Architectural Design and Test Case Analysis of a Simulator for Failure Localization

Xiangzhu Lu[1], Yan Jiao[1], and Pin-Han Ho[1]
[1]Department of Electrical and Computer Engineering, University of Waterloo,
200 University Ave W, Waterloo, ON, N2L 3G1, Canada

This paper presents the design and implementation of a simulator comprising the following components: a rule database, a topology generator, a failure generator, and an alarm generator. The topology generator produces network topologies to simulate various network conditions, while the failure generator generates simulated failures. Subsequently, the alarm generator utilizes the rule database to generate corresponding alarm data. The generated data structures include failures/alarms, alarm flows, alarm chains, and alarm correlation trees. Furthermore, a test case is introduced to validate the accuracy of the simulator.

***Index Terms*—Optical Transport Network (OTN), Failure Localization, OTN-based Simulator.**

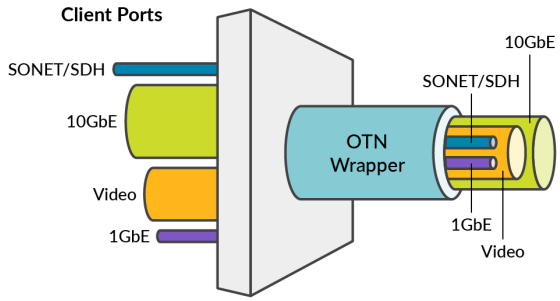


Fig. 1: OTN Wrapper [2]

I. Introduction

TELECOMMUNICATIONS networks have evolved significantly to handle large data volumes at high speeds. Traditional wired technologies like copper cables are facing limitations in bandwidth, distance, and interference. Fiber optic communication, on the other hand, overcomes these challenges by transmitting data as light signals, offering greater capacity, speed, and reliability. To ensure compatibility across manufacturers, standards like Optical Transport Network (OTN) have emerged. OTN, an ITU-T standard, efficiently transports, switches, and multiplexes various services onto a single high-capacity optical lightpath [1]. Referred to as a "digital wrapper," OTN encapsulates data frames from different clients, including IP, Ethernet, storage, digital video, and SONET/SDH, for transporting across optical networks, as depicted in Figure 1. Additionally, OTN supports error detection and correction through forward error correction (FEC) overhead.

However, even the most robust networks are susceptible to equipment malfunctions and environmental threats. When such challenges arise, localizing the source of failures becomes crucial. Failure localization is the process of tracing signals within the network to determine or localize the initial malfunctioning node(s). A failure event can occur unexpectedly at any board or fiber segment, disrupting the optical signals traversing through it. Subsequently, the failure may trigger alarms that propagate across multiple boards nearby and/or those situated in geographically distant areas, depending on network topology and traffic distribution. However, only alarm events are recorded in the OTN control plane, and those events encompass not only the root alarms triggered directly by a failure but also alarms that have propagated from other alarms. To ensure the efficient troubleshooting and maintenance of the network, it is essential to pinpoint the root location of failure by analyzing these alarm records.

Given the complexity and critical nature of this task, developing an efficient failure localization method is imperative. Such a method significantly reduces the time required for manual tracing back to the failure location. However, training this method requires a substantial amount of data and real-world data is difficult to obtain and often limited in scope. Therefore, developing a simulator is crucial. The simulator serves as a virtual environment where different network conditions and failure scenarios can be replicated in a controlled manner. It not only accurately reproduces real-world scenarios but also proactively generates potential failure scenarios.

This paper introduces an enhanced version of the OTN-based simulator, building upon the groundwork laid by its original version as discussed in [3]. Unlike the original version, which required manual coding of network topologies, the new simulator integrates a topology generator to automatically create diverse topologies and traffic scenarios. Furthermore, it operates at the board level rather than the node level, improving modularity and enabling the reuse of board-search functions across all node and board types. Additionally, the alarm generator has been improved with features such as random propagation times between alarm pairs, unique Alarm IDs, and the introduction of noise alarms to test the anti-noise capabilities of failure localization methods.

The rest of the paper is organized as follows. Section II provides a comprehensive overview of the main components and concepts defined in the simulator. Section III delineates the architecture of the simulator. Section IV demonstrates a test case for the proposed simulator. Section V summarizes

Manuscript received April 19, 2024; revised August 21, 2024. Corresponding author: Pin-Han Ho (email: p4ho@uwaterloo.ca).

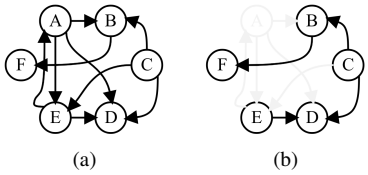


Fig. 2: Example of Dynamic Topology

this paper and delineates its limitations.

II. Key Concepts and Definitions

A. Topology

Topology is the physical arrangement of nodes and connections in the network. In the context of alarm propagation, if alarm A triggers alarm B, then their locations must be connected in the topology, either directly or through a path.

