A Practical Approach to Minimizing Delays in Internet Routing

J.J. Garcia-Luna-Aceves† and **Srinivas Vutukury**§
Computer Engineering Department†
Computer Science Department§
School of Engineering
University of California, Santa Cruz, CA 95064

William T. Zaumen
Sun Microsystems Inc.,
901 San Antonio Avenue, Palo Alto, CA 94043

`http://www.soe.ucsc.edu/research/ccrg/`
`{jj,vutukury}@cse.ucsc.edu, zaumen@eng.sun.com`

Abstract—**We present a practical approach to internet routing that provides near-minimum delays over multiple loop-free paths to destinations. The new protocol, which we call NEAR-OPT, obtains multiple loop-free paths to destinations using long-term delay measures, and allocates destination-oriented flows over such paths using short-term delay measures to minimize delay. We compare the performance of NEAR-OPT with traditional single-path routing and the only known adaptation for dynamic networks of Gallager's minimum-delay routing algorithm. Using actual Internet traffic traces and other traffic source models, we show that NEAR-OPT provides delays comparable to the lower bounds achievable with Gallager's algorithm for static networks, provides *lower* delays than implementations of Gallager's algorithm in networks subject to fractal traffic, and renders far smaller delays and better use of resources than traditional single-path routing. NEAR-OPT does not depend on any global constant and is completely distributed, making it easy to implement as a loop-free "distance-vector" protocol similar to Cisco's EIGRP.**

I. Introduction

Congestion at the links and in the routers is the main cause of large delays in the Internet. All the Internet routing protocols in use today rely on single-path routing algorithms which not only under-utilize resources, but also cannot cope with congestion as all traffic for a destination should be routed through a single successor and when that link becomes congested its whole traffic has to be rerouted. If link costs are made functions of congestion or delays in order to support QoS routing, routing-table entries can become unstable in single-path routing protocols [2], [3].

To diffuse congestion and minimize delays, the routing protocol must provide multiple paths for each destination at each router that are loop-free at every instant and use link costs that are a function of congestion at the links. Loop-free routing paths are crucial because looping, even temporary looping, renders longer delays.

Many optimal routing algorithms exist, but they all assume the input traffic and the network topology to be stationary or very slowly changing, and require global constants that guarantee convergence. This makes optimal-routing algorithms impractical for internetworking, because Internet traffic is very bursty at any time scale, the Internet topology may experience changes, and defining global constants that work for all input traffic patterns are impossible to determine.

Gallager [7] proved the necessary and sufficient conditions for minimum-delay routing and described a distributed multipath algorithm that is loop-free at every instant and obtains minimum delays when input traffic and the network topology are stationary or slowly changing (quasi-static). The basic result shows that, to avoid oscillatory behavior and to achieve minimum delays, the routing protocol must provide multiple paths for each destination at each router that are loop-free at every instant and use marginal(incremental) delays as dynamic link costs and it must incrementally shift traffic from congested links to less congested links. Segall and Sidi [11], [12] extended Gallager's minimum-delay routing algorithm to handle topological changes using techniques developed by Merlin and Segall [9]. In this paper, we call this algorithm GSS and compare it against our approach. These algorithms, however, are not suited for practical use in the Internet because of two critical requirements: speed of convergence and relative insensitivity of performance to variations in input traffic. In Bertsekas and Gallager [1], an improved version of Gallager is presented that converges quicker, but the algorithm is still limited to small networks due to dependence on global constants.

Our basic design approach in NEAR-OPT protocol consists of first establishing multiple loop-free paths to a destination at each router using long-term delay information, and then allocating flows to the destination over next-router choices available at each router for a destination using short-term delay information. The complexity of implementing NEAR-OPT is similar to the complexity of routing protocols that provide single-path routing.

Section II formulates the minimum-delay routing problem. Section III discusses Gallager's minimum-delay routing algorithm and its drawbacks. Section IV describes NEAR-OPT routing which adapts techniques in Gallager's algorithm to the Internet while overcoming its drawbacks. Section V present simulation results that demonstrate the advantages of NEAR-OPT over single-path routing and Gallager's algorithm under dynamic traffic conditions.

