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**

2 Probabilistic Verification of Network Configuration **3**

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. 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 and 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.

In addition, the algorithm considers the shortest path from every node traversed by the forwarding graph (set $\mathcal{D}$ Lin.2) and sets every traversed edges as hot (Lin.3). As even if the forwarding graph doesn't traverse the shortest path, as it is still considered by the nodes, any failure along it would change the shortest path.

As an example, after the application of the algorithm on the example of Fig.1, the E - C edge is marked as hot even thought it is not in the forwarding graph.

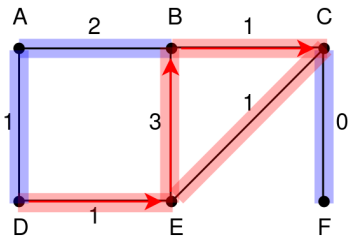

Figure 1: Cold Edges in a network configuration. E - B is a static route. Blue and red highlight show cold and hot edges. The forwarding graph is represented by the red arrows

Algorithm 1 Hot edges for static routes and shortest path

1: **procedure** HOTSPSTATIC(u, d, E_{fwd}, L)
2: $\mathcal{D} \leftarrow \{u\} \cup \{y|(x,y) \in Static_d \cap E_{fwd}\}$
3: $\mathcal{H} \leftarrow AllSP(\mathcal{D}, \{d\}, L)$
4: **return** $\mathcal{H} \cup (Static_d, E_{fwd})$
5:
6: **procedure** ALLSP($\mathcal{S}, \mathcal{T}, \mathcal{L}$)
7: **return** $\bigcup_{s \in \mathcal{S}, t \in \mathcal{T}} SP_{\mathcal{L}}(s,t)$

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 this end, 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.

It is assumed that a *check* function returning, given a property and forwarding graph, if the property holds or not.

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)

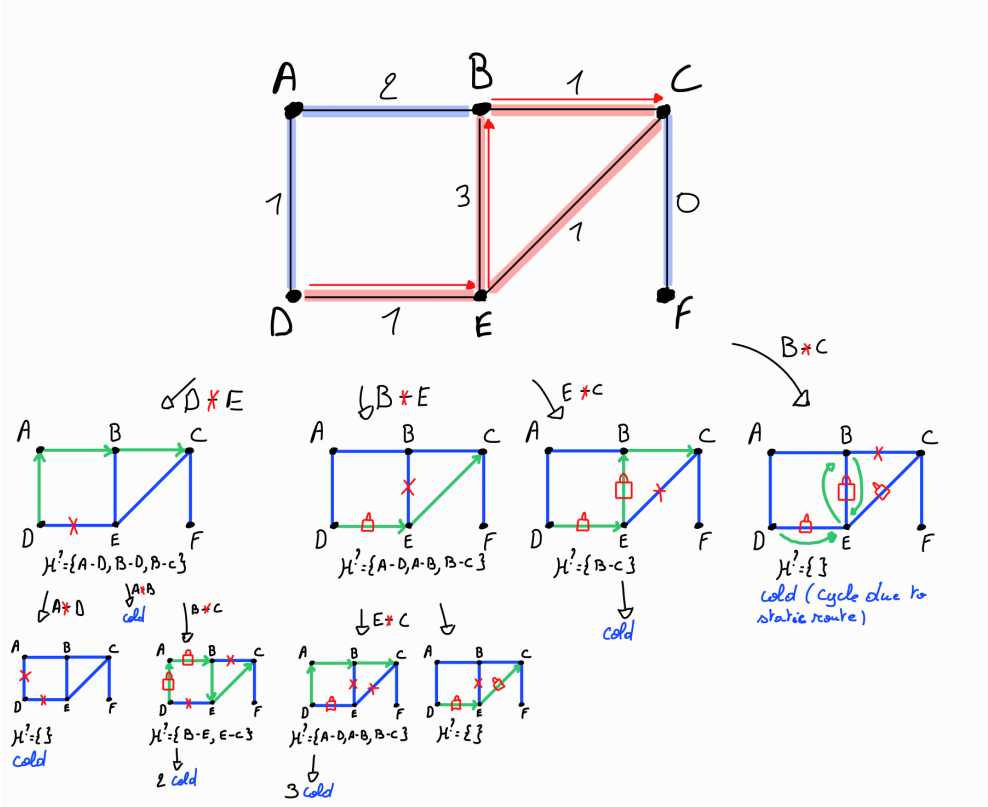

Figure 2: Execution Tree of the exploration of failure scenarios with property ϕ"Traffic from D to C traverses E". Crosses represents down links, padlocks represents up links. Absence of symbol represents undefined links.

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 Fig.1 with property ϕ "Traffic from D to C traverses E" is shown at Fig.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.