Programmer's Manual for Multi-Level Error Detection (MLED) Architecture

Prateek Jain

January 23, 2025

Contents

Chapter 1

Introduction

1.1 Overview of MLED

The MLED is structured with n levels, where $n \geq 3$ and each level i can consist of j layers. The layer L_{ij} represents the layer located at level i with identifier j, referred to as the LayerID. Each layer L_{ij} is governed by a specific policy P_{ij}, which dictates the operations of the layer over its scope. The scope of a layer is defined as the portion(s) of the network over which it operates. Portion of a network refers to the successive part of the physical network between the source and the destination node having the same network characteristics, such as noise interference, that influences the Bit Error Rate (BER) of that part. Layers at level i have smaller scope than the layers at level $i+1$. For example, in Figure 1.1, each physical link between two consecutive nodes represents a part of the network with a distinct BER. The scope of layer L_{21} spans from node 1 to node 3. This scope includes two portions of the network: the portion between node 1 and node 2, and the portion between node 2 and node 3.

In general, an MLED framework is denoted as MLED(n, P), where n denotes the maximum number of levels and P denotes the set of policies such that $P_{ij} \in P$. The value of n, j and the policy to be used at each layer L_{ij} in MLED is determined by the admissible UEP (γ) specified by the user.

Now, consider a file transfer example shown in Figure 1.1 where the MLED framework with four levels, MLED$(4, P)$, is started by an application at the source to transfer a file to the destination. Both sending and receiving applications are marked with white squares. In the diagram, each horizontal band extending from the source node to the destination node represented by a different color is a different level of the MLED framework. The levels are numbered in a sequentially decreasing order starting from the top level that is represented by n . Each level has a distinct set of MLED processes shown by the colored squares.

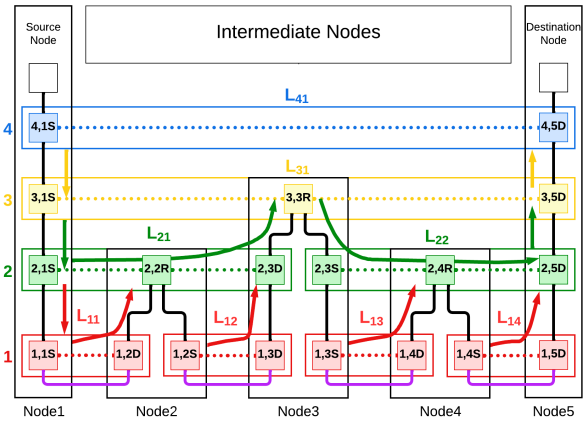

Figure 1.1: MLED$(4, P)$ configured to run on 5 nodes.

Each process is named according to the level it belongs to, the node it operates on, and an identifier (S, D, or R) to specify whether the process functions as a Source, Destination, or Relay. For example, a process named "3,1S" indicates that it is a source process running on node 1 at level 3. These processes perform functions to realize the MLED. The vertical rectangles group the MLED processes at different

levels that run on the same node. The configuration shown in Figure 1.1 has 5 nodes. Each physical node is connected to its neighboring nodes via a physical link or an overlay link in unbalanced MLED.

Within each level, different layers are represented by colored horizontal rectangles. Each layer L_{ij} starts and ends with its own source (S) and destination (D) processes, referred to as LayerSource(L_{ij}) and LayerDestination(L_{ij}), abreviated as LS(L_{ij}) and LD(L_{ij}), to differentiate them from the source and destination nodes between which the file transfer takes place. Depending on the level and the position of the layer, it may or may not contain a relay process. For instance, in Figure 1.1, the processes (3,1S), (3,3R), and (3,5D) represent the LS(L_{31}), the relay process, and the LD(L_{31}), respectively, for the layer L_{i31}. Similarly, processes (4,1S) and (4,1D) are the LS(L_{41}) and the LD(L_{41}), respectively, for the layer L_{41} but it has no relay process.

The layer specific policy P_{ij} defines various parameters and configurations that may include routing, addressing, congestion control, flow control or any other communication functionality. In this paper, we focus on the error detection policy.

Each MLED layer L_{ij} represents a logical connection between its LS(L_{ij}) and LD(L_{ij}) process. This logical connection is realised by one or more virtual links represented by horizontal dotted lines in Figure 1.1, connected by the relay process.

