our new forwarding method for suffix matched path sets and show that it uses (1) a per packet overhead of a small, fixed-length path identifier, and (2) router space overhead linear in K, the number of calculated paths between source/destination pairs. In Sections III and IV, we derive multipath forwarding schemes for suffix matched path sets computed by the distance vector and link state algorithms. We also prove that (1) distance-vector algorithm compute suffix matched multipath sets, and (2) for the criterion of ranked k-shortest paths, the link state algorithm yields suffix matched multipath sets. We conclude with related work in forwarding and multipath routing and discuss current work on handling non-suffix matched path sets.

II. Multipath Routing and Forwarding

We model a network as a graph $G = (V, E)$, where V is the set of nodes in the network and E is the set of links or edges. Each link $(x_i, x_j) \in E$ has an associated positive cost $c_{x_i x_j}$. A path p in G is a list of nodes $(x_1, \ldots, x_n)$ such that $\forall i, 1 \leq i < n$, $(x_i, x_{i+1}) \in E$. Path p has cost $\sum_{i=1}^{n-1} c_{x_i x_{i+1}}$. No node appears more than once in a *simple* or loop-free path. To route packets between nodes in a network, each node $x \in V$ implements two functions:

$$h_x : V \to \Lambda$$

$$\mathcal{G}_x : \Lambda \to V$$

The *path selection* function h_x chooses a path identifier ψ (on a per-packet or per-connection basis) for a given destination node $d \in V$ from the set Λ. Path identifiers encode distinct paths from x to d; they are names for paths in the network. We use path identifiers to name paths so that we can use a fixed name ψ for a path, while having its composition change with time. The forwarding function $\mathcal{G}_x$ takes a path identifier $\psi \in \Lambda$ and provides the next hop, i.e., the node following x in the path that is encoded by the identifier ψ. As an example, if node x wishes to send a packet to node d on the path named ψ, it adds the label ψ to the packet, and forwards it to $\mathcal{G}_x(\psi)$: the next hop on the path named ψ from x to d.

Single shortest path routing is a special case where there is no need to have a path identifier set Λ. In this path setting, the path selection function is the identity function: $h_x(d) = d,\ d \in V$. This is because exactly one path p from x to d is used at any given time — there is no choice of paths.[1] By the Bellman optimality principle, the shortest path from any intermediate node n in p to node d is the subpath of p from n to d; thus we can uniquely identify the path p along any node in p by the destination address d. We can replace the path identifier argument by the destination and simplify the forwarding function in a single shortest path setting to be $\mathcal{G}_x : V \to V$ which identifies the next hop for a given destination. $\mathcal{G}_x$ is implemented by forwarding tables at node x. The selection function h_x is normally evaluated once for a given packet at the originating host/router.

In a multipath setting, an originating node x has a choice of several paths to a particular destination at any given time (i.e, the range of the function h_x is a set of size ≥ 1). The particular path $h_x(d)$ chosen by x for a destination d is a function of the specific objectives of the routing system as a whole. We will have little to say about these functions in this paper. Examples of objectives include: increasing end-to-end path capacity (bandwidth) [15], providing QoS guarantees [22], or reducing service blocking probabilities [12].

The forwarding functions $\mathcal{G}_x$, for each $x \in V$ need to respect the following constraint:

$$\forall \psi \in \Lambda \text{ such that } \psi \text{ identifies path} = (x_1, \ldots x_n),$$
$$\mathcal{G}_{x_i}(\psi) = x_{i+1}, 1 \leq i < n.$$

Efficiently enforcing the above constraint on the $\mathcal{G}_x$ functions forms the crux of the multipath forwarding problem.

To implement the $\mathcal{G}$ functions, we need a scheme for constructing path identifiers that unambiguously identify paths between nodes in a network. One common approach, called source routing, is to use the path description itself as its identifier. In this method, path identifiers are of variable length and the overhead of stamping individual packets with path identifiers increases as the size of the network grows and the lengths of paths between nodes increase. The implementation of the $\mathcal{G}$ functions at each node x_i, for $1 \leq i < n$ requires reading the path identifier on each packet and computing $\mathcal{G}_{x_i}(x_1, \ldots, x_n)$ to be x_{i+1}. No state information (e.g., forwarding table) is needed at intermediate nodes and complete path information is needed at the sending nodes.

In this paper we focus on encoding paths with short, fixed length identifiers. As an example, if K best paths are maintained between each source and destination pair in an N node network, a path can be uniquely identified by the triple (s, d, i), where $s, d, \in V$ and $1 \leq i \leq K$. The space required for all the forwarding tables in the network is $O(KN^2 * N) = O(KN^3)$ which is the size of the mapping from each path identifier to the next hop (there are KN^2 paths), and this is maintained by each node (there are N nodes) in the network. The space requirement of this method can be reduced by observing that a node only needs to maintain paths that pass through it. Let L be the average path length, then each path passes through L nodes on average. There-

[1] The actual identity of the shortest path between two nodes in the network may change as the network changes, but at any given time a node can route to a destination along exactly one path.

fore it suffices that each node maintains only KNL path identifiers on average, reducing the total space requirements on forwarding tables to $O(KN^2L)$. In contrast, single shortest path routing imposes a fixed overhead on a per packet basis (each packet is labeled with the destination address), and the forwarding tables in the network need $O(N^2)$ space.

Is it possible to forward packets efficiently in a multipath setting with packets annotated by short fixed-length path identifiers and with space overhead for forwarding tables no more than $O(KN^2)$, which is K times the cost for forwarding in single shortest path systems? We answer this question in the affirmative for a very interesting class of multipath sets called *suffix matched* path sets.

Suffix matched path sets: A path set P is *suffix matched* iff for all paths $(x_1, \ldots, x_n) \in P$, $\forall i, 1 \leq i < n$, $(x_i, \ldots, x_n)$ is also in P.

Consider the path set P consisting of the single shortest paths between every pair of nodes in a network. The Bellman optimality principle ensures that P is suffix matched. If the shortest path p from node x_1 to x_n is $(x_1, \ldots, x_n)$, then for $1 < i \leq n$, the shortest path from x_i to x_n is the subpath $(x_i, \ldots, x_n)$ of p.

Proposition: Packets in an N node network with a suffix matched multipath set P can be forwarded correctly with forwarding table space $O(KN^2)$ where no more than K paths are maintained between any pair of nodes.

Proof: Let the path identifier for path $p = (x_1, \ldots, x_n)$ from x_1 to x_n in a suffix matched multipath set P be $f_{x_1}(x_n)$. Let the path identifier for the suffix paths $p_i = (x_i, \ldots, x_n)$ for $1 < i \leq n$ be $f_{x_i}(x_n)$. Recall that all suffixes of p are in P by the definition of suffix matched multipath sets.

To implement path forwarding correctly, we need for node x_i to know that the suffix of its path $f_{x_i}(x_n)$ (the path $(x_{i+1}, \ldots, x_n)$) at its neighbor x_{i+1} is called $f_{x_{i+1}}(x_n)$. Therefore we need an identifier translation or swapping function $\mathcal{F}_{x_i} : \Lambda \rightarrow \Lambda$ at each node that maps the path identifier $f_{x_i}(x_n)$ of suffix path p_i to the path identifier $f_{x_{i+1}}(x_n)$ for $1 \leq i \leq n$. That is

$$f_{x_{i+1}}(x_n) = \mathcal{F}_{x_i}(f_{x_i}(x_n)) \text{ and}$$

$$x_{i+1} = \mathcal{G}_{x_i}(f_{x_i}(x_n)).$$

In other words, when node x_i receives a packet with identifier $f_{x_i}(x_n)$ from its neighbor x_{i-1}, it forwards it to node $x_{i+1} = \mathcal{G}_{x_i}(f_{x_i}(x_n))$ along with the new identifier $f_{x+1}(x_n) = \mathcal{F}_{x_i}(f_{x_i}(x_n))$. Each x_j , $i < j < n$, performs the same two operations until the packet reaches x_n.

Notice that the definition of a suffix matched path set in a distributed environment implies that if router x_0 computes the path $(x_0, \ldots, x_n)$, then router x_i, $\forall i, 0 < i < n$, x_i must also compute the path $(x_i, \ldots, x_n)$. This ensures that there exists a $\mathcal{F}_{x_i}$ implemented by x_i that is guaranteed to translate a path identifier of x_i to x_{i+1}'s corresponding path identifier.

The space requirements for the implementation of the mapping $\mathcal{F}$ is proportional to the total number of paths in the network, i.e. $O(KN^2)$. This is because each path $O(KN^2)$ is matched with at most one other path, therefore the matching table is at most $2KN^2$. The space requirement for the mapping $\mathcal{G}$ is simply the number of paths $O(KN^2)$ $\Box$.

