Fall 2014

User Manual

Media Delivery Network Simulator

Sponsor	**Ericsson**			
Point of Contact	Vladimir Katardjiev	Alvin Jude		
Faculty Advisor	Jia Zhang			
Team	Jeremy Fu	Jigar Patel	Vinay Kumar Vaili	Hao Wang

Version	Release Date	Revised By
0.1	Dec 6, 2014	Jeremy Fu, Jigar Patel, Vinay Kumar Vavili, Hao Wang
1.0	Dec 9, 2014	Jeremy Fu, Jigar Patel, Vinay Kumar Vavili

ERICSSON

Carnegie Mellon University

Table of Content

1. Introduction

The media delivery network (MDN) simulator is a distributed simulation framework that can be used to test and evaluate efficiency of media delivery network. The simulator generates and distributes actual traffic across the IP backbone network.

The following sections will give a brief introduction of how to use MDN simulator. The chapter 2 quickly walks through various components in the system. Chapter 3 instructs how to deploy the system and the resources to control the system. The input script of MDN simulator to generate a simulation is described in chapter 4. Finally, chapter 5 introduces the panel of MDN simulator.

2. Components of MDN Simulator

The MDN simulator consists of following main components of the simulator:

- Master
- Web Client
- Node Containers
- Nodes

In the following sections, we will give a brief overview of the functions and features of each of these components.

Master

This is the main process of the simulator that interacts with all the components in the simulator including the web client, node containers and the nodes. All components register with the master node. The master node hosts the message bus server that is used to pass control messages. And it also hosts the web client which provides control panel and visualization of the simulator.

Web Client

This is the user facing interface of the simulator. It is hosted by the master process of the simulator. It is hosted at:

Port: 8888
Path: /static/index.html

For example: http://localhost:8888/static/index.html

The web client has ability to upload a user input to the master. On the other hand, it renders to the front-end the topology of the current simulation that is running and gives immediate feedback to the user on the state of the network using node and edge labels in the topology graph.

Node Container

Each node container runs in an independent JVM process. It represents a single machine in the media delivery network chain. The node container should be regarded as a virtualized node in the media delivery network, whose functionalities can be software-defined. The functionalities

can be source, sink, relay or processing on media data, which will be elaborated in Node section.

Nodes

The nodes in MDN simulator represent some functionalities in the media delivery network chain. They are objects in node container process. The node container can host one or more of these node objects so that they can simulate various functions such as tree-based topology network and a peer-to-peer network.

Currently the following four node types have been defined and implemented:

- Source Node: The node that generates media data and sends it to downstream nodes. This is the first node in the media delivery chain.
- Processing Node: This node receives data from an upstream node, simulates some processing by consuming a configurable amount of CPU and memory and forwards the packet to a downstream node. It takes one input and emits one output.
- Relay Node: This node is used to broadcast data to downstream nodes. When a packet is received for a stream stream, it makes copies of that packet and sends to all downstream nodes for that stream.
- Sink Node: This node consumes the media data. It is the end of the media delivery chain

3. Deployment of MDN Simulator

Start MDN Simulator

There are 3 major components to start in order to make this simulator running.

1. Master: Run the master. The class path for master is: **edu.cmu.mdnsim.server.Master.java**. When the log shows similar information as follows you can start to proceed next procedure:
   [INFO] 20:28:11.086 { embedded.mdn-manager} - Now registered…

2. Web Client: Open a web browser (preferably Chrome) and go to **http://localhost:8888/static/index.html** . If you try to access the web client remotely, you can replace "localhost" with your master IP address in the URL. Please note that, when your page has been loaded, a log which shows that the web client has registered:
   Web client URI is warp://warp:gw//jFEnVgNfWMc
   If the log is missing, it means the web client doesn't get registered on Master. If you continue next step without having web client registered, you will not be able to talk to master. You must try to refresh your page until you get this message before you can continue next step. The team hasn't figured out where the bug is from.

