UNIVERSITY OF CALIFORNIA
SANTA CRUZ

HERDING PACKETS: HOW TO ROUTE PACKETS ON THE BEST PATH THROUGH A NETWORK WITH MULTIPLE CRITERIA

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Judith T. Samson

June 2016

The Thesis of Judith T. Samson
is approved:

_____

Professor Bradley R. Smith, Chair

_____

Professor Katia Obraczka

_____

Professor J.J. Garcia-Luna-Aceves

_____

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

Abstract

Herding Packets: How to Route Packets on the Best Path Through a Network with Multiple Criteria

by

Judith T. Samson

Algorithms for finding the shortest path between two nodes in a graph have been heavily studied for decades. Half a century ago solutions were found for the specific class of problems that involves finding the path of least length, when edges are weighted with a single metric. For example, although the Bellman-Ford and Dijkstra algorithms use different methodologies, both techniques depend on a single metric where the lengths of paths are found by adding path weights in the conventional way. The situation becomes more complicated when multiple metrics or single metrics other than simple Euclidean distance are involved. In these cases, the definition of "shortest," or "best" is not straightforward. In addition, it is not always a simple matter to verify that packets in a distributed network are forwarded on best paths that are guaranteed loop-free.

We call the set of edge weights, plus the rules that determine how paths are concatenated and compared, the *path algebra* [2]. We find that the structure of a path algebra is the key to understanding how packets will be forwarded in a distributed network; specifically whether they will be forwarded on best, loop-free paths. This is independent of any path-finding algorithm. We show that "best" paths and loop-free paths are separate (albeit subtly related) properties, that are dependent on whether or not a path algebra is *monotonic* and *strictly bounded*.

We examine examples of path algebras that possess various combinations of those two properties in order to illustrate how they affect the construction of forwarding paths. Finally, we prove that if a path algebra is monotonic and strictly bounded, it is guaranteed to produce forwarding paths that are best and loop-free.

Previous works [1, 2, 5, 6, 9, 11] have introduced these ideas in separate contexts. We provide a consistent framework and prove general concepts that have only been introduced or discussed in special cases.

To Lloyd

1 Introduction

One of the fundamental problems in computer science is finding the shortest path between nodes in a network. Although single-metric shortest path algorithms such as Dijkstra and Bellman-Ford were developed decades ago, they are still used in today's networking protocols. The protocols used in the Internet routing layer were designed to route homogeneous, text-based traffic in order to minimize delay. Network traffic today is much more diverse than when the protocols were designed, and the criteria for determining the best path do not always mesh well with existing shortest-path algorithms. For example, Quality-Of-Service routing must optimize multiple criteria such as bandwidth and jitter, as well as latency. Policy-based routing must also optimize on multiple criteria, such as routing where both security and reliability are a concern. Even single-metric routing is not always straightforward (although not mathematically precise, we will use "metric" and "criterion" interchangeably). Shortest-path algorithms which simply add edge weights together to find the path length are easily understood, but other metrics (such as bandwidth) are not as intuitive.

To generalize, there are a few things that we always need in order to route packets on the best forwarding paths in a distributed network. Specifically, we must have a set of metrics that forms the weight of each edge (such as bandwidth and delay), a rule for combining the edge weights to form paths, and a rule for comparing two paths to determine which path is best. This set of rules is called the *path algebra*. The properties of the path algebra also help determine the algorithm that must be used for routing and the nature of the *forwarding paths* that are created, i.e., whether they provide best, loop-free paths upon which packets travel through

the network.

It is important to note that a *forwarding path* is somewhat different from a *path* through the network. In distributed communications networks such as the Internet, a routing calculation is made independently at every node in order to determine the next hop a packet must take towards its destination. Each node has a routing table that lists the distance (as calculated from that node) and the next hop to each destination. In addition, since each node has different information (i.e. there is no information on paths upstream of the node) the resulting forwarding paths may be different than the paths computed at individual hops on the path. Thus, the best *path* calculated by a routing protocol may be different from the actual *forwarding path* that a packet takes through the network.

When more than one metric is involved in routing, it can be difficult to predict how metrics will actually behave unless the path algebras behind the metrics are carefully examined. Algorithms such as Dijkstra or Bellman-Ford tend to work as expected with metrics that behave well, but may do unexpected things with metrics that do not. Protocols based on these algorithms may compute loop-free, best paths for multiple criteria, but the actual forwarding paths taken by packets are neither best nor loop-free.

We provide a general set of tests by which *any* path algebra can be tested for whether packets will be forwarded on best, loop-free forwarding paths. Using this method we can also show that a path algebra will provide forwarding paths that may be not best but are guaranteed loop-free or vice versa. Finally, the methodology can prove when a path algebra will provide forwarding paths that are *optimal*, (meaning that subpaths of best paths are also best paths). The

methodology is independent of the routing algorithm used, since it is concerned with the algebraic structure of a path algebra.

There is a certain amount of previous work exploring this topic but much work is limited to special cases, providing no general rules and using different terminology for the same concepts. In addition, different aspects of the routing problem are attacked, which makes results difficult to compare. We first develop a single, comprehensive framework so that new work can seamlessly build upon previous work.

We begin by proving that in a hop-by-hop, destination-based network, the algebraic structure of the path algebra must have two properties: *monotonicity* and *boundedness* [6, 8, 5, 2], for packets to be forwarded over best, loop-free paths. A path algebra is *monotonic* if for subpaths a, b, c, $a \preceq b \Leftrightarrow a \oplus c \preceq b \oplus c$. A path algebra is *bounded* if $a \preceq a \oplus b$ (we fully define the meaning of $\preceq$ and $\oplus$ in the next section). The difference between boundedness and monotonicity is subtle but distinct and will be discussed in detail. By determining the algebraic structure of a path algebra we can prove which specific properties forwarding paths will (and will not) possess.

Boundedness and monotonicity are discussed in various ways in [6, 8, 5, 11, 2]. We generalize the properties of monotonicity and boundedness for every type of path algebra.

Specifically, we will show that

1. A path algebra that is *strictly bounded* produces loop-free forwarding paths, but they may not necessarily be best.

2. A *monotonic* path algebra guarantees best forwarding paths, but they may contain loops.

3. A path algebra that is both *monotonic* and *strictly bounded* guarantees best, loop-free forwarding paths.

4. A path algebra that is *strictly bounded* and *strictly monotonic* guarantees optimal forwarding paths.

2 Fundamental Definitions

The goal is to establish the significance of monotonicity and boundedness (separately and together) for path algebras. But first we must define the fundamental concepts necessary to establish the argument. In particular, we define a *forwarding path*, illustrate how it differs from a *path* and show how a forwarding path is created. We also formally define a *path algebra* and discuss the fundamental mathematical notation and concepts.

Network We define a *network* as a pair of sets $G = (V, E)$, where V is the set of nodes in the network, E is the set of edges that connect two nodes, and where every edge $e_{i,j} = (v_i, v_j)$ has a a weight w. For the following examples, w may consist of a single value such as delay ($w = d$), or an ordered pair such as delay and bandwidth ($w = (d, b)$). But there is nothing to prevent w from consisting of any number of values.

Path A *path* p is an ordered sequence of nodes $p = (v_0, v_1, \ldots, v_{n-1}, v_n)$, each connected by an edge $e_{i,j} = (v_i, v_j)$, from a source node $s = v_0$ to a destination

node $t = v_n$ in the network. The *path length* is determined by combining the weights of the edges with some kind of addition operation. The binary operation "addition" is specifically defined in each path algebra.

We make a distinction between a *path* and a *forwarding path* in a *hop-by-hop*, *destination-based* routing environment. Hop-by-hop and destination-based are defined as follows:

Hop-by-Hop As a packet travels through a distributed network, the node at each hop independently selects the next hop based on the forwarding table computed at that node. This is called *hop-by-hop* routing. [11]

Destination-Based With *destination-based* routing a node makes a forwarding decision based only on the destination of the packet. Each node maintains a routing table that lists the next hop to reach every destination in the network. Nodes have no information about upstream hops, previous nodes visited, or the original source of the packet.

