Implementing Multicasting and Broadcasting of Multimedia Data in ONE Simulator

Suman Bhattacharjee
Department of CST
IIEST, Shibpur
India
sumanbhattacharjee@acm.org

Ranit Chatterjee
Department of CST
IIEST, Shibpur
India
ranit4297@gmail.com

Tamal Pal
Department of CST
IIEST, Shibpur
India
tamalpal91@yahoo.co.in

Sipra DasBit
Department of CST
IIEST, