Interface Split Routing for Finer-Grained Traffic Engineering

Saqib Raza,

Department of Computer Science
University of California, Davis

Chen-Nee Chuah

Department of Computer and Electrical Engineering
University of California, Davis

Abstract

Legacy IP routing restricts the efficacy of traffic engineering solutions. This restriction stems from the constraint that traffic at a node must be uniformly split across all next-hop nodes corresponding to equal cost shortest paths to a destination [1]. Proposals that alleviate this constraint either completely overhaul legacy IP routing e.g. [2–5], or introduce complex control and/or forwarding plane components [6–8]. This additional complexity departs from the elegant simplicity of legacy routing protocols where statically optimized [9–14] link weights embed all traffic engineering semantics.

We present Interface Split Routing (ISR), which retains the basic forwarding and control mechanism of legacy IP routing. Furthermore, a set of link weights embed all traffic engineering semantics in ISR. However, ISR makes possible finer-grained traffic engineering by configuring *independent* sets of next-hops to a destination at each incoming interface. This lends itself well to modern router architectures where each incoming interface has its own forwarding table [15]. Consequently, at the aggregated node level, traffic to a particular destination may be non-uniformly distributed across next-hop nodes. Hence, ISR allows additional flexibility in routing traffic as compared to default IP routing while retaining its simplicity. We conduct simulation studies on representative ISP topologies to compare ISR with traditional link-weight optimized routing. ISR reduces the difference between optimal routing and weight-optimized routing by 50%.

Key words: Intra-Domain Routing, Traffic Engineering, Autonomous System, Link-State Routing Protocol, OSPF, IS-IS

Email addresses: sraza@ucdavis.edu (Saqib Raza), chuah@ucdavis.edu

1 Introduction

Traffic engineering constitutes an indispensable function in large autonomous systems, maximizing the operational efficiency of such networks [12]. Traffic engineering mechanisms are instrumental in achieving performance objectives with a given set of resources and deferring cost-intensive deployment of additional resources. The optimal solution to the traffic engineering problem yields an explicit set of paths across which the given set of demands need to be routed [1]. We refer to routing across this optimal set of explicit routes as *optimal routing*. The extent to which traffic can actually flow across such a designated set of routes is a function of the underlying routing protocol and the precise forwarding mechanism.

The destination-based routing paradigm of legacy IP routing does not support routing across explicit routes. Legacy intra-domain routing protocols like OSPF [16] and IS-IS [17] involve assigning and disseminating integer link weights for each directed link in the network. Data forwarding in the context of legacy IP routing must obey the following pair of stipulations:

- At any given node x, traffic destined to a destination node d can be forwarded to a node y, *if and only if* there exists a path ρ such that ρ is a shortest path from x to d with respect to the assigned link weights, and y immediately follows x in ρ. We refer to y as a *next-hop* from x to d, and label this constraint the *Shortest Path (SP)* constraint.
- In case there exist more than one next-hops from x to d, all traffic at x destined to d must be distributed uniformly across the next-hops. This constraint is referred to as the *Equal Cost Multi Path (ECMP)* constraint.

Legacy IP routing draws upon ECMP as a heuristic to load balance traffic across multiple next-hops. The challenge is to ascertain the degree to which traffic can be engineered while adhering to the legacy SP and ECMP implementations. A retinue of solutions [9–14] have been proposed that are geared towards finding link weight assignments that achieve or approach the performance of optimal routing. These solutions work in conjunction with ECMP and assume knowledge of the traffic matrix. However, the difference between optimal routing and link-weight-optimized routing using legacy routing protocols may be significant [14]. Other solutions necessitate deployment of new technology [2–5], or introduce additional control and/or forwarding plane components and extra management complexity [6–8].

We present *Interface Split Routing (ISR)*, that leverages the fact that modern router architectures maintain separate forwarding tables for each incoming interface [15]. However, these forwarding tables are *interface-independent*, i.e.

(Chen-Nee Chuah).

forwarding tables at different incoming interfaces of a router still yield the *same* set of next-hops for a particular destination. ISR uses *multiple* weights per link to configure *interface-specific* routing tables that yield *different* sets of next-hops to the same destination. This serves to relax the ECMP constraint allowing greater flexibility in engineering traffic.

In this paper we demonstrate how *interface-specific* forwarding tables can be configured while retaining the basic forwarding and control mechanism of legacy IP routing. Specifically, we show how statically assigned link weights can embed all traffic engineering semantics required for ISR. An important contribution of this work is that it sets up the weight assignment problem such that *any* algorithm for computing optimal link weights for legacy IP routing can be adapted to compute multiple link weights for ISR. We prove that our scheme results in judicious configuration of *interface-specific* forwarding tables, and present simulation results that showcases the performance of ISR.

The rest of this paper is organized as follows: Section 2 formalizes the optimal routing problem and discusses solutions proposed in previous literature. Section 3 introduces the ISR scheme in the context of optimal routing and presents the general framework for ISR. Sections 4 & 5 demonstrate how multiple link weights per link are calculated and used to compute interface-specific forwarding tables. Section 6 gives the results of our simulation experiments and we put forth our conclusions in Section 7.

2 Background and Related Work

2.1 The Optimal Routing Problem

Traffic engineering involves routing a given set of traffic demands on a network to optimize user performance and network resources. Traffic demands are provided in the form of a traffic matrix that represents traffic intensity at the granularity of ingress-egress pairs. We let $G(V, E)$ represent the given network. Furthermore, let $\{\Omega_{uv}\}_{(u,v)\in V\times V}$ be the given traffic matrix. We define c_{ij} to be the capacity of link $(i, j) \in E$, and X_{ij}^{uv} to be the fraction of traffic between ingress-egress pair $(u, v) \in V \times V$ that traverses $(i, j) \in E$. The utilization of a link (i, j) is given by $\mu_{ij} = {c_{ij}}^{-1} \times \sum_{(u,v)\in V\times V} (X_{ij}^{uv} \times \Omega_{uv})$. Routing must obey the flow conservation constraints:

$$\sum_{i:(i,j)\in E} X_{ij}^{uv} - \sum_{k:(j,k)\in E} X_{jk}^{uv} = \begin{cases} 0 & \text{if } j \neq u \text{ and } j \neq v \\ -1 & \text{if } j = u \\ 1 & \text{if } j = v \end{cases} \qquad \forall (u, v) \in V \times V \quad (1)$$

$$0 \leq X_{ij}^{uv} \leq 1 \qquad \forall (i,j) \in E, \quad \forall (u,v) \in V \times V \tag{2}$$

The more universal traffic engineering objective is to route all the demands while optimizing some measure of link utilization. One such objective is minimizing the utilization of the most congested link, referred to as the *min-max-utilization* objective. The optimal routing problem with the min-max-utilization objective can be formulated as an LP [7]. Fortz et. al propose an alternative linear programming formulation that seeks to minimize a piecewise increasing linear convex envelope of a non-linear link cost function. The reader can refer to [14] for details of the *Fortz & Thorup* metric.

The LPs solve for $\{X_{ij}^{uv}\}_{(i,j)\in E,(u,v)\in V\times V}$. In order to realize such a traffic distribution through legacy IP routing, link weights should be assigned that achieve the distribution in the presence of the SP and ECMP constraints. The SP constraint does not restrict us due to an important result [1] from LP Duality theory that states that *given a set of explicit paths, there exists a link weight assignment, such that either the explicit paths are equivalent to shortest path routing with respect to the weights, or there exists another set of paths which are shortest paths with respect to the assigned weights, and routing along this set of paths results in link utilization for each link being less than or equal to the values corresponding to the original paths.* On the other hand, ECMP is a restrictive constraint and implies that explicit routing may not be achievable by shortest path routing with statically assigned link weights. We will provide an illustrative example in Section 3.1.

