Routing Using Potentials: A Dynamic Traffic-Aware Routing Algorithm

Anindya Basu
Bell Laboratories
basu@research.bell-labs.com

Alvin Lin
MIT
alvinl@mit.edu

Sharad Ramanathan
Bell Laboratories
sharadr@physics.bell-labs.com

Abstract

We present a routing paradigm called PB-routing that utilizes steepest gradient search methods to route data packets. More specifically, the PB-routing paradigm assigns scalar potentials to network elements and forwards packets in the direction of maximum positive force. We show that the family of PB-routing schemes are loop free and that the standard shortest path routing algorithms are a special case of the PB-routing paradigm. We then show how to design a potential function that accounts for traffic conditions at a node. The resulting routing algorithm routes around congested areas while preserving the key desirable properties of IP routing mechanisms including hop-by-hop routing, local route computations and statistical multiplexing. Our simulations using the *ns* simulator indicate that the traffic aware routing algorithm shows significant improvements in end-to-end delay and jitter when compared to standard shortest path routing algorithms. The simulations also indicate that our algorithm does not incur too much control overheads and is fairly stable even when traffic conditions are dynamic.

Categories & Subject Descriptors: C.2.2 Routing Protocols.
General Terms: Algorithms.
Keywords: Congestion, Potential, Routing, Steepest Gradient, Traffic Aware.

1. Introduction

Routing mechanisms in the Internet have typically been based on shortest-path routing for best effort traffic. This often causes traffic congestion, especially if bottleneck links on the shortest path severely restrict the effective bandwidth between the source and the destination.

Traditionally, congestion control in the Internet has been provided by end-to-end mechanisms. An example is the TCP congestion control mechanism that works by adjusting the sending rate at the source when it detects congestion at a bottleneck link (for details, see [25]). If multiple traffic streams share the same bottleneck link, each gets only a fraction of the bottleneck link bandwidth even though there may be bandwidth available along alternate paths in the network. Moreover, queueing delays at the bottleneck link can add significantly to end-to-end delays. Finally, varying traffic conditions can make this queueing delay variable, thereby adding to jitter.

SIGCOMM'03, August 25–29, 2003, Karlsruhe, Germany.

One way to address the problem of end-to-end delay and jitter is to use traffic engineering (TE) techniques in conjunction with circuit-based routing. In this case, the routing process assumes that the traffic demands between source-destination pairs are known apriori and computes end-to-end paths (circuits) satisfying the traffic demands. End-to-end circuits are then set up along the computed paths using a resource reservation protocol such as RSVP [13]. Data packets are now source routed along these pre-computed paths.

There are some drawbacks to this kind of circuit-based routing. First, if traffic sources are bursty (which is mostly the case in the Internet — see, for example, [21]), resources may be reserved unnecessarily, thereby negating the benefits of statistical multiplexing. Second, traffic demands between network nodes are hard to estimate apriori. Also, if traffic patterns change, it is possible that a global recomputation is necessary to determine the most optimal routing. Third, when future demands are unpredictable, it is difficult to route current demands such that a future demand has the maximum chance of being routed successfully without requiring the rerouting of existing demands. Indeed, it can be shown that this problem is NP-hard even for the simplest cases [12].

In this paper, we present an alternate methodology for traffic-aware routing that is based on steepest gradient search methods. We call this methodology *potential based routing*, or *PB-routing*. It preserves the hop-by-hop routing philosophy of the Internet, and does not require a priori knowledge of traffic demands between network nodes. At the same time, PB-routing is able to route packets around the congested hot-spots in the network by utilizing alternate routes that may be non-optimal. This reduces end-to-end delays and jitter and increases the bandwidth utilization in the network. Since packets are not source routed, PB-routing can adapt to changes in traffic conditions without requiring any global recomputation of routes. Furthermore, end-to-end resource reservation is not required — hence, the benefits of statistical multiplexing are still available. Note that PB-routing can only provide performance improvements of a statistical nature, and not explicit worst-case bounds on delay and jitter (as opposed to TE techniques). However, our simulations indicate that there is significant improvement in delay and jitter for most traffic streams when PB-routing is used.

The key idea in PB-routing is to define a scalar field on the network, which is used to define a potential on every network element (NE). The routing algorithm at each NE now computes the route to the destination as the direction (i.e., the next hop) in which the potential field decreases fastest (direction of maximum force or steepest gradient). We show that by assigning the NE potentials differently, a whole family of routing algorithms can be designed. For example, the standard shortest path algorithm can be shown to be a special case of PB-routing if the potential at each NE is set to be a linear, monotonically increasing function of the shortest distance from the NE to the destination. The routing algorithm can be made traffic-aware by setting the poten-

tial at each NE to be a weighted sum of the shortest path potential and a metric that represents the traffic potential at the NE. We show how to define such a metric later in the paper.

More intuitively, the PB-routing algorithm views the entire network as a terrain, with many negotiable obstacles created by congestion. The high obstacles represent areas of the network with high congestion (and therefore, high potential). The idea is to find a path from the source to the destination by avoiding the high obstacles as much as possible.

Since PB-routing depends on traffic information at the various NEs, it is necessary that this information be disseminated efficiently without compromising end-to-end packet delays and jitter. In our simulations, we adapt a link-state routing protocol for this purpose. We also describe an optimization that significantly reduces the control overheads of this protocol without sacrificing the performance of the routing algorithm.

The main contributions of this paper are as follows. While steepest gradient search methods have been well-studied, the novel idea in this paper is the design of a potential field for traffic-aware routing that guarantees desirable properties such as loop-free routing. We demonstrate that our design of a potential function, which is a hybrid of traffic metrics and link costs, ensures that packets avoid congested areas but do not traverse the network using random walks. In fact, how far the path of a packet deviates from the standard shortest path can be controlled by a configurable parameter. In our simulations, we have observed significant improvements in end-to-end delay and jitter over a variety of networks and traffic conditions without requiring too much control overheads. We believe that the general framework of PB-routing could be adapted for optimizing various other metrics through careful design of potential functions. This is especially true of overlay networks (see, for example, [2]) where application specific metrics that require optimization could be converted into appropriate potential functions.

The rest of the paper is organized as follows. In the next section, we describe the PB-routing model in greater detail and prove some of its properties analytically. This is followed by Sections 3 and 4 where we describe our implementation of traffic-aware routing using the PB-routing paradigm and evaluate its performance. We then describe related work in Section 5 and conclude in Section 6.

2. The PB-routing Paradigm

In this section, we present the key theoretical ideas underlying the PB-routing paradigm. We emphasize here that the PB-routing paradigm represents a *family of network routing algorithms*. Hence, we first provide a description of the generic PB-routing algorithm. Using this generic formulation, we prove the key properties that are common to all the algorithms in the PB-routing family. We then describe two specific instantiations of the PB-routing paradigm. The first instantiation is the standard shortest path algorithm — we refer to this as **SPP**. We then show how **SPP** can be modified to be traffic-aware (called the **PBTA** algorithm) and analyze its desirable properties.

System Model. In order to describe the PB-routing paradigm, we first need to define some terminology and a system model. We model a network of nodes connected by bidirectional links as a directed graph $G = (N, E)$. The set of nodes in the network is represented by the set of vertices N in G. Similarly, the set of edges E in G corresponds to the set of links in the network, where e_{uv} is a directed edge from vertex u to vertex v with cost metric c_{uv} that is strictly positive. Since the network links represented by the edges in E are bidirectional, it is easy to see that if edge $e_{uv} \in E$, then $e_{vu} \in E$. For the rest of this paper, we shall use the terms nodes (links) and vertices (edges) interchangeably. Each node v can act as a traffic source and/or sink. Furthermore, every node v has a set of $Z(v)$ neighbors denoted by $nbr(v)$. Thus, the indegree and outdegree of any node v are both equal to $Z(v)$.

2.1 Routing with Potentials

The PB-routing paradigm defines a scalar field on the network over which packets are routed. The potential at any node v is a function of v and the destination d for which we need to find a route. More formally, with each node v (and destination d), we associate a potential $V^d(v)$ that is *single-valued*. Note that if the destination d changes, the potential function for v changes as well. We prove all the properties of PB-routing assuming that the destination d is fixed. Since the potential functions for different destinations are independently defined, it follows that our assumption about a fixed destination is not restrictive. For the rest of this paper, we shall use $V(v)$ to denote the potential at a node v when the destination is clear from the context.

Now consider a packet p at a node v whose destination is node d. In order to reach d, p must be forwarded to one of the $Z(v)$ neighbors of v. To determine this "next hop" neighbor, we define a "force" on the packet p at v based on the potentials at v and its neighbors. For a neighbor $w \in nbr(v)$, we can define the force $F_{v\to w}$ as the discrete derivative of V with respect to the link metric as

$$F_{v\to w} = \frac{(V(v) - V(w))}{c_{vw}} \tag{1}$$

The packet p is now directed to the neighbor $x \in nbr(v)$ for which the force $F_{v\to x}$ is maximum and positive. In other words, each packet follows the direction of the steepest gradient downhill to reach its destination. We now prove the following general property of the PB-routing paradigm.

