Trail Blazer: A Routing Algorithm Inspired by Ants

Eran Gabber
Google, Inc.
1440 Broadway, 21st Floor
New York, NY 10018
eran@google.com

Mark A. Smith
Lucent Technologies – Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
marks@research.bell-labs.com

Abstract

We propose a new intra-domain IP routing algorithm called TRAIL BLAZER *(*TB*) that alleviates network congestion by local decisions based on latency measurements collected by scout packets.* TB *is a member of a class of traffic-aware routing algorithms based on the behavior of ants.* TB *maintains in every router a probability table that controls the selection of outgoing links for a given destination. This table is modified by passing scout packets. Some scout packets follow existing high probability paths, and other scout packets explore new paths by making random "mistakes" in order to find detours around congested areas of the network. Scout packets do not have global knowledge of the network topology. Rather, they are influenced by probability trails left by previous scout packets, and leave a trail of updated probability information.* TB *is meant to be an extension of existing link-state protocols such as OSPF or IS-IS, which provide shortest-path information to initialize the probability table. Simulations of UDP traffic in congested networks show a reduction of packet drops by factors of 3.4–38.4 and 1.8–8.2, for single-path and multi-path shortest-path routing, respectively, with a bounded reordering window. Even though* TB *may cause packet reordering which may affect the congesting control mechanisms of TCP, TCP traffic sent over* TB *has a similar bandwidth to shortest-path routing. Simulation of a hybrid routing algorithm that routes TCP traffic over shortest paths and UDP traffic over* TB *show reduced interference between TCP and UDP traffic.*

1. Introduction

Many autonomous systems (AS) suffer from transient or permanent link congestion due to increasing or shifting workloads, bursty traffic or link failures. An example of a transient congestion is a flash crowd [12], which is characterized by a sudden increase in the number of people trying to access the same site.

The dominant AS routing algorithm today, OSPF [17] (Open Shortest Path First), is a link-state protocol. It sends traffic along the shortest path to the destination, where path length is determined by either hop count or link weights. Link weights are often set to be inversely proportional to the link bandwidth, or the weights are set by the network administration based on observed traffic patterns [11]. The problem is that traffic patterns are often dynamic, and may not match the precomputed link weights, which causes OSPF to send more traffic blindly on congested shortest paths.

Network congestion was traditionally solved by adjusting the transmission rate by a congestion control protocol, such as TCP. However, congestion could be avoided by dynamic traffic-aware protocols, which adjust routing to avoid congested links. Potential-based routing [2] and the "new" ARPANET routing protocol [15] are examples of such routing algorithms.

We suggest a new traffic-aware routing algorithm called TRAIL BLAZER (TB)[1] that adapts its routing tables according to measured packet latency to avoid congestion and to reduce latency. TB belongs to a large class of distributed algorithms that mimic the behavior of social insects, like ants or bees. These algorithms are often called swarm intelligence [4] or Ant Colony Optimization (ACO) [7]. These algorithms emulate the principle of stigmergy [10], which is the indirect communication between agents by changes to their environment. For example, ants find the shortest path from their nest to a food source without any knowledge of geometry but with a keen sense of smell. A large number of ants try random paths and deposit pheromones along their paths. Any ant that finds the food retraces its path to the nest. The shortest path to the food has the strongest smell, which future ants follow.

TB employs link probability tables that determine the probability of selecting an outgoing link for a particular des-

1 TRAIL BLAZER is simply named after its most distinguishing activity, which is to discover new paths, or blaze new trails.

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE

tination. TB assumes that an underlying link-state protocol, such as OSPF, is always active. The link-state protocol initializes the link probability table with shortest-path information and adjusts the probability table after topology changes, such as link failures or additions. The initial value of the link probability table ensures that shortest paths have the highest probability. TB sends scout packets, which are allowed to deviate from the highest probability paths in order to discover detours with smaller latency. Scout packets measure the latency to the destination and update the link probability tables accordingly. A scout packet that discovers a path with a lower latency increases the probability along its path, whereas a scout packet that discovers a path with a higher latency decreases the probability along its path.

This paper has several contributions. First, it shows that TB reduces the packet drop rate significantly relative to single-path and multiple-path shortest-path routing. Second, it shows that a reordering buffer of 75 slots is sufficient to alleviate out-of-order delivery of UDP packets routed by TB in the networks we simulated. Third, it shows that TCP traffic routed with TB has similar bandwidth to single-path and multiple-path shortest-path routing in congested networks. Fourth, it shows that a hybrid routing algorithm that routes TCP traffic over the shortest paths and UDP traffic with TB offers improved TCP bandwidth and reduced UDP packet drop rate relative to shortest-path routing of all traffic. The hybrid routing algorithm could be the most effective way of introducing a new routing algorithm in existing networks.

The rest of the paper is organized as follows. Section 2 compares TB with existing routing algorithms. Section 3 explains the concepts of the TB routing, and Section 4 explains the operation of TB in detail. Section 5 discusses the properties of TB, and Section 6 presents preliminary simulation results of UDP and TCP traffic over TB and shortest-path routing on several network topologies and several traffic patterns. Section 7 concludes and discusses future work.

2. Related work

Dynamic routing algorithms that try to alleviate congestion or guarantee quality of service based on network traffic measurements are not new. The "new" ARPANET routing protocol [15] computed link weights from link delay measurements, and recomputed routing tables when link delays changed significantly. However, this protocol caused wide oscillations in link utilization under heavy load. Khanna and Zinky [13] revised the delay metric to improve the stability of this protocol. Since then, many other adaptive algorithms have been proposed in the literature.

Several routing algorithms based on the behavior of ants have been published in the artificial intelligence community. AntNet [5] is a routing algorithm that employs two-way mobile agents to update link probability tables, which is similar to TB. The two-way agents are sent from the source to the destination, and they update the link probability tables on their return trip. The main difference between TB and AntNet is that TB does not require a lengthy learning period to discover the network topology, since TB assumes that the underlying link-state protocol provides this information. Moreover, the AntNet paper did not investigate the interaction between ant-based routing and congestion control protocols, such as TCP. TB is also simpler than the AntNet algorithm.

Another ant-like algorithm was developed by Subramanian *et al.* in [20]. This algorithm sends one-way messages from the destination to the source. Like AntNet, this paper did not investigate the interaction with congestion control protocols. This algorithm also requires a lengthy learning period to discover the network topology. Schoonderwoerd *et al.* [18] developed an ant-based algorithm for call setup in a circuit-switched network.