Forwarding Path A *forwarding path* is defined as the path along which a packet actually travels in a hop-by-hop, destination-based network. It is crucial to note that the forwarding path may not necessarily be the same as the path computed at a node. In a distributed communications network there is no protocol that calculates the forwarding paths for the entire network. Routing decisions are made independently at each node based on the destination of a packet and are stored locally in each node's routing table. The forwarding path is the path that is implied collectively by the forwarding entries in all the nodes along the

path. In other words, the forwarding path is the path that is constructed by the independent decisions of the participating nodes, each of which have distinct points of view. Forwarding paths are an emergent property of the collective routing tables of all nodes and are not known by any single node.

In short, the *forwarding path* is the collective result of the routing algorithm run at all nodes in a network.

Best Path A *best path* algorithm computes the best paths from source to destination. Note that we use the term "best" rather than "shortest." "Shortest" denotes the specific path algebra that computes the single-criterion path of least cost (delay, for example). Also note that since routing is distributed hop-by-hop, we cannot assume that the best path produced by the algorithm is the same as a best path on which packets are actually forwarded.

Order and Total Order A binary relation $\preceq$ on set S is an *order* if it is *reflexive, transitive* and *antisymmetric*:

- $a \preceq a$ (reflexivity)
- If $a \preceq b$ and $b \preceq c$ then $a \preceq$ c (transitivity)
- If $a \preceq b$ and $b \preceq a$, then $a = b$ (antisymmetry).

An order is a *total order* if for all $a, b \in S$, $a \preceq b$ or $b \preceq a$. In other words, all elements in S can be compared.

An order in which some elements are incomparable is called a *partial order*. We assume that the path algebra induces all multi-metric orders into total orders.

Future work exploring multiple, multi-metric forwarding paths will involve partial orders [7].

Path Algebra In order to create best, loop-free forwarding paths we must first define what "best" means. The path algebra provides this information. A *path algebra* consists of the set of possible edge weights in the network, the metric or metrics that determines the path weights, and the binary relations (operations) needed to compute and compare paths. The path algebra determines how the best forwarding path through the network is found.

Specifically, a path algebra is a set $(W, \preceq, \oplus, \overline{\infty}, \overline{0})$, where:

- W is the set of all possible edge weights in the network
- $\preceq$ is an order relation we use to compare forwarding paths, for example, **min** when dealing with latency, **max** when dealing with bandwidth or reliability, or $(\mathbf{min}, \mathbf{max})$ when dealing with a multi-criteria path weight. Note that $a \preceq b$ does not necessarily indicate that the weight of path a is "less than" the weight of path b. Rather, $a \preceq b$ indicates that path a is "better" than path b for the path algebra we have defined. We chose the $\preceq$ operator as an indicator of preference.
- The comparison identity $\overline{\infty}$ returns a when compared to forwarding path a. For example, if the path algebra has defined $a \preceq b$ as $\mathbf{min}(a, b) = a$, then $\mathbf{min}(a, \overline{\infty}) = a$. But if $a \preceq b$ is defined as $\mathbf{max}(a, b) = a$, then $\mathbf{max}(a, \overline{\infty}) = a$.
- the operator $\oplus$ defines how edges are combined into forwarding paths, such as $+$ for delay, or **min** for bandwidth.

- The combination identity element $\overline{0}$ returns a when combined with forwarding path a using $\oplus$, i.e. $a \oplus \overline{0} = a$.

Optimal A path algebra is said to be *optimal* if for the best path $p = (v_0, \ldots, v_n)$ between source node v_0 and destination node v_n, all subpaths of p between internal nodes v_i and v_j are also best paths.

3 Examples–How Path Algebras can Fail

We now discuss specific ways in which path algebras can fail, using a number of examples. These examples illustrate the importance of monotonicity and boundedness by showing the consequences of their absence.

Since Euler posed the problem of the Seven Bridges of Königsberg, there has been a wealth of research concerned with finding the single best path through a network. The Dijkstra, Bellman-Ford and lesser-known Floyd-Warshall algorithms were designed to find the best path through a network based on a single metric. Many routing protocols use these algorithms to find the best path in a distributed, hop-by-hop network. Figure 1 illustrates finding the best path from all nodes to node z, where weights are defined by an integer value, and "best" is defined as "shortest." Notice that if we draw the shortest path from each node to destination z, we observe that all paths from intermediate nodes are subpaths of the paths from more distant nodes. Specifically, Path $C = y \rightarrow z$ is a subpath of Path $B = x \rightarrow y \rightarrow z$, which is a subpath of Path $A = w \rightarrow x \rightarrow y \rightarrow z$. Thus, each node on the path calculates the same best path from itself to the destination, and packets are actually forwarded along the same best path computed at

each node (this is not necessarily the case for all path algebras). An example of an optimal path algebra, contrasted with several examples of non-optimal path algebras, will be explained in detail.

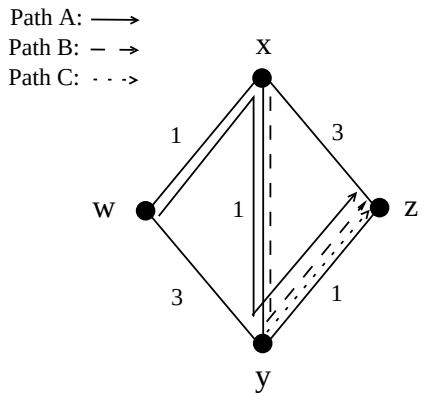

Figure 1: Finding the Shortest Path

The situation becomes complicated if we want to find the best path through the network based on more than one metric. The behavior of a best-path algorithm can change depending on which metrics we use and the order in which we apply them. For example, say we want to find the path that has the least delay, and then of those paths with least delay we want to choose paths with the greatest bandwidth. We call this metric *Widest-Shortest*. To find the Widest-Shortest path to destination t, node i routes packets to the next hop j that has least delay in the routing table for t. If two hops have equal delay, the packet is forwarded along the hop with greatest bandwidth. (Note that bandwidth of a path is determined by the edge with the minimum bandwidth. For example, if a path contains an edge with bandwidth 5 and an edge with bandwidth 10, the bandwidth of the entire path is 5).

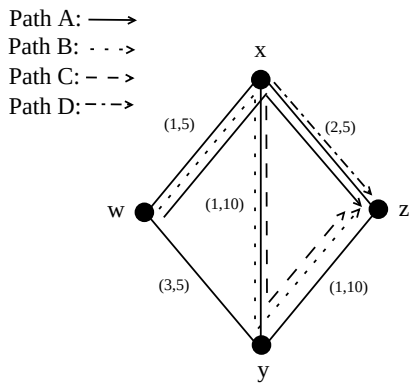

Figure 2: Finding the Widest-Shortest Path

Figure 2 shows an example of finding the Widest-Shortest path from source to destination. Node w sees that there are two best paths from itself to node z: either path A, which goes through x to z for a total cost of $(3, 5)$, or path B, which goes through x to z via node y, also for a total cost of $(3, 5)$. From node x's perspective, however, it sees only one best path to z via node y, path C, since Path $C = x \rightarrow y \rightarrow z$ has equal distance but greater bandwidth than Path $D = x \rightarrow z$. Observe that path A is not optimal because $w \rightarrow x \rightarrow z$ is best but $x \rightarrow z$ is not. Ultimately, packets are forwarded along a path seen as "best" by node w.

The trouble begins when we try to reverse the metrics, i.e. find the Shortest-Widest path. To find the Shortest-Widest path to destination t, node i routes packets to the next hop j that has greatest bandwidth in the routing table for t. If two hops have equal bandwidth, the packet is forwarded along the hop with least delay. Figure 3 shows that Shortest-Widest does not necessarily result in best forwarding paths.

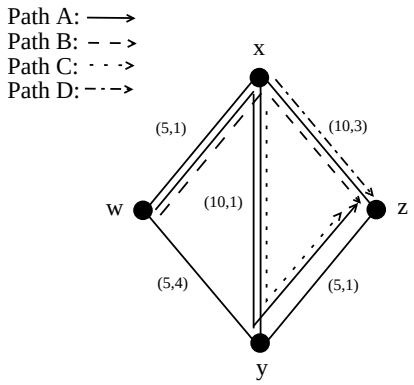

Figure 3: Finding the Shortest-Widest Path

To see this, imagine a packet is traveling from node w to node z. From w's perspective, the best path is path $A = w \rightarrow x \rightarrow y \rightarrow z$, for a distance of $(5, 3)$. w will route packets with destination z to node x, which is the next hop on the path. But from the viewpoint of x, the best path from x to z is not path C via y, with a distance of $(5, 2)$, but directly to z (path D), for a distance of $(10, 3)$. x has no upstream information about where the packets it encounters came from, so there is no way for it to know that the bandwidth of the forwarding path had already been set to 5 by the edge connecting w to x. Therefore, the final forwarding path is path B, which is not best.

