Traffic Engineering and Routing in IP Networks with Centralized Control

Jing Fu[1], Peter Sjödin[2], and Gunnar Karlsson[1]

[1] ACCESS Linnaeus Center, School of Electrical Engineering
[2] School of Information and Communication Technology
KTH, Royal Institute of Technology, SE-100 44, Stockholm, Sweden
{jing, psj, gk}@kth.se

Abstract. There have been research initiatives in centralized control recently, which advocate that the control of an autonomous system (AS) should be performed in a centralized fashion. In this paper, we propose an approach to perform traffic engineering and routing in networks with centralized control, named *LP-redirect*. LP-redirect is based on an efficient formulation of linear programming (LP) that reduces the number of variables and constraints. As LP is not fast enough for runtime routing, LP-redirect uses a fast scheme to recompute routing paths when a network topology changes. The performance evaluation of LP-redirect shows that it is more efficient in both traffic engineering and computation than an approach using optimized link weights. In addition, LP-redirect is suitable for runtime traffic engineering and routing.

Key words: traffic engineering, routing, centralized control

1 Introduction

Traffic engineering aims at making more efficient use of network resources in order to provide better and more reliable services to the customers. Current traffic engineering and routing in IP networks are performed in both the management and the control planes. In the management plane, network operators configure weights of the links in a network, which indirectly control the selection of routing paths. These weights can be set inversely proportional to the link capacities, as recommended by Cisco [1], or they may be optimized for traffic engineering objective functions in a network with given traffic demands. Computing optimal link weights is NP-hard, therefore heuristics have been developed [2] [3]. However, these heuristics are still computationally intensive: for current backbone networks they may require hours of computation time. When the link weights are computed in the management plane, routing is performed in the control plane based on link-state routing protocols such as OSPF and IS-IS. The forwarding path between two routers is the shortest path considering the sum of the link weights along the path. When a topology changes, as when a link fails, the forwarding paths are recomputed in the routers individually using Dijktra's algorithm [4], which is efficient and can be done in milliseconds. However, this approach does not provide efficient traffic engineering since the link weights are not re-optimized for the new topology.

There have been research initiatives in centralized control recently [5] [6] [7], which advocate that the control of an AS should be performed in a centralized fashion with *direct control*: Instead of manipulating link weights, which influence indirectly the forwarding decisions in individual routers, a centralized server controls the routers' forwarding decisions directly. A centralized control scheme simplifies the decision process, and reduces the functions required in the routers. As the architectural advantages of centralized control are well discussed in these papers, we focus our work on traffic engineering and routing.

In our approach, instead of tuning link weights, the centralized server computes routers' forwarding tables directly. To compute optimal routes for traffic engineering purposes is a multi-commodity network flow problem, which is computationally tractable and can be solved efficiently using linear programming (LP) [8]. Solving an LP problem to obtain optimal routes is much more efficient than computing optimized link weights, and it can be performed in ranges of seconds and tens of seconds. However, LP is still not sufficiently fast for runtime route computation, since new forwarding tables should be computed in no more than hundreds of milliseconds when a topology changes. Therefore, we present a fast scheme to recompute forwarding tables when a topology changes as a link fails. The basic idea is to fast redirect the flows on the failed link based on current link utilizations. Then in a background process, LP can be used to recompute optimal routing paths. After presenting this approach, named *LP-redirect*, we experimentally study its performance and compare it with an approach using optimized link weights [2] [9].

The rest of the paper is organized as follows. Section 2 overviews the related work. Section 3 presents the static routing model, which describes our LP formulation. Section 4 introduces dynamic route computation, which describes how to redirect traffic when a topology changes. Section 5 shows the experimental setup for our performance evaluation. Section 6 presents the results. Finally, section 7 concludes the paper.

2 Related Work

There has been a considerably amount of research in Internet traffic engineering recently. One area of research is to tune OSPF link weights in order to achieve desired traffic engineering objectives [2] [3]. Tuning the link weights relies on having a network topology and an estimate of the traffic demands. Approaches to derive a traffic demand matrix from an operational network are described in [10] [11].

