Load-Balanced IP Fast Failure Recovery

Mingui Zhang[1], Bin Liu[1], and Beichuan Zhang[2]

[1] Computer Science and Technology Dept., Tsinghua University, Beijing, 100084, P.R. China
[2] Computer Science Dept., University of Arizona, Tucson, Arizona 85721, USA
zmg06@mails.tsinghua.edu.cn, bzhang@cs.Arizona.edu

Abstract. As a promising approach to improve network reliability, Proactive Failure Recovery (PFR) re-routes data traffic to backup paths without waiting for the completion of routing convergence after a local link failure. However, the diverted traffic may cause congestion on the backup paths if it is not carefully split over multiple paths according to their available capacity. Existing approach assigns new link weights based on links' load and re-calculates the routing paths, which incurs significant computation overhead and is susceptible to route oscillations. In this paper, we propose an efficient scheme for load balancing in PFR. We choose an adequate number of different types of loop-free backup paths for potential failures, and once a failure happens, the affected traffic is diverted to multiple paths in a well balanced manner. We formulate the traffic allocation problem as a tractable linear programming optimization problem, which can be solved iteratively and incrementally. As a result, only the flows affected by the failures are re-allocated to backup paths incrementally without disturbing flows not directly affected by the failures. Simulation results show that our scheme is computationally efficient, can effectively balance link utilization in the network, and can avoid route oscillations.

Keywords: OSPF; failure recovery; load balance; Linear Programming.

1 Introduction

One of the primary design goals of the Internet was to continue to function despite of the component failures [1]. At the IP level, when routers or links fail, the network should still be able to deliver the packets as long as alternative paths exist. In current routing protocols, such as OSPF (Open Shortest Path First) and IS-IS (Intermediate System to Intermediate System), routers are informed about network topology changes by update messages, and then re-calculate their routing paths accordingly. However, this approach incurs convergence delay, as it takes considerable long time for the update messages to propagate throughout the network and for the routers to re-calculate routing paths [2]. During the convergence period, packets may be delayed, dropped, or fall in temporary routing loops, which will definitely lead to significant performance degradation of on-serving applications.

A different approach for handling physical failures is Proactive Failure Recovery (PFR) ([3]), in which routers compute and store backup paths for potential failures beforehand, and once a *local* link failure is detected, a router will redirect traffic to backup paths right away instead of waiting for the completion of network-wide routing convergence. PFR has short failure recovery time and reduces the overhead of both update propagation and path re-calculation. Given that a large portion of failures in IP networks is short, transient failures [4], PFR should be able to improve the quality of packet delivery significantly.

In both the conventional routing convergence approach and the early work on PFR, routers redirect traffic affected by failures to alternative paths without load balancing considerations. The diverted traffic will easily cause uneven traffic reassignments, which either congests on already heavily loaded links, or under-utilizes lightly loaded links. Especially if other links are congested due to the diverted traffic, it defeats the goal of minimizing the impacts of failures to the application performance. Therefore, to be robust against failures, post-failure load balancing is particularly important.

Recently, "weight-setting" method [5] was used for load balancing in PFR [6]. Given that the demands have been projected from previous measurements, in order to find the weight set that can avoid congestions, the weights of links are re-assigned according to the load they carry: large weights for heavily loaded links and small weights for lightly loaded links, and then the routing paths are re-calculated based on the new link weights. In this way, part of the traffic will be shed from heavily loaded links to lightly loaded ones. The results in [5] show that changing weights for just a small percentage of links can significantly re-balance traffic load over the entire network.

However, re-calculating routing paths (e.g., running Dijkstra's algorithm) network-wide is expensive and time-consuming, and it may easily lead to routing oscillations as well. Weight-setting is often formalized as an optimization problem of Integer Linear Programming (ILP), which is NP-hard. Existing work has to resort to heuristics such as Tabu search [4] and local search [5]. The result does not guarantee convergence in that new weight assignment redistributes traffic, which will in turn affect weight assignment in the next round, and so forth. In some network settings, route oscillations can easily happen [7]. More recent work ([4], [6]) has put much efforts to try to refine the weight-setting method but no substantial improvements have been achieved.

Thinking in a different way, in this paper, instead of working on the "weight-seting", we turn to propose another light-weight yet efficient and stable scheme for post-failure load balancing with PFR. Each router prepares multiple backup paths for potential failures based on the best-path routing tables when the network is stable. Once a failure happens, the router will distribute the affected traffic over multiple backup paths through solving a Linear Programming (LP) problem incrementally, which is tractable and requires much less computation than ILP. The diverted traffic will be allocated on the multiple backup paths and the allocation will be further refined with the subsequent LP iterations. With this scheme, link weights keep unchanged and routers do not need to re-calculate their routing paths, but the load-balancing goal is perfectly achieved.

