Asynchronous Convergence of Policy-Rich Distributed Bellman-Ford Routing Protocols

SIGCOMM 2018

Matthew Daggitt [1] Alexander Gurney [2] Timothy Griffin [1]

21st of August 2018

[1] University of Cambridge

[2] Comcast

Contributions

New correctness results for general **policy-rich** distance-vector and path-vector protocols:

- A new proof technique which allows us to avoid directly reasoning about the asynchronous nature of the protocols.
- Using it we prove stronger and more general results than previous work.
- All results are fully formalised in the theorem-prover Agda and can be easily extended.

Motivation

“I study distance-vector routing for my PhD":

Non-computer scientists: “Wow, that must be hard!"

Computer scientists: “Why are you doing that? Isn’t that solved?"

Motivation

“I study distance-vector routing for my PhD":

Non-computer scientists: “Wow, that must be hard!"

Computer scientists: “Why are you doing that? Isn’t that solved?"

Motivation

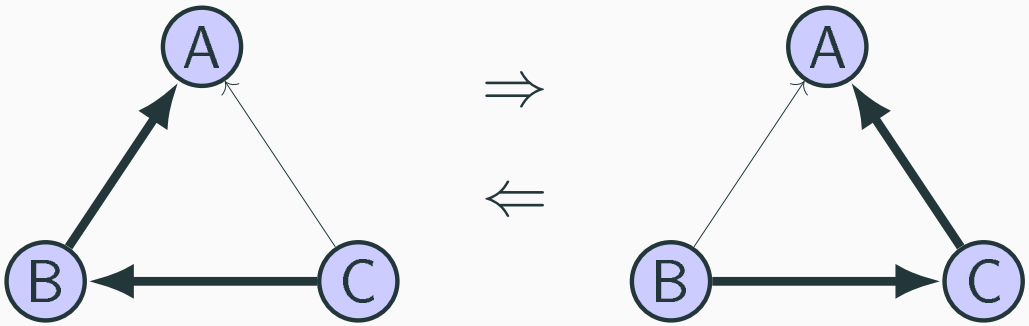
“I study distance-vector routing for my PhD":

Non-computer scientists: “Wow, that must be hard!"

Computer scientists: “Why are you doing that? Isn’t that solved?"

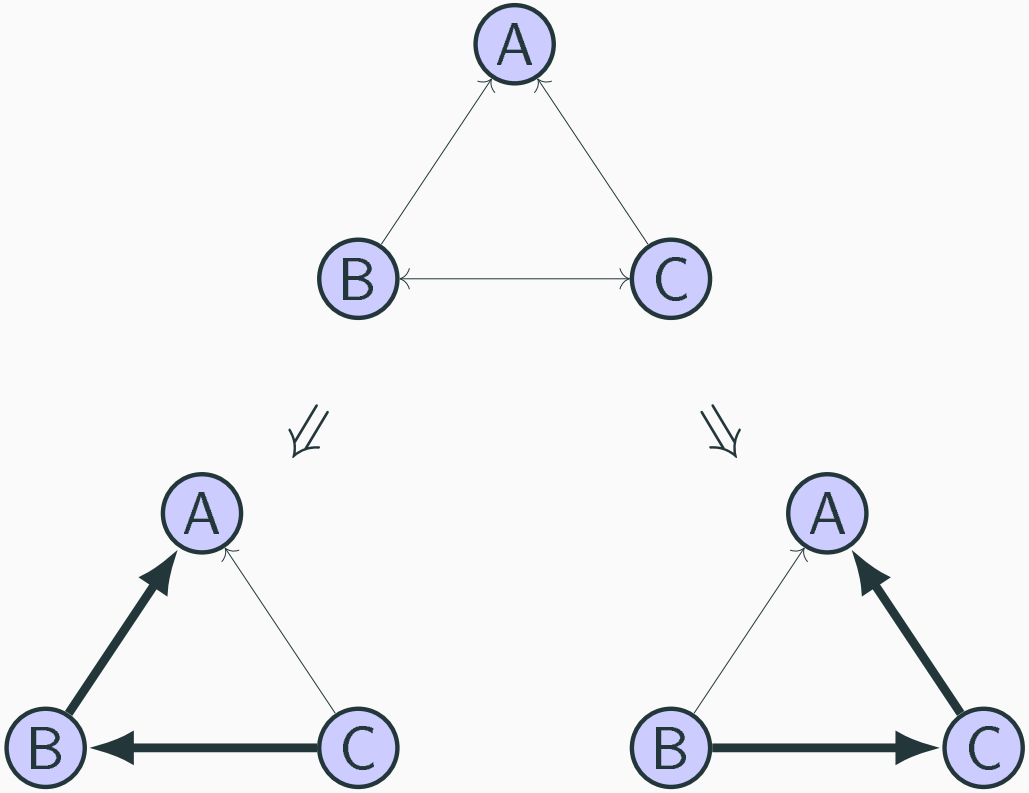
Motivation

- Routing oscillations (i.e. non-convergence)