2.2 Existing Solutions

One of the general class of solutions put forth in previous literature is *optimized link weight assignment* [9–14]. These solutions are geared towards intelligently computing link weights for which shortest path routing with ECMP yields the best traffic engineering performance. They assume knowledge of the traffic matrix. One of the most cited works in this domain is [14]. The authors of [14] demonstrate that statically assigning link weights using a local search heuristic performs close to optimal routing, for topologies that are representative of actual networks, . However, they also prove that for certain networks, the difference between optimal routing and *any* link-weight-optimized routing using legacy routing protocols may be significant. This difference is attributable to the simplifying ECMP heuristic for load balancing [1]. Optimal routing can be realized if it were possible to configure arbitrary distribution of traffic across the set of next-hop nodes [1]. A number of proposals have been presented towards the end of achieving, or more closely approximating, optimal routing. Some of these proposals necessitate deployment of new technology such as MPLS, ATM, or Frame Relay PVCs [2–5]. For instance, MPLS with

its characteristic segregation between the control and forwarding plane, affords the ability to establish virtual connections between two points on an IP network, maintaining the flexibility and simplicity of an IP network while exploiting the ATM-like advantage of a connection-oriented network. Ingress routers of an MPLS network can classify packets into forwarding equivalence classes and encapsulate them with labels before forwarding them along precomputed paths. These label switched paths can be set up to realize a desired set of explicit routes.

Other solutions [6–8] have been proposed to leverage the widespread deployment of legacy IP routing protocols and avoid an overhaul of the underlying technology. They, however, introduce additional control and/or forwarding plane components and extra management complexity. For instance, [7] performs centralized flow optimization and requires traffic split information to be disseminated across the network at regular intervals. [6] assumes knowledge of traffic intensity information at the granularity of destination prefixes and involves separately configuring the set of next-hops on a per-prefix basis at each node. [8] requires dynamic link load information to be disseminated across the network and changes to the forwarding mechanisms in the router's data path. These proposals do achieve relatively fine-grained routing of traffic. However, their additional complexity represents a significant departure from the simple elegance of legacy routing protocols with optimized link weights, wherein statically assigned link weights embed traffic engineering semantics.

3 Interface Split Routing

3.1 Motivation for ISR

We now discuss the general idea and motivation behind ISR. Most IP networks use link-state protocols such as OSPF [16] and IS-IS [17] for intra-domain routing. In such networks, every link is assigned a weight (or cost) and traffic between nodes is routed along minimum cost paths. These networks are characterized by the destination-based forwarding paradigm. A routing lookup comprises of a longest-prefix match on a destination IP address to determine the set of nodes to which a packet may be forwarded. Since a prefix corresponds to a unique egress node in the network, this set comprises of nodes that are *next-hops* from the node performing the lookup to the egress node.

Formally, let Λ_{xd} be the set of next-hops from x to d with respect to the assigned link weights. Each destination prefix ρ has an egress node denoted by $e^\rho \in V$. Let λ_x^ρ be the set of next-hops configured for prefix ρ at node x. Shortest path routing necessitates that $\lambda_x^\rho \subseteq \Lambda_{xe^\rho}$. In reality, ECMP only

implies that all traffic with prefix ρ at node x gets uniformly distributed across the set of next-hops λ_x^ρ. However, legacy IP routing sets $\lambda_x^\rho = \Lambda_{xe^\rho}$. It follows that all traffic from x to d gets uniformly distributed across the set of next-hops Λ_{xd}.

We stated in Section 2, that ECMP restricts us such that optimal routing may not be achievable by shortest path routing with statically assigned link weights. We now present an example to illustrate this claim. Figure 1 shows

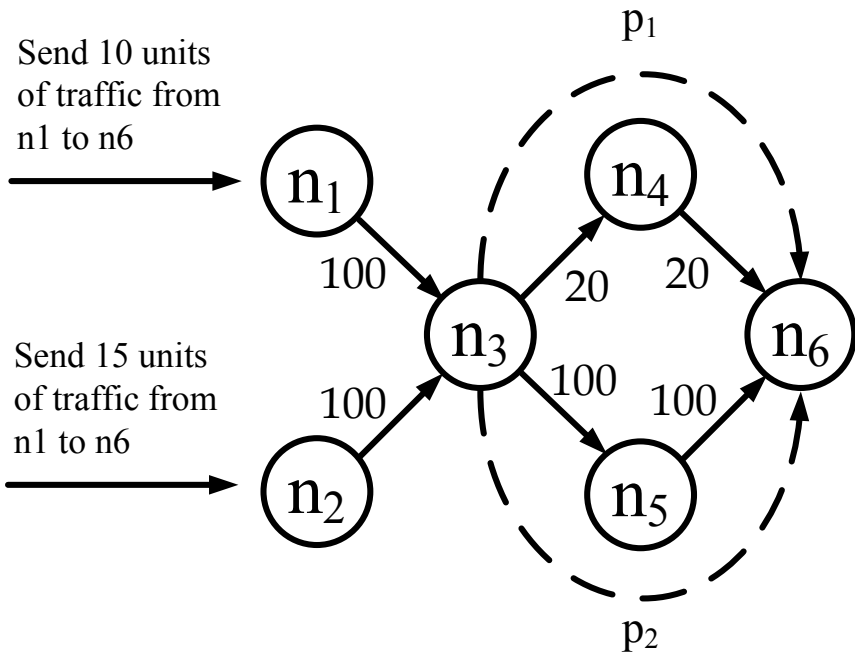


Fig. 1. ECMP constraint

our example network where the arc labels represent link capacities. We have to route 10 units of traffic from n_1 to n_6, and a further 15 units of traffic from n_2 to n_6. All traffic must aggregate at n_3. The figure shows that there only exist two paths from n_3 to n_6, denoted by p_1 and p_2. Let their costs be c_1 and c_2 respectively. Any link weight assignment will result in one of the following three: $c_1 = c_2$, $c_1 < c_2$, or $c_1 > c_2$. These translate into $\Lambda_{n_3n_6} = \{n_4, n_5\}$, $\Lambda_{n_3n_6} = \{n_4\}$, or $\Lambda_{n_3n_6} = \{n_5\}$ respectively. The corresponding values for the maximum link utilization are 1/2, 1, and 1/4 respectively. Therefore, we can not achieve maximum link utilization less than 1/4 with legacy IP routing. If it were possible to arbitrarily split traffic across next-hops, we could achieve a maximum link utilization of 1/5 in our example. This would be the case if we route one fifth of the traffic at n_3 along p_1 and the rest along p_2.

The idea presented in [6] leverages the observation that λ_x^ρ must only be a subset of Λ_{xe^ρ}. For instance, consider three destination prefixes ρ_1, ρ_2, and ρ_3 such that $e^{\rho_1} = e^{\rho_2} = e^{\rho_3} = t$. Further suppose that $\Lambda_{xt} = a, b, c, d$. Let the traffic corresponding to ρ_1, ρ_2, and ρ_3 that arrives at x be 4, 6, and 10 respectively. We wish to forward 2 units of traffic to next-hop b, 4 to d, and 7

each to a and c. Default IP routing only allows us to route 5 units of traffic across each of the four next-hops. However, we can configure $\lambda_x^{\rho_1} = \{b, d\}$, $\lambda_x^{\rho_2} = \{a, c, d\}$, and $\lambda_x^{\rho_3} = \{a, c\}$ to achieve the desired distribution. Hence, judiciously configuring the set of next-hops for each prefix gives a greater degree of freedom in engineering traffic.

Our solution, Interface-Specific Routing (ISR) follows in the same spirit as [6]. However, we judiciously configure the set of next-hops at a per incoming interface rather than per-prefix granularity. We again refer to our earlier example (Figure 1) to illustrate the motivation to do so. Let λ_{xd}^i denote the set of next-hops for traffic destined to d that arrives at x through the incoming interface [1] (i, x). If it were possible to configure per-interface next-hops, we can set $\lambda_{n_3 n_6}^{n_1} = \{n_4, n_5\}$, and $\lambda_{n_3 n_6}^{n_2} = \{n_5\}$. This causes 5 units of traffic to be routed across p_1 and 20 units to be routed across p_2, for a maximum link utilization of only 1/5.

The example illustrates how ISR affords greater flexibility in engineering traffic than default IP routing. The degree of freedom of configuring next-hops per interface is less than the per-prefix case. However, the solution in [6] assumes knowledge of per-prefix traffic intensity information for every prefix whose next-hop is to be configured. In addition [6] requires additional control mechanisms to explicitly configure the next-hops. This contrasts sharply with the simple mechanism of legacy IP routing wherein statically assigned link weights suffice for traffic engineering. The distinguishing feature of ISR is that it does not add complex control mechanisms or change the forwarding plane.

