Formally Verified Convergence of Policy-Rich DBF Routing Protocols

Matthew L. Daggitt and Timothy G. Griffin

Abstract—In this paper we present new general convergence results about the behaviour of the Distributed Bellman-Ford (DBF) family of routing protocols, which includes distance-vector protocols (e.g. RIP) and path-vector protocols (e.g. BGP). Our results apply to “policy-rich” protocols, by which we mean protocols that can have complex policies (e.g. conditional route transformations) that violate traditional assumptions made in the standard presentation of Bellman-Ford protocols. First, we propose a new algebraic model for abstract routing problems which has fewer primitives than previous models and can represent more expressive policy languages. The new model is also the first to allow concurrent reasoning about distance-vector and path-vector protocols. Second, we explicitly demonstrate how DBF routing protocols are instances of a larger class of asynchronous iterative algorithms, for which there already exist powerful results about convergence. These results allow us to build upon conditions previously shown by Sobrinho to be sufficient and necessary for the convergence of path-vector protocols and generalise and strengthen them in various ways: we show that, with a minor modification, they also apply to distance-vector protocols; we prove they guarantee that the final routing solution reached is unique, thereby eliminating the possibility of anomalies such as BGP wedgies; we relax the model of asynchronous communication, showing that the results still hold if routing messages can be lost, reordered, and duplicated. Thirdly, our model and our accompanying theoretical results have been fully formalised in the Agda theorem prover. The resulting library is a powerful tool for quickly prototyping and formally verifying new policy languages. As an example, we formally verify the pose a new algebraic model for abstract routing problems which has fewer primitives than previous models and can represent more expressive policy languages. The new model is also the first to allow concurrent reasoning about distance-vector and path-vector protocols. Second, we explicitly demonstrate how DBF routing protocols are instances of a larger class of asynchronous iterative algorithms, for which there already exist powerful results about convergence. These results allow us to build upon conditions previously shown by Sobrinho to be sufficient and necessary for the convergence of path-vector protocols and generalise and strengthen them in various ways: we show that, with a minor modification, they also apply to distance-vector protocols; we prove they guarantee that the final routing solution reached is unique, thereby eliminating the possibility of anomalies such as BGP wedgies; we relax the model of asynchronous communication, showing that the results still hold if routing messages can be lost, reordered, and duplicated. Thirdly, our model and our accompanying theoretical results have been fully formalised in the Agda theorem prover. The resulting library is a powerful tool for quickly prototyping and formally verifying new policy languages. As an example, we formally verify the

I. INTRODUCTION

A. What Is Policy-Rich Routing?

This paper proves new, general results about the convergence of asynchronous routing protocols in the Distributed Bellman-Ford (DBF) family, which includes distance-vector (RIP-like) and path-vector (BGP-like) protocols. In particular, the results apply to what we call policy-rich protocols, and so we begin by informally explaining this terminology.

Suppose that a router participating in a DBF computation has a best route \( r \) to a destination \( d \). If it then receives a route \( r' \) for \( d \) from an immediate neighbour it will first apply some policy \( f \) associated with that neighbour and produce a candidate route \( f(r') \). It will then compare \( r \) with \( f(r') \) to determine which is best. Let us denote the outcome of this selection process as \( \text{best}(f(r'), r) \). A very simple example is the shortest-paths problem where \( r \) is just an integer representing distance, \( f_w(r) = w + r \) for some weight \( w \), and \( \text{best}(r, r') = \min(r, r') \).

If we dig deeply into the classical theory underlying best-path algorithms — both synchronous [1], [2], and asynchronous [3] — we find that it always assumes the following equation, or something equivalent, must hold:

\[
f(\text{best}(r_1, r_2)) = \text{best}(f(r_1), f(r_2))
\]

This property is referred to as distributivity. The left-hand side of the equation can be interpreted as a decision made by a router sending routes \( r_1 \) and \( r_2 \) while the right-hand side is a decision made by the neighbour receiving those routes. Assuming the equality holds, the classical theory proves that routing protocols arrive at globally optimal routes — where every pair of routers use the best path between them.

By a policy-rich language we mean one in which distributivity does not hold. A clear example of how distributivity violations might arise in routing can be seen in the use of route maps which are functions (scripts) that take routes as input and return routes as output. For example, if \( g \) and \( h \) are route maps, then we might define another route map \( f \) as:

\[
f(r) = \text{if } P(r) \text{ then } g(r) \text{ else } h(r),
\]

where \( P \) is a predicate on routes (such as “does this route contain the BGP community 17?”). To see how easily distributivity can be violated, suppose that:

\[
P(a) = \text{true},
\]
\[
P(b) = \text{false},
\]
\[
\text{best}(a, b) = a,
\]
\[
\text{best}(g(a), h(b)) = h(b).
\]
Then the left-hand side of Equation 1 is:

\[ f(\text{best}(a, b)) = f(a) = g(a), \]

while the right-hand side becomes:

\[ \text{best}(f(a), f(b)) = \text{best}(g(a), h(b)) = h(b) \]

For Eq 1 to hold we need \( g(a) = h(b) \), which may not be the case (indeed, if \( g(a) = h(b) \) were always true, then there would be no point in defining \( f \)’). Perhaps the most common example of such conditional policies is route filtering, where \( h(r) \) is equal to the invalid route.

A specialist schooled in the classical theory of path finding might be tempted to forbid the use of such “broken” policies when using the Bellman-Ford or Dijkstra algorithms. Yet, once again, practice has outstripped theory. The Border Gateway Protocol (BGP) — a key component of the Internet’s infrastructure — is a policy-rich routing protocol.

Hence a pertinent question is how to tame the policy language of BGP-like protocols to ensure good behaviour? Of course we could mandate that all protocols must conform to Equation 1. However, we would then not be able to implement typical inter-domain policies that are based on commercial relationships [5], [6]. In addition, something as simple as shortest-path routing with route filtering would be banned.

Gao & Rexford [7] showed that the simple commercial relationships described in [5] and [6], if universally followed, are enough to guarantee convergence. However their model is BGP-specific and gives us no guidance on how policy-rich protocols should be constrained in general. Furthermore their conditions impose constraints on the topology of the network and therefore they require continuous verification as the topology changes.

Instead an alternative middle ground for generic policy-rich protocols has been achieved for both Dijkstra’s algorithm [8] and the DBF family [9]. Rather than insisting on distributivity, we require that for all routes \( r \) and policies \( f \) we have:

\[ r = \text{best}(r, f(r)). \tag{3} \]

In other words, applying policy to a route cannot produce a route that is more preferred. Although Equation 3 is sufficient for Dijkstra’s Algorithm [8], it must be strengthened for DBF algorithms [9] to:

\[ r = \text{best}(r, f(r)) \land r \neq f(r). \tag{4} \]

That is, applying policy to a route cannot produce a route that is more preferred and it cannot leave a route unchanged. We call policy languages that obey Equation 3 increasing and those that obey Equation 4 strictly increasing. Note that if policies \( g \) and \( h \) are strictly increasing, then the conditional policy \( f \) defined in Equation 2 is also strictly increasing. In other words, a strictly increasing policy language remains strictly increasing when route maps are introduced.

However, without distributivity we can no longer achieve globally optimal routes and so we must be content with locally optimal routes [8] – each router obtains the best routes possible given the best routes revealed by its neighbours.

It is natural to ask if the strictly increasing condition is too strong and prohibits desirable policies? Sobrinho [9] shows that it is the weakest possible property that guarantees an extra stable state. Therefore, the next natural question to ask is: given a protocol with a non-strictly increasing policy language, which networks does the protocol converge over? Sobrinho shows that a network topology being free with respect to the policy language is a sufficient and necessary condition for a DBF protocol to converge over that network topology.

Subsequent to these foundational results, the algebraic approach has been extended to model other more complex methods of routing such as multi-path routing [13], [14], hierarchical routing [15], and quality of service routing [16].

B. Our Contributions

1) Answers to Big-Picture Questions: Given these prior results, the reader might assume that the convergence of standard DBF routing protocols is a solved problem. However, this is far from the case. In this paper, we address the following open questions:

Q1) Sobrinho’s proofs only guarantee that the protocols converge. Do they always converge to the same stable state? Answer: We prove that the answer is yes. From a theoretical perspective, this implies that the set of DBF protocols that converge and the set of DBF protocols that converge deterministically are the same. From a practical perspective, this implies that as long as a DBF protocol is guaranteed to converge then you don’t have to worry about problems like BGP wedgies.

Q2) The proofs of Sobrinho and Gao-Rexford assume TCP-like in-order, reliable communication between routers. Is this a fundamental requirement for convergence? Answer: We prove that the answer is no, by showing that the results still hold for a weaker, UDP-like model of communication in which route advertisements may be delayed, lost, reordered and duplicated. From a theoretical perspective, this provides strong evidence that the difficulty of solving a routing problem in a distributed environment is entirely independent of the communication model used. This is not the case in other distributed protocols, e.g. consensus protocols [17].

Q3) Sobrinho reasons about path-vector protocols. Do there exist corresponding results for distance-vector protocols?
Thanks to this existing theory, we construct a mathematical model for DBF protocols. A more technical discussion of these contributions can be found in Section II-H.

M1) We propose a new algebraic structure for modelling policy languages which is both simpler and more expressive than that of Sobrinho. In particular, it contains only 6 algebraic primitives instead of 8, and is capable of naturally modelling path-dependent policies such as route filtering and path-inflation.

M2) We identify that DBF protocols are a single instance of a much broader class of asynchronous iterative algorithms, for which there exist prior models and convergence results. From a theoretical point of view, this allows us to cleanly abstract the underlying routing problem being solved from the distributed environment in which it is being solved. Practically, it greatly simplifies the proofs and allows us to prove asynchronous convergence purely by reasoning about the behaviour of the protocol in an entirely synchronous environment. In contrast, previous work [7], [9] has typically created their own custom model of message passing in an asynchronous environment and therefore have to directly reasoning about message timings and have avoided reasoning about unreliable communication.

M3) Thanks to this existing theory, we construct a mathematical model of an abstract DBF routing protocol that is the first to precisely describe the entire evolution of the protocol, including exact message timings, routers and links failing, routers and links being added, and routers being reset. As far as we are aware, this is therefore the first fully provably correct executable model of an abstract DBF protocol. In contrast, previous proofs have focused on proving results about a snapshot of a single network configuration in which all routers, links and policies remain static.

3) Formal Verification: In the past decade there has been a strong trend towards the verification of infrastructure-related software in areas as diverse as operating systems [18], compilers [19], [20] and networking [21], [22].

In line with this trend we have fully formalised all the results in this paper in the Agda theorem prover [23]. As Agda is capable of expressing both proofs and programs, and as our model is executable, this constitutes the first ever fully formalised, executable model of a general DBF protocol as far as we are aware. The advantages of this formalisation effort are as follows:

V1) It provides a much stronger guarantee of the correctness of our proofs. Reasoning about algorithms in distributed environments is notoriously tricky, and as Sobrinho themselves say in [9], their proof of convergence is only “semi-formal”. Having a computer check our proofs is a valuable and powerful assurance as to their correctness.

V2) The generality of our verified proofs means that they also function as a library that can be used by others to rapidly prototype, explore the behaviour of and formally verify the correctness of novel policy languages. This is a significant step towards reusable and modular formal verification of routing protocols. The library is freely available [24], and we hope that it will prove useful to the community.

V3) To demonstrate the utility of our library, we construct a model of a path-vector routing protocol that contains many of the features of BGP including local preferences, conditional policy, path inflation and route filtering. We formally verify in Agda that the policy language is strictly increasing, and therefore no matter what policies are used the protocol will always converge deterministically to a single stable state given a sufficient period of stability. Such a protocol is therefore provably safe-by-design.

C. Road Map

In Section II, we use an algebraic approach inspired by that of Sobrinho [9] and existing literature on asynchronous iterative algorithms [25], to construct an executable model of an abstract, fully asynchronous, distributed DBF routing protocol. This approach provides a clear, implementation-independent specification of the family of protocols which we can reason about. In Section III we discuss how our novel use of an existing theory of asynchronous iterative algorithms does much of the heavy lifting, and finish by applying existing results from this field to prove our new results about distance-vector and path-vector protocols. In Section IV we describe our Agda formalisation of these results, and in Section V we construct a safe-by-design algebra using the library.

D. Previous Versions

Previous versions of this work have appeared before in conference proceedings [26] and in the thesis of the first author [27]. The many improvements in this paper over these prior versions are discussed in Section VI-C.

II. MODEL

In this section we construct a model of an arbitrary DBF routing protocol. For the reader’s convenience, a glossary of the notation used in the paper can be found in supplementary material.

A. Generality Vs Implementation Details

In any mathematical model of a real-world system, there is a fundamental tension between capturing the most general form of the system and capturing every detail of the implementation. More implementation details can always be added to better model a particular system, at the cost of losing the ability to apply the work more generally.
In this paper we err on the side of generality. While we acknowledge that BGP is the dominant policy-rich path-vector protocol, the aim of our work is not only to inform how BGP policies may be constrained to provide better guarantees, but also to guide the design of future policy-rich protocols.

For example, we do not explicitly model the details of either a soft-state or a hard-state implementation approach for DBF routing protocols. RIP [28] and EIGRP [29] are examples of the soft-state approach: routing information expires after a fixed time and so must be periodically refreshed. BGP [4] is the primary example of the hard-state approach: routing information is preserved at a router until it is explicitly modified by a neighbour’s update message or a local link failure. We use a very general model of asynchronous computation which we believe can accommodate such implementation choices.

B. Paths

Prior to anything else, it is important to discuss exactly what is meant by a path in the context of a distributed, asynchronous routing protocol. Traditionally paths are defined with respect to some graph $G = (V, E)$. However because a network’s topology changes over time, the path along which an entry in a router’s routing table was generated may not correspond to a valid path in the current topology. Furthermore, and perhaps counter-intuitively, such a path may be formed from an amalgamation of paths from previous topologies, and therefore may not even have existed in any single past topology. Consequently, out of necessity, our notion of a path cannot be tied to the network’s topology.

Therefore, we begin by defining $V$ as the set of all valid router identifiers. An edge is an arbitrary pair of router identifiers, and the set of edges is $E = V \times V$. A path is a sequence of contiguous edges and a cycle is a path which finishes at the same router it started at. A simple path is a path that doesn’t contain any cycles. The trivial path of zero length from a node to itself is denoted as $\lambda$. For our purposes, we also consider an additional simple path $\bot$, which represents the absence of a path. We refer to the set of simple paths as $P$.

An edge $e$ is aligned with a path $p$, written $e \rightsquigarrow p$, if either $p$ is the trivial path or the destination of $e$ is equal to the origin of $p$. If $e \rightsquigarrow p$ then $e$ may be concatenated to $p$ using the $\cd$ operator to form a new path $e \cd p$.

C. Routing Algebras

In order to model all possible DBF protocols rather than one particular protocol, we first abstract over the routing problem being solved by representing it as a raw routing algebra.

Definition 1 (Raw routing algebra): A raw routing algebra is a tuple $(S, \oplus, E, \otimes, \infty, f^\infty)$ where:

- $S$ is the set of path-weights. A path-weight is the data that the routing protocol associates with each candidate path.
- $\oplus : S \times S \to S$ is the choice operator, which given two path-weights returns the preferred path-weight. In the introduction, this was informally referred to as “best”.
- $E : E \to 2^S \to S$ is a map from an edge to the set of policy functions that may be used on that edge. Each policy in $E_{ij}$ is a function $f : S \to S$, which when given a path-weight returns a new path-weight formed by extending the original path by the edge $(i,j)$. For notational convenience we define the complete set of policies for all edges as $E^* = \bigcup_{(i,j) \in E} E_{ij}$.
- $\otimes \in S$ is the path-weight of the trivial path from any router to itself.
- $\infty \in S$ is the invalid path-weight, representing the absence of a path.
- $f^\infty : ((i,j) \in E) \to E_{ij}$ is the set of constantly invalid policies. For example $f^\infty_{ij} : S \to S$ is the policy for edge $(i,j)$ that maps every input path-weight to the invalid path-weight.

These are the core primitives that we will define the operation of the protocol with respect to. The “raw” in the name refers to the fact that, at this stage, we do not assume the primitives of the algebra to satisfy any axioms. We discuss various possible axioms in Sections II-F & II-G.

Example 1 (Shortest-paths algebra): In the algebra for the shortest-paths routing problem, the path-weights $S$ are the lengths of the paths i.e. $S_{\infty}$. When choosing between two path-weights, the $\min$ operator is used to choose the one with the shortest length. The set of extensions functions are those that add the length $w$ of the edge to the path-weights current length. The trivial path-weight has length $0$ and the invalid path-weight has length $\infty$. The constantly invalid policies are those that add $\infty$ to the length.

Table I contains additional examples of algebras representing a variety of simple routing problems. Note that in all of these examples, every edge has the same set of policy functions i.e. $E_{ij}$ and $f^\infty_{ij}$ do not depend on $i$ and $j$. This indicates that the policy functions do not use the identity of the routers at either end of the edge. The utility of assigning every edge its own set of policy functions will become clearer when we model path-vector protocols in Section II-G where the extension functions will add the current router to the path that the path-weight was generated along. A significantly more complex example algebra that uses these features is discussed in-depth in Section V. We also note that in practice, a policy function $f \in E_{ij}$ may be the composition of an export policy of router $j$ and an import policy of router $i$.

D. Synchronous DBF Protocol

Now that we can model what problem our abstract DBF protocol is solving, we move on to modelling how it solves
it. We start by defining a synchronous version of the protocol where update messages between routers are exchanged instantaneously and in parallel. Let $V \subseteq \mathbb{V}$ be the finite set of routers that will ever participate in the protocol and let $n = |V|$. We represent an instantaneous snapshot of the network topology by an $n \times n$ adjacency matrix $A$ where $A_{ij} \in E_{ij}$ is the current policy function on the link from router $i$ to router $j$. The absence of a link from $i$ to $j$ is represented by the constantly invalid function $f_{ij}$. The global state of the synchronous protocol can be entirely represented by the product of the state of each router’s routing table. We therefore represent an individual state by a matrix $X$ where row $X_i$ is router $i$’s routing table and so the element $X_{ij}$ is the path-weight of $i$’s best current route to router $j$. We will use $S^n$ to denote the set of routing table states (i.e. the set of vectors over $S$ of length $n$), and $S^{n \times n}$ to denote the set of global states (i.e. the set of $n \times n$ matrices over $S$).

We define the sum of two matrices $X$ and $Y$ as:

$$(X \oplus Y)_{ij} = X_{ij} \oplus Y_{ij}$$

and the application of $A$ to $X$ as:

$$A(X)_{ij} = \bigoplus_k A_{ik}(X_{kj})$$

In one synchronous round of a distributed Bellman-Ford computation every router in the network propagates its routing table to its neighbours who then update their own tables accordingly. We model this operation as:

$$F_A(X) = A(X) \oplus I$$

where $I$ is the identity matrix:

$$I_{ij} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}$$

The nature of the underlying computation becomes clearer if we expand out the definition of a single element of $F_A(X)$. Under the mild assumptions that the trivial path-weight $0$ is chosen over any other path-weight and that any other path-weight is chosen over the invalid path-weight $\infty$, we have:

$$F_A(X)_{ij} = A(X)_{ij} \oplus I_{ij}$$

$$= \bigoplus_k A_{ik}(X_{kj}) \oplus \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

$$= \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

i.e. router $i$’s always uses the trivial path-weight to route to itself, and its new route to router $j$ is the best choice out of the extensions of the routes to $j$ offered to it by each of its neighbours $k$.

Note that, although $F_A$ is defined in terms of $A$ and $X$ which represent the global network topology and protocol state respectively, we are still modelling a distributed protocol as each router only has a purely local view of the network and protocol state. This can be seen by again considering the expansion above. $F_A(X)_{ij}$, the new entry for destination $j$ in the routing table of $i$ after a single update, only depends on $A_{ik}$ for all $k$ (i.e. the policies on the incoming edges to $i$) and $X_{kj}$ (i.e. the entry for $j$ in the routing table of $k$, as broadcast by $k$). As previously discussed, if there is no working connection from $k$ to $i$ then $A_{ik} = \infty$ the constantly invalid function, and no information from $k$ arrives.

We can now model the evolution of the synchronous version of protocol, $\sigma$, as repeated applications of $F_A$.

**Definition 2 (Synchronous state function):** Given a network topology $A$ and an initial state $X$, the state of the synchronous DBF protocol at time $t$ is defined as:

$$\sigma^0(X) \triangleq X$$

$$\sigma^{t+1}(X) \triangleq F_A(\sigma^t(X))$$

### E. Asynchronous DBF Protocol

Although we have defined a synchronous model, in reality DBF protocols are distributed and asynchronous. Update messages are dispatched at non-deterministic intervals and may be delayed, reordered, duplicated or even lost along the way. New routers and links may be added to the network, existing routers and links may fail, and routers may be rebooted and reset to their original state. Reasoning about the outcomes of such unpredictable events is known to be extremely challenging.

One advantage of first defining the synchronous model is that it highlights that the underlying algorithm is inherently iterative. Consequently, an important contribution of this paper is to recognise that a DBF protocol is simply a special instance of the family of dynamic asynchronous iterative algorithms. We can therefore use existing work [25], [30] that construct general mathematical models of such iterations. We now describe a specialisation of the model presented in [25] to DBF protocols.

We assume a discrete and linear notion of time, $T$, denoting the times of events of interest in the network (a routing table updating, a router failing etc.). Layered on top of time is the notion of epochs, $E$, which represent contiguous periods of time during which the network topology remains unchanged, i.e. no links or routers fail or are added and no policies are changed. Given this, we can now model a dynamic network, $N$, as a function from epochs to adjacency matrices where $N^e$ is the topology of the network during epoch $e$.

The non-deterministic evolution of the protocol is described by a schedule.

**Definition 3 (Schedule):** (Definition 9 in [25]) A schedule is a tuple of functions $(\alpha, \beta, \eta, \pi)$ where:

- $\alpha : T \rightarrow 2^V$ is the activation function, where $\alpha(t)$ is the set of routers which update their routing table at time $t$.
- $\beta : T \times V \times V \rightarrow T$ is the data flow function, where $\beta(t, i, j)$ is the time at which the route advertisements used by router $i$ at time $t$ was sent by router $j$.
- $\eta : T \rightarrow E$ is the epoch function, where $\eta(t)$ is the epoch at time $t$.
- $\pi : E \rightarrow 2^V$ is the participants function, where $\pi(e)$ is the subset of routers currently participating in the protocol during epoch $e$. 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Using this machinery it is now possible to define the full asynchronous data flow function $\delta$. $\delta(t, i, j)$ is the time at which the latest message to arrive at $i$ up to time $t$ was sent from $j$. Messages from router $j$ to router $i$ may be reordered, lost or even duplicated. The only constraint is that every message must arrive after it was sent. Reproduced from [25].

This models the evolution of the protocol as follows. If a router is not currently participating then it adopts its non-participating state $I_{ij}$. If a router has just begun to participate, either because $t = 0$ or because it was not participating at the previous time step, it adopts the initial state $X_{ij}$. If the router is a continuing participant and is currently inactive then its state remains unchanged from its state at the previous time step, $\delta t−1(X)$. Otherwise, it is currently active, and it updates its state in accordance with router $i$'s local view, $L_i$, of the global state as currently advertised by its neighbours. More concretely, $L_{ij}$ is the last route advertisement for router $j$ that router $i$ received from its neighbour $k$.

Note that this is a straightforward generalisation of the synchronous model as we can immediately recover $\sigma$ from $\delta$ by setting $\alpha(t) = V$, $\beta(t, i, j) = t − 1$, $\eta(t) = 0$ and $\pi(t) = V$ i.e. there is a single epoch in which at each time step, every router is both participating and active and all messages only take a single time step to propagate.

### F. Axioms for Distance-Vector Protocols

Having defined the evolution of an abstract DBF protocol, $\delta$, parametrised by an algebra $(S, \oplus, E, \perp, \infty, f^{\infty})$, it is clear that $\delta$’s behaviour will be dependent on the properties of the algebra. So far, we have informally claimed that the algebraic primitives represent certain operations, for example $\oplus$ chooses between two candidate path-weights. We now make this formal by defining the axioms the primitives should obey in order for them to match our basic intuition about how they should operate in a standard distance vector protocol.

**Definition 6 (Routing algebra):** A raw routing algebra $(S, \oplus, E, \perp, \infty, f^{\infty})$ is a routing algebra if it obeys the following axioms:

| Axiom | Description |
|-------|-------------|
| **R1** | $\oplus$ is selective - choosing between two path-weights always returns one of the two path-weights. |
| **R2** | $\oplus$ is associative - the order doesn’t matter when making a sequence of choices between path-weights. |
| **R3** | $\oplus$ is commutative - the order doesn’t matter when choosing between two path-weights. |

\[
\forall x, y \in S : x \oplus y \in \{x, y\}
\]

\[
\forall x, y, z \in S : x \oplus (y \oplus z) = (x \oplus y) \oplus z
\]

\[
\forall x, y \in S : x \oplus y = y \oplus x
\]
$R4 \otimes \text{ is an annihilator for } \oplus - \text{ the trivial path-weight is more desirable than any other path-weight.} \forall x \in S : x \oplus \otimes = \otimes = \otimes \oplus x$

$R5 \otimes \text{ is an identity for } \oplus - \text{ any path-weight is preferable to the invalid path-weight.} \forall x \in S : x \oplus \otimes = x \otimes \oplus x$

$R6 \otimes \text{ is a fixed point for all policy functions - the extension of the invalid path-weight is also the invalid path-weight.} \forall f \in E^* : f(\otimes) = \otimes$

$R7 f^{\otimes} \text{ is the set of constantly invalid policies - no information flows through non-existent links.} \forall (i,j) \in E, \ x \in S : f^{\otimes}_{ij}(x) = \otimes$

It is important to note that these axioms are only those necessary and sufficient to guarantee that the algebraic primitives obey our intuition about them. As discussed in Section III, they are not sufficient to guarantee that $\delta$ itself is well-behaved.

We also note that some of these axioms may be relaxed when modelling more complex variations on standard routing protocols. For example, in multi-path routing the choice operator may not be selective as it can return a mix of the candidate path-weights from both of its arguments [13], [26]. Although an important line of work, this paper will not consider these variants further.

G. Axioms for Path-Vector Protocols

Path-vector protocols (e.g. BGP) are a subclass of DBF protocols which track the path along which a path-weight was generated. Routers then use this information to eliminate any path-weights that were generated via a cycle. It is well known that path-vector protocols have better convergence behaviour than distance-vector protocols. We would therefore like to extend our model to handle path-vector protocols as well.

The standard approach is to take an algebra $(S, \oplus, E, \otimes, \otimes, f^{\otimes})$ and model the path-vector protocol’s algorithm as operating over the set $S \times P$ instead of the set $S$. The synchronous iteration function $F_A$ in Equation 5 is then updated to track paths and eliminate path-weights generated along looping paths.

However, this approach has significant drawbacks. Firstly, the model of a path-vector protocol is no longer strictly a special case of the model for DBF protocols as they operate over different sets and have different algorithms. In turn, this means that they require different proofs of correctness. While this can (perhaps) be finessed in a pen-and-paper proof, it would result in a significant duplication of work during the computer formalisation described in Section IV. Secondly, it means that the choice operator $\oplus$ can’t make decisions based on properties of the path (e.g. its length), as $\oplus$ still operates over $S$ rather than $S \times P$. Lastly, we lose abstraction over how the paths are stored. For example, both $S \times P$ and $P \times S$ should be valid sets for a path-vector protocol to operate over, but the standard approach artificially elevates $S \times P$ over $P \times S$.

We therefore now propose a new approach to modelling path-vector protocols which overcomes all of these problems. We argue that paths should be modelled as being stored inside the path-weight $S$, rather than outside of it. Likewise $\oplus$ and $E$ should handle the tracking of paths and removal of looping paths, rather than lifting these additional operations to the algorithm. However, how can we enforce that paths are stored abstractly within elements of $S$? We cannot directly inspect the internal structure of $S$, $\oplus$ and $E$ and, in order for the model to be as general as possible, nor is it desirable to.

The answer is to assume that there exists a projection function $path : S \rightarrow P$ that extracts the path stored within the path-weight. We then axiomitise $path$ to interact with $\oplus$ and $E$ in the right way. Using this approach, we now define axioms for an algebra that can be used to model path-vector-specific behaviour.

Definition 7 (Path algebra): A path algebra is a raw routing algebra $(S, \oplus, E, \otimes, \otimes, f^{\otimes})$ equipped with an additional function $path : S \rightarrow P$ that obeys the following properties:

$P1$ A path-weight stores the invalid path if it is the invalid path-weight.