The constraints in OSPF, imposed by using the shortest path and equal splitting of traffic over equal cost paths, make it difficult to achieve optimal routing. Therefore, there are studies on how to split traffic arbitrarily, for example using hash functions [12]. In addition, it has been suggested that the traffic should not only be routed on the shortest path, but also be routed on non-shortest paths, with a penalty for longer paths [13].

Finally, optimal routing can be achieved using LP [8]. Although LP is not fast enough for runtime route computation, it is commonly used as a reference to evaluate other traffic engineering approaches.

3 Static Routing Model

Our network is a directed and connected graph G composed of a set of nodes V and a set of links E. A link $a \in E$ can be represented as $(x,y) \in V \times V$, where x and y are the two end nodes of the link. Furthermore, there is a traffic demand matrix D. For each pair of nodes $(s,t) \in V \times V$ the demand $D(s,t)$ represents the amount of traffic from s to t. For each link a, we let $f_a^{(s,t)}$ denote the amount of traffic from s to t that is offered to link a. In addition, we let f_a^t to denote the amount of traffic from all nodes to t that is offered to link a. The total amount of traffic over link a is denoted f_a and the link capacity is denoted $c(a)$. Finally, the link utilization $u(a)$ could be obtained by dividing the total amount of traffic by the link capacity: $u(a) = f_a/c(a)$.

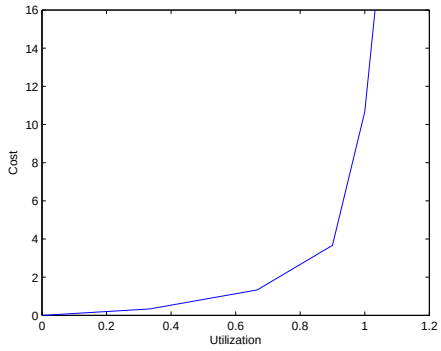


Fig. 1. Link cost ϕ_a as a function of utilization $u(a)$.

The objective of traffic engineering is to keep the traffic loads within the link capacities, and particularly, the loads on the links should be balanced. A frequently used objective function in network optimization is to minimize the total cost in the network, which is known as *minimum cost network flow* [4]. The total cost ϕ is the sum of the cost on each link, $\phi = \sum_{a \in E} \phi_a$. In a link a, the cost function ϕ_a can be modelled using a piecewise linear function of the utilization, as shown in Fig. 1. The idea behind ϕ_a is that it is cheap to transmit flows over a link when the utilization is low. As the utilization increases, it becomes more expensive. The exact definition of the objective function may vary and is a choice for the network operators, however the objective function should be piecewise linear increasing and convex. We have taken the objective function that is used in [2] [3], where $\phi_a(0) = 0$ and its derivative

$$\phi_a'(u(a)) = \begin{cases} 1 \text{ for} & 0 \le u(a) \le 1/3 \\ 3 \text{ for} & 1/3 \le u(a) \le 2/3 \\ 10 \text{ for} & 2/3 \le u(a) \le 9/10 \\ 70 \text{ for} & 9/10 \le u(a) \le 1 \\ 500 \text{ for} & 1 \le u(a) \le 11/10 \\ 5000 \text{ for} & 11/10 \le u(a) \le \infty. \end{cases}$$

The definition above specifies that the cost to transmit a unit amount of traffic over a link with a utilization below 1/3 is 1, when the utilization is above 1/3 but below 2/3,

the cost is 3. Using the above definition, the optimal routing problem can be formulated as the following LP:

$$\text{minimize } \phi = \sum_{a \in E} \phi_a$$

subject to

$$\sum_{x:(x,y)\in E} f^t_{(x,y)} - \sum_{z:(y,z)\in E} f^t_{(y,z)} = \begin{cases} -D(y,t) & \text{if y} \neq \text{t} \\ \sum_{s \in V} D(s,t) & \text{if y = t} \end{cases} \quad y,t \in V, \tag{1}$$