Modern routers implement separate forwarding tables per incoming interface for lookup efficiency [15]. However, these forwarding tables are *interface-independent*. In other words, the forwarding table at each interface yields the same set of next-hops for a particular destination. The presence of a separate forwarding table per interface means that routing lookups are performed independently at each interface. Hence, our previous definition of the ECMP constraint is more precisely qualified to stipulate that all traffic destined to node d, arriving at node x through interface (i, x) must be uniformly split across the set of next-hops from x to d yielded by the forwarding table at interface (x, i) . This set is given by λ_{xd}^i. Since $\lambda_{xd}^i = \Lambda_{xd}$ in legacy IP Routing, the qualified definition is equivalent to our earlier definition.

The challenge is to configure the *interface-specific* forwarding tables without incurring significant control overhead. We show how to use multiple weights per link to have optimized forwarding tables that are *interface-specific*, as opposed to being *interface-independent*. Specifically, we assign deg_{in}^i weights to each link (i, j), where deg_{in}^i is the number of incoming links at node i. Link

[1] Without loss of generality we assume an interface corresponds to a single incoming link. ISR can easily be extended to multiple links per interface.

weights can be assigned so that they yield different sets of next-hops to the same destination at different incoming interfaces. Therefore, at the aggregated node level, traffic to a particular destination may be non-uniformly distributed across the next-hops. This allows additional flexibility in routing traffic as compared to default IP routing whilst retaining its simplicity.

3.2 ISR Framework

We stated in section 3.1 that ISR does not require any changes to the forwarding plane. The ISR control plane has two fundamental components: (a) *statically assigning link weights*, and (b) *configuring interface-specific forwarding tables*. As alluded to previously, (a) involves assigning multiple weights to each link, which are disseminated through a link-state routing protocol. (b) involves using the multiple link weights to compute the interface-specific forwarding tables. The following presents a synopsis of the overall ISR architecture (Figure 2). A more detailed exposition follows in the next section.

(1) Given the *original network* $G(V, E)$, construct the *extended network* denoted by $\hat{G}(\hat{V}, \hat{E})$.
(2) Given the *original traffic matrix* $\{\Omega_{uv}\}_{(u,v)\in V\times V}$, construct the *extended traffic matrix* denoted by $\{\hat{\Omega}_{uv}\}_{(u,v)\in \hat{V}\times \hat{V}}$.
(3) Assign a *single* weight to links in $\hat{E}$ towards the end of optimizing the placement of traffic given by the extended traffic matrix $\{\hat{\Omega}_{uv}\}_{(u,v)\in \hat{V}\times \hat{V}}$ on the extended network $\hat{G}(\hat{V}, \hat{E})$.
(4) Use the weights assigned in the previous step to compute deg^i_{in} weights for each link $(i, j) \in E$, where deg^i_{in} is the in-degree of node i.
(5) Disseminate the *multiple* link weights across the network using a link-state routing protocol. As a result distributed nodes can populate their *extended link state database*.
(6) At each node, locally reconstruct the extended network $\hat{G}(\hat{V}, \hat{E})$ and weights assigned to links in $\hat{E}$ from the extended link state database.
(7) Run Dijkstra locally to compute the set of next-hop nodes for node-destination pairs with respect to $\hat{G}(\hat{V}, \hat{E})$.
(8) Translate the set of next-hops into interface-specific forwarding tables at each interface and subsequently populate the forwarding table at each incoming interface.

1-5 outline the computation and assignment of multiple link weights. The weights are computed to represent the optimal choice of interface-specific next-hops for each node in the network. This is explained in Section 4. 6-8 delineate how interface-specific forwarding tables are computed from the multiple link weights. The details of this computation follow in Section 5.

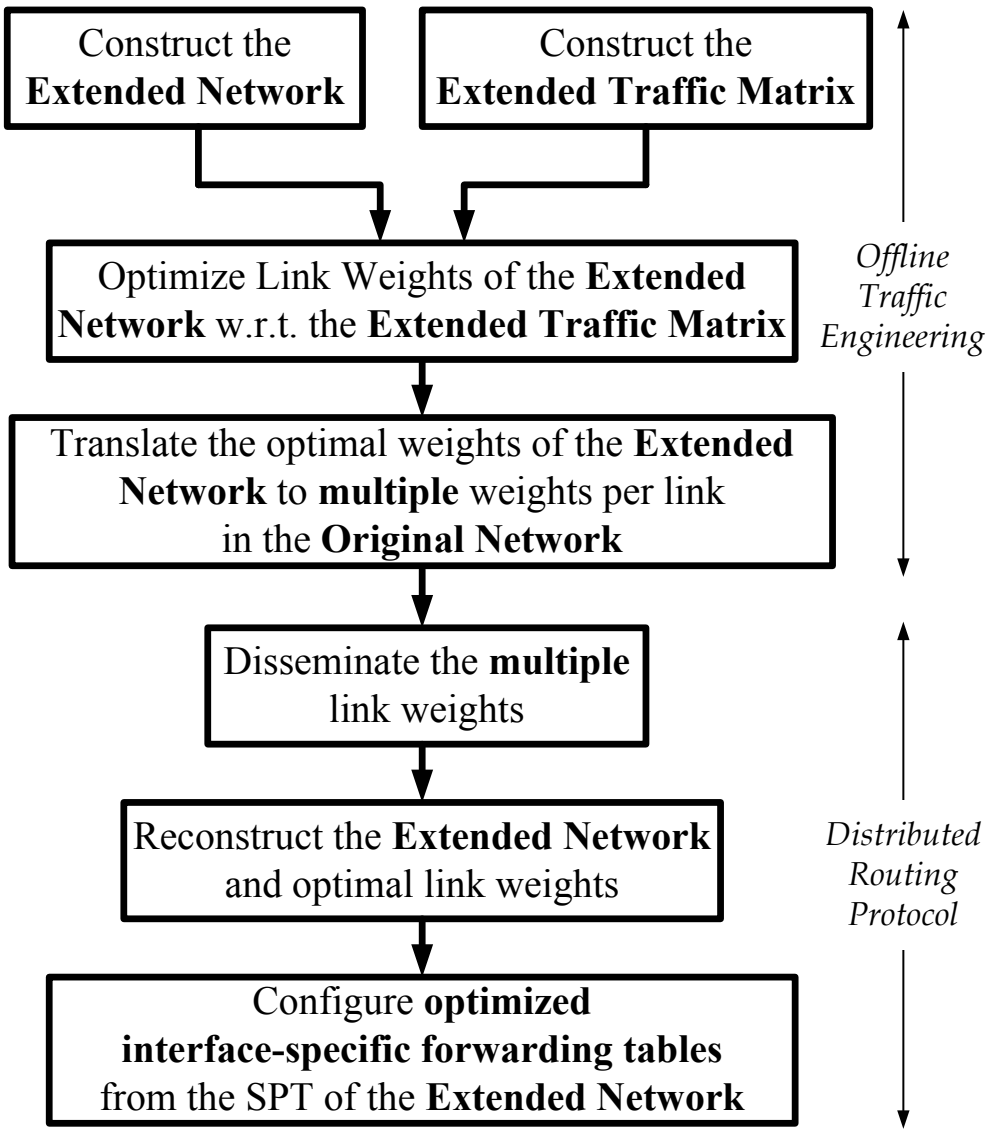


Fig. 2. ISR Framework

4 Multiple Link Weight Assignment

In default IP routing, we use knowledge of the traffic matrix $\{\Omega_{uv}\}_{(u,v)\in V\times V}$ to optimize weights for links in the network $G(V, E)$. These weights are then assigned to links and propagated across the network via link-state routing protocols. Nodes can then run a shortest-path algorithm. Interface-independent forwarding tables are simply configured by setting next-hop nodes for a prefix to be nodes that are immediately downstream the shortest path from the node to the egress node for the prefix. In ISR we must optimize weights in a way that leverages the ability to have interface-specific forwarding tables. Secondly, we must distribute enough information that allows each node to configure the desired interface-specific forwarding tables.

We will show how this can be accomplished by statically assigning and thereafter disseminating deg^i_{in} weights for each link $(i,j) \in E$, where deg^i_{in} is the in-degree of node i. The following discussion shows how this is done.