$\forall x \in S : x = \otimes \iff path(x) = \bot$

$P2$ The trivial path-weight stores the empty path.

$\forall x \in S : x = \otimes \Rightarrow path(x) = []$

$P3$ Applying an edge’s policy function to a path-weight simply extends the path-weight’s path by that edge, unless it does not result in a non-simple path in which case it becomes the invalid path.

$\forall x \in S, (i,j) \in E, f \in E_{ij} :$

$\begin{align*}
path(f(x)) = \\
\begin{cases}
\bot & \text{if } i \in \text{path}(x) \\
\bot & \text{if not } j \sim \text{ path}(x) \\
(i,j) :: \text{path}(x) & \text{otherwise}
\end{cases}
\end{align*}$

The path function allows us to extract the path stored inside the path-weight, along which the path-weight was generated, without revealing how the path is stored internally inside the path-weight. Assumptions $P1$–$P3$ ensure that the algebra’s implementation is tracking the paths correctly. In P3, the first case $i \notin \text{path}(x)$ guarantees that path-weights generated along paths with cycles in them are eliminated. The second case when $j \sim \text{path}(x)$ does not hold (i.e. $\text{path}(x)$ is not actually a path to $j$), ensures that invalid routing announcements are eliminated. Note that being able to state P3 is one of the reasons that we require that each edge $(i,j)$ comes with its own set of policy functions $E_{ij}$ in Definition 1. If instead we had only required the existence of a single set of policy functions (i.e. $E^*$) then we would be unable to define it.

As an example, we now construct a simple path-algebra for a shortest-paths path-vector protocol. We construct a more interesting path-algebra that models a BGP-like protocol in Section V.

Example 2 (Shortest-paths path-vector algebra): The set of path-weights $S$ is $(P \times N) \cup \{\infty\}$ where $\infty$ is the invalid path-weight. The trivial path-weight is the pair $(0, \mathbf{[]})$.
The choice operator, \(\oplus\), first compares the lengths of the paths, and then breaks ties by comparing the paths lexicographically:

\[
x \oplus \infty \triangleq x \\
\infty \oplus y \triangleq y
\]

\[
(m, p) \oplus (n, q) \triangleq \begin{cases} 
  x & \text{if } m < n \\
  y & \text{else if } n < m \\
  x & \text{else if } p \leq \text{lex} q \\
  y & \text{otherwise}
\end{cases}
\]

The policy functions \(E_{ij}\) for an arbitrary edge \((i, j)\) is the set of \(f_{ij}^l\) for all lengths \(l \in \mathbb{N}\) where:

\[
f_{ij}^l(\infty) \triangleq \infty
\]

\[
f_{ij}^l(m, p) \triangleq \begin{cases} 
  \infty & \text{if } i \in p \text{ or not } j \sim p \\
  (m + l, (i, j) :: p) & \text{otherwise}
\end{cases}
\]

plus the additional constantly invalid function \(f_{ij}^\infty\). Finally the required path function is defined as:

\[
\text{path}(\infty) \triangleq \bot
\]

\[
\text{path}(m, p) \triangleq p
\]

We note that as a result of the flexibility of Definition 7 we can easily flip the ordering of the pair in the algebra’s path-weights (i.e. from \(\mathbb{N} \times \mathbb{P}\) to \(\mathbb{P} \times \mathbb{N}\)) and it would still be a path algebra and the path function would simply change which component of the pair it returned.

In order to construct a full model of the path-vector, it is sufficient to note that a path algebra is simply a routing algebra with extra assumptions. Consequently, unlike in the standard approach, the algorithms described in Section II-D & II-E can be immediately reused to construct an asynchronous model of a path-vector protocol.

H. Discussion and Novel Contributions

To recap, we have now successfully constructed \(\delta\), an abstract model of the execution of a generic asynchronous DBF protocol. The model is parameterised by:

1) \((S, \oplus, E, \ominus, \infty, f^\infty)\) - a routing algebra which represents the routing problem being solved.

2) \(N\) - a function from epochs to adjacency matrices which represents the topology of the network over time.

3) \((\alpha, \beta, \eta, \pi)\) - a schedule which represents the timings of events in the network, e.g. node activations, message arrivals.

and we have defined sensible sets of axioms for the algebra that represent distance-vector and path-vector protocols.

We will now briefly revisit the main advantages of this new model over models proposed in previous work.

a) Simpler, more general algebraic structure: The algebraic structure we propose in Definition 1 is both more general and simpler than that proposed by Sobrinho [9], which in turn was inspired by the classical semiring approach [1], [31]. The main difference is how we model path-weight extension. In semiring algebras it is modelled by a second binary operator \(\otimes : S \times S \rightarrow S\). However this is incapable of modelling modern routing policies such as those described in Sections I & V. Sobrinho generalised this binary operator by introducing labels \(L\), signatures \(\Sigma\), an operator \(\triangleright : L \times \Sigma \rightarrow \Sigma\) and a function \(g : \Sigma \rightarrow S\) mapping signatures to path-weights. However, by modelling policies as a set of functions, \(E\), and as demonstrated in Section V, we retain the ability to model modern routing polices while reducing the size of the algebra from 8 primitives to 6 primitives.

b) Paths as part of the algebra: The second way in which the algebra is more general is that, by assigning each edge its own set of policy functions, we model the tracking of paths in path-vector protocols inside of the algebra itself. In contrast, Sobrinho [9] models the tracking of paths in path-vector protocols as a set of policy functions, which are only provided as inputs to his \(\oplus\) and \(E\) operators. Therefore the algebra cannot alter or make decisions based on those paths. Consequently, unlike ours, Sobrinho’s approach cannot model policies such as route filtering or path inflation in BGP.

c) Abstraction of paths: A third advantage of our new Definition 7 of a path algebra is that it only requires that the path function can extract a simple path from a path-weight, but does not force the protocol to reveal how paths are actually stored internally. This abstraction results in proofs that are far more robust to variations in how paths are stored internally by the protocol. For example, in Section V we show that our algebras can model BGP-like path inflation by simply adjusting the path function to strip out the inflated ASs, without having to adjust the proofs of convergence. In contrast, the model of path-vector protocols in Sobrinho [9] uses a concrete definition of paths, and therefore is incapable of modelling path inflation without altering the proof of convergence itself.

d) Use of existing asynchronous theory: The additional complexity introduced by the more expressive dynamic asynchronous model \(\delta\), such as routers failing and rebooting, at first glance appears daunting. However, as we will discuss further in Section III, our use of the existing theory of dynamic iterative algorithms means that we have access to a range of powerful theorems that, impressively, are capable of totally abstracting away the asynchronicity of the system.

e) Executable model of the protocol: Sobrinho [9] only provides an abstract temporal model of individual events that occur during the operation of the protocol. In contrast, as far as we are aware, ours is the first fully formal and executable model of the full dynamic behaviour of an abstract DBF protocols in which routers and links can fail, join and reboot. This means that our model can be run and its behaviour observed, and that our proofs can be written in a style amenable to the formal verification discussed in Section IV.

III. Convergence Results

A. Some Useful Concepts

Initially, we define some concepts that, although not required for constructing the model in Section II, are used in stating and reasoning about its convergence behaviour.
1) Ordering Over Path Weights: Given any routing algebra $(S, \oplus, E, \overline{0}, \infty, f^\infty)$, the choice operator $\oplus$ induces a preference order over path-weights as follows:

**Definition 8 (Preference relation over path-weights):** A path-weight $x$ is preferred to path-weight $y$, written $x \leq \oplus y$, if $x$ is chosen over $y$:

\[
x \leq \oplus y \quad \overset{\Delta}{=} \quad x \oplus y = x \\
x < \oplus y \quad \overset{\Delta}{=} \quad x \leq \oplus y \land x \neq y.
\]

As $\oplus$ is associative, commutative and selective, we have that $\leq \oplus$ is a total order and that for all path-weights $x$ we have $\overline{0} \leq \oplus x \leq \oplus \infty$.

2) Assignments: In our model which router is using which path-weight is implicitly defined by the path-weight’s location in the routing state (i.e. given a state $X$ then the path-weight $X_{ij}$ is being used by router $i$). Nonetheless a more explicit notion of which router is using which path-weight is crucial for stating sufficient and necessary conditions for the convergence of the model over a given network. We shall refer to such a path-vector protocol one would expect the first node in a couplet stores the entire path along which the path-weight was generated, an assignment only stores the first router in the path. This generalisation is necessary as we seek to reason about distance-vector as well as path-vector protocols, and in the former case we do not have access to the full path.

Several binary relations may be defined over the set of assignments. The first is the lifting of the preference order over path-weights to assignments.

**Definition 9 (Assignment):** An assignment $(i, x)$ is a pair of a router $i \in V$ and a path-weight $x \in S$, and represents that router $i$ is currently using path-weight $x$. The set of assignments $A \triangleq V \times S$.

Given an assignment $(i, x)$, during the correct operation of a path-vector protocol one would expect the first node in $\text{path}(x)$ to be equal to $i$. Note that an assignment is a generalisation of a couplet as defined by Sobrinho [9]. Whereas a couplet stores the entire path along which the path-weight was generated, an assignment only stores the first router in the path.

Several binary relations may be defined over the set of assignments. The first is the lifting of the preference order over path-weights to assignments.