3. NodeContainer(s): Run node container(s) one by one. The class path for node container is: **edu.cmu.mdnsim.nodes.Server.java**. The node container takes an argument: label:<label name>. Again, due the unresolved bug in registration, you need to ensure each node container you start has registered on master. This can be checked by whether all logs with labels you specify show in the browser, for example:
   [INFO] 09:47:30.441 {bedded.mdn-manager.master} - Register new node container label:orange from warp://warp:gw/mw8ZGbsVBg
   This log shows that a node container labeled with orange has registered on master.

Run Simulation

Once the Master and the node containers are started, we can start running simulations and even control simulations on the simulator. You can either choose to control the simulator from the browser which will be described in chapter 5, or you can directly talk the master at following resources to control the simulator:

- Start Flow
  Purpose: To start a new flow or add new flow to existing simulation.

Method: POST
Path: <master warp URI>/work_config

- Stop Flow
  Purpose: To stop existing flow.
  Method: DELETE
  URL: <master warp URI>/work_config

- Reset Simulation
  Purpose: To reset the entire simulation. Resetting the simulator will not kill the node containers. Node container will release all resources related to nodes within itself and kill corresponding threads as well.
  Method: DELETE
  URL: <master warp URI>/simulations

4. MDN Simulator Input Script

The MDN simulator takes a script called Work Specification which describes the flows and streams in a simulation. The Work Specification is a JSON formatted file and is used to specify the following information:

- SimID: The Simulation ID
- StreamList: The list of streams that are part of the simulation
- StreamID: For every stream in stream list, a StreamID for uniquely identifying a stream.
- DataSize: The DataSize specifies size of a stream to be transferred in bytes.
- KiloBitRate: The KiloBitRate specifies the transfer rate in kbps of the stream.
- FlowList: A list of flows that belong to a stream. Each unique path from the source to the sink represents a separate flow for that stream.
- NodeList: A list of NodeMaps that represent a unique path for a stream from the source to the sink.
- NodeMap: A single NodeMap represents a node in a flow. Key-value pairs in a NodeMap are properties of the node. The NodeMap within the NodeList are ordered from the sink to the source. In other word, the (i+1)th NodeMap in the NodeList represents the upstream node for ith NodeMap. Every (i-1)th NodeMap is the downstream node for the node at i. The sink NodeMap doesn’t have a downstream node and the source NodeMap doesn't have a upstream node.

The following examples will give more details on how to write a work specification for a MDN of your choice.

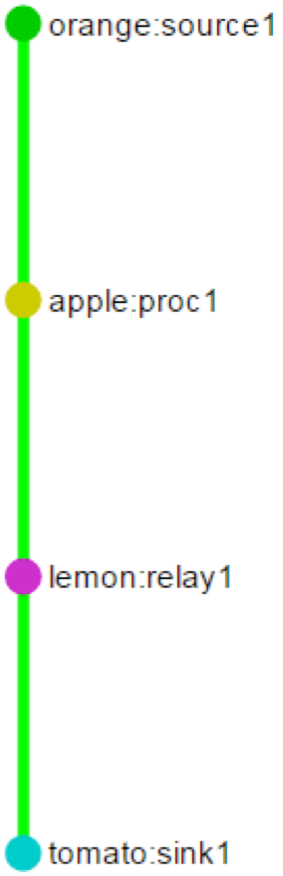
Example 1

```
{
  "SimId":"demo",
  "StreamList":
  [
    {
      "StreamId":"Stream1",
      "DataSize":"20000000",
      "KiloBitRate":"1000",
      "FlowList":
      [
        {
          "NodeList":
          [
            {
              "NodeType":"SinkNode",
              "NodeId":"tomato:sink1",

"UpstreamId":"orange:source1"
            },
            {
              "NodeType":"SourceNode",
              "NodeId":"orange:source1",
              "UpstreamId":"NULL"
            }
          ]
        }
      ]
    }
  ]
}
```

The above work specification represents one flow for a stream from a source to a sink. In the node list, the sink always appears first followed by its upstream. The UpstreamId and the NodeId of the next NodeMap should always match. The source's UpstreamId is always NULL.