4.1 Construct the Extended Network

We first construct an extended network $\hat{G}(\hat{V}, \hat{E})$ from our original network $G(V, E)$. We construct a *source* node s^i for every node $i \in V$. Let $S = \bigcup_{i \in V} s^i$ be the set of all source nodes. We construct a *destination* node t^i for every node $i \in V$. Let $T = \bigcup_{i \in V} t^i$ be the set of all destination nodes. For every link $(i,j) \in E$, we construct two nodes in^{ij} and out^{ij} which we call the *link ingress* node and *link egress* node respectively. Let $I = \bigcup_{(i,j) \in E} in^{ij}$ and $O = \bigcup_{(i,j) \in E} out^{ij}$. The set of nodes in our extended network are given by $\hat{V} = S \cup T \cup I \cup O$.

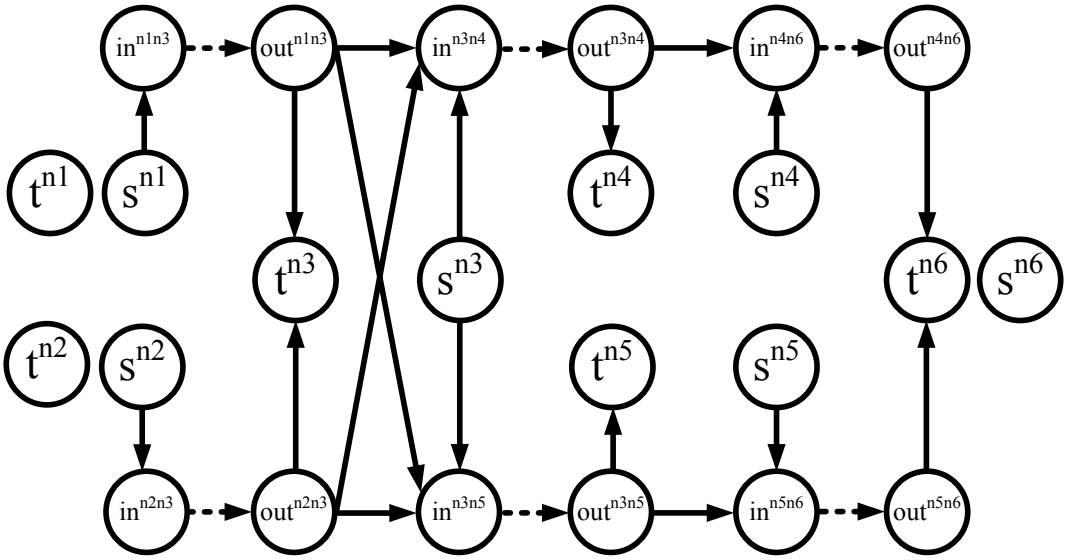


Fig. 3. Extended Network

We now discuss the set of links $\hat{E}$ in our extended network. We define an *originating* link (s^i, in^{ij}) for all $(i,j) \in E$. Similarly, we define a *terminating* link (out^{ij}, t^j) for all $(i,j) \in E$. Let $S' = \bigcup_{(i,j) \in E} (s^i, in^{ij})$ and $T' = \bigcup_{(i,j) \in E} (out^{ij}, t^j)$. We define a *capacity bottleneck* link (in^{ij}, out^{ij}) for all $(i,j) \in E$ and let $C' = \bigcup_{(i,j) \in E} (in^{ij}, out^{ij})$. Finally, we define *transit* links (out^{ij}, in^{jk}) for every $\{(i,j),(j,k)\} | (i,j),(j,k) \in E$ and $i \neq k$. Let X' be the set of all transit links. The set of links in our extended network are given by $\hat{E} = S' \cup T' \cup C' \cup X'$.

Figure 3 gives the extended network corresponding to our example network from Figure 1. The capacity of all links other than the *capacity bottleneck* links (corresponding to physical resources) is set to infinity. These links are shown

by the solid arcs. The capacity of *capacity bottleneck* link (in^{ij}, out^{ij}) is set to c_{ij}, where c_{ij} is the capacity of link (i, j) in the original network $G(V, E)$. Figure 3 shows the *capacity bottleneck* links by dashed arcs. We fix the weights of all links $(i, j) \in \hat{E} - (S' \cup X')$ to be 0. The links belonging to $S' \cup X'$ (shown in bold) are the ones for which we will optimize weights.

From our construction we see that $|\hat{V}| = |S| + |T| + |I| + |O| = 2|V| + 2|E|$. Hence our example extended network has $2 \times 6 + 2 \times 6 = 24$ nodes. Likewise, $|\hat{E}| = |S'| + |T'| + |C'| + |X'| = 3|E| + \sum_{i \in V} deg^i_{in} deg^i_{out}$. We can see that our example extended network has $3 \times 6 + 6 = 24$ links.

4.2 Construct the Extended Traffic Matrix

We construct the *extended traffic matrix* $\{\hat{\Omega}_{uv}\}_{(u,v) \in \hat{V} \times \hat{V}}$ from the original traffic matrix $\{\Omega_{uv}\}_{(u,v) \in V \times V}$. This is a simple step given by:

$$\hat{\Omega}_{uv} = \begin{cases} \Omega_{ij} & \text{if } (u, v) = (s^i, t^j) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

For instance, in our example given by Figure 1, $\Omega_{n_1 n_6} = 10$, $\Omega_{n_2 n_6} = 15$, and all other entries of $\{\Omega_{uv}\}$ are zero. Hence, $\hat{\Omega}_{s^{n_1} t^{n_6}} = 10$, $\hat{\Omega}_{s^{n_2} t^{n_6}} = 15$, and all other entries of $\{\hat{\Omega}_{uv}\}$ are zero.

4.3 Assign Optimal Link Weights

As explained above, we fix as 0 the weights of certain links in our extended network. Specifically, these are the *terminating* and *capacity bottleneck* links given by $T' \cup C'$. The weights for the *originating* and *transit* links, given by $S' \cup X'$ are tuned in order to optimize the placement of traffic represented by the extended traffic matrix $\{\hat{\Omega}_{uv}\}$. These weights for the extended network are assigned under the routing model characterized by the SP and ECMP constraints.

This optimized link weight assignment problem is exactly similar to the one used to compute link weights for default IP routing given the traffic matrix. The only difference is that we fix the weights of some links. The authors of [14] prove that finding the optimal weight setting is NP hard. Therefore, we can employ any of the heuristic algorithms or search techniques proposed to assign weights for default IP routing [9–14]. Most of these search-heuristics or algorithms randomly initialize the weights to be found. We specify an optimization for ISR. We first compute the optimal link weights for our original

network $G(V, E)$ and original traffic matrix $\{\Omega_{uv}\}_{(u,v)\in V\times V}$. Let w_{ij} be the weight assigned to link $(i, j) \in E$. We initialize (out^{ij}, in^{jk}) to have the weight w_{jk} for all $(out^{ij}, in^{jk}) \in X'$. We also initialize (s^i, in^{ij}) to have the weight w_{ij} for all $(s^i, in^{ij}) \in S'$. We denote the optimal weights found for the extended network by $\hat{w}_{ij}$ for $(i, j) \in \hat{E}$.

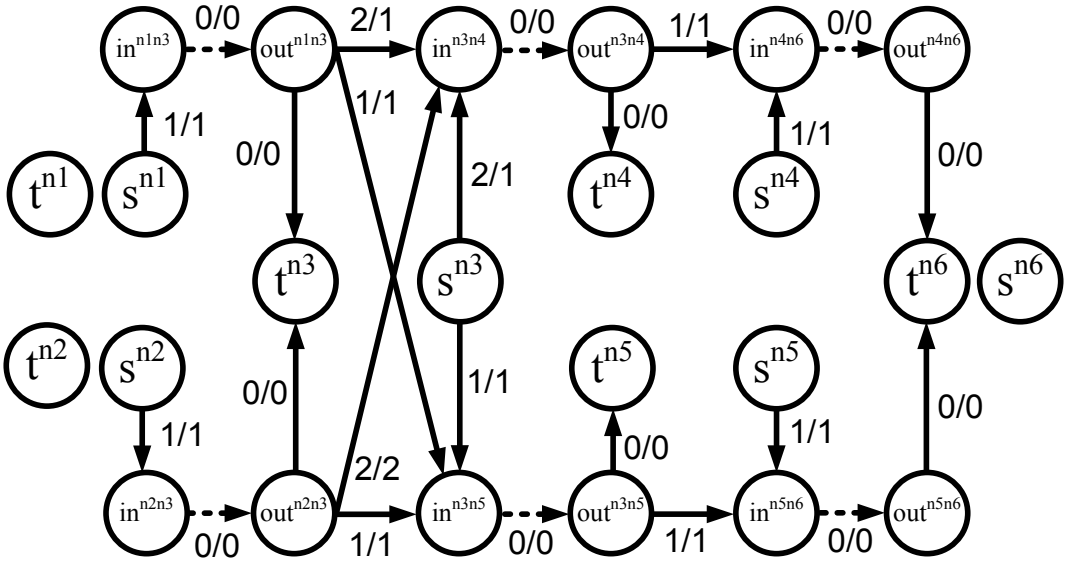


Fig. 4. Optimized Link Weight Assignment

Figure 4 delineates the optimal link weight assignment for our example network. We seek to optimize the *min-max-utilization* metric. The optimal link weight assignment for the given traffic for our original network in Figure 1 is setting $w_{(n_3,n_4)} = 2$ and setting all other link weights equal to 1. This yields the optimal max-utilization of $1/4$ achievable with default IP routing. Figure 4 shows two weights x/y for each link in the extended network. x is the initial weight set for a link derived from the solution for the original network and original traffic matrix. y is the optimized link weight with respect to the extended traffic matrix. Observe that the the optimized link weights with respect to the extended network and extended traffic matrix yield a max-utilization of $1/5$ in our example.

4.4 Assign Multiple Weights to Links

The optimized link weights with respect to the extended network and extended traffic matrix constitute the information we require to configure interface-specific next-hop at network nodes. However, we want this information to be made available to distributed network nodes by using existing link-state routing protocols. This is easily accomplished by modifying *Link State Advertisements* (LSAs) so that instead of having one weight for link $(i, j) \in E$, we

now have deg^i_{in} weights for $(i,j) \in E$. Therefore, every link has the following set of weights: $W_{ij} = \{\hat{w}^h_{ij} | (h,i) \in E \text{ or } h = j\}$. The value of $\hat{w}^h_{ij}$ is set as:

$$\hat{w}^h_{ij} = \begin{cases} \hat{w}_{out^{hi} in^{ij}} & \text{if } (h,i) \in E \text{ and } h \neq j \\ \hat{w}_{s^i in^{ij}} & \text{if } h = j \end{cases} \tag{4}$$

$\hat{w}^h_{ij}$ and $\hat{w}^h_{ik}$ for $h \neq j \neq k$ can be interpreted as signifying the relative preference of link (i,j) and (i,k) for traffic arriving at node i through node h. Also, $\hat{w}^h_{ij}$ and $\hat{w}^g_{ij}$ for $h \neq g \neq j$ can be interpreted as signifying the relative preferability of link (i,j) by traffic arriving at node i through nodes h and g, respectively. Notice that we would never want to forward traffic arriving at node i through node j back to node j. We abuse the notation $\hat{w}^j_{ij}$ to signify the preferability of link (i,j) for traffic originating at node i.

5 Interface-Specific Forwarding Tables

Thus far we have shown how, given the original network $G(V,E)$ and the original traffic matrix $\{\Omega_{uv}\}_{(u,v) \in V \times V}$, we can compute a set of weights $W_{ij} = \{\hat{w}^h_{ij} | (h,i) \in E \text{ or } h = j\}$ for a link $(i,j) \in E$. We further claimed that $\{W_{ij}\}_{(i,j) \in E}$ holds all the information required to optimally configure the interface-specific forwarding tables at each node. This section explores how to do that.

A link state routing protocol ensures that the link-state databases of a network node gets populated with $\{W_{ij}\}_{(i,j) \in E}$. The routing protocol also ensures that each node learns about the original topology $G(V,E)$. Using the deterministic construction defined in Section 4.1, each node can reconstruct the extended network $\hat{G}(\hat{V}, \hat{E})$.

Furthermore, we can also reconstruct the link weights that define our optimal solution for the extended network and traffic matrix as follows:

$$\hat{w}_{uv} = \begin{cases} 0 & \text{if } (u,v) \in \{T' \cup C'\} \\ \hat{w}^h_{ij} & \text{if } (u,v) = (out^{hi}, in^{ij}) \\ \hat{w}^j_{ij} & \text{if } (u,v) = (s^i, in^{ij}) \end{cases} \tag{5}$$

We now wish to compute the optimal interface-specific forwarding tables. We revert to our earlier definition from Section 3.1 wherein λ^i_{jd} denotes the set of next-hops for traffic destined to d that arrives at j through the incoming interface (i,j). Note that i, j, and d are nodes in the original network and (i,j) is a link in the original network. Let $\hat{\Lambda}_{jd}$ be the set of next-hops from $j \in \hat{V}$ to $d \in \hat{V}$ with respect to $\{\hat{w}_{uv}\}_{(u,v) in \hat{E}}$, in the extended network. The

interface specific forwarding tables are configured as follows:

$$\lambda^i_{jd} = \begin{cases} \{k | in^{jk} \in \hat{\Lambda}_{s^i t^d}\} & i = j \\ \{k | in^{jk} \in \hat{\Lambda}_{out^{ij} t^d}\} & \text{otherwise} \end{cases} \quad (6)$$

We use our example network to elucidate how interface-specific forwarding tables are configured. Consider node n_3 in the original network. It has two incoming interfaces (n_1, n_3) and (n_2, n_3). In the extended network with optimal weights shown in Figure 4, $\hat{\Lambda}_{out^{n_1 n_3} t^{n_6}} = \{in^{n_3 n_4}, in^{n_3 n_5}\}$, and $\hat{\Lambda}_{out^{n_2 n_3} t^{n_6}} = \{in^{n_3 n_5}\}$. Therefore, $\lambda^{n_1}_{n_3 n_6} = \{n_4, n_5\}$, and $\lambda^{n_2}_{n_3 n_6} = \{n_5\}$. In other words, the set of next-hops at n_3 to n_6 for traffic arriving through n_1 is $\{n_4, n_5\}$, and for traffic arriving through n_2 is $\{n_5\}$. Looking at Figure 1, this results in traffic arriving at n_3 from n_1 to be equally split across p_1 and p_2. Similarly, the all the traffic arriving at n_3 from n_2 gets routed along p_2. We, therefore, achieve a maximum link utilization of 1/5, which is superior to that which could have been achieved through default IP routing. Since we initialize the link weights to correspond to the optimal weight setting for default IP routing, ISR is guaranteed to achieve performance that is equal to or better than default IP routing

We defined $\hat{\Lambda}_{jd}$ as the set of next-hops from $j \in \hat{V}$ to $d \in \hat{V}$ with respect to $\{\hat{w}_{uv}\}_{(u,v) in \hat{E}}$, in the extended network. $\hat{\Lambda}_{jd}$ is determined by computing *Shortest-Path Trees* (SPT) with respect to the extended network. Since the extended network is larger than the original network, computing SPTs consumes greater CPU time. This can potentially mean that nodes could take a longer time to converge to a consistent forwarding state upon failure. However, solutions exist that ensure prompt computation of SPTs from the structure of previous SPTs [18, 19]. Such dynamic SPT computation lowers both the asymptotic complexities and the practical running times of the shortest path computation. For instance, SPT computation only takes O(100ms) for a network of more than 600 nodes [20]. Furthermore, since updating the forwarding engines constitutes the major bottleneck in convergence time [21], the extra computational overhead of SPT is an acceptable cost.

