Fall | 2014

Technical Report

Team

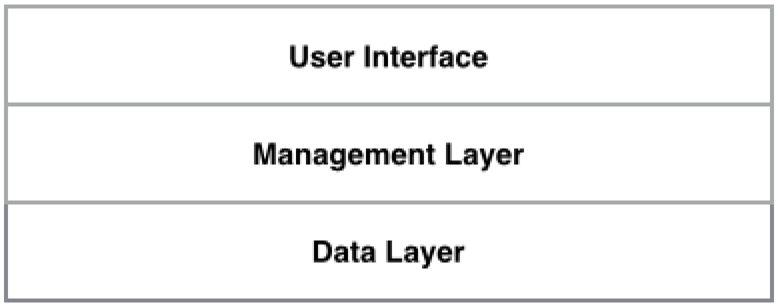
Enclosed in this document is the technical report of the <Project Name> sponsored by <Sponsor Name>.

Table of Contents

4.1 Overview

Media delivery network(MDN) simulator is a distributed systems. The system follows master-slave architecture. Components running on different machines simulate an overlay network over the Internet. Therefore, the simulated network is an IP-independent network.

The figure 4.1 illustrates how the MDN simulator can be divided by the functionality of different components. The data layer contains components nodes (e.g. source node, sink node, processing node and relay node) and data path. The management layer consists of message bus, node containers and master, which is used to control the system. The user interface is denoted as web client, which provides interactions with users.

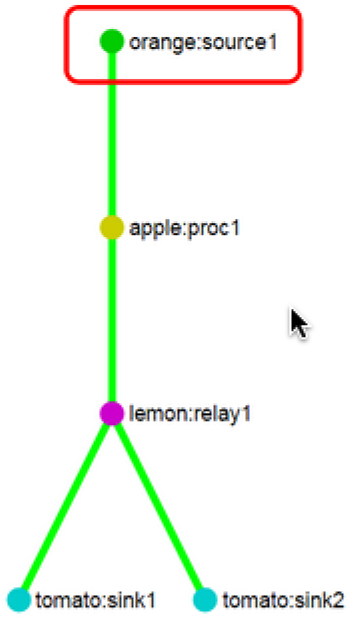


(Figure 4.1 Layers of MDN Simulator)

4.2 Data Layer

The data layer forms the actual media delivery network. The data layer generates the life-sized media traffic, performs some processing operations on media data and delivers the data from the source of the media chain to the clients. 4 basic types of nodes are identified and implemented: Source Node, Processing Node, Relay Node and Sink Node.

Source Node: Source node generates the life-sized media data. The size of data and the transmit rate can be configured by users input. Source node also inserts markers such as packet ID and sending timestamp for sake of computing packet loss rate and end-to-end latency at downstream nodes. Figure 4.2 is generated by MDN simulator at user interface. The components circled by red is the source node. The source node doesn't take any input and is the start of a media chain.

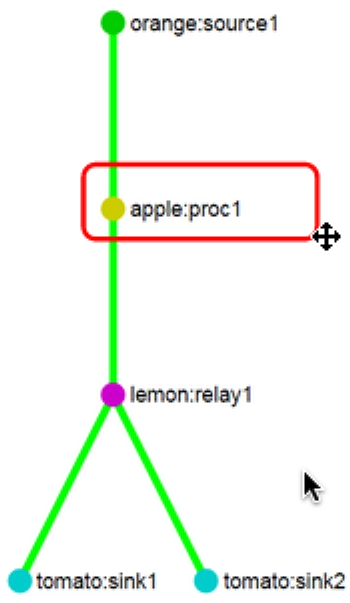


(Figure 4.2 Source Node)

Processing Node: Processing node simulates actual processing operations on media data such as encoding of stream data, adds insertion and subtitle addition. As a simulator, MDN simulator doesn't perform real processing operations on data. Instead, it just consumes CPU by for loops and memory by allocating useless objects. The processing load is configurable as well. Users can add corresponding load to the media data based on actual processings performed on it.