Example2

```
{
  "SimId":"demo",
  "StreamList":
  [
    {
      "StreamId":"Stream1",
      "DataSize":"20000000",
      "KiloBitRate":"1000",
      "FlowList":
      [
        {
          "NodeList":
          [
            {
              "NodeType":"SinkNode",
              "NodeId":"tomato:sink1",
              "UpstreamId":"lemon:relay1"
            },
            {
              "NodeType":"RelayNode",
              "NodeId":"lemon:relay1",
              "UpstreamId":"apple:proc1"
            },
            {
              "NodeType":"ProcessingNode",
              "NodeId":"apple:proc1",

"UpstreamId":"orange:source1",
              "ProcessingLoop":"100000",
              "ProcessingMemory":"1000"
            },
            {
              "NodeType":"SourceNode",
              "NodeId":"orange:source1",
              "UpstreamId":"NULL"
            }
          ]
        }
      ]
    }
  ]
}
```

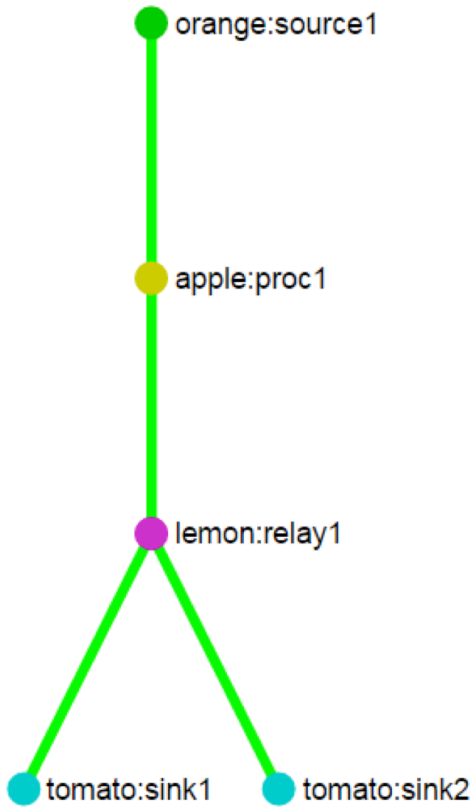


The above example represents one flow in a stream with all the four node types. The NodeMap for processing node has the properties ProcessingLoop and ProcessingMemory that will control the amount of CPU and memory consumed for a flow by a processing node.