Proof of Correctness

Correctness of our ISR implementation will demonstrate that optimizing the *min-max-utilization* and *Fortz & Thorup* metrics for $\hat{G}(\hat{V}, \hat{E})$ with respect to $\{\hat{\Omega}_{uv}\}$, results in optimizing the metrics for $G(V, E)$ with respect to $\{\Omega_{uv}\}$. Let χ be the set of $i \rightarrow j$ paths in $G(V, E)$ and Υ be the set of $s^i \rightarrow t^j$ paths in $\hat{G}(\hat{V}, \hat{E})$, where $i, j \in V$. We define a function $\Re : \Upsilon \rightarrow \chi$, that maps each path in Υ to a corresponding path in χ. This function is represented by the

following:

$$\Re((s^1, in^{12}, out^{12}, in^{23}, out^{23}, ..., in^{(k-1)k}, out^{(k-1)k}, t^k)) = (1, 2, 3, .., k-1, k) \tag{7}$$

As an example, consider the path $\rho_{14} = (s^{n_1}, in^{n_1 n_3}, out^{n_1 n_3}, in^{n_3 n_4}, out^{n_3 n_4}, t^{n_4})$ in the extended network in Figure 3. Then, $\Re(\rho_{14})$ is given by the path $(1, 3, 4)$. We further claim that $\Re$ defines a 1:1 correspondence. This can be proved by observing that there are only four types of links in a $s^i \rightarrow t^j$ path in $\hat{G}(\hat{V}, \hat{E})$, where $i, j \in V$. These are *originating* links of the form (s^i, in^{ij}), *terminating* links of the form (out^{ij}, t^j), *capacity bottleneck* links of the form (in^{ij}, out^{ij}), and *transit* links of the form (out^{ij}, in^{jk}). From our construction of the extended network in 4.1, the following is true:

$$(i, j) \in E \Leftrightarrow (s^i, in^{ij}) \in \hat{E} \tag{8}$$

$$(i, j) \in E \Leftrightarrow (out^{ij}, t^j) \in \hat{E} \tag{9}$$

$$(i, j) \in E \Leftrightarrow (in^{ij}, out^{ij}) \in \hat{E} \tag{10}$$

$$(i, j), (j, k) \in E \Leftrightarrow (out^{ij}, in^{jk}) \in \hat{E} \tag{11}$$

Therefore, there exists a unique path $\Re(\hat{\rho}) \in \chi$, $\forall \hat{\rho} \in \Upsilon$. Similarly, there exists a unique path $\Re^{-1}(\rho) \in \Upsilon$, $\forall \rho \in \chi$.

Let $\hat{\rho} \in \Upsilon$ be a path from s^i to t^d, and $\Re(\hat{\rho})$ be given by $\rho = (n_1 = i, n_2, n_3, ..., n_t = d)$. Consider *only* the traffic between s^i to t^d on $\hat{G}(\hat{V}, \hat{E})$, and between i to d on $G(V, E)$. We claim that the amount of this traffic that arrives at node n_j through interface n_{j-1} is equal to the amount of traffic at $out^{n_{j-1} n_j}$, for $1 < j \leq t$. We prove our claim by induction. We assume that the amount of traffic that arrives at n_j through interface n_{j-1} is equal to the amount of traffic at $out^{n_{j-1} n_j}$. From 5 we know that $in^{n_j n_{j+1}} \in \hat{\Lambda}_{out^{n_{j-1} n_j} n_t}$ implies that $n_{j+1} \in \lambda^{n_{j-1}}_{n_j n_t}$. Also from 5, we know that $|\hat{\Lambda}_{out^{n_{j-1} n_j} n_t}| = |\lambda^{n_{j-1}}_{n_j n_t}|$. ECMP implies that amount of traffic forwarded along $(out^{n_{j-1} n_j}, in^{n_j n_{j+1}})$ is equal to the traffic forwarded along (n_j, n_{j+1}) that arrived at n_j through n_{j-1}. Since $(in^{n_j n_{j+1}}, out^{n_j n_{j+1}})$ is the only outgoing link from $in^{n_j} n_{j+1}$, the amount of traffic that arrives at n_{j+1} through interface n_j must be equal to the amount of traffic at $out^{n_j n_{j+1}}$. The basis is proved for $j = 2$, by noting that the traffic at $s^{n_1 = i}$ and $n_1 = i$ is the same. This follows from 3 wherein $\hat{\Omega}_{s^i, t^d} = \Omega_{id}$. Again from 5, we know $in^{n_1 n_2} \in \hat{\Lambda}_{s^{n_1} n_t}$ implies that $n_2 \in \lambda^{n_1}_{n_1 n_t}$. Also from 5, we know that $|\hat{\Lambda}_{s^{n_1} n_t}| = |\lambda^{n_1}_{n_1 n_t}|$. Hence, the same amount of traffic gets forwarded along $(s^{n_1}, in^{n_1 n_2}$ and (n_1, n_2). Since $in^{n_1 n_2}$ has only one outgoing link, it follows that the amount of traffic that arrives at n_2 through interface n_1 must be equal to the amount of traffic at $out^{n_1 n_2}$.

Suppose setting link weights for links in $\hat{G}(\hat{V}, \hat{E})$ with respect to the extended

traffic matrix, results in M units of traffic to traverse $\hat{\rho} \in \Upsilon$. The definition of $\Re$ and the preceding discussion imply that configuring the forwarding tables as per 5 and routing the original traffic matrix along $G(V, E)$ results in M units of traffic to traverse $\Re(\hat{\rho}) \in \chi$. Furthermore, from the definition of $\Re$, $\Re(\hat{\rho}) = \rho$ implies that if a *capacity bottleneck* link $(in^{ij}, out^{ij}) \in \hat{\rho}$ then $(i, j) \in \rho$, and vice versa. Observe that *capacity bottleneck* links in $\hat{V}$ are the only links with finite capacity. Suppose that we route the extended traffic matrix along $\hat{G}(\hat{V}, \hat{E})$ and the original traffic matrix along $G(V, E)$ by computing interface-specific forwarding tables according to 5. The link utilization on each link $(i, j) \in E$ is equal to the utilization of $(in^{ij}, out^{ij}) \in \hat{E}$. Therefore, heuristics optimizing the *min-max-utilization* and *Fortz & Thorup* metrics for $\hat{G}(\hat{V}, \hat{E})$ with respect to $\{\hat{\Omega}_{uv}\}$, also result in optimizing the metrics for $G(V, E)$ with respect to $\{\Omega_{uv}\}$.

We initialize link weights in the extended traffic matrix as described in Section 4.3. This results in $\lambda^i_{xd} = \hat{\Lambda}_{xd} = \Lambda_{xd}$. Hence, ISR always achieves equal or better traffic engineering than default IP routing.

6 Simulation Results

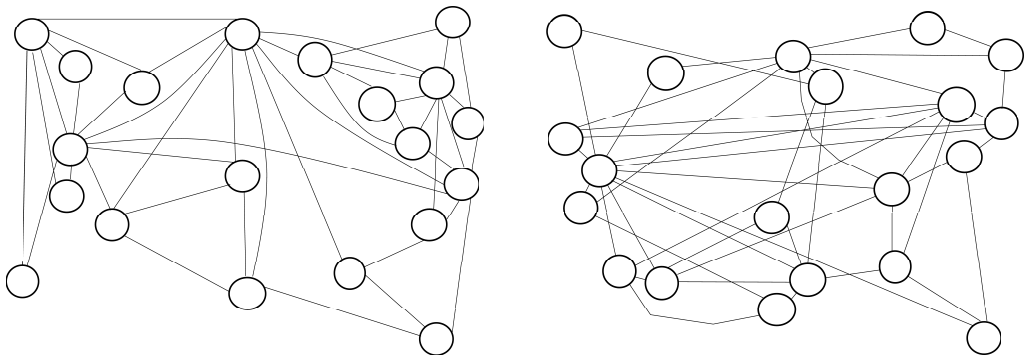
We now present results of our simulation study to quantify the performance of Interface Split Routing. We take into consideration both the performance metrics mentioned in Section 2. The *min-max-utilization* objective seeks to minimize the maximum utilization over all links. Optimizing for *min-max-utilization* avoids network bottlenecks that would otherwise act as points of congestion due to unexpected spikes in traffic. The other performance metric is the *Fortz & Thorup* metric [14], where the cost of a traffic engineering solution is the sum of the costs of all links. The cost of a link is determined by a piece-wise increasing linear convex envelope of a non-linear cost function of link utilization. The *Fortz & Thorup* metric takes into account the load on all links, rather than just the one with the maximum utilization.