In addition to ant-based algorithms, several other types of adaptive algorithms have been developed. Basu *et al.* [2] propose a class of routing algorithms based on potentials. They also develop a specific traffic-aware routing protocol using potentials. Traffic awareness is achieved by defining the potential of a link for a given source-destination pair as a function of the queue length of the link and the distance to the destination. Another approach to adaptive routing is presented by Wang and Crowcroft [22]. In this paper, a protocol that uses *emergency exits* to alleviate congestion on shortest paths is proposed. These emergency exits are precomputed and are only used when queue lengths are detected to be above a certain threshold. In a somewhat different approach, Shaikh *et al.* [19] present a protocol that distinguishes between long-lived and short-lived IP flows. Adaptive routing is performed only on long-lived flows, which are detected at the routers by keeping track of the number of bytes for the flow. Once a flow is determined to be long-lived, another mechanism such as MPLS is is used to create a dynamic route. Chen *et al.* [6] also present a dynamic routing protocol that only applies adaptive techniques to some flows in the network. Their approach is to focus on destination networks that receive a lot of traffic. These "hot" destinations initiate paths to themselves using dynamic link metrics. Packets for all other destinations use the normal link state protocol. The protocol of [6] is called *Scout*, and it uses scout packets sent by the popular destinations to periodically update the routing tables in other nodes. The *Scout* protocol and the behavior of their scout packets are very different from ours. This protocol is based on the periodic flooding of scout packets to discover link metrics and to perform the path updates. Our protocol does not use flooding, instead our scout packets are sent only along the paths of ac-

tive flows, and they occasionally (based on our probability tables) deviate from the best paths to try to discover better paths.

If the traffic inputs are known *a priori* or the statistics of the inputs change slowly with time, then optimal performance in terms of minimum average delay can be achieved. The seminal work here is the minimum delay routing algorithm by Gallager [9]. This work developed an iterative algorithm that converges to the optimal solution for stationary input traffic statistics. The algorithm of [9] is extended by Bertsekas *et al.* [3]. Vutukury and Garcia-Luna-Aceves [21] present an approximation to minimum delay routing is presented. If the traffic matrix that gives the point to point volume is available, Fortz and Thorup [8] propose an algorithm that precomputes OSPF weights to alleviate congestion and to achieve good utilization. TB differs from these algorithms in that it does not depend on any prior knowledge of traffic patterns.

3. The TRAIL BLAZER algorithm

In this section we present an informal description of the TB protocol. In the next section we provide a detailed description. The TB routing protocol is an extension of a link-state protocol, such as OSPF or IS-IS. TB enhances the link-state protocol by adapting the routing decisions to prefer links with smallest latency to the destination. In this way, the traffic is directed toward less congested paths, which do not suffer from long queuing delays.

In each router TB maintains a link probability table, which contains m rows, one for each destination subnet.[2] TB routes data packets according to the probabilities of the corresponding entries in the link probability table. However, TB does not consider links with low probability for the routing decision, because very low probability indicates a path with significantly higher latency.

TB operates in either one-way or two-way mode. One-way TB sends scout packets from the destination of active flows towards their respective sources. One-way scout packets accumulate the latency to the destination as they travel towards the source, and use this accumulated latency to update the link probability tables at the intermediate routers. See Section 4.6 for details. One-way scout packets do not require clock synchronization in order to compute the accumulated latency. One-way scout packets that arrive at the source are destroyed. Two-way TB sends scout packets from the source of active flows towards their respective destinations. Two-way scout packets keep the local time stamp of all intermediate routers on the way to the destination. Two-way scout packets that arrive at the destination change direction and then retrace their exact path toward the source. Two-way scout packets compute the latency from the current router to the destination on the return trip by subtracting the previous time stamp kept in the packet from arrival time to the destination. In Section 7 we present a different method for computing the latency of two-way scout packet that does not require clock synchronization. Two-way scout packets update the links probability table on their way back from the destination to the source.

The advantage of one-way scouts is that every scout packet can update the link probability tables of intermediate nodes, even if the packet is eventually lost (e.g. due to a dead end or a loop). Although one-way scouts compute the correct latency to the destination by considering the queuing delay, transmission time and link delay, one-way scout packets are routed using the link probability tables for the reverse direction, that is, from the destination to the source. These tables may be different from the tables from the source to the destination due to asymmetric traffic. The advantage of two-way scouts is that every scout packet is routed using the link probability table and link utilization on the direction to the destination. However, two-way scouts may be lost due to a dead-end or a loop without performing any useful work. Section 6.1 compares the performance of one-way and two-way TB.

TB sends scout packets only for active flows. The amount of scout packets is a fixed portion of the the data traffic in the flow. Note that both end points of a TCP connection send scout packets to each other, even when the TCP connection carries data only in one direction, because TCP sends acknowledgment packets, which trigger the creation of scout packets.

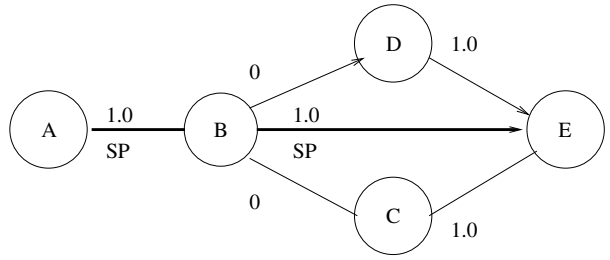
Scout packets contain the list of nodes they visited on their travel, which facilitates detecting loops and backtracking of two-way scout packets from the destination to the source. See Table 1 for details.

There are two kinds of scout packets: best-path scout packets, which travel along existing high probability paths, and exploratory scout packets, which are allowed to make "wrong turns" occasionally and deviate from the high probability paths. A scout packet never selects an outgoing link that leads to a node that appeared earlier in its path (a loop). If there are only such links available, the scout packet self-destructs. Exploratory packets prefers outgoing links with low congestion over links with high congestion.

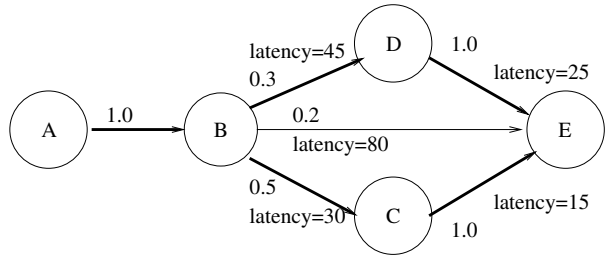
Scout packets update the links probability table using the algorithm described in Section 4.7. Scout packets eventually increase the probability of links that have low latency to the destination, and decrease the probability of links that have high latency.

Fig. 1 illustrates the operation of TB. Fig. 1(a) shows the initial state of the link probability tables for destination node *E*. Since the shortest path from *A* to *E* is $A \rightarrow B \rightarrow E$,

2 In the following discussion we will use the term "destination" to refer to a destination subnet.



(a) initial link probabilites



(b) link probabilities after scout measurements

Legend: SP = shortest path ⟶ selected path

Figure 1. TRAIL BLAZER **routing.**

and there is only one shortest path from *A* to *E*, the probabilities of the links $A \rightarrow B$ and $B \rightarrow E$ are all one. The probabilities of the links $B \rightarrow C$ and $B \rightarrow D$ are initially zero. Thus, every data packet from *A* to *E* is initially sent over $A \rightarrow B$ and $B \rightarrow E$. Eventually, some best-path and exploratory scout packets will be sent from *A* to *E*. These scout packets measure the latency shown in Fig. 1(b), and update the link probability tables accordingly. Eventually, the probability of the link $B \rightarrow C$ will be highest, since the latency of $B \rightarrow C \rightarrow E$ is lowest, and the probability of the link $B \rightarrow E$ will be lowest, since its latency is the highest. Fig. 1(b) shows the final probabilities of the links from *B* to *E*. TB routes data packets only on the links with the highest probability, as explained in Section 4.4. Thus, it will use only the links $B \rightarrow C$ and $B \rightarrow D$.