At the input interface, where processing acts as a receiver to its upstream, processing node also monitors the packet loss rate.

The circled component in figure 4.3 is the processing node. Processing node simply takes one input and has one output.



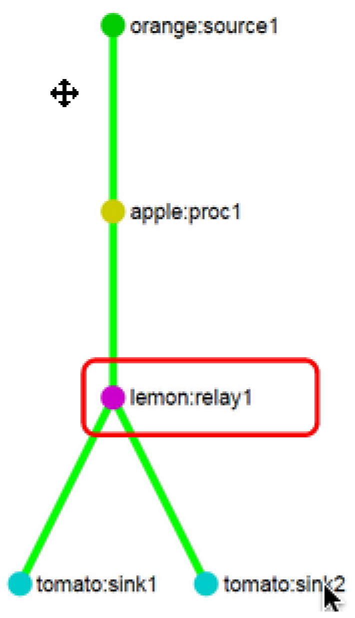
(Figure 4.3 Processing Node)

Relay Node: Relay node takes one input, duplicates data and delivers to multiple outputs. Relay node does broadcast to downstreams. However, it is worthwhile to mention that this broadcast doesn't mean broadcast in IP layer. Since the data is sent via UDP socket, it should be regarded as unicast in IP layer and broadcast at application layer.

The relay node can be dynamically attached with a new downstream. Upon an addition of downstream, it automatically duplicates a new copy of data and sends to it since the time of addition. On the other hand, a downstream can be dynamically removed from the relay node. Upon the removal, the relay node needs to decide whether continues working. If a relay node doesn't have any active downstream that consumes the media data, relay node notifies its upstream to stop sending data and stops its functionality. If there is at least one downstream consuming data, relay node continues working.

As like the processing node, relay node acts as receiver to its upstream. Therefore it also monitors the packet loss.

The circled component in figure 4.4 is a typical relay node. In this scenario, the relay nodes has two downstreams. The relay node can take and only can take one input and forward data to at least one downstream at output.

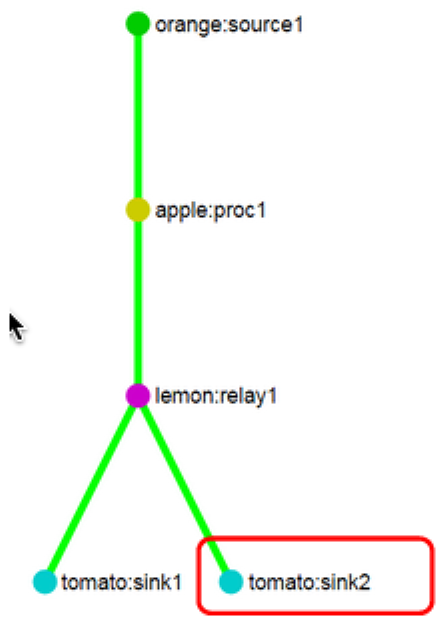


(Figure 4.4 Relay Node)

Sink Node: Sink node acts as the consumers of media data. The consumers can be audience in real scenario. In P2P network, it can also act as a peer in the phase of caching data. It can later sends the data to its peer. The role of this node is changed later, which calls for the requirement of interchangeable functionality during runtime from the system.

In terms of monitoring the performance of the system, sink node collects packet loss rate and computes the end-to-end delay of each packet.

Figure 4.5 demonstrates sink nodes. Sink nodes are always the end of a media chain and doesn't generate any output as itself consumes data.