1.2 File Transfer in MLED

The file transfer process in MLED starts at level n. The fundamental operational unit of MLED is the MLED process. Various MLED processes operate to facilitate this transfer:

1.2.1 LS(L_{n1}) and LD(L_{n1})

At level n there is only one layer L_{n1}. The LS(L_{n1}) and LD(L_{n1}) enable inter-process communication (IPC) between application layer processes.

Given the likelihood of handling very large files that may not be stored locally, LS(L_{n1}) reads files in chunks from a remote disk. It starts a server locally or remotely that streams the file in chunks. Based on its payload length, the LS(L_{n1}) receives the file to be transferred in chunks, and forwards these chunks to the underlying layer at level $(n-1)$. Layer L_{n1} could implement its own error detection policy for the chunks it handles. The LD(L_{n1}) collects the chunks from the lower layer at level $(n-1)$, reassembles them according to its payload length, and writes data to the disk or passes it to the destination application process.

If *verifyWholeFileHash* is enabled, then LS(L_{n1}) conveys the SHA256 hash of the whole file to the LD(L_{n1}). Once the file is written to the disk successfully, LD(L_{n1}) computes the hash of transferred file and compares it with the received hash value to verify the integrity of the file transfer. This check acts as an additional error detection mechanism over and above the layer specific error detection policies used in MLED.

1.2.2 LS(L_{ij}) and LD(L_{ij}), $i < n$

Except for level n, the LS(L_{ij}) receives the data from the layer at level $(i+1)$, splits the data according to its payload length, calculates a checksum according to the error detection policy P_{ij}, encapsulates the data and the checksum to create the *Protocol Data Unit (PDU)* for layer L_{ij} and pushes it down to the layer at level $(i-1)$, except at level 1. At level 1, the LS(L_{1j}) transmits the data onto the underlying physical network.

At level 1, the LD(L_{1j}) on the next node receives the data from the physical link. At level $i > 1$, the LD(L_{ij}) receives the data from layer at level $(i-1)$, reassembles this data to form its PDU, decapsulates the checksum from the data, and verifies this checksum according to the error check policy P_{ij}. If no errors are detected in the data, it sends an *acknowledgement (ACK)*, otherwise it sends a *Negative ACKnowledgement (NACK)* to its corresponding LS(L_{ij}) and requests for a retransmission of the PDU.

1.2.3 Relay Process and Virtual Links

For a layer L_{ij}, any process between the LS(L_{ij}) and LD(L_{ij}) is a relay process (R). In Figures 1.1, the process (2,2R), (3,3R) and (2,4R) are relay processes for layers L_{21}, L_{31} and L_{22} respectively. The relay process at level i connects two underlying layers by transferring the data from the LD($L_{(i-1)j}$) to the LS($L_{(i-1)(j+1)}$). With this data transfer, it establishes the functionality of the layer at level i that

governs the policy over a larger part of the underlying physical network, encompassing multiple portions of the physical network.

Except for the layer L_{n1} and the layers at level 1, each layer contains at least two virtual links. The virtual link connects the LS(L_{ij}) to a relay and this relay to the LD(L_{ij}). For example, in Figure 1.1 the layer L_{31} is a logical connection between LS(L_{31}) and LD(L_{31}) composed of two virtual links connecting LS(L_{31}) to relay process (3,3R) and relay process (3,3R) to LD(L_{31}).

1.2.4 File Transfer Example

To understand the MLED architecture, consider the file transfer from source node to the destination node in Figure 1.1. Level 4 has one layer L_{41} with LS(L_{41}) and LD(L_{41}). LS(L_{41}) starts a server and receives chunks according to its payload length. It forwards these chunks to layer L_{31} for delivery to the LD(L_{41}) on Node 5. The LS(L_{31}) splits the data according to the payload length of L_{31} and adds its own header creating its own PDU. This header includes a checksum, which is computed based on the error detection policy defined by P_{31} and the sequence number for this chunk. This PDU is passed to layer L_{21} to transfer it to LD(L_{31}) on Node 5 via the relay process (3,3R). Similarly, layer L_{21} is responsible for transporting the PDUs received from layer L_{31}. LS(L_{21}) encapsulates these PDUs within its own PDUs and forwards them to LD(L_{21}) through the relay process (2,2R). Finally, the layer L_{11} works with the physical links represented by the purple line, moving the PDUs of layer L_{11} with their own unique checks from one node to the next one. After LD(L_{11}) receives its PDU via the physical link, it decapsulates the PDU, validates the data with the attached checksum and sends an ACK or NACK to LS(L_{11}). Once verified as error-free, it discards the header of layer L_{11} from the PDU and passes the data to relay process (2,2R), which then forwards it to LS(L_{12}). Here relay process (2,2R) connects layer L_{11} and L_{12} to realize the functionality of layer L_{21}. This procedure repeats until LD(L_{21}) reassembles the incoming data chunks into a PDU of layer L_{21}, conducts the integrity check specific to policy P_{21} and sends an ACK/NACK to LS(L_{21}). LD(L_{21}) passes the verified data to the relay process (3,3R). Process (3,3R) forwards the data to LS(L_{22}) which in turn forwards it to the LD(L_{22}). Process (3,3R) relays the data from layer L_{21} to layer L_{22} realizing the functionality of layer L_{31}. When LD(L_{31}) forms its PDU by reassembling the data received from LD(L_{22}), it checks the PDU for errors in accordance with the error detection policy P_{31} and sends an ACK/NACK to LS(L_{31}). The data transfer sequence culminates at the destination node at level 4, where LD(L_{41}) writes the data to disk storage.

Chapter 2

Configuration File

The MLED architecture relies on a configuration file to set up and customize the system for file transfer operations. This configuration file is a JSON document that specifies critical parameters, policies, and settings for each component of the architecture. The configuration file enables users to define the behavior of nodes, layer managers, and error detection policies, providing flexibility and control over the MLED's operation. Understanding these settings is essential for deploying and troubleshooting the MLED architecture effectively.

Below is an explanation of the keys used in the configuration file:

Explanation of Configuration Keys

Key	Description
`FileName`	Name of the file to be transferred.
`ReadChunkSizeInKB`	Size of each chunk (in KB) when reading the file at level n.
`WriteChunkSizeInKB`	Size of each chunk (in KB) when writing the file at level n.
`MemoryUsageLimitInGB`	Maximum memory usage allowed (in GB) to buffer the chunks read fromt he file. The system must have at least this much free memory.
`PrefetchInputFile`	Boolean value indicating whether to prefetch the input file into memory before starting the transfer. The system must have free memory equal to the size of the file to be transferred. Otherwise, MLED automatically disables this option.
`MLEDServerPort`	Port used by the MLED central manager server to communicate with the workers.
`HeartBeatTimeInterval`	Time interval (in milliseconds) between heartbeat messages exchanged between the workers and the central manager server.
`HeartBeatTimeOut`	Timeout (in milliseconds) for heartbeat messages. In current implementation, no action is taken on timeout as we don't handle machine failures.
`RetryTimeInterval`	Time interval (in milliseconds) before retrying a failed connetion request between various MLED servers and clients.
`FacilityPortIP`	IP address of the remote server where the file is stored. Use the local IP if the file is on the stored locally.
`RemoteFilePath`	Path to the directory where file to be transferred is be stored. It could be a local file path or remote file path. This path must end with a '/' and not contain '.'.
`RemoteSSHKey`	Name of the SSH key used for accessing the remote file. If the file is stored locally then you need an SSH key to SSH into local machine as well. This key should be stored in `$mled_dir_home/node_Node1/` and `$mled_dir_home/scripts/` directory. The name should not contain any '.'
`RemoteUsername`	Username for remote access or username on the local machine for local access.
`FileServerPort`	Port used for file reading operations remotely or locally.