TB handles link failures by updating the link probability table with new shortest-path information. In this way, the underlying link-state protocol quickly notifies TB in all nodes about topology changes, and TB resets the link probability tables to reflect the new topology. Later scout packets will modify the link probability tables to reflect the new network latencies.

4. TRAIL BLAZER details

4.1. Data structures

TB maintains two data structures in each router: a probability table *pt* and an average transmission delay table avg. *Pt* has m rows, one for each destination or a destination subnet. Each row has k entries, one for each outgoing link of the router. The entry $pt[d, i]$ is the probability of sending a packet to destination d on the outgoing link i. The following relation holds for every row d in pt:

$$\forall 1 \leq d \leq m \ \sum_{i=1}^{k} pt[d, i] = 1 \qquad (1)$$

The table avg holds the average transmission delay to destinations from the current node. *Avg* has m entries, one for each destination or destination subnet. The entry $avg(d)$ is the average transmission delay from the current node to the destination d, which is computed from the last M scout packets that arrived from d. We selected $M = 10$, which is a compromise between memory consumption and sensitivity to recent measurements.

Table 1 shows the structure of a scout packet.

4.2. Parameters

Table 4.2 shows a summary of the parameters of the TB algorithm; we shall provide more details when we first mention the use of each parameter. The number of tunable parameters reflect the experimental nature of our work. We believe that with more experimental testing many of these parameters can be made into constants.

4.3. Initializing the links probability table

The initial value of *pt* is computed from the shortest path information collected by the underlying link-state protocol. Let *SP(d)* denote the set of outgoing links that belong to one or more shortest paths from the current node to destination d. The initial value of *pt* is:

$$pt[d, i] = \begin{cases} \frac{1}{|SP(d)|} & i \in SP(d) \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

src	source address of the packet
dest	destination address of packet
is_two_way	TRUE: a two-way scout FALSE: a one-way scout
is_exploratory	TRUE: an exploratory scout; FALSE: a best-path scout
td	one-way scout packets: latency from current node to source
arrival_time	two-way scout packets only: arrival time to the destination
nhops	hops count from the source
hop	an array of *nhops+1* entries, one for each hop from the source
hop[i].address	address of current node
hop[i].timestamp	two-way scout only: local time on forward direction

Table 1. Structure of a Scout packet

name	range	value	description
α_1	$[0,1]$	0.5	low threshold of the probability of outgoing links in 1st pass
α_2	$[0,1]$	0.8	low threshold of the probability of outgoing links in 2nd pass
β	$[0,1]$	0.5	combining factor of historical probability and new shortest-path information
γ	$[0,1]$	0.25	probability of sending best-path scout packets
δ	$[0,1]$	0.2	learning rate
η	$[0,1]$	0.5	the probability of an exploratory scout selecting a random outgoing link
θ	$[0,1]$	0.5	the weight of congestion correction term in exploratory scout packets
M		10	keep the average of the last M latency measurements for every destination
N		50	send a scout packet every so many regular packets
MAX_HOPS		32	maximal path length of a scout packet in hops

Table 2. TB **parameters. The "value" column is the values in the simulations.**

Whenever the underlying link-state protocol discovers a topology change, such as after a link failure or addition of new links, TB combines the new shortest path information with the historical values accumulated in *pt*. The new value of *pt* is:

$$pt[d,i] \;=\; \begin{cases} \beta \times pt[d,i] + \frac{1-\beta}{|SP(d)|} & i \in SP(d) \\ \beta \times pt[d,i] & \text{otherwise} \end{cases} \qquad (3)$$

The parameter β controls the amount of accumulated information that is kept after a topology change. $0 \le \beta \le 1$. $\beta = 0$ ignores all historical information, whereas $\beta = 1$ ignores the new shortest path information (not a wise idea).

4.4. Routing data packets

In order to avoid sending packets on links with low (but not zero) probability, TB selects a set $H(d)$ of outgoing links for every destination d, and sends the data packet on one of the links in $H(d)$. The set $H(d)$ is chosen so that minor changes in *pt* do not change $H(d)$.

To compute $H(d)$, we first compute $high(d)$, which is the highest probability of any link that leads to d.

$$high(d) \;=\; \max_{\forall 1 \le i \le k} pt[d,i] \qquad (4)$$

Next we compute $low(d)$, which is the lowest probability of any link that is larger than α_1 times $high(d)$.

$$low(d) \;=\; \min pt[d,i];\, pt[d,i] \ge \alpha_1 \times high(d) \qquad (5)$$

The set $H(d)$ is

$$H(d) \;=\; \{i | pt[d,i] \ge \min(\alpha_1 \times high(d), \alpha_2 \times low(d))\} \qquad (6)$$

The constants α_1 and α_2 implement a "soft cutoff" that includes links with probabilities close to $low(d)$. For example, if link probabilities are $\{0.5, 0.3, 0.2\}$, $\alpha_1 = 0.5$, and $\alpha_2 = 0.8$, then $low(d)$ is 0.3, and $H(d)$ is $\{0.5, 0.3\}$. However, if the link probabilities are $\{0.5, 0.26, 0.24\}$, $H(d)$ will include all links.

The probability of sending a data packet on an outgoing link i, $i \in H(d)$, is

$$prob(d,i) \;=\; \frac{pt[d,i]}{\sum_{j \in H(d)} pt[d,j]} \qquad (7)$$

4.5. Scout packet types

TB generates two types of scout packets: best-path packets and exploratory packets. Best-path scout packets and exploratory scout packets differ only by the way they are routed. Best-path scout packets are routed similarly to data packets according to the probabilities in *pt*. These packets never try new paths, and they are used to measure the latency of the current best path. Exploratory scout packets try to discover better alternate paths (detours), and their path is influenced by current congestion measurements. When a scout packet of any type discovers a loop, or if its path is longer then *MAX_HOPS*, it self-destructs. TB generates best-path scout packets with a probability of γ, and generates exploratory scout packets with a probability of $1 - \gamma$. It is important to keep γ reasonably high in order to measure the current paths used by data packets frequently, so that we discover congested data paths quickly and shift the traffic to less congested paths.

All scout packets are initialized at the source by setting the *src* and *dest* fields to the corresponding addresses of the source and destination nodes, and setting *td = 0*, *nhops = 0*, and *hop[0].address = src*.