THEOREM 2.1. *The PB-routing paradigm is loop-free if the potential function $V(v)$ is time invariant.*

Proof: We prove this by contradiction. Consider a packet p that is routed along a closed loop on the network, beginning and ending at node v. Let this closed loop be the directed path $v = v_0 \to v_1 \to v_2 \to \cdots \to v_{k-1} \to v_0 = v$. For p to be routed along this path, the work done defined by the forces in equation (1) must be strictly positive. This is because the routing algorithm always directs packets in the direction of the maximum positive force. More formally,

$$\sum_{i=0}^{k-1} F_{v_i \to v_{(i+1) \bmod k}} \cdot c_{v_i v_{(i+1) \bmod k}} > 0 \tag{2}$$

Using equation (1), we get

$$\sum_{i=0}^{k-1} V(v_i) - V(v_{(i+1) \bmod k}) > 0 \tag{3}$$

Since $V(v)$ is a time invariant, single valued function of v, the LHS of equation (3) must be identically zero, which is a contradiction. Hence, the PB-routing paradigm is loop-free as long as the potential function is single-valued and time invariant. ∎

Now consider any packet p at a node v. Since p always moves in the direction of the maximum positive force, p will be forwarded to a neighbor w that satisfies the following

$$V(v) - V(w) > 0 \tag{4}$$

However, this is not possible if v is a local minima,[1] i.e., we have

$$\forall w \in nbr(v)\ V(v) - V(w) < 0 \tag{5}$$

[1] The term *local minima* means that the potential of p is lower than that of any of its neighbors.

In other words, p will get stuck at a node v if v is not the destination but is a local minima. This implies the following

LEMMA 2.2. *If the potential function has a minimum only at the destination and no other local minima, all packets will eventually reach their destinations.*

2.2 The SPP Algorithm using PB-routing

We now describe **SPP** in detail. In the traditional shortest path algorithms, a packet traverses the shortest path from source s to destination d. Typically, shortest paths can be computed using well known algorithms (e.g. Dijkstra's shortest path algorithm [7]).

Now, let p be a packet at a node v going to destination d, and let D_{vw} be the length of the shortest path in the network graph connecting nodes v and w. We set the potential at node v to be

$$V^d(v) = V(D_{vd}) \tag{6}$$

where $V(x) = ax + b, a > 0$, is a single-valued, monotonically increasing, linear function of x. To reach destination d, a packet p at node v selects the next hop $w \in nbr(v)$ such that the force

$$F_{v \to w} = \frac{V(D_{vd}) - V(D_{wd})}{c_{vw}} \tag{7}$$

is maximum and positive. It is now easy to see that **SPP** is loop-free by Theorem 2.1 in the absence of topology changes since the potential function V is single-valued and time invariant everywhere.

To show that each packet p eventually gets to its destination if **SPP** is used, we first prove the following lemma.

LEMMA 2.3. *The potential function V has no local minima.*

Proof: The proof is by contradiction. Let v be a node that has a local minima. In other words, we have

$$\forall w \in nbr(v)\ V(D_{vd}) < V(D_{wd}) \tag{8}$$

Since v is a monotonic increasing function of D_{wd}, equation (8) implies that

$$\forall w \in nbr(v)\ D_{vd} < D_{wd} \tag{9}$$

Now let u be the next hop on the shortest path from v to d. Then, using the properties of the shortest path computation algorithms, we have

$$D_{vd} = c_{vu} + D_{ud} \tag{10}$$

where c_{vu} represents the cost metric for the link e_{vu}. However, since $u \in nbr(v)$, using equation (9), we have

$$D_{vd} < D_{ud} \tag{11}$$

Using equations (10) and (11), we conclude that $c_{vu} < 0$ which contradicts the assumption that link metrics are strictly positive. ∎

Now consider the destination d. Using the fact that link metrics are strictly positive, and that V is a monotonically increasing function, we have $V(D_{dd}) = V(0) < V(D_{vd})$, where $v \in N$ and $v \neq d$. We therefore conclude that the potential function has a minimum at the destination, and no other local minima. Using Lemma 2.2, we assert that every packet p is guaranteed to eventually reach its destination.

Finally, we show that **SPP** does indeed route using the shortest path. To prove this property, we use the following lemma.

LEMMA 2.4. *For any node v, and destination d, if the next hop computed by the shortest path algorithm is u, then the next hop computed by **SPP** is also u.*

Proof: Wlog, let $w \in nbr(v)$ be such that $w \neq u$. Then, by the shortest path property, we have

$$c_{vu} + D_{ud} = D_{vd} \leq c_{vw} + D_{wd} \tag{12}$$

This implies that

$$\frac{D_{vd} - D_{wd}}{c_{vw}} \leq 1 \tag{13}$$

and

$$\frac{D_{vd} - D_{ud}}{c_{vu}} = 1 \tag{14}$$

Using equations (13) and (14), we get

$$\frac{D_{vd} - D_{ud}}{c_{vu}} \geq \frac{D_{vd} - D_{wd}}{c_{vw}} \tag{15}$$

Using equation (15) and the fact that $V(x)$ is a monotonically increasing linear function of x, we conclude that

$$\frac{V(D_{vd}) - V(D_{ud})}{c_{vu}} \geq \frac{V(D_{vd}) - V(D_{wd})}{c_{vw}} \tag{16}$$

In other words, the force in the direction of u is maximum and positive. Therefore, **SPP** chooses u as the next hop. Note that we make the implicit assumption here that if there are multiple paths with the same minimum cost, both algorithms use the same deterministic procedure to break ties. ∎

COROLLARY 2.5. *Let V be of the form $V(x) = ax + b, a > 0$. Then, for any node v, and a node $u \in nbr(v)$, we have $\frac{V(D_{vd}) - V(D_{ud})}{c_{vu}} \leq a$.*

It is now easy to see that **SPP** simulates the standard shortest path routing. We know that both the algorithms compute the same next hop at every node for every packet with destination d. Thus, every packet from source s to destination d follows the same sequence of links in both cases. Therefore, we have the following property

THEOREM 2.6. ***SPP** correctly simulates the standard shortest path routing algorithms.*

2.3 Generalizing Potentials to be Traffic-Aware

We now show how the PB-routing paradigm can be used to construct traffic-aware routing algorithms. In order to do this, we have to design a potential that includes a traffic component. For the purposes of this paper, we use the outgoing queue sizes at a network node v as a measure of traffic at that node.[2] In the rest of this section, we describe the design of the traffic potential in greater detail followed by proofs of its relevant properties.

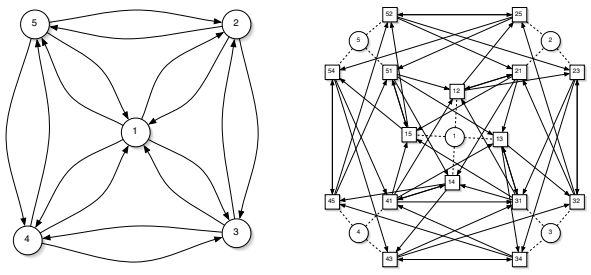
2.3.1 Design of the traffic potential.

In order to design a traffic potential, we first introduce some more notation. Let Q_{vu} denote the queue length on the outgoing link e_{vu} adjacent to node v in the original network graph. The quantity Q_{uv} is defined similarly. Let BW_{vu} be the bandwidth associated with e_{vu} — we assume that $BW_{vu} = BW_{uv}$ and $c_{vu} = c_{uv}$, where c_{vu} is the cost metric associated with the link e_{vu} in the network graph. Finally, let the normalized queue length on a link e_{vu} in the original graph be

$$q_{vu} = \frac{Q_{vu}}{BW_{vu}} \tag{17}$$

The normalized queue length on a link represents the time it will take for the current queue on that link to drain. For the rest of this paper, we use the terms *queue length* and *normalized queue length* interchangeably.

[2]Note that instead of the outgoing queue sizes, some other metric may also be used as a measure of traffic.



(a) (b)

Figure 1: (a) A network represented by a directed graph, and (b) the corresponding transformed graph over which the traffic potentials are defined. The square boxes in (b) are the nodes that correspond to the edges in (a). The lines with double arrows represent two directed edges.

Graph Transformation. In order to design a traffic potential based on queue lengths, we need to take into account the fact that queues exist on network links and not on network nodes. This is because the queues for buffering packets reside on the linecards in most modern routers. For simplicity, we assume that there is a 1-1 mapping between router linecards and links. For this purpose, we define a transformation on the original graph $G = (N, E)$ that represents the actual network. Let the transformed graph be $G' = (N', E')$, where N' and E' are defined as follows

$$N' = \{e_{vu} | e_{vu} \in E\} \tag{18}$$

$$E' = \{(e_{vu}, e_{uw}) | (e_{vu} \in E \wedge e_{uw} \in E)\} \tag{19}$$

Thus, the nodes of the transformed graph G' are the directed edges of the network graph G. The edges of the transformed graph represent common nodes between edges in the original graph. Two edges in the original graph have a common node iff the head of the first edge is the tail of the second edge. It is easy to see that any node e_{vu} in N' has an outdegree $Z(u)$, and an indegree $Z(v)$. The generalized degree $\mathcal{Z}(e)$ of the node $e = e_{vu}$ is defined as

$$\mathcal{Z}(e) = \mathcal{Z}(e_{vu}) = \frac{Z(u)}{c_{vu}} \tag{20}$$

We denote the maximum generalized degree of a node in G' as $\mathcal{Z}_{max}$.

A network graph and the corresponding transformed graph are shown in Figures 1(a) and (b), respectively. Each square box in the transformed graph corresponds to an edge in the original graph. The edges in the transformed graph obey the rules in (19). The circles in Figure 1(b) are the nodes from the original graph that have been superposed — they are not part of the transformed graph. The dotted lines connect a superposed node to each of its outgoing edge nodes in the transformed graph.

Next, we define a matrix operator A on the transformed graph G' as follows

$$A_{vu,xw} = \begin{cases} \frac{1}{c_{vu}} & \text{if} \quad u = x \\ 0 & \text{if} \quad u \neq x \end{cases} \tag{21}$$

The matrix element between the node e_{vu} and the node e_{uw} is the inverse of the cost metric for the edge e_{vu} in the original network graph.[3] Note that this matrix A is not symmetric.

We are now ready to define a traffic potential Φ_{vu} on each node e_{vu} of the transformed graph G'. This means that the traffic potential is really defined on every edge e_{vu} of the original network graph G. We require that the traffic potential at each node e_{vu} in G' be the maximum of q_{vu} and the solution to the discrete laplace's equation (see [17])

$$\sum_{w \in nbr(u)} A_{vu,uw}(\Phi_{uw} - \Phi_{vu}) + (\mathcal{Z}_{max} - \mathcal{Z}(e_{vu}))(0 - \Phi_{vu}) = 0 \tag{22}$$

The physical interpretation of the second term in the sum is that for every vertex e_{vu} on the new graph G' with generalized degree $\mathcal{Z}(e_{vu}) \leq \mathcal{Z}_{max}$, there are $\mathcal{Z}_{max} - \mathcal{Z}(e_{vu})$ *ghost nodes* connected to e_{vu} with edges of unit cost. We require that the value of the scalar field at these ghost nodes is zero, which defines the boundary conditions for the discrete laplace's equation above. The equation (22) then has a unique solution (see [14] for a proof).

We now present a more intuitive picture of the traffic potential. The traffic potential function corresponds to the surface of a taut elastic membrane that covers the network like a tent. The queues on the links can be thought of as vertical poles that hold up the membrane, which is "pegged down" at the ghost nodes. This implies that the tent surface is "propped up" by the larger queues and the smaller queues do not touch the tent surface. Hence the potential at a node e_{vu} in the transformed graph is the larger of the solution to equation (22) and the normalized queue length on the edge e_{vu} in the original network graph. More formally,

$$\Phi_{vu} = \max\left(\frac{1}{\mathcal{Z}_{max}} \sum_{w \in nbr(u)} A_{vu,uw}\Phi_{uw},\ q_{vu}\right) \tag{23}$$

Let $r(v)$ be the ratio of the maximum link cost metric to the minimum link cost metric among all the outgoing links adjacent to node v. Then, we can define the potential Φ_v at a node v in the original network graph as

$$\Phi_v = \max\left(\frac{1}{Z(v)} \sum_{w \in nbr(v)} \Phi_{vw},\ \frac{r(v)\Phi_{max} + \Phi_{min}}{r(v) + 1}\right) \tag{24}$$

where Φ_{max} is the maximum traffic potential on an outgoing link adjacent to node v, and Φ_{min} is the minimum. In other words, the traffic potential on any node in the original network graph G is the maximum of two quantities — the average of the potentials on the outgoing edges of the node, and a weighted average of the maximum and minimum potentials on the outgoing edges of the node.

2.4 PBTA — Traffic-Aware Routing with Potentials

We now describe how to route packets on the network graph using the traffic potential. Consider a packet p at node v with destination d. We define an effective potential on the graph that combines the effect of traffic load with the standard shortest path routing algorithm. The value of this potential at node v given by

$$\mathcal{V}(v) = (1 - \alpha)V(D_{vd}) + \alpha\Phi_v \tag{25}$$

and its value on the edge e_{vu} given by

$$\mathcal{V}(vu) = (1 - \alpha)V(D_{ud}) + \alpha\Phi_{vu} \tag{26}$$

where V is the shortest distance potential function defined in equation (6), and Φ_{vu} and Φ_v and defined by the equations (23) and (24). The parameter α ($0 < \alpha < 1$) sets the relative weights of the traffic potential Φ and the shortest distance potential V. We can then define a "force" on the packet p at node v towards a neighbor $u \in nbr(v)$ as

$$F_{v \to u} = \frac{\mathcal{V}(v) - \mathcal{V}(vu)}{c_{vu}} \tag{27}$$

[3] If there are multiple edges connecting v to u, there will be separate rows in A for each such edge.

This equation can be rewritten as

$$F_{v\to u} = (1-\alpha)F_{spp}(v,u) + \alpha F_t(v,u) \tag{28}$$

where $F_{spp}(v,u) = \frac{V(D_{vd})-V(D_{ud})}{c_{vu}}$ is the shortest path component, and $F_t(v,u) = \frac{\Phi_v - \Phi_{vu}}{c_{vu}}$ is the traffic component. The packet at v is then transmitted on the the edge e_{vw} towards which the total force is maximum and positive.

2.5 Properties of the routing Algorithm

If the traffic patterns (and hence, the queue sizes) are stationary, and the network topology does not change, then the single valued potential function $\mathcal{V}$ is time invariant. Thus, by Theorem 2.1, the **PBTA** algorithm is loop free. We address the issue of dynamic traffic patterns in Section 2.6.

We now show that the **PBTA** algorithm is stable. Following [26], we define stability to mean that no single queue length grows unbounded independently. The stability property ensures that the network load is evenly distributed — no single queue is allowed to become a bottleneck. To prove this property, we first prove the following lemma.

LEMMA 2.7. *Let the potential function for shortest path be of the form $V(x) = ax + b, a > 0$. Then, $|F_{spp}(v,u)| \leq a$ for any pair of nodes v, u such that $u \in nbr(v)$.*

Proof: From the definition of $F_{spp}(v,u)$, we know that

$$F_{spp}(v,u) = \frac{V(D_{vd}) - V(D_{ud})}{c_{vu}} \tag{29}$$

From Corollary 2.5, we know that $F_{spp}(v,u) \leq a$. Furthermore, we know by the shortest path property (applied to node u) that

$$D_{ud} \leq c_{uv} + D_{vd} \tag{30}$$

Using the fact that $c_{uv} = c_{vu}$, we have

$$-1 \leq \frac{D_{vd} - D_{ud}}{c_{vu}} \tag{31}$$

This, together with equation (29), and the fact that $V(x) = ax + b$ implies that

$$-a \leq \frac{V(D_{vd}) - V(D_{ud})}{c_{vu}} = F_{spp}(v,u) \tag{32}$$

Therefore $|F_{spp}(v,u)| \leq a$. ∎

THEOREM 2.8. *The queue length on any single link never grows without bound independently.*

Proof: We prove this by contradiction. Consider a link e_{vu} that has the largest queue length, say q_{max}. Let q_{max} be much higher compared to the queue length q_{wx} on any other link e_{wx} such that the resulting traffic potential at all links except e_{vu} satisfy the condition $\Phi_{wx} > q_{wx}$. Thus q_{max} is large enough such that no other queue besides itself has any effect on the traffic potential.

To add to the queue length q_{max} on link e_{vu}, packets must be sent along the link e_{vu} by node v to some destination d (say). Now, we know that

$$F_{v\to u} = (1-\alpha)F_{spp}(v,u) + \alpha F_t(v,u) \tag{33}$$

as defined earlier. We observe that $F_t(v,u)$ is really the slope of the surface corresponding to the traffic potential field. The force $F_t(v,u)$ from v to u must be negative since the largest queue lies in the direction from v to u. Hence, the maximum value of $F_t(v,u)$ is given by

$$F_t(v,u) \leq -\frac{q_{max}}{D_{max}} \tag{34}$$

where D_{max} is the maximum shortest path distance between any two nodes in the network. Combining equations (33) and (34), we have

$$F_{v\to u} \leq (1-\alpha)F_{spp}(v,u) - \alpha\frac{q_{max}}{D_{max}} \tag{35}$$

Using Lemma 2.7, we have

$$F_{v\to u} \leq (1-\alpha)a - \alpha\frac{q_{max}}{D_{max}} \tag{36}$$

Thus, if $q_{max} > \left(\frac{1-\alpha}{\alpha}\right) aD_{max}$, then the net force from v to u is negative. This means that every packet is directed away from the queue from v to u when the queue grows sufficiently large. Therefore, no queue on the network can grow unbounded independently. ∎

Finally, we show that the effective potential field has no minima. This implies that no packet ever gets stuck at any node except the destination.

THEOREM 2.9. *There is no node v in a graph (except the ghost nodes) where the force on a packet due to the combined shortest path and traffic potentials is negative in all directions.*

Proof: Consider a node v in the network graph that is not a ghost node. Let the shortest path potential function V be of the form $V(x) = ax + b, a > 0$. By definition, and using equation (27), the force from v in the direction of e_{vu} is

$$F_{v\to u} = (1-\alpha)F_{spp}(v,u) + \alpha F_t(v,u) \tag{37}$$

Now consider the shortest path direction. It is possible to show that if u is the next hop node in that direction, we have

$$F_{v\to u} = (1-\alpha)a + \alpha F_t(v,u) \tag{38}$$

If this force is positive, v is not a local minima. Otherwise, we have

$$(1-\alpha)a + \alpha\frac{\Phi_v - \Phi_{vu}}{c_{vu}} < 0 \tag{39}$$

Since $(1-\alpha)a$ is positive, the second term must be negative, and we have

$$(1-\alpha)a < \alpha\frac{\Phi_{vu} - \Phi_v}{c_{vu}} \tag{40}$$

Now consider the link e_{vw} such that

$$\Phi_{vw} = \min_{x\in nbr(v)} \Phi_{vx} = \Phi_{min} \tag{41}$$

Then the total force in the direction of e_{vw} is

$$F_{v\to w} = (1-\alpha)F_{spp}(v,w) + \alpha\frac{\Phi_v - \Phi_{min}}{c_{vw}} \tag{42}$$

Using Lemma 2.7, we can show that

$$F_{v\to w} \geq -(1-\alpha)a + \alpha\frac{\Phi_v - \Phi_{min}}{c_{vw}} \tag{43}$$

Using equation (40), the definition of $r(v)$, and the definition of Φ_v, we have

$$\begin{aligned} &-(1-\alpha)a + \alpha\frac{\Phi_v - \Phi_{min}}{c_{vw}} \\ &> \alpha\left(\frac{\Phi_v - \Phi_{min}}{c_{vw}} - \frac{\Phi_{vu} - \Phi_v}{c_{vu}}\right) \\ &> \frac{\alpha}{c_{vw}}\left((\Phi_v - \Phi_{min}) - r(v)(\Phi_{vu} - \Phi_v)\right) \\ &\geq 0 \end{aligned}$$

Therefore, the force $F_{v\to w}$ is positive, and v is not a local minima, which proves the theorem. ∎

2.6 Stability Issues

Earlier, we have proved that the **PBTA** algorithm is stable (implicitly) assuming that the queue length information is instantaneously available at all nodes whenever a change occurs. In other words, the stability properties that we have proved may not hold if network nodes have stale (queue length) information. Since the packet propagation delay across a network is finite, it is possible that very rapid changes in queue lengths will cause instabilities in the routing algorithm. However, in our simulations (described in Section 4), we have observed that the **PBTA** algorithm continues to be stable. We now provide some physical arguments as to why this is the case.

The nature of the **PBTA** algorithm is such that the effect of a large queue on the traffic potential is spread out over the neighbors of the queue. Consider, for instance, a graph with nodes on a regular square lattice, with exactly one queue of size q. Then, the scalar field Φ at a point at distance d from the queue satisfies $\Phi \sim q \ln(\frac{1}{d})$. Similarly, for a regular cubic lattice, we have $\Phi \sim \frac{q}{d+1}$. Thus, at least for regular graphs, we see that the scalar potential Φ due to a queue decays slowly with distance from the queue. While it is difficult to compute such expressions for arbitrary graphs, we postulate that the nature of the decay is qualitatively the same. A slow rate of spatial decay means that a large queue will influence the shape of the potential surface even at points that are not in the immediate neighborhood.

Following the tent analogy, the large queues are the high "poles" that poke into the taut elastic tent fabric, propping it up and determining the shape of the surface. In contrast, smaller queues (i.e. queues for which $q_{vu} < \frac{1}{\mathcal{Z}_{max}} \sum_{w \in nbr(v)} A_{vw,wx} \Phi_{wx}$) do not touch the tent fabric and therefore have no effect on the contour of the potential surface.

We also observe that the relative changes in large queues occur at a slow rate — more specifically, for a large queue of size q, $\frac{\delta q}{q}$ changes slowly since q is large. Thus, we can say that *the shape of the potential surface is almost wholly determined by the large queues, which change relatively slowly.*. For the most part, we have observed that the rate of this change is slower than the mean packet (carrying queue length information) propagation delay across the network.

Furthermore, the *flooding optimization* described in Section 3 shows that the network can (in effect) be partitioned into "sub-domains". A key property of such a partition is that the queue lengths on links outside the sub-domain do not affect the traffic potential field inside the sub-domain. Given that a sub-domain is typically smaller than the whole network, the mean packet propagation delay across a sub-domain is much smaller than the time required to flood queue length information across the whole network. Hence, it is even more unlikely that the shape of the potential surface changes faster than it takes for a packet (containing queue length information) to traverse a sub-domain (on an average). In other words, it is very improbable that the traffic potential field computed using (possibly stale) queue length information at a node would be significantly different from the field computed with the latest queue length information. Therefore, it is highly unlikely that the **PBTA** algorithm will exhibit instability.

In summary, we note that a key requirement for the **PBTA** algorithm to be stable is that *the traffic potential surface changes slower than the time it takes for a packet to traverse a sub-domain*. While this could be viewed as a limitation, we have informally argued above that for most realistic network scenarios, such is not the case. This is reflected in our simulations, which show stable performance even in the face of stale queue length information. Of course, a more formal analysis of the regimes where the **PBTA** algorithm is stable for arbitrary network topologies and traffic matrices is intractable.

2.7 Other Traffic-Aware Routing Algorithms

It is possible to define other traffic-aware routing algorithms as alternatives to the **PBTA** algorithm. However, we are not aware of any such algorithm that provides the following key benefits:

- Provable stability properties — no single queue height diverges independently when the **PBTA** algorithm is used. As we have argued in Section 2.6, the stability properties hold in a dynamic setting so long as the mean packet propagation delay within a "domain" is smaller than the rate at which the potential surface within a domain changes.
- We provide a *physical* criterion for deciding which queue lengths are relevant for the route computation process, without using any heuristic rules. Consequently, we are able to develop a flooding optimization (see Section 3) that limits the regions over which queue length information has to be propagated. This reduces overheads significantly without compromising performance.

We provide a more concrete illustration of the advantages of the **PBTA** algorithm by comparing it to a heuristic that uses shortest path routing where the link metrics are set to be proportional to the queue lengths on the links. A detailed analytical examination of the properties of this heuristic is beyond the scope of this paper. However, qualitatively speaking, the queue length based approach has the following disadvantages:

Bottlenecks. Since the link metrics in shortest path computations are additive, a path with many medium queues may be rejected in favor of a path with mostly small queues and a single large one. This means that individual links may become bottlenecks causing the corresponding queue lengths to diverge independently. In contrast the **PBTA** algorithm is able to distribute the network load more evenly over all queues.

Avoiding Congested Areas. The queue length algorithm determines the link metrics based solely on the queue lengths on the link, and does not take into account neighboring links. Consequently, packets may avoid congested links by minimal deviations around them. Therefore, the packets do not avoid areas of congestion from far enough away (as in the **PBTA** algorithm, where the effect of large queues can be felt far enough away). As a result, large queues on congested links can take longer to drain, causing bottlenecks to last longer.

2.8 Limitations

We now discuss some limitations of the **PBTA** algorithm. The first limitation is related to how fast the senders transmit packets. Clearly, if the sending rates are low compared to the link capacities, queues do not build up significantly. Therefore, no significant improvements in end-to-end delays is observed. On the other extreme, if sending rates are extremely high compared to link capacities, it is possible that all links in the network get saturated, and there are no alternate paths left for packets to traverse. In such a case also, the **PBTA** algorithm fails to improve performance. While it is difficult to estimate (for arbitrary network topologies) the range within which the improvements shown by the **PBTA** algorithm are significant, we have actually been able to verify this using simulations.

The second limitation relates to the nature of end-to-end delays. We know that the **PBTA** algorithm improves performance by attempting to route around large queues in the network. Therefore, significant improvements in end-to-end delays will not be observed in networks where queueing delays are very small compared to link latencies, such as in satellite networks. Furthermore, if the link propagation delays are very long, the routing algorithm may not be able to adapt to changing network conditions fast enough, thereby causing instabilities (see Section 2.6 for details).

