Distributed Computing, Peer-to-Peer and GRIDS (ID2210)
Content Distribution Assignment

Amir H. Payberah Fatemeh Rahimian
payberah@sics.se fatemeh@sics.se

April 29, 2008

Contents

1 Introduction

The aim of this lab is to understand how a content distribution system works. You will be given a simulator on top of which you will simulate a peer-to-peer media streaming system. This document introduces the simulator, explains the relevant code for the assignment and describes what you have to do. This document has the following sections. In section 2, we will talk about the simulator and how it works. Reading this section is optional, but it is recommended to read it before doing your assignment. It helps you to have a better view on the structure of simulator. In section 3, we will explain the scenario of the system that you should implement. We will look at the source code of simulator and explain the classes that you need to know, in section 4, and in section 5 your tasks for this assignment will be explained.

2 Simulator

In this section we look at the different approaches on network simulation and then we talk about the structure of simulator that we are using for this assignment.

2.1 A few words on network simulation

Discrete-event simulation is a widespread technique for computer network simulation. In this approach, operation of a system is represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of the state in the system. Such simulators have a queue known as *Future Event List (FEL)*, which includes the events that are to be processed in the order of their start time.

A traditional model to simulate network traffic is *packet-level* simulation, which employs packet by packet modeling of network activities. In this model for each packet departure or arrival one event will be generated. Considering the effect of any single packet makes this approach accurate, but heavy weighted. If the size of network grows, huge number of events will be generated, which results in costly processing time and significant complexity. Therefore, packet-level simulation can not scale well.

Another model, which simplifies simulating network traffic is *flow-level* model. In this model the underlying layers of network are abstracted away and the events are generated only when the rate of flows change. This abstraction, enables simulations at large scale, but at the cost of losing accuracy. This is because the effects of underlying layers are ignored.

2.2 SICSSIM-B

SICSSIM-B, the simulator we will use for this assignment, is a *flow-level* and *discrete-event* model simulator. The network topology in SICSSIM-B consists

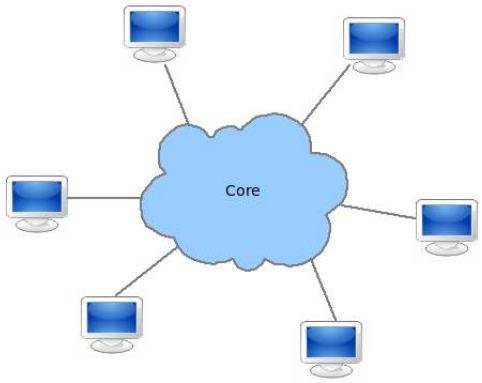

Figure 1: SICSIM-B overall structure

of *peer*, *link* and a *core network*. As depicted in Figure 1, each peer connects to the core network through a link with a specified capacity and latency.

To model bandwidth we draw two random values for incoming and outgoing bandwidth of each node. Then, we use a matrix to monitor data transfer rate between nodes. Each row in the bandwidth matrix is associated to one peer in the system and shows the upload rate of that peer to the other peers at an instant of time. Likewise, each column represents the downloading status of peers. Apparently, sum of the cells in each row should not be greater than the total outgoing bandwidth of that peer, and sum of the values in a column should not be greater than the total incoming bandwidth of the associated peer. Figure 2 shows a sample snapshot of the bandwidth matrix.

To model link latency, we calculate the latency between each pair of nodes

	A	B	C	D	Total Upload
A		56	0	0	56
B	128		1000	56	2000
C	256	1500		0	5000
D	0	0	56		56
Total Download	512	2000	10000	56	

Figure 2: Bandwidth matrix shows the download/uploads status of peers in Kbps.

as follows. We add the latency of the two links, which connect the peers to the core, to the latency of the core. The core latency itself consists of two parts: a

value, which is uniquely generated for each pair of nodes, and a random value, which represents the fluctuation in the network every time a data transfer takes place.