Thus in w's routing table, the entry for destination z lists the distance to z as $(5, 3)$ and the next hop as x, but x's routing table has the distance to z as $(10, 3)$ and the next hop as z itself. Therefore, packets from w to z are forwarded over path B for a total distance of $(5, 4)$, instead of the best path A, which has a distance of only $(5, 3)$. We will show in Section 5 why this is the case.

The next example, Widest, illustrates how under certain conditions, a path algebra that finds the best path can still contain loops. Figure 4 illustrates.

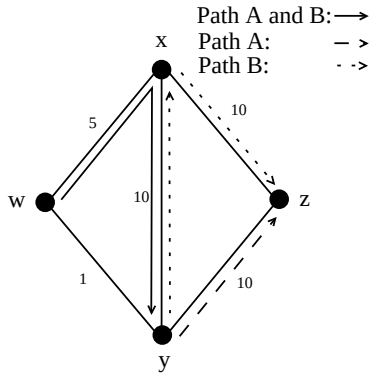

Figure 4: Finding the Widest Path

If the only metric is to forward packets along the widest path, then the path with greatest bandwidth from source to destination will be found. The problem is that there is nothing in the structure of the path algebra to prevent loops. Specifically, a loop is caused when the path from an intermediate node can intersect a path from a downstream node at a point *upstream* from the intermediate node. As Figure 4 shows, say that there are packets at node w to be forwarded to z. w forwards the packets to x, which is the next hop in its routing table for destination z. The path from x directly to z has a bandwidth of 10. But, the path from x to z via y also has a bandwidth of 10. Therefore, x is equally likely to forward packets to y as to forward packets directly to z. Say x forwards packets to y. When the packets arrive at y, there are two equally good paths to z: Path A, which goes directly from y to z, and Path B, which goes back to x and then on to z. From y's point of view, Paths A and B are equally desirable, since they both have a bandwidth of 10. In this way, packets can be forwarded back and forth from x to y and back to x indefinitely, creating a loop. (Note that this problem is different from a flow problem. Rather than the residual capacity of an edge determining the path a packet takes, the path is determined by the routing table

of the node).

The loop cannot be prevented unless there is some cost to forwarding packets repeatedly, which Widest alone does not impose. We will see later that the property of *strict boundedness* provides that cost, thus preventing loops.

The final example, Slope, produces forwarding paths that are neither best nor loop-free. The Slope path algebra weight contains two components–a distance value and a cost value (d, c). Routes are calculated by adding the distance and cost values, respectively. In other words, to add an edge with weight (d_i, c_i) to a subpath with weight (d_j, c_j), the resulting subpath has weight $(d_i + d_j, c_i + c_j)$. Paths are selected by comparing the quotient of cost/distance and selecting the path with the smallest value. Figure 5 illustrates what can happen.

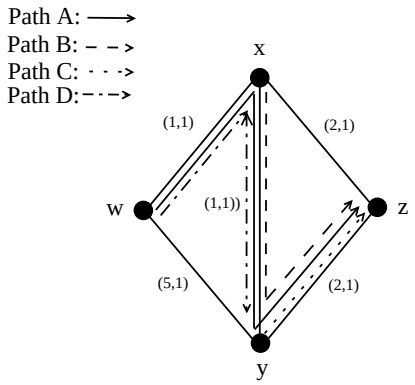

Figure 5: Slope

In the example, Path $A = w \rightarrow x \rightarrow y \rightarrow z = \frac{1+1+2}{1+1+1} = \frac{4}{3}$ is better than subpath $B = x \rightarrow y \rightarrow z = \frac{1+2}{1+1} = \frac{3}{2}$, which is better than subpath $C = y \rightarrow z = \frac{2}{1} = 2$. If distance increases more than cost as each edge of the path is added, then the longer path (in terms of hops) is better than a shorter path. This can cause loops, as path D illustrates.

Say that w forwards packets destined for z to x for a distance/cost of $\frac{4}{3}$. x calculates that forwarding the packets on to y for a distance/cost of $\frac{3}{2}$ is better than forwarding them directly to z, which has a distance/cost value of $\frac{2}{1} = 2$. But when the packets reach y, y calculates that forwarding them back to x (to be forwarded on to z from x), results in a distance/cost of $\frac{1+2}{1+1} = \frac{3}{2}$, which is better than forwarding them directly to z, which from y's perspective has a distance/cost value of $\frac{2}{1} = 2$. In turn, upon receiving the packets destined for z from y, x calculates (again) that the better path to z is via y, and sends the packets back to y, and so on. This results in a loop between x and y.

As these examples show, it is not immediately obvious whether or not a path algebra will in fact produce best, loop-free forwarding paths. In the next section we define the properties which are required in order to guarantee that forwarding paths are best and loop-free.

4 Properties to Guarantee Best, Loop-Free Forwarding Paths

Boundedness and *monotonicity* are the fundamental properties that guarantee loop-freedom and best-path. These properties were originally articulated in [6] and [11].

Monotonic We say that a path algebra is *monotonic* [6] if given a path algebra

$(W, \oplus, \preceq, \overline{0}, \overline{\infty})$ for $a, b \in W, a \neq b$ the following relation is true:

$$a \preceq b \Rightarrow a \oplus x \preceq b \oplus x$$

Strictly Monotonic We say that a path algebra is *strictly monotonic* if:

$$a \prec b \Rightarrow a \oplus x \prec b \oplus x$$

Bounded A path algebra is *bounded* when for all $a, b \neq \overline{0} \in W$:

$$a \preceq a \oplus b$$

.

Whereas boundedness is certainly possible in a path algebra, we will see in the next section that it is not a very useful property when it comes to determining if a path algebra guarantees best, loop-free forwarding paths. The more important definition is that of *strict boundedness*:

Strictly Bounded A path algebra is *strictly bounded* when

$$a \prec a \oplus b$$

.

5 Requirements for Best, Loop-Free Forwarding Paths

We show that in order to prove that a path algebra produces best, loop-free paths, we need only prove the existence of certain algebraic properties, i.e.:

1. Traffic is forwarded over loop-free paths if and only if the path algebra is strictly bounded.
2. Traffic is forwarded over best paths if and only if the path algebra is monotonic.
3. Packets will be routed over best, loop-free forwarding paths if and only if the path algebra is monotonic and strictly bounded.
4. Packets will be routed over optimal forwarding paths if and only if the path algebra is optimal.

5.1 Path Algebra Loop-Free If and Only If Strictly Bounded

If a path algebra is monotonic (not necessarily strictly) but not strictly bounded, then traffic may be fowarded over paths that contain loops. Intuitively, a loop occurs when adding a hop seems to make the path better. A simple example is when paths have negative edges in the Shortest path algebra. Repeatedly adding a negative edge decreases the path length indefinitely, resulting in a loop.

To ensure that a path algebra cannot cause loops, we must prove that the path algebra is strictly bounded. We prove that a path algebra is strictly bounded if

and only if it is loop-free. A version of this proof was originally given in [4].