Finally, if the network graph is sparsely connected (a linear graph in the worst case), the **PBTA** algorithm does not perform any better than

the standard shortest path algorithm. This is because typically, there are no alternate paths to the destination. In some cases it is possible that there is a bottleneck link which *must* be traversed in order to get to the destination, i.e., a cut-edge connecting two biconnected components. In such a case, if the link has a very high queue size, there are two possible alternatives for a packet wishing to traverse the link. The first possibility is to add to the queue irrespective of its size, and potentially get dropped. The other possibility is to go into a "holding pattern" (by traveling in some other direction away from the destination) till the bottleneck queue drains. In practice, the choice of the alternative depends on the setting of α, the maximum queue size, and the TTL (time to live) parameter of a packet. These can be tuned to achieve the desired behavior.

3. Implementation

In this section we describe our implementation of the **PBTA** algorithm using the *ns* simulator and elaborate on some of the implementation issues that arose. The *ns*-based implementation is used to evaluate the performance of the **PBTA** algorithm in the next section.

The implementation of the **PBTA** algorithm is based on a link-state routing protocol (such as OSPF [19]). Routers running a link-state protocol compute routes to different destinations based on information obtained from a link-state database. Each router (i.e., network node) has a copy of the link-state database that reflects the state of the network — the copies of the link-state database at the different routers are kept consistent by the routing protocol. Whenever the network state changes (for example, if a link fails), the network node that detects the change floods this information across the entire network by encapsulating it in a link state advertisement packet (or LSA). When new information is received, a network node recomputes routes to various destinations (typically) using a shortest path algorithm.

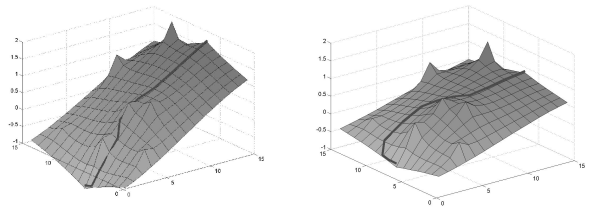
Our implementation modifies both the route computation algorithm and the information dissemination process. The route computation algorithm uses the link metric information and the queue length information in the link state database to compute routes, as described in Section 2.4. The information dissemination process propagates queue length information in addition to other link-state information. This raises three main issues: when and how frequently are link state updates (especially, queue length updates) sent out, how is the parameter α (the parameter that assigns the relative weights of the shortest path and traffic potentials) determined for a network, and how far is a given link state update (carrying queue length information) propagated.

Frequency of Update Information. The link-state update process is "trigger-based". Whenever a link/node failure or recovery occurs, a new LSA is flooded across the network. In addition, each node monitors the queue length associated with each of its outgoing links. An LSA is sent out whenever the relative change in queue length exceeds a pre-configured threshold value, q_f, where $0 < q_f < 1$.

In addition, when a link state update is scheduled for transmission, the packet encapsulating it is always placed at the head of the appropriate outgoing queue. Thus, control packets are given priority over data packets in the implementation. This ensures that LSAs are disseminated across the network in a timely manner without being slowed down by heavily congested queues. Such a practice is fairly common in modern routers.

Configuring α. The setting of α is more involved. Obviously, α depends to some extent on the network topology. We now describe a criterion that can be applied without detailed knowledge of the network topology.

Let the shortest path distance between maximally separated nodes in the graph be D_{max}. In response to a change δq at the queue q on



(a) (b)

Figure 2: Choice of α: (a) shows a potential surface when α is low – the SPP metric has more significance here which is reflected by the higher tilt of the plane. (b) shows a potential surface when α is high — the tilt is lower since the traffic potential has more significance. The dark line on each surface shows a route avoiding the high peaks, which are areas of high congestion.

link e, the force due to the traffic potential at a neighboring node u will change at least by

$$\delta F \sim \alpha \frac{\delta q}{D_{max}} \tag{44}$$

This worst-case scenario is achieved if the graph is effectively one-dimensional. For example, in an effectively two dimensional graph, we have a smaller change in F given by $\delta F \sim \alpha \frac{\delta q}{D_{max} \ln D_{max}}$.

Using equations (25) and (26), we can conclude that the maximum force is totally dominated by the traffic component (no matter what the shortest path related potential is) if

$$\delta F \sim \alpha \frac{\delta q}{D_{max}} \gg (1 - \alpha) \tag{45}$$

This means that the packets can get routed in a fashion that tries to avoid large queues as much as possible, without any regard to what the shortest path is. Hence, α should be set such that $\alpha \frac{\delta q}{D_{max}} \approx (1 - \alpha)$. We therefore choose α to be

$$\alpha = \alpha_0 \cdot \frac{D_{max}}{\delta q + D_{max}} \tag{46}$$

where $0 < \alpha_0 < 1$ is an initial value that is scaled by a factor given by the rest of the expression in (46). In our implementation, we set $\alpha_0 = 0.33$. The effect of choosing different values of α on the potential surface is shown in Figure 2. In our simulations we found that for a large variety of networks, α_0 can take on values over a fairly wide range (between 0 and 1) and still provide (close to) optimal performance. Therefore, it does not seem necessary to carefully tune the initial value α_0 by trial and error for a given network.

Distributing the Update Information. We now describe a way to reduce the control overheads by restricting how far a particular queue length update is sent from the originating node. We call this optimization the *flooding optimization*.

To understand qualitatively why such an optimization works, consider the behavior of the traffic potential Φ due to the queue on some link e. As explained earlier (see Section 2.6), this potential Φ decays (albeit slowly) with distance d from the actual queue. If indeed, as we postulate earlier, this decay obeys the power law, we can say that the change in traffic potential $\delta\Phi$ at a distance d from the queue whose length has changed by δq scales as $\frac{\delta q}{d^\gamma}$. Thus, the change in force δF, derived from the expression in equation (27) is given by

$$\delta F \sim (1 - \alpha)C + \alpha \frac{\delta q}{d^\gamma} \tag{47}$$

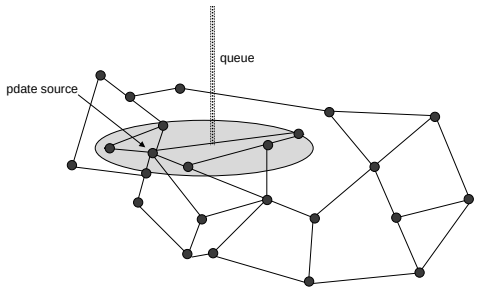


Figure 3: The shaded area represents the sub-domain to which the update source sends information about the queue length. This is more efficient than flooding, which would require the queue length information be sent to every network node.

where the first term is due to the shortest path potential. From Lemma 2.7, we know that $|C| \leq a$, where the shortest distance potential V is of the form $V(x) = ax + b, a > 0$. Thus, beyond a distance

$$d_{max}(\delta q) \sim \left[\alpha \frac{\delta q}{a(1-\alpha)}\right]^{\frac{1}{\gamma}} \quad (48)$$

the scale of the force due to the traffic potential becomes smaller than the scale of the force due to the shortest path potential, and hence does not affect the route computation any more. Therefore, the changes in the queue information do not have to be transmitted to the nodes on the graph beyond d_{max}. This distance decreases with decreasing value of α. Thus, the graph is divided into (α-dependent) sub-domains and the exchange of traffic information is localized among nodes in the same sub-domain (see Figure 3).

In our implementation, we simplified this process as follows. Each node that detects that the relative queue length has changed by more than q_f, sends an update to each of its neighbors. Every neighbor (i.e., all the nodes that are one hop away from the origin of the update) recomputes its routing table using the new information. If there are no changes in the routing table, the update is suppressed — this is the boundary of the current sub-domain. Otherwise, the update is forwarded, and the process repeats. Note that the topology related updates are flooded across the entire network.

The intuition behind *never* suppressing queue length updates at the source of the update is as follows. Consider a network that has exactly one link with a very large queue. Let the set of nodes adjacent to this bottleneck link be S. Since the other queues in the network are much smaller, the shape of the potential surface is exclusively determined by the bottleneck queue. Thus the potential at every node $v \in S$ is dominated by the traffic component, and is higher than at any node $u \notin S$. As the bottleneck queue keeps growing, none of the nodes $v \in S$ will observe any changes in their routing table, being at the highest points in the potential surface (such that everything appears downhill from there). If all the nodes $v \in S$ were to suppress their update packets, then none of the nodes $u \notin S$ would receive information about the congested link. Hence, they would continue to route packets through the congested link.

Computational Complexity. At every update, for a network with $|N|$ nodes and $|E|$ edges, the computation of the shortest path related potential takes $O(|N|^3)$ time using the Floyd Warshall "all pairs shortest path" algorithm [1]. The computation of the traffic potential at the node receiving the update takes $O(|E|)$ time [22].

Topology related updates that require recomputation of shortest paths are infrequent. Queue length updates are more frequent, but are subject to the flooding optimization. The computation time for the field Φ is an overestimate because we assume that the initial value for the iterative process is $\Phi = 0$ everywhere. With every update, we do not expect Φ to change drastically. Hence, the solution should converge faster when initialized with the previous values (i.e., before the update occurs) of Φ .