In the transient state when suffix matched path sets are in the process of being computed at each router, correct path forwarding is not guaranteed. This is because transient path sets are not guaranteed to be suffix matched. In fact, this absence of forwarding guarantees during route transitions effects most distributed routing algorithms, whether single or multipath.

III. Distance Vector Extension

In this section, we describe a multipath extension to a Distance Vector (DV) based routing algorithm. We first review the basic DV algorithm and then extend the algorithm to compute multiple paths. We then prove that the new multipath DV algorithm computes suffix matched paths sets. Finally, we apply our forwarding method to the extended DV algorithm, show its space complexity, and prove its correctness.

A. The Distance Vector Algorithm

Distance Vector is a distributed adaptive algorithm that computes shortest path between all pairs of nodes in a dynamic network. It is based on a centralized method known as the Bellman-Ford [4, 8] algorithm, and is the basis of many practical routing algorithms in current use [18, 2, 20].

Routers implement the forwarding function $\mathcal{G}$ via a lookup of the *forwarding table*. Each entry in the DV forwarding table contains three elements: the destination address, the next-hop neighbor on the known least-cost path to the destination, and the cost of the known least-cost path to the destination. To forward a packet to its destination D, the router sends the message to the next-hop neighbor specified in the forwarding table entry for D. Every router on the path repeats this step. The destination address and the next-hop neighbor fields are used to evaluate $\mathcal{G}$ as part of the packet forwarding process. The cost field is used only by the DV algorithm.

The task of the distributed DV algorithm is to continually update the forwarding table in each router to ensure that packets travel on the shortest path to their destination. Routers construct their forwarding tables by exchanging path information with their neighbors in the form of *distance vector packets* (DVP). Distance vector packets are excerpts of a router's forwarding table. A DVP carries the identity of the originating router, and a list of distance vectors of the form (*dst_addr*, *cost*), taken from the originating router's forwarding table.

Let C_d^r be the least path cost to destination d recorded in r's forwarding table, let N_d^r be the next hop neighbor on the least cost path to d recorded in r's forwarding table, and let c_{rs} be the cost of the link from r to s. It is assumed that the path cost metric is additive, that is, the cost C of a path $(x_1, \ldots, x_n)$ is $C = c_{x_1 x_2} + c_{x_2 x_3} + \ldots + c_{x_{n-1} x_n}$. Initially, the forwarding tables in each router r are initialized as follows:

$$C_r^r = 0;\ \forall s : s \neq r, C_s^r = \infty$$

When a router r receives a DVP from a neighbor s, r updates its forwarding table as follows. For all destinations d in s's DVP,

$$if\ (C_d^s + c_{rs} < C_d^r\ or\ N_d^r = s)\ then\ C_d^r = C_d^s + c_{rs}$$

where $N_d^r = s$. Notice the cost added to C_d^s is c_{rs}, the cost from r to s. This addition reflects the correct path cost from r to d through s.

Routers exchange distance vector packets with all their neighbors periodically, or in response to significant changes in the forwarding table. It can be shown that after a bounded number of DVP exchanges following a topology or link cost change, all routers' forwarding tables will contain values reflecting the shortest paths to all destinations.

B. Methods of Calculating Multiple Paths in DV

The single shortest path DV algorithm can be extended to allow the calculation of multiple paths between each pair of nodes. For instance, to calculate the ranked k shortest paths to each destination, each router maintains in its forwarding table (and advertises in its DVPs) k entries for each destination. Upon receiving DVPs from its m neighbors, the router computes its own k best paths from the $m * k$ paths received from its neighbors.

Path selection criteria other than k ranked paths, such as link or node disjoint paths, can also be implemented in a DV environment. We show that any DV-based multipath routing algorithm that satisfies the following conditions calculates suffix matched path sets.

1. A router r may install a forwarding table entry with $N_d^r = s$ only upon receiving a corresponding distance vector for destination d from its neighbor s.
2. A router r never advertises a distance vector for destination d unless it has a corresponding forwarding table entry for d.

Proposition: The path set P calculated by a multipath DV routing algorithm that satisfies conditions 1 and 2 are suffix matched.