$$f_a = \sum_{t \in V} f^t_a \quad a \in E, \tag{2}$$

$$\phi_a \geq f_a \quad a \in E, \tag{3}$$

$$\phi_a \geq 3f_a - \frac{2}{3}c(a) \quad a \in E, \tag{4}$$

$$\phi_a \geq 10f_a - \frac{16}{3}c(a) \quad a \in E, \tag{5}$$

$$\phi_a \geq 70f_a - \frac{178}{3}c(a) \quad a \in E, \tag{6}$$

$$\phi_a \geq 500f_a - \frac{1468}{3}c(a) \quad a \in E, \tag{7}$$

$$\phi_a \geq 5000f_a - \frac{16318}{3}c(a) \quad a \in E, \tag{8}$$

$$f^t_a \geq 0 \; a \in E; t \in V, \tag{9}$$

Constraints (1) are flow conservation constraints that ensure the desired traffic flow is routed to the destination. Constraint (2) defines the load of a link as the sum of all flows over it. Constraints (3) to (8) define the cost of a link depending on its link capacity and amount of traffic over it. Finally, constraint (9) ensures that each flow is positive.

The above is a complete LP formulation of the general routing problem, and hence it can be solved optimally in polynomial time [14]. In particular, our LP formulation is different from formulations in other papers where each flow is considered as a source-destination pair [2] [3] [15]. Thus, the number of commodities in these approaches is $|V|^2$ and the number of flow variables is $|E| \times |V|^2$. Basically, for each link a and a pair of nodes s and t, there is a flow variable $f^{s,t}_a$ to denote the amount of traffic from source s to destination t that goes via link a. As we model the flows as *flow to a destination*, the number of commodities is $|V|$ and the number of flow variables is $|E| \times |V|$ only. Therefore, for each link a there is a flow variable f^t_a for each destination t. With our formulation, the number of variables and equality constraints in the LP is only $1/|V|$ of the traditional LP formulations, hence it can be solved much more efficiently.

4 Dynamic route computation

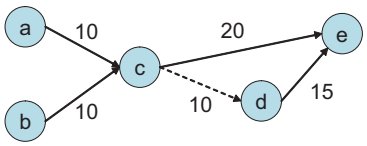
Although LP provides optimal routing, it is not fast enough for runtime route computation. Therefore a fast approach to reroute the packets is required upon link failures.

When a link fails, the first step in LP-redirect is to select a set of *failed flows*. A failed flow is a flow that is considered to be affected by the link failure, and whose traffic needs to be rerouted (whole or in part). When the failed flows have been selected, the next

step is to establish the *residual traffic demand* (*rD*) for those flows. The residual traffic demand of a failed flow represents the amount of traffic that needs to be rerouted for that flow. After that all residual traffic demands are determined, routing paths for these demands are recomputed. After that the packets are rerouted using the fast dynamic approach, LP can be used again in the background to recompute optimal paths. The recomputed optimal paths can be also be used later in routers.

4.1 Finding residual traffic demands

The approaches to find residual traffic demands are best illustrated using an example. Consider a network topology shown in Fig. 2, we study flows towards e when link (c,d) fails. For each link (x,y) in the figure, the value of $f^e_{(x,y)}$ is shown, representing the amount of traffic to e that goes via that link. The initial traffic demands to e are also shown in the figure. When link (c,d) fails, a simple approach is to set residual traffic demand $rD(c,e) = 10$. After that, 10 units of flow should be removed from $f^e_{(d,e)}$, since the flow from c to e was transmitted through the link (d,e) before the failure. An alternative approach is to set residual traffic demand $rD(a,e) = 10$, and remove 10 units of traffic from both $f^e_{(a,c)}$ and $f^e_{(d,e)}$. We could also set residual demands from several sources, for example to set both $rD(a,e) = 5$ and $rD(b,e) = 5$. In this case, we need to remove 5 units of traffic from both $f^e_{(a,c)}$ and $f^e_{(b,c)}$, and remove 10 units of traffic from $f^e_{(d,e)}$.



Traffic demands
D(a,e) = 10
D(b,e) = 10
D(c,e) = 10
D(d,e) = 5

Fig. 2. A network with traffic demands.

When there are several flows to choose from, LP-redirect tries to find flows from links with higher utilization. For example, if $u(a,c) > u(b,c)$, residual traffic demand $rD(a,e) = 10$ will be chosen. This choice leads to higher decrease in cost ϕ when removing the traffic, since the cost of a high-utilized link might be higher than that of a low-utilized link. In addition, LP-redirect prefers to select residual demands for flows where source and destination are further away. Using this approach, residual demands $rD(a,e)$ and $rD(b,e)$ are preferred over $rD(c,e)$. The intuition behind this is that as traffic from more links are removed, the decrease in cost will be higher.

4.2 Recomputing forwarding paths

When the residual traffic demands have been determined, LP-redirect uses Dijkstra's algorithm to compute the shortest paths for these residual demands. The link weights for this computation are obtained as *dynamic weights* for each individual link, which is based on its capacity and current utilization. It is defined as $w(a) = \phi'_a(u(a))/c(a)$. Basically, the dynamic link weight is inversely proportional to the link capacity and is proportional the cost function $\phi'_a(u(a))$ specified in section 3.

One consideration in LP-redirect is how often the dynamic link weights should be updated? The dynamic link weights can be updated when all new routing paths have been computed after a failure, or be updated whenever flows are added to or removed from a link. In addition, if a residual demand $rD(s,t)$ is large, it might be inappropriate to assign the entire flow to a single shortest path, since it might congest the links on the path. An alternative approach is to break down the demand into pieces and assign a part of the demand to the shortest path, update the dynamic link weights, and then recompute the shortest path to reassign remaining demands.

The approach that LP-redirect uses to recompute the paths is as follows. For each node s that has a positive residual demand to any destination t, $rD(s,t) > 0$, the shortest paths from s to all other nodes are computed. If the demand from s to any other node t is smaller than a certain percentage p of the lowest link capacity along the path, $rD(s,t) < p \cdot c(a)$, the entire demand is assigned to the links. Otherwise a part of the demand equal to $p \cdot c(a)$ is assigned to the links. After assigning the demands from the source node s to all destination nodes, the dynamic link weights are recomputed. If the residual demands from s have not been fully assigned, LP-redirect recomputes the shortest-path tree from s, and reassigns the remaining demands. After assigning all demands from s, LP-redirect continues with the next source node with positive residual demand.

The choice of p is also an interesting parameter. When p is too small, demands from a source need to be assigned in many iterations. When p is too large, assigning a large demand on a single path may congest the links. Through experimentation we have found that $p = 1\%$ provides a suitable trade-off for our performance studies.

4.3 Removing potential loops

The forwarding paths computed above can be loopy since they are based both on the initial LP optimization and on shortest paths derived from dynamic link weights. Therefore, before installing the new routing paths, flow loops need to be removed. A flow loop can be detected and removed by using a breadth-first search [4].

5 Evaluation

In this section, we present a method to evaluate the performances of LP-redirect and optimized link weights. Thereafter, we present experimental setups for the performance studies.

5.1 Performance studies

The performance of a solution is expressed using the cost function specified in section 3. We define the minimal cost obtained when solving the LP as the *optimal solution*, ϕ_{opt}. In addition, the minimal cost obtained using optimized link weights is denoted ϕ_{ospf}. We further define *optimality gap* of an approach as its normalized difference in cost to ϕ_{opt}. For example, the optimality gap of optimized OSPF link weights is $(\phi_{ospf} - \phi_{opt})/\phi_{opt}$.

In the dynamic case, let G' be a topology when one or several of the links in G fails. We let ϕ'_{opt} denote the optimal solution obtained by solving the LP with topology G'. Furthermore, we let ϕ'_{ospf} denote the solution obtained using the original optimized link weights in G, but with the new topology G'. Then the optimality gap for OSPF after link failures is $(\phi'_{ospf} - \phi'_{opt})/\phi'_{opt}$. When using LP-redirect, $\phi'_{LP-redirect}$ is computed after redirection of flows.