**Definition 10 (Preference relation over assignments):** An assignment $a$ is preferred to assignment $b$, written $a \leq \oplus b$, if they have the same router and the path-weight of $a$ is preferred to the path-weight of $b$:

\[
(i, x) \leq \oplus (j, y) \quad \overset{\Delta}{=} \quad i = j \land x \leq \oplus y \\
(i, x) \leq (j, y) \quad \overset{\Delta}{=} \quad i = j \land x \leq \oplus y
\]

Note that unlike $\leq \oplus$, which is a total order over path-weights, $\leq \oplus$ is only a partial order over assignments, as it can only relate assignments to the same router. Intuitively, this makes sense as routers in a DBF protocol only compare paths that begin at themselves.

**Definition 11 (Extension relation):** An assignment $a$ extends assignment $b$ in the current topology $A$, written $a \triangleright= A b$, if the extension of the non-invalid path-weight of $a$ across the relevant link is equal to the path-weight of $b$:

\[
(j, y) \triangleright= A (i, x) \quad \overset{\Delta}{=} \quad A_{ij}(y) = x \land y \neq \infty
\]

In the context of a path-vector protocol, the axioms of a path algebra are sufficient to guarantee that if $(j, y) \triangleright= A (i, x)$ then $\text{path}(x) = (i, j) : \text{path}(y)$.

**Definition 12 (Threatens relation):** An assignment $a$ threatens assignment $b$ in the current topology $A$, written $a \leq A b$, if the extension of the non-invalid path-weight of $a$ across the relevant link is preferred to the path-weight of $b$:

\[
(j, y) \leq A (i, x) \quad \overset{\Delta}{=} \quad A_{ij}(y) \leq \oplus x \land y \neq \infty
\]

Intuitively $(j, y) \leq A (i, x)$ means that after the next update router $i$ may switch away from $x$ and instead use a path-weight that is equal to or preferred to $A_{ij}(y)$. It is easy to see that if $a \triangleright= A b$ then $a \leq A b$.

3) Non-Participating Nodes: As seen in Definition 4, the constraint that non-participating routers do not advertise routes to or receive routes from other routers is enforced at the level of the adjacency matrix $A^{EP}$ and therefore the iteration function $F_{\text{AEP}}$. Consequently, after a single iteration the state of any non-participating router’s routing table will always be the corresponding entry of the identity matrix, i.e. the trivial route to itself and invalid routes to all other routers in the network. In [25] such states are said to be accordant with the current set of participants.

**Definition 13 (Accordant states):** (Definition 15 from [25] with $\bot = I$) The set of routing states that are accordant with set of participants $p$, $S_{p \times n}$, is defined as:

\[
S_{p \times n} \triangleq \{ X \in S_{n \times n} \mid \forall i, j : i \notin p \Rightarrow X_{ij} = I_{ij} \}
\]

**B. Definition of Convergence**

Before proving convergence results about our model, we must first establish what does convergence even mean for a dynamic, asynchronous, iterative algorithm such as a DBF protocol. In this section, will build up a suitable definition by discussing in turn the complications introduced by each of these adjectives.

a) Iterative: The definition of convergence for a synchronous iterative algorithm is relatively simple. After a finite number of iterations, the algorithm must reach a stable state such that further iterations result in no further change. In terms of the synchronous model defined in Section II-D, this would translate to:

\[
\exists k^* \in \mathbb{N} : \forall k \in \mathbb{N} : k \geq k^* \Rightarrow \sigma^k(X) = \sigma^{k^*}(X)
\]

The resulting stable state $X^* = \sigma^{k^*}(X)$ is a fixed point for $F_A$, as:

\[
F_A(X^*) = F_A(\sigma^{k^*}(X)) = \sigma^{k^*+1}(X) = \sigma^{k^*}(X) = X^*
\]

From the perspective of a routing protocol, $X^*$ being a fixed point is equivalent to the statement that in state $X^*$ no router can improve its selected path-weights by unilaterally choosing to switch. Such a state can therefore be viewed as a locally optimal solution to the optimisation problem of finding the best consistent set of path choices in the current network topology. As will be discussed further in Section III-D, whether such a state is necessarily globally optimal depends on the properties of the routing algebra being used.
b) Asynchronous: At first glance, it appears relatively easy to adapt the above definition to a basic asynchronous model with only a single epoch: simply require that for every schedule the asynchronous iteration reaches a stable state. Unfortunately due to the asynchronous and distributed nature of the computation, and as discussed in Section II-E, our definition of a schedule does not assume reliable communication between routers. Consequently, in an arbitrary schedule a router may never activate or a pair of routers may never succeed in communicating. In such a schedule, it is clearly impossible for the protocol to converge to the desired stable state. Therefore, we need to restrict ourselves to only requiring that the protocol converges over reasonable schedules.

As long established in the asynchronous iterations literature [30], the right definition of “reasonable” is that the schedule contains pseudocycles. Informally, a pseudocycle can be thought of the asynchronous counterpart of a single synchronous iteration. We now present the formal definition of this concept from [25].

Definition 14 (Expire period, Definition 12 in [25]): A period of time \([t_1, t_2]\) is a expire period for node \(i\) if \(\eta(t_1) = \eta(t_2)\) and for all nodes \(j\) and times \(t \geq t_2\) then \(t_1 \leq \beta(t, i, j)\).

In other words, an expire period is a period of time \([t_1, t_2]\) within a single epoch, such that after time \(t_2\) node \(i\) never receives a message which was sent before \(t_1\), i.e. all messages to \(i\) that were in flight before \(t_1\) have either been dropped or have arrived by time \(t_2\).

Definition 15 (Dynamic activation period, Definition 11 in [25]): A period of time \([t_1, t_2]\) is a dynamic activation period for node \(i\) if \(\eta(t_1) = \eta(t_2)\) and there exists a time \(t \in [t_1, t_2]\) such that \(i \in \alpha(t)\).

Intuitively, an activation period for node \(i\) is a period of time within a single epoch in which router \(i\) updates its routing table at least once.

Definition 16 (Dynamic pseudocycle, Definition 13 in [25]): A period of time \([t_1, t_2]\) is a dynamic pseudocycle if \(\eta(t_1) = \eta(t_2)\) and for all nodes \(i \in \rho(t_1)\) there exists a time \(t \in [t_1, t_2]\) such that \([t_1, t]\) is an expire period for node \(i\) and \([t, t_2]\) is an activation period for node \(i\).

A pseudocycle is a period of time within a single epoch during which every participating node undergoes an expire period followed by an activation period. Lemma 8 in [25] proves that during a pseudocycle, the asynchronous model \(\delta\) makes at least as much progress towards a stable state as the synchronous model \(\sigma\) would with a single synchronous iteration. This is therefore the right notion of a schedule behaving reasonably.

A second aspect that we must consider when moving from the synchronous to the asynchronous model is that of nondeterminism. As discussed in Section I-A, BGP suffers from non-deterministic convergence where the protocol may end up in multiple stable states, and which stable state is reached depends on the schedule and the initial state, rather than just the topology. This problem is known colloquially as a BGP wedge [12]. Our definition of convergence should therefore require it to be deterministic, i.e. in each epoch only one stable state may be reached and that it is dependent only on the topology and the set of participants.

c) Dynamic: Finally, we now consider the full dynamic model with multiple epochs. In this scenario the network topology may change so frequently (i.e. the epochs may be so short), that the protocol never gets sufficient time to converge to a solution. Therefore we would like our definition of convergence to only require that the protocol will converge if given a sufficient period of stability. It is suggested in [25] that dynamic iterative algorithms, such as DBF routing protocols, should therefore be referred to as convergent rather than saying that they converge.

The final issue is that the protocol may only be convergent over some, rather than all, epochs and sets of participants. As shown by Sobrinho [9] and discussed further in Section III-D, DBF protocols only converge over network topologies that are free. Therefore the definition of convergent must be granular enough to express that the protocol is only convergent over a subset of epochs and sets of participants. A pair containing an epoch and a set of participants is referred to as a configuration. The set of all configurations \(C\) is therefore defined as:

\[ C = \mathcal{E} \times 2^V \]

Taking all the above points into account, we therefore arrive at the following definition of what it means for our protocol to be convergent:

Definition 17 (Convergent): (Definition 14 from [25]) The DBF protocol is convergent over a set of configurations \(C \subseteq C\) iff:

1) for every configuration \((e, p)\) there exists a fixed point \(X^*_{ep}\) for \(F_{\mathcal{E}^{ep}}\) and a number of iterations \(k^*_{ep}\).

2) for every initial state \(X\), schedule \((\alpha, \beta, \eta, \pi)\) and time \(t_1\) then if \((\eta(t_1), \rho(t_1)) \in C\) and the time period \([t_1, t_2]\) contains \(k^*_{(t_1)\rho(t_1)}\) pseudocycles then for every time \(t_3\) such that \(t_3 \geq t_2\) and \(\eta(t_2) = \eta(t_3)\) then \(\delta^{(\pi)}(X) = X^*_{\eta(t_1)\rho(t_1)}\).

The essence of this definition is that for every valid configuration, there exists a fixed point and a number of pseudocycles such that if, at an arbitrary point in time the configuration of the current epoch is in the set of valid configurations and the schedule contains that number of pseudocycles, then after that number of pseudocycles has passed the protocol will have reached the fixed point and will remain in it until the end of the current epoch.

C. An Existing Convergence Theorem

One of the most important advantages to using an established model of asynchronous iterations in Section II-E is that there already exist theorems describing sufficient conditions of such an iteration being convergent. The most powerful (and perhaps initially surprising) feature of these results is that the sufficient conditions only involve properties of the functions being iterated, \(F_{\mathcal{E}^{ep}}\), rather than the full asynchronous iteration, \(\delta\). Consequently one can use them to prove that \(\delta\) is convergent without ever having to directly reason about the unreliable, asynchronous and dynamic nature of the underlying network, e.g. messages being lost or reordered, routers and links failing or being added.
In this paper, we will use the AMCO conditions from [25]. For a broader survey of the range of alternative sufficient conditions available see [30].