Proof: Assume the algorithm calculates a path $(x_0, \ldots, x_n) \in P$ that is not suffix matched. Then there must exist a router x_i, $0 < i < n$ such that $(x_{i-1}, \ldots, x_n) \in P$, but $(x_i, \ldots, x_n) \notin P$. That is, x_{i-1} has a forwarding table entry with $N_d^{x_{i-1}} = x_i$, but x_i does *not* have a forwarding table entry with $N_d^{x_i} = x_{i+1}$. Thus, either x_{i-1} has installed the forwarding table entry without receiving a corresponding distance vector from x_i, or x_i has advertised a distance vector without having a corresponding forwarding table entry. But, these possibilities are ruled out by conditions 1 and 2 $\square$.

C. Multipath DV Extensions

In this section, we show how the proposed multipath forwarding method presented in Section II can be applied to any DV based multipath algorithms that satisfies conditions 1 and 2, and thus calculates suffix matched path sets.

The implementation of functions $\mathcal{F}$ and $\mathcal{G}$, defined in Section II, in a DV environment is defined as follows. Each router assigns an identifier ϕ for each of its paths to a given destination. Let $\phi_{r,d}^k$ be the ID router r assigns its k^{th} path to destination d. Then, the path ID used by router r in our MPDV algorithm is $pid = [d,\ \phi_{r,d}^k]$. Adding ϕ and evaluating $\mathcal{F}$ and $\mathcal{G}$ during the packet forwarding process requires the following additions to the DV data structures.

- Forwarding table entries have two additional elements, ϕ_1 and ϕ_2. An entry in router r's forwarding table for the path $(r, s, \ldots, d)$ is now a five tuple $(d, C_{d,\phi}^r, s, \phi_1, \phi_2)$. ϕ_1 is r's identifier for the path $(r, s, \ldots, d)$ and ϕ_2 is neighbor s's identifier for the corresponding suffix path $(s, \ldots, d)$. Note that $N_d^r = s$. $C_{d,\phi}^r$ is the cost of r's ϕ path to d.
- DVP entries have one additional element, ϕ. A DVP originating from router r and advertising r's path to destination d with identifier ϕ_1 is now a three tuple $(d, C_{d,\phi_1}^r, \phi_1)$.

Let D be the set of destinations and Φ_d^r be r's set of path identifiers to destination d. In its

DVP, each router r advertises a distance vector $(d, C^r_{d,\phi_1}, \phi_1)$ corresponding to its forwarding table entry $(d, C^r_{d,\phi_1}, s, \phi_1, \phi_2)$, for each $d \in D$ and for each $\phi_1 \in \Phi^r_d$.

To install a route in its forwarding table upon receiving a distance vector $(d, C^s_{d,\phi}, \phi)$ from s, router r adds the entry $(d, C^r_{d,\phi_1}, s, \phi_1, \phi)$, where ϕ_1 is chosen by r to be unique in Φ^r_d, i.e., unique in its set of path identifiers to destination d. For instance, in the DV multipath algorithm for k ranked paths, r's identifier for the i^{th} path to destination d with rank k can be chosen as k. C^r_{d,ϕ_1}, the cost of this path from s to d is calculated as $C^s_{d,\phi_1} + c_{rs}$.

The functions $\mathcal{F}$ and $\mathcal{G}$ can now be implemented directly via lookup of the forwarding table. For path ID $[d, \alpha]$, node r evaluates $\mathcal{F}_r([d, \alpha])$ as $[d, \beta]$ via lookup of the forwarding table entry for d and local identifier $\phi_1 = \alpha$, with $\beta = \phi_2$. Similarly, r evaluates $\mathcal{G}_r([d, \alpha])$ as $N^r_{d,\alpha}$ via lookup of the same forwarding table entry. A matching forwarding entry is guaranteed to exist when evaluating $\mathcal{F}$ and $\mathcal{G}$ because the path sets are suffix matched.

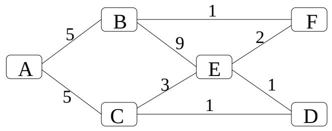
The additional storage required for the MPDV forwarding method is two path identifier fields in each forwarding table entry and one in each DVP entry. Let K be the maximal number of paths calculated to each destination, then the storage required for path identifiers is $\lceil(\log_2(K))\rceil$ bits. The additional entries in the forwarding tables and DVP for MPDV correspond to the number of alternate paths being calculated. That is, the storage overhead is a factor of $O(K)$ more than single shortest path DV. The aggregate router storage complexity of MPDV is therefore $O(KN^2)$.