We consider three different routing schemes:

- Optimal Routing (OR)
- Default Weight Optimized Routing (dWOR), e.g. achieved by current OSPF implementations
- Interface-Split Routing (ISR)

OR gives us a theoretical bound on the best-case performance of routing with link weights. The OR LP formulations for the *min-max-utilization* and the *Fortz & Thorup* metrics can be found at [7] and [14] respectively. We use CPLEX 10.0 to solve for OR. Both dWOR and ISR involve assigning link

weights to determine routing. In the case of dWOR, link weights are assigned to optimally route the original traffic matrix on the original network. In the case of ISR, link weights are assigned to optimally route the extended traffic matrix on the extended network. A few common heuristics are setting link weights to be the inverse of link capacity, identical link weights (minimum-hop routing), and setting link weights proportional to delay or Euclidean distance. We use the widely-cited local search meta-heuristic proposed in [14] to set weights for both our metrics. Their search strategy avoids getting stuck in a local loop and cycling by using hashing tables and random diversification. The latter has been shown to be significantly superior and represents the state-of art with respect to optimizing link weights. We, therefore, ignore these simpler heuristics in our simulation study. Since the OR solution represents the lower bound on the performance of traffic engineering, we wish to compare the percentage difference of the dWOR and ISR schemes with OR.



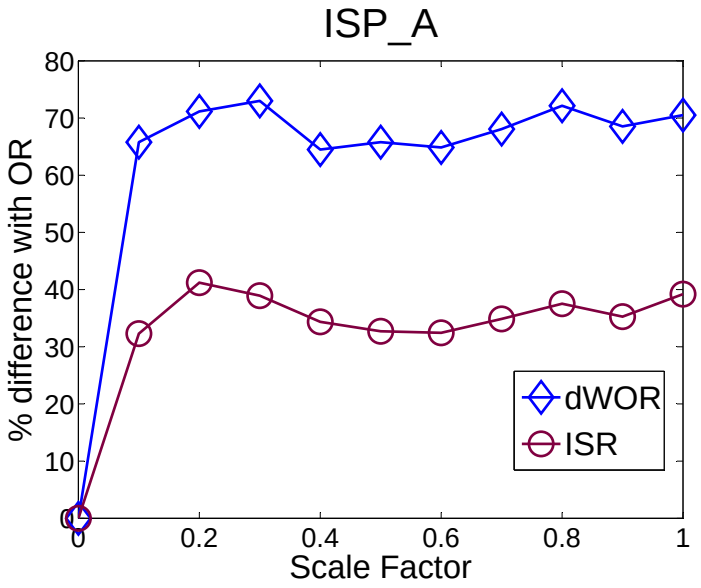
(a) ISP_A (b) ISP_B

Fig. 5. Simulation Networks

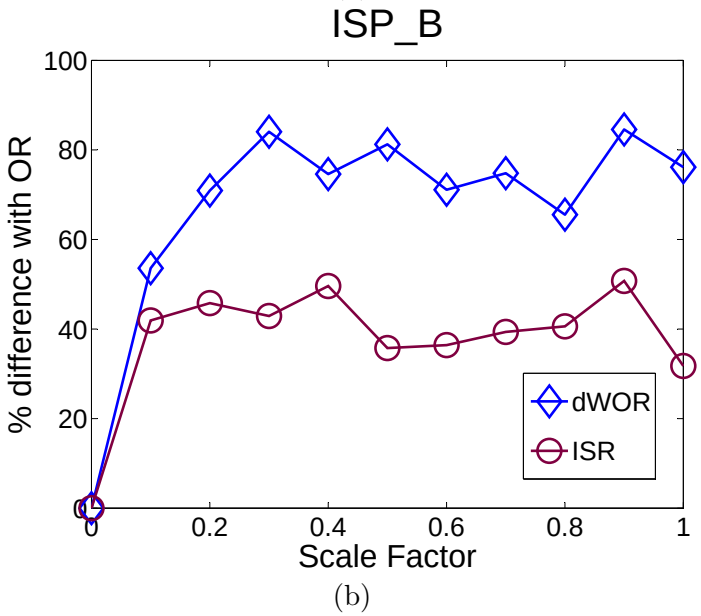
We conduct our first set of experiments on two tier 1 POP-level topologies ISP_A and ISP_B, shown in Figure 5. The link capacity for each link is set as 1200 units in each direction, modeling the capacity of OC-12 circuits. There are 190 ingress-egress pairs in both the networks. The total demand to be routed between an ingress-egress pair is drawn from a uniform distribution with $\mu = 45k$ and $\sigma = 0.3\mu$, where k is the *scale factor*. We vary k between 0 and 1 to change the amount of traffic on the network, with $k = 1$ representing the greatest volume of traffic. For every k, we generate 10 traffic matrices. For every traffic matrix, we run 10 independent iterations of the link weight assignment heuristic for both dWOR and ISR. This translates into a total of 100 iterations of the weight setting problem for each value of the scaling factor k. The results presented are averaged over the 100 run. The results for OR are averaged over the 10 traffic matrices per value of the scaling factor k.

Figures 6 and 7 give the results for the *min-max-utilization* and the *Fortz & Thorup* metrics respectively. We plot the percentage difference of dWOR and ISR with OR. In case of the *min-max-utilization* metric, we observe that

the gap between dWOR and OR is almost twice that between ISR and OR. The difference for the *Fortz & Thorup* metric is less pronounced when k is low. This is because the link costs approximate a convex function of link utilization. It is only when the overall network load is high that link costs vary sharply with increased link utilization and the difference between the schemes becomes significant. We see in Figure 7 that for $k > 0.5$ the gap between ISR and OR is significantly lower than the corresponding gap between dWOR and OR.



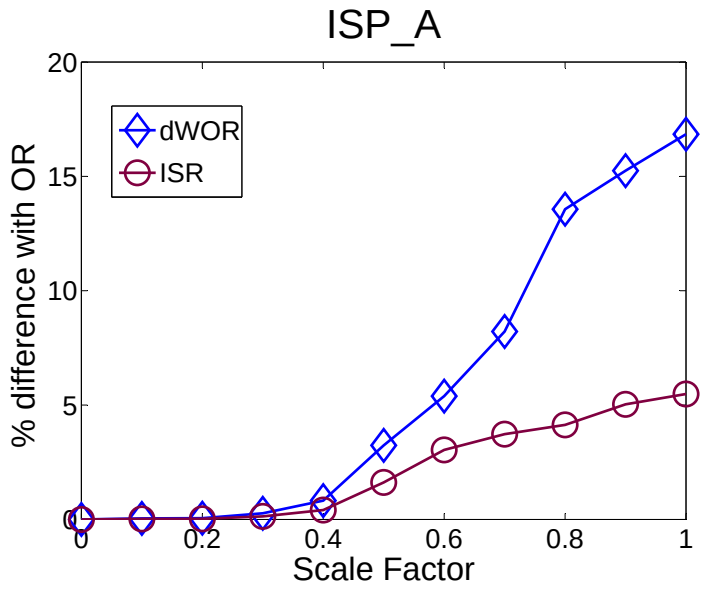
(a)



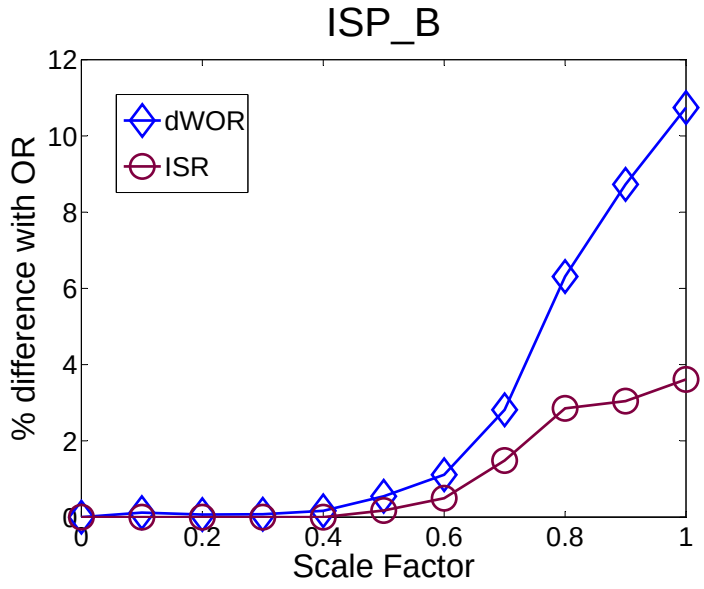
(b)

Fig. 6. Minimizing the Maximum Link Utilization

We conducted the same experiments as above for traffic matrices generated in different ways. We generated ingress-egress traffic intensity using a Pareto and a bimodal gaussian distribution. For the first case, the total demand to



(a)