A topology can be static or dynamic. A **static topology** assumes that the network components (nodes and connections) remain unchanged over time. On the contrary, a **dynamic topology** assumes that the nodes and connections can be added or removed to form different network states. In the simulator, a **network state** is represented by a sub-topology of the entire network. To be more specific, the nodes and connections in a network state constitute a subset of those present in the entire network. Figure 2 is an example of dynamic topology, where Figure 2a is the entire network and Figures 2b is one of its network state. A static topology has only one network state.

B. Board

A **board** serves as the smallest unit in the topology. There are five types of boards in the simulator: FIU, OA, OM, OD and OTU. Fiber interface unit (FIU) serves as an intermediary between fiber and another node. A **connection** exists between two nodes when their FIUs are linked by two fibers in opposite directions. Optical multiplexer (OM) and optical demultiplexer (OD) boards facilitate the addition/dropping of optical signals, while the optical amplifier (OA) boosts signal strength. Lastly, the optical transponder unit (OTU) serves as the endpoint of a lightpath, where each OTU corresponds to at most one lightpath.

Similarly, a **fiber** connects two boards together. It is essential to recognize that fibers possess directionality. For example, an OA_FIU fiber denotes a fiber from an OA board to a FIU board. There are eight types of fiber: FIU_FIU, OA_FIU, FIU_OA, OA_OD, OD_OM, OM_OA, OD_OTU and OTU_OM.

C. Node

A **node**, comprising boards and fibers, represents an equipment in the network. There are two types of nodes involve in the simulator: reconfigurable optical add-drop multiplexer (ROADM) and optical line amplifier (OLA). Their structures are shown in Figures 3 and 4.

In the case of OLA, all boards and fibers have a fixed count and layout. However, the number of OTUs in a ROADM depends on the quantity of lightpaths originating from and terminating at it.

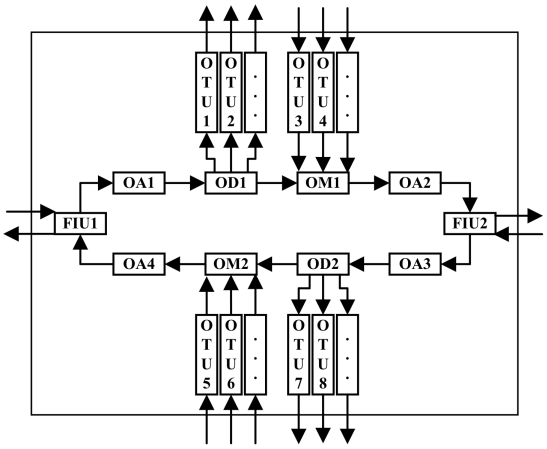


Fig. 3: ROADM Structure

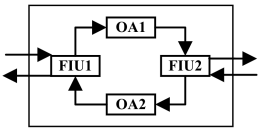


Fig. 4: OLA Structure

D. Lightpath

A **lightpath** is a directional path between two OTUs in the topology. Below are the three scenarios in which a node can possess a lightpath, along with their corresponding board-level paths:

- The lightpath originates from a node:
  - ROADM: OTU $\rightarrow$ OM $\rightarrow$ OA $\rightarrow$ FIU $\rightarrow \ldots$
  - OLA: there will be no lightpath originating from it.
- The lightpath terminates at a node:
  - ROADM: $\ldots \rightarrow$ FIU $\rightarrow$ OA $\rightarrow$ OD $\rightarrow$ OTU
  - OLA: there will be no lightpath terminating at it.
- The lightpath traverses through a node:
  - ROADM: $\ldots \rightarrow$ FIU $\rightarrow$ OA $\rightarrow$ OD $\rightarrow$ OM $\rightarrow$ OA $\rightarrow$ FIU $\rightarrow \ldots$
  - OLA: $\ldots \rightarrow$ FIU $\rightarrow$ OA $\rightarrow$ FIU $\rightarrow \ldots$

We can observe that the board-level path has a fixed pattern. Therefore, given the initiating and ending OTUs, the board-level lightpath can be derived from the node-level lightpath. In the simulator, a lightpath is represented by a list of boards. However, to enhance clarity and simplicity, this paper will depict lightpaths at the node level.

A **traffic** comprises a list of lightpaths, and a network state encompasses only one traffic, including all the active lightpaths transporting signals in the network state.

E. Alarm

Failure is an exceptional event that can trigger alarms in the OTN. It can occur on any board or fiber in the topology.