Key	Description
`MaxPortsToTry`	In case the assigned `FileServerPort` is occupied, this parameter defines the maximum number of ports to attempt before failing.
`MaxAttemptsPerPort`	Maximum number of attempts to try to connect per port to read the file.
`Network`	Type of network protocol used (e.g., `tcp`). Currently, only `tcp` is supported as a valid protocol.
`Routing`	Type of routing policy (e.g., `static`). Currently, onl `static` routing is supported.
`MaxLayers`	Maximum number of layers in the architecture i.e., the value of n
`VerifyWholeFileHash`	Boolean indicating whether to verify the entire end-to-end file hash for integrity. It uses `SHA256` as the hashing algorithm.
`CreateFileToBypassCrcAndChecksum`	Boolean indicating whether MLED is running to get the metadata to create an *adversarial test file*. User should not set this value to `True`. This parameter is set by MLED automatically depending on the value of `BypassBothCrcAndChecksum` parameter.
`BypassBothCrcAndChecksum`	Boolean indicating whether the errors introduced by the adversarial error model are undetected by both CRC and checksum policy employed at different levels. If `False`, then MLED introduces adversarial errors that are undetected only at the level of introduction.
`Nodes`	Array of objects representing each node in the system, including their configuration, processes, and forwarding rules.
`NodeId`	Unique identifier for the node.
`NodeName`	Name of the node (e.g., `Node1`, `Node2`).
`NodeIp`	IP address of the data server running on node.
`NodeType`	Type of the node (e.g., `Source`, `Intermediate`, `Destination`).
`NodePort`	Port used by the data server for communication.
`forwardTo`	Array of objects indicating the next MLED worker to which data will be forwarded.
`processes`	Array of objects specifying the MLED processes running on the node, including details like process ID and level depth for checks.
`ProcessId`	Unique ID assigned to the each process on a node.
`LevelDepthUptoWhichCheckIsPerformed`	Starting from level 1 what is the depth up to which this process will go in the MLED hierarchy.
`ExecutorDetails`	This key represents the MLED processes that will run on this node. Each triplet separated by a hyphen (`-`) represents an MLED process. A triplet is structured as follows: the first part indicates the **node ID of the LS** for the process, the second part specifies whether the process is **enabled (E)** or **disabled (D)**, and the third part denotes the **node ID of the LD** of the process. It is important to note that all relay processes are always disabled, which means their status in the second part of the triplet will always be represented by `D`. For example, consider the sequence `1.D.2-2.E.3-1.D.3`. The first triplet, `1.D.2`, indicates that the process has **Node 1** as its LS, is **disabled**, and **Node 2** is its LD. The second triplet, `2.E.3`, signifies a process with **Node 2** as LS, is **enabled**, and **Node 3** is its LD. Finally, the third triplet, `1.D.3`, shows a process that has **Node 1** as LS, is **disabled**, and is has **Node 3** as its LD. This structured representation provides a clear and concise way to describe the state and LS and LD of processes at various layers within the MLED system, eliminating the need for *destination resolution* in the current implementation.

Key	Description
	The order in which these processes are defined is related to their visual position in MLED architecture. For each node, we define processes from level 1 to level n and on each level, we define them from left to right. For example, in Fig. 1.1, the configuration of Node 2 would define its processes in the following order: process(1,2D), process(1,2S) and process(2,2R).
`IsRelayProcess`	A boolean option to indicate if the processes defined in `ExecutorDetails` has a relay process or not. If set, this means that the last process defined in the key `ExecutorDetails` is the relay process.
`NodeAckIp`	IP address used by the ACK/NACK server for acknowledgment messages.
`NodeAckPort`	Port used by the ACK/NACK server for acknowledgment messages.
`forwardAckTo`	Array of objects indicating the next MLED worker to which `ACK/NACK` will be forwarded.
`LayerManagers`	Array of objects representing layer managers, their configurations, such as the `NodeID`, `NodeName`, `NopeIp` and `NopePort` of the node on which a particular manager would run and the routing tables for the layers that are managed by this layer manager.
`LayerManagerId`	Unique identifier for the layer manager, representing the level it manages.
`LayerStructure`	Represents the structure of the level managed by the layer manager. It essentially consists of node IDs separated by a '.' (dot), which indicate a virtual link between processes running on these nodes at the level managed by the layer manager.
`SubLayerDetails`	Array of objects describing the details of each layer at the level manged by the layer manager, such as window size, check policy, parameter, chunk size, PER, and cross-traffic generation. Cross-traffic generation parameters allow precise control over the generation and management of cross traffic in the system, enabling customization based on specific requirements. Cross traffic could be used for simulating multiple MLED flows through a particular source process to study the impact of increasing processing load on MLED.
`NodeDetails`	This array defines the LS and LD of the layer on which these particular parameters defined in `SubLayerDetails` would be applied.
`GenerateCrossTraffic`	A boolean parameter that determines whether cross traffic should be generated. If set to `true`, cross traffic will be enabled; if set to `false`, it will be disabled.
`CrossTrafficGenerationRateInBps`	Specifies the rate at which cross traffic is generated, measured in bytes per second (Bps).
`NumberofInstancesOfCrossTraffic`	Indicates the number of threads or instances responsible for generating cross traffic. Each instance will generate traffic at the rate specified by `CrossTrafficGenerationRateInBps`.
`routingTable`	Array of objects defining the routing rules for the layer manager, including source, destination, and next-hop details. The `Priority` key in the `routingTable` is not utilized in the current implementation. However, it is reserved for potential future use, where it could be employed to prioritize one MLED path over another. At present, the implementation only supports a single path, so prioritization is not necessary or applicable.

