CSE 5231 Computer Networks

Class Project Report, Fall 2013

Andreas Bjoru & Srini Venkatesh
Florida Institute of Technology

November 26, 2013

Contents

1. Summary

The objective of this project is to demonstrate the functionality of some of the OSI layers in the communication stack. In particular, key functions performed by layers 2 (Data Link), 3 (Network) and 4 (Transport) is to be emulated for the purpose of transferring a file from one network node to another [1].

2. Architecture

The architecture of the network simulator is fairly simple. The main application class, `Simulator`, is responsible for reading the topology and creating each network node. A network node, *host or router*, are represented as system threads. Each of these network nodes share a common super-class `AbstractNetworkNode` that defines common functionality and basic implementation for the OSI layer methods. The OSI layer methods shares the same pattern in which they handle both incoming and outgoing data. This is achieved through the `Transmit` enum type that is a required argument along with the data payload.

The `physicalLayer(byte[], Transmit, Address)` method will send a given packet to every node attached to the same network, thereby simulating the task of writing a packet on the 'wire'. This is achieved by collecting every node attached to the sender's network address through the use of utility methods on the `Topology` class. When the physical layer method receives a packet, it will just hand it off to the layer above it.

The `linkLayer(byte[], Transmit, Address)` method will packet a given payload in an `EthernetFrame` along with the source and destination MAC addresses from the `Address` argument. The frame will then be handed down to the physical layer method to complete the sending responsibilities of the layer. When the link layer receives a packet it will first verify the cyclic redundancy check (CRC) sum of the packet. If this check fails, the packet will be dropped. The next check performed is the destination MAC address check which determines if the packet is actually for this node. If the destination MAC address matches the network node, then the payload of the packet is handed off to the layer above it. Otherwise, the packet is simply dropped since it is not addressed for this node.

2.1. Network hosts

The `transport(payload, transmit, addr)` method slices payload into segments while sending and assembles the segments when it receive the payload. When sending a given payload, it slices the payload into many segments based on the payload size. First it will check the length of the payload to make sure it needs to be sliced to MTU's size or not. If the payload size is more than the MTU's size then it will slice the segments considering the header length. These segments has TCP header along with data. Source and destination ports are included in the header. SYN and FIN flags are used to identify segments starting and ending point. This payload will be sent to network layer for adding IP header and route this segment to right node. Sending parameters would be `networkLayer(byte[], Transmit, Address)` method.

When receiving a payload, this method will convert the payload into segments. It will then extract the chechsum to make sure segment is valid, if it's not valid it drops. If not, it keeps accumulating the segments into a buffer. along with sequence number. Then it will check for the FIN flag in the header to confirm that that's the last segment. If it's not FIN (it must be SYN), then it keeps assembling the segments to build a complete payload. Once the payload is built, it will be sent to application layer for creating a file in destination node.

2.2. Network routers

The functionality of the network layer is overridden for routers since they will just route packages to the next network or host. Similar to the inherited version of this method, it will start by verifying the IP header checksum. If the verification fails, the package is dropped. Otherwise, the correct network interface is found and the MAC address of that interface is used as the new source MAC address. The destination MAC address is then resolved through `Topology arpResolve(IP)`, and the payload is sent to the `linkLayer(byte[], edu.fit.cs.computernetworks.AbstractNetworkNode.Transmit, Address)` method if the router.

2.3. Topology

The topology for the network described by the assignment was fixed (see figure 1). In order to avoid having to hard-code information extracted from this topology (routing tables, MAC tables, etc), we decided to represent the topology in an external format as seen in appendix B. This will also allow us to change the topology of the network without having to change the implementation. Each node in the topology is represented by a corresponding node in the *JSON* file including relevant information (see table below). Furthermore, the topology representation also contains the hardware address lookup table for the network.

Element	**Type**	**Description**
id	Host, Router	Hostname for the network node
ip	Host	IP address for a *host*
mask	Host	Network mask for a *host*
mac	Host	Hardware address for a *host*
mtu	Host	Maximum transmit unit for a *host*
gateway	Host	Default gateway for a *host*
routing	Host, Router	Routing table for the network node
ports	Router	Array of network interfaces for a *router*
port → ip	Router	IP address for a given port on a *router*
port → mask	Router	Network mask for a given port on a *router*
port → mac	Router	Hardware address for a given port on a *router*
port → mtu	Router	Maximum transmit unit for a given port on a *router*

The topology representation is mapped to corresponding Java classes using a 3rd-party library called `Jackson`, which sole purpose is to process the `JSON` data format. The Java package `edu.fit.cs.computernetworks.topology` contains the mapped types which is briefly explained below.

Topology is the root class of the topology model.

Node defines an abstract superclass for both Host and Router.

Host describes a network host.

Router describes a network router.

Port describes a network interface on a Router.

RoutingEntry describes an entry in a node's routing table.

MACTableEntry describes an entry in the global ARP table.

3. Data flow

Basically how data flows between network nodes. Explain how the layer methods work..

4. Execution

This project is built on eclipse environment with some additional packages. So first we need to download and install these packages by following the given steps.