**Definition 18 (AMCO, Definition 23 in [25]):** The functions \( F_{\mathbf{A}\times p} \) are a set of asynchronously metrically contracting operators (AMCO) over a set of configurations \( C \) if for every epoch \( e \) and set of participants \( p \) such that \((e, p) \in C\) then for all \( i \in p \) there exists a dissimilarity function \( d_i^{ep} : S^n \times S^n \rightarrow \mathbb{N} \) such that:

1) \( d_i^{ep} \) is indiscernible

\[ \forall \tau, \bar{\tau} : d_i^{ep}(\tau, \bar{\tau}) = 0 \iff \tau = \bar{\tau} \]

2) \( d_i^{ep} \) is bounded

\[ \exists n : \forall \tau, \bar{\tau} : d_i^{ep}(\tau, \bar{\tau}) \leq n \]

and if \( D^{ep}(X, Y) = \max_{i \in p} d_i^{ep}(X_i, Y_i) \) then:

1) \( F_{\mathbf{A}\times p} \) is strictly contracting on orbits over \( D^{ep} \)

\[ \forall X \in S^n^{|p|} : F_{\mathbf{A}\times p}(X) \neq X \]
\[ \Rightarrow D^{ep}(X, F_{\mathbf{A}\times p}(X)) > D^{ep}(F_{\mathbf{A}\times p}(X), (F_{\mathbf{A}\times p})^2(X)) \]

2) \( F_{\mathbf{A}\times p} \) is strictly contracting on fixed points over \( D^{ep} \):

\[ \forall X \in S^n^{|p|} : F_{\mathbf{A}\times p}(X^*) = X^* \land X \neq X^* \]
\[ \Rightarrow D^{ep}(X^*, X) > D^{ep}(X^*, F_{\mathbf{A}\times p}(X)) \]

3) \( F_{\mathbf{A}\times p} \) enforces accordancy:

\[ \forall X \in S^n^{|p|} : F_{\mathbf{A}\times p}(X) \in S^n^{|p|} \]

where \((F_{\mathbf{A}\times p})^2\) is application of \( F_{\mathbf{A}\times p} \) twice.

The properties required of an AMCO may seem complicated at first glance but each have clear interpretations. Firstly for every configuration in \( C \) there must exist a function \( d_i^{ep} \) which measures the dissimilarity between two routing table states (although it need not satisfy the formal definition of a metric). D1 requires that the dissimilarity between two states is zero if the states are equal, and D2 that there exists a maximum dissimilarity. The notion of dissimilarity over routing tables, \( d_i^{ep} \), is then naturally lifted to dissimilarity over routing states \( D^{ep} \) by calculating the maximum dissimilarity between the participating routing tables. D3 requires that the dissimilarity between the states after consecutive iterations strictly decreases, and D4 that if there exists a stable state then the dissimilarity to it strictly decreases after each iteration. Finally, D5 requires that applying the operator to an accordant state must always result in another accordant state.

**Theorem 1:** If \( F_{\mathbf{A}\times p} \) is a dynamic AMCO over a set of configurations \( C \) then \( \delta \) is convergent over \( C \).

**Proof:** See Theorem 5 in [25] and the Agda proof [24].

Informally, a synchronous version of the proof is as follows: D3 requires that each synchronous iteration reduces the dissimilarity between consecutate states. Consequently, as the dissimilarity is a bounded natural number by D2, then the dissimilarity must eventually reach zero, which by D1 means that a stable state has been reached. D4 then guarantees that this stable state is unique. With some non-trivial work, it is possible to lift this synchronous argument to an asynchronous argument because the dissimilarity function over routing states is decomposable into individual dissimilarity functions over routing tables.

The key point to take away is that by using this theorem in our later proofs, we reduce the problem of proving our model of the DBF protocol, \( \delta \), is convergent to merely constructing a suitable dissimilarity function \( d^{ep}_i \) which interacts with \( F_{\mathbf{A}\times p} \) in the right way.

**D. Types of Routing Algebras**

In this section, we adapt the hierarchy of necessary and sufficient conditions for convergence established by Sobrinho [9] to our new simpler algebraic model.\(^1\) The key insight of Sobrinho is that each of the routing algebra axioms in Definition 6 only reference either choice or extension, and that is the relationship *between* them that determines the convergence behaviour of the protocol.

As discussed in Section I-A, the gold-standard are distributive algebras:

**Definition 19 (Distributive algebra):** A raw routing algebra is distributive if:

\[ \forall f \in E^*, x, y \in S : f(x \oplus y) = f(x) \oplus f(y) \]

As has been proven many times in the classic algebraic routing literature and, again by Sobrinho, if the algebra is distributive then any stable state will necessarily be a globally optimal state in which every pair of routers is assigned the optimal path between them.

The next most desirable property for the algebra to have is to be increasing/strictly increasing:

**Definition 20 (Increasing algebra):** A raw routing algebra is increasing if:

\[ \forall f \in E^*, x \in S : x \underline{\leq} f(x) \]

**Definition 21 (Strictly increasing algebra):** A raw routing algebra is strictly increasing if:

\[ \forall f \in E^*, x \in S : x \neq \infty \Rightarrow x < f(x) \]

Sobrinho proved that the algebra being strictly increasing is a necessary and sufficient condition for a path-vector protocol to converge to some stable state in a given epoch. However, if the algebra is not distributive then that stable state will only be locally optimal rather than globally optimal, and some pairs of routers will end up using suboptimal paths.

Finally there is the *freeness* property over a particular network topology:

**Definition 22 (Free network topology):** A network topology \( \mathbf{A} \) is free with respect to the raw routing algebra if there does not exist a cycle \([v_1, v_2, (v_2, v_3), \ldots, (v_m, v_1)]\) and a set of path-weights \( \{x_1, x_2, \ldots, x_m\} \) such that:

\[ \forall i : (v_{i-1}, v_i) \leq_{\mathbf{A}} (v_i, v_1) \]

where \( i - 1 \) is calculated mod \( m \).

\(^1\)Our terminology differs slightly from that of Sobrinho as we use the traditional algebraic/order theoretic names for these conditions. In particular we use *distributive* instead of *isotonic* and *increasing* instead of *monotonic*. This is because the order theoretic notion of monotonicity, \( x \leq y \Rightarrow f(x) \leq f(y) \), is much more closely related to the notion of distributivity, \( f(x \oplus y) = f(x) \oplus f(y) \), than increasingness, \( x \leq f(x) \).
Intuitively a network is free if there does not exist a cycle where each router’s current assignment can threaten the assignment of the previous router in the cycle. Sobrinho proved that the network topology being free with respect to the underlying algebra is a necessary and sufficient condition for a path-vector protocol to converge over that network, and that in the shortest-paths algebra this is equivalent to there being no negative weight cycles. Let $C^{free}$ be the set of configurations in which the participating topology is free, i.e.

$$C^{free} = \{(e, p) \in C \mid A^{ep} \text{ is free}\}$$

We now prove a couple of small lemmas showing some of the relationships between these properties.

**Lemma 1:** If a routing algebra is strictly increasing then every network topology is free.

**Proof:** See Theorem 2 & subsequent discussion in [9] or our Agda proof [24].

**Lemma 2:** If a path algebra $(S, \oplus, E, \emptyset, \infty, f^{\infty}, path)$ is increasing then it is also strictly increasing.

**Proof:** Consider arbitrary $i, j \in V$ and $f \in E_{ij}$ and $x \in S$ such that $x \neq \infty$. We already have that $x \leq f(x)$ as the algebra is increasing so it remains to show that $x \neq f(x)$.

Suppose that $f(x) = x$. We then know that $f(x) \neq \infty$ as $x \neq \infty$. Likewise we know that $path(x) \neq \perp$ as otherwise by Assumption 7 we would have $x = \infty$. Therefore, by Assumption 7, we have that $path(x) = path(f(x)) = (i, j) :: path(x)$ which is a contradiction. Therefore $x \neq f(x)$ and we have the required result.

**E. DBF Convergence Results**

We now proceed to state our main convergence results, the proofs of which may be found in the supplementary material. We start by considering distance-vector protocols.

**Theorem 2:** If $(S, \oplus, E, \emptyset, \infty, f^{\infty})$ is a routing algebra and $S$ is finite then $\delta$ is convergent over $C^{free}$.

**Proof:** See supplementary material.

Theorem 2 says that for any distance-vector protocol with a finite set of path-weights then for any epoch and set of participating routers such that the network topology is free and given a sufficient period of stability in which the participating routers continue to activate and communicate, the protocol will always converge to the same stable state no matter the timing of messages and even when using unreliable communication.

A more useful result for strictly increasing, finite algebras is then immediately obtainable which shows that such a distance-vector protocol is always convergent.

**Theorem 3:** If $(S, \oplus, E, \emptyset, \infty, f^{\infty})$ is a strictly increasing routing algebra and $S$ is finite then $\delta$ is always convergent.

**Proof:** As the algebra is strictly increasing then by Lemma 1 every configuration is free, and therefore we have the required result immediately by Theorem 2.

In practice the finiteness condition is restrictive and excludes many routing algebras of interest. For example even the shortest-path algebra uses the infinite set $\mathbb{N}$ as path weights. However recall that Theorems 2 & 3 guarantee that, given a sufficient period of stability, the protocol will reconverge even in the presence of messages from previous epochs.

On the other hand shortest-path distance-vector protocols experience count-to-infinity problems when the state at the start of the epoch contains path-weights generated along paths that do not exist in the current topology. Finiteness is a sufficient condition to reduce count-to-infinity problems to merely count-to-convergence in distance-vector protocols. In the real world this is reflected in the design of RIP which artificially imposes a maximum hop count to ensure that $S$ is finite.