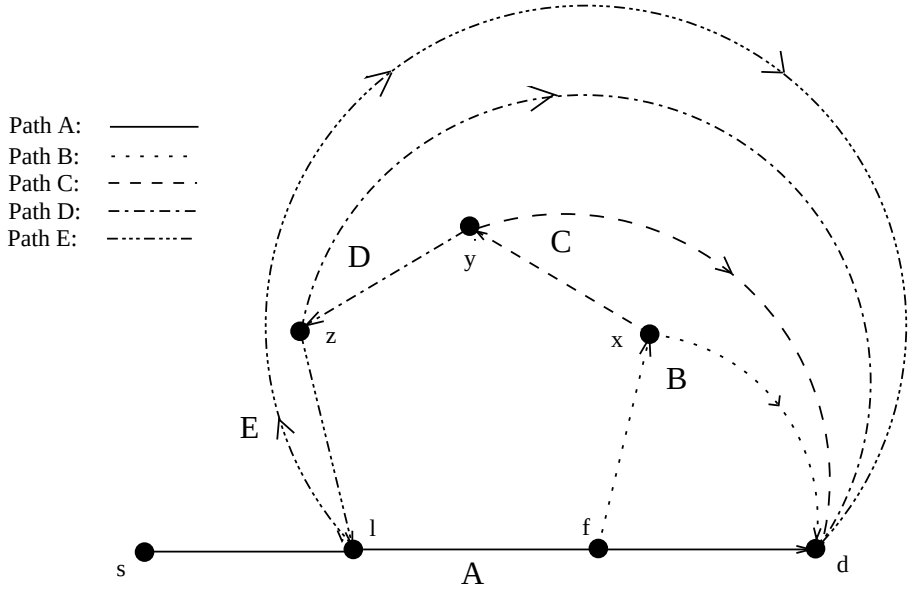

Figure 6: Strictly bounded guarantees loop-free.

Lemma 5.1. *If a path algebra is strictly bounded, then it guarantees loop-free forwarding paths.*

Proof. Assume, for the sake of contradiction, that a path algebra $(W, \preceq, \oplus, \overline{\infty}, \overline{0})$ is strictly bounded, but there are best paths computed at different nodes that result in forwarding paths that loop. Figure 6 shows such a network. The example contains five nodes as an illustration, but note that the concept would hold for any number of nodes > 1, so there is no loss of generality. Each dotted line segment represents a subpath containing 0 or more nodes, with no branches occurring in the subpath (i.e. forwarding tables in the subpath are consistent with the path computed by the node at the start of the subpath, so any nodes in the subpath can safely be ignored).

Nodes l and f are intermediate nodes in the path between source s and des-

tination d. The weight of path $w(f, \ldots, d)$ is called A, the weight of path $w(f, \ldots, x, \ldots, d)$ is called B, and so on through paths C, D and E.

At certain nodes (for example at node f), the path "forks," i.e. the router forwards traffic to a next hop that is different to the path computed at the source. At each fork, the alternate path (for example, path B taken by node f), is no worse than the subpath of the original path that had been computed at the node upstream of the fork (for example $B \preceq A_{f,d}$, where $A_{f,d}$ is the weight of the subpath of path A between nodes f and d).

Recalling that we assumed a strictly bounded path algebra, we trace the forwarding paths at each node. Beginning at source s, packets are forwarded to node l, which forwards the packets on to node f along path A. A fork is encountered at f. The routing table at f calculates path B to be no worse than subpath $A_{f,d}$ ($B \preceq A_{f,d}$), and so f forwards packets on to node x.

x then calculates path C to be no worse than the subpath $B_{x,d}$ ($C \preceq B_{x,d}$). Since the path algebra is strictly bounded, we know

$$B_{x,d} \prec B_{f,x} \oplus B_{x,d} = B$$

therefore, we have

$$C \preceq B_{x,d} \prec B = w(f, \ldots, x) \oplus B_{x,d}$$

and packets are forwarded on to node y. At y we have a similar situation:

$$D \preceq C_{y,d} \prec C = w(x, \ldots, y) \oplus C_{y,d}$$

and packets are forwarded on to node z, where

$$E \preceq D_{z,d} \prec D = w(y, \ldots, z) \oplus D_{z,d}$$

and packets are forwarded on to node l. But at l, path A looks better than path E and we have:

$$A_{l,d} \preceq E_{l,d} \prec E = w(z, \ldots, l) \oplus E_{l,d}$$

Therefore, by transitivity we have:

$$A_{l,d} \prec E \prec D \prec C \prec B \prec A_{f,d}$$

This means:

$$A_{l,d} = A_{l,f} \oplus A_{f,d} \prec A_{f,d}$$

which contradicts our assumption of a strictly bounded path algebra. Thus, if a path algebra is strictly bounded it must be loop-free. □

Lemma 5.2. *If a path algebra produces forwarding paths that are guaranteed loop-free, then the path algebra is strictly bounded.*

Proof. The proof is by contradiction. Assume we have a path algebra $(W, \preceq, \oplus, \overline{\infty}, \overline{0})$ that is not strictly bounded, but produces guaranteed loop-free forwarding paths. Since the path algebra is not strictly bounded, there can exist an $a, b \in W$, where

$$a \not\prec a \oplus b \Rightarrow a \succeq a \oplus b$$

We can construct a graph as shown in Figure 7, where there are paths between nodes x, y and z (as in Lemma 5.1, the dotted lines represent paths that are consistent with the paths computed by the node at the source, so we can safely ignore the details). Some of the paths are of length a and some are of length b, as shown.

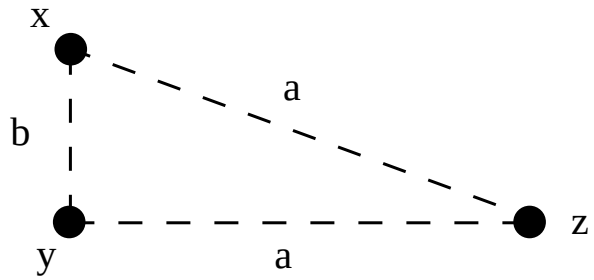

Figure 7: Loop-free guarantees strictly bounded.

Say that x is the source and z is the destination. Because $a \succeq a \oplus b$, packets from x may be forwarded to z via y along path $b \oplus a$, rather than directly to z from x along path a. Since $a \succeq a \oplus b$, there are two possibilities: $a \oplus b \prec a$ or $a = a \oplus b$.

1. If $a + b \prec a$, then x will calculate that the best path to z is via y and will forward packets to y. But when the packets reach y, node y will calculate

that the best path to z is $b \oplus a$ via x. y will forward packets back to x, inducing a loop.

2. If $a = a \oplus b$, then the situation is similar, except that packets are equally likely to be forwarded directly from x to z along path a as they are to be forwarded from x to z via y, along path $a \oplus b$. If packets are forwarded from x to y, the path back to x and then on to z looks equally attractive from y's perspective than forwarding directly from y to z. Therefore, depending on the vagaries of the implementation of the shortest path algebra, the computations could result in a loop. An example of this path algebra is Widest.To see this, say we have path lengths $a = 5$ and $b = 10$. In this case, $a = a \oplus b$, i.e. $5 = \mathbf{min}(5, 10)$.

We have shown that if a path algebra is not strictly bounded, it is not necessarily loop-free. Therefore, if a path algebra is guaranteed loop-free, then it must be strictly bounded. □

Theorem 5.3. *A path algebra is loop-free if and only if it is strictly bounded.*

Proof. Lemma 5.1 proves that if a path algebra is strictly bounded, then it is loop-free. Lemma 5.2 proves that if a path algebra is loop-free, then it is strictly bounded. □

5.2 Best Paths and Monotonicity

We have shown that strict boundedness is the key property to guarantee loop-free forwarding paths. We now show that *monotonicity* is the defining property of a

path algebra that guarantees *best* forwarding paths.

Lemma 5.4. *If a path algebra is monotonic, then traffic will be forwarded over best forwarding paths in a hop-by-hop network.*

Proof. For the sake of contradiction, assume that the path algebra is monotonic and every node forwards traffic over the best next hop, but that this does not result in traffic being forwarded on best forwarding paths. This implies that we can construct a topology such as that given in Figure 8, where Path (A, N, B) is the best path, but traffic is forwarded over Path (A, S, B). Assume that traffic is forwarded over path A to a node f, where there are two possible paths. Since the path algebra is monotonic, the fact that f forwards packets via (S, B) implies that weight $w(S, B) \prec w(N, B)$. Then, by monotonicity $w(A) \oplus w(S, B) \preceq w(A) \oplus w(N, B)$. But this contradicts the assumption that (A, N, B) is best, which would imply that $w(A) \oplus w(N, B) \prec w(A) \oplus w(S, B)$. Therefore, if a path algebra is monotonic, then traffic is forwarded over best forwarding paths. □

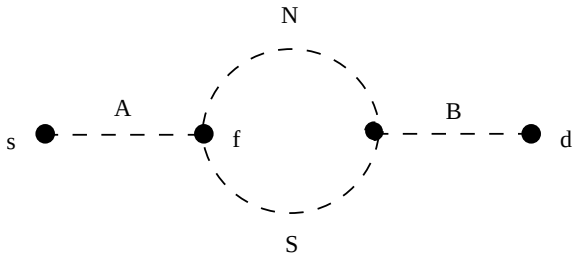

Figure 8: Monotonicity and best forwarding paths.