D. *MPDV Example*

The graph in Figure 1 represents a network and the number above each link represents the cost of that link[2]. In table below are forwarding tables of selected routers to node F. The forwarding tables are computed using the Distance Vector multipath extension algorithm. For illustrative purposes, the path calculation algorithm calculates all paths with cost less than 10. Notice that this path calculation algorithm does not exclude paths that traverse a node more than once. The number of paths to any destination is at most 4, ie. $K = 4$. We assume this DV version implements split-horizon.

Forwarding table construction. The process of calculating paths to F begins by F sending a DVP to

[2]For simplicity, we assume the link costs are symmetric in this example. We note that MPDV also functions correctly with non-symmetric link costs.



E's forwarding table to F

cost	Φ_1	Φ_2	next-hop
2	1	1	F
10	2	1	B
7	3	2	D
7	4	2	C

D's forwarding Table to F

cost	Φ_1	Φ_2	next-hop
3	1	1	E
6	2	2	C
8	3	4	E

C's forwarding table to F

cost	Φ_1	Φ_2	next-hop
5	1	1	E
4	2	1	D
10	3	3	E
9	4	3	D

Fig. 1 Example of MPDV.

its neighbors (E and B). The DVP contains the path $(F, 0, 1)$, where F is the destination address, 0 is the cost for F to reach F, and $1 = \phi^1_{F,F}$. When neighbor E receives this DVP, E adds the cost $C_{EF} = 2$ to every element in F's DVP. Since the path to F has cost $2 = 0+2$, it satisfies the path calculation criteria. E keeps this path and augments its forwarding table with the entry (F, 2, 1, 1, F): F is the destination address, 2 is the cost of this path to F, the first $1 = \phi^1_{E,F}$ (E's name for this path to F), the second $1 = \phi^1_{F,F}$ (taken from F's DVP), and $F = N^E_{F,\phi^1_{F,F}}$ is this path's next-hop neighbor.

Next time E sends out its DVP, it will contain the path entry (F, 2, 1) advertising a path to F with cost 2 and $\phi^1_{E,F}$ of 1. When neighbor C receives this DVP containing this path to F, C augments its forwarding table with the path (F, 5, 1, 1, E) because this path is admissible under the path calculation criteria. In this example, C also chose the name of this path to be $\phi = 1$. This process of path computation continues until all routers have admissible paths to all other routers. The convergence time is proportional to the longest path between any two nodes. The complete forwarding tables to F for routers E, D, and C are shown in Figure 1.

Packet forwarding. Suppose router D wishes to send a packet to router F and D decides to send on its third best path, the one with cost 8. According to the algorithm, D tags the packet with the path ID $[F, 4]$, where F is the destination address and $4 = \phi_2$ in the forwarding entry, and forwards the packet to the next-hop neighbor E. On receiving this packet, E looks up its forwarding table and matches F and ϕ_1 with the packet's path ID. The matching entry is the last entry in E's forwarding table. Once found, E replaces the packet's path ID with [F, 2] (E's ϕ_2 for this path is 2) and forwards the packet to C. That is $\mathcal{F}_E([F, 4]) = [F, 2]$ and $\mathcal{G}_E([F, 4]) = C$. C does the same lookup and forwards the packet back to D with path ID $[F, 1]$ (here $\mathcal{F}_C([F, 2]) = [F, 1]$ and $\mathcal{G}_C([F, 2]) = D$). Following the algorithm, D again performs the lookup and forwards the packet to E

with the path ID $[F, 1]$. On receiving this packet, E finally forwards the packet to F with the ID $[F, 1]$.

For simplicity, the MPDV path calculation algorithm presented in this example did not eliminate finite looping paths. Since our path forwarding method ensures forwarding on all calculated paths, the algorithm will forward packets on finite-looping paths (eg. path (D, E, C, D, E, F)) as well as non-looping paths (eg. (D, E, F)). There are well known methods of ensuring loop-free path calculation in distance vector, refer to [14] for more detail.

IV. Link State Extension

In this section, we discuss the use of the proposed multipath forwarding method in conjunction with link-state (LS) routing algorithms. The link state algorithm, like the DV algorithm, is widely used and well understood. However, unlike DV, it relies on a centralized route calculation algorithm. A topology broadcast mechanism ensures that each router knows the current state of the entire network (i.e., topology and link cost), and routes are calculated in a centralized manner in each router. We begin with a description of the single path LS algorithm, discuss multipaths extension, and then show how our multipath forwarding method applies.

A. The Link State Algorithm

The Link State algorithm is the basis of many widely used routing algorithms. Protocols based on this algorithm are used in the Internet and in ATM networks [17, 1, 19].