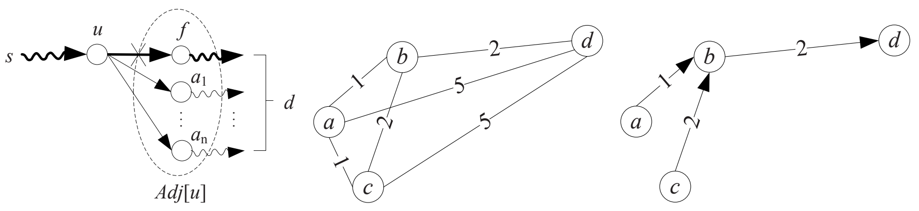
Two design challenges are addressed here in PFR and its post-failure load balancing: 1) how to select multiple loop-free backup paths with small overhead, and 2) how to decide the amount of affected traffic to be shed on each backup path in a well balanced manner. By proposing a unique solution, we make the following four contributions. 1) We explore and identify another two additional types of loop-free alternative paths locally besides the Equal-Cost-Multiple-Path [8] [9], which gives more path diversity for post-failure delivery; 2) In deciding traffic distribution over multiple paths, we formulate it as a LP problem minimizing the sum of link utilization. By solving the problem iteratively and assigning penalty factors to heavily loaded links, we actually achieve the goal of minimizing the maximum link utilization in the network; 3) We introduce a bias factor into the LP iteration to damp the oscillations; 4) Once a failure happens, the LP problem can be solved *incrementally* since only the traffic affected by the failures is taken into consideration by the LP objective function. Simulation results show that our scheme can effectively balance the load over multiple paths, has small computation overhead and converges fast.

The rest of the paper is organized as follows. Section 2 presents how we choose loop-free backup paths for failure recovery. Section 3 formulates the traffic allocation as a LP problem and describes how we solve it iteratively and incrementally. Section 4 evaluates the scheme's performance. Section 5 concludes the paper.

2 Alternative Paths Setup

In PFR, routers try to detect and recover failures locally rather than relying on network-wide routing convergence. Generally, a link failure will trigger the physical detection, which reports the event to the IP layer immediately. A router can locally find multiple paths forwarding the affected traffic on the failed link(s) via its neighbors. Figure 1(a) illustrates this by a simple example. Node u's next hop on its shortest path towards d is f. Node u also maintains alternative paths to reach d through other neighbors such as $a_1, a_2, \ldots, a_{\mathrm{n}}$. When a local link $u \rightarrow f$ fails, u will shift its affected traffic to alternative paths, e.g., via a_1, without waiting for the completion of routing convergence. The diverted packets are marked (e.g., using the Type Of Service (TOS) field in the packet), so that the downstream routers will know that the packet is diverted and can make appropriate forwarding decisions.

PFR works mainly in intra-domain routing, or IGP, such as OSPF and IS-IS. It can handle single-link failures and multiple-link failures that do not affect each other. In other words, as long as the multiple failures do not disable all the alternative paths and the packets diverted to the alternative paths do not encounter another failure again, PFR works fine. Our work is within the same intra-domain routing scope and inheres the same limitation regarding multiple link failures. Our contribution is to set up and utilize multiple alternative paths (e.g., paths via $a_1, a_2, \ldots, a_{\mathrm{n}}$ in Figure 1(a)) efficiently



(a) Local alternative paths (b) Non-directional graph (c) All the shortest paths to d

Fig. 1. Demo graphs

to spread the adjusted traffic load to avoid potential post-failure congestions. This section describes how we choose multiple paths, and the next section describes how we distribute traffic over the multiple paths.

On one hand, given a destination, there can be a huge number of alternative paths from a router to reach the destination. It is not only impractical but also unnecessary to find all the possible alternative paths and store them. On the other hand, the number of alternative paths cannot be too small, otherwise they will not be able to serve the purpose of failure recovery and load balancing. Therefore, we need to find *adequate* number of *loop-free* alternative paths without significant computation overhead.

One way of doing multi-path routing is to split traffic among paths with the same IGP costs [8]. Those ECMPs should be able to be used for post-failure load balancing without introducing routing loops. We extend the Dijkstra's algorithm to calculate ECMP from a router to all destinations. We use ECMP for load balancing, but they do not have the adequate path diversity yet. At this point, we need other types of loop-free alternative paths.

We represent the network by a positive weighted bi-directional graph $G = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of vertexes and $\mathcal{A}$ is the set of arcs. Running Dijkstra's algorithm on G will generate shortest paths that form a tree rooted at the source vertex s. This tree is known as Shortest Paths Tree (SPT) [10]. We call the shortest paths generated by Dijkstra's algorithm the type-0 paths. Running the extended Dijkstra's algorithm will generate ECMP, which are called the type-1 paths. Type-0 paths are the primary forwarding paths, and type-1 paths are alternative paths. Further, we will calculate another two types of paths and prove that they can be used as alternative paths too.