Motivation

- Multiple final states (i.e. non-deterministic convergence)



Routing algebra

We abstract the key features of a routing protocol into a *routing algebra* [1]:

- A set of path weights
- A choice operator $\oplus$ defined on weights
  - such that $x \oplus y$ is either equal to x or is equal to y
- A set of policy functions F used to label edges
  - if you've configured a router think of an $f \in F$ as a route map
- Some weak axioms for $\oplus$ and F.

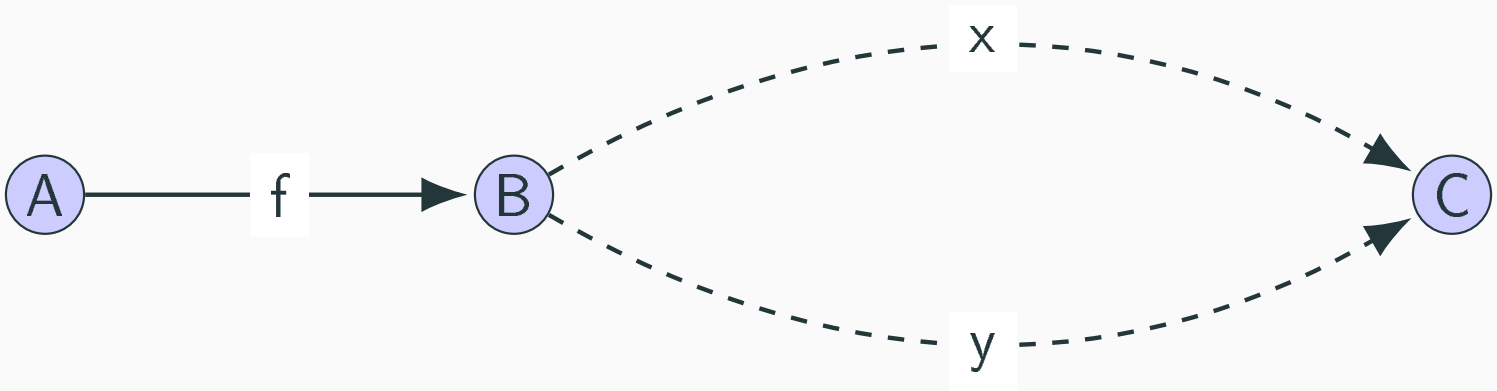
[1] Inspired by Sobrinho 2005

The hidden assumption

Q: So why do computer scientists think that routing is so easy?

A: In all common best-path problems the algebra is **distributive**.