(Figure 4.5 Sink Node)

Media data flows in and out at different nodes via data path. In the data path, media data is sent via UDP socket. The reasons to use UDP sockets are as follows:

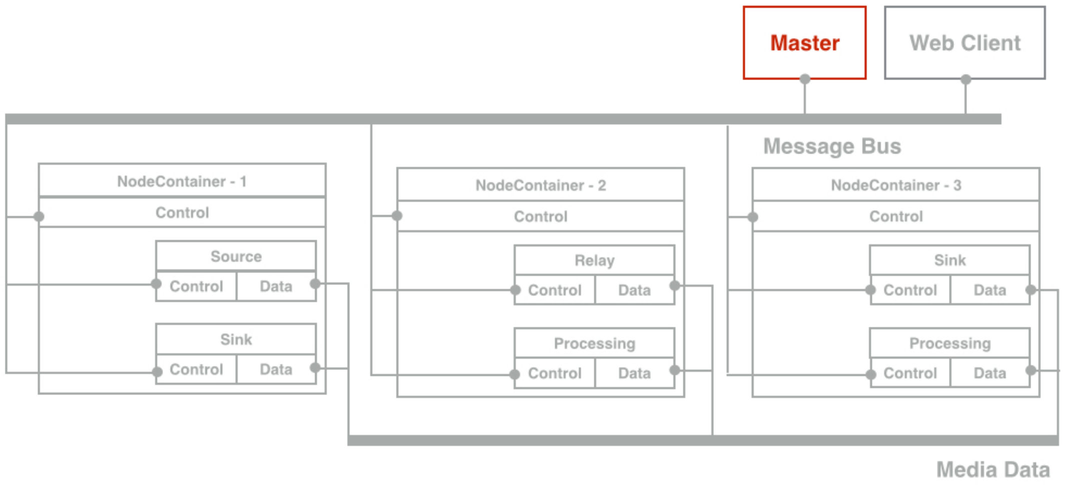
1. UDP protocol just wrap up IP protocol providing much less service than TCP protocol. Since UDP doesn't have any flow control and congestion control, it is possible to generate enough traffic loads to the reach the limitation of the network. And it is possible to observe packets loss by UDP sockets compared to TCP sockets because TCP protocol ensures an in-order and reliable end-to-end communication.
2. In spite of the fact most current media delivery technologies using TCP as underlying transport protocol, emerging technologies such as QUIC are trying to move to UDP sockets. For purpose of research, it is more interesting to use UDP sockets.

Nevertheless the system used UDP sockets in data path, it is very easy to change to TCP sockets.

4.3 Management Layer

Management layer forms the backbone of the distributed systems and orchestrates components to function. And it is also in charge of collecting metrics from different components. Management layer consists of three components: master, node containers and message bus.

Master: The master is the core of management layer and is the core of the systems as well as is shown in figure 4.6. Master communicates with every component in the system via message bus.



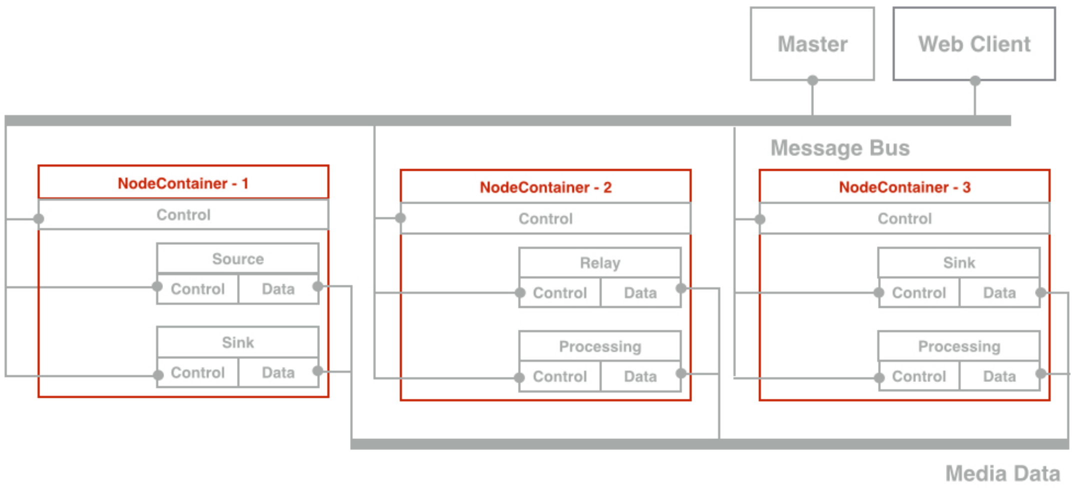
(Figure 4.6 Mater)

Web client: In the communication between master and web client, web client sends the control message generated by user, such as start or stop a simulation, attach a new flow to a media chain and reset the whole systems. Master sends the view that reflects the status of the system.