We reverse the direction of each arc in the graph G, and then run Dijkstra's algorithm to get SPT regarding vertex d as the "root". Finally, by reversing back the directions of the arcs, we get a subgraph that gives every vertex's shortest path to the given destination d. Let G_π be the subgraph we get. Figure 1(c) is an example of G_π for the graph given in Figure 1(b). In the following discussion, we focus on the paths to a single given destination, i.e., G_π. It is difficult and unnecessary for u to get all the information of G_π locally. However, we can get a part of the information from u and its neighbors locally, as depicted in Figure 1(a). This part of information is enough for u to set up the other two types of alternative paths, as described in the following. We define the shortest path from s to d on G_π by $s \rightsquigarrow d$. $s \rightsquigarrow u \rightsquigarrow d$ is the shortest path from s to d going through u, $s \rightsquigarrow u \rightarrow f \rightsquigarrow d$ is the shortest path from s to d via arc $u \rightarrow f$, and $s \rightsquigarrow d/u \nrightarrow f$ represents the shortest path from s to d not via arc $u \rightarrow f$. Suppose there is a shortest path $p \rightsquigarrow q$. $s \rightsquigarrow d/p \not\rightsquigarrow q$ denotes the shortest path from s to d does not use any vertex or arc on $p \rightsquigarrow q$. When p can not reach q due to failures, s can still reach d safely using $s \rightsquigarrow d/p \not\rightsquigarrow q$.

On G_π, there is a single successor for each vertex. Assume f is the successor of u. We disable the connection of arc $u \rightarrow f$ proactively to prepare the alternative paths for u, which is $u \rightsquigarrow d/u \nrightarrow f$. In our work, we resolve $u \rightsquigarrow d/u \nrightarrow f$ locally instead of fixing up $s \rightsquigarrow d/u \nrightarrow f$ directly. When the arc $u \rightarrow f$ really fails, we use $s \rightsquigarrow u \rightsquigarrow d/u \nrightarrow f$ to replace $s \rightsquigarrow d/u \nrightarrow f$. u must go through its neighbors except f to bypass the failure: $u \rightsquigarrow d/u \nrightarrow f = u \rightarrow a \rightsquigarrow d/u \nrightarrow f$, here $a \in Adj[u] - \{f\}$.

Let $\mathbb{A}(u) = Adj[u] - \{f\}$. Since a's shortest path to d may or may not be affected by the failure, $\mathbb{A}(u)$ can be divided into 2 disjoint sets: $\mathbb{A}_1(u) = \{a|a \in \mathbb{A}(u)$ and $a \rightsquigarrow d = a \rightsquigarrow d/u \nrightarrow f\}$ and $\mathbb{A}_2(u) = \{a|a \in \mathbb{A}(u)$ and $a \rightsquigarrow d \neq a \rightsquigarrow d/u \nrightarrow f$(i.e. $a \rightsquigarrow d = a \rightsquigarrow u \rightarrow f \rightsquigarrow d)\}$.

Theorem 1. *If $a \in \mathbb{A}_1(u)$ then $a \rightsquigarrow d = a \rightsquigarrow d/s \not\rightsquigarrow u$, i.e., $a \rightsquigarrow d$ does not use any vertex or arc on $s \rightsquigarrow u$.*

Proof. Assume $a \rightsquigarrow d$ uses any vertex v on $s \rightsquigarrow u$, v will go through $u \rightarrow f$ to d, then $a \rightsquigarrow d$ must be affected by the failed arc $u \rightarrow f$, namely $a \rightsquigarrow d \neq a \rightsquigarrow d/u \nrightarrow f$, i.e., $a \notin \mathbb{A}_1(u)$. We obtain a contradiction here. □

Theorem 1 tells us that when $u \rightarrow f$ is failed, the shortest path of the neighbors in set $\mathbb{A}_1(u)$ is safe to be used by u without any risk of incurring a loop. Thus, we safely replace $u \rightsquigarrow d/u \nrightarrow f$ with $u \rightarrow a \rightsquigarrow d$.

The failure of arc $u \rightarrow f$ on the shortest path from s to d can be recovered by u locally. These alternative paths are stored *proactively* in router u, and we call them the type-2 paths.

All neighbors in set $\mathbb{A}_2(u)$ are not safe to be used, but we still have a method to obtain the alternative paths via part of them. Assume $u' \in \mathbb{A}_2(u)$. u' has its own type-2 alternative paths to bypass their shortest path failure of arc $u' \rightarrow f'$. Let $a' \in \mathbb{A}_1(u')$. If the type-2 alternative path $u' \rightarrow a' \rightsquigarrow d$ luckily does not go through the failed arc $u \rightarrow f$, i.e. $u' \rightarrow a' \rightsquigarrow d = u' \rightarrow a' \rightsquigarrow d/u' \nrightarrow f' = u' \rightarrow a' \rightsquigarrow d/(u' \nrightarrow f', u \nrightarrow f)$, u may use this path to establish its own alternative path as well. The paths established in this way are also safe for u, which can be insured by the following theorem.

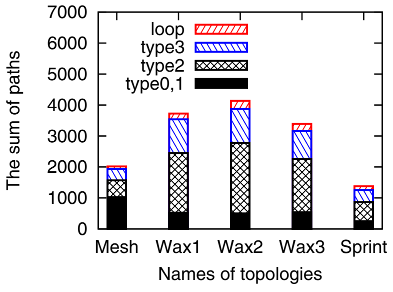
Theorem 2. *If $u' \in \mathbb{A}_2(u)$, and $a' \in \mathbb{A}_1(u')$, assume $a' \rightsquigarrow d = a' \rightsquigarrow d/u \nrightarrow f$, then $a' \rightsquigarrow d = a' \rightsquigarrow d/s \not\rightsquigarrow u$.*

Proof. There are two possibilities for u': a) $u' \in \mathcal{V}[s \rightsquigarrow u]$ and b) $u' \notin \mathcal{V}[s \rightsquigarrow u]$. Here, $\mathcal{V}[s \rightsquigarrow u]$ denotes the set of all the vertexes on the shortest path from s to u. We get the contradictions respectively as following to prove the theorem.