II. Problem formulation and background

The minimum-delay routing problem can be formulated as follows [7]. A computer network $G = (N, L)$ is made up of N routers and L links between them. Each link is bidirectional with possibly different costs in each direction.

Let $r_j^i \geq 0$ be the expected input traffic, measured in bits per second, entering the network at router i and destined for router j. Let t_j^i be the sum of r_j^i and the traffic arriving from the neighbors of i for destination j. Finally, let routing parameter ϕ_{jk}^i be the fraction of traffic t_j^i that leaves router i over link (i, k). Assume that the network does not lose any packets. Then, from conservation of traffic we have

$$t_j^i = r_j^i + \sum_{k \in N^i} t_j^k \phi_{ji}^k \quad (1)$$

where N^i is the set of neighbors of router i.

Let f_{ik} be the expected traffic, measured in bits per second, on link (i, k). Because $t_j^i \phi_{jk}^i$ is the traffic destined for router j on link (i, k) we have the following equation to find f_{ik}.

[0] This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grants F30602-97-1-0291 and F19628-96-C-0038.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **1999**	2. REPORT TYPE	3. DATES COVERED **00-00-1999 to 00-00-1999**
4. TITLE AND SUBTITLE **A Practical Approach to Minimizing Delays in Internet Routing**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**5**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

$$f_{ik} = \sum_{j \in N} t^i_j \phi^i_{jk} \tag{2}$$

Note that $0 \leq f_{ik} \leq C_{ik}$ where C_{ik} is the capacity of link (i,k) in bits per second.

Property 1: For each router i and destination j, the routing parameters ϕ^i_{jk} satisfy the following conditions:

1. $\phi^i_{jk} = 0$ if $(i,k) \notin L$ or $i = j$. Clearly if the link does not exist, there will be no traffic on it.
2. $\phi^i_{jk} \geq 0$. Negative amount of traffic cannot, of course, be allocated.
3. $\sum_{k \in N^i} \phi^i_{jk} = 1$. All traffic must be allocated.

Let D_{ik} be defined as the expected number of messages or packets per second transmitted on link (i,k) times the expected delay per message or packet, including the queuing delays at the link. We assume that messages are delayed only by the links of the network and D_{ik} depends only on flow f_{ik} through link (i,k) and link characteristics such as propagation delay and link capacity. $D_{ik}(f_{ik})$ is a continuous and convex function. The total expected delay per message times the total expected number of message arrivals per second is given by

$$D_T = \sum_{(i,k) \in L} D_{ik}(f_{ik}) \tag{3}$$

Note that the router traffic flow set $t = \{t^i_j\}$ and link flow set $f = \{f_{ik}\}$ can be obtained from $r = \{r^i_j\}$ and $\phi = \{\phi^i_{jk}\}$. D_T can therefore be expressed as a function of r and ϕ using (1) and (2). The minimum-delay routing problem can now be stated as follows: For a given fixed topology with link capacities C_{ik} and input traffic flow set r, and delay function $D_{ik}(f_{ik})$ for each link (i,k), the minimization problem consists of computing the routing parameter set ϕ such that the total expected delay D_T is minimized.

III. A Minimum Delay Algorithm

To solve the minimization problem described in the previous section, Gallager [7] derived the necessary and sufficient conditions and described an algorithm to compute routing parameter set ϕ such that those conditions are satisfied. To find the conditions, first obtain the partial derivatives of the total delay D_T of (3) with respect to r and ϕ, that is,

$$\frac{\partial D_T}{\partial r^i_j} = \sum_{k \in N^i} \phi^i_{jk}[D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r^k_j}] \tag{4}$$

$$\frac{\partial D_T}{\partial \phi^i_{jk}} = t^i_j[D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r^k_j}] \tag{5}$$

where $D'_{ik}(f_{ik}) = \frac{\partial D_{ik}(f_{ik})}{\partial f_{ik}}$ and is called as marginal or incremental delay. $\frac{\partial D_T}{\partial r^i_j}$ is the marginal distance from node i to j. The Gallager theorem is stated here for discussion.