4.6. Handling scout packets in intermediate nodes

One-way scout packets carry with them the latency *td* from the current node back to the source of the scout packet. One-way scout packets update their accumulated latency *td* in every intermediate node by $td \;=\; td + t(i)$, where $t(i)$ is the current latency of outgoing link i, which is the sum of the queuing delay and link delay. It does not include the transfer time, which is a function of the packet size and link bandwidth. Link i is the link on which the packet arrived from the previous node. $t(i)$ is computed by:

$$t(i) = \frac{q_{size} + cp_{size}}{l_{bandwidth}} + l_{delay} \quad (8)$$

where q_{size} is current output queue size of link i in bytes, cp_{size} is the remaining size of the currently transmitted packet on link i, $l_{bandwidth}$ is link i bandwidth in bytes/s, and l_{delay} is link i transmission delay.

Every intermediate node increases the hops count *nhops*, and adds an entry to the *hops* array. The entry contains the node address (one-way scouts) or both the node address and the local time stamp (two-way scouts).

In every intermediate node (not the destination), TB computes the set S of outgoing links that are *not* connected to nodes that appeared in the packet's path. If S is empty, the scout packet is destroyed.

TB routes a best-path packet to destination d on link i, only if $i \in S \wedge H(d)$, using the probability $prob\_best(d,i)$.

$$prob\_best(d,i) = \frac{pt[d,i]}{\sum_{i \in S} pt[d,i]} \quad (9)$$

The packet is destroyed if $S \wedge H(d)$ is empty.

Unlike best-path scout packets, which use only the information in *pt* and S for routing decisions, exploratory packets also consider the current congestion of outgoing links. The reason is that exploratory packet should notice congestion conditions as soon as possible in order to explore alternate paths. We assume that the traffic on the links is symmetrical for one-way scout packets (same amount of traffic flows through the link in both directions), which is often incorrect. If the traffic is symmetric, a high value of $t(i)$ on one direction of the link is a likely indication of a congestion on the other direction as well. Because two-way scout packets measure link latency in the same direction as data packets, they handle asymmetric traffic correctly.

TB computes a congestion correction term $c(i)$ for every outgoing link i of the current node. The congestion correction term is a value in the range $[0,1]$ that indicates the congestion of link i relative to other links.

$$c(i) = \frac{t(i)}{\sum_{j=1}^{k} t(j)} \quad (10)$$

where $t(i)$ is the latency of outgoing link i, and the node has k outgoing links. Note that $\sum_{i=1}^{k} c(i) = 1$.

TB selects to send the exploratory packet on a random outgoing link with probability η. The probability of selecting a random outgoing link i, $i \in S$, is

$$prob\_exp\_rand(d,i) = \frac{\frac{1}{|S|} + \theta \times (1 - c(i))}{1 + \theta \times (k-1)} \quad (11)$$

θ is the weight of the congestion correction term. A large value of θ gives preference to the outgoing link with the smallest current congestion factor. $\theta = 0$ forces a uniform selection (probability $1/|S|$) of outgoing links without any regard to their congestion. $\theta = 0.5$ is a compromise between uniform selection and current congestion.

TB selects to send the exploratory packet using best path information with probability $1 - \eta$. The probability of selecting outgoing link i, $i \in S$, is

$$prob\_exp\_best(d,i) = \frac{prob\_best(d,i) + \theta \times (1 - c(i))}{1 + \theta \times (k-1)} \quad (12)$$

Again, θ is the weight of the congestion correction term. $Prob\_best(d,i)$ is the probability of sending the best-path scout packet on the outgoing link i. TB selects only the links with a high *prob_best*(d,i) values that are also included in S using the link selection algorithm of Section 4.4. If none of links with high *prob_best*(d,i) values are included in S, then TB selects a random link using *prob_exp_rand* above. We set $\eta = 0.5$ to have an equal chance of selecting a random outgoing links and a best-path link.

4.7. Updating the links probability table

One-way scout packets uses the accumulated latency *td* to update the *pt* table on their way from the source to the destination. Two-way scout packets compute the *td* on the return path from the destination to the source by $td = arrival\_time - hops[k].timestamp$, where k is the index of the entry of the current node, and $arrival\_time$ is the arrival time to the destination. We assume global clock synchronization here. However, Section 7 outlines a different method for computing the latency of two-way scout packets that does not require clock synchronization. For both one-way and two-way scout packets, the node updates the average latency table $avg(d)$.

When a scout packet updates *pt*, it refer to the link i, which is the link through which the packet arrived to the node. This link points to the destination of corresponding data packets. Note that one-way scout packets are sent from the destination of the flow, and two-way scout packets are sent from the source of the flow, but they update *pt* when they return from the destination.

All scout packets update *pt* using their *td* and current $avg(d)$ values as follows.

$$f(td) = \max(\min(\frac{avg(d)}{td}, 10), 0.1) \quad (13)$$

$$\Delta p = \delta \times f(td) \quad (14)$$

$$pt[d,i] = \frac{pt[d,i] + \Delta p}{1 + \Delta p} \quad (15)$$

$$pt[d,j]_{j \neq i} = \frac{pt[d,j]}{1 + \Delta p} \quad (16)$$

The average latency $avg(d)$ is used to scale the positive reinforcement value of the scout packet. A larger value of $f(td)$ indicates a better (shorter) path. $f(td)$ is limited to

the range $[0.1, 10]$ to prevent wide fluctuations in Δp, which is the the reinforcement value of $pt[d, j]$.

δ is the learning rate of the algorithm. It should be set high enough so that a scout that discovered a short path to the destination will have some influence on *pt*. However, it should not be set too high to avoid oscillations. We used the δ value from [5]. All entries in *pt* of the same destination d are scaled by $1 + \Delta p$ to ensure that their sum remains 1.

5. TB properties

5.1. Low overhead

TB has low communication overhead, since scout packets constitute a small fixed percentage of data packets. We simulated scout packet frequency of 2% (one scout packet every 50 data packets), which was sufficient to reduce packet drops significantly. Moreover, scout packets are short, since they comprise a fixed header and a 4 or 8 byte entry for every node on the scout's path, for one-way and two-way scout packets, respectively. See Table 1 for details on scout packet structure. Another advantage is that TB never broadcasts state information to every node.

5.2. Constructive interference

TB benefits from constructive interference between the scouts sent to the same destination from different sources. A congested link along the way to a popular destination will be probed by multiple scouts from every source. Once the first exploratory scout packet finds a path around the congestion, other data packets from all sources to the same destination will follow it.

5.3. Reduced path oscillations

TB reduces path oscillations by a combination of several mechanisms. First, TB does not broadcast new link costs or any other global state. It rather measures new path latencies gradually by sending scout packets on active flows. The process of measuring path latencies is inherently asynchronous. Second, TB updates the link probability table with a combination of old and new information. The new information is further scaled by a learning rate, as explained in Section 4.7. This mechanism prevents wide swings based only on the most recent measurement. Third, TB routes packets on links with high probability using a "soft cutoff", so that small changes in the probability of any link will normally not change the set of links that are used to route packets. See Section 4.4 for details.