2.1 Example Main Configuration File

Listing 2.1: Main Configuration File created by the user to customize MLED

```
1
2 {
3   "FileName": "100MB",
4   "ReadChunkSizeInKB": 1024,
5   "WriteChunkSizeInKB": 1024,
6   "MemoryUsageLimitInGB": 10,
7   "PrefetchInputFile": false,
8   "MLEDServerPort": 50000,
9   "HeartBeatTimeInterval": 1000,
10  "HeartBeatTimeOut": 10000,
11  "RetryTimeInterval": 1000,
12  "FacilityPortIP": "10.52.0.16",
13  "RemoteSSHKey": "mledcpp",
14  "RemoteUsername": "cc",
15  "FileServerPort": 51000,
16  "MaxPortsToTry": 10,
17  "MaxAttemptsPerPort": 5,
18  "Network": "tcp",
19  "Routing": "static",
20  "MaxLayers": "3",
21  "VerifyWholeFileHash": true,
22  "CreateFileToBypassCrcAndChecksum": false,
23  "BypassBothCrcAndChecksum": false,
24  "Nodes": [
25    {
26      "NodeId": 1,
27      "NodeName": "Node1",
28      "NodeIp": "10.52.0.16",
29      "NodeType": "Source",
30      "NodePort": 8000,
31      "forwardTo": [
32        {
33          "Ip": "10.52.0.16",
34          "Port": 8100,
35          "NodeName": "Node2",
36          "NodeId": 2
37        }
38      ],
39      "processes": [
40        {
41          "ProcessId": 1,
42          "LevelDepthUptoWhichCheckIsPerformed": 3,
43          "ExecutorDetails": "1.E.2-1.E.3-1.E.5",
44          "IsRelayProcess": false
45        }
46      ],
47      "NodeAckIp": "10.52.0.16",
48      "NodeAckPort": 9000,
49      "forwardAckTo": []
50    },
51    {
52      "NodeId": 2,
53      "NodeName": "Node2",
54      "NodeIp": "10.52.0.16",
55      "NodeType": "Intermediate",
56      "NodePort": 8100,
57      "forwardTo": [
58        {
59          "Ip": "10.52.0.16",
60          "Port": 8200,
61          "NodeName": "Node3",
62          "NodeId": 3
63        }
64      ],
65      "processes": [
66        {
67          "ProcessId": 1,
68          "LevelDepthUptoWhichCheckIsPerformed": 1,
69          "ExecutorDetails": "1.E.2-2.E.3-1.D.3",
```

```
        "IsRelayProcess": true
      }
    ],
    "NodeAckIp": "10.52.0.16",
    "NodeAckPort": 9500,
    "forwardAckTo": [
      {
        "Ip": "10.52.0.16",
        "Port": 9000,
        "NodeName": "Node1",
        "NodeId": 1
      }
    ]
  },
  {
    "NodeId": 3,
    "NodeName": "Node3",
    "NodeIp": "10.52.0.16",
    "NodeType": "Intermediate",
    "NodePort": 8200,
    "forwardTo": [
      {
        "Ip": "10.52.0.16",
        "Port": 8300,
        "NodeName": "Node4",
        "NodeId": 4
      }
    ],
    "processes": [
      {
        "ProcessId": 1,
        "LevelDepthUptoWhichCheckIsPerformed": 2,
        "ExecutorDetails": "2.E.3-3.E.4-1.E.3-3.E.5-1.D.5",
        "IsRelayProcess": true
      }
    ],
    "NodeAckIp": "10.52.0.16",
    "NodeAckPort": 9200,
    "forwardAckTo": [
      {
        "Ip": "10.52.0.16",
        "Port": 9500,
        "NodeName": "Node2",
        "NodeId": 2
      }
    ]
  },
  {
    "NodeId": 4,
    "NodeName": "Node4",
    "NodeIp": "10.52.0.16",
    "NodeType": "Intermediate",
    "NodePort": 8300,
    "forwardTo": [
      {
        "Ip": "10.52.0.16",
        "Port": 8400,
        "NodeName": "Node5",
        "NodeId": 5
      }
    ],
    "processes": [
      {
        "ProcessId": 1,
        "LevelDepthUptoWhichCheckIsPerformed": 1,
        "ExecutorDetails": "3.E.4-4.E.5-3.D.5",
        "IsRelayProcess": true
      }
    ],
    "NodeAckIp": "10.52.0.16",
    "NodeAckPort": 9300,
    "forwardAckTo": [
      {
```

```
          "Ip": "10.52.0.16",
          "Port": 9200,
          "NodeName": "Node3",
          "NodeId": 3
        }
      ]
    },
    {
      "NodeId": 5,
      "NodeName": "Node5",
      "NodeIp": "10.52.0.16",
      "NodeType": "Destination",