Theorem 1: (*Gallager[7]*): The necessary condition for a minimum of D_T with respect to ϕ for all $i \neq j$ and $(i,k) \in L$ is

$$\frac{\partial D_T}{\partial \phi^i_{jk}} = \begin{cases} = & \lambda_{ij} \quad \phi^i_{jk} > 0 \\ \geq & \lambda_{ij} \quad \phi^i_{jk} = 0 \end{cases} \tag{6}$$

where λ_{ij} is some positive number, and the sufficient condition to minimize D_T with respect to ϕ is for all $i \neq j$ and $(i,k) \in L$ is

$$D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r^k_j} \geq \frac{\partial D_T}{\partial r^i_j} \tag{7}$$

□

Equation (4) shows the relation between node's marginal distance and the marginal distances of neighbors to a particular destination. Equations (5), (6) and (7) indicate that under *perfect load balancing*, ie., when routing parameter set ϕ gives the minimum delay, the marginal distances through neighbors in the successor set are equal, and the marginal distances through neighbors *not* in the successor set are higher than those in the successor sets. Let D^i_j denote the marginal distance $\partial D_T/\partial r^i_j$ from i to j. Let l^i_k denote the marginal delay $D'_{ik}(f_{ik})$ as the cost of the link from i to j. Let S^i_j be the set of neighbors through which router i forwards traffic towards j. Now the minimum delay routing problem becomes one of determining routing parameters $\{\phi^i_{jk}\}$ at each node i for each destination j, such that the following equations are satisfied.

$$D^i_j = \sum_{k \in N^i} \phi^i_{jk}(D^k_j + l^i_k) \tag{8}$$

$$S^i_j = \{k | \phi^i_{jk} > 0 \wedge k \in N^i\} \tag{9}$$

$$D^i_j \leq D^k_j + l^i_k \qquad k \in N^i \tag{10}$$

$$(D^p_j + l^i_p) = (D^q_j + l^i_q) \qquad p, q \in S^i_j \tag{11}$$

$$(D^p_j + l^i_p) < (D^q_j + l^i_q) \qquad p \in S^i_j \quad q \notin S^i_j \tag{12}$$

Gallager [7] describes a distributed routing algorithm for stationary and quasi-stationary networks based upon the above equations. Segall and Sidi [11], [12], extended Gallager's basic algorithm to handle topological changes. We refer to this algorithm as GSS. A detailed description of Gallager's algorithm and its extensions handling topological changes are presented in [7], [11]. GSS, however, is not practical for several reasons. A major drawback of GSS is that a global step size η needs to be chosen and every router must use it to ensure convergence; because this constant depends on the traffic pattern, it is impossible to determine one in practice that works for all input traffic pattern. In GSS, the routing parameters are directly computed and the multiple loop-free paths are indirectly implied by the routing parameters in (9). The computation of routing parameters in GSS is, for all practical purposes, a very slow process because it is a destination-controlled process; i.e., the destination must initiate every iteration that adjusts the routing parameters at every router, making the algorithm slow converging and useful only for quasi-static routing, where traffic and topology are static long enough for all routers to adjust their routing parameters between changes. In a network with a large diameter, each iteration will take a time proportional to the diameter of the network and a number of messages proportional to number of links. Depending on the global constant η, several iterations are needed to converge to the final routing tables. The number of iterations needed for convergence tends to be large for a small η, and small for a large value of η. However, η cannot be made arbitrarily large to reduce number of iterations and to speed up convergence, because the algorithm may not converge at all for large η. Hence, GSS is basically a method for obtaining lower bounds under stationary traffic rather than an algorithm for use in practical networks. In the next section, we show how the theory and technique available in the Gallager method can be adapted to practical networks, while overcoming its drawbacks.

IV. Near-optimum Delay Routing

We noted that in GSS the computation of the routing parameter set ϕ was slow converging and works only in the case of stationary or quasi-stationary traffic. In the Internet, traffic is hardly stationary and perfect load balancing is neither possible nor necessary. An *approximate* load balancing scheme based on some heuristic which can be quickly adapt to dynamic traffic is actually sufficient to minimize delays substantially. The key idea in our approach is that multiple paths are *first* computed by other means and then routing parameters are assigned to each successor using some heuristics, i.e., the process of computing successor sets is decoupled from computation of routing parameters. This gives NEAR-OPT the ability to quickly respond to traffic bursts using short-term metrics while providing shortest paths based on long-term metrics. So in our approach, we approximate (8) to (12) with the following:

$$\begin{aligned} D_j^i &= min\{D_j^k + l_k^i \mid k \in N^i\} & (13)\\ S_j^i &= \{k \mid D_j^k < D_j^i \land k \in N^i\} & (14)\\ \phi_{jk}^i &= \Psi(k, A_j^i, B_j^i) \quad k \in N^i & (15)\end{aligned}$$

where $A_j^i = \{D_j^p + l_p^i \mid p \in N^i\}$ and $B_j^i = \{\phi_{jp}^i \mid p \in N^i\}$. We observe that (13) is the Distributed Bellman-Ford (DBF) equation for computing the shortest path and (14) is the successor set that includes the neighbors that are closer to the destination than the node itself. Note that this automatically includes the neighbor that offers the shortest path. The function Ψ in (15) is a heuristic function that finds the routing parameters.

It is well documented that routing algorithms based on DBF suffer from severe performance problems due to count-to-infinity and looping problems [13]. To confront these problems NEAR-OPT uses diffusing computations first described by Dijkstra-Scholten[5], and extends the basic mechanism to permit multiple successors per node for each destination. Because of space limitation, to see how NEAR-OPT computes D_j^i and S_j^i, the reader is referred to a detailed description and correctness proof of an algorithm called DASM [14]. To understand the key point of this paper it is sufficient to know that NEAR-OPT computes S_j^i such that the successor graph to the destination j implied by them is loop-free at every instant.

A. Handling Network Congestion

With variable traffic, the flows over links are continuously changing, causing continuous link cost changes. To respond to these changes, queuing delays at the links should be measured periodically and routing paths recomputed accordingly. However, frequent recomputation of routing paths consumes excessive bandwidth and computation resources because of which routing path changes should only be done at sufficiently long intervals. Unfortunately, a network cannot be responsive to short-term traffic bursts if only long-term updates are used. Therefore, NEAR-OPT performs two types of link-cost updates: short-term updates which are made every T_s seconds, and long-term updates which are made every T_l seconds. In general $T_s << T_l$. The long-term updates are designed to handle long-term traffic changes and result in updating the successor sets at each router, such that the new routing paths are the shortest paths.

The short-term updates made every T_s seconds are designed to handle short-term traffic fluctuations that occur between long-term routing path updates and are used to compute the routing parameters ϕ_{jk}^i in (15) at each router; successor sets and, therefore, successor graphs do not change. Since traffic allocation heuristic in (15) is a function of the congestion at the links it effectively reduces traffic on congested links and increases traffic on less congested links. Ψ in general can be any function that satisfies Property 1, but to get maximum performance it must be made a function of the marginal distances through the successors. Various heuristics can be used to compute the routing parameters; we present one such heuristic later in this section.

Unlike η in GSS, T_l and T_s are local constants that are set independently at each router. Also, they need not be static constants and can be made to vary according to level of congestion at the node. The value of T_l, however, should be such that it is sufficiently longer than the time it takes for NEAR-OPT to compute shortest paths. The long-term update periods should be phased randomly at each router, because of the problems that would result due to synchronization of updates [2].

```
**Algorithm** $IH$
    (1) For each $k \notin S_j^i$ set
        $\phi_{jk}^i \leftarrow 0$.
    (2) If $|S_j^i| = 1$, then for $k \in S_j^i$ set
        $\phi_{jk}^i \leftarrow 1$.
    (3) If $|S_j^i| > 1$, then for each $k \in S_j^i$ set
        $\phi_{jk}^i \leftarrow \frac{1 - \frac{D_{jk}^i + l_k^i}{\sum_{m \in S_j^i} (D_{jm}^i + l_m^i)}}{(|S_j^i| - 1)}$.
**End** IH
```

Fig. 1. Heuristic for initial load assignment.

In the minimization problem in Section III the link-cost metric used is the *marginal delay*, which is the rate of change of delay with respect to flow through the link. That is, if $D_{ik}(f_{ik})$ is the delay of link (i,k) as a function of flow in the link, then the marginal delay is $D'_{ik}(f_{ik}) = \frac{\partial D_{ik}}{\partial f_{ik}}$. In NEAR-OPT, therefore, we use marginal delays for link costs.