5.4. Asymmetric routing

Packets sent on the forward direction of a bidirectional flow may travel on a different path than the backward direction of the same flow. The reason is that the link probability tables from a given source to a given destination may be different from the probability tables from the destination back to the source. This may occur when one of the directions is congested. A practical implication of this phenomenon is that TCP acknowledgments may travel on a different path than the data.

5.5. Some packets may enter a loop

TB may route data packets in a loop with a very small probability, since the link probability tables are updated asynchronously by different scout packets. However, TB sends packets only on links with relatively high probability. A link with non-zero probability implies that either this link is (or was) on a shortest path, or that a scout packet arrived from the destination to the current node through the corresponding link. In other words, a non-zero probability implies that there is (or was) a path from the current node to the destination through that link.

The links with the highest probability are either the links that are currently on the shortest path to the destination or the links through which many scout packets that measured low latency to the destination have passed. Both cases indicate that sending a packet on a link with a high probability will bring the data packet closer to the destination. However, a few data packets will have a very long path, as described in Section 6.1. A negligible amount of data packets were lost due to an excessive hop count (more than 32 hops in our simulations). The average, median, and maximal data packet loss due to an excessive hop count across all simulations are 0.0003%, 0, and 0.011%, respectively.

6. Simulations

We compared the performance of TRAIL BLAZER and OSPF using the *ns-2* network simulator [14]. We used the "LS" (link-state) protocol that is provided with *ns-2* to simulate link-state shortest-path routing, which is essentially OSPF. In the case of hybrid routing (TB for UDP traffic and link-state shortest-path routing for TCP traffic), we used a single routing protocol that selected TB for UDP packets, and performed shortest-path routing for TCP packets. We did not simulate link failures.

We simulated three topologies: ISP, BRITE-25, and BRITE-50. The ISP topology is meant to be representative of the network topology of a typical US Internet service provider. This topology has been used by Apostolopoulos *et al.* [1]. The ISP topology has two kinds of

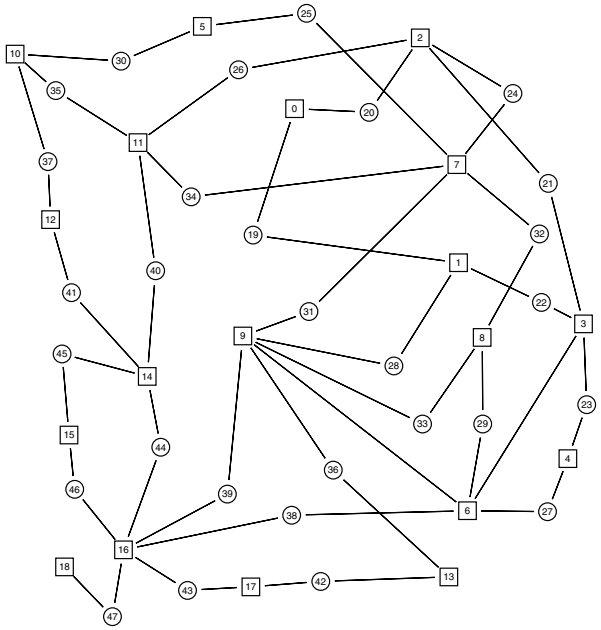


Figure 2. The ISP topology. Squares represent source and destination routers, and circles represent transit networks.

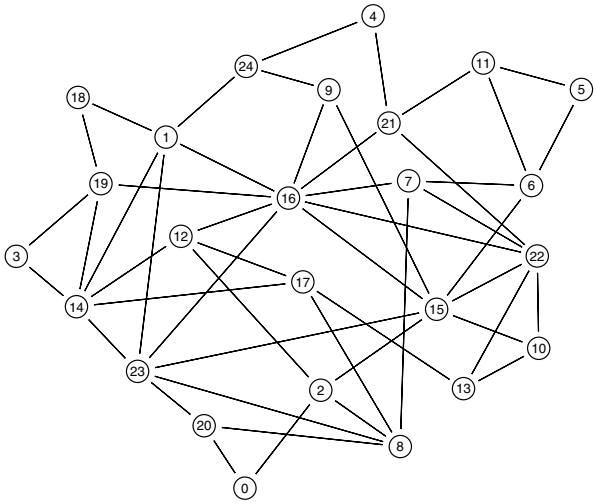


Figure 3. The BRITE-25 topology.

nodes: "router" nodes, and "transit network" nodes. Only "router" nodes serve as sources and destinations of traffic. This topology has a total of 48 nodes, out of which 19 (about 40%) are "router" nodes. Fig. 2 shows the ISP topology. The BRITE-25 and BRITE-50 topologies were generated by the BRITE topology generator [16] using the Waxman [23] model and the default settings. BRITE-25 has 25 nodes, and BRITE-50 has 50 nodes. Fig. 3 shows the BRITE-25 topology. The bandwidth and delay of all links in all topologies are 1Mbps and 5ms, respectively. We simulated drop-tail queues containing up to 50 packets on all links.

We simulated an active network with a shifting traffic pattern. All simulations generated a traffic pattern comprising four consecutive phases of 15 seconds each. In each phase we randomly divided the network nodes into clients and servers. The clients group contains $\lfloor \frac{n}{2} \rfloor$ nodes, and the servers group contains $\lceil \frac{n}{2} \rceil$ nodes, where n is the total number of nodes. Then we selected $\lfloor \frac{n}{2} \rfloor$ pairs of nodes, each comprising one client and one server. The traffic was sent from the client to the corresponding server in each pair. Each client sends traffic to one server, and each server receives traffic from at most one client. The first phase starts one second after the start of the simulation to enable the "LS" protocol to discover the network topology. We used a packet size of 500 bytes for all traffic (UDP and TCP).