In SICSSIM-B we differentiate between the *control messages* and *data messages*. Data messages are the real data and can be of any type, e.g. video, audio, text, images, etc. For the sake of simplicity, we do not transfer real data in the simulator. We just assume there is a flow of data from node to node with a rate, specified in the bandwidth matrix, and a latency, calculated for those nodes. Although there is no real data, any change in the bandwidth matrices represents a change in a data flow. On the other hand, the control messages are considered to have a very small size, which would use zero bandwidth; but they will be queued in a list, i.e. FEL, which contains all the events of the system. Then an event scheduler let the simulator handle the control messages in the order of their increasing start time. The simulation loop proceeds by selecting the next event from FEL, executing it and inserting newly generated events in the queue.

The simulator has a *clock* to keep track of when to execute which event, and when to deliver a message sent earlier.

3 The Overlay Structure

The overlay that you should implement in this assignment, is consists of a number of *peers* and a *media server*, called *origin node*. The origin node is streaming the media, and we assume that it splits the media into a number of segments.

Each node in the system (peers and origin node) periodically broadcasts the *membership message* to others, to acknowledge that it is in the system. Each peer has a *partner list*, which is updated whenever the peer receives the membership message from other peers. The size of partner list can be changed according to your policy.

The peers and the origin node periodically multicast their *buffer map*, and their available upload bandwidth to their partners. The buffer map shows the segments that a peer has in its buffer. The peers after receiving this message, store it in their *data availability* list. The peers use this list to find the proper supplying peer for each segment. An important question here is, from which peer, which segment should be fetched.

A peer can start to play the media when it has the required segments in its buffer. We define the *playback point* as a segment number, which is playing at each peer. The peers should download the segments before their playback time. The scheduling algorithm plays an important role here.

4 Inside The Code

SICSSIM-B consists of different packages and classes. Here we only explain those classes that you need to know in order to do this assignment.

4.1 Package sicssim.coolstream.types

This package contains two data structures that are used in our algorithm.

▷ `MembershipMessage.java`: This class implements the membership message that peers broadcast periodically to show their presence in the system. It has two main fields:

- `id`: It is the node ID of a peer.
- `seqNum`: It is the sequence number of membership message. The membership message is broadcasted periodically, and it is possible that an older message will be received after the newer one. We can detect this situation by using this field.

▷ `PartnerInfo.java`: This class has two main fields:

- `bufferMap`: It shows the availability of segments in the buffer of a peer.
- `uploadBw`: It shows the available upload bandwidth of a peer at each time.

4.2 Package sicssim.coolstream.utils

This package contains three classes that can be used inside the peers and origin node.

▷ `Buffer.java`: Each peer pulls segments from other peers and puts them in their buffer. Some of the important methods of this class are as follow:

- `addSegment(int segment)`: It adds the segment `segment` to the buffer.
- `getPlaybackPoint()`: It returns the playback point of the media. The playback point is the number of segment that is playing.
- `containsSegment(int segment)`: It returns true if the buffer contains the segment `segment`, otherwise it returns false.
- `numOfMissedSegments()`: It returns total number of segments that could not be downloaded before their playback time.

▷ `DataAvailability.java`: This class implements a structure to store the buffer map and the upload bandwidth of peers. It has four important methods:

- `put(String node, PartnreInfo info)`: It adds the partner information, `info`, of peer `node` to its internal structure.

- `size()`: It returns the number of peers it has their data (their buffer map and upload bandwidth).
- `containsSegment(int segment)`: It return true if it finds a peer in its list who can provide the segment `segment`, otherwise it returns false.
- `findSupplier(int segment)`: It return the node ID of a peer from its list that can provide segment `segment`.

▷ `Broadcast.java`: Each peer can broadcast a message to all the peers in the system or multicast it to a group of peers. This class implements two methods for this purpose.

- `broadcast(Object data, Network network, BandwidthPeer peer)`: It broadcasts the message `data` to all peers in system. The `network` is a pointer to the overlay network and `peer` is a pointer to the peer who calls this method (you can use `this.network` for `network` field, and `this` for `peer` field, if you call this method inside a peer).
- `multicast(Object data, Set<String> group, BandwidthPeer peer)`: It multicasts the message `data` to all peers in `group`. The `peer` is a pointer to the peer who calls this method.