For the purposes of simulations and providing intuition we have approximated the link behavior to an $M/M/1$ queue. We, therefore, obtain the following formula for marginal delay at a link [2], where τ_{ik} is the propagation delay of link (i,k).

$$D'_{ik}(f_{ik}) = \frac{C_{ik}}{(C_{ik} - f_{ik})^2} + \tau_{ik} \qquad (16)$$

Although (16) is useful for describing the intuition in NEAR-OPT, it is limited because links are hardly $M/M/1$ in real networks due to the characteristics of the traffic over the links. Accordingly, in real implementation of our approach, the marginal delay of a link, $D'_{ik}(f_{ik})$, should be computed on-line by measuring traffic and the delays at the link. Segall [11] and Cassandras [4] suggests some practical techniques for estimating $D'_{ik}(f_{ik})$. Average link costs are measured over two different intervals; link costs measured over short period T_s are used for routing parameter computation and link costs measured over long period T_l are used for routing path computation.

B. Assigning Flows to Successor Sets

We now describe how ϕ_{jk}^i in (15) are computed using Ψ. When S_j^i is computed for the first time or recomputed again due to long-term route changes, traffic should be freshly distributed among the successors. In this case, since traffic is not already distributed the allocation heuristic function Ψ is a function of only the marginal distances through the successor set. That is (15) reduces to the form $\{\phi_{jk}^i\} = \Psi(k, \{D_j^p + l_p^i \mid p \in N^i\})$. When a new successor set S_j^i is computed, algorithm IH in Fig. 1 is first used to distribute traffic over the successor set. Note that $\{\phi_{jk}^i\}$ computed in IH satisfy Property 1. Furthermore, when more than one successor is present, if $D_{jp}^i + l_p^i > D_{jq}^i + l_q^i$ for successors p and q, then $\phi_{jp}^i < \phi_{jq}^i$. The heuristic makes sense because the greater the marginal delay through a particular neighbor becomes, the smaller the fraction of traffic that is forwarded to that neighbor.

After the first flow assignment is made over a newly computed successor set using algorithm IH, a different flow allocation heuristic algorithm AH shown in Fig. 2 is used to adjust the routing parameters every T_s seconds until the successor set changes again. The heuristic function Ψ computed in AH is incremental and, unlike IH, is a function of current flow allocation on the successor sets and the marginal distances

```
Algorithm $AH$
    (1) Find $D^{ij}_{min} \leftarrow min\{D^i_{jk} + l^i_k \,|k \in S^i_j\}$.
        Let $k_0$ be the neighbor that offers this minimum.
        That is $D^{ij}_{min} = (D^i_{jk_0} + l^i_{k_0})$.
    (2) Let $a^i_{jk} \leftarrow D^i_{jk} + l^i_k - D^{ij}_{min}$ for each $k \in S^i_j$.
    (3) Let $\Delta \leftarrow \frac{1}{2} min\{\frac{\phi^i_{jk}}{a^i_{jk}} | k \in S^i_j \wedge a^i_{jk} \neq 0\}$
    (4) For each $k \neq k_0 \wedge k \in S^i_j$ set $\phi^i_{jk} \leftarrow \phi^i_{jk} - \Delta \times a^i_{jk}$
    (5) For $k = k_0$ set $\phi^i_{jk} \leftarrow \phi^i_{jk} + \sum_{q \in S^i_j} \Delta \times a^i_{jq}$.
End AH
```

Fig. 2. Heuristic for incremental load adjustment.

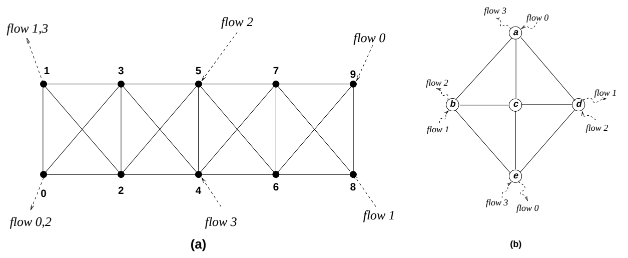

Fig. 3. The network models used in simulations

through the successors. AH also preserves property 1 at every instant. In AH traffic is incrementally moved from the links with large marginal delays to links with the least marginal delay. The amount of traffic moved away from a link is proportional to how large the marginal delay of the link is compared to the best successor link. The heuristic tends to eventually distribute traffic in such a way that the marginal delays of all the successors are equal which is indeed the necessary condition in the Gallager theorem.