A common approach to avoid the finiteness requirement (and count-to-convergence issues) is that of path-vector protocols which track the paths along which the path-weights are generated. Path-weights are then removed if their path contains a cycle.

**Theorem 4:** Given an algebra $(S, \oplus, E, \emptyset, \infty, f^{\infty}, path)$ which is a routing algebra and a path algebra then $\delta$ is convergent over $C^{free}$.

**Proof:** See supplementary material.

Again a more useful version of this proof for strictly increasing path algebras is immediately obtainable.

**Theorem 5:** Given an increasing algebra $(S, \oplus, E, 0, \infty, f^{\infty}, path)$ which is a routing algebra and a path algebra then $\delta$ is always convergent.

**Proof:** As the path algebra is increasing then by Lemma 2 it is also strictly increasing, and therefore by Lemma 1 every configuration is free. Hence we have the required result immediately by Theorem 4.

**F. Novel Contributions**

We now recap the main contributions of these proofs.

a) **Deterministic convergence:** Sobrinho’s proofs only guarantee that the protocol will converge, and nothing in his proof forbids the stable state reached depending on the state at the start of the epoch or the ordering of messages between routers during the epoch. In contrast our proofs guarantee that the protocol always converges to the same stable state, no matter the initial state or what the ordering of messages is. This therefore eliminates the possibility of phenomena such as BGP wedgies.

b) **Unreliable communication:** The definition of the data-flow function, $\beta$, in Definition 3, Section II-E means that we only assume unreliable communication between routers. This differs from previous proofs of convergence, for both general DBF protocols [9] and BGP in particular [7], that assume reliable communication. Consequently, our model and resulting proofs in later sections in theory apply to protocols built on-top of unreliable transport protocols such as UDP [32].

c) **Extension to distance-vector protocols:** Theorems 2 & 3 apply to distance-vector protocols, whereas Sobrinho’s proofs only applied to path-vector protocols. In particular, this shows that convergence could still be guaranteed if complex conditional policies were added to distance-vector protocols like RIP.

**IV. Formalisation in AGDA**

We have formalised every mathematical result in this paper, down to the most trivial lemma, in the Agda theorem prover.
This includes the results about asynchronous iterative algorithms from [25]. As our proofs have been checked by a computer, we are far more confident in their correctness than usual. Even where we omit details or use standard informal mathematical reasoning to improve readability (e.g. Lemma 13 in supplementary material), the proofs are backed by mathematical arguments which are guaranteed to be fully rigorous.

Furthermore the act of formalisation itself was invaluable in creating and shaping these proofs. For example the unstructured way in which we initially laid out the pen-and-paper proof of Lemma 12 in supplementary material led us to overlook Case 2.3. Only when formalising the result, did we notice that this case remained unproven. This in turn led us to presenting the proof in the much cleaner structure displayed in this paper.

The library of proofs is freely available [24] and is highly modular. To prototype a new policy language users need only define it as an algebra and prove that the algebra obeys the strictly increasing conditions in order to guarantee the resulting protocol is always convergent. Furthermore the protocol is then executable so that its dynamic behaviour can be observed for a given network and schedule. In Section V we give an outline of how to define such an algebra in Agda. We hope that the library’s extensible nature means that it will be of use to the community when designing new policy languages.

V. A Safe-by-Design Algebra

We now present an example of how one could use our Agda library to develop a safe-by-design routing protocol. In particular, we explore the construction of a path-vector algebra that contains many of the features of BGP such as local preferences, community values [33] and conditional policies. These policies can perform operations such as path-filtering, path-inflation, modifying local preferences and communities. The conditions themselves are implemented using a simple language of predicates that includes the ability to inspect path-weights have a local preference, a set of communities and a path. The trivial path-weight, $\emptyset$, is defined as:

\[
\emptyset : \text{PathWeight}
\]

\[
\emptyset = \text{valid}(2^{32} - 1) \cup \emptyset
\]

where $2^{32} - 1$ is the highest possible local preference, $\emptyset$ is the empty set of communities and $\emptyset$ is the empty path. The invalid path-weight, $\infty$, is defined as:

\[
\infty : \text{PathWeight}
\]

\[
\infty = \text{invalid}
\]

The choice operator, $\oplus$, is defined as:

\[
\_ \oplus _ : \text{Op}_1 \text{PathWeight} \to \text{PathWeight}
\]

\[
x \oplus y@(\text{invalid}) = x
\]

\[
x@(\text{valid} \ l \ cs \ p) \oplus y@(\text{valid} \ m \ ds \ q) \text{ with compare } l m
\]

\[
\ldots \mid \text{tri} > \text{m} = x
\]

\[
\ldots \mid \text{tri} < \text{m} = y
\]

\[
\ldots \mid \text{tri} \approx \text{m} \text{ with compare } (\text{length } p) (\text{length } q)
\]

\[
\ldots \mid \text{tri} < \text{p} \approx \text{q}
\]

\[
\ldots \mid \text{yes } p \leq q = x
\]

\[
\ldots \mid \text{no } q \leq p = y
\]

which operates as follows:

1) If either $x$ or $y$ is invalid return the other.
2) otherwise if the local pref of either $x$ or $y$ is strictly greater than the other return that path-weight.
3) otherwise if the length of the path of either $x$ or $y$ is strictly less than the other return that path-weight.
4) finally break ties by a lexicographic comparison of paths.

Next we construct the set of policy functions \( E_{ij} \). We start by defining a simple yet expressive language for conditions that can be used by our policy language to make decisions.

\[
data \text{ Condition } : \text{Set where}
\]

\[
_/\text{and}_/: \text{Condition} \to \text{Condition} \to \text{Condition}
\]

\[
_/\text{or}_/: \text{Condition} \to \text{Condition} \to \text{Condition}
\]

\[
_/\text{not}_/: \text{Condition} \to \text{Condition}
\]

\[
_/\text{inPath}_/: \text{Node} \to \text{Condition}
\]

\[
_/\text{inComm}_/: \text{Community} \to \text{Condition}
\]

\[
_/\text{hasPref}_/: \text{LocalPref} \to \text{Condition}
\]

The semantics for these conditionals is defined as follows:

\[
\text{eval } : \text{Condition} \to \text{PathWeight} \to \text{Bool}
\]

\[
\text{eval } (\text{p and q}) \ x = \text{eval } p x \land \text{eval } q x
\]

\[
\text{eval } (\text{p or q}) \ x = \text{eval } p x \lor \text{eval } q x
\]

\[
\text{eval } (\text{not } p) \ x = \text{not } (\text{eval } p x)
\]

\[
\text{eval } (\text{inComm } c) \ (\text{valid } l \ cs \ p) = [ c \in? \ cs ]
\]

\[
\text{eval } (\text{inPath } i) \ (\text{valid } l \ cs \ p) = [ i \in? \ p ]
\]
We next define the policy language as follows:

We define the type of policy functions for each edge $(i, j)$.

Concretely there exists such a policy function for every policy

where the $\text{inflate}$ function prepends $n$ copies of the path’s source on the front of the path. Note that since the policy language provides no way of increasing a path-weight’s local preference or decrease the length of its path, it is not possible to define a non-increasing policy.

Agda is based on type-theory rather than set-theory, and therefore we cannot define a set of functions directly. Instead we define the type of policy functions $E_{ij}$ for each edge $(i, j)$. Concretely there exists such a policy function for every policy:

and the application of a policy function is defined as:

where $(i, j)$ tests if the edge $(i, j)$ is a valid extension of path $p$ (i.e. if $j = \text{src}(p)$), and $i \ p$ tests whether or not $i$ already exists in $p$ (i.e. if the resulting path would loop).

The $\text{hasPref}$ function tests whether or not $p$ has a preference for $i$.

The $\text{reject}$ policy function is always convergent, we would like to apply our formalisation of Theorem 5:

In order to apply it, we need to prove that the algebra $\text{bgpLite}$ is a routing algebra, a path algebra and increasing:

where $\text{delComm}$ strips any consecutive duplicate routers that might have been introduced by $\text{inflate}$. The reader might have noted at this point that all our proofs in this paper work over simple (i.e. loop free) paths, yet this algebra internally stores non-simple, inflated paths. However this apparent contradiction is solved by observing that, as was discussed in Section II-H, the definition of a path algebra only requires that we can extract a simple path from a path-weight, rather than that the path-weight stores a simple path. This is yet another demonstration of the power of the path algebra abstraction.

By combining the primitives together, we can now construct a $\text{RawRoutingAlgebra}$ object for the protocol:

Agda is a little more strict than pen-and-paper mathematics and so the “…” hide a couple of additional extra fields that are required in the formalisation but are left in implicit in the definitions in this paper. For example, Agda requires us to give an explicit definition for what we mean when we say that two path-weights are equal.

Using the algebra we can automatically construct the asynchronous state function from Definition 5 as follows:

where $n$ is the number of routers in the protocol. This function is fully computable, so if you provide a network, a schedule and a time $t$ it will compute the $\text{RoutingMatrix}$ that represents the current protocol state.

In order to prove that is always convergent, we would like to apply our formalisation of Theorem 5:

In order to apply it, we need to prove that the algebra $\text{bgpLite}$ is a routing algebra, a path algebra and increasing:

The $\text{path}$ function from path-weights to simple paths required by a path algebra can be defined as:
We omit the formalisations of the 13 proofs required above, as they are too long to reproduce here and can be found in the library. Some of them are very easy. For example, as they are too long to reproduce here and can be found in Chapters 4 & 5 in the PhD thesis of the first author [27].

There are many other features of BGP that are safely increasing but, for space reasons, are not included.