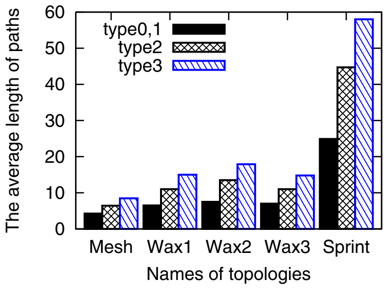
a) $u' \in \mathcal{V}[s \rightsquigarrow u]$.Because f' is the successor of u' on G_π, u' is on the path $s \rightsquigarrow u$ and $u' \neq u$, f' must be on the path $s \rightsquigarrow u$ too. Then, $u' \rightarrow f' \in \mathcal{A}[s \rightsquigarrow u]$. $\mathcal{A}[s \rightsquigarrow u]$ denotes the set of all the arcs on the shortest path $s \rightsquigarrow u$. Here, f' may happen to be u. For the purpose of contradiction, we suppose $a' \rightsquigarrow d \neq a' \rightsquigarrow d/s \not\rightsquigarrow u$. That is to say the type-2 alternative path of u', $u' \rightarrow a' \rightsquigarrow d$ must go through a vertex v on the shortest path $s \rightsquigarrow u$, i.e. $v \in \mathcal{V}[s \rightsquigarrow u]$. If u uses this path, a loop will occur. According to theorem 1, $v \notin \mathcal{V}[s \rightsquigarrow u']$, thus $v \in \mathcal{V}[f' \rightsquigarrow u]$. $v \rightsquigarrow d$ must go through arc $u \rightarrow f$ as well. Consequently, $a' \rightsquigarrow d \neq a' \rightsquigarrow d/u \nrightarrow f$, which contradicts our assumption.

b) $u' \notin \mathcal{V}[s \rightsquigarrow u]$.
Obviously, $u' \rightarrow f' \notin \mathcal{A}[s \rightsquigarrow u]$, the failure of $u' \rightarrow f'$ will not break the connectivity of $s \rightsquigarrow u$. If the type-2 alternative path of u' goes through the vertex on $s \rightsquigarrow u$, it must go through the arc $u \rightarrow f$ as well. The contradiction is easy to get. □

Therefore, according to Theorem 2, if u' meets the assumption in the theorem, we can safely replace $u \rightsquigarrow d/u \nrightarrow f$ with $u \rightarrow u' \rightarrow a' \rightsquigarrow d$. We call this kind of alternative



(a) The sums of paths



(b) The average lengths of paths

Fig. 2. The statistics of different kinds of paths

paths type-3 paths. u's remaining neighbors whose type-2 paths are affected by the failure of arc $u \rightarrow f$ can not be used or else loops will occur.

Always, the alternative paths can be configured explicitly using MPLS (Multi-protocol Label Switching) [11]. Nevertheless, MPLS does not appeal to ISPs due to its configuration overhead and vulnerability to human errors [9]. Mechanisms that can be employed in pure IP networks appear to be more potential in the near future [12]. When a failure occurs, the upstream router adjacent to the failure takes charge of the failure before the finish of the time-consuming routing convergence. It uses the alternative paths to bypass the failure, and mark whether the packets should be detoured or normally delivered by its downstream. The downstream router will determine locally which kind of paths should be used. The TOS field is a natural choice for us to do the marking, and there is similar work in IETF that deals with Multi-Topology routing [13]. However, our scheme need only one bit while Multi-Topology routing has to exploit several bits.

Figure 2 shows the statistics of different topologies including the Mesh-16-4 topology[1], three topologies with 20 nodes produced by the Waxman model [14] and the PoP-level North American Sprint IP backbone topology got from [4]. We use "Mesh", "Waxman" or "Wax" and "Sprint" for short respectively in this paper. Figure 2(a) shows that the alternative paths are plentiful and Figure 2(b) shows that the length of them are not very long despite of the bypass.

So far, the first design question has been solved efficiently and sufficiently. We move on to the next question: divert and balance the affected traffic onto the alternative paths.

3 Post-failure Load Balancing

After choosing the extra loop-free backup paths (type-2, type-3) besides ECMPs (type-1), we use them to balance the load in case of link failures. This section describes how to divert traffic onto backup paths so that the risk of causing congestions is minimized.

[1] There are 16 nodes and every node connects to 4 neighbors. The weight of each link is 1.

Table 1. $\mathbf{ODL}$ matrix

	$\mathbf{ODL}(:,:,1)$				$\mathbf{ODL}(:,:,4)$				$\mathbf{ODL}(:,:,5)$			
	a	b	c	d	a	b	c	d	a	b	c	d
a	0	0	0	4	0	0	0	0	0	0	0	4
b	0	0	0	0	0	0	3	0	0	0	0	0
c	0	0	0	0	0	0	0	10	0	0	0	10
d	0	0	0	0	0	0	0	0	0	0	0	0

3.1 ODL Matrix

The traffic matrix of an IP network can be represented by the Origination-Destination matrix, i.e., the $\mathbf{OD}$ matrix [7]. $\mathbf{OD}(j,k)(j,k=1,2,...,N; j \neq k)$ denotes the traffic volume originated from node j and destined to node k. Here N is the total number of nodes in the network.

Based on $\mathbf{OD}$ matrix an $N \times N \times L$ matrix, called $\mathbf{ODL}$ (Origin-Destination-Link) matrix, is devised in our work to record the details of the load distributed onto network links. Here, L is the total number of the links in the network. On the network in Figure 1(b), for example, router a wants to send 4 units of traffic to router d, router b wants to send 3 units of traffic to router c, and router c wants to send 10 units of traffic to router d. Given that only type-0 paths are used. The $\mathbf{ODL}$ matrix is given in Table 1.

All the elements of $\mathbf{ODL}(:,:,2)$, $\mathbf{ODL}(:,:,3)$ and $\mathbf{ODL}(:,:,6)$ are zeros which are not exhibited in Table 1.

The load of every link can be calculated conveniently from the data structure $\mathbf{ODL}$ through the manipulation of summarizing specific matrix's elements. We use vector $\mathbf{LLV}$ to denote the load on every link. For $\mathbf{ODL}$ given in Table 1, $\mathbf{LLV} = [4, 0, 0, 13, 14, 0]$.

3.2 The Formulation of Linear Programming

Previous work of the weight-setting approach formulates the traffic allocation problem as an ILP problem, which is NP-hard. We formulate it to a different LP problem instead, which is more tractable. With the multiple alternative paths established and the details encoded by the data structure $\mathbf{ODL}$, the LP problem can be formulated as follows.

Suppose $\mathcal{P}_{jk}$ is the set of the paths from j to k. x_{jkp} is the share of traffic originated from j and destined to k through the pth path of $\mathcal{P}_{jk}$. Let $\mathbf{X}$ be the vector $[-x_{jkp}-](j,k = 1,2,\dots,N; j \neq k; p = 1,2,\dots,|\mathcal{P}_{jk}|)$. Denote the links' capacities as $\mathbf{c} = [c_1, c_2, \dots, c_L]$ and link utilization as $\mathbf{u}=[u_1, u_2, \dots, u_L] = \mathbf{LLV}./\mathbf{c}$, where the operator "./" denotes the element-by-element division of two vectors, i.e., $u_l = \mathbf{LLV}(l)/c_l$.

We simply sum up every link's utilization to get the objective function $f(\mathbf{X})$. An optimization problem of Linear Programming can be depicted as follows. By solving this problem, we get the optimal share of the traffic demand $\mathbf{X}_{\mathrm{OPT}}$.

$$\min \ f(\mathbf{X}) = \sum_{l=1}^{L} u_l \tag{1}$$

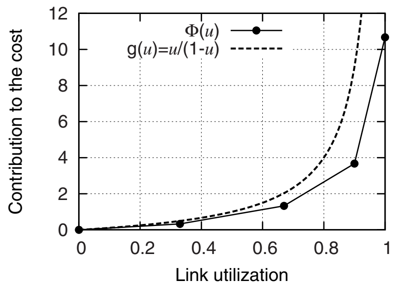


Fig. 3. The comparison between two cost functions. The curve of $\Phi(u)$ is similar to that of g(u).

$$\text{subject to} \quad \sum_{p=1}^{|\mathcal{P}_{jk}|} x_{jkp} = 1 \tag{2}$$

$$x_{jk(p-1)} \geq x_{jkp} \quad p = 2, \ldots, |\mathcal{P}_{jk}| \tag{3}$$

$$0 \leq x_{jkp} \leq 1 \tag{4}$$

Intuitively, longer paths should get less traffic. The paths in $\mathcal{P}_{jk}$ are indexed in an increasing order of the paths' lengths.

Weight-setting method has to resort to heuristic search to solve the ILP which is NP-hard [5]. We know that LP is always simpler than ILP. Therefore, our method should be simpler than weight-setting method.

When LP is finished, the backup paths already get their proportion of traffic according to their available capability. But we are not content with this and try to further refine the result in the next subsection.

3.3 LP Iteration

We use $f(\mathbf{X}) = \sum_{l=1}^{L} u_l$ in the last subsection as the objective function of LP, which treats the contribution of every link utilization to the overall cost equally. But $f(\mathbf{X})$ is not the true objective function of optimization. We introduce the *LP iteration* to do the refinement in the following.

First of all, the contribution of each link utilization u_l to the overall cost function can be estimated by

$$\mathbf{Ctb}(u_l) = \begin{cases} u_l/(1-u_l) & , 0 \leq u_l < 0.9 \\ 9 & , u_l \geq 0.9 \end{cases} \tag{5}$$

This is based on the following analysis. Assume the IP network is a Jackson queuing network. A frequently used formula from [7] is:

$$D_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij} - F_{ij}} + d_{ij} F_{ij} \tag{6}$$

where F_{ij} is the traffic load on link (i, j) expressed in bit/s, C_{ij} is the transmission capacity of link (i, j) measured in the same units as F_{ij} and d_{ij} is the processing and propagation delay.