      "NodePort": 8400,
      "forwardTo": [],
      "processes": [
        {
          "ProcessId": 1,
          "LevelDepthUptoWhichCheckIsPerformed": 3,
          "ExecutorDetails": "4.E.5-3.E.5-1.E.5",
          "IsRelayProcess": false
        }
      ],
      "NodeAckIp": "10.52.0.16",
      "NodeAckPort": 9400,
      "forwardAckTo": [
        {
          "Ip": "10.52.0.16",
          "Port": 9300,
          "NodeName": "Node4",
          "NodeId": 4
        }
      ]
    }
  ],
  "LayerManagers": [
    {
      "LayerManagerId": 1,
      "NodeId": 1,
      "NodeName": "node1",
      "NodeIp": "10.52.0.16",
      "NodePort": 7000,
      "LayerStructure": "1.2.3.4.5",
      "SubLayerDetails": [
        {
          "NodeDetail": [
            "1-S",
            "2-D"
          ],
          "WindowSize": 461,
          "CheckPolicy": "CRC",
          "CheckParameter": "0x04C11DB7",
          "ChunkSize": 65536,
          "PacketErrorRate": 0.1,
          "GenerateCrossTraffic": false,
          "CrossTrafficGenerationRateInBps": 9000,
          "NumberofInstancesOfCrossTraffic": 30
        },
        {
          "NodeDetail": [
            "2-S",
            "3-D"
          ],
          "WindowSize": 446,
          "CheckPolicy": "CRC",
          "CheckParameter": "0x04C11DB7",
          "ChunkSize": 65536,
          "PacketErrorRate": 0,
          "GenerateCrossTraffic": false,
          "CrossTrafficGenerationRateInBps": 10000,
          "NumberofInstancesOfCrossTraffic": 20
        },
        {
          "NodeDetail": [
```

```
            "3-S",
            "4-D"
          ],
          "WindowSize": 552,
          "CheckPolicy": "CRC",
          "CheckParameter": "0x04C11DB7",
          "ChunkSize": 65536,
          "PacketErrorRate": 0,
          "GenerateCrossTraffic": false,
          "CrossTrafficGenerationRateInBps": 11000,
          "NumberofInstancesOfCrossTraffic": 40
        },
        {
          "NodeDetail": [
            "4-S",
            "5-D"
          ],
          "WindowSize": 475,
          "CheckPolicy": "CRC",
          "CheckParameter": "0x04C11DB7",
          "ChunkSize": 65536,
          "PacketErrorRate": 0,
          "GenerateCrossTraffic": false,
          "CrossTrafficGenerationRateInBps": 12000,
          "NumberofInstancesOfCrossTraffic": 60
        }
      ],
      "routingTable": [
        {
          "Source": 1,
          "Destination": 5,
          "Priority": "1",
          "NextHopNodeId": 2
        },
        {
          "Source": 1,
          "Destination": 4,
          "Priority": "1",
          "NextHopNodeId": 2
        },
        {
          "Source": 1,
          "Destination": 3,
          "Priority": "1",
          "NextHopNodeId": 2
        },
        {
          "Source": 1,
          "Destination": 2,
          "Priority": "1",
          "NextHopNodeId": 2
        },
        {
          "Source": 2,
          "Destination": 5,
          "Priority": "1",
          "NextHopNodeId": 3
        },
        {
          "Source": 2,
          "Destination": 4,
          "Priority": "1",
          "NextHopNodeId": 3
        },
        {
          "Source": 2,
          "Destination": 3,
          "Priority": "1",
          "NextHopNodeId": 3
        },
        {
          "Source": 3,
          "Destination": 5,
```

```
            "Priority": "1",
            "NextHopNodeId": 4
          },
          {
            "Source": 3,
            "Destination": 4,
            "Priority": "1",
            "NextHopNodeId": 4
          },
          {
            "Source": 4,
            "Destination": 5,
            "Priority": "1",
            "NextHopNodeId": 5
          },
          {