4.1. Windows Environment:

1. Go to command prompt and navigate to the project's root location
2. Execute this command: gradlew.bat eclipse
3. The gradlew command is a wrapper around the gradle build tool and will download the binaries for the build tool before executing the task.
4. When gradle finishes (i.e. eclipse project files have been generated), you can import the project in eclipse by selecting import then existing project from the eclipse menus.
5. Create sample folder to place the file to tansfer to different nodes.
   For Example: Like, C:\tmp\A, C:\tmp\B, C:\tmp\C
6. Execute the following command:
   gradlew.bat run -ProotPath=C:\tmp

4.2. Unix Environment:

1. Go to command prompt and navigate to the project's root location
2. Execute this command: gradlew eclipse
3. The gradlew command is a wrapper around the gradle build tool and will download the binaries for the build tool before executing the task.
4. When gradle finishes (i.e. eclipse project files have been generated), you can import the project in eclipse by selecting import -¿ existing project from the eclipse menus.
5. Create sample folder to place the file to tansfer to different nodes.
   For Example: Like, /tmp/A, /tmp/B, /tmp/C
6. Execute the following command:
   gradlew run -ProotPath=/tmp

Notice the property rootPath that specifies the root path from where the threads will listen for files. The default path is still /tmp/id where id is the name of the host node

A. Assignment Topology

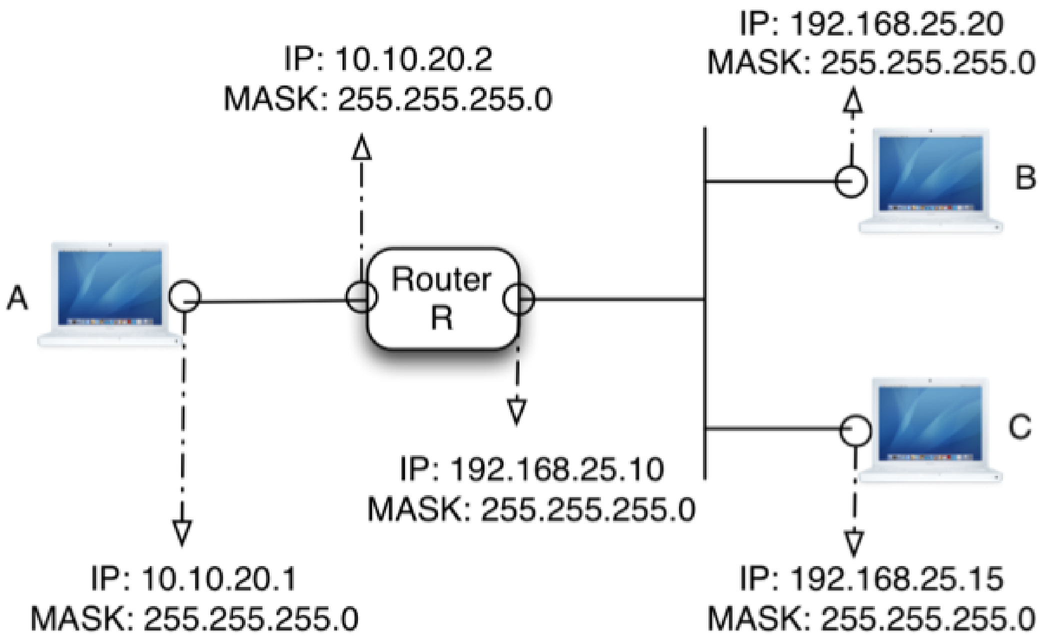

Figure 1: Network topology for the assignment

B. Topology Definition JSON

```json
{
  "nodes": [{
    "type": "host",
    "id": "A",
    "ip": "10.10.20.1",
    "mask": "255.255.255.0",
    "mac": "00:B0:D0:86:BB:F7",
    "mtu": 1400,
    "gateway": "10.10.20.2",
    "routing": [
      {"network": "192.168.25/24", "nextHop": "10.10.20.2"}
    ]
  },{
    "type": "host",
    "id": "B",
    "ip": "192.168.25.20",
    "mask": "255.255.255.0",
    "mac": "00:B0:D0:86:F7:BB",
    "mtu": 1400,
    "gateway": "192.168.25.10",
    "routing": [
      {"network": "192.168.25.15", "nextHop": "192.168.25.15"},
      {"network": "10.10.20/24", "nextHop": "192.168.25.10"}
    ]
  },{
    "type": "host",
    "id": "C",
    "ip": "192.168.25.15",
    "mask": "255.255.255.0",
    "mac": "00:B0:D0:F7:86:BB",
    "mtu": 1400,
    "gateway": "192.168.25.10",
    "routing": [
      {"network": "192.168.25.20", "nextHop": "192.168.25.20"},
      {"network": "10.10.20/24", "nextHop": "192.168.25.10"}
    ]
  },{
    "type": "router",
    "id": "R",
    "ports": [{
        "ip": "10.10.20.2",
        "mask": "255.255.255.0",
        "mac": "D0:B0:00:86:BB:F7",
        "mtu": 1400
      }, {
        "ip": "192.168.25.10",
        "mask": "255.255.255.0",
        "mac": "B0:00:D0:86:BB:F7",
        "mtu": 1400
      }
    ],
    "routing": [
      {"network": "10.10.20.1", "nextHop": "10.10.20.1"},
      {"network": "192.168.25.20", "nextHop": "192.168.25.20"},
      {"network": "192.168.25.15", "nextHop": "192.168.25.15"}
    ]
  }],
  "macTable": [
    {"ip": "10.10.20.1", "mac": "00:B0:D0:86:BB:F7"},
    {"ip": "192.168.25.20", "mac": "00:B0:D0:86:F7:BB"},
    {"ip": "192.168.25.15", "mac": "00:B0:D0:F7:86:BB"},
    {"ip": "10.10.20.2", "mac": "D0:B0:00:86:BB:F7"},
    {"ip": "192.168.25.10", "mac": "B0:00:D0:86:BB:F7"}
  ]
}
```

References

[1] Dr. Marco Carvalho, *CSE5231 - Class Project*, Florida Institute of Technology, Fall 2013.