Node container: Master configures the functionalities of each node container by instantiating functional nodes at data layer. Node containers report the states of each node they host to the master. Therefore master can take appropriate action based on the the states of nodes.

Node: Master orchestrates nodes to form the median delivery network. Master contains the whole picture of the topology and translate this knowledge to each node. Nodes only contains the knowledge of their upstream node and downstream nodes if apply. Master also controls the behavior of each node directly, such as start or stop sending data to the data path. Nodes report the performance metrics to master in return.

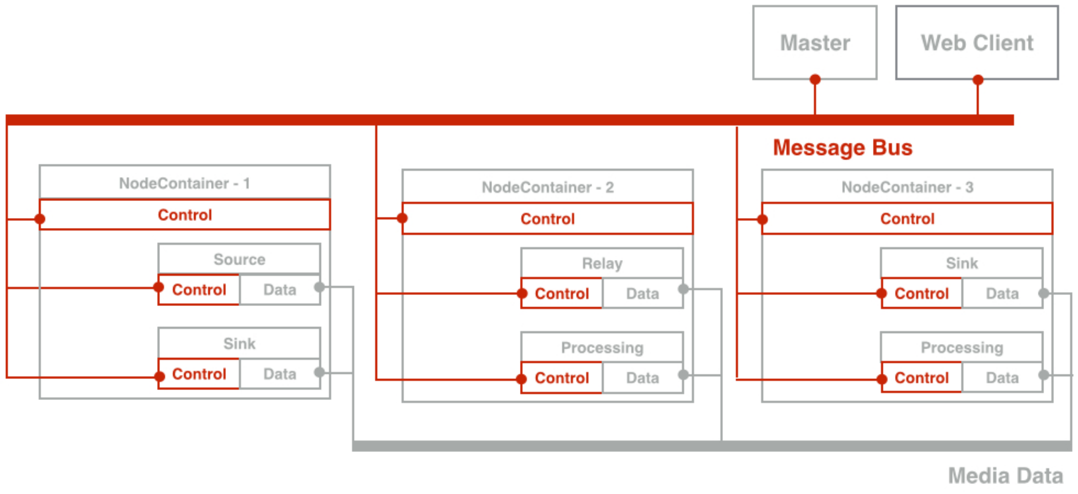
Node Container: Node containers are different processes running on different machines. Node containers are denoted by the red part in figure 4.6. Each node container can host multiple nodes. Therefore node containers act as composite nodes. Node containers provide flexibility to the system as each machine can be configured with different functionalities. The functionalities are configured by the master during the runtime and thus are interchangeable during run time.



(Figure 4.6 Node Container)

Message Bus: Message bus is used as a channel to pass all control messages. Message bus connects every component in the system. Message bus is backed by reliable communication to ensure the control messages will be delivered.

As is shown in figure 4.7, each component contains the control logic (master and web client are complete control logic). Control logic represents message listeners. All control messages are asynchronous.Control messages comprised of messages used by master to control the behavior of node containers and nodes, messages between master and web client, performance metric reports sent from nodes to master.



(Figure 4.7 Message bus)

4.4 User Interface Layer

User interface is the layer that handles the interaction with users. The user interface is divided into two parts: input and output. The input takes the users input in terms of the action users take and

User interface contains two components, i.e. work specification and web client. Work specification defines the simulation in terms of topology of the network and the network load. Web client visualizes the work specification after user submits it and provides the interface to control the system.