            "Source": 2,
            "Destination": 1,
            "Priority": "1",
            "NextHopNodeId": 1
          },
          {
            "Source": 3,
            "Destination": 2,
            "Priority": "1",
            "NextHopNodeId": 2
          },
          {
            "Source": 3,
            "Destination": 1,
            "Priority": "1",
            "NextHopNodeId": 2
          },
          {
            "Source": 4,
            "Destination": 3,
            "Priority": "1",
            "NextHopNodeId": 3
          },
          {
            "Source": 4,
            "Destination": 2,
            "Priority": "1",
            "NextHopNodeId": 3
          },
          {
            "Source": 4,
            "Destination": 1,
            "Priority": "1",
            "NextHopNodeId": 3
          },
          {
            "Source": 5,
            "Destination": 1,
            "Priority": "1",
            "NextHopNodeId": 4
          },
          {
            "Source": 5,
            "Destination": 2,
            "Priority": "1",
            "NextHopNodeId": 4
          },
          {
            "Source": 5,
            "Destination": 3,
            "Priority": "1",
            "NextHopNodeId": 4
          },
          {
            "Source": 5,
            "Destination": 4,
            "Priority": "1",
```

```
          "NextHopNodeId": 4
        }
      ]
    },
    {
      "LayerManagerId": 2,
      "NodeId": 1,
      "NodeName": "node1",
      "NodeIp": "10.52.0.16",
      "NodePort": 7100,
      "LayerStructure": "1.3.5",
      "SubLayerDetails": [
        {
          "NodeDetail": [
            "1-S",
            "3-D"
          ],
          "WindowSize": 293,
          "CheckPolicy": "CRC",
          "CheckParameter": "0x9B",
          "ChunkSize": 131072,
          "PacketErrorRate": 0,
          "GenerateCrossTraffic": false,
          "CrossTrafficGenerationRateInBps": 17000,
          "NumberofInstancesOfCrossTraffic": 80
        },
        {
          "NodeDetail": [
            "3-S",
            "5-D"
          ],
          "WindowSize": 419,
          "CheckPolicy": "CRC",
          "CheckParameter": "0x9B",
          "ChunkSize": 131072,
          "PacketErrorRate": 0,
          "GenerateCrossTraffic": false,
          "CrossTrafficGenerationRateInBps": 20000,
          "NumberofInstancesOfCrossTraffic": 90
        }
      ],
      "routingTable": [
        {
          "Source": 1,
          "Destination": 5,
          "Priority": "1",
          "NextHopNodeId": 2
        },
        {
          "Source": 1,
          "Destination": 4,
          "Priority": "1",
          "NextHopNodeId": 2
        },
        {
          "Source": 1,
          "Destination": 3,
          "Priority": "1",
          "NextHopNodeId": 2
        },
        {
          "Source": 1,
          "Destination": 2,
          "Priority": "1",
          "NextHopNodeId": 2
        },
        {
          "Source": 2,
          "Destination": 5,
          "Priority": "1",
          "NextHopNodeId": 3
        },
        {
```

```
435           "Source": 2,
436           "Destination": 4,
437           "Priority": "1",
438           "NextHopNodeId": 3
439         },
440         {
441           "Source": 2,
442           "Destination": 3,
443           "Priority": "1",
444           "NextHopNodeId": 3
445         },
446         {
447           "Source": 3,
448           "Destination": 5,
449           "Priority": "1",
450           "NextHopNodeId": 4
451         },
452         {
453           "Source": 3,
454           "Destination": 4,
455           "Priority": "1",
456           "NextHopNodeId": 4
457         },
