Faithfully Emulating Large Production Networks

Xiuqi Lu

2024.3.25

1 Reading & Writing

1.1 What problem is this paper solving?

For large-scale networks, existing vulnerabilities such as equipment failures, configuration errors, and human errors can rapidly lead to widespread network interruptions, resulting in significant losses. However, current network verification and testing tools (such as Batfish/MiniNet) are not effectively compatible with large production networks. Therefore, there is an urgent need in the industry for a large-scale, high-fidelity network emulation tool.

1.2 What is the key idea?

The key idea of this paper is to provide a highly scalable, cost-effective network emulator that can effectively capture potential vulnerabilities and prevent network outages. CrystalNet has three key features as below:

1. Ability to scale out using public clouds
2. Ability to transparently mock up physical networks
3. Ability to transparently mock up external networks

1.3 Any idea to improve the proposed solution?

1. In addition to OSPF/BGP, consider introducing more routing protocols.
2. Although it is currently capable of meeting the needs of simulating large networks, there is a lack of automated capabilities to efficiently locate and correct errors. Technologies such as machine learning could be introduced to efficiently identify and fix errors.
3. The algorithm for recognizing simulation boundaries is not generalizable and has high time complexity. Additionally, this algorithm requires recalculating new boundaries from scratch with every minor modification to the network topology, without utilizing previous results (production networks may undergo frequent, small-scale device configuration changes). This leads to a significant waste of computational resources, so algorithms that make incremental modifications to boundaries should be considered to avoid extensive redundant computations.
4. The algorithm successfully proves that it can confirm a safe bound, but does not evaluate how much larger this bound is than must-have devices? Starting from this, can the algorithm continue to be extended to find a minimum boundary?

1.4 Ask three questions you don't understand about the paper

1. I don't know how the production network is configured and run in the real environment, so I would like to know what factors contribute to its complexity?
2. How long does the simulator need to run to assess the reliability of the network system?
3. In addition to testing network reliability, is it possible to simulate the highest throughput supported by the server node?

2 Math formulation

2.1 Symbols and Definitions

Symbol	Definition
U	Network topology composed of routers and links
$E(U)$	Set of edges in the network topology (Links)
$V(U)$	Set of vertices in the network topology (Routers)
ED	Set of emulated devices
MD	Set of must-have devices
BD	Set of boundary devices
ID	Set of internal devices
SD	Set of speaker devices
DR	Set of designated router(OSPF)
BDR	Set of backup designated router(OSPF)
$AS(x)$	The Autonomous System (AS) to which router x belongs
$path(u, v)$	Set of routers that the BGP routing message passes from router u to v
$edges(U, V)$	Set of links between router set U and V

Table 1: **Notations**

2.2 Could you mathematically formulate *the problem* discussed in Sec 5?

The problem discussed in Section 5 is to determine a safe static boundary ED for a set of must-have devices MD in a given network topology U. We are seeking a set ED that meets the following criteria:

- $MD \subseteq ED$
- Routing information generated by any node in ED does not pass through any node in $V(U) \setminus ED$ and then return to ED.

2.3 Given your problem formulation and notations, could you formally express the Lemma and Propositions in Sec 5?

- ***Lemma 5.1.*** $\forall u, v \in ED, path(u, v) \subseteq ED \iff$ Boundary is safe

- ***Proposition 5.2.***

$$\left.\begin{cases} \forall x, y \in BD, & AS(x) = AS(y) \\ \forall x, y \in SD \text{ and } x \neq y, & AS(x) \neq AS(y) \end{cases}\right\} \Longrightarrow \text{Boundary is safe}$$

- ***Proposition 5.3.*** $\forall x, y \in BD, \quad \nexists u \in V(U) \backslash ED, \quad$ such that $u \in \text{path}(AS(x), AS(y)) \Longrightarrow$ Boundary is safe
- ***Proposition 5.4.*** No change in $edges(SD, BD)$ and $DR, BDR \subseteq ED \Longrightarrow$ Boundary is safe

3 Programming

3.1 Could you implement and test Algorithm 1 with a sample network?

Listing 1: **Demo.py**

```
"""
[Note] This script is used to give a sample network topology to test Algorithm-1.
[Details of the network topology]
    - Treat the topology as a multi-root tree with border switches being the roots
    - Starting from each input device, we add all its parents, grandparents and so on until the border switches
        into the emulated device set.
    - Clos-like datacenter network
        -- The whole network topology islayered
        -- Valley routing is now allowed
        -- The border switches connected to wide area network (WAN) are on the highest layer and usually share a
            single AS number.
[Network topology layer] Tor -> Leaf -> Spine(Border -> WAN)
"""

import queue

# Define a switch class
class Switch:
    def __init__(self, name, layer, as_num):
        self.name = name
        self.layer = layer
        self.as_num = as_num
        self.children = []
        self.parents = []

    def add_parent(self, parent):
        self.parents.append(parent)

# Add edge between two switches
def add_edge(parent, child):
    child.add_parent(parent)

def checkIsHighestLayer(switch):
    if switch.layer == "spine":
        return True
```

```
    return False

def findSafeDCBoundary(D):
    Dqueue = queue.Queue()
    DCB = []
    # Initialize queue with D
    for switch in D:
        Dqueue.put(switch)
    while not Dqueue.empty():
        switch = Dqueue.get()
        DCB.append(switch)
        if checkIsHighestLayer(switch):
            continue
        for parent in switch.parents:
            # not in DCB and not in Dqueue
            if parent not in DCB and parent not in  list(Dqueue.queue):
                Dqueue.put(parent)
    return DCB

if __name__ == "__main__":
    # Create switches
    t1 = Switch("t1", "tor", 200)
    t2 = Switch("t2", "tor", 200)
    t3 = Switch("t3", "tor", 300)
    t4 = Switch("t4", "tor", 300)
    t5 = Switch("t5", "tor", 400)
    t6 = Switch("t6", "tor", 400)
    l1 = Switch("l1", "leaf", 200)
    l2 = Switch("l2", "leaf", 200)
    l3 = Switch("l3", "leaf", 300)
    l4 = Switch("l4", "leaf", 300)
    l5 = Switch("l5", "leaf", 400)
    l6 = Switch("l6", "leaf", 400)
    s1 = Switch("s1", "spine", 100)
    s2 = Switch("s2", "spine", 100)

    # Add edges
    for tor in [t1, t2]:
        for leaf in [l1, l2]:
            add_edge(parent=leaf, child=tor)
    for tor in [t3, t4]:
        for leaf in [l3, l4]:
            add_edge(parent=leaf, child=tor)
    for tor in [t5, t6]:
        for leaf in [l5, l6]:
            add_edge(parent=leaf, child=tor)
    for leaf in [l1, l3, l5]:
        add_edge(parent=s1, child=leaf)
    for leaf in [l2, l4, l6]:
        add_edge(parent=s2, child=leaf)
```

```
84
85      # Only emulate L1 - L4
86      G = [t1, t2, t3, t4, t5, t6, l1, l2, l3, l4, l5, l6, s1, s2]
87      D = [l1, l2, l3, l4]
88
89      # Get boundary switches according to Algorithm-1
90      DCB = findSafeDCBoundary(D)
91      print(f"Safe DC boundary includes: {[switch.name for switch in DCB]}")
```

4 Conclusion

4.1 What is the most challenging part of your assigned tasks?

As a beginner, it takes a considerable amount of time to learn background knowledge while reading this paper, but the understanding of the knowledge is still very limited. Therefore, the most challenging part is to propose optimization strategies for this paper. Especially, it is difficult to check whether an idea is feasible.