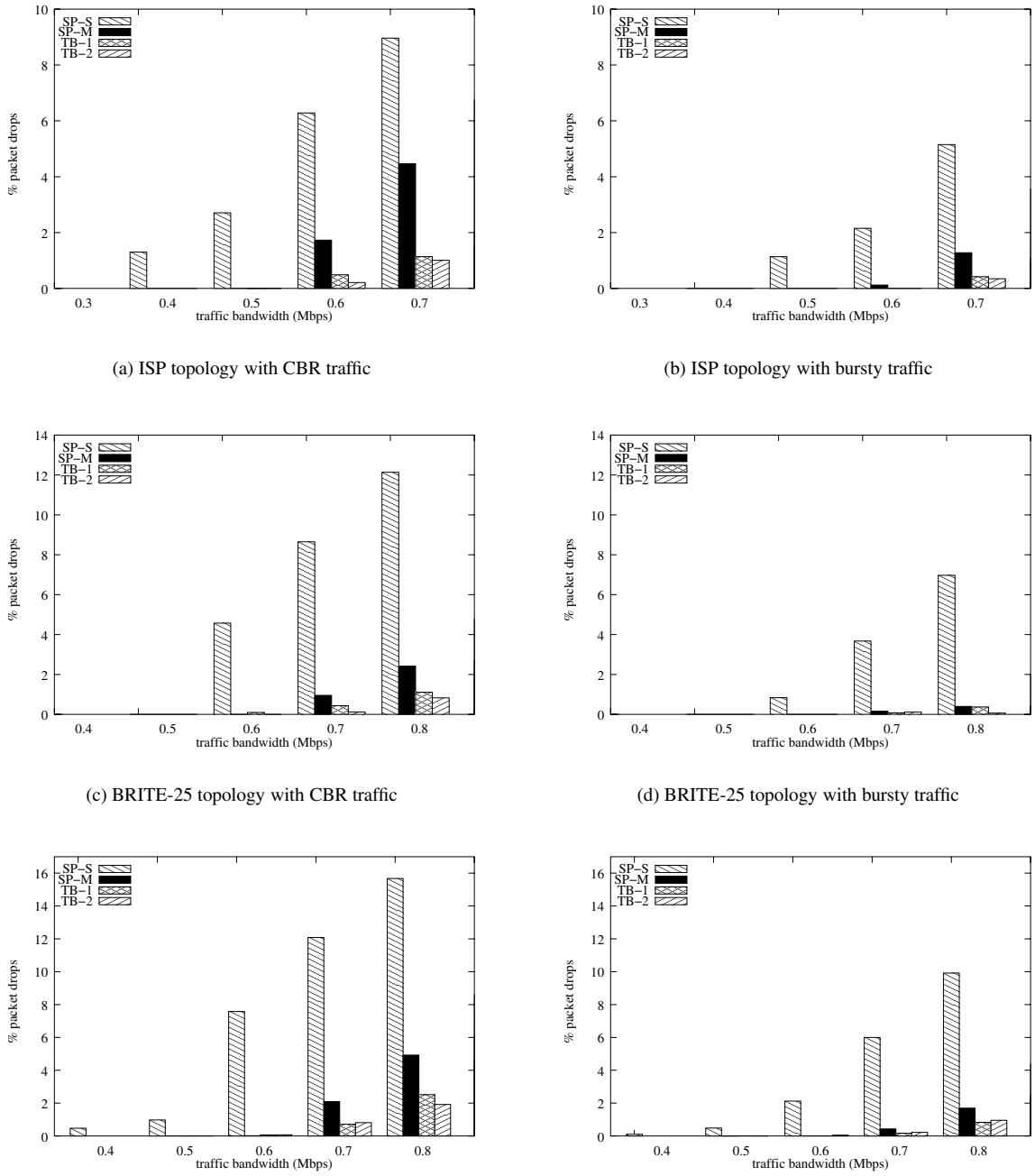
Note that the clients and servers changed in each phase, as well as the association between clients and servers. In particular, we did not enforce any spatial locality on the communication in each phase. The phase changes are designed to stress TB, because link latencies change in each phase. TB has to learn the new latencies and forget old latencies without causing traffic oscillations.

6.1. UDP traffic over TRAIL BLAZER

We measured the packet drop rate of CBR and bursty UDP traffic in the three topologies and four routing protocols. The routing protocols are: link-state shortest-path with single-path routing (SP-S), link-state shortest-path with multi-path routing (SP-M), TB with one-way scouts (TB-1), and TB with two-way scouts (TB-2).

Fig 4 shows the packet drop rate due to network congestion for CBR and bursty traffic of varying bandwidth. CBR traffic sends one stream of UDP packets with the specified bandwidth from every client to the corresponding server. Bursty traffic sends one stream of UDP packets from every client to the corresponding server using the *ns-2* Pareto on/off traffic generator with "shape" value of 1.4. Data is sent with the specified bandwidth during the "on" period, and no data is sent during the "off" period. The average duration of the "on" and "off" periods are 800ms. and 200ms., respectively.

There are several interesting observations in Fig 4. First, the BRITE-25 and BRITE-50 topologies can withstand higher bandwidth without congestion (as indicated by lower packet drops for the same traffic bandwidth) than the ISP topology. This is expected, since the typical node in the BRITE-25 and BRITE-50 topologies has a larger degree of connectivity than the ISP topology. In other words, there are more alternate paths between nodes in the BRITE-25



(a) ISP topology with CBR traffic

(b) ISP topology with bursty traffic

(c) BRITE-25 topology with CBR traffic

(d) BRITE-25 topology with bursty traffic

(e) BRITE-50 topology with CBR traffic

(f) BRITE-50 topology with bursty traffic

Figure 4. UDP packet drop rate for CBR and bursty traffic.

and BRITE-50 topologies than in the ISP topology. Second, the multi-path shortest-path routing (SP-M) has lower packet drops rate than the single-path shortest-path routing (SP-S). This is also expected, since the traffic can use alternate paths, which reduces the congestion. Third, all topologies can withstand a higher bandwidth of bursty traffic. This too is expected, since the average load of the bursty traffic is lower than the CBR traffic of the same bandwidth. Fourth, TB with two-way scout packets (TB-2) achieves a lower packet drop rate than TB with one-way scout packets (TB-1) in almost all topologies, traffic patterns and network loads, This is the reason why we will consider only TB with two-way scouts for the rest of this section.

Fig. 4 also shows that TB-2 reduces the UDP packet

topology	bandwidth (Mbps)	protocol	avg. latency (ms.)	path length avg.	path length 99 pct.	max out of order	% window drop rate reorder buffer size 25	50	75
ISP	0.6	SP-M	72.2	5.015	8	30	3.066	0	0
		TB-2	70.2	5.243	10	51	1.183	0	0
	0.7	SP-M	83.3	5.137	8	35	4.506	0	0
		TB-2	81.3	5.495	12	106	3.927	0.378	0.007
BRITE-25	0.7	SP-M	54.3	2.361	4	35	4.316	0	0
		TB-2	45.9	2.777	7	195	2.324	0.254	0.031
	0.8	SP-M	57.9	2.389	4	78	4.250	0.285	0.239
		TB-2	60.9	2.876	7	166	4.892	0.637	0.070
BRITE-50	0.7	SP-M	77.4	2.875	4	68	5.355	0.591	0
		TB-2	76.8	3.443	9	292	4.439	0.756	0.106
	0.8	SP-M	89.0	2.933	4	78	5.942	0.716	0.427
		TB-2	112.4	3.619	9	210	10.101	2.703	0.544

Table 3. UDP CBR traffic characteristics of representative configurations. 99 pct. is the 99 percentile path length. % window drop rate is the percentage of packets dropped due to insufficient reordering window size.

drop rate by a factor of 3.4–58.3 compared with single-path shortest-path routing (SP-S), and by a factor of 1.4–8.2 compared with multi-path shortest-path routing (SP-M). There was even one case where TB-2 was able to eliminate all packet drops (ISP with 0.5Mbps bursty traffic). There are only four cases where TB-2 has more packet drops than SP-M. It is difficult to see these cases due to the scale of Fig. 4. In all cases the contention is low (0.4 and 0.5Mbps traffic) and the overall packet drop is also low (less than 0.077%).