V. Simulations

In this section, we address the average performance of NEAR-OPT, which is a direct consequence of its use of multiple loop-free paths and the flow allocation heuristic. We performed a series of simulations to compare the delay and throughput of NEAR-OPT, GSS and a single-path routing algorithm DUAL[8] under identical settings. We chose DUAL for comparative study because it provides loop-free single paths at every instant and it is an algorithm being used in an existing Internet routing protocol, EIGRP [6]. Furthermore, DUAL has been shown to converge faster than the traditional topological broadcast protocols (e.g., OSPF) after link-cost changes [13], and constitutes a better single-path routing approach with which to compare NEAR-OPT.

The network topology models for which we present simulation results in this paper are shown in Fig. 3. These small network topology was chosen to speed up simulations while not favoring NEAR-OPT, and have some similarities with the sparse connectivity of routers in the Internet. All the links have the same link capacity of 1.0Mbs. The propagation delay of the links is 10μs. The long-term update period T_l is set at 1.0s, while the short-term update period T_s is set at 0.1s. We used (16) for the marginal delay in our simulations for simplicity and it does not favor any particular algorithm.

A. Experiment I

In this experiment we compare NEAR-OPT, GSS and DUAL under stationary traffic. The flows are exponential with constant rate. Flow-0 starts at router 9 and is destined for router 0. Flow-1 starts at router 8 and is received at router 1. Flow-2 and flow-3 are intervening flows,

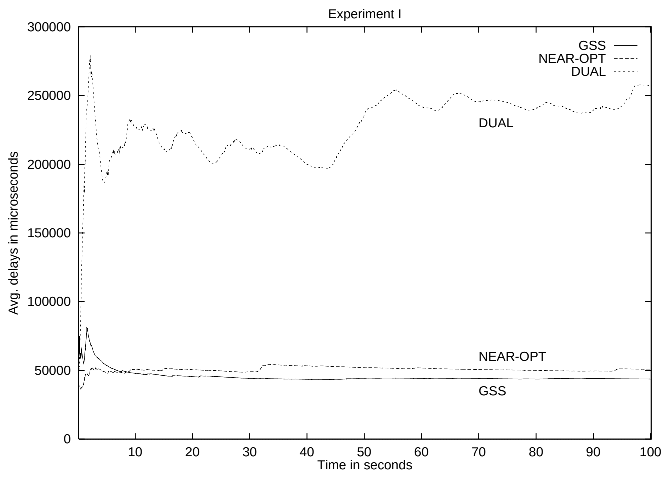

Fig. 4. Avg. delays under stationary traffic

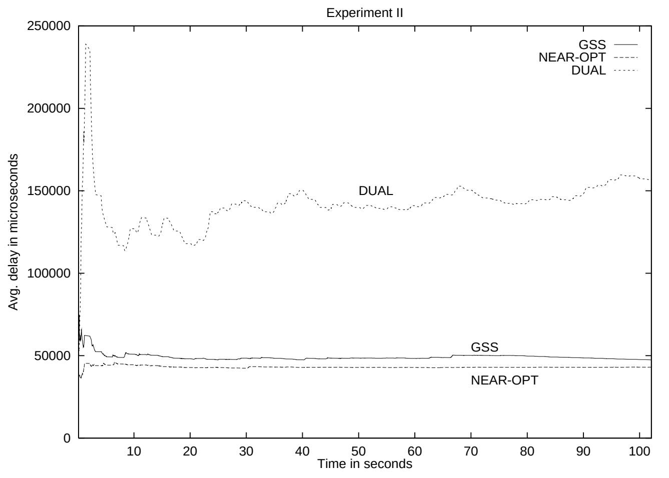

Fig. 5. Avg. delays under variable traffic

that start at routers 5 and 4 respectively and are received at routers 0 and 1, respectively. The flow in the link is obtained by measuring the number of bits (data and control) that are transmitted on the link in a time interval. The link capacities, propagation delays and data packet sizes are kept constant and are the same for all algorithms.

The delays obtained for NEAR-OPT, GSS and DUAL for flow-0 are shown in the Fig. 4. We observe that NEAR-OPT and GSS significantly outperform DUAL, because of the use of multiple paths. We see that GSS performs better than NEAR-OPT, but the delays from NEAR-OPT are within 20% of those of GSS. An important observation from this experiment is that the response of NEAR-OPT to a step function is far better than GSS. In this case, the step function consists of starting a given flow at time 0. The reason NEAR-OPT has a much less underdamped response to drastic changes in input traffic is that the propagation of routing information is much faster in NEAR-OPT than in GSS. Another observation is that the difference in delays obtained in NEAR-OPT and GSS after GSS attains the initial flow assignments on the routers is always smaller than the initial deficit in GSS. This predicts that NEAR-OPT should have a favorable performance over GSS when either traffic is very bursty or the topology experiences drastic changes. Note that NEAR-OPT's performance compared to DUAL can be even better if the connectivity, ie., the average node degree is higher, because then NEAR-OPT can use more successors.

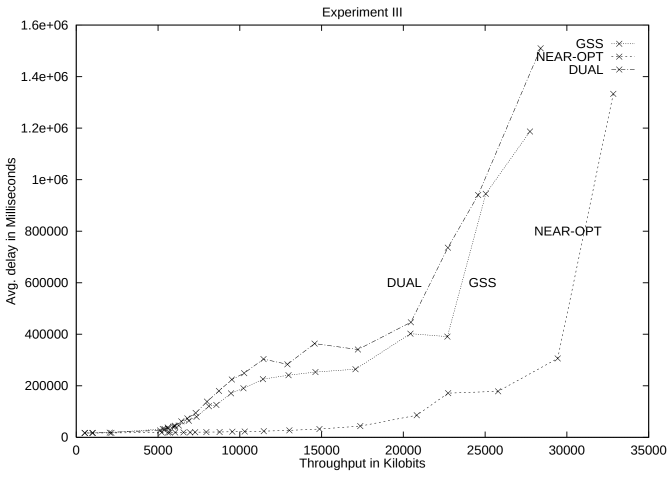

Fig. 6. Throughput comparison

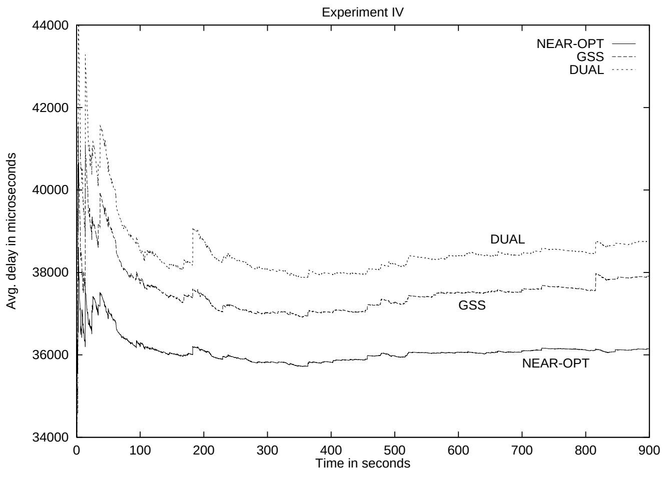

Fig. 7. Avg. delays for flow-0 under internet traffic

B. Experiment II

This experiment studies the responsiveness of the protocols under bursty traffic by repeating Experiment I using ON-OFF traffic pattern for the flows. The delays obtained for flow-0 are shown in Fig. 5 for NEAR-OPT, GSS and DUAL. We observe that GSS and NEAR-OPT again outperformed DUAL by virtue of multiple paths. NEAR-OPT's better performance under variable traffic is due to the slower propagation of long-term path information in GSS. Note that the network on which this simulation is performed is small; in a more realistic topology the slow convergence of GSS becomes more pronounced and comparatively NEAR-OPT performs even better than this experiment shows. This experiment indicates that the basic intuition in NEAR-OPT's design is correct, i.e., building loop-free paths using long-term delay information first and then assigning flows based on heuristics is appropriate, because the latency of making routers agree on loop-free paths dominates the latency of the local flow assignment decisions at each router.

