

Media Delivery Network Simulator - User Manual

Introduction:

The Media Delivery Network Simulator is a distributed simulation framework that can be used to test and evaluate a media delivery network. This involves node deployment, distributed control of the simulation, data collection/aggregation and visualization of the distributed simulation. The simulator generates and distributes actual ip packets across the ip backbone based on user input. The following sections will give a brief introduction of various components of the system and gives a short tutorial on how to deploy the system, specify the input for the simulation(work specification) and controlling, visualizing the simulation using the web interface of the simulator.

Components of the simulator framework:

The following are the main components of the simulator:

1. Master
2. Web Client
3. Node Container
4. 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 Container and the nodes. All the components register with the master node. The master node hosts the Message Bus server that is used to pass control messages. The master node also hosts the Web Client frontend and backend.

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

Example: <http://localhost:8888/static/index.html>

The Web Client has ability to upload a user specification to the master to deploy the nodes and start the simulation. The front end also displays 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:

The Node Containers are JVM processes that can be either in the localhost or can be remote. They represent a single node/machine in the media delivery network chain. The node containers can host various node functionalities including source, processing, relay and sink. The above mentioned node functionalities are the basic elements that have been implemented as of now.

In order to create a new node type that can be used inside a Node Container, one simply has to extend the following classes to create new classes

1. `edu.cmu.mdnsim.nodes.AbstractNode` to `edu.cmu.mdnsim.nodes.<NodeType>Node`
2. `edu.cmu.mdnsim.nodes.NodeRunnable` to `edu.cmu.mdnsim.nodes.<NodeType>Runnable`

If the user wants this Node type to be part of the Node Container functionality in the simulation, the user has to set the `NodeType` key in the `NodeMap` of the `WorkSpecification` to `<NodeType>`. The `NodeContainer` will automatically pick up the right class from the `edu.cmu.mdnsim.nodes` package during runtime.

Nodes:

The nodes represent some functionality of nodes in the media delivery network chain. They run as threads in the Node Container process. The node container can host one or more of these node threads so that they can simulate various functions like a peer in a network that can both send and receive media data. Currently the following four node types have been defined and implemented.

1. **SourceNode:**
The node that generates media data and sends it downstream. This is the first node in the media delivery chain.
2. **ProcessingNode:**
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 for a packet
3. **RelayNode:**
This node is used to multicast data to multiple downstream nodes. It keeps track of all downstream nodes for a stream. When a packet is received for that stream, it makes copies of that packet and sends to all downstream nodes for that stream.
4. **SinkNode:**
This node consumes the media data. It is the end of the media delivery chain

Deploying the Media Delivery Network Simulator:

There are 3 majors components to start to make this application running.

1. Master node - Run the Master.java file and when it says that it is registered then start the other two components.
2. WebClient - Open any web browser (preferably Chrome) and go to `http://localhost:8888/static/index.html` (replace localhost by Master Node IP Address)
3. NodeContainer(s) - Run the NodeContainer.java file with argument `label:NodeContainerName`

Once the Master and the node containers are deployed, we can start running simulations on the simulator. It is not necessary for the Web Client to be running in order to control a simulation. The master uses a rest framework for controlling the simulation.

1. Start Flow
 - Purpose: To start a new flow or add new flow to existing simulation.
 - Method: POST
 - URL: `WebClientURI/work_config`
2. Stop Flow
 - Purpose: To stop existing flow.
 - Method: DELETE
 - URL: `WebClientURI/work_config`
3. Reset Simulation
 - Purpose: To reset the entire simulation.
 - Method: DELETE
 - URL: `WebClientURI/simulations`

Note:

1. The Master Node has to be started before any other component is started as other components need to register with the master
2. The Node containers have to be deployed manually before the master can deploy nodes on the node container
3. Resetting the simulator will not kill the node containers. They will kill only the nodes

The Work Specification:

The Work Specification is a JSON formatted file that is used as the input script for the simulator. The Work Specification is used to specify the following information,

- SimID
The Simulation ID
- StreamList
List of streams that are part of the simulation
- StreamID/DataSize/KiloBitRate
For every stream in stream list, a StreamID for uniquely identifying a stream, the

DataSize in bytes and the KiloBitRate of the stream at which the source will send 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