Example 3

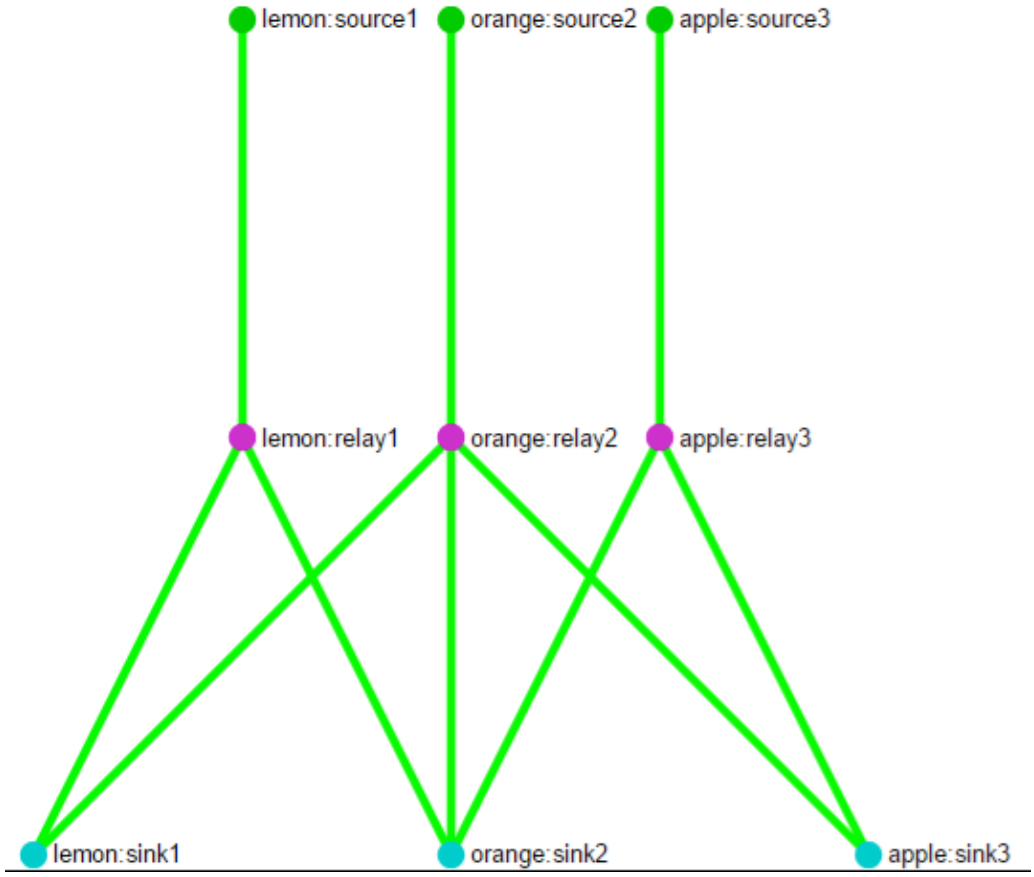
The below example represents how to add multiple downstream nodes to a relay for multicasting data to those nodes. To achieve this topology, multiple flows of the same stream will have the same NodeMap from the relay node onwards. After this relay node the NodeMaps in each flow will diverge to represent another path for that flow.

```
{
  "SimId":"demo",
  "StreamList":
  [
    {
      "StreamId":"Stream1",
      "DataSize":"20000000",
      "KiloBitRate":"1000",
      "FlowList":
      [
          …(same as flow example 2)
        {
          "NodeList":
          [
            {
              "NodeType":"SinkNode",
              "NodeId":"tomato:sink2",
              "UpstreamId":"lemon:relay1"
            },
            {
              "NodeType":"RelayNode",
              "NodeId":"lemon:relay1",
              "UpstreamId":"apple:proc1"
            },
            …(same as upstream nodes in
              previous flow)
          ]
        }
      ]
    }
  ]
}
```



Example 4

Peer to Peer Topology



The above diagram represents how to simulate a peer to peer MDN topology. In the above example, each NodeContainer is a peer in the network. That is, NodeContainer lemon, orange and apple are peers in the network. Each NodeContainer is capable of both sending data using the Source Node functionality and receiving data using the sink node functionality. The above network also demonstrates the concepts of NodeContainers and Nodes. They clearly distinguish the difference between a NodeContainer that is a physical node in the network and a Node that is a functional element in the NodeContainer.