C. Experiment III

In this experiment, we measure throughput of NEAR-OPT, GSS and DUAL. The throughput of the network is the total data delivered by all flows in the network in a given time interval. Fig. 3(b) shows the network and the flows used in this experiment. We use smaller network because it is easy to saturate the network. The flows use ON-OFF traffic with exponential distribution during the ON period. Fig. 6 shows that, for a given delay, a much greater throughput and hence greater network utilization can be achieved in NEAR-OPT than in GSS and DUAL.

D. Experiment IV

This experiment repeats experiment II, but uses actual Internet traffic traces obtained from Lawrence Berkeley National Laboratory [10]. Fig. 7 shows the results when traffic traces extracted from traffic trace LBL-PKT-4 are used for flows 0, 1, 2 and 3. We can see that NEAR-OPT outperforms GSS and DUAL which is expected from the results in Experiment II and III, because traffic is only made more bursty in this experiment. This points to the fact that GSS is basically a method for obtaining lower bounds under stationary traffic, and that NEAR-OPT is fast enough to take advantage of multiple paths with very bursty traffic and can dramatically increase performance in current internets.

VI. Conclusions

We have presented a practical approach to near-minimum delay routing in computer networks and in the Internet. We first described Gallager's method to derive the lower bounds for delays that are theoretically possible. We pointed out the drawbacks of the algorithms directly based on Gallager's method, namely, slow convergence and requirement of a global constant η for ensuring convergence and rectified them in our algorithm.

Simulation results show that NEAR-OPT performs significantly better than single-path routing, and that NEAR-OPT performance comes within a small percentage range of the lower bound delays under stationary traffic, and that under varying traffic conditions, NEAR-OPT performs better than GSS.

In closing, it must be emphasized that, although theoretically minimum delays cannot be achieved in practice simply because of the latencies involved in propagation of routing information, NEAR-OPT demonstrates that far better performance can be obtained using practical approaches similar to those already used in the Internet for single-path routing.

References

[1] D. Bersekas and R. Gallager. Second Derivative Algorithm for Minimum Delay Distributed Routing in Networks. *IEEE Trans. Commun.*, 32:911–919, 1984.

[2] D. Bersekas and R. Gallager. *Data Networks.* Prentice-Hall, 1992.

[3] D. Bertsekas. Dynamic Behavior of Shortest-Path Algorithms for Communication Networks. *IEEE Trans. Automatic Control*, 27:60–74, 1982.

[4] C.G. Cassandras, M.V. Abidi, and D. Towsley. Distributed Routing with Onn-Line Marginal Delay Estimation. *IEEE Trans. Commun.*, 18:348–359, March 1990.

[5] E.W.Dijkstra and C.S.Scholten. Termination Detection for Diffusing Computations. *Information Processing Letters*, 11:1–4, August 1980.

[6] D. Farinachi. Introduction to enhanced IGRP(EIGRP). *Cisco Systems Inc.*, July 1993.

[7] Robert G. Gallager. A Minimum Delay Routing Algorithm Using Distributed Computation. *IEEE Trans. Commun.*, 25:73–84, January 1977.

[8] J.J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Trans. Networking*, 1:130–141, February 1993.

[9] P. M. Merlin and A. Segall. A Failsafe Distributed Routing Protocol. *IEEE Trans. Commun.*, 27:1280–1287, September 1979.

[10] V. Paxson, P. Danzig, J. Mogul, and M. Schwartz. http://ita.ee.lbl.gov/html/traces.html. *Lawrence Berkeley National Laboratory*, July 1997.

[11] A. Segall. The Modeling of Adaptive Routing in Data Communication Networks. *IEEE Trans. Commun.*, 25:85–95, January 1977.

[12] A. Segall and M. Sidi. A Failsafe Distributed Protocol for Minimum Delay Routing. *IEEE Trans. Commun.*, 29:689–695, May 1981.

[13] W. Zaumen and J.J. Garcia-Luna-Aceves. Dynamics of Link-State and Loop-Free Distance-Vector Routing Algorithms. *Journal of Internetworking*, 3:161–188, 1992.

[14] W. T. Zaumen and J.J. Garcia-Luna-Aceves. Loop-Free Multipath Routing Using Generalized Diffusing Computations. *Proc. IEEE INFOCOM*, March 1998.