Suppose the utilization of link (i, j) is $u_{ij} = F_{ij}/C_{ij}$, and $d_{ij} = 0$, then

$$\sum_{(i,j)} D_{ij}(F_{ij}) = \sum_{(i,j)} \frac{u_{ij}}{1 - u_{ij}} \tag{7}$$

Let u be the utilization of a specific link, then this link's contribution to the cost function is

$$g(u) = u/(1 - u), u \in [0, 1] \tag{8}$$

The curve of $g(u)$ is depicted in Figure 3.

Previous work in [5] and [6] uses a piece-wise cost function:

$$\Phi(l(a)) = \begin{cases} l(a) & , 0 \leq u(a) < 1/3 \\ 3l(a) - 2c(a)/3 & , 1/3 \leq u(a) < 2/3 \\ 10l(a) - 16c(a)/3 & , 2/3 \leq u(a) < 9/10 \\ 70l(a) - 178c(a)/3 & , 9/10 \leq u(a) < 1 \\ 500l(a) - 1468c(a)/3 & , 11/10 \leq u(a) < \infty \end{cases} \tag{9}$$

Here, $l(a)$ is the load carried by link a, $c(a)$ is the capacity of a and $u(a)$ is the utilization of a. Suppose $c(a) = 1$, then $l(a) = u(a)$. Depict the curve of $\Phi(u)$ in the same figure as $g(u)$. We find that the two curves are similar to each other. However, the objective function in Equation (9) is used arbitrarily without any analysis in [5] and [6]. It is still too complex to be used in the n-objective function, as it is a piece-wise linear function rather than a linear function. Based on our analysis of a simplified network model, we use cost function (5) to estimate the contribution of a link to the overall cost.

With $\mathbf{Ctb}$, we define a penalty vector $\mathbf{p}$ as follows:

$$\mathbf{p} = (\mathbf{Ctb}/\max(\mathbf{Ctb}) + bias)/(1 + bias) \tag{10}$$

At each iteration, $f(\mathbf{X})$ is substitute with $\mathbf{p}.{*}f(\mathbf{X})$ and the LP is solved over and again. The operator ".*" denotes the element-by-element multiplication of two vectors. $bias$ is a substantial bias factor used to damp potential route oscillations. $bias \in [0, \infty)$ and $\mathbf{p} \in [0, 1]$. In [7], $bias$ has been mentioned as a measure for damp oscillations. Here, we use the $bias$ in a simple way in order not to introduce much computation cost. In our experiments, oscillations can happen too if the $\mathbf{Ctb}$ is used directly without $bias$. By comparison, weight-setting method mentioned in [4], [6] and [5] even does not address the issue of potential route oscillations.

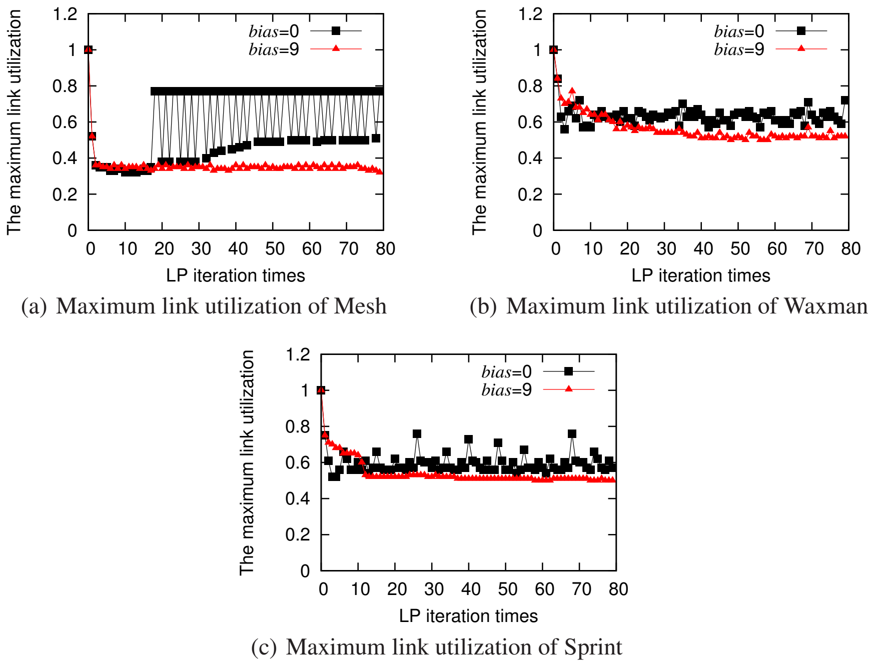
3.4 Incremental LP Iteration

The traffic of the working paths affected by the failures should be shed on the alternative paths locally. If there is no alternative path for recovering the failures, the upstream router can send an ICMP (Internet Control Message Protocol) redirection message to the source, and then the source redirects its traffic to its own alternative paths. However, the traffic unaffected by the failures should remain undisturbed. That is to say, the shifting of the diverted traffic should be done *incrementally*. With the help of $\mathbf{ODL}$ matrix, the detail of the affected traffic can be easily gotten. We re-allocate affected traffic based on the current traffic distribution state with LP once. When LP is finished, the failure is recovered and the affected traffic is shifted onto the backup paths wisely. Then we further refine the traffic allocation with *incremental LP iteration*.