In the Link State routing algorithm, every router periodically broadcasts (via flooding) its local connectivity. This information is contained in a link state packet (LSP) and consists of the router's ID, a list of its neighbor's IDs, and the cost of the connecting links. After a round of broadcasts by each router, every router has information about the entire network's topology. Each router then performs a shortest-path spanning tree computation rooted at itself. From the computation of this tree, every router knows the shortest path to all destinations in the network. This knowledge is encoded in the router's forwarding table. A LS forwarding table is a list of tuples of the form (*dst_addr*, *next-hop*), where *dst_addr* is the address of the destination and *next-hop* is the neighboring router on the shortest path to the destination.

Routers forward packets in the same way as in the DV protocol. Packets are tagged with a path ID consisting only of their destination address. Routers forward a packet by looking up their forwarding table using the packet's destination address and forwarding the packet to the specified next-hop neighbor.

B. Multipath LS Extensions

In this section, we show how the proposed forwarding method applies to link state multipath algorithms that calculate suffix matched paths. Since routers in the link state algorithm have access to the entire network topology, any centralized graph algorithm for computing multiple paths can be used. We are particularly interested in algorithms that calculate suffix matched path sets. As we will show, the k ranked paths algorithm [6], and the initial link disjoint k paths algorithm [23] are in this class. For such algorithms, our multipath forwarding algorithm can be directly applied.

As in MPDV, let $\phi^k_{r,d}$ be the identifier that router r assigns its k^{th} path to destination d. The path ID used by router r is of the form $pid = [d, \phi^k_d]$. Forwarding table entries have two additional elements, ϕ_1 and ϕ_2. An entry in router r for the path $(r, s, \ldots, d)$ is a quadruple (d, s, ϕ_1, ϕ_2). ϕ_1 is r's identifier for the path $(r, s, \ldots, d)$ and ϕ_2 is neighbor s's identifier for the corresponding suffix path $(s, \ldots, d)$. Note that $N^r_{d,\phi_1} = s$.

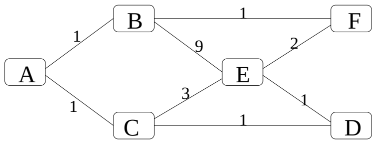
Forwarding entries are installed by the centralized link-state path calculation process. As in MPDV, ϕ_1 is chosen to be unique in r's set of paths identifiers for destination d (ie. Φ^r_d). To determine ϕ_2 router r calculates the next-hop neighbor router s's paths set for destination d. Since the path sets are assumed to be suffix matched, s's path set to d is guaranteed to include the suffix path $(s, \ldots, d)$. Then, r sets ϕ_2 equal to s's identifier ϕ_1 for that suffix path.

As in MPDV, the functions $\mathcal{F}$ and $\mathcal{G}$ can now be implemented directly via lookup of the forwarding table. Node r evaluates $\mathcal{F}_r([d, \alpha])$ as $[d, \beta]$ via lookup of the forwarding table entry for d and local identifier $\phi_1 = \alpha$, with $\beta = \phi_2$. Similarly, r evaluates $\mathcal{G}_r([d, \alpha])$ as $N^r_{d,\alpha}$ via lookup of the same forwarding table entry.

The number of messages exchanged by MPLS routers is exactly the same as in the traditional LS algorithm. The storage overhead per router is $O(NK)$ for the forwarding tables, where N is the number of destinations and K is the number of paths to each destination. In addition, our multipath forwarding method requires each router to calculate each of their neighbor's K paths as well. Therefore, forwarding increases the computational complexity of the multipath route calculation used in the routers by a factor m, where m is the number of neighbors.

C. Example of LS Extension

Consider the network shown in Figure 2. The path calculation algorithm calculated three paths



A's forwarding Table to F

Dst	next-hop	Φ_1	Φ_2
F	B	1	1
F	C	2	2
F	C	3	3

C's forwarding table to F

Dst	next-hop	Φ_1	Φ_2
F	A	1	1
F	D	2	1
F	E	3	1

E's forwarding table to F

Dst	next-hop	Φ_1	Φ_2
F	F	1	1
F	C	2	2
F	D	3	1

Fig. 2 Example of the multipath Link State Algorithm.

