University of Mons
Sciences Faculty
Computer Science Department

Probabilistic Verification of Network Configuration

Explanatory Report

Supervisor :
Bruno QUOITIN

Author :
Maxime BARTHA

Academic Year 2025-2026

Contents

1 Intro

2 Probabilistic Verification of Network Configuration

In this section, We explore the implementation of NetDice [1], a probabilistic verification tool for networks configuration. It verifies property with respect to a single (e.g. reachability) or multiple flows (e.g. balanced loads). With a flow from u to d and a failure scenario L, a forwarding graph (V_{fwd}, E_{fwd}) from u to d is considered instead of a simple path to be able to consider equal-cost multiple path routing.
The key insight of NetDice is, given a network configuration and a property ϕ, to prune failure scenarios where links are guaranteed to have no impact on the state of ϕ. To this end, NetDice introduces the concept of cold edge.

Definition 1 *Given a network configuration, a property ϕ and a flow (u, d), a set of edges $\mathcal{C} \subseteq E$ is cold iff any combination of failures in $\mathcal{C}$ is guaranteed not to change the forwarding graph for (u, d). An edge is called cold if it belongs to $\mathcal{C}$ and hot else.*

In practice, only hot edges are used to prune the failure scenarios. An algorithm to compute hot edges considering the shortest paths and static routes is presented in Alg.1.
All edges along the shortest path and the traversed static routes are hot as a failure in any of them would change if ϕ holds due to unreachability or changes in the forwarding graph.
To this end, every edge along the shortest path from the source u to the destination d is considered hot (Lin.2).
Then edges along the shortest path from every the end of a traversed static route are added as hot (Lin.4). As those static routes deviates the forwarding graph and the end node of a static route considers shortest path routing.
Finally, every edge on traversed static route is added as hot (Lin.5).

Algorithm 1 Hot edges for static routes and shortest path

```
1: procedure HotSpStatic(u, d, $E_{fwd}$, L)
2:     $\mathcal{H} \leftarrow AllSP(\{u\}, \{d\}, L)$                  ▷ initial shortest path
3:     $\mathcal{S} \leftarrow \{y \mid (x, y) \in Static_d \cap E_{fwd}\}$     ▷ end nodes of static route
4:     $\mathcal{H} \leftarrow \mathcal{H} \cup AllSP(\mathcal{S}, \{d\}, L)$
5:     return $\mathcal{H} \cup (Static_d \cap E_{fwd})$
6:
7: procedure AllSP($\mathcal{S}, \mathcal{T}, L$)
8:     return $\bigcup_{s \in \mathcal{S}, t \in \mathcal{T}} SP_L(s, t)$
```

As an example, Figure 1 represents the hot and cold edges of a certain network configuration considering no failure and a flow (D, C). Note that the E - C edge is marked as hot even thought it is not in the forwarding graph.

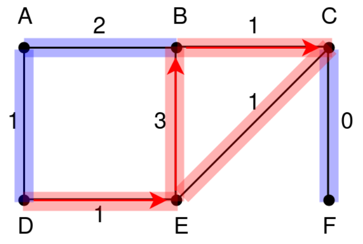

Figure 1: Cold Edges in a network configuration with no failures and a flow (D, C). E - B is a static route. Blue and red highlight show cold and hot edges. The forwarding graph is represented by the red arrows

Computing P(ϕ) is done by pruning the failure tree by only exploring failures of hot edges. A simple probability function mapping each edge to a certain failure probability is considered here, more complex probability failure model are discussed in 2. To explore the failure space, NetDice considers 3 possible states for an edge during the exploration process :

1. down : the edge cannot be traversed in current and future scenarios
2. up : the edge can be traversed in current and future scenarios
3. undefined : the edge is currently traversable but might not be in the future

Those state are only considered during the exploration process. In terms of forwarding graph computation, within a certain scenario, down edges are not considered whereas up and undefined are considered usable.

The exploration procedure can be described as follows :

1. initially, start with all edges as undefined as the scenario
2. compute the forwarding graphs with respect to the failure scenario
3. compute the hot edges with ϕ and the forwarding graphs
4. within the hot edges find the undefined
5. if ϕ holds in the failure scenario, compute the probability of it

6. explore recursively by setting undefined edges to down and up
7. return the sum of all probabilities (from recursive calls and own if ϕ holds)

In Pt.2, multiple forwarding graphs are possibly computed as equal-cost multi path (ECMP) is considered.

An example of execution on the network configuration from the Figure 1 with property ϕ "Traffic from D to C traverses E" is shown at Figure 2. In the example, only 16 failure scenarios are explored. In contrast to the $2^7 = 128$ scenarios a Brute Force algorithm would have had to explore.

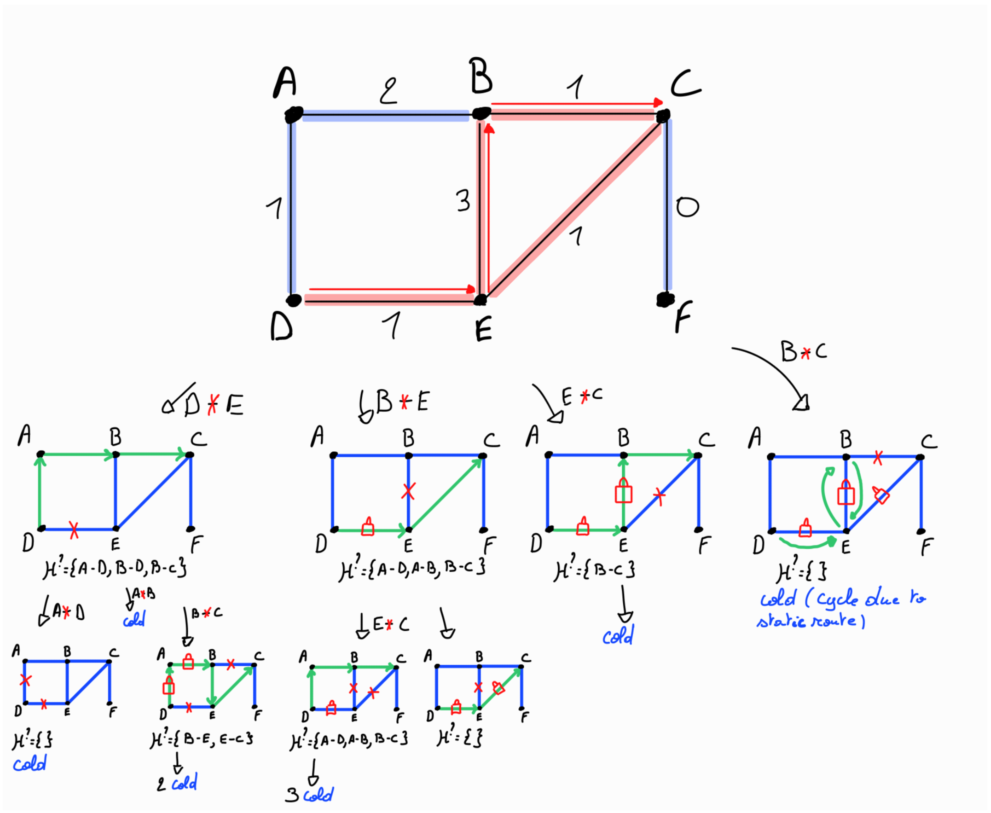

Figure 2: Execution Tree of the exploration of failure scenarios with property ϕ"Traffic from D to C traverses E". Crosses represents down links, padlocks represents up links. Absence of symbol represents undefined links.