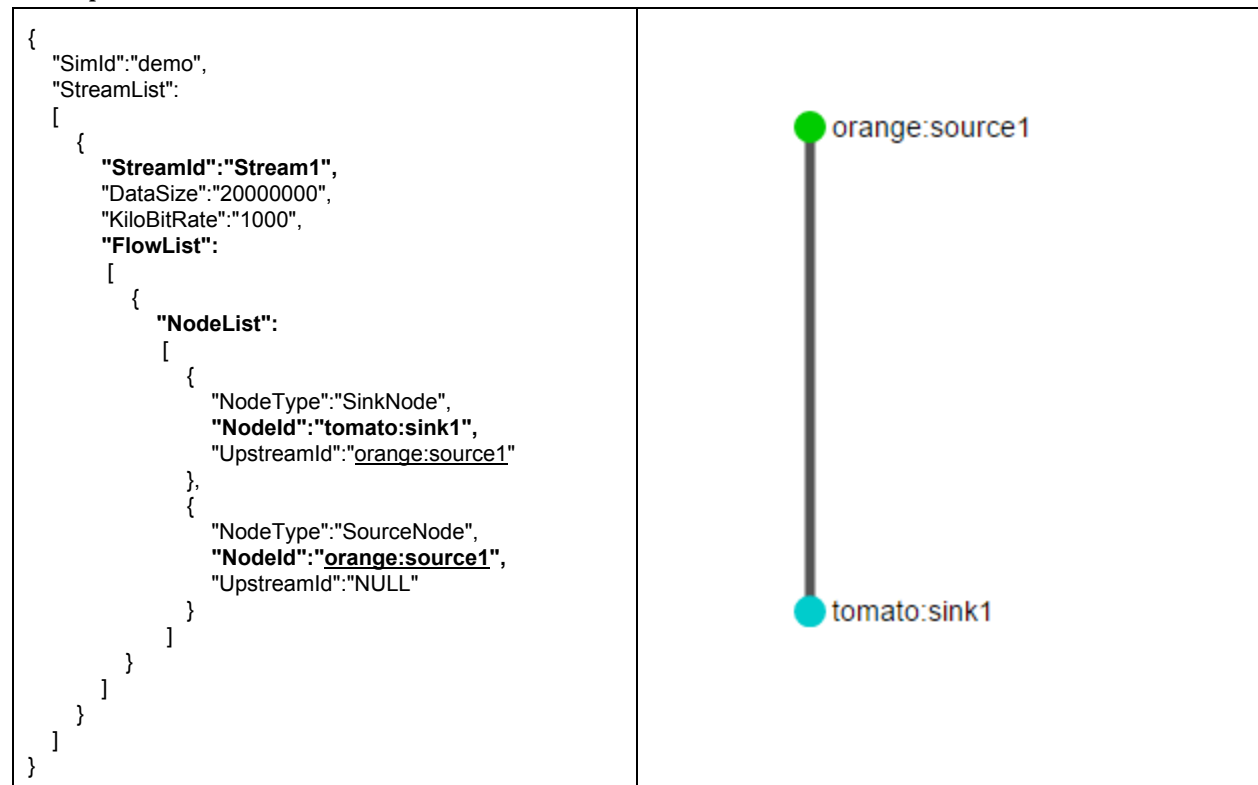
A list of NodeMap's that represent a unique path for a stream from the source to the sink

- NodeMap

A map of key, values for representing the properties of each node in the flow. The NodeMap within the NodeList are ordered from the sink to the source. Every n+1 NodeMap in the NodeList represents the upstream for node at n. Every n-1 NodeMap is the downstream node for the node at n. The sink NodeMap will not have a downstream node and the source NodeMap will not have a upstream Node

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

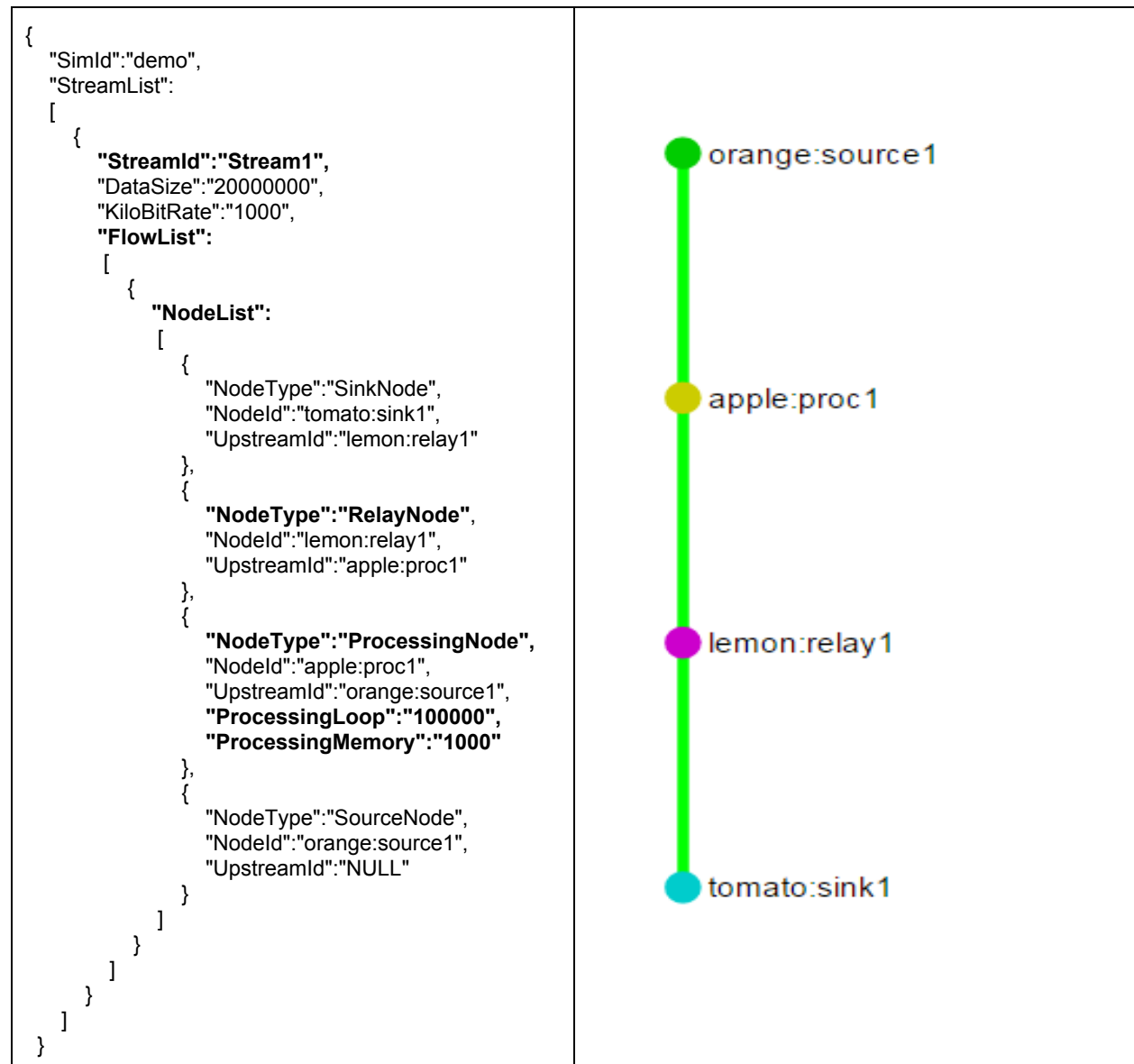
Example 1:



The above work specification represents one flow for a stream from a source to the destination. 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:

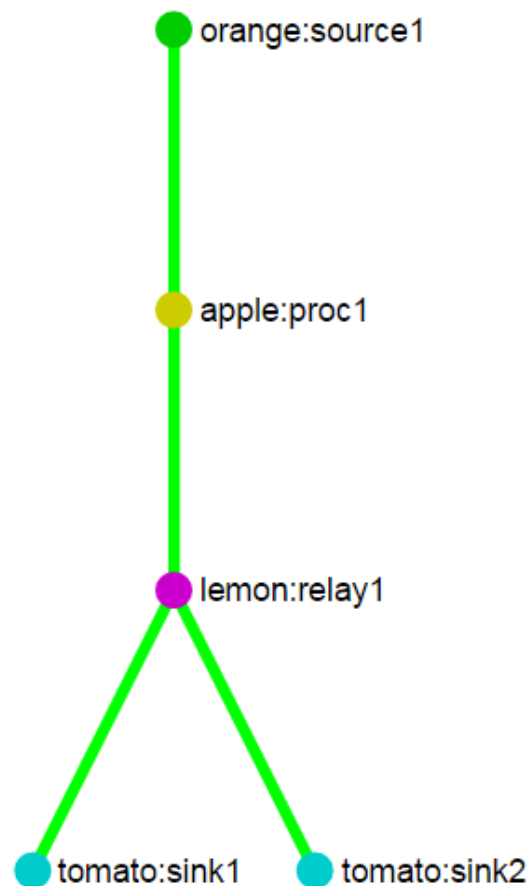


The above example represents one flow in a stream with all the four node types. The NodeMap for processing node has the properties ProcessingLoop and ProcessingMemory that will control the amount of CPU and memory consumed by a processing node for a flow.