458         {
459           "Source": 4,
460           "Destination": 5,
461           "Priority": "1",
462           "NextHopNodeId": 5
463         },
464         {
465           "Source": 2,
466           "Destination": 1,
467           "Priority": "1",
468           "NextHopNodeId": 1
469         },
470         {
471           "Source": 3,
472           "Destination": 2,
473           "Priority": "1",
474           "NextHopNodeId": 2
475         },
476         {
477           "Source": 3,
478           "Destination": 1,
479           "Priority": "1",
480           "NextHopNodeId": 2
481         },
482         {
483           "Source": 4,
484           "Destination": 3,
485           "Priority": "1",
486           "NextHopNodeId": 3
487         },
488         {
489           "Source": 4,
490           "Destination": 2,
491           "Priority": "1",
492           "NextHopNodeId": 3
493         },
494         {
495           "Source": 4,
496           "Destination": 1,
497           "Priority": "1",
498           "NextHopNodeId": 3
499         },
500         {
501           "Source": 5,
502           "Destination": 1,
503           "Priority": "1",
504           "NextHopNodeId": 4
505         },
506         {
507           "Source": 5,
```

```
          "Destination": 2,
          "Priority": "1",
          "NextHopNodeId": 4
        },
        {
          "Source": 5,
          "Destination": 3,
          "Priority": "1",
          "NextHopNodeId": 4
        },
        {
          "Source": 5,
          "Destination": 4,
          "Priority": "1",
          "NextHopNodeId": 4
        }
      ]
    },
    {
      "LayerManagerId": 3,
      "NodeId": 1,
      "NodeName": "node1",
      "NodeIp": "10.52.0.16",
      "NodePort": 7300,
      "LayerStructure": "1.5",
      "SubLayerDetails": [
        {
          "NodeDetail": [
            "1-S",
            "5-D"
          ],
          "WindowSize": 325,
          "CheckPolicy": "checksum",
          "CheckParameter": "2",
          "ChunkSize": 262144,
          "PacketErrorRate": 0,
          "GenerateCrossTraffic": false,
          "CrossTrafficGenerationRateInBps": 22000,
          "NumberofInstancesOfCrossTraffic": 15
        }
      ],
      "routingTable": [
        {
          "Source": 1,
          "Destination": 5,
          "Priority": "1",
          "NextHopNodeId": 3
        },
        {
          "Source": 1,
          "Destination": 3,
          "Priority": "1",
          "NextHopNodeId": 3
        },
        {
          "Source": 3,
          "Destination": 5,
          "Priority": "1",
          "NextHopNodeId": 5
        },
        {
          "Source": 3,
          "Destination": 1,
          "Priority": "1",
          "NextHopNodeId": 1
        },
        {
          "Source": 5,
          "Destination": 1,
          "Priority": "1",
          "NextHopNodeId": 3
        },
        {
```

```
            "Source": 5,
            "Destination": 3,
            "Priority": "1",
            "NextHopNodeId": 3
          }
        ]
      }
  ]
}
```

Due to the complex structure of the configuration file required for customizing the MLED framework, users may find it challenging to manually create or modify the file accurately. To simplify this process, a graphical user interface (GUI) is available at `https://mled.bu.edu` to create configuration files for both balanced and unbalanced MLED architectures. This GUI streamlines the process by providing an interface to generate the configuration file with minimal effort. By leveraging the GUI, users can avoid potential errors and ensure the configuration aligns with their specific network requirements.