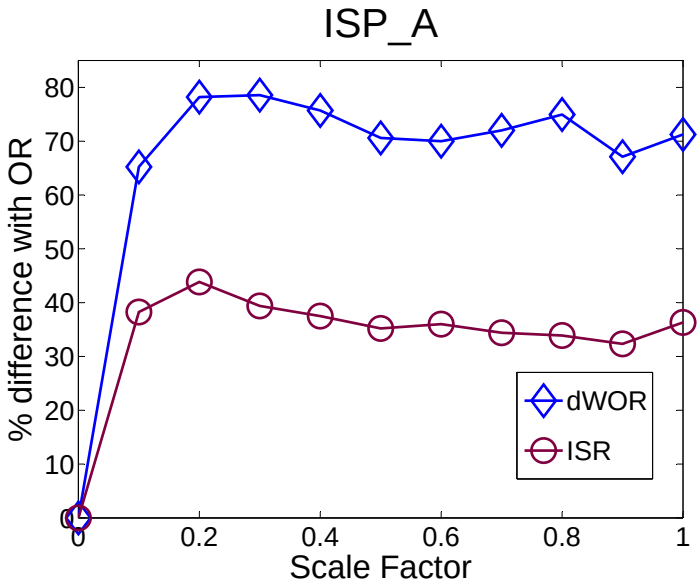


(b)

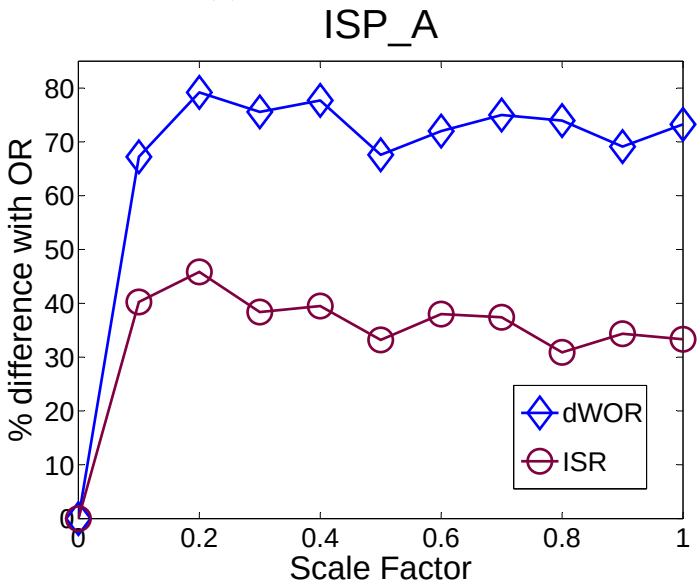
Fig. 7. Optimizing Fortz & Thorup

be routed between an ingress-egress pair is drawn from a Pareto distribution with $\mu = 45k$ and exponent $\alpha = 3$. As above, k is the *scale factor*. For the second case, the total demand to be routed between an ingress-egress pair is drawn from a gaussian $\mathcal{N}_1(\mu_1 = 30, \sigma_1 = 0.3\mu_1)$ with probability 0.8 and from $\mathcal{N}_2(\mu_2 = 105k, \sigma_2 = 0.3\mu_2)$ with probability 0.2. These distributions have been chosen to model actual traffic matrices [22]. Figure 8 presents the result for the *min-max-utilization* metric for traffic matrices generated using described distributions. Again, we see that ISR that the gap between ISR and OR is half of that between dWOR and OR. We performed similar experiments with different network topologies and traffic distributions and obtained similar

results.



(a) Pareto Distribution



(b) Bimodal Gaussian Distribution

Fig. 8. Minimizing the Maximum Link Utilization

In our last set of experiments we study the impact of average node degree on ISR. We randomly generate 20 node topologies such that the probability of an edge existing between two nodes is given by p. We only consider connected topologies. Higher values of p represent higher average node degree. Table 1 shows that the percentage difference between ISR and OR (δ) decreases as the average node degree (d) increases. The traffic matrix generation and experiment methodology is similar to our first set of experiments with $k = 0.5$. This result can be attributed to *interface split* routing since a greater number of incoming interfaces afford greater degree of freedom in engineering traffic.

p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
d	1.9	3.8	5.7	7.6	9.5	11.4	13.3	15.2	17.1	19
δ	30.2	29.5	28.9	26.9	21.2	14.9	5.7	2.7	2.8	2.5

Table 1
ISR Performance as a function of Average Node Degree

7 Conclusions

We presented Interface Split Routing (ISR) that allows greater flexibility in routing traffic by configuring independent forwarding tables at each interface. This lends itself well to router architectures that maintain separate forwarding tables per interface for performance efficiency. The interface-specific forwarding tables serve to relax the ECMP constraint allowing greater flexibility in traffic engineering. We showed how the information required to configure interface-specific forwarding tables can be disseminated using a link-state routing protocol. We further set the ISR weight computation problem in way that any algorithm for computing optimal link weights for legacy IP routing can be adapted to compute link weights for ISR. Our simulation results show that ISR significantly reduces the gap between legacy IP routing and optimal routing.

References

[1] Y. Wang, Z. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *Proceedings of IEEE Infocom*, 2001.

[2] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in *Proceedings of IEEE Infocom*, 2001.

[3] K. Kar, M. Kodialam, and T. Lakshman, "Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications," *IEEE Journal on Selected Areas in Communication*, vol. 18, no. 12, pp. 2566–2579, 2000.

[4] X. Xiao, A. Hannan, B. Bailey, and L. Ni, "Traffic engineering with MPLS in the Internet," *IEEE Network*, vol. 14, no. 2, pp. 28–33, 2000.

[5] D. Awduche, "MPLS and traffic engineering in IP networks," *IEEE Communications Magazine*, vol. 37, no. 12, pp. 42–47, 1999.

[6] A. Sridharan, R. Guerin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 234–247, 2005.

[7] H. Abrahamsson, B. Ahlgren, J. Alonso, A. Andersson, and P. Kreuger, "A Multi Path Routing Algorithm for IP Networks Based on Flow Optimisation," *Intl Workshop on Quality of Future Internet Services*, 2002.

[8] C. Villamizar, "OSPF optimized multipath OSPF-OMP," February 1999, internet-draft: draft-villamizar-ospf-omp-01.txt.

[9] S. Srivastava, G. Agrawal, and D. Medhi, "Dual-based link weight determination towards single shortest path solutions for OSPF networks," in *Proceedings of International Teletraffic Congress*, 2005.

[10] M. Ericsson, M. Resende, and P. Pardalos, "A Genetic Algorithm for the Weight Setting Problem in OSPF Routing," *Journal of Combinatorial Optimization*, vol. 6, no. 3, pp. 299–333, 2002.

[11] A. Nucci, B. Schroeder, S. Bhattacharyya, N. Taft, and C. Diot, "IGP Link Weight Assignment for Transient Link Failures," in *Proceedings of International Teletraffic Congress*, 2003.

[12] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional IP routing protocols," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 118–124, 2002.

[13] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 4, pp. 756–767, 2002.

[14] ——, "Internet Traffic Engineering by Optimizing OSPF Weights," in *Proceedings of IEEE Infocom*, Mar. 2000.

[15] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, "Proactive vs Reactive Approaches to Failure Resilient Routing ," in *Proceedings of IEEE Infocom*, 2004.

[16] D. Katz, K. Kompella, and D. Yeung, "Traffic Engineering (TE) Extensions to OSPF Version 2," RFC 3630, Tech. Rep., 2003.

[17] H. Smit and T. Li, "Intermediate System to Intermediate System (IS-IS) Extensions for Traffic Engineering," RFC 3784, Tech. Rep., 2004.

[18] P. Narvaez, K. Siu, and H. Tzeng, "New dynamic SPT algorithm based on a ball-and-string model," *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 706–718, 2001.

[19] ——, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 6, pp. 734–746, 2000.

[20] G. Iannaccone, C.-N. Chuah, S. Bhattacharyya, and C. Diot, "Feasibility of IP Restoration in a Tier-1 Backbone," *IEEE Network*, Mar. 2004.

[21] R. Keralapura, C. Chuah, G. Iannaccone, and S. Bhattacharyya, "Service Availability: A New Approach to Characterize IP Backbone Topologies," in *Proceedings of International Workshop on Quality of Service*, 2004.

[22] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic Matrix Estimation: Existing Techniques and Future Directions," in *Proceedings of ACM Sigcomm*, Aug. 2002.