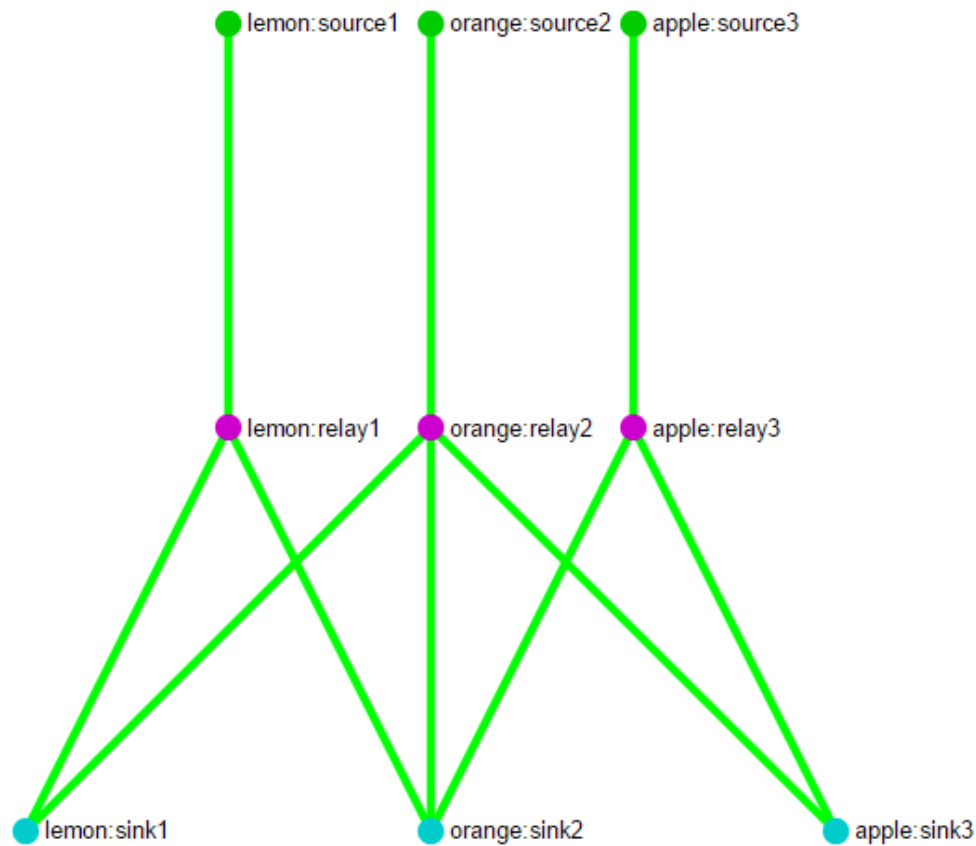
Example 3:

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": [
        {
          "NodeList": [
            {
              "NodeType": "SinkNode",
              "NodeId": "tomato:sink1",
              "UpstreamId": "lemon:relay1"
            },
            {
              "NodeType": "RelayNode",
              "NodeId": "lemon:relay1",
              "UpstreamId": "apple:proc1"
            },
            ....(same as example 2)
          ]
        },
        {
          "NodeList": [
            {
              "NodeType": "SinkNode",
              "NodeId": "tomato:sink2",
              "UpstreamId": "lemon:relay1"
            },
            {
              "NodeType": "RelayNode",
              "NodeId": "lemon:relay1",
              "UpstreamId": "apple:proc1"
            },
            ... (same as example 2)
          ]
        }
      ]
    }
  ]
}
```



Example 4: Peer to Peer Simulation



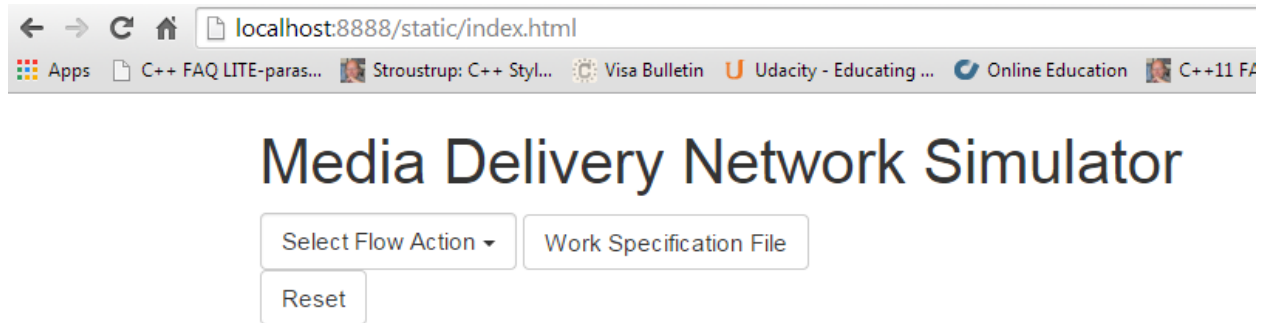
The above diagram represents how to simulate a peer to peer media delivery network. In the above example, each NodeContainer is a peer in the network. That is, NodeContainer lemon, orange and apple are peers in the network. Each NodeContainer is capable of both sending data using the source Node functionality and receive 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 the Nodes, that are functionalities (a functional element) in the NodeContainers