from router A to F: (A, B, F), (A, C, D, E, F), (A, C, E, F), with costs 3, 5, 6 and names $\phi^1_{A,F} = 1$, $\phi^2_{A,F} = 2$, $\phi^3_{A,F} = 3$ respectively. Suppose A sends a packet to F using its third best path. Following our algorithm, A tags the packet with the ID $[F,3]$, where F is the destination address and 3 is the ϕ_2 for that forwarding table entry. A then forwards the packet to C, the next-hop node for this path. When C receives this packet, it searches its forwarding table to find an entry with the destination address and $\phi_1 = 3$. C's third forwarding entry matches this path ID. C then replaces the packet's old path ID with $[F,1]$ and forwards the message to E, the next-hop element of the matching entry. That is, $\mathcal{F}_C([F, 3]) = [F, 1]$ and $\mathcal{G}_C([F, 3]) = E$. Similarly, E matches the ID $[F,1]$ and forwards the packet with the ID $[F,1]$ to F.

D. Suffix Matched Multipath Routing Algorithms

We presented an efficient forwarding method for link state based multipath calculation algorithms that calculate suffix matching path sets. Several existing multipath algorithms generate suffix matched paths. Here we show that the k ranked path algorithm [6] produces suffix matched paths.

Proposition: The unconstrained k shortest (ranked) simple paths algorithm [6, 7] produces suffix matched path sets.

Proof: The unconstrained k shortest path algorithm uses the following invariant for computing multiple paths between all pairs of nodes in the network. A path of rank k between node x_1 and x_n is composed of subpaths all of whose ranks are $\leq k$. Since each node maintains k paths between itself and every other node, all subpaths of a k ranked path between x_1 and x_n are guaranteed to exist in the multipath set computed by the algorithm. □

The k initial link disjoint paths algorithms proposed by Topkis [23] also produces suffix matched path sets. In addition, we have already shown in Section III that any DV based multipath algorithm that satisfies two straightforward conditions produces suffix matched paths.

E. Non-Suffix Matching Paths in LS

We have shown above that several important multipath calculation algorithms produce suffix matched paths. However, many other paths selection algorithms, including algorithms that maximize flow between a pair of nodes, are not guaranteed to produce suffix matched paths. Unfortunately, our efficient multipath forwarding method cannot be directly applied to such algorithms.

We have conducted experiments with two important non-suffix matched path selection algorithms. The first algorithm computes k ranked paths subject to the constraint that the paths are link disjoint. The second algorithm calculates k ranked paths subject to capacity constraints. For this purpose, link costs reflect the capacity (i.e., bandwidth) of the link, and path costs correspond to the cost of the link on the path with least capacity. Calculations suggest that on networks with a topology typical for wide-area networks, these algorithms produce paths sets that are over 90% suffix matched. We are currently exploring extensions to our forwarding method that can take advantage of this fact and permit efficient forwarding for such multipath sets.

V. Related Work

One of the earliest multipath routing networks is the IBM SNA network [11]. The network uses global, fixed length path IDs, corresponding to different types of services. The multiple paths and path IDs require manual entries to forwarding tables. Sidhu et al. [21] present a multipath algorithm based on the distance vector algorithm. The path IDs used by this algorithm are a compressed listing of the paths, and are of variable length.

Bahk and Zarki [3] present a centralized, exponential algorithm for calculating multiple paths to reduce network load. The work does not address issue of path forwarding. Li [16] proposes a multipath routing scheme (called bifurcated routing) based on flooding. The path ID used in this algorithm is also of variable length. Currently, the only multipath routing protocol in widespread use is OSPF [5]. This algorithm uses alternate paths using only the destination address as the path ID. This implies that only multiple paths of equal or near equal cost can be used.

Optimizations to source routing have been proposed in [9, 13]. The idea is to reduce the per packet overhead by initially using source routing to establish local path identifiers, and subsequent packets only need to carry the established fixed length path

ID. Given that the path recalculation occurs infrequently, the amortized packet overhead is constant. However, this method has the memory requirement of $O(LN^2K)$, where L is the average path length, N is the number of nodes and K is the number of path calculated between nodes.

The basis of our forwarding method—namely the use of a local path identifier that is translated as a packet is forwarded through the network—has some commonalities with the way virtual circuit switching is implemented in ATM networks. However, the context in which the idea is applied and the goals are quite different. In ATM, the virtual circuit identifiers (VCI) are used to forward cells along a previously established connection (virtual circuit). Using local identifiers reduces the space requirements for the identifier in the cell header and minimizes necessary coordination among ATM switches. Our forwarding method, on the other hand, allows path forwarding without requiring additional setup. By exploiting suffix matching path sets through the use of local path identifiers, our method significantly reduces table space in the routers.