Table 3 shows the characteristics of UDP CBR traffic for the highest bandwidth combinations of Fig 4. The results for bursty traffic are similar. Table 3 compares the multi-path shortest-path routing (SP-M) with two-way TB, since one-way shortest-path routing does not suffer from out-of-order delivery. Table 3 shows the average packet latency, average packet path length, and the 99th percentile of packet path length. The last four columns show the effect of a limited reordering buffer on packet delivery. We simulated a reordering buffer that contains the last n slots up to the packet with the highest received sequence number. We drop all packets with a sequence number that is lower than the lowest slot of the reordering buffer. The column "% window drop rate" shows the resulting packet drop rate for several sizes of the reordering buffer. The table also shows the maximal out-of-order delivery range, which is the maximal range of packet reordering in the network. It is also the minimal size of a reordering buffer that can prevent all packet drops due to insufficient history in the reordering buffer.

There are several interesting observations in Table 3. First, the average packet latency of SP-M and TB-2 is similar. It shows that TB-2 both reduces packet drops and maintains a similar average packet latency. Second, it shows that TB path length is typically longer than SP-M. This is expected, since TB sends packets on detours in order to avoid congestion. However, TB sends some packets on an excessively long path, as indicated by the 99th percentile of the path length. This is the result of using probability tables, which may send some packets in a loop. In all cases there is a negligible amount of packets that were dropped due to an excessive hop count (more than 32 hops in our simulations). See Section 5.5. This result indicates that most packets do not travel in endless loops, even if the TB is not loop-free. Third, both SP-M and TB-2 suffer from excessive packet reordering. SP-M suffers from load imbalance on the shortest paths, whereas TB-2 suffers from occasional packets that travel through an especially long path. A reordering buffer of 50 slots is sufficient in most cases to ensure in-order delivery of most UDP packets with SP-M, and a reordering buffer of 75 slots is sufficient for TB-2. However, this reordering buffer is not enough for both routing algorithms on the BRITE-50 topology with 0.8Mbps CBR traffic. Based on our simulations, we believe that the reordering buffer needed to ensure in-order packet delivery in much larger networks may be larger only by a constant than the size we used.

6.2. TCP traffic over TRAIL BLAZER

Since the two-way TB (TB-2) has better performance than the one-way TB (TB-1), we use TB-2 for the rest of the experiments and refer to it as simply TB. The previous section showed that TB causes packet reordering in many cases, which may interfere with the operation of TCP. The reason is that TCP receivers generate duplicate acknowledgment packets when they receive out-of-order data packets, which in turn causes TCP servers to initiate congestion avoidance. The result is a reduction in the data transfer rate.

We evaluated the effects of packet reordering caused by TB on TCP bandwidth by sending TCP traffic using the same traffic pattern described in the previous section. The only change is that clients send FTP traffic greedily to the corresponding servers instead of UDP traffic. Again, we simulate four phases of 15 seconds each, and we selected a random group of clients and servers in each phase. We compare the average FTP bandwidth of three routing protocols: single-path shortest-path routing (SP-S), multi-path

topology	TB	SP-S	SP-M	% improv.	
				SP-S	SP-M
ISP	93.32	97.22	90.48	-4.0	3.1
BRITE-25	101.03	102.21	104.25	-1.2	-3.1
BRITE-50	93.51	93.08	94.87	0.5	-1.4

Table 4. Average TCP bandwidths in KB/s. The last columns show the percentage improvement of TB relative to SP-S and SP-M.

shortest-path routing (SP-M), and two-way TB (TB). We report the overall bandwidth, which is the total amount of data received by all servers divided by the time it takes to receive all the data.

Table 4 shows the average TCP bandwidth achieved for the three topologies and the three routing algorithms. Note that the bandwidth is presented in KB per second. This experiment indicates that the reordering caused by TB has a small effect on TCP performance. The average bandwidth of TB ranged from an increase of 3.13% to a decrease of 4.01% relative to SP-S and SP-M.

6.3. Combined TCP and UDP traffic

In this section we evaluate the performance of hybrid protocols that uses link-state shortest-path routing for TCP and TB for UDP traffic. We evaluated two hybrid protocols: TB+SP-S, which is the combination of a two-way TB and a single-path shortest-path routing, and TB+SP-M, which is the combination of a two-way TB and a multi-path shortest-path routing. We compared TCP bandwidth and UDP packet drop rate of the hybrid protocols to SP-S and SP-M that route all traffic. The experimental setup is the similar to the previous sections. The difference is that in each phase a client sends both UDP and TCP data to the corresponding server. UDP traffic is sent at a constant bit rate ranging from 0.3Mbps to 0.5Mbps, and TCP traffic is a greedy FTP transfer.

The results of the experiments for single- and multi-path shortest path are shown in Table 5 and Table 6, respectively. In almost all cases the hybrid protocols improved the average TCP bandwidth and the UDP packet drop rate. In both the single-path and multi-path cases, the best improvement in TCP bandwidth is for the BRITE-25 topology with UDP traffic at a rate of 0.5Mbps. For single-path the improvement is 35.3% and for multi-path it is 53.9%. The best improvement in packet drop rate (52%) in the single-path case is for the BRITE-50 topology with UDP traffic of 0.4Mbps, and for the multi-path case the best improvement in packet drop rate is 69.6% for the ISP topology with UDP traffic of 0.5Mbps. The average improvement in TCP bandwidth over all topologies and UDP traffic rates for single-path and

topology	UDP bw	avg. TCP bandwidth / UDP drop %		% improv. TCP bw/ UDP drop
		TB+SP-S	SP-S	
ISP	0.3	64.47 / 0.49%	65.88 / 0.74%	-2.1 / 33.8
ISP	0.4	54.92 / 1.02%	54.93 / 2.08%	0 / 51
ISP	0.5	45.45 / 2.60%	44.45 / 3.61%	2.2 / 28
BRITE-25	0.3	76.18 / 0.24%	66.11 / 0.47%	16.1 / 48.9
BRITE-25	0.4	67.95 / 0.74%	54.25 / 1.0%	25.3 / 26
BRITE-25	0.5	66.04 / 0.92%	42.74 / 1.18%	35.3 / 22
BRITE-50	0.3	62.44 / 0.49%	59.2 / 0.80%	5.5 / 38.8
BRITE-50	0.4	53.73 / 0.86%	47.81 / 1.79%	12.4 / 52
BRITE-50	0.5	49.44 / 1.88%	37.48 / 2.84%	31.9 / 33.8

Table 5. Comparison of TCP and UDP performance with single-path routing.