5.2 Experimental setup

We have created a synthetic router-level topology using the Waxman's model, with 50 nodes and an average link degree of four [16]. In addition, we have used an ISP topology measured in Rocketfuel [17], the Ebone network (*AS 1755*) that consists of 88 routers and 161 links. Finally, we assume uniform link capacities in both topologies.

The traffic demands we used are based on the Newton's gravity model of migration. The model is initially for migration of people between different towns. As the size of the towns increases, there will be an increase in movement between them. The further apart the two towns are, the movement between them will be less. The gravity model and its modified versions have been widely used in communication networks [2] [18]. In our demand matrix generation, each node is given two uniformly distributed random variables o and i for outgoing and incoming traffic. The traffic demand from node s to t is specified as follows:

$$D(s,t) = \alpha \frac{o_s i_t}{d(s,t)^2},$$

where $d(s,t)$ denotes the distance and a constant α is used to scale the demands to appropriate values. Our goal is to set α so that the average link utilizations are about 30% and 70% respectively, in order to evaluate the efficiency of traffic engineering in varying network conditions.

Finally, we have used the GNU linear programming kit (GLPK) for LP modelling and solving [19]. To generate near-optimal link weights, we have used IGPWO [9] that is based on the algorithm in [2].

6 Results

Fig. 3 shows the optimality gap for four setups. The optimality gaps are shown when links fail one after another. We study up to ten link failures from the initial network topology, as the frequency of link failures for a single link can be as low as several minutes [20]. As link failures at different locations may have varying impact on the optimality gap, in each setup, our results are based on 20 simulations, with median, 10th and 80th percentile shown.

The optimality gap for LP-redirect is smaller than optimized weights in all setups. Therefore, LP-redirect provides clearly more efficient traffic engineering. In addition, the optimality gap in the static case (with no link failure) for optimized weights is about 0.1 when average link utilization is 30%, while it is approximately 0.6 when average utilization becomes 70%. This indicates that the performance of optimized link weights decreases when the average link utilization increases.

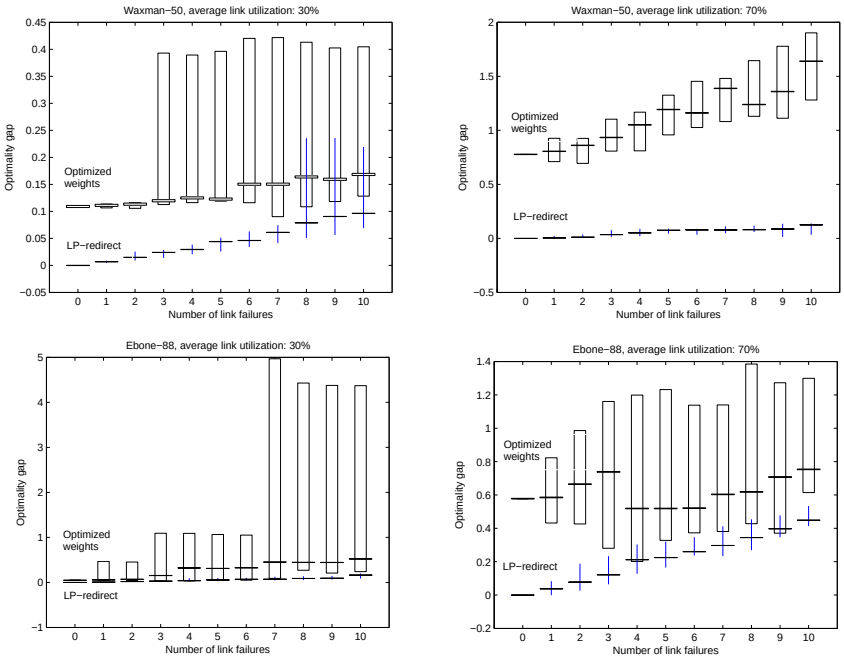


Fig. 3. A comparison of optimality gap between LP-redirect and optimized weights.