The computation time at a node v is further reduced because the solution for Φ does not need to converge beyond a distance $d_{max}(\delta q)$ from v (since beyond that distance, queues do not affect the traffic potential). Thus at every update, if N_d is the typical number of edges connected to nodes within the radius $d_{max}(\delta q)$ of node v, the computation time should be $O(N_d)$ which is smaller than $O(|E|)$.

Finally, we consider the issue of storage space. As explained earlier in Section 2.4, the force exerted on a packet at node v towards node u has two components: the traffic component ($F_t(v,u)$) and the shortest path component ($F_{spp}(v,u)$). To compute $F_t(v,u)$, each node must know the queue lengths on each of the edges in the network — this requires $O(|E|)$ storage. To compute $F_{spp}(v,u)$, each node must know the routing tables of all its neighbors, along with its own. This requires $O(Z(v)|V|)$ storage, where $Z(v)$ is node degree. Thus the total storage required is $O(Z(v)|V| + |E|)$.

4. Performance Evaluation

In this section, we evaluate the performance of the **PBTA** algorithm using simulations. The simulations were performed using the network simulator *ns* [20]. We used three different network topologies and both constant bit rate (CBR) as well as bursty traffic sources. We first describe the network topologies, followed by the experimental results.

4.1 Network Topologies

In order to evaluate the **PBTA** algorithm, we use three different network topologies. The characteristics of each topology are summarized in Table 1. The first two topologies were generated using the BRITE topology generator [18], and the third topology is based on a real ISP topology. In each case, we only used a single-level hierarchy of routers (i.e., a single AS consisting of multiple routers) since we envision our algorithms to be useful for intra-domain routing.

The first topology, labeled **WAX** , was generated using the Waxman [29] model, with randomly placed nodes on a 2-dimensional plane. Nodes were added incrementally, with each new node connecting to 2 existing nodes. The values of the Waxman-specific α and β parameters were set to 0.15 and 0.2, respectively.

The **BA** topology was generated using the Barabasi-Albert model [4]. This model postulates that a common property of large networks is that the vertex connectivities follow a scale-free power-law distribution. As before, the nodes for the **BA** topology were randomly placed on a 2-dimensional plane, with each new node connecting to 3 existing nodes.

Finally, the **ISP** topology is based on the network topology of a real Internet Service Provider (ISP). Owing to scaling limitations of *ns*, we have only used a representative subset of the entire topology. This subset consisted of all the nodes in the core of the ISP network. The nodes at the edge were removed since there was little redundancy in the topology near the edges. Furthermore, we have slightly modified the topology for reasons of confidentiality. For uniformity of comparisons, we have set the link cost metrics to 1, the link bandwidths to 1Mbps and the delays to 5ms in all the three topologies, even though the link costs, bandwidths and delays in the **ISP** topology were not identical for all the links.

Experimental Methodology. We ran simulations over all the three network topologies shown in Table 1. In each case, half of the nodes sent traffic to the other half for 60 seconds. During this time, the sender-receiver node-pairs were chosen at random and changed every 10 seconds. The simulation then continued till all the data packets

Label	Nodes	Links	Type	BW	Delay
WAX	35	70	Waxman	1Mbps	5ms
BA	35	99	Barabasi-Albert	1Mbps	5ms
ISP	28	56	real ISP	1Mbps	5ms

Table 1: The three network topologies that were used in the simulation experiments.

in transit reached their destinations. By choosing the sender-receiver node-pairs in this fashion, we were able to ensure that the generated traffic was not restricted to some specific part of the network topology. Therefore, the routing algorithms were not able to leverage the special characteristics (if any) of a specific part of the network to improve the end-to-end packet delays and jitter. For the rest of this section, we refer to the set of source-destination pairs (randomly) selected for generating traffic in this manner as the set of *viable* source-destination pairs. For all these experiments, we set α_0 to 0.33 and q_f (trigger for queue length update LSAs) to 0.9. Finally, the relative change in queue length was computed as $\frac{|curr-last|}{last}$, where $curr$ is the current queue length and $last$ is the queue length the last time an LSA was sent out. To bootstrap this process, the first queue length update packet was sent out when the queue length exceeded 90000 packets for the first time.

To compare the performance of the **PBTA** algorithm to the standard shortest path algorithm, we ran the simulations after setting the maximum queue size at each network node to infinity. In other words, the queues at each node were allowed to grow without bounds. This enabled us to make meaningful comparisons between the delay (or jitter) values for viable source-destination pairs when the two different routing schemes are used.

Note that if the maximum queue sizes were set to some finite value apriori, the end-to-end delay under heavy load would max out at some function of this maximum value. Once this happens, it would no longer be possible to fairly compare the delay and jitter values generated by the two routing schemes. Of course, in such cases, the packet loss rate would be a good indicator of performance. We first focus on the end-to-end delay and jitter metrics — loss metrics are discussed later.

Once the per packet data was obtained for each network using simulation runs, we computed the average delay and jitter for all the viable source-destination pairs for both the **PBTA** algorithm and shortest path routing. The same set of (randomly chosen) source-destination pairs were used in all the experiments. The average delay and jitter data for each network were then converted to two scatter plots, one for delay and the other for jitter. We discuss the performance of the **PBTA** algorithm for both CBR and bursty sources in the next subsection.

4.2 Performance Analysis

CBR Traffic. We first show the performance data for constant bit rate (CBR) traffic. We tested each of the three networks with 5 different sending rates — 0.5Mbps, 0.8Mbps, 1Mbps, 1.2Mbps, and 1.5Mbps. We show the data for a subset of these data rates in Figures 4(a) through (f).[4] Each graph represents either the mean end-to-end delay or jitter for one of the three topologies in Table 1. The Y-axis (labeled SPP) shows the mean delay (respectively, jitter) for shortest path routing, and the X-axis (labeled PBTA) shows the mean delay (respectively, jitter) for the **PBTA** algorithm. Each point in the graph represents a viable source-destination pair for the given topology. The Y-coordinate of the point indicates the mean delay (or jitter) for shortest path routing and the X-coordinate indicates the value of the same metric for the **PBTA** algorithm. The diagonal line indicates the break-even point, where the average delay (or jitter) for the **PBTA** algorithm is the same as that for shortest path. The sector above and to the left of the diagonal denotes the regime where the **PBTA** algorithm outperforms the shortest path algorithm.

The scatter plots show that a large majority of points lie above the diagonal for each data rate shown. Therefore, we can conclude that for the majority of the viable source-destination pairs in each network, the **PBTA** algorithm outperforms the shortest path routing scheme with respect to both delay and jitter. As the sending rate increases, the **PBTA** algorithm typically performs better than shortest path routing. This is because at low sending rates, there is not enough queue buildup in the network for the **PBTA** algorithm to really differentiate itself.

Bursty Traffic. We now come to the bursty traffic scenario. For this we used the same scenario as in the CBR experiments, except that the traffic sources sent out traffic using a Pareto distribution instead of at a constant rate. The Pareto distribution has been shown to be a good approximator of Internet traffic which is inherently bursty in nature [21]. The sources in our experiments sent out traffic in bursts for 800ms ("on" time) followed by 200ms of no traffic ("off" time). The three burst rates used were 1.5Mbps, 1.2Mbps and 1.0Mbps, and the value of the shape parameter was set to 1.4 (it must lie between 0 and 2).

The results are shown in Figures 5(a) through (c). This time, we only present the mean end-to-end delay times due to lack of space. The improvements in jitter numbers are similar. We find that with bursty traffic also, the performance improves for all the three networks. We conclude that the **PBTA** algorithm can potentially improve end-to-end delay and jitter if deployed in the Internet.

Interaction with TCP. One of the interesting aspects of the **PBTA** algorithm is that it may route different packets belonging to the same traffic flow[5] over different paths in the network. This can happen if some link along the current path gets congested and the routing component chooses a different path. Consequently, packets may arrive at the destination out of order.

Such an event may be detrimental for congestion control algorithms, notably TCP, that use out-of-order packet arrivals as a sign of congestion. Whenever a TCP receiver gets an out-of-order packet, it sends a duplicate acknowledgment to the sender. If the sender receives three duplicate acknowledgments in a row, it considers this as a sign of congestion and goes into congestion avoidance mode or slow start mode [25]. As a consequence, the sending rate gets reduced.

Clearly, this could be a serious problem with the **PBTA** algorithm (and the PB-routing paradigm in general) since the algorithm depends on finding alternate paths for routing packets (possibly belonging to the same flow) in order to avoid congestion. In order to determine the effect of packet re-ordering, we performed experiments on all three networks using TCP sources. In each case, we ran an FTP sender over TCP that sent out back-to-back packets to its destination. The source-destination pairs were chosen at random and changed every 10 seconds for a total of 60 seconds, as in the previous two experiments. We then computed the number of packets that arrived out of order at their destinations. The results are shown in Table 2.