4.3 Package sicssim.coolstream.peers

This package contains two type of peers: *peer* and the *media server* or *origin node*. The origin node is streaming the media as a sequence of segments and the peers are trying to find the supplying peers and pull appropriate segments from them. These two type of classes are extended from `BandwidthPeer` class.

▷ `Peer.java`: Following are some of the useful methods of this class:

- `join(long currentTime)`: This method is called by simulator whenever a new node joins to the system. `currentTime` shows the current *clock* of the simulator.
- `leave(long currentTime)`: The simulator calls this method when a node decides to leave the system.
- `failure(NodeId failedId, long currentTime)`: We assume that each peer in the system has an eventual failure detector. So whenever a node in the system fails, the simulator calls this method of each peer in the system in a random time. `failedId` is the node ID of the failed peer and `currentTime` is the current *clock* of the system.
- `receive(NodeId srcId, Object data, long currentTime)`: This method is called whenever a new message received by a peer. `srcId` is the node ID of sender peer, and `data`, which is an instance of Object class, is the message sent by that peer. `currentTime` is the current *clock* of the system.

- `updatePeer(Object data, long currentTime)`: This method is an optional method that can be called at each *clock* of simulator. In our assignment, this is a good place to update playback pointer of the media stream.
- `pullSegment(NodeId partner, int segment)`: The peers use this method to pull the segment `segment` from `partner`.
- `registerEvents()`: This is the place that we can define different handler for different events.

▷ `OriginNode.java`: Most of the methods in this class are the same as what explained in `Peer.java`.

- `create(long currentTime)`: This method is called by simulator whenever the media server as a first node in the overlay comes to the system. `currentTime` shows current *clock* of the simulator.

4.4 Package sicssim.coolstream.auxiliaryoverlay

SICSSIM-B provides a middleware between the network layer and the overlay layer. The simulator uses this middleware to communicate directly with each peer of the overlay in each *clock*.

▷ `UpdateOverlay.java`: This class has one important method:

- `update(long currentTime)`: This method calls the `updatePeer` method of each peer in the network in each *clock*.

4.5 Other Packages and Classes

The packages that have been explained belong to *coolstream* package. Besides these packages, there are some other classes that you should know to do your assignment.

▷ `Data.java`: This is a data type that you can use to send message between peers. It has two main fields:

- `data`: It is an object of `Object` type.
- `type`: It is an object of `EventType` type.

We recommend to use this data structure to send control data between peers. But you are free to use any other structures you like.

▷ `BandwidthPeer.java and AbstractPeer.java`: As we explained earlier, `OriginNode.java` and `Peer.java` are extended from `BandwidthPeer.java`. As mentioned in section 2.2, there are two types of messages in SICSSIM-B: *data messages* and *control messages*. The data messages consume bandwidth, while

the control messages do not. To model sending data messages in simulator, we use one control message to show the start of sending data and another control message to show the end of sending data. `AbstractPeer.java` provides the methods for sending control messages and `BandwidthPeer.java`, which is extended from `AbstractPeer.java`, provides the methods for sending and receiving data messages. You can find these classes at `sicssim.peers` package. Some of the more important methods of these classes are as follows:

- `getId()`: This method returns the node ID of the peer.
- `sendControlData(NodeId destId, Data controlData)`: This method sends a control message `controlData` of type `Data` to another peer with `destId` address.
- `sendInstantData(Data controlData)`: This method is used by peers to send a message directly to the simulator. The difference between this method and `sendControlData` is that in this method the message is not sent through the network, so the message is received by simulator at the same time is was sent.
- `loopback(Data controlData, long time)`: This method is used by peers to send a control message to itself. The related handler is triggered after `time` clock.
- `startSendData(NodeId destId, Object msg)`: The peers use this method to show the start of sending data messages. If a peer does not have enough upload bandwidth to send the data, it return false, otherwise it returns true.
- `stopSendData(NodeId destId, Object msg)`: The peers use this method to show the stop of sending data messages.
- `getUploadBandwidth()`: This method returns upload bandwidth of a peer.
- `getDownloadBandwidth()`: This method returns download bandwidth of a peer.
- `getAvailableUploadBandwidth()`: This method returns available upload bandwidth of a peer.
- `getAvailableDownloadBandwidth()`: This method returns available download bandwidth of a peer.