Lemma 5.5. *If a path algebra always results in best forwarding paths, then it is monotonic.*

Proof. By contradiction. Assume that the use of the path algebra always results

in best forwarding paths, but the path algebra is not monotonic. Then there must exist three weights $a, b, c \in W$ where $a \preceq b$, but $a \oplus c \succ b \oplus c$. Using Figure 8 again, let $a := w(N, B), b := w(S, B), c := w(A)$ and since the path algebra is not monotonic, let $a \prec b$, but $a \oplus c \succ b \oplus c$. (For an example see *Shortest-Widest* in Section 3).

We see that path (A, S, B) is best. But when packets are forwarded to f, since $a \prec b$, it sees path (N, B) as best, and forwards packets along (N, B) (i.e. $b \oplus c \prec a \oplus c$, therefore $w(A, N, B) \prec w(A, S, B)$). This contradicts the assumption that the path algebra always results in best forwarding paths. Thus if packets are forwarded over best forwarding paths, the path algebra must be monotonic. □

Theorem 5.6. *Traffic is forwarded over best forwarding paths if and only if the path algebra is monotonic.*

Proof. By Lemma 5.4 we have that if a monotonic path algebra is used in a destination-based, hop-by-hop routing computation, then the set of routes computed at each node in the network is guaranteed to result in best forwarding paths. By Lemma 5.5 we have that if the path algebra used in a destination-based, hop-by-hop routing computation always results in best forwarding paths, then the path algebra must be montonotonic. Thus we prove that the relationship between best paths and monotonicity is bijective. □

5.3 Optimal Path Algebras and Strict Monotonicity

As we have shown in Section 3, for a given best path computation, where the entire forwarding path is best, the subpaths between internal nodes are not nec-

essarily also best paths. The Widest-Shortest path algebra is an example of this phenomenon.

Generally, a problem whose solution is made up of solutions to smaller subproblems is called *Markovian* [8]. We define this property as *optimal*. We show that it is a special case of best path problems, and that strict monotonicity is the defining property of an optimal path algebra. First, we show that strict monotonicity is actually a special case of strict boundedness.

Note: In future work we will show that the Dijkstra and Bellman-Ford algorithms each perform well for some non-Markovian, non-optimal path algebras but fail with others.

Recall the definition for optimal:

Optimal A path algebra is said to be *optimal* if for the best forwarding path $p = (v_0, \ldots, v_n)$ between source node v_0 and destination node v_n, all subpaths of p between internal nodes v_i and v_j are also best forwarding paths.

Theorem 5.7. *A path algebra is strictly monotonic if and only if it results in optimal forwarding paths.*

Proof. Assume for sake of contradiction that the path algebra is strictly monotonic but not optimal. Using Figure 8, to illustrate, given that the path algebra is not optimal, we can construct a case where both paths from s to d are best paths, i.e., $w(A, N, B) = w(A, S, B)$, but only subpath $w(S, B)$ is best from node f, so that $w(S, B) \prec w(N, B)$. Since we assume that the path algebra is strictly monotonic, it must also be true that $w(A) \oplus w(S, B) \prec w(A) \oplus w(N, B)$, and

therefore $w(A, S, B) \prec w(A, N, B)$. But this contradicts the assumption that $w(A, N, B) = w(A, S, B)$. Therefore if the path algebra is strictly monotonic, it must result in optimal forwarding paths.

The reverse is also true. Assume, for sake of contradiction, that a path algebra provides optimal forwarding paths, but is not strictly monotonic. Since it is not strictly monotonic, there exist $a, b, c \in W$ where $a \prec b$ but $a \oplus c \succeq b \oplus c$. Again using Figure 8, let $a := w(N, B), b := w(S, B)$ and $c := w(A)$. Since we assumed that the path algebra is optimal and $a \prec b$, then $w(N, B) \prec w(S, B)$ implies that $w(A, N, B) \prec w(A, S, B)$. But as we assumed that $a \oplus c \succeq b \oplus c$, then $w(A, N, B) \succeq w(A, S, B)$, contradicting our assumption of optimality. Therefore, if a path algebra provides optimal forwarding paths, it must be strictly monotonic.

We proved that if a path algebra is strictly monotonic then traffic is guaranteed to follow optimal forwarding paths. We also proved that if a given path algebra always results in optimal forwarding paths, it must be monotonic. Therefore, a path algebra is strictly monotonic if and only if it results in optimal forwarding paths. □

5.4 Relationship between Boundedness and Monotonicity

The relationship between boundedness and monotonicity is subtle. Although they are closely related, they are not the same property. We have shown that if a path algebra is monotonic (but not strictly), it must also be strictly bounded for best, loop-free forwarding paths to be guaranteed. We will now show that

if a path algebra is strictly monotonic, then it is also strictly bounded. But a strictly bounded path algebra does not guarantee monotonicity. For example, Shortest-Widest is strictly bounded, but not monotonic.

Intuitively, when a path algebra is strictly monotonic, then if we extend two different paths with the same edge, the ordering of the two new paths must stay the same as those of the original paths. But if a path algebra is merely monotonic, then extending two different paths with the same edge could result in two new paths with equal ordering. In this case, the property of strict boundedness is necessary to ensure that packets are forwarded on best, loop-free forwarding paths, as the property of monotonicity is not strong enough to ensure this on its own.

If a path algebra is bounded (but not strictly), then if we extend a path with an edge, the weight of the extended path will not be better than the original weight, but might be equal to the weight of the original path. But if the path algebra is *strictly bounded*, then a path weight always gets *worse* as it is extended by an edge. Thus, strict boundedness on its own is enough to ensure loop-freeness. This is because paths that always get worse as they are extended will never double back on themselves. However, strict boundedness alone cannot guarantee that adding an edge to two unequal paths will not reverse the order of the new paths. This is why monotonicity is also needed.

In summary:

- If a path algebra is monotonic but not strictly monotonic, then it needs the additional property of strict boundedness to guarantee best paths.

- A path algebra can be monotonic without being strictly bounded
- A path algebra can be strictly bounded without being monotonic (for example, Shortest Widest).
- If a path algebra is strictly monotonic, it is also strictly bounded.

We now prove the previous assertion:

Theorem 5.8. *If a path algebra is strictly monotonic, then it is also strictly bounded.*

Proof. Strict boundedness is a special case of strict monotonicity. Recall that the definition of strictly bounded is: $b \neq 0, a \prec a \oplus b$. Let $a = 0$. Then since $b \neq 0$ and the path algebra is strictly monotonic, we have:

$$0 \prec b \Rightarrow a \oplus 0 \prec a \oplus b$$
$$\Rightarrow a \prec a \oplus b$$

□

Note that the relationship is *not* bijective. A path algebra that is strictly bounded is *not* necessarily strictly monotonic. For example, Widest-Shortest is strictly bounded and monotonic (but not strictly). Shortest-Widest is strictly bounded, but not monotonic at all.

6 Review of Examples–Properties of Loop-Free and Best

Once the properties of the path algebra are understood, we can predict the behavior of forwarding paths based on the path algebra before ever applying a routing algorithm. To better illustrate this concept, we review the examples introduced in Section 3 on page 8.

6.1 Shortest

The Shortest path algebra has been conventionally implied when describing "shortest" path problems. Edge weights are the positive real numbers, and "best", or $\preceq$, is defined as "less than": $\leq$. To determine the length of a path, edge weights are combined with the "+" operator and paths with minimum length are considered "best." Formally, the path algebra is written:

$$(W, \preceq, \oplus, \overline{\infty}, \overline{0}) := (\mathbb{R}^+, \mathbf{min}, +, \infty, 0)$$

Theorem 6.1. *Shortest is strictly monotonic.*

Proof. Shortest is defined on the real numbers, with + as the addition operator and $<$ as the binary relation. By the ordering of the positive real numbers,

$$a < b \Leftrightarrow a + c < b + c$$

$$a = b \Leftrightarrow a + c = b + c$$

□

More detailed, algebraic proofs are provided in [5] and [2].