$$f(x \oplus y) = f(x) \oplus f(y)$$

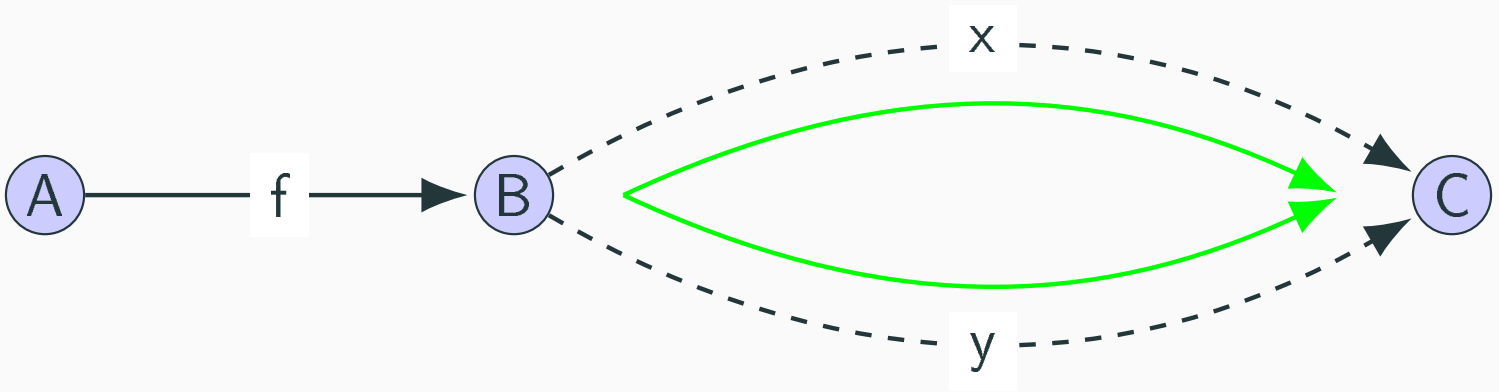


The hidden assumption

Q: So why do computer scientists think that routing is so easy?

A: In all common best-path problems the algebra is **distributive**.

$$f(x \oplus y) = f(x) \oplus f(y)$$

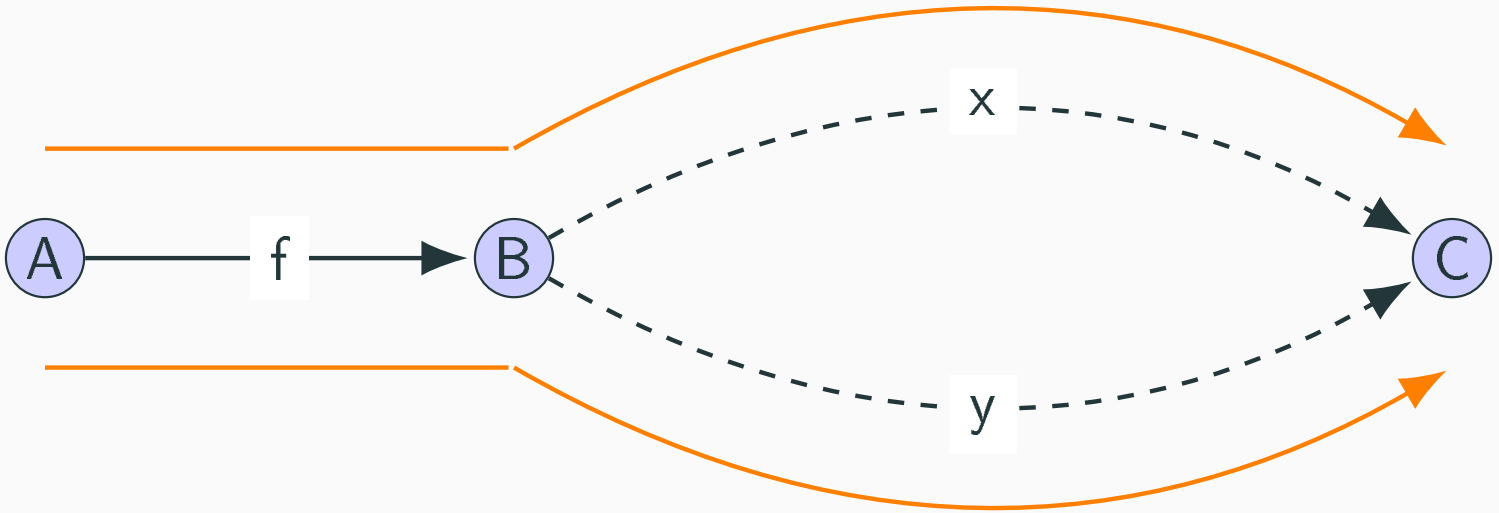


The hidden assumption

Q: So why do computer scientists think that routing is so easy?

A: In all common best-path problems the algebra is **distributive**.