VI. Conclusion

Multipath routing has important advantages over single shortest path routing. Multipath routing is able to better exploit the underlying physical network resources, provide greater end-to-end throughput, reduce blocking probability in networks with QoS guarantees, increase reliability of packet delivery, and reduce route oscillation and congestion.

A difficult problem in multipath routing is the forwarding of packets on their intended path. Because each router individually decides where to forward packets, ensuring that packets travel a chosen path requires a form of agreement amongst all routers in the network. This agreement on paths is encoded in the form of a path ID.

In this paper, we presented a novel multipath forwarding method that uses small, fixed-length path identifiers and requires router state space proportional to the number of different paths for each destination. When compared to the general method for multipath forwarding with fixed-length identifiers, this reduces the router state space needed to enforce path agreement by a factor of L, where L is the average path length in the network. We define the *suffix matched* property of multipath sets, and show that many important multipath selection algorithms produce paths sets with this property. Our forwarding method exploits this property.

We apply our forwarding method to two well known classes of routing algorithms—the distributed distance vector algorithms, and the decentralized link state algorithms. For multipath extensions of DV, the storage and message overhead required is proportional to the number of extra paths discovered. In the LS multipath extension, no additional messages are needed. The additional computation and storage overhead for suffix matching path calculation algorithms are also proportional to the number of paths discovered.

References

[1] PNNI draft specification. ATM Forum 94-0471R13, 1994.

[2] G. Alkin and F. Baker. RIP version 2 MIB extension. RFC 1724, Xylogics, Inc., Cisco Systems, November 1994.

[3] S. Bahk and M. E. Zarki. Dynamic multi-path routing and how it compares with other dynamic routing algorithms for high speed wide area networks. In *Proceedings of SIGCOMM*, pages 53–64, 1992.

[4] R. Bellman. *Dynamic Prgramming.* Princeton University Press, Princeton, N.J., 1957.

[5] R. Coltun. OSPF: An Internet routing protocol. *ConneXions*, 3(8):19–25, 1989.

[6] J. A. de Azevedo, M. E. O. S. Costa, J. J. E. R. S. Madeira, and E. Q. V. Martins. An algorithm for the ranking of shortest paths. *European Journal of Operational Research*, 69:97–106, 1993.

[7] D. Eppstein. Finding the k shortest paths. In *Proc. 35th Symp. Foundations of Computer Science*, pages 154–165. Inst. of Electrical & Electronics Engineers, November 1994.

[8] L. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, N.J., 1962.

[9] M. L. Gardner, I. S. Loobeek, and S. N. Cohn. Type of service routing with loadsharing. In *IEEE/IECE Global Telecommunications Conference*, pages 1144–50, 1987.

[10] P. Georgatsos and D. Griffin. A management system for load balancing through adaptive routing in multiservice ATM networks. In *Proceedings of IEEE INFOCOM*, pages 863–870, 1996.

[11] J. Gray and T. McNeill. SNA multiple-system networking. IBM Systems Journal, Vol. 18, No. 2, 1979.

[12] K. Ishida, Y. Kakuda, and T. Kikuno. A routing protocol for finding two node-disjoint paths in computer networks. In *Internation Conference on Network Protocols*, pages 340–347, Nov. 1992.

[13] J. M. Jaffe, F. H. Moss, and R. A. Weingarten. SNA routing: Past, present, and possible future. *IBM Systems Journal*, 22(4):417–434, 1983.

[14] S. Keshav. *An Engineering Approach to Computer Networking.* Addison-Wesley Publishing Company, Reading, Mass., 1997.

[15] R. Krishnan and J. A. Silvester. Choice of allocation granularity in multipath source routing schemes. In *Proceedings of IEEE INFOCOM*, 1993.

[16] W. Lai. Bifurcated routing in computer networks. *Computer Communication Review*, 15(3):28–49, 1986.

[17] J. McQuillan, I. Richer, and E. C. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications*, COM-28(5):711–719, 1980.

[18] J. McQuillan and D. Walden. The ARPANET design decisions. *Computer Networks*, 1, 1977.

[19] J. Moy. The OSPF specification. RFC 1131, October 1989.

[20] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, T.J. Watson Research Center, IBM Corp., Cisco Systems, March 1995.

[21] D. Sidhu, R. Nair, and S. Abdallah. Finding disjoint paths in networks. In *Proceedings of SIGCOMM*, pages 43–51, 1991.

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

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