TABLE I: Example of Failure and Corresponding Alarms

Alarm type	Location	Time	ID	Failure ID	Is root	Is noisy
board faulty	ROADM5-OD1	06:00:00	40	40	false	false
OCh_LOS_P	ROADM5-OTU1	06:00:10	264	40	true	false
OCh_LOS_P	ROADM7-OTU93	06:00:04	265	40	true	false
OMS_A_P	ROADM5-OD1	06:00:07	266	40	true	false
OCh_A_P	ROADM5-OTU1	06:00:14	267	40	false	false
OCh_A_P	ROADM7-OTU93	06:00:15	268	40	false	false

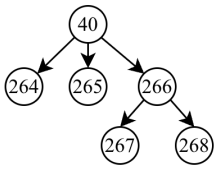


Fig. 5: Example of Alarm Correlation Tree

Alarm is an event triggered by an alarm/failure, which can only be located on a board. When an alarm directly results from a failure, it is considered the **root alarm** of that failure. It is worth noting that a single failure can generate multiple root alarms. Table I provides an illustrative example of a failure and its corresponding alarms, with the definition of each field detailed below.

- *Alarm type* denotes the category of the alarm/failure. The simulator defines a total of 25 alarm types and 2 failure types.
- *Location* denotes the board/fiber where the alarm/failure occurs.
- *Time* denotes the moment when the alarm/failure happens, measured in seconds.
- An unique *ID* is assigned to every alarm and failure.
- *Failure ID* is the ID of the failure which leads to the alarm.
- *Is root* indicates whether the alarm/failure is a root alarm.
- To test the anti-noise capability of the failure localization method, random noisy alarms can be added to the alarm set. *Is noisy* indicates whether the alarm is a noisy alarm.

To demonstrate the causal relationship within the failure and alarm set, the concept of alarm flow is introduced. An **alarm flow** consist of either a failure and an alarm or two alarms, where A $\rightarrow$ B indicates that failure/alarm A triggers alarm B. The aggregation of all alarms and alarm flows of one failure forms an **alarm correlation tree**, as given in the Figure 5, using the same failure example in Table I.

Each leaf alarm in the alarm correlation tree is associated with an **alarm chain**, which is a path from failure to respective leaf alarm. In Figure 5, there are four alarm chains: 40 $\rightarrow$ 264, 40 $\rightarrow$ 265, 40 $\rightarrow$ 266 $\rightarrow$ 267 and 40 $\rightarrow$ 266 $\rightarrow$ 268.

III. Simulator Architecture

A simplified architecture of the simulator is depicted in Figure 6, where dynamic topology is employed. To generate the desired alarm data, the topology generator will first create a topology according to the specified requirements. Then, for each network state, traffic will be generated based on the required number of lightpaths. The board/fiber list comprises all boards and fibers in each network state. Subsequently, the failure generator will randomly select failure locations from these boards and fibers. Finally, the alarm generator can generate alarm data for each failure based on the rules provided by the rule database.

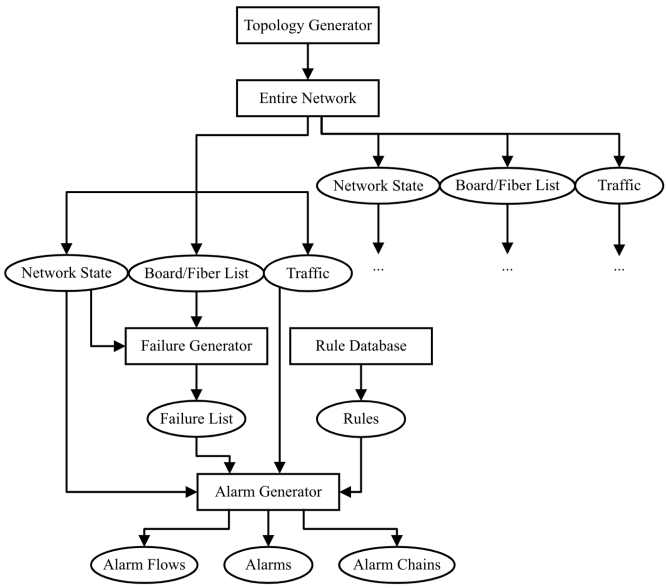


Fig. 6: Simulator Architecture

A. Rule Database

The rule database is stored in MySQL and accessed by the simulator through the PyMySQL library. It provides information about the propagation behavior of each alarm type and failure type. To make a query, the simulator needs to input "Board" and "Receive Detect Event", after which MySQL will output the values of "Output Board", "Output Type", and "Output". Here is an explanation of each attribute:

- *Board* is the location type of the input failure/alarm. It is required because the occurrence of an alarm type on different location types may result in different outcomes. For example, if an OMS_LOS_A alarm is detected by an OD, it will propagate an OCh_LOS_P alarm to the OTU. However, if the same alarm type is detected by an OM, it will trigger an OMS_SSF_E alarm on the OD.
- *Receive Detect Event* refers to the type of the input failure/alarm.
- *Output Board* is the location type of the alarm that will be triggered.
- *Output Type* specifies the direction of propagation, including transmit downstream, transmit upstream and locally report. If the output type is transmit downstream, the alarm propagates along the direction of fibers. Conversely, if the output type is transmit upstream, the alarm propagates in the opposite direction of fibers. Locally report means that the alarm/failure triggers another alarm at the same location.
- *Output* is the type of the alarm that will be triggered.