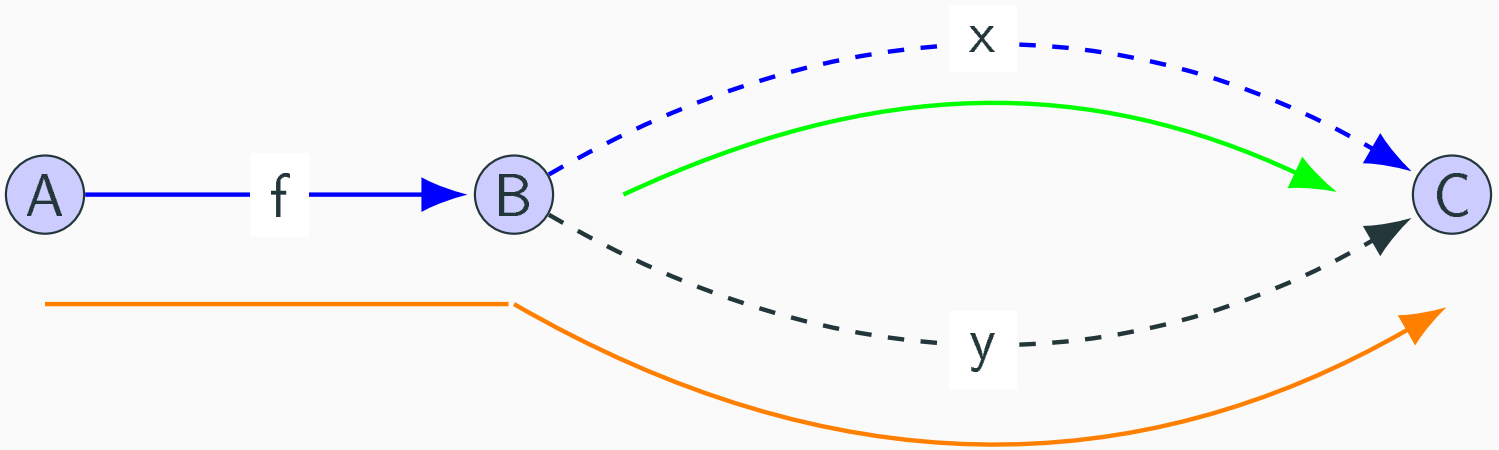
$$f(x \oplus y) = f(x) \oplus f(y)$$



Example of violating distributivity

Shortest-paths with "no more than one blue edge".

- Path x has one blue edge in it
- Path y has no blue edges in it
- The edge A to B is blue
- Node B prefers route x, i.e. $x \oplus y = x$



We will define a **policy-rich algebra** as one that allows us to violate distributivity.

Previous work: Sobrinho

Sobrinho 2005 showed there is a hierarchy of correctness conditions for path-vector protocols:

- Distributive algebras (long history of semirings)
- Strictly increasing algebras
- Networks with only algebraically free-cycles

Distributivity

The algebra is distributive iff:

$$\forall f, x, y : f(x \oplus y) = f(x) \oplus f(y)$$

i.e. everyone has a common preference order

Advantages:

- Global optimum!

Disadvantages:

- Very restrictive as to the problems you can solve.

Cycle-freeness

A network's cycles are free with respect to an algebra if going around a cycle is never an improvement.

Advantages:

- Least restrictive. Necessary for convergence.

Disadvantages:

- Conditions on the *topology* not just the algebra.
  - Requires global coordination to check!
- Verifying whether a network is cycle-free is NP-hard.

Strictly increasing

An algebra is *strictly increasing* if extending a route makes it worse:

$$\forall f, x : x < f(x)$$

where $x \leq y \triangleq x \oplus y = x$, $\quad x < y \triangleq x \leq y$ and $x \neq y$.

Advantages:

- Property of the algebra, hence independent of the network.
- Much more expressive than distributive algebras.
- Conditional policies ("route maps") preserve this property.

Disadvantages:

- Local optimums instead of global optimums.
- Only a sufficient rather than a necessary condition.

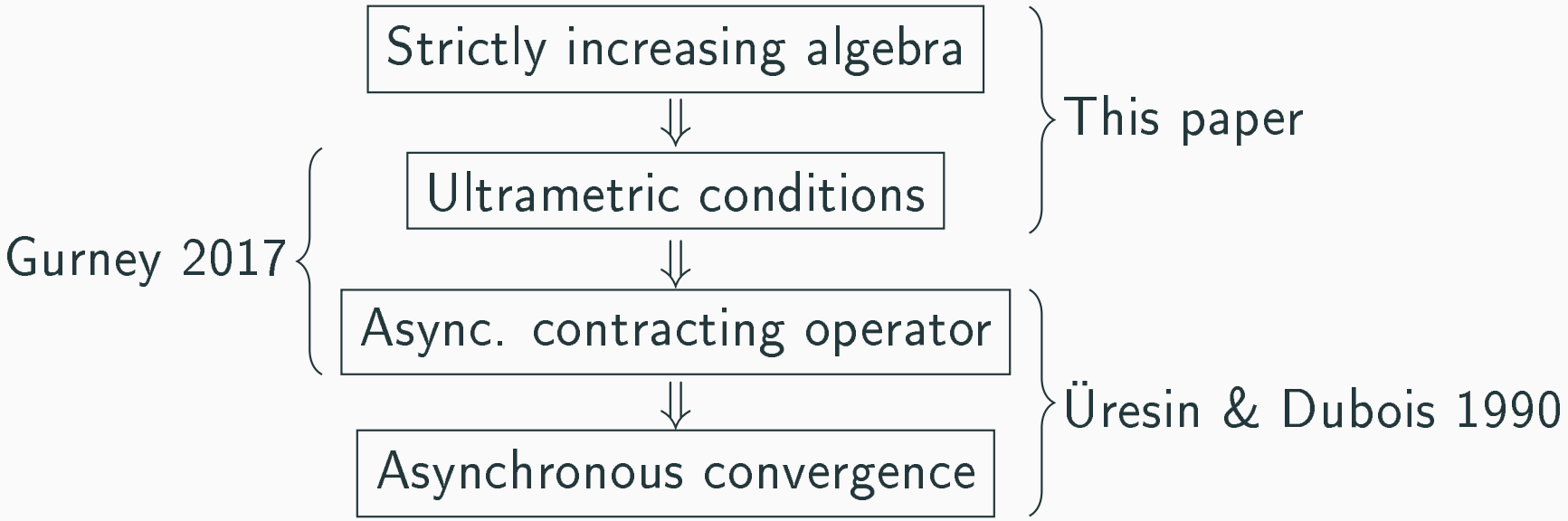
Our results

If the algebra is strictly increasing then:

- Path-vector protocols converge to a **unique** solution.
- Distance-vector protocols with a finite set of weights converge to a unique solution.
  - even policy rich ones
- Doesn't require in-order reliable delivery.

Our proofs use new techniques.

New techniques



This talk will not go into the details. If interested please read the paper!

Agda

All results formalised in the theorem proving language Agda.

The library is freely-available and easily extendable.

- `http://wiki.portal.chalmers.se/agda/pmwiki.php`
- `https://github.com/MatthewDaggitt/agda-routing`

Applications to BGP?

What are the problems with the *algebra* of BGP?

1. The MED attribute means that the $\leq$ order is not transitive, which is nearly impossible to reason about.
2. Local preferences are erased so the algebra is not strictly increasing i.e. $f(x) < x$.

It may be impossible to use our results in inter-domain BGP but we hope our results may be applicable to intra-domain BGP applications such as data centres.

Beyond BGP we hope that the results will be used when designing new policy-rich distance-vector and path-vector protocols.

Policy-rich example

We have defined a policy-rich shortest paths algebra, inspired by BGP.

Routes can be tagged with community values and these can be used by policies to make decisions.

Unlike BGP the local preferences are never erased.

This has been defined in Agda and proved correct.

It can easily implement the “no more than two blue links” example.

Open questions

- Can we incorporate checks for strict increasingness into automatic policy generation tools?
  - e.g. Propane (Beckett et al. SIGCOMM 2016)
- Is it possible to have hidden information in a strictly increasing algebras without global coordination?
- What's the rate of convergence of strictly increasing algebras?
  - Partial answer in our upcoming ICNP paper.

Agda example: routes

Routes are defined as follows:

```
data Route : Set where
  invalid : Route
  valid : LocalPref → CommunitySet → SimplePath n → Route
```

Agda example: choice

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

```
_⊕_ : Route → Route → Route
invalid ⊕ s = s
r       ⊕ invalid = r
r@(valid l cs p) ⊕ s@(valid k ds q) with <-cmp l k
... | tri< l<k _ _ = r
... | tri> _ _ k<l = s
... | tri≈ _ l=k _ with <-cmp (length p) (length q)
...   | tri< |p|<|q| _ _ = r
...   | tri> _ _ |p|>|q| = s
...   | tri≈ _ |p|=|q| _ with p ≤lex? q
...     | yes p≤q = r
...     | no p≰q = s
```

Agda example: extensions

```
data Policy : Set₁ where
  reject     : Policy
  addComm    : Community → Policy
  delComm    : Community → Policy
  compose    : Policy → Policy → Policy
  condition  : Condition → Policy → Policy
  incrPrefBy : ℕ → Policy
```

Agda example: extensions

```
f : (Node × Node × Policy) → Route → Route
f _                  invalid          = invalid
f (i , j , policy) (valid l cs p) with i ∉? p
... | no  i∈p = invalid
... | yes i∉p = apply policy (valid l cs ((i , j) :: p | i∉p))
```