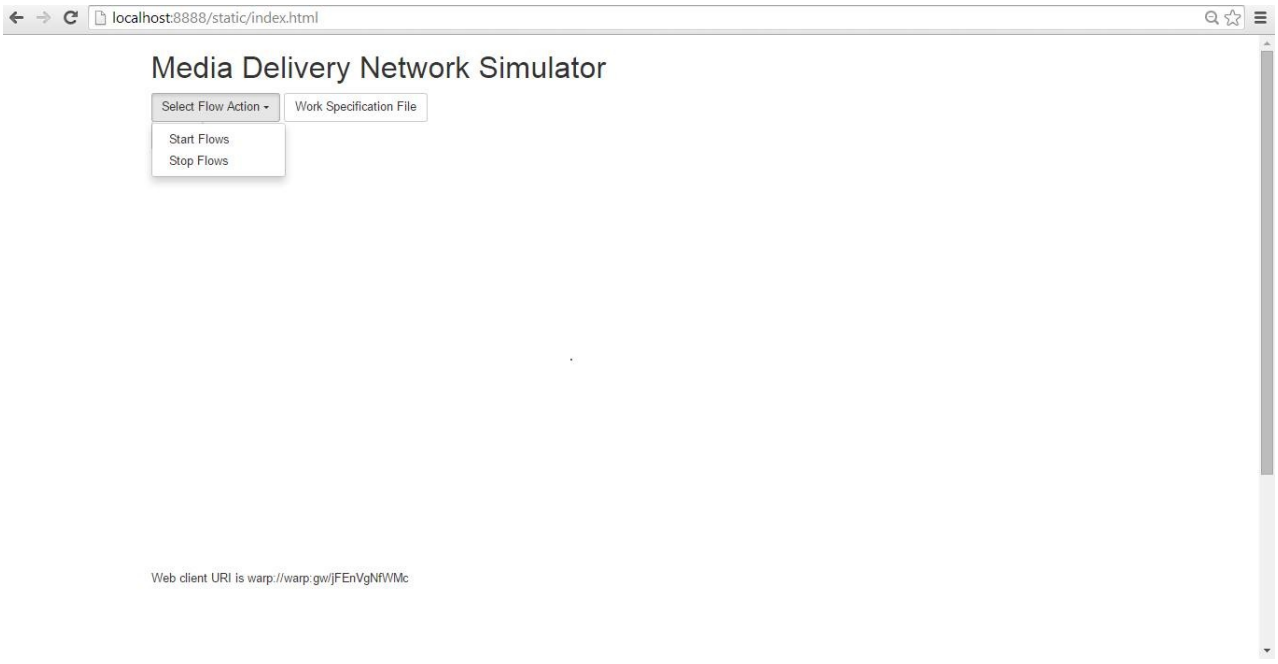
The Work Specification example for this can be found at:
$PROJECT_HOME/src/test/demo/demo-5-p2p.json

5. MDN Simulator Panel

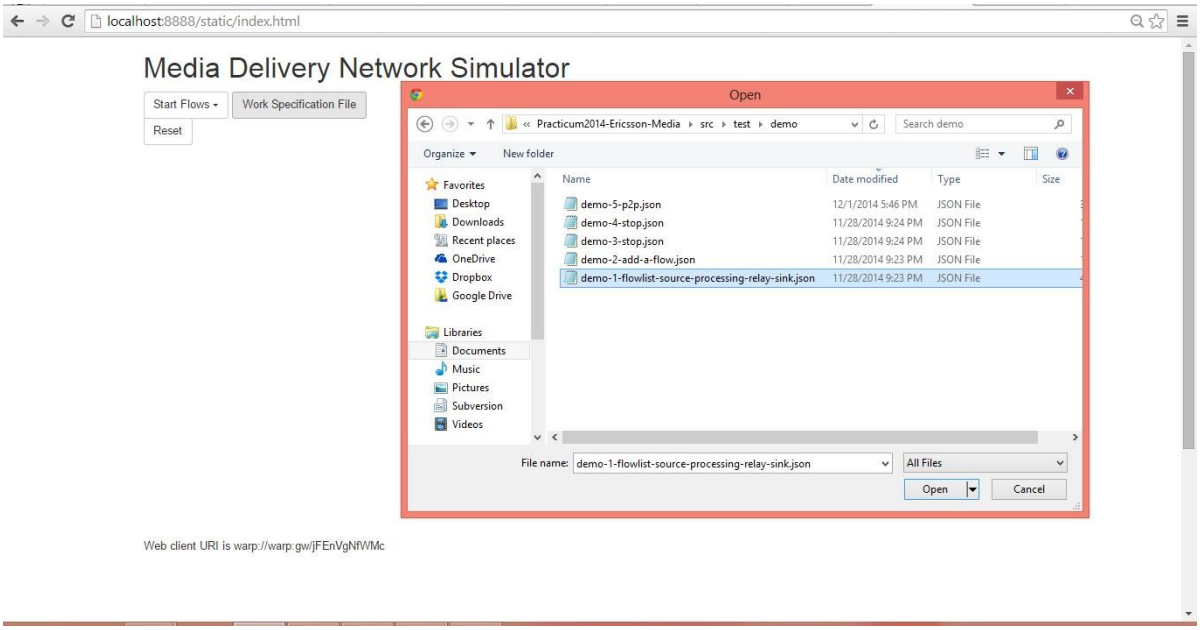
When you start the web browser and connect to the master, you will get the MDN simulator panel. The following figures demonstrate the various capabilities of the panel.