From the results, we observe that the percentage of packets that arrive out-of-order is very small. Hence we do not expect the **PBTA** algorithm to pose severe penalties in terms of bandwidth utilization due to packet re-orderings in individual TCP flows. In that sense, the **PBTA** algorithm (or more generally, the PB-routing paradigm) can be described as compatible with TCP.

[4] We were able to run the simulations for the **BA** topology for a maximum sending rate of 1Mbps. For higher data rates, *ns* ran out of memory.

[5] We assume that packets belonging to the same traffic flow have the same source/destination address/port numbers as well as protocol ID.

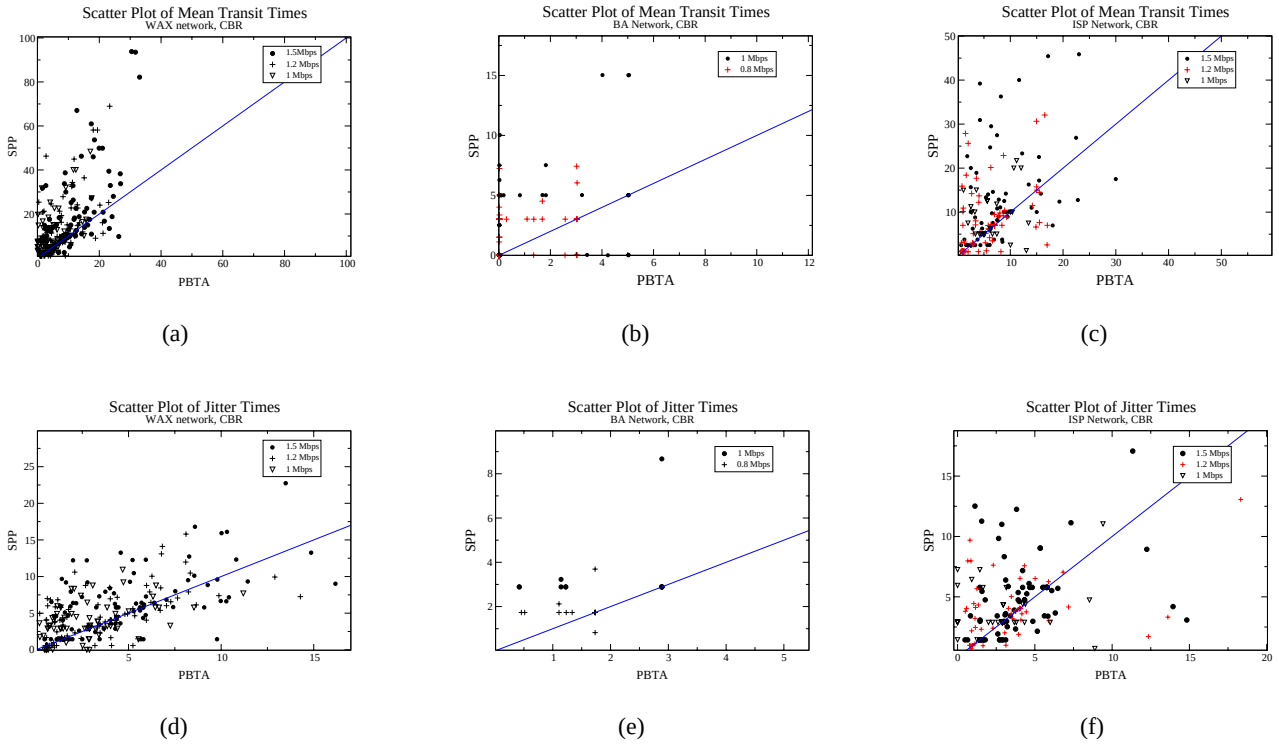


Figure 4: Performance Numbers: (a) through (c) show the mean end-to-end delay times for various sending rates using CBR traffic, (d) through (f) show the jitter in the end-to-end delay times for various sending rates using CBR traffic.

Network	Total packets	Out-of-order packets	% of out-of-order packets
WAX	78677	102	0.13
BA	100880	102	0.10
ISP	69341	84	0.12

Table 2: Number of out-of-order packets received when the PBTA algorithm is used with TCP traffic sources.

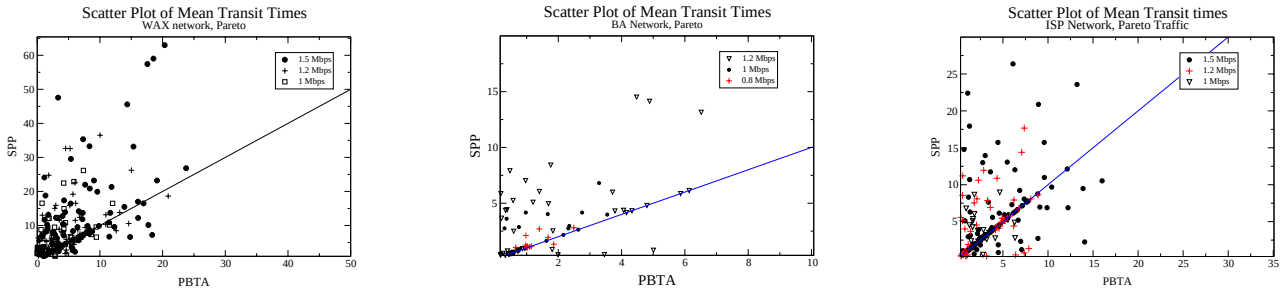
Topology/ Rate	CBR	Pareto
WAX 1.5Mbps	2.37%	2.95%
WAX 1.2Mbps	2.72%	3.20%
WAX 1.0Mbps	3.06%	3.72%
BA 1.2Mbps	NC	3.69%
BA 1.0Mbps	3.12%	4.00%
ISP 1.5Mbps	2.22%	2.66%
ISP 1.2Mbps	2.65%	2.97%
ISP 1.0Mbps	2.65%	3.09%

Table 3: Control overheads as a percentage of successfully received data packets. These numbers are for the optimized flooding algorithm. NC stands for not complete, i.e., *ns* ran out of memory.

Control Overhead. The **PBTA** algorithm requires the dissemination of queue length information across the network. This information is used to compute the traffic related potential on each link and node, as shown in Section 2.3. The less stale this information is, the more accurate is the route computation. Thus, there is a tradeoff between the quality of the computed paths (and hence end-to-end delays and jitter) and the frequency of queue length updates which translates to control overheads.

In Section 3, we proposed a flooding optimization that reduces the control overhead without compromising end-to-end delays and jitter. To determine how effective our flooding optimization is, we estimated the control overheads by computing the ratio of the number of routing protocol packets that are sent out to the number of data packets that reach their destination (in the experiments described earlier). During flooding of LSAs, each leg in the flooding was counted as a separate control packet. The control overheads for the optimized algorithm are shown in Table 3. We see that the control overheads typically vary between 2 and 4%. We also note that the control overheads mostly decrease as the data send rate increases. This is because the triggering process uses relative changes in queue lengths to send updates. Hence, as the sending rates increase, the queue lengths increase, and the updates go out less frequently. In other words, the rate at which updates get sent out by the triggering process increases much more slowly than the sending rates. Finally, the overheads for the bursty traffic sources are slightly higher than that for the CBR sources — this is to be expected since bursty sources cause more unpredictable changes in traffic patterns than CBR sources.

The effect of the flooding optimization is shown in Figure 6 for the bursty traffic sources only. For each topology, we have shown the comparison figures for the highest sending rate for which *ns* was able to complete execution in the non-optimized case without running out of memory. We see that there is a 7-fold improvement in terms of the percentage of control packets while the improvements in the delay and the jitter times were similar in both cases (the delay and the jitter data for the unoptimized case are not shown here due to lack of space). We point out here that all the previous results were shown for the optimized version of the **PBTA** algorithm.



(a) (b) (c)

Figure 5: (a) through (c) show the mean end-to-end delay times for the three networks using a Pareto traffic generator.

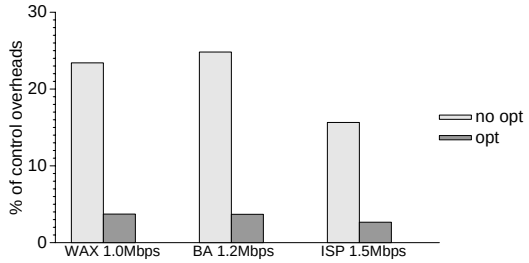


Figure 6: The effect of the flooding optimization on control overheads for bursty sources. The bars show the percentage of control overheads in both the optimized and the unoptimized cases.

Topology	Rate	CBR	Pareto
WAX	1.2Mbps	3.03	3.00
BA	1.2Mbps		4.81
BA	1.0Mbps	5.00	
ISP	1.2Mbps	3.14	4.44

Table 4: Loss rate improvements for the three topologies. All sending rates were 1.2Mbps, except the CBR rate for the BA topology, which was 1.0Mbps (*ns* ran out of memory for the 1.2Mbps rate). The table shows the factor by which the number of dropped packets went down.