topology	UDP bw	avg. TCP bandwidth / UDP drop %		% improv. TCP bw/ UDP drop
		TB+SP-M	SP-M	
ISP	0.3	59.68 / 0.24%	59.52 / 0.32%	0.3 / 25
ISP	0.4	52.87 / 0.83%	49.20 / 0.91%	7.5 / 8.8
ISP	0.5	44.81 / 0.86%	39.79 / 2.83%	12.6 / 69.6
BRITE-25	0.3	79.25 / 0.07%	68.55 / 0.15%	15.6 / 53.3
BRITE-25	0.4	73.39 / 0.19%	57.41 / 0.32%	27.8 / 40.6
BRITE-25	0.5	71.44 / 0.45%	46.41 / 0.63%	53.9 / 28.6
BRITE-50	0.3	66.27 / 0.25%	61.66 / 0.27%	7.5 / 7.4
BRITE-50	0.4	56.35 / 0.46%	50.88 / 0.62%	10.8 / 25.8
BRITE-50	0.5	52.21 / 1.37%	40.58 / 1.25%	28.7 / -9.6

Table 6. Comparison of TCP and UDP performance with multi-path routing.

multi-path routing is 14% and 18.3%, respectively. The average improvement in UDP packet drop rate over all topologies and UDP traffic rates for single-path and multi-path routing is 37.1% and 27.7%, respectively. However, there are two instances where the hybrid protocols performed worse than the shortest-path protocols. The ISP topology with UDP rate of 0.3Mbps and single-path shortest-path shows that SP-S had a TCP bandwidth which was 2.1% better than TB+SP-S, but in this instance the UDP drop rate for TB+SP-S was 33.8% better. The other instance is the BRITE-50 topology with UDP traffic at 0.5Mbps and multi-path shortest-path, in which TB+SP-M had a 9.6% higher UDP packet drop rate. However, in this case the TCP bandwidth of TB+SP-M was 28.7% higher than SP-M.

7. Summary and future work

This paper presents the TRAIL BLAZER (TB) routing protocol, which augments shortest-path routing by sending occasional scout packets without a global sense of network topology. TB is inspired by the way ants forage for food. To evaluate the performance TB, we performed extensive simulations with the *ns-2* network simulator. The experiments indicate that TB significantly improves packet drop rate for

UDP with both constant bit rate and bursty traffic. Because TB uses alternate routes to avoid congestion, some packet reordering occurs. However, our simulations indicate that a reordering buffer of size 75 is enough to ensure in-order delivery in most cases. We also showed that the reordering has a minor effect on TCP performance. We propose a hybrid algorithm that uses TB for routing UDP packets and shortest path routing for TCP packets. Our simulations indicate that the hybrid protocol reduces the interference between UDP and TCP traffic and results in improved performance for both. The hybrid protocol could be the best way to introduce a new routing protocol to existing networks.

Although TB requires more processing to route data packets than current shortest-path routing, we believe that hardware acceleration could make TB processing time similar to current protocols.

We plan to investigate three issues in the near future. First, we plan to implement a different method for computing the latency of two-way scout packets, which does not require clock synchronization. This method will keep the latency of the current outgoing link in the packet instead of the current time stamp. This latency is the same as the $t(i)$ of one-way scout packet (see Section 4.6). Two-way scout packets will compute the latency to the destination in the return path by $td = td + hops[k].link\_latency$, where k is the index of the entry of the current node. Second, we plan to investigate the parameter space of TB thoroughly to find optimal combinations of parameters. Third, we plan to investigate the performance of TB with link failures.

Acknowledgement

The authors would like to thank Yigal Bejerano of Bell Labs for many interesting discussions.

References

[1] G. Apostolopoulos, R. Guérin, and S. Kamat. Implementation and performance measurements of QoS routing extensions to OSPF. In *Proceedings of IEEE INFOCOM '99*, pages 680–688, New York, NY, March 1999.

[2] A. Basu, A. Lin, and S. Ramanathan. Routing using potentials: A dynamic traffic-aware routing algorithm. In *Proceedings of ACM SIGCOMM '03*, Karlsruhe, Germany, August 2003.

[3] D. P. Bertsekas, E. M. Gafni, and R. G. Gallager. Second derivative algorithms for minimum delay distributed routing in networks. *IEEE Transactions on Communications*, com-32(8):911–919, August 1984.

[4] E. Bonabeau and G. Théraulaz. Swarm smarts. *Scientific American*, pages 72–79, March 2000.

[5] G. D. Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communication netowrks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[6] J. Chen, P. Druschel, and D. Subramanian. A new approach to routing with dynamic metrics. In *Proceedings of IEEE INFOCOM '99*, pages 661–670, New York, NY, March 1999.

[7] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.

[8] B. Fortz and M. Thorup. Internet traffic engieneering by optimizing OSPF weights. In *Proceedings of IEEE INFOCOM '00*, pages 519–528, Tel Aviv, Israel, March 2000.

[9] R. G. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communications*, com-25(1):73–85, January 1977.

[10] P. P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez *bellicositermes natalensis et cubitermes sp.* La théorie de la stigmergie: Essai d'interprétation du comportment des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[11] G. Huston. *ISP Survival Guide Strategies for Running a Competitive ISP*, chapter Six, pages 205–208. John Wiley and Sons, 1999.

[12] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the International World Wide Web Conference*, pages 252–262. IEEE, May 2002.

[13] A. Khanna and J. Zinky. The revised ARPANET routing metric. In *Proceedings of ACM SIGCOMM '89*, pages 45–56, Austin, TX, September 1989.

[14] S. McCanne and S. Floyd. ns – network simulator. http://www-mash.cs.berkeley.edu/ns/.

[15] J. McQuillan, I. Richer, and E. Rosen. The new routing algorithm for the ARPANET. *IEEE Transcation on Communications*, pages 711–719, May 1980.

[16] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: An approach to universal topology generation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems - MASCOTS'01*, pages 346–356, Cincinnati, Ohio, August 2001.

[17] J. Moy. OSPF version 2. RFC 2328, IETF, April 1998.

[18] R. Schoonderwoerd, O. Holland, and J. Bruten. Ant-like agents for load balancing in telecommunications networks. In *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 209–216, Marina del Rey, CA, February 5-8 1997.

[19] A. Shaikh, J. Rexford, and K. G. Shin. Load-sensitive routing of long-lived ip flows. In *Proceedings of ACM SIGCOMM '99*, pages 215–226, Cambridge, MA, August 1999.

[20] D. Subramanian, P. Druschel, and J. Chen. Ants and reinforcement learning: A case study in routing in dynamic networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 832–839, Nagoya, Japan, August 23-29 1997.

[21] S. Vutukury and J. J. Garcia-Luna-Aceves. A simple approximation to minimum-delay routing. In *Proceedings of ACM SIGCOMM '99*, pages 227–238, Cambridge, MA, August 1999.

[22] Z. Wang and J. Crowcroft. Shortest path first with emergency exits. In *Proceedings of ACM SIGCOMM '90*, pages 166–176, Philadelphia, PA, August 1990.

[23] B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.