In the dynamic case when links fail one after another, the gap for optimized weights increases, in particular in some setups where the 80th percentile gap can be up to 5. These gaps suggest that optimized weights may provide very poor traffic engineering for some of the link failures. While using LP-redirect, the optimality gap increases slowly when links fail, and the 80th percentile gap is fairly close to the median, suggesting that LP-redirect provides efficient traffic engineering for most of the link failures.

7 Conclusion

In this paper, we have presented an approach to perform traffic engineering and routing in networks with centralized control, named LP-redirect. LP-redirect is based on a formulation of LP that reduces the number of variables and constraints, and thus can be solved efficiently, in ranges of seconds to tens of seconds. In addition to the LP formulation, LP-redirect relies on a fast scheme to recompute routing paths when a network topology changes, by identify failed flows and recomputing routes for these flows based on current link utilizations and link capacities.

As a comparison, we have evaluated both LP-redirect and an approach using optimized link weights. The results show that LP-redirect provides more efficient traffic engineering and computation than optimized link weights.

Finally, traffic in a network varies over time and it can be difficult to estimate the traffic demands in advance. In a network with centralized control, the traffic demands and link utilizations can be reported to the centralized sever in runtime, and this infor-

mation can be used by LP-redirect to compute routing paths. While with optimized link weights, when observing a change in traffic demands, it is not possible to re-optimize link weights in a short time range. The unpredictability and abrupt changes of traffic demands make our approach even more suitable for traffic engineering and routing.

References

1. Cisco, *Configuring OSPF*, 2006.
2. B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," IEEE Journal on *Selected Areas in Communications*, vol.20, no.4, pp.756-767, May 2002.
3. M. Ericsson, M.G.C. Resende and P.M. Pardalos, "A genetic algorithm for the weight setting problem in OSPF routing," Journal of *Combinatorial Optimization*, vol.6, no.3, Nov. 2002.
4. T. H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein "Introduction to algorithms, 2nd edition," MIT Press, Cambridge Massachusetts, 2001.
5. J. Rexford, A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang, "Network-wide decision making: Toward a wafer-thin control plane," in Proc. of *ACM SIGCOMM HotNets Workshop*, November 2004.
6. A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "Clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, Volume 35, Issue 5, October 2005.
7. András Császár, Gábor Enyedi, Markus Hidell, Gábor Rétvári, and Peter Sjödin, "Converging the evolution of router architectures and IP networks", *IEEE Network Magazine*, Vol. 21, No. 4, 2007.
8. G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems". in Proc. of *ACM-SIAM Symposium on Discrete Algorithms*, 2002.
9. Interior Gateway Protocol Weight Optimization Tool (IGPWO). Available: http://www.poms.ucl.ac.be/totem/.
10. A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford and F. True, "Deriving traffic demands for operational IP networks: methodology and experience," *IEEE/ACM Transactions on Networking*, v.9 n.3, p.265-280, June 2001.
11. A. Gunnar, M. Johansson and T. Telkamp, "Traffic Matrix Estimation on a Large IP Backbone - A Comparison on Real Data", in Proc. of *ACM SIGCOMM Internet Measurement Conference*, Oct. 2004.
12. Z. Cao, Z. Wang and E. Zegura, "Performance of hashing based schemes for Internet load balancing," in Proc. of *IEEE INFOCOM*, March 2000.
13. D. Xu, M. Chiang and J. Rexford, "DEFT: Distributed exponentially-weighted flow splitting", in Proc. of *IEEE INFOCOM*, May 2007.
14. L. G. Khachian, "A polynomial time algorithm for linear programming," - Doklady Akademii Nauk SSSR, vol 244, pp.1093-1096, 1979.
15. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows: theory, algorithms and applications", Prentice Hall, 1993.
16. B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617-1622, Dec. 1988.
17. N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in Proc. of *ACM SIGCOMM '02*, August 2002.
18. A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: Existing techniques and new directions," in Proc. of *ACM SIGCOMM*, Aug. 2002.
19. The GNU Linear Programming Kit (GLPK). Available: http://www.gnu.org/software/glpk/.
20. G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya and C. Diot, "Analysis of link failures in an IP backbone," in Proc. of *ACM SIGCOMM Internet Measurement Workshop*, 2002.