You can add your own handler for these classes. For more information about how to do it, look at the first assignment.

▷ `Scenario.java`: This class is used to define the scenario. The only important field of this class that you should know is `ScenarioList`. You can define two types of event inside this field: `LotteryEvent` and `DelayEvent`.

The constructor for these two events are as follow:

- `LotteryEvent(long count, int time, int joins, int leaves, int failures, Class nodeType, Class linkType)`: This constructor has seven parameters as input. `count` is the number of events. `time` is the interval between each two events. `joins`, `leaves` and `failures` define the ratio of join, leave and failure of peers in the system. `nodeType` defines the type of node for this event. You can define different scenario for different type of peers. And `linkType` defines the type of link that the peer through it connects to the network.
- `DelayEvent(int time)`: The `time` defines the time that you want to have delay in simulation.

For example, to add events for two different types of peer, e.g. *peer* and *origin node*, with a delay event between them, we can implement it as bellow:

```
private ScenarioEvent ScenarioList[] = new ScenarioEvent[] {
   new LotteryEvent(1, 1, 1, 0, 0, OriginNode.class, ReliableLink.class),
   new DelayEvent(200),
   new LotteryEvent(100, 5, 10, 2, 1, Peer.class, ReliableLink.class),
}
```

▷ `Network.java`: This is one of the core classes of the simulator that maintains the nodes in the system. This class is located at `sicssim.network` package. The only method of this class that might be useful to know is:

- `getNodeList()`: This method returns a vector of current nodes in the system.

▷ `EventType.java`: In this class you can define different type of events for different purposes. This class belongs to `sicssim.types` package. There are two important events that you should know.

- `START_RECV_DATA`: The simulator knows this event. It consumes the bandwidth between two peers that are transferring data, when receives this message.
- `STOP_RECV_DATA`: The same as `STOP_RECV_DATA`, but the simulator release the bandwidth when it receives this event.

▷ `SicsSimConfig.java`: This is the configuration file of the system. You can configure the system from this file. It is placed in `sicssim.config` package. Some of its important parameters are as follow:

- `SIM_TIME`: It shows the duration of simulation. In another word, it shows the maximum value that `clock` can get.
- `MAX_NODE`: It defines the maximum number of peers that simulator can support.
- `SEGMENT_RATE`: It shows the bandwidth rate of each segment.

- `UPLOAD_BW and DOWNLOAD_BW:` These two values show the upload and download bandwidth of peers. The bandwidth is calculated as bellow:

```
uploadBandwidth = ((new Random()).nextInt(SicsSimConfig.UPLOAD_BW) + 1)
* SicsSimConfig.SEGMENT_RATE;
downloadBandwidth = SicsSimConfig.DOWNLOAD_BW * SicsSimConfig.SEGMENT_RATE;
```

- `NUM_OF_EVENT:` It shows number of events in scenario.
- `EVENT_INTERVAL:` The interval between each two events is defined by this parameter.
- `NUM_OF_JOIN, NUM_OF_LEAVE and NUM_OF_FAILURE:` These values show the ratio of join, leave and failure events.
- `MEDIA_SIZE:` It shows the size of media stream, or the number of segment in stream.
- `BUFFER_SIZE:` It defines the size of buffer in each peer.
- `BUFFERING_TIME:` It is the time that each peer uses to buffer the data before start palyback the media.