VI. OPEN QUESTIONS

A. Rate of convergence of path-vector protocols

One aspect of convergence that this paper does not address is how long it takes for convergence to occur. Consider a network of \( n \) routers. It has long been known that path-vector protocols with distributive, strictly increasing policies require \( \Theta(n) \) time to converge in the worst-case [1], [2]. However, it was only recently proved that non-distributive, strictly increasing path-vector protocols require \( \Theta(n^2) \) time to converge in the worst-case [36]. Meanwhile, a tight bound for the time required for the worst-case convergence of path-vector protocols in free networks is still not known however. Current results have it lying between \( \Omega(n^2) \) and \( O(n!) \).

Another question of interest is that it appears that not all non-distributive, strictly increasing algebras require \( \Omega(n^2) \) iterations (e.g. the shortest-widest-paths algebra). We suspect that a careful analysis of policy language features might be able to tease apart distinct classes with respect to worst-case convergence time.

B. Hidden Information

Unlike in (external) BGP, in the algebra described in Section V the local preference attribute is not deleted when exporting a route. This raises a more general issue for routing protocols that allow information to be hidden. It is an open question as to whether it is possible to have increasing algebras with hidden information without requiring global coordination.

Ensuring increasing policies in today's BGP may require communicating lost information with some other mechanism such as community values. Of course only the relative ranking of local preference values assigned within an AS matter. For example one AS might use a local preference of 100 for its most preferred routes, while another could use 2000. In this context can we ensure increasing policies using only bilateral agreements between neighbouring networks or does it truly require global coordination? If the latter, then a political rather than a technical solution is required.

C. Verification of Data-Center Policies

BGP is widely used today to implement (private) connectivity within and between data-centers [37], [38], [39]. In such an environment the network architects have total control of the global topology and therefore, unlike in inter-domain routing, hidden information is not an issue. Recent work [39] provides good examples of how conditional policies, combined with filtering and the manipulation of local preference on routes, are used at Facebook to achieve design goals in this environment. Perhaps tools such as Propane [40] could be extended to either ensure that all policies are strictly increasing, or at the very least provide warnings when they are not.

PREVIOUS VERSIONS OF THIS WORK

This version makes the following new contributions, listed in order of appearance:

1) in this version, the set of policy functions \( E \) in a routing algebra in Definition 1 are parameterised by edges, which allows us to define a path algebra in Definition 7 without having it depend on \( A \), the current topology of the network. Although an apparently minor change, this makes the path algebras significantly easier to work with.

2) in this version we use the new dynamic model of asynchronous iterations from [25] rather than the static model from [30] which only captures the dynamics of a single epoch. This is not only a more accurate model as it can capture messages from non-participating nodes from previous epochs, but it also results in the model being fully executable.

3) as a consequence of the previous point, in this version we therefore use the dynamic AMCO convergence results from [25] instead of the static ultra-metric convergence results from Gurney [41] used in the previous version.

4) previously we only proved results about strictly increasing algebras (Theorems 3 & 5), whereas here we include results about free networks (Theorems 2 & 4).

Except for Theorems 2 & 4, much of this work appears in Chapters 4 & 5 in the PhD thesis of the first author [27].
CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] M. Gondran and M. Minoux, *Graphs, Dioids and Semirings: New Models and Algorithms*, vol. 41. Cham, Switzerland: Springer, 2008.

[2] J. S. Baras and G. Theodorakopoulos, “Path problems in networks,” *Synth. Lectures Commun. Netw.*, vol. 3, no. 1, pp. 1–77, Jan. 2010.

[3] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data Networks*, vol. 2. Upper Saddle River, NJ, USA: Prentice-Hall, 1992.

[4] Y. Rekhter, T. Li, and S. Hares. (2006). A *Border Gateway Protocol 4 (BGP-4)*. RFC 4271. [Online]. Available: http://www.ietf.org/rfc/rfc4271.txt

[5] G. Huston, “Interconnection, peering and settlements: Part I,” *Internet Protocol J.*, vol. 2, no. 1, p. 1, Jun. 1999.

[6] G. Huston, “Interconnection, peering and settlements: Part II,” *Internet Protocol J.*, vol. 2, no. 2, p. 1, Jun. 1999.

[7] L. Gao and J. Rexford, “Stable internet routing without global coordination,” *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 681–692, Dec. 2001.

[8] J. L. Sobrinho and T. G. Griffin, “Routing in equilibrium,” in *Mathematical Theory of Networks and System*, 2010. [Online]. Available: https://www.conferences.hu/mtns2010/

[9] J. L. Sobrinho, “An algebraic theory of dynamic network routing,” *IEEE/ACM Trans. Netw.*, vol. 13, no. 5, pp. 1160–1173, Oct. 2005.

[10] D. McPherson, V. Gill, D. Walton, and A. Retana. (2002). *Border Gateway Protocol (BGP) Persistent Route Oscillation Condition*. RFC 3345. [Online]. Available: http://www.ietf.org/rfc/rfc3345.txt

[11] K. Varadhan, R. Govindan, and D. Estrin, “Persistent route oscillations in inter-domain routing,” *Comput. Netw.*, vol. 32, no. 1, pp. 1–16, Jan. 2000.

[12] T. G. Griffin and G. Huston. (2005). *BGP Wedgies*. RFC 4264. [Online]. Available: http://www.ietf.org/rfc/rfc4264.txt

[13] C.-K. Chau, “Policy-based routing with non-strict preferences,” in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, Aug. 2006, pp. 387–398.

[14] J. L. Sobrinho and M. A. Ferreira, “Routing on multiple optimality criteria,” in *Proc. Annua. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 211–225.

[15] A. J. T. Gurney and T. G. Griffin, “Lexicographic products in metarouting,” in *Proc. IEEE Int. Conf. Netw. Protocols*, Oct. 2007, pp. 113–122.

[16] H. Geng, X. Shi, X. Yin, Z. Wang, and S. Yin, “Algebra and algorithms for multipath QoS routing in link state networks,” *J. Commun. Netw.*, vol. 19, no. 2, pp. 189–200, Apr. 2017.

[17] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process,” *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[18] G. Klein et al., “sel4: Formal verification of an OS kernel,” in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ. (SOSP)*, 2009, pp. 1–14.

[19] R. Kumar, M. O. Myreen, M. Norrish, and S. Owens, “CakeML: A verified implementation of ML,” in *Proc. POPL*, 2014, pp. 179–191.

[20] X. Leroy, “Formal verification of a realistic compiler,” *Commun. ACM*, vol. 52, no. 7, pp. 107–115, Jul. 2009.

[21] A. Guha, M. Reizblatt, and N. Foster, “Machine-verified network controllers,” in *Proc. 34th ACM SIGPLAN Conf. Program. Lang. Design Implement.*, Jun. 2013, pp. 483–494.

[22] A. Zaostrovnykh, S. Pirelli, L. Pedrosa, K. Argyraki, and G. Candea, “A formally verified NAT,” in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 141–154.

[23] U. Norell, “Dependently typed programming in Agda,” in *Proc. 4th Int. Workshop Types Lang. Design Implement.*, Jan. 2009, pp. 1–2.

[24] M. L. Daggitt, R. Zniarod, L. van der Stoep, and T. G. Griffin. (2021). *Agda Routing Library*. [Online]. Available: https://github.com/MatthewDaggitt/agda-routing/tree/ton2021

[25] M. L. Daggitt and T. G. Griffin, “Dynamic asynchronous iterations,” *J. Parallel Distrib. Comput.*, vol. 164, pp. 168–177, Jun. 2022.

[26] M. L. Daggitt, A. J. T. Gurney, and T. G. Griffin, “Asynchronous convergence of policy-rich distributed Bellman–Ford routing protocols,” in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 103–116.

[27] M. L. Daggitt, “An algebraic perspective on the convergence of vector-based routing protocols,” Ph.D. dissertation, Dept. Comput. Sci., Univ. Cambridge, Cambridge, U.K., 2019. [Online]. Available: https://www.repository.cam.ac.uk/handle/1810/292067

[28] G. Malkin. (1998). *Rip Version 2*. RFC 2453. [Online]. Available: http://www.ietf.org/rfc/rfc2453.txt

[29] D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, and R. White. (2016). *Cisco’s Enhanced Interior Gateway Routing Protocol (EIGRP)*. RFC 7868. [Online]. Available: http://www.ietf.org/rfc/rfc7868.txt

[30] T. G. Griffin, “Exploring the stratified shortest-paths problem,” *Netw. Sci.*, vol. 1, nos. 1–4, pp. 2–14, Mar. 2012, doi: 10.1007/s13111-0003-6.

[31] T. G. Griffin and G. Wilfong, “Analysis of the MED oscillation problem in BGP,” in *Proc. 10th IEEE Int. Conf. Netw. Protocols*, 2002, pp. 90–99.

[32] M. L. Daggitt and T. G. Griffin, “Rate of convergence of increasing path-vector routing protocols,” in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 335–345.

[33] S. Jain et al., “B4: Experience with a globally-deployed software defined WAN,” in *Proc. ACM SIGCOMM Conf.*, Aug. 2013, pp. 1–12.

[34] P. Lapukhov, A. Premji, and J. Mitchell. (2016). *Use of BGP for Routing in Large-Scale Data Centers*. RFC 7938. [Online]. Available: http://www.ietf.org/rfc/rfc7938.txt

[35] A. Abhishekumar et al., “Running BGP in data centers at scale,” in *Proc. Symp. Networked Syst. Design Implement.*, 2021, pp. 65–81.

[36] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, “Don’t mind the gap: Bridging network-wide objectives and device-level configurations,” in *Proc. ACM SIGCOMM Conf.*, Aug. 2016.

[37] A. J. T. Gurney, “Asynchronous iterations in ultrametric spaces,” 2017, arXiv:1701.07434.