What incremental LP iteration regards as variables is the share of the failed traffic among alternative paths which can be denoted as $\mathbf{X}' = [-x_{jkp}-]$ (**OD** pairs (j, k) are the pairs whose type-0 paths are broken by the failures, $p = 2, 3, \ldots, |\mathcal{P}_{jk}|$). The sum of the variables here is smaller than that in the vector **X**.

Incremental LP iteration does its best to avoid congestions and not to disturb the failure-unaffected traffic being carried by the network simultaneously. Weight-setting method usually decreases the lightly loaded links' weights to attract traffic while increasing the heavily loaded links' weights to reduce traffic. However, changing links' weights may affect a wide range of traffic, including those not directly affected by the failures. In certain network settings, this can easily cause route oscillations or performance degradation.

The speed-up of our work compared with the weight-setting method is three-fold: It replaces ILP with light-weighted LP; It damps oscillations so that the iteration converges quickly; Finally, it balances the diverted load locally without resorting to the time-consuming re-calculation of the routing paths in the whole network.



(a) Maximum link utilization of Mesh (b) Maximum link utilization of Waxman

(c) Maximum link utilization of Sprint

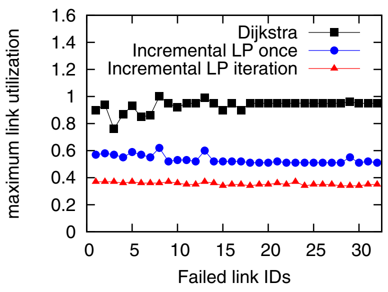
Fig. 4. The comparison between LP iteration with and without *bias*. The *bias* damps the oscillations of LP iteration and makes it converge quickly.

4 Performance Evaluation

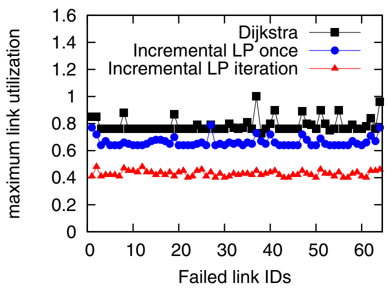
In our experiments, we use three kinds of different topologies mentioned in Section 2: Mesh, Waxman and Sprint. We generate the $\mathbf{OD}$ matrices synthetically according to the Gravity Model ([4], [15]) to conduct simulations to evaluate our scheme's performance.

We first evaluate the convergence of the LP iteration. Figure 4 compares the maximum link load resulted from using different $bias$ values. It clearly shows that without the $bias$ factor, oscillations can easily happen. The punishment from the penalty vector $\mathbf{p}$ is too harsh. The max link load varies greatly with iteration times and the algorithm does not converge. In the Mesh topology, the objective function even begins to get no feasible solution after a few iterations. With the $bias$ factor, LP iteration is able to avoid route oscillations and converges fast. The value of $bias$ can be tuned for particular IP networks as the operator usually has the full knowledge of the network. As to the topologies used in our simulation, the $bias$ 9 is enough to damp the oscillations. Too large $bias$ slows down the convergence of the LP interation. The stop condition can be easily got by setting an upper threshold for the maximum link utilization, e.g., 80%.

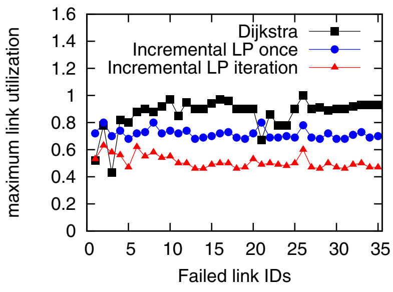
We fail each link independently in the networks of three kinds of topologies separately to evaluate the performance of incremental LP iteration. Figure 5 shows its results in comparison with what are achieved by Dijkstra's algorithm used in the "routing



(a) Maximum link utilization of Mesh

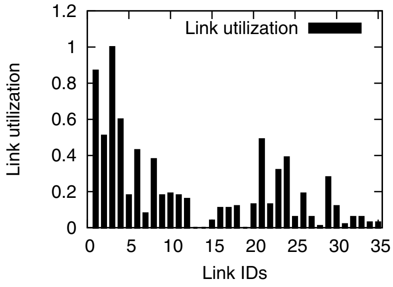


(b) Maximum link utilization of Waxman

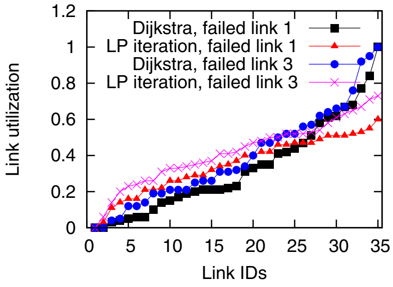


(c) Maximum link utilization of Sprint

Fig. 5. The maximum link utilization when incremental LP iteration and routing convergence happen respectively after each link's failure



(a) Link utilization of Sprint before failures.