The Web Interface initially looks like,

The Select Flow Action has the following options,



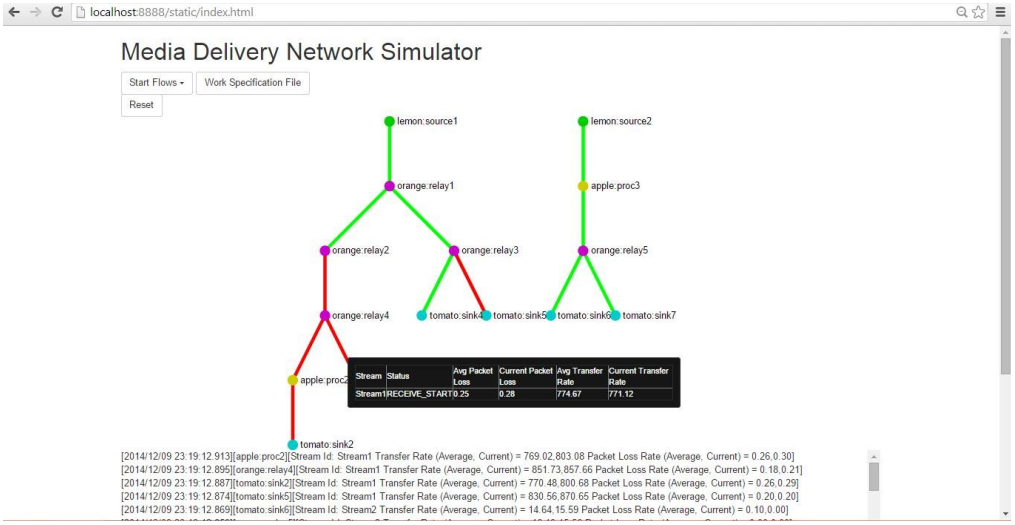
We can select one of these options and choose the Work Specification file that has the streams and flows to be either started or stopped based on the selected option.



Once the file is selected for starting the flows, the master node deploys the nodes on the Node Containers and starts the flows in the Work Specification by sending the flow to the sink node for every flow in the file.

Initially, the edge between the nodes will be grey. This indicates that the flows have not started flowing yet. Once the flows start, the edges turn green. If there is packet loss beyond a threshold, the edges turn red. The edges go black after the flow has stopped.

The below image shows various aspects of the Web Client for a simulation that is running.



The above simulation is from the script:
$PROJECT_HOME/src/test/demo/demo-1-flowlist-source-processing-relay-sink.json

The major elements of the panel include:

- Control actions for the simulation
- Graph Area
- Label showing state of link between nodes
- Log Area

The label appears when you hover over a graph element including the node and the edge.

- The node label will show the status (started/ended) of various streams that are flowing through that node.
- The edge label will show average and current value of packet loss and transfer rate for all the streams flowing through the edge in table format.

The Reset button can be used to bring the simulator to the initial state by killing all the flows and nodes in the node containers. All the state information in the master is also cleared. This way we can start a new simulation without re-deploying the system.