The algebraic structure is strictly monotonic and by Theorem 5.8 strictly bounded. Thus Shortest is also optimal, in that subpaths of the best forwarding path are also best paths.

6.2 Widest-Shortest

Widest-Shortest is the first multiple-metric example we study. Although it may appear to be as complicated as the other examples, it is actually both monotonic and strictly bounded. Widest-Shortest is defined by the following path algebra $(W, \preceq, \oplus, \overline{\infty}, \overline{0})$:

First, the set W is defined as:

$$W = (d, b) \in (\mathbb{R}^+ \times \mathbb{R}^+)$$

In this equation b represents bandwidth and d represents delay.

The rule for comparison ($\preceq$) is as follows:

$$(d_1, b_1) \prec (d_2, b_2) \Rightarrow \begin{cases} & \text{if } d_1 < d_2 \\ & \text{if } (d_1 = d_2 \text{ and } b_1 > b_2) \end{cases}$$

$$(d_1, b_1) \succeq (d_2, b_2) \text{ otherwise}$$

The rule for combining edges ($\oplus$) is:

$$(d_1, b_1) \oplus (d_2, b_2) = (d_1 + d_2,\ \min(b_1, b_2))$$

The comparator identity is $\overline{\infty} = (\infty, 0)$ and the additive identity is $\overline{0} = (0, \infty)$.

Widest-Shortest is monotonic, but not strictly. For example,

$$(2, 10) \prec (2, 5),\ \text{but}\ (2, 10) \oplus (1, 5) = (2, 5) \oplus (1, 5) = (3, 5).$$

However, Widest-Shortest is strictly bounded, so best, loop-free forwarding paths will still be found. Widest-Shortest differs from an optimal path algebra such as Shortest in that forwarding paths may be calculated differently at different nodes. For example, in Figure 2, Path A ($w \to x \to z$) is a best path, but the subpath $x \to z$ is not, since subpath $x \to y \to z$ is better from the perspective of node x. Therefore, node w will calculate the best forwarding path as Path $A : w \to x \to z$, but when packets arrive at node x, x will forward packets via path $x \to y \to z$ (Path B). Regardless, the total weight of the forwarding path from w to z is still best at $(3, 5)$.

We now prove that Widest-Shortest is strictly bounded and monotonic, and will therefore always produce forwarding paths that are guaranteed best and loop-free.

Theorem 6.2. *Widest-Shortest is strictly bounded.*

Proof. Recall the definition of strictly bounded: for $b \neq 0, a \prec a + b$. By the definition of Widest-Shortest, this translates to: $(d_a, b_a) \prec (d_a, b_a) \oplus (d_b, b_b)$.

Given nonzero weights a, b of two paths, there are two possible ways that path a can be better than path $a \oplus b$: either $d_a < d_a + d_b$, or $d_a = d_a + d_b$ and $b_a < b_b$. But since we assumed that $b \neq 0$, then it is always true that $d_a < d_a + d_b$, as in Shortest. Therefore, Widest-Shortest is strictly bounded. □

Theorem 6.3. *Widest-Shortest is monotonic.*

Proof. Recall the definition of monotonic:

$$a \preceq b \Rightarrow a \oplus c \preceq b \oplus c$$

In the Widest-Shortest path algebra this translates to:

$$(d_a, b_a) \preceq (d_b, b_b) \Rightarrow (d_a, b_a) \oplus (d_c, b_c) \preceq (d_b, b_b) \oplus (d_c, b_c)$$

In other words, we check for delay, and if the two delays are equal, we check for bandwidth:

$$\begin{aligned} & d_a < d_b \\ \text{or } & d_a = d_b \text{ and } \min(b_a, b_b) = b_b \\ \Rightarrow & (d_a + d_c, \min(b_a, b_c)) \preceq (d_b + d_c, \min(b_b, b_c)) \end{aligned}$$

We can see that there are a number of possibilities on both the left and right sides of the equation that must be explored.

Beginning with the left side of the equation, there are two possibilities for comparing the delay of a and b: either $d_a < d_b$ or $d_a = d_b$.

If $d_a < d_b$, then from the proof that Shortest is monotonic we know that $d_a < d_b \Rightarrow d_a + d_c < d_b + d_c$. Therefore in this case Widest-Shortest is also monotonic.

On the other hand, if $d_a = d_b$, then $d_a + d_c = d_b + d_c$, also by the proof of Shortest. Therefore, we must compare bandwidths and prove monotonicity for all possible cases.

Specifically, we must show that, given $\min(b_a, b_b) = b_b$, in all cases $\min(b_a, b_c) \geq \min(b_b, b_c)$. There are three possible cases to investigate:

1. $b_a \geq b_b \geq b_c$
2. $b_a \geq b_c \geq b_b$
3. $b_c \geq b_a \geq b_b$

We show that $b_a \geq b_b \Rightarrow \min(b_a, b_c) \geq \min(b_b, b_c)$ for all three cases.

- Case 1: $b_a \geq b_b \geq b_c$

  We must show that $\min(b_a, b_c) \geq \min(b_b, b_c)$. By assumption, both b_a and b_b are better ($\geq$), than b_c, so we have

  $$\min(b_a, b_c) = b_c \geq \min(b_b, b_c) = b_c$$
  $$\Rightarrow b_c \geq b_c$$

  Case 1 is true.

- Case 2: $b_a \geq b_c \geq b_b$

  We want to show that $\min(b_a, b_c) \geq \min(b_b, b_c)$.

$$\min(b_a, b_c) = b_c \geq \min(b_b, b_c) = b_b$$
$$\Rightarrow b_c \geq b_b$$

Case 2 is true.

- Case 3: $b_c \geq b_a \geq b_b$

  We want to show that $\min(b_a, b_c) \geq \min(b_b, b_c)$.

$$\min(b_a, b_c) = b_a \geq \min(b_b, b_c) = b_b$$
$$\Rightarrow b_a \geq b_b$$

  Case 3 is true.

$b_a \geq b_b \Rightarrow \min(b_a, b_c) \geq \min(b_b, b_c)$ is true in all possible cases, therefore, Widest-Shortest is monotonic. □

6.3 Shortest-Widest

Shortest-Widest has the same metrics as Widest-Shortest, but the order in which the metrics are evaluated in the path algebra is switched. This produces a different algebraic structure, which in turn produces different forwarding-path behavior.

The path algebra $(W, \preceq, \oplus, \overline{\infty}, \overline{0})$ for Shortest-Widest is defined as follows:

W is defined as:

$$W = (b, d) \in (\mathbb{R}^+ \times \mathbb{R}^+)$$

As in Widest-Shortest, b represents bandwidth and d represents delay.

The rule for comparison ($\preceq$) is as follows:

$$(b_1, d_1) \prec (b_2, d_2) \Rightarrow \begin{cases} & \text{if } b_1 > b_2 \\ & \text{if } (b_1 = b_2 \text{ and } d_1 < d_2) \end{cases}$$
$$(b_1, d_1) \succeq (b_2, d_2) \text{ otherwise}$$

The rule for combining edge weights is:

$$(b_1, d_1) \oplus (b_2, d_2) = (\mathbf{min}(b_1, b_2), d_1 + d_2)$$

The comparator identity is $\overline{\infty} = (0, \infty)$ and the additive identity is $\overline{0} = (\infty, 0)$.

Shortest-Widest is not monotonic, for example,

$$(10, 3) \prec (5, 1), \text{ but } (10, 3) \oplus (5, 1) = (5, 4) \succ (5, 1) \oplus (5, 1) = (5, 2).$$

However, we prove that Shortest-Widest is strictly bounded.