The WorkSpecification for this can be found at
<projectroot>/src/test/demo/demo-5-p2p.json

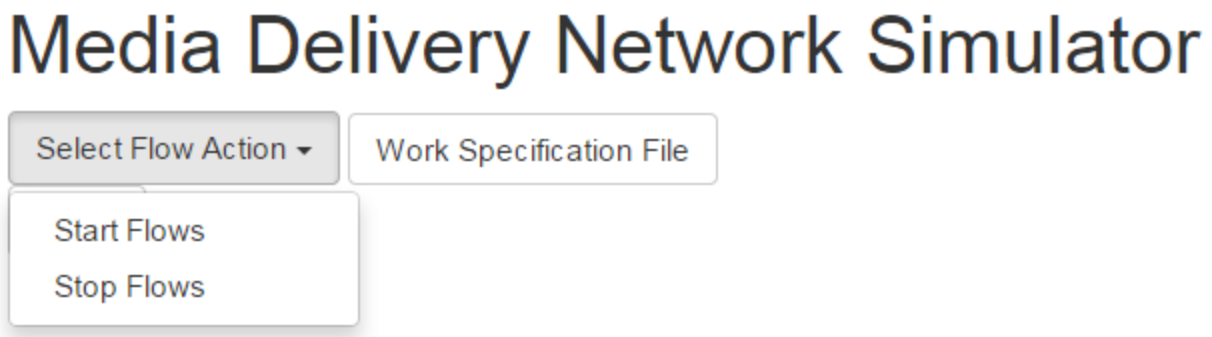
Controlling and Visualizing the simulation:

The following images demonstrate the various capabilities of the Web Client.

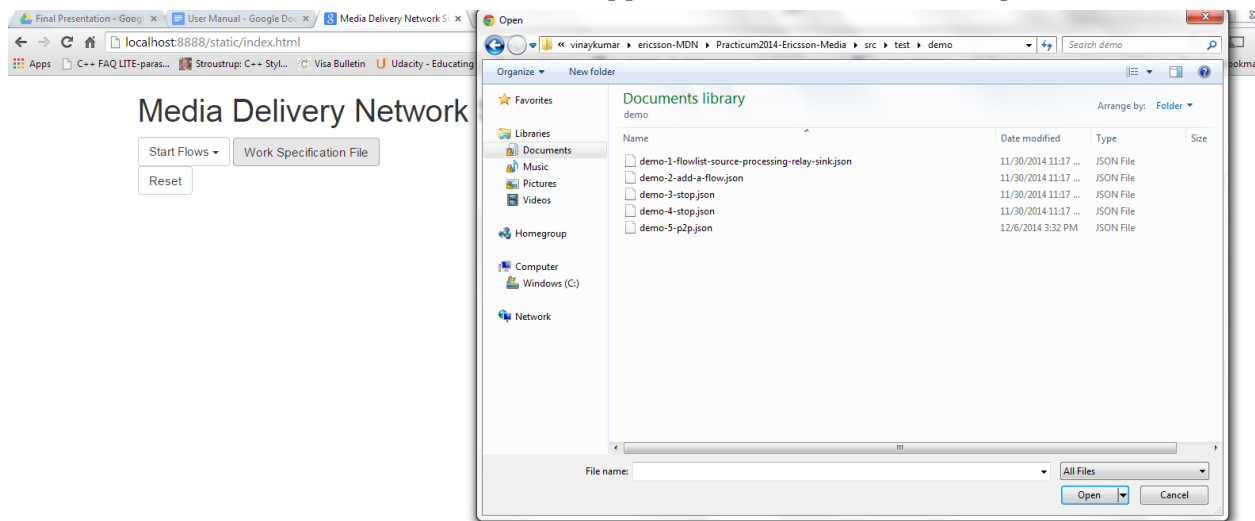
The Web Interface initially looks like,



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.

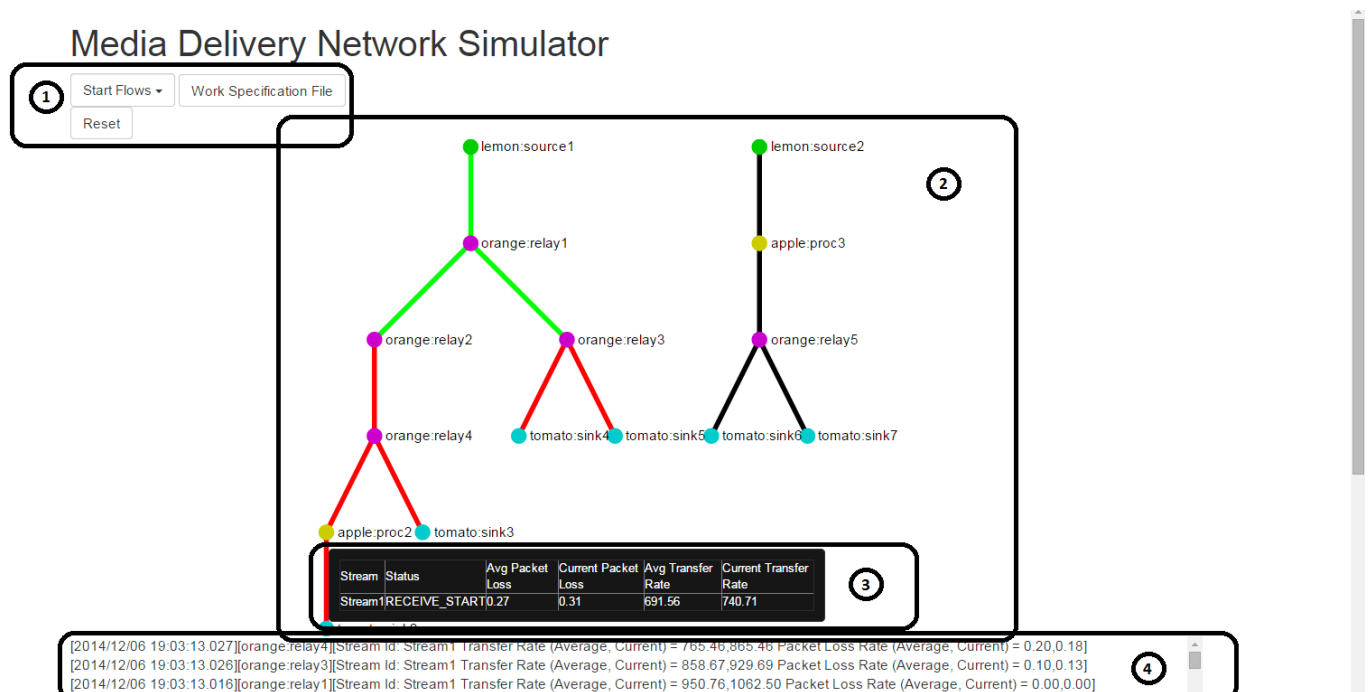


Once the file is selected for starting the flows, the master node deploys the nodes on the Node Containers and starts the flows in the Work Specification by sending the flow to the sink node for every flow in the file.

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.

Edge Color	Flow Status
	Not started
	Started
	Significant packet loss
	Finished

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



The above simulation is from the script

`<projectroot>/src/test/demo/demo-1-flowlist-source-processing-relay-sink.json`

The major elements of the WebClient include

1. Control actions for the simulation
2. Graph Area
3. Label showing state of link between nodes
4. 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 lost. This way we can start a new simulation without re-deploying the system.

For more details on the design of the system please refer to the docs folder in the project.
<https://github.com/cmuv-sc/Practicum2014-Ericsson-Media>