We also note that these results indirectly indicate that for any network node v, the routes to destinations that are distant change much less frequently than routes to destinations that are close by. In other words, route fluctuations are confined to destinations that are close to v and thereby improves the stability of routes.

Packet Loss Rates. Finally, we estimated the effect of the **PBTA** algorithm on packet losses in the network by running simulations on all the networks using both CBR and bursty traffic sources. The queue size for each scenario was set to 10000 packets. In Table 4, we show the results corresponding to the maximum send rates for which we could run *ns* without running out of memory. The numbers in the table show the ratio of the number of packets dropped using the shortest path algorithm to the number of packets dropped using the **PBTA** algorithm. We see that the number of packets dropped improved by a factor of 3 to 5 when the **PBTA** algorithm was used. The improvement factor is approximately the same for both CBR and bursty traffic for all the three topologies.

Summary. We have shown that the **PBTA** algorithm produces significant improvements in end-to-end delay, jitter, and packet loss rates, has very reasonable control overheads, and reorders a small fraction of TCP packets. These characteristics make the **PBTA** algorithm an attractive alternative for routing in the future Internet.

5. Related Work

The steepest gradient search method [23] has been well studied in the past. This method has been extensively used for optimization problems and has had applications in such diverse disciplines as path planning in robotics [8, 15], artificial intelligence [30], and monte-carlo simulations in statistical physics [6]. The basic idea in steepest gradient search is to optimize a (non-linear) function by evaluating the function at an initial point and then moving towards an optimal point by executing small steps in the direction of the steepest gradient. In our work, we have adapted this method to identify the direction in which to route packets in a data network such that highly congested areas in the network are avoided. This is accomplished by assigning carefully designed potentials based on traffic experienced at a network node.

The area of traffic-aware routing (i.e., routing techniques that take into account traffic conditions) has also been studied extensively both from both practical and theoretical perspectives. One practical approach [2] uses probe packets to estimate delays and loss rates that are used to compute long term averages of these metrics. These averages are then used to route packets over paths with low delays and/or loss rates. A different method is to use emergency exits [28] where, on encountering congestion, a packet is routed along a previously computed alternative path to the destination.

Another work has proposed the idea of splitting the traffic into long lived and short lived IP flow patterns [24]. Traffic-aware (or load sensitive, as this work calls it) routing is used only for the long lived flows. Such a technique improves the stability of the routing algorithm and the allocation of resources for long lived flows. This is achieved by periodic (but rare) distribution of link-state updates that limits the control overheads. We believe that it is possible to use PB-routing in conjunction with the methods described in this work to route the long lived flows.

From a theoretical perspective, almost all of the algorithms that have been developed require a point-to-point traffic demand matrix to be specified. One class of algorithms uses link utilization as a measure of traffic and attempts to minimize the maximum (or worst-case) link utilization, given a set of point-to-point traffic demands. These algorithms are known as "maximum concurrent flow" algorithms for which both exact [1] and approximate [9, 11] solutions are known.

Other algorithms have attempted to minimize end-to-end delay mod-

eled as a convex function of traffic. One of the earliest works in this area was Gallager's minimum-delay routing algorithm [5]. Modifications to this work assume minor fluctuations of traffic about an otherwise "equilibrium pattern" [3, 27]. A similar work in this area [16] uses a closed queueing model to optimize network delays.

A different technique is to reduce delay (and congestion) by assigning static OSPF link weights based on a known traffic demand matrix [10]. All of the algorithms that optimize delay assuming known (and static) traffic patterns can be shown to be optimal (or close to optimal), typically by following Gallager's bounds. Our work complements these algorithms in the sense that it tolerates more dynamic traffic patterns but provides performance improvements that are of a statistical nature.

6. Conclusion and Future Work

In this paper, we have applied the idea of steepest gradient based search to Internet routing to develop a routing paradigm called PB-routing. The key concept here is to define a scalar potential field on the network and route packets in the direction of maximum force (steepest positive gradient in the potential field). We have described how this idea can be adapted to provide dynamic, traffic-aware routing by designing a traffic-based potential. Since our traffic-based potential is dominated by large, (and hence) slowly-varying queues, our routing algorithm is relatively stable and tolerant of dynamic traffic conditions. Using a combination of analytic methods and simulations, we have shown that this methodology produces loop-free routes and causes significant improvements in end-to-end delays and jitter without requiring too much control overhead.

We believe that the general PB-routing paradigm can be adapted for a variety of applications. First, we can envisage providing differentiated services by setting the relative weight of the traffic-based potential differently for different traffic classes. Priority traffic would thus experience lower end-to-end delays and jitter. Second, it is possible to apply similar techniques to adhoc networks. For example, we can use PB-routing to compute paths to route around congested areas, and even create new links on the fly to alleviate congestion by following directions of maximum force. Third, PB-routing could be used as an alternative routing methodology for overlay networks by using application specific metrics as potential functions. These and other applications would be the subject of future work in this area.

Acknowledgments

We would like to thank our shepherd Sugih Jamin, the program chairs Jon Crowcroft and David Wetherall, Steve Simon, and the anonymous referees for their valuable comments.

7. References

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating System Principles*, pages 131–145, Chateau Lake Louise, Banff, Canada, October 2001.

[3] E. J. Anderson, T. E. Anderson, S. D. Gribble, A. R. Karlin, and S. Savage. A Quantitative Evaluation of Traffic-Aware Routing Strategies. *ACM SIGCOMM Computer Communication Review*, 32(1):67–67, January 2002.

[4] A. L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.

[5] D. Bertsekas and R. Gallager. Second Derivative Algorithm for Minimum Delay Distributed Routing in Networks. *IEEE Transactions on Communications*, 32(8):911–919, 1984.

[6] K. Binder and D. W. Heermann. *Monte Carlo Simulation in Statistical Physics*. Springer Verlag, Berlin, Germany, 4th edition, August 2002.

[7] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] B. Faverjon and P. Tournassoud. A Local Based Approach for Path Planning of Manipulators with a High Number of Degrees of Freedom. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1152–1159, March 1987.

[9] L. K. Fleischer. Approximating Fractional Multicommodity Flow Independent of the Number of Commodities. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 24–31, New York, NY, October 1999.

[10] B. Fortz and M. Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. In *Proceedings of Infocom '00*, pages 519–528, Tel Aviv, Israel, March 2000.

[11] N. Garg and J. Konemann. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 300–309, Palo Alto, CA, November 1998.

[12] V. Guruswami, S. Khanna, R. Rajaraman, F. B. Shepherd, and M. Yannakakis. Near-Optimal Hardness Results and Approximation Algorithms for Edge-Disjoint Paths and Related Problems. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 19–28, Atlanta, GA, May 1999.

[13] S. Herzog. RSVP Extensions for Policy Control. RFC 2750, IETF, January 2000.

[14] J. D. Jackson. *Classical Electrodynamics*. John Wiley & Sons, New York, NY, 3rd edition, August 1998.

[15] O. Khatib and J. F. Le Maitre. Dynamic Control of Manipulators Operating in A Complex Environment. In *Proceedings of the 3rd CISM-IFToMM*, Udine, Italy, 1978.

[16] H. Kobayashi and M. Gerla. Optimal Routing in Closed Queuing Networks. *ACM Transactions on Computer Systems*, 1(4):294–310, November 1983.

[17] E. Kreyszig. *Advanced Engineering Mathematics*. John Wiley & Sons, New York, NY, 8th edition, December 1998.

[18] 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*, Cincinnati, OH, August 2001.

[19] J. Moy. OSPF Version 2. RFC 2328, IETF, April 1998.

[20] The Network Simulator — ns-2. http://www.isi.edu/nsnam/ns/.

[21] V. Paxson and S. Floyd. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[22] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, January 1993.

[23] R. L. Rardin. *Optimizations in Operations Research*. Prentice Hall, Upper Saddle River, NJ, 1998.

[24] A. Shaikh, J. Rexford, and K. G. Shin. Load-Sensitive Routing of Long-Lived IP Flows. In *Proceedings of SIGCOMM '99*, pages 215–226, Boston, MA, August–September 1999.

[25] W. R. Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, Boston, MA, January 1994.

[26] L. Tassiulas and A. Ephremides. Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, December 1992.

[27] S. Vutukury and J. J. Garcia-Luna-Aceves. A Simple Approximation to Minimum Delay Routing. In *Proceedings of SIGCOMM '99*, pages 227–238, Boston, MA, August–September 1999.

[28] Z. Wang and J. Crowcroft. Shortest Path First with Emergency Exits. In *Proceedings of SIGCOMM '90*, pages 166–176, Philadelphia, PA, August 1990.

[29] B. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1998.

[30] P. H. Winston. *Artificial Intelligence*. Addison Wesley, Boston, MA, 3rd edition, January 1992.