Theorem 6.4. *Shortest-Widest is strictly bounded.*

Proof. Recall that the definition of strictly bounded is $a \prec a \oplus b$. By the definition of Shortest-Widest, this translates to:

$$(b_a, d_a) \prec (b_a, d_a) \oplus (b_b, d_b)$$

By the definition of Shortest-Widest, for $(b_a, d_a) \prec (b_a, d_a) \oplus (b_b, d_b)$ to be true,

one of the following relations must be true:

$$(b_a, d_a) \prec (\min(b_a, b_b), d_a + d_b) \begin{cases} & \text{if } b_a > \min(b_a, b_b) \text{ or} \\ & \text{if } b_a = \min(b_a, b_b) \text{ and } d_a < d_a + d_b \end{cases}$$

Specifically, we must prove that the following Shortest-Widest precedence relation is true in all possible cases:

$$(b_a, d_a) \overset{?}{\prec} (\min(b_a, b_b), d_a + d_b)$$

We must first compare b_a to b_b. There are three possible cases:

1. $b_a > b_b$
2. $b_a < b_b$
3. $b_a = b_b$

We must show that $(b_a, d_a) \prec (\min(b_a, b_b), d_a + d_b)$ for all three cases.

- Case 1: $b_a > b_b$

  In this case, $\min(b_a, b_b) = b_b$, therefore $b_a > \min(b_a, b_b)$, which implies $a \prec a \oplus b$. Case 1 is true.

- Case 2: $b_a < b_b$

In this case, $\min(b_a, b_b) = b_a$, so we must verify that $d_a < d_a + d_b$. But this is the same case is Shortest, and so is always true. Therefore. Case 2 is true.

- Case 3: $b_a = b_b$

  This case is the same as Case 2, and so is true.

We have shown that for Shortest-Widest, $a \prec a \oplus b$ in all possible cases, therefore, Shortest-Widest is strictly bounded.

□

Although Shortest-Widest does not always produce the best forwarding path, if a path choice is made downstream of an edge with limiting (meaning lower) bandwidth, packets will be routed on the better path. If the limiting edge is upstream of a path choice, packets will not be routed on the better path.

Figure 9 and Figure 10 show two examples of Shortest-Widest. In the first example, for a packet traveling from x through y to z, node y will route the packet on the path with weight $(10, 2)$. This is actually the worse path because of the upstream limiting path between x and y, which has weight $(5, 1)$, But, y has no information about the $(5, 1)$ upstream path.

In the second example node x will route packets on the correct southern path with weight $(5, 1)$, because its routing table has information about the downstream path from y to z with weight $(5, 1)$.

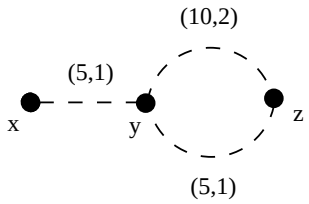

Figure 9: Packets will be routed on the worse path

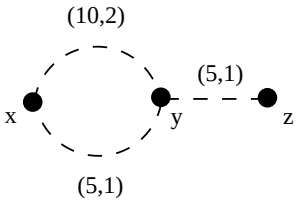

Figure 10: Packets will be routed on the better path

6.4 Widest

The Widest path algebra is defined by:

$$(W, \preceq, \oplus, \overline{\infty}, \overline{0}) := ((\mathbb{R}^+ \cup \infty^+), \mathbf{max}, \mathbf{min}, 0, \infty)$$

In other words, the values in W comprise the positive real numbers, which represent capacity, such as bandwidth. Two path weights are compared by taking the maximum value as best. Two paths a and b are concatenated, or combined, by setting the value of $a \oplus b = \mathbf{min}(a, b)$. The comparator identity $\overline{\infty} = \infty$, therefore so $a \preceq \infty$ for all a. The additive identity is $\overline{0} = 0$, so $a \oplus 0 = a$.

Although Widest is bounded, it is not strictly bounded. In addition it is not strictly monotonic. For example, $5 \succ 10$, but $5 \oplus 5 = 5 \oplus 10$.

Thus, Widest produces the best forwarding path, for example it will always find the path with the greatest bandwidth, but that path will not be guaranteed

loop-free.

Theorem 6.5. *Widest is bounded, but not strictly bounded.*

Proof. Recall the definition of bounded: for $b \neq 0, a \preceq a + b$. By the definition of Widest, this translates to: $b_a \preceq b_a \oplus b_b \Rightarrow b_a \geq \min(b_a, b_b)$.

There are three cases: 1) Either $b_a > b_b$, 2) $b_a < b_b$ or 3) $b_a = b_b$.

1. If $b_a > b_b$, then $\min(b_a, b_b) = b_b$, and so $b_a \geq \min(b_a, b_b)$. Case 1 is true.
2. If $b_a < b_b$, then $\min(b_a, b_b) = b_a$ and so $b_a \geq \min(b_a, b_b) = b_a$. Case 2 is true.
3. If $b_a = b_b$ then $\min(b_a, b_b) = b_a = b_b$ and so $b_a \geq \min(b_a, b_b) = b_a = b_b$. Case 3 is true.

For all possible cases, $b_a \geq \min(b_a, b_b)$, therefore, Widest is bounded, but not strictly bounded (due to Case 2). □

Theorem 6.6. *Widest is monotonic.*

Proof. Recall the definition of monotonic:

$$a \preceq b \Rightarrow a \oplus c \preceq b \oplus c$$

For the Widest path algebra to be monotonic, we must prove that in the following statement is always true:

$$b_a \geq b_b \Rightarrow \min(b_a, b_c) \geq \min(b_b, b_c)$$

As in Widest-Shortest, given $b_a \geq b_b$, there are three possible cases:

1. $b_a \geq b_b \geq b_c$
2. $b_a \geq b_c \geq b_b$
3. $b_c \geq b_a \geq b_b$

We show that $b_a \geq b_b \Rightarrow \min(b_a, b_c) \geq \min(b_b, b_c)$ for all three cases.

- Case 1: $b_a \geq b_b \geq b_c$

  We must show that $\min(b_a, b_c) \geq \min(b_b, b_c)$. By assumption, both b_a and b_b are better ($\geq$), than b_c, so we have

  $$\min(b_a, b_c) = b_c \geq \min(b_b, b_c) = b_c$$
  $$\Rightarrow b_c \geq b_c$$

  Case 1 is true.

- Case 2: $b_a \geq b_c \geq b_b$

  We want to show that $\min(b_a, b_c) \geq \min(b_b, b_c)$.

  $$\min(b_a, b_c) = b_c \geq \min(b_b, b_c) = b_b$$
  $$\Rightarrow b_c \geq b_b$$

  Case 2 is true.

- Case 3: $b_c \geq b_a \geq b_b$

  We want to show that $\min(b_a, b_c) \geq \min(b_b, b_c)$.

$$\min(b_a, b_c) = b_a \geq \min(b_b, b_c) = b_b$$
$$\Rightarrow b_a \geq b_b$$

  Case 3 is true.

$b_a \geq b_b \Rightarrow \min(b_a, b_c) \geq \min(b_b, b_c)$ is true in all possible cases, therefore, Widest is monotonic. □

6.5 Slope

At first glance, Slope appear to be a perfectly reasonable metric. When its algebraic structure is examined, however, it is actually quite deviant. Slope is neither monotonic nor strictly bounded, and so the forwarding paths created by Slope are neither best nor loop-free. The path algebra is defined below, where c represents a postive real number cost and d represents positive real number delay.

$$W = (c, d) \in (\mathbb{R}_+ \cup \infty) \times (\mathbb{R}_+ \cup \infty)$$

The comparator identity is $\overline{\infty} = (\infty, \infty)$ and the additive identity is $\overline{0} = (0, 0)$. The comparator rule is:

$$(c_1, d_1) \prec (c_2, d_2) \Rightarrow \mathbf{min}(c_1/d_1, c_2/d_2) = c_1/d_1$$
$$(c_1, d_2) \succeq (c_2, d_2) \qquad \text{otherwise}$$

The rule for combining edge weights is:

$$(c_1, d_1) \oplus (c_2, d_2) = (c_1 + c_2, d_1 + d_2)$$

Slope is neither monotonic nor bounded, therefore, the Slope path algebra will not find best or loop-free paths. To see that Slope is not strictly bounded, recall the definition of addition for Slope:

$$(c_1, d_1) \oplus (c_2, d_2) = (c_1 + c_2, d_1 + d_2)$$

Thus, if Slope were strictly bounded, then the following relation would always be true:

$$a \prec a \oplus b \Leftrightarrow \frac{c_a}{d_a} < \frac{c_a + c_b}{d_a + d_b}$$

As a counterexample, let path $a = (100, 100)$ and path $b = (1, 4)$. Then $a \oplus b = \frac{100+1}{100+4} = \frac{101}{104}$. Clearly $a = \frac{100}{100} > a \oplus b = \frac{101}{104}$. Therefore Slope is not only not strictly bounded, it is not bounded at all.

