We want to **synthesize routing configurations** from high-level objectives and constraints. This will support new enabling tools for operator control and understanding of their networks.

**Routing** is chosen because it admits a well-known theory, has challenging policy, and needs to be efficient.

Our initial use case is **repair** of a failed assertion about the routing outcome, by making **minimally invasive changes**.

This means that we aim to keep the traffic levels on each network link as similar as possible to those from before intervention. However, we still want to fix whatever is broken!

Assertions may fail due to operator error, device or link failure, or malicious attack.

Let’s consider a **sample network** of just 11 nodes and 34 directed links.

Each node is a network router. They run the Border Gateway Protocol (BGP) with one another, configured as route reflectors; and with external peers not shown.

They also run Open Shortest Paths First (OSPF) to select intra-network paths.

Some OSPF link weights will cause total BGP convergence failure. Such combinations are bad!

Our repair task is to find link weights that are good for BGP, but don’t shift traffic too badly.

Administrators have **high-level objectives** for network behavior...

“don’t put too much traffic on this link”

“don’t send more than twice what they send me”

“use this as a backup if the other one fails”

“avoid slow convergence”

“prefer my peer’s routes”

“certain flows should be blackholed”

“I don’t want my peer to know this route exists”

“keep regional flows local”

“maintain QoS”

“marked routes must be de-preferred”

…but only **low-level tools** to implement it.

Our current best method uses **numeric methods** with integrated **logical constraints**.

The relaxed (real, convex) problem is tractable in theory and practice.

We tested our approach against actual network protocol implementations, with the Bird software router (bird.network.cz) on Emulab (emulab.org).

BGP oscillations were induced, causing spikes in the control plane traffic.

The “start repair” marker shows when we invoked our program; “end repair” shows when BGP traffic has returned to normal. This happened in all tested cases (hundreds) and many more in simulation (tens of thousands).

The middle graph here shows an ineffective repair attempt using a purely random approach to identifying which weights to change – the problem appears to be solved, but recurs again moments later.

We compared the “TE cost” (traffic engineering cost) before and after intervention. The ratio is usually close to 1, indicating that major traffic shifts do not occur.

In contrast, a purely logical method that ignores traffic engineering has much worse performance – usually doing silly things like dumping a third of all traffic on a single link.

Let’s consider a sample network of just 11 nodes and 34 directed links.

Each node is a network router. They run the Border Gateway Protocol (BGP) with one another, configured as route reflectors; and with external peers not shown.

They also run Open Shortest Paths First (OSPF) to select intra-network paths.

Some OSPF link weights will cause total BGP convergence failure. Such combinations are bad!

Our repair task is to find link weights that are good for BGP, but don’t shift traffic too badly.

So it’s hard to

- verify invariants
- safely transform configurations
- restore broken assertions
- optimize the network
- reason about trade-offs

![Image of network](image1.png)

**Our initial use case is repair of a failed assertion about the routing outcome, by making minimally invasive changes.**

This means that we aim to keep the traffic levels on each network link as similar as possible to those from before intervention. However, we still want to fix whatever is broken!

Assertions may fail due to operator error, device or link failure, or malicious attack.

Let’s consider a **sample network** of just 11 nodes and 34 directed links.

Each node is a network router. They run the Border Gateway Protocol (BGP) with one another, configured as route reflectors; and with external peers not shown.

They also run Open Shortest Paths First (OSPF) to select intra-network paths.

Some OSPF link weights will cause total BGP convergence failure. Such combinations are bad!

Our repair task is to find link weights that are good for BGP, but don’t shift traffic too badly.

So it’s hard to

- verify invariants
- safely transform configurations
- restore broken assertions
- optimize the network
- reason about trade-offs

**Administrators have high-level objectives for network behavior...**

“don’t put too much traffic on this link”

“don’t send more than twice what they send me”

“use this as a backup if the other one fails”

“avoid slow convergence”

“prefer my peer’s routes”

“certain flows should be blackholed”

“I don’t want my peer to know this route exists”

“keep regional flows local”

“maintain QoS”

“marked routes must be de-preferred”

**...but only low-level tools to implement it.**

**We want to synthesize routing configurations from high-level objectives and constraints. This will support new enabling tools for operator control and understanding of their networks.**

**Routing** is chosen because it admits a well-known theory, has challenging policy, and needs to be efficient.

**Synthesis** encompasses automatic and user-guided methods for generating network configurations.

We focus on the **configuration of existing protocols** – in essence, programming the network in a special language.

Our initial use case is **repair** of a failed assertion about the routing outcome, by making **minimally invasive changes**.

This means that we aim to keep the traffic levels on each network link as similar as possible to those from before intervention. However, we still want to fix whatever is broken!

Assertions may fail due to operator error, device or link failure, or malicious attack.

Let’s consider a **sample network** of just 11 nodes and 34 directed links.

Each node is a network router. They run the Border Gateway Protocol (BGP) with one another, configured as route reflectors; and with external peers not shown.

They also run Open Shortest Paths First (OSPF) to select intra-network paths.

Some OSPF link weights will cause total BGP convergence failure. Such combinations are bad!

Our repair task is to find link weights that are good for BGP, but don’t shift traffic too badly.