Table II are two sample rules in the rule database. The first rule is a failure $\rightarrow$ alarm rule. If OA_OD detects a

TABLE II: Example of Rules

Board	Receive Detect Event	Output Board	Output Type	Output
OA_OD	fiber cut	OD	transmit downstream	OMS_LOS_A
FIU	OTS_LOS_B	FIU	transmit downstream	OTS_LOS_A
		OD	transmit downstream	OMS_SSF_B

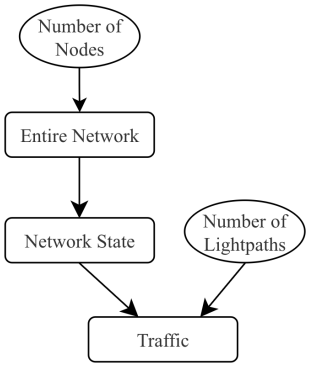


Fig. 7: Flowchart of Static Topology Generator

fiber cut failure, it will transmit an OMS_LOS_A alarm along the direction of fibers to the next OD. The second rule is a alarm $\rightarrow$ alarm rule. After an OTS_LOS_B is reported on FIU, it will trigger two alarms: an OTS_LOS_A on FIU and an OMS_SSF_B on OD. Let a (location type, failure/alarm type) pair be a rule event in the rule database. It is noticeable that a rule event can trigger multiple other rule events, and conversely, multiple rule events can trigger the same rule event.

B. Topology Generator

1) Static Topology Generator

To generate a static topology, two parameters are required: *Number of Nodes*, which determines the size of the topology, and *Number of lightpaths*, representing the traffic volume. The flowchart of the static topology generator is shown in Figure 7, and the step-by-step procedures are outlined below:

- A random node-level directed graph is generated with the help of NetworkX package. The ratio of ROADMs is randomly selected between 0.85 and 0.95.
- Since static topology has only one network state, there is no necessity to select a subset of nodes and connections from the entire network. Therefore, the network state is regarded as identical to the entire network.
- Next, a number of lightpaths will be generated based on the network state, where the pseudo-code is provided in Algorithm 1. The number of available OTUs should be significantly larger than the number of required lightpaths.

2) Dynamic Topology Generator

The process of generating a dynamic topology is similar to that of a static topology:

- A random node-level directed graph is generated to form the entire network.
- To mimic the real-life scenario where lightpaths are added and removed over time, we first establish the list of traffics, and then construct the network state based on each traffic. The pseudo-code is provided in Algorithm 2 and the flowchart is given in Figure 8.

Algorithm 1 Generate Lightpaths

Input: network state NS, number of lightpath N_L
Output: N_L unique lightpaths
  $O =$ all OTUs in NS
  $L = \emptyset$
  for $k = 1$ to N_L **do**
    b_s = randomly select an OTU from O
    n = node location of b_s
    $l_n =$ node-level lightpath with start node n
    while $\exists$ a neighbour r of n **do**
      add r to l_n
      $n = r$
    end while
    b_e = randomly select an OTU from N
    $l_b =$ board-level lightpath given l_n, b_s, b_e
    add l_b to L
    remove b_s, b_e from O
  end for
  return L

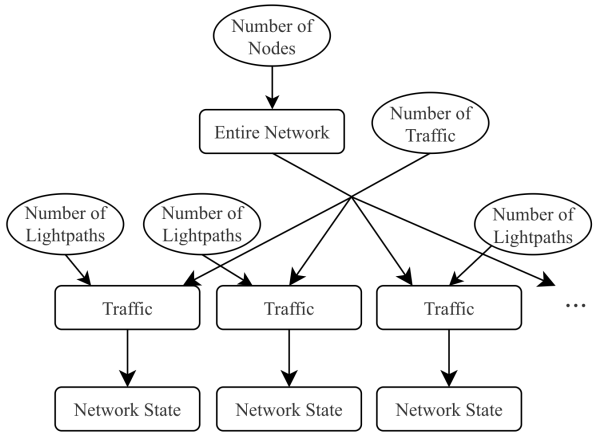


Fig. 8: Flowchart of Dynamic Topology Generator

Algorithm 2 Generate Traffics

Input: entire network EN, number of traffic N_T, number of lightpaths for each traffic $N_1, N_2, \cdots N_{N_T}$
Output: N_T unique traffics $T_1, T_2, \cdots, T_{N_T}$ and N_T network states $NS_1, NS_2, \cdots, NS_{N_T}$
  $T_1 =$ Generate Lightpaths(EN, N_1)
  $NS_1 =$ all nodes and connections used in T_1
  for $k = 2$ to N_T **do**
    T_k = randomly remove some lightpaths from T_{k-1}
    $NS_k =$ all nodes and connections used in T_k
    NS_k = randomly add some nodes and connections from $EN - NS_k$
    $T_k = T_k +$ Generate Lightpaths(NS_k, $N_k - |T_k|$)
  end for