Work Specification

Stream

Flow

Web Client

5. System implementation ()

6. Experiments and analysis (Research purpose)

7. Conclusions and future work

We will improve the system to increase the performance in the following aspects. A script will be used to start node containers. Currently, one thread is assigned with one stream in one node. The maximum number of nodes is limited by the capacity of the JVM. In the future, one node will only use one thread, So the number of nodes the system can support will increase. A marker in traffic packets will be applied to induce processing and encoding workloads on processing nodes. So processing node will generate workload based on what is gotten in packet instead of fixed parameters in work specification. A better way to visualize a large number of nodes needs to be used by providing some functions such as filtering the nodes. So users can choose to see nodes within a specific stream and other nodes won't be shown.

Also, all the nodes within one node container can be put together as a cluster in UI. We will provide an UI to fill out html form to generate a work specification to avoid misspelling.

8. Things we tried and discarded (Take from progress report)

We made the initial design and implementation as the following diagram. Our initial design and implementation uses HTML5, JQuery and Sigma.js as front end technologies. We used Tomcat and Java Servlet for backend. There were three technologies that were used for communication within the system. Browser uses Ajax to talk with Tomcat Server; Tomcat server uses Java RMI to talk with the master; Master uses Rabbit MQ to talk with data layer.

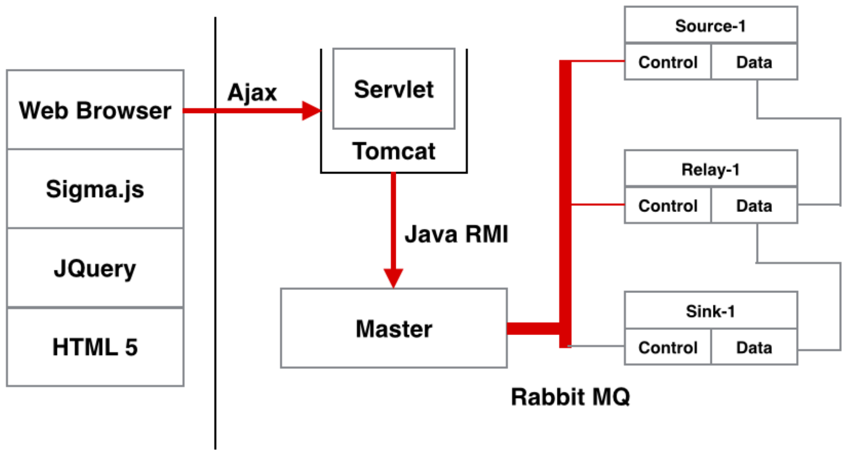


Figure. Initial system topology that was discarded

We discarded it based on the following analysis. The major drawback of this design is multiple communication mechanisms are involved in the system and lead to a big complexity to debug and maintain it. Also, in the practical level, there were a lot of dependencies we need to rely on for running the system. It is also hard to define a unified interface for control messages that all components of the system can use for communication.

9. Acknowledgements

We would like to thank Ericsson Research for sponsoring this project and providing us with this opportunity. We would like to specially thank Vlad, Jia and Alvin for guiding us through this project providing us with valuable feedback on design of the system.

Appendix:

-Check in everything onto GitHub under the predefined directory including the following items

- -Readme file: Describe briefly the purpose of the project, how to download and install the software, how to use the software
- -API (sub-directory): instruct APIs as well as descriptions and examples
- -Test Suite (sub-directory): a collection of test examples and descriptions
- -src (sub-directory): include all source code categorized by packages
- -lib (sub-directory): include all related library packages needed to support the project
- -conf (sub-directory): include any configurations settings and files
- -app (sub-directory): any applications built on top of the APIs
- -contact: please provide every team member's contact information (cell number, personal email)
- -Documents (sub-directory): in different WORD files
  - -access information: URL, user name/password
  - -download and installation documents with step-wise descriptions
  - -executive summary
  - -background and motivation
  - -assumptions and considerations
  - -design documents (architectural design documents and various diagrams e.g., UML files)
  - -discussions (MoM files)
  - -presentations (ppt file)
  - -tutorial: step-by-step usage file with screen shots included
  - -future work: to-do list and descriptions
  - -technical report

-Transit the knowledge to either Advisor or a signed student (schedule time to sit down for transition)