To see that Slope is not monotonic, again let path $a = (100, 100)$ path $b = (2, 1)$ and path $c = (1, 4)$. If Slope were monotonic, the following relation would be

true:

$$a \preceq b \Leftrightarrow a \oplus c \preceq b \oplus c$$

$$\frac{c_a}{d_a} \leq \frac{c_b}{d_b} \Leftrightarrow \frac{c_a + c_c}{d_a + d_c} \leq \frac{c_b + c_c}{d_b + d_c}$$

But, in our example, $a \oplus c = \frac{100+1}{104} \approx 0.97$ and $b \oplus c = \frac{2+1}{1+4} = 0.6$. So $a \prec b$ but $b \oplus c \prec a \oplus c$, and therefore Slope is not monotonic.

7 Related Work

Previous work on this topic ([11] [1] [6] [2]) is not extensive, and addresses limited aspects of the problem. In addition, different definitions and notation are used for similar concepts.

For example, what Sobrinho calls "isotonicity," we refer to as *monotonicity*, to keep in line with [5] and [2]. Also, Sobrinho does not have a concept of boundedness as a separate property, but instead describes the concept of "strict isotonicity." We assert that although the concepts of monotonicity and boundedness are related (a path algebra that is strictly monotonic is also strictly bounded), the effects that each property has on the path algebra–best and loop-free, are disjoint, and therefore worth considering separately. We believe our approach is more elegant and more readily implemented.

In addition, [11] does not decouple the algebraic structure of the path algebra with the routing protocols used to implement it. Wedo not go into details of the algebraic underpinnings, which will be handled in future work. However, the structure of the path algebra is essential to understand which algorithms

work best under different circumstances. This approach enables us to consider exactly what weaknesses the algebraic structure may have, which gives us greater flexibility and precision in determining exactly how a protocol should correct that weakness. Future work will explore this area in detail.

In [11] Sobrinho explains the necessity of hop-by-hop routing in order to maintain scalability in very large networks, the largest being the Internet itself. He also argues that hop-by-hop routing has limitations in QoS networks, because of the issues with various path algebras. Only one example is provided, (Shortest-Widest), which does not sufficiently cover the various deviances of path algebras that are not monotonic or strictly bounded.

[11] defines the path algebra as *requiring* the property of monotonicity (which he calls "isotonicity"). Thus, Sobrinho concludes that hop-by-hop forwarding paths cannot be found with Shortest-Widest path algebra, because of the way he defines the concept of path algebra. We consider path algebras that are not monotonic in order to explore the exact properties that non-monotonicity induces.

[11] refers to the best path as being "lightest" or "optimal." Note that his definition of optimal differs from our definition of optimal (where optimal is defined as a path algebra that is both strictly monotonic and strictly bounded). Sobrinho attempts to solve the loop-freeness problem by a concept called "lexicographic lightness." Our solution of proving strict boundedness is more general, as lexicographic lightness must be implemented with the confines of an algorithm, rather than relying on the algebraic properties of the path algebra itself. However, lexicographic lightness may be a potential solution for a path algebra that is not strictly bounded [9].

Gouda and Schneider ([6]) present the concept that that boundedness and monotonicity are necessary for loop-freedom and best paths, which they call "maximizable routing metrics." They claim that if the original metrics are monotonic and bounded, then a composite metric is also monotonic and bounded (and thus maximizable). Our work builds upon [6], by generalizing and extending the concepts that Gouda and Schneider initially prove. Specifically, [6] discuss single metrics and composite metrics separately. In addition, they do not discuss path algebras or what they term "maximizable routing metrics" with more than two criteria. Our method does not distinguish between single-metric and composite metrics, but handles both the same way.

We also frame the criteria as properties of algebraic structures, which provides a more general result. Furthermore, Gouda and Schneider ([6]) provide a method to determine whether or not a best path can be found, but do not show which properties produce specific behavior. We extend their work by specifying that monotonicity provides best paths, and strict boundedness provides loop-freedom. Furthermore, [6] observe that the flow metric (what we refer to as Widest) is bounded. We show that although it is bounded, it is not strictly bounded, and so forwarding paths may produce loops.

Baras and Theodokopoulos ([2]) provide an extensive survey of work that has been done on the algebraic interpretation of the network path problem, but do not address the different behavior of forwarding paths. Baras and Theodokopoulos also explore the relationship between the distributive property of path algebras and how this prevents paths from being monotonic. Intuitively, in these path problems, how an edge weight contributes to a total path weight depends not only on the weight of the actual edge, but on where in the path the edge appears.

Gondran and Minoux ([5]) provide the pure mathematics upon which the entire algebraic approach to routing is based. Future work will use the algebraic structure which they provide in detail in [5] to motivate best, loop-free multipath routing.

8 Conclusion and Future Work

We have shown that in order for packets to be forwarded on best, loop-free forwarding paths the path algebra must be both monotonic and strictly bounded at the least. If a path algebra possesses the more stringent property of being optimal, then subpaths are guaranteed to also be best, loop-free forwarding paths. On the other hand, it is possible for path algebras that are not monotonic to produce forwarding paths that are loop-free, but the best path might not necessarily be found. In addition, if a path algebra is monotonic but not strictly bounded, then forwarding paths will be best but not necessarily loop-free. Monotonicity and strict boundedness are the *only* properties a path algebra must have to ensure best, loop-free forwarding paths. This fact is independent of algorithm, protocol or path algebra metrics.

We have been concerned with the specific issues of distributed routing (i.e. the Internet). To further generalize the concepts, exploration under different constraints is needed. Possible extensions could include routing where the path from source to current node is known, but not the destination. Another possible extension could be where each node only has information about its neighbors.

Future work will apply the properties studied here to finding multiple paths

through a network with multiple criteria. Currently, finding the k-shortest disjoint $s - t$ paths with multiple criteria is an open problem [2]. In addition, it is not clear how to find a semiring for a path problem that determines a set of best paths and best forwarding paths, rather than the single best path [10, 12, 3]. Gondran and Minoux in [5] suggest that distributive lattices may provide a key.

References

[1] Jo ao Luís Sobrinho and Timothy G. Griffin. Routing in equilibrium. In *Mathematical Theory of Networks and System*, 2010.

[2] John S. Baras and George Theodorakopoulos. *Path Problems in Networks*. Synthesis Lectures on Communication Networks. Morgan & Claypool Publishers, 2010.

[3] David Eppstein. Finding the k shortest paths. In *Proc. 35th Symp. Foundations of Computer Science*, pages 154–165. IEEE, 1994.

[4] J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *Networking, IEEE/ACM Transactions on*, 1(1):130–141, 1993.

[5] Michel Gondran and Michel Minoux. *Graphs, Dioids and Semirings: New Models and Algorithms (Operations Research/Computer Science Interfaces Series)*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[6] Mohamed G. Gouda and Marco Schneider. Maximizable routing metrics. *IEEE/ACM Trans. Netw.*, 11(4):663–675, August 2003.

[7] B.S.W. Schröder. *Ordered Sets: An Introduction*. Birkhäuser, 2003.

[8] Arunabha Sen, K. Selcuk Candan, K. Selcuk C, Afonso Ferreira, Bruno Beauquier, and Stephane Perennes. On shortest path problems with "non-markovian" link contribution to path lengths.

[9] B. R. Smith. Using dijkstra to compute hop-by-hop qos paths. 2011.

[10] Brad Smith and J.J. Garcia-Luna-Aceves. Best effort quality-of-service. In *17th International Confereonce on Computer Communications and Networks (ICCCN '08)*, August 2008.

[11] João Luís Sobrinho. Algebra and algorithms for qos path computation and hop-by-hop routing in the internet. *IEEE/ACM Trans. Netw.*, 10(4):541–550, August 2002.

[12] Jin Y. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.