(b) Link utilization after routing convergence.

Fig. 6. Link 1 and link 3 are the most loaded links

convergence" after the failure. For most links (except link 1 and link 3 in Sprint topology), the "Incremental LP iteration" achieves much smaller maximum link load than Dijkstra's algorithm. It clearly demonstrates the effectiveness of using incremental LP iteration for post-failure load balancing. The lines with dots show the maximum link utilization when the incremental LP iteration is run one round immediately after the failure. Most of the time, the results of "Incremental LP once" is already much better than that of Dijkstra's.

Link 1 and link 3 are the links with the smallest weights in the Sprint topology derived from [4]. From Figure 6(a), we find that they are the most heavily loaded links. When either one fails, there will be lots of traffic which needs to be shed onto other paths. Conventional routing convergence and the weight-setting method re-calculate all routing paths, resulting in a widespread re-shuffling of traffic and smaller maximum link load. Incremental LP iteration keeps the impact within a small range of the network instead, and in these two particular cases, results in slightly higher link load.

If we let LP iteration take into consideration all traffic regardless of whether they are affected by the failures, our scheme can achieves better load balancing. Figure6(b) shows the results after the failure of link 1 and link 3 in Sprint topology. This time, LP iteration takes into consideration all the traffic carried on the network. Clearly it is able to shed more traffic from heavily loaded links to lightly loaded ones.

5 Conclusion

In this paper, we propose an efficient scheme to achieve load balancing with Proactive Failure Recovery (PFR). The algorithms are devised to set up loop-free alternative paths proactively. The redundancy of Internet offers us enough margin to set up alternative paths to re-allocate the traffic affected by the link failures.

Through Incremental Linear Programming, traffic demands are split among multiple paths optimally for the objective function and hence potential congestions are avoided. After the failed working routes shed their load and the diverted traffic is absorbed by the multiple alternative paths, the load can be further balanced with the help of incremental LP iteration. Our simulation results show that the proposed scheme is effective in reducing the maximum link load in the network after failures.

Acknowlegements

We would like to thank the anonymous reviewers for their feedback. This work was partly supported by NSFC (60573121, 60625201), China 973 program (2007CB310701), the Cultivation Fund of the Key Scientific and Technical Innovation Project, MoE, China (705003), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20060003058), Tsinghua Basic Research Foundation(JCpy2005054), 863 high-tech(2007AA01Z216) and the China/Ireland Science and Technology Collaboration Research Fund (2006DFA11170).

References

1. Clark, D.D.: Design philosophy of the darpa internet protocols. Computer Communication Review 25(1), 102–111 (1995)
2. Francois, P., Filsfils, C., Evans, J., Bonaventure, O.: Achieving sub-second igp convergence in large ip networks. Computer Communication Review 35(3), 35–44 (2005)
3. Kvalbein, A., Hansen, A.F., Cicic, T., Gjessing, S., Lysne, O.: Fast ip network recovery using multiple routing configurations. In: Proceedings - IEEE INFOCOM, Barcelona, Spain, pp. 23–29 (2006)
4. Nucci, A., Bhattacharyya, S., Taft, N., Diot, C.: Igp link weight assignment for operational tier-1 backbones. IEEE/ACM Trans. Networks 15(4), 789–802 (2007)
5. Fortz, B., Thorup, M.: Internet traffic engineering by optimizing ospf weights. In: Proceedings - IEEE INFOCOM, Tel Aviv, Isr, vol. 2, pp. 519–528 (2000)
6. Kvalbein, A., Cicic, T., Gjessing, S.: Post-failure routing performance with multiple routing configurations. In: Proceedings - IEEE INFOCOM, Anchorage, AK, United States, pp. 98–106 (2007)
7. Bertsekas, D., Gallager, R.: Data Networks, 2nd edn. Prentice-Hall, Inc., Upper Saddle River (1992)
8. Chim, T.W., Yeung, K.L., Lui, K.S.: Traffic distribution over equal-cost-multi-paths. Computer Networks 49(4), 465–475 (2005)
9. Iselt, A., Kirstadter, A., Pardigon, A., Schwabe, T.: Resilient routing using mpls and ecmp. In: IEEE Workshop on High Performance Switching and Routing, HPSR, Phoenix, AZ, United States, pp. 345–349 (2004)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
11. Elwalid, A., Jin, C., Low, S., Widjaja, I.: Mate: Mpls adaptive traffic engineering. In: Proceedings - IEEE INFOCOM, Anchorage, AK, vol. 3, pp. 1300–1309 (2001)
12. Shand, M., Bryant, S.: Ip fast reroute framework. IETF Draft, Work in progress (June 2007)
13. Psenak, P., Mirtorabi, S., Roy, A.: Multi-topology (mt) routing in ospf. IETF RFC 4915 (June 2007)
14. Waxman, B.M.: Routing of multipoint connections. IEEE Journal on Selected Areas in Communications 6(9), 1617–1622 (1988)
15. Medina, A., Taft, N., Salamatian, K., Bhattacharyya, S., Diot, C.: Traffic matrix estimation: existing techniques and new directions. In: Proceedings - ACM SIGCOMM 2002, pp. 161–174. ACM, New York (2002)