5 Your Task

For this assignment, you will have to implement the explained overlay in Section 3. As mentioned in lecture, you should find the answer to two main questions in each peer-to-peer content distribution network: *node discovery* and *data delivery*. Your task has two parts: in the first part assume that each peer knows all other peers in the system, so your task is only to find a solution for data delivery. In the second part you should complete your solution by implementing a proper way for node discovery. Note that:

- To design your scheduling, you can use the idea of coolstreaming\DONet algorithm, e.g. consider the playback deadline of each segment and also the available upload bandwidth of potential providers.
- We assume that each segment takes one second to be played, which is equivalent to 10 clocks in the simulator. So to play a media with 100 segments, 1000 clocks are required.
- You can not start to play a segment as soon as you start downloading it. You should wait until the whole segment is downloaded.
- We assume that each peer has enough download bandwidth to download all segments simultaneously. The upload bandwidth of peers are uniformly distributed between 1 to 5 times the segment size, and the upload bandwidth of the origin node is 8 times the segment size.

5.1 Task 1 - Data delivery

Your first task is to implement the data delivery between peers. As mentioned in section 3, each peer shows its existence in the network by broadcasting membership messages. The peers who receive a membership message, add the sender of that message to their partner list. In this task assume the size of partner list is big enough to maintain all peers in the network. By this assumption, after a while each peer knows all other peers in the network. In this task you should complete the source code, such that peers fetch the required data segments. The code contains a package for the assignment, called *sicssim.coolstream*. Complete the source where it is marked by *TODO*.

▷ `Peer.java`:

- `join(long currentTime)`: When a peer joins to the system, this method is called by simulator.
- `failure(NodeId failedId, long currentTime)`: When a peer fails, the simulator calls the `failure` method of all peers in the system and informs them the node ID of failed node.
- `handleScheduling()`: You should define from which peer, which segment should be fetched. It is important to design this scheduler such that each peer receives the segments before they playback point. The scheduling method should be called periodically. As a sample, the loopback method can be as follow:
  ```
  Data msg = new Data();
  msg.type = EventType.SCHEDULING;
  this.loopback(msg, SicsSimConfig.SCHEDULING_PERIOD);
  ```
- `handleSendBufferMap()`: Here the peer should multicast its buffer map and its upload bandwidth to the peers in its partner list.
- `handleRecvBufferMap()`: This method is called when a peer receives the buffer map of other peers.
- `handleSendMembershipMsg()`: Here the peer should broadcast the membership message to all peers in system.
- `handleRecvMembershipMsg()`: This method is called when a peer receives the membership messages of other peers.

▷ `OriginNode.java`:

- `create(long currentTime)`: The origin node is the first node in the system, and this is the first method called by simulator for this node.
- `failure(NodeId failedId, long currentTime)`: When a peer fails, the simulator calls the `failure` method of all peers in the system and informs them the node ID of failed node.

- `handleSendBufferMap()`: Here the peer should multicast its buffer map and its upload bandwidth to the peers in its partner list.
- `handleSendMembershipMsg()`: Here the peer should broadcast the membership message to all peers in system.
- `handleRecvMembershipMsg()`: This method is called when a peer receives the membership messages of other peers.

▷ `DataAvailability.java`:

- `findSupplier(int segment)`: You should find a supplier for the segment. Better heuristic better result.
- `containsSegment(int segment)`: To get a better result you can modify this method also.

To test your code, assume the media has 100 segments and there are 20 peers in the network, and the peers only join the system and they don't leave or fail. Configure the system to have partner size equal to the number of peers in system (`PARTNET_LIST_SIZE = 20`). The goal is to minimize the number of segments which miss their playback deadline at each node. The points you get for this part will depend on the efficiency of your solution **(7 points)**.

Then consider the peers can fail. You should handle the failures by completing the body of this `failure` method **(2 points)**.

5.2 Task 2 - Node discovery

In Task 1, we assume that the size of the partner list is equal to the number of nodes in the system. Now, assume the partner list size is half the number of nodes. What's the problem in this case and how would you solve it **(2 points)**? Implement your solution **(4 points)**.

6 Things to submit

You will have to show your assignments in the lab session on Friday 9/5. The course assistant will check your solution for different scenarios. After that, you will have to upload your source cod of three classes: `Peer.java`, `OriginNode.java` and `DataAvailability.java`, at the "assignment-submission" web-page for your group.

Good luck for the assignment ⌣̈