C. Failure Generator

The simulator can generate failures based on either a list of location types or a list of locations. To produce a list of failures, the simulator will first generate a list of time instances, indicating when each failure occurs. By default, we assume that each failure occurs at the beginning of each minute. Moreover, the number of failures per minute should be specified with the default set to 1. Then, the simulator will select a list of locations if not provided. With a location type, the generator will first determine the eligible node type. For example, if the location type is OM, then only ROADMs are valid. Subsequently, a random node with the required node type(s) is selected from the network state, and a random location with the required location type is chosen from the node. After determining the location, the failure type can be derived. Currently, there are only two failure types: fiber cut and board faulty, which occur on fiber and board correspondingly. Since each location can fail only once, the ID of the failure is set to the ID of the location. Now that all the attributes are prepared, the list of failures can be constructed.

D. Alarm Generator

The architecture of the alarm generator comprises two buffers, one event processor and two containers. Buffer A stores all the failure events that haven't been processed, while buffer B stores all the events resulting from a single failure event. The event processor takes an event as input and outputs all the alarms triggered by the event. The two containers store all the alarms and alarm flows generated. The steps to generate alarms given a list of failures are outlined below:

Initially, buffer A contains all the failures, while buffer B is empty. Then, the first failure F_1 in buffer A is moved to buffer B for processing.

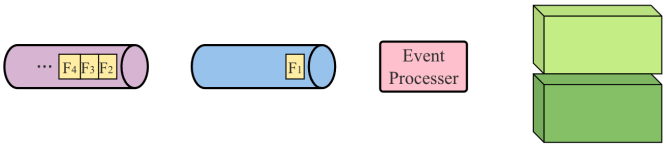


Fig. 9: Alarm Generator (Status 1)

F_1 is input to the event processor, which then returns alarms A_1 and A_2. Then the alarms, along with their corresponding alarm flows $F_1 \rightarrow A_1$ and $F_1 \rightarrow A_2$, are added to the containers. Additionally, a copy of each alarm is appended to buffer B, as they may trigger further alarms.

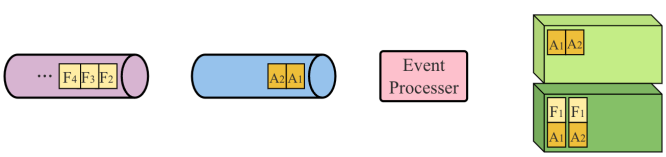


Fig. 10: Alarm Generator (Status 2)

Then, the next event in buffer B is passed to the event processor, and the new outputs are added to both the containers and buffer B.

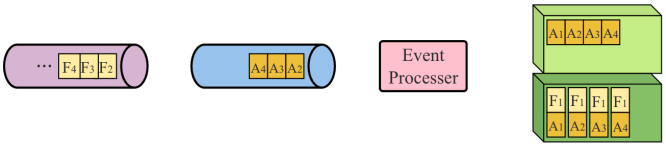


Fig. 11: Alarm Generator (Status 3)

If no alarms are returned by the event processor, nothing will be added to the containers and buffer B. Keep processing the events until buffer B is empty. Now we have all the alarms and alarm flows that result from F_1. Then the next failure in buffer A is moved to buffer B, and the aforementioned steps are reiterated. The alarm generation process terminates until buffer A and B are both empty. At this point, the containers hold all alarms and alarm flows generated from the list of failures.

E. Event Processor

Upon receiving an event e_i as input, the event processor will first extract the board type from its location. Subsequently, it will input this information, along with e_i's alarm type, into the rule database. Then the rule database will return a list of tuple (*Output Board*, *Output Type*, *Output*), as elaborated in Section III-A.

For each tuple retrieved, the event processor will generate a list of alarm events. The number of alarm events produced corresponds to the number of locations that meet the specified criteria. For example, if the retrieved tuple is (Transmit downstream, OTU, OCh_LOS_P), and there are two OTUs located downstream of e_i, then two alarm events will be created, each corresponding to one of these locations. The subsequent content in this section expounds on the procedure for generating each attribute in the alarm event.

- The *alarm type* will match the value of the *Output* specified in the tuple.
- The *location* is determined by all the values within the tuple, alongside the board-level network state. There are three cases:
  - Case 1: *Output Type* is "to board".
    A single alarm event will be generated, and its *location* will be identical to that of e_i.
  - Case 2: *Output Type* is "transmit downstream/upstream", *Output* is an OTS/OMS alarm.
    The processor will search for all boards in the network state that fulfill the following criteria:
    - The board possesses a board type matching *Output Board*.
    - There exists a path between e_i and the board in the direction indicated by the *Output Type*.
    - No other boards with the *Output Board* type are located along the path.

    To optimize runtime, the processor will not manually search through all boards in the network state. Instead, it will conduct a breadth-first search starting from the location of e_i.
    A queue is utilized, where boards are dequeued sequentially at each step, and their neighbors are

inspected to determine whether they should be enqueued or discarded, according to Algorithm 3. It is worth noting that this algorithm only addresses the scenario where the *Output Type* is "transmit downstream." In the event of "transmit upstream," the predecessors of each board will be explored instead.

Algorithm 3 Search Boards

Input: network state NS, e_i's location b_i
Output: $boardlist$

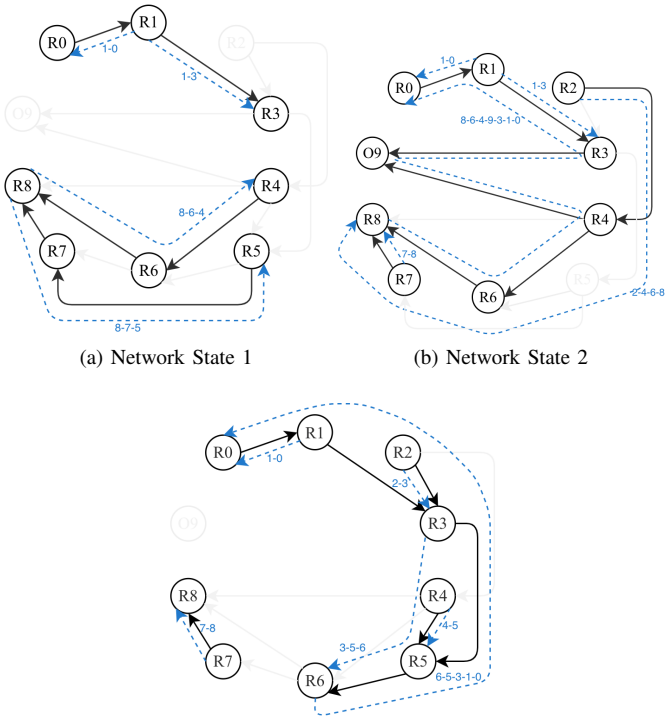
```
Q = Queue(b_i)
boardlist = ∅
while Q is not empty do
    newQ = Queue()
    while Q is not empty do
        b = Q.dequeue()
        for n in b.successors do
            if n.type == Output Board then
                boardlist.add(n)
            else
                newQ.enqueue(n)
            end if
        end for
    end while
    Q = newQ
end while
```

  - Case 3: *Output Type* is "transmit downstream", *Output* is an OCh alarm.
    The processor will iterate through all lightpaths, specifically identifying those that traverse the faulty board and subsequently pass through the location of e_i. It will then return the endpoints (OTUs) corresponding to these identified lightpaths. In the context of OCh alarm, there is no scenario where the *Output Type* is "transmit upstream".

- To simulate the real-life scenario, a random integer will be selected to represent the time required for an event to trigger another. To be more specific, for any event e_o triggered by e_i, its occurrence *time* will be the time of e_i plus a randomly generated duration ranging from 1 to 10 seconds.
- The alarm container will keep track of the number of alarms it receives and increment the *ID* by 1 each time a new alarm is inputted.
- The alarm event will inherit the *failure ID* from the event that triggers it.
- *Is root* will be set to true if the alarm type of the parent event e_i indicates a failure; otherwise, it will be set to false.
- By default, *Is noisy* is set to false unless explicitly modified.

IV. Case Studies

In this section, a dynamic topology with three network states will be generated to evaluate the performance of the topology generator. Furthermore, to validate the functionality of the failure generator and alarm generator, a single-failure test case will be introduced.



(a) Network State 1 (b) Network State 2

(c) Network State 3

Fig. 12: Network States in Dynamic Topology

TABLE III: Parameters Input to Topology Generator

	Number of Nodes	Number of Lightpaths
Network State 1	10	4
Network State 2		5
Network State 3		6

TABLE IV: Traffic in Network State 1

(OTU44)→ROADM1→ROADM0→(OTU87)
(OTU72)→ROADM8→ROADM7→ROADM5→(OTU3)
(OTU2)→ROADM1→ROADM3→(OTU17)
(OTU104)→ROADM8→ROADM6→ROADM4→(OTU83)

TABLE V: Traffic in Network State 2

(OTU44)→ROADM1→ROADM0→(OTU87)
(OTU8)→ROADM8→ROADM6→ROADM4→OLA9→ROADM3→ ROADM1→ROADM0→(OTU3)
(OTU18)→ROADM2→ROADM4→ROADM6→ROADM8→(OTU5)
(OTU106)→ROADM7→ROADM8→(OTU41)
(OTU2)→ROADM1→ROADM3→(OTU17)

Given the inputs outlined in Table III, the topology generator returns a topology with three network states, depicted in Figure 12, alongside the corresponding traffics delineated in Tables IV, V, and VI. It is noticeable that each network state utilizes a subset of nodes and connections within the entire network. Furthermore, the traffics exhibit duplicate lightpaths, simulating scenarios where lightpaths are added and removed over time.

TABLE VI: Traffic in Network State 3

(OTU106)→ROADM7→ROADM8→(OTU41)
(OTU16)→ROADM6→ROADM5→ROADM3→ ROADM1→ROADM0→(OTU43)
(OTU90)→ROADM2→ROADM3→(OTU77)
(OTU44)→ROADM1→ROADM0→(OTU87)
(OTU110)→ROADM4→ROADM5→(OTU45)
(OTU2)→ROADM3→ROADM5→ROADM6→(OTU1)

TABLE VII: Alarms in Network State 1

Alarm type	Location	Time	ID	Failure ID	Is root	Is noisy
board faulty	ROADM3-FIU2	06:00:00	61	61	false	false
OTS_A_P	ROADM3-FIU2	06:00:07	188	61	true	false

TABLE VIII: Alarms in Network State 2

Alarm type	Location	Time	ID	Failure ID	Is root	Is noisy
board faulty	ROADM3-FIU2	06:00:00	29	29	false	false
OTS_LOS_C	OLA9-FIU1	06:00:02	204	29	true	false
OTS_A_P	ROADM3-FIU2	06:00:10	205	29	true	false
OTS_BDI_A	ROADM3-FIU2	06:00:10	206	29	false	false
OMS_LOS_A	ROADM4-OM1	06:00:08	207	29	false	false
OMS_SSF_P	ROADM4-OD1	06:00:08	208	29	false	false
OTS_PMI	OLA9-FIU2	06:00:06	209	29	false	false
OTS_BDI	ROADM3-FIU1	06:00:13	210	29	false	false
OMS_BDI	ROADM3-OM1	06:00:13	211	29	false	false
OMS_SSF	ROADM4-OD1	06:00:19	212	29	false	false
OMS_SSF_E	ROADM6-OD1	06:00:14	213	29	false	false

TABLE IX: Alarms in Network State 3

Alarm type	Location	Time	ID	Failure ID	Is root	Is noisy
board faulty	ROADM3-FIU2	06:00:00	50	50	false	false
OTS_LOS_C	ROADM5-FIU1	06:00:06	218	50	true	false
OTS_A_P	ROADM3-FIU2	06:00:03	219	50	true	false
OTS_BDI_A	ROADM3-FIU2	06:00:13	220	50	false	false
OTS_BDI_A	ROADM4-FIU2	06:00:15	221	50	false	false
OMS_LOS_A	ROADM5-OM1	06:00:09	222	50	false	false
OMS_SSF_P	ROADM5-OD1	06:00:10	223	50	false	false
OTS_PMI	ROADM5-FIU2	06:00:13	224	50	false	false
OTS_BDI	ROADM3-FIU1	06:00:22	225	50	false	false
OMS_BDI	ROADM3-OM1	06:00:20	226	50	false	false
OMS_SSF	ROADM5-OD1	06:00:21	227	50	false	false
OTS_BDI	ROADM4-FIU1	06:00:17	228	50	false	false
OMS_BDI	ROADM4-OM1	06:00:20	229	50	false	false
OMS_SSF	ROADM5-OD1	06:00:24	230	50	false	false
OMS_SSF_E	ROADM6-OD1	06:00:11	231	50	false	false
OCh_LOS_P	ROADM6-OTU1	06:00:20	232	50	false	false
OCh_LOS_P	ROADM6-OTU1	06:00:23	233	50	false	false
OCh_LOS_P	ROADM6-OTU1	06:00:30	234	50	false	false
OCh_LOS_P	ROADM6-OTU1	06:00:21	235	50	false	false

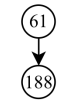


Fig. 13: Alarm Correlation Tree in Network State 1

To investigate the influence of network state and traffic on alarm propagation, the failure event with the same location is applied to all three network states. The outcomes are presented in Tables VII, VIII, IX and Figures 13, 14, 15.

We can observe that a slight alteration in network state and traffic can yield significant variations in the alarms generated. As depicted in Figure 16, a board faulty on FIU results in two root alarm events. While the FIU-OTS_A_P rule event

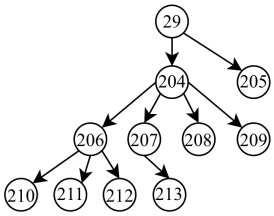


Fig. 14: Alarm Correlation Tree in Network State 2

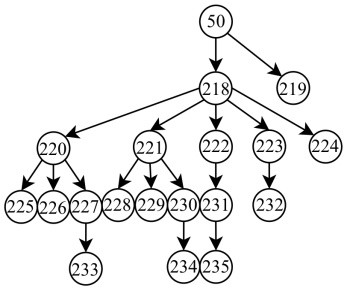


Fig. 15: Alarm Correlation Tree in Network State 3

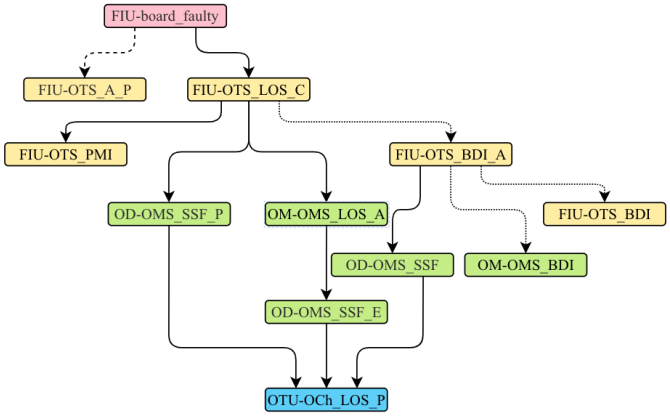


Fig. 16: Rule Events for FIU Board Faulty

is successfully triggered across all three network states, the FIU-OTS_LOS_C event fails to be invoked in network state 1. This is because there is no node located downstream of R3, thus halting the propagation of subsequent alarms.

To compare the alarms generated in network state 2 and 3, we can first examine the physical placement of nodes and connections in these two network states. In comparison to network state 2, network state 3 removes node O9 and introduces node R5. Additionally, connections R2 → R4, R3 → O9, R4 → O9, R4 → R6 and R6 → R8 are removed, while connections R2 → R3, R3 → R5, R4 → R5 and R5 → R6 are added. As the downstream node of R3 changes from O9 to R5, the root alarm OTS_LOS_C is reported on different locations in these two network states. Furthermore, in network state 3, R5 has two upstream nodes, R3 and R4, whereas O9 in network state 2 has only one upstream node, R3. Consequently, the OTS_LOS_C alarm triggers an additional OTS_BDI_A alarm in network state 3.

Another difference between the alarm sets in these two network states lies in the absence of OCh alarms in net-

work state 2. According to Figure 16, the rule events capable of triggering OCh alarms are OD-OMS_SSF_P, OD-OMS_SSF_E and OD-OMS_SSF, which occur on ROADM4-OD1 and ROADM6-OD1 in network state 2. However, although lightpath$_{8-6-4-9-3-1-0}$ traverses the faulty board and these two boards respectively, the lightpath passes through these boards before encountering the faulty board. Hence, no OCh alarms will be generated. In contrast, these rule events occur on ROADM5-OD1 and ROADM6-OD1 in network state 3, and there is lightpath$_{3-5-6}$ traversing through these boards and the faulty board in the desired order. Consequently, OCh alarms are successfully triggered by these rule events.

V. Conclusion

This paper presents the design and implementation of a simulator aimed at replicating alarm propagation behavior across diverse network topologies. Nevertheless, it is imperative to acknowledge the subsequent limitations:

- While the simulator effectively propagates alarms across OTS, OMS, and OCh layers, it currently overlooks the digital layers. Future enhancements should encompass the digital layers within the simulator to provide a more comprehensive representation of OTN.
- Manual editing of the rule database is currently required, which is time-consuming and prone to errors. Additionally, the rule database lacks consideration for the many-to-one case, where a rule event may be triggered by multiple rule events simultaneously. To address these shortcomings, future updates of the simulator could incorporate a rule generator capable of efficiently generating more complex rules.
- The current simulation framework lacks support for a sufficient range of node types and board types, limiting its ability to accurately simulate the complexities inherent in OTN. Expansion of the simulator to include a broader node and board types would enable more realistic modeling of OTN environments.

References

[1] "G.709: Interfaces for the optical transport network," International Telecommunication Union, ITU-T Recommendation, 2020. [Online]. Available: https://www.itu.int/rec/T-REC-G.709-202006-I/en2020

[2] "Ethernet Interfaces User Guide for Routing Devices," 2023. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/interfaces-ethernet/topics/topic-map/ethernet-otn-options-overview.html

[3] Z. Li, P.-H. Ho, Y. Jiao, B. Li, and Y. You, "Design of an OTN-based Failure/Alarm Propagation Simulator," in *2022 International Conference on Networking and Network Applications*, 2022, pp. 1–5.

Xiangzhu Lu (x244lu@uwaterloo.ca) received a B.S. degree from the University of Waterloo in 2022 and a M.S. degree from the University of Waterloo in 2024.

Yan Jiao (y42jiao@uwaterloo.ca) received a B.S. degree from China University of Geosciences Beijing in 2018 and a M.S. degree from the University of Waterloo in 2020. She is pursuing a Ph.D. degree at the University of Waterloo.

Pin-Han Ho (p4ho@uwaterloo.ca) (Fellow, IEEE) is currently a Full Professor in the Department of Electrical and Computer Engineering, University of Waterloo. He is the author/co-author of over 400 refereed technical papers, several book chapters, and the co-author of two books on Internet and optical network survivability. His current research interests cover a wide range of topics in broadband wired and wireless communication networks, including wireless transmission techniques, mobile system design and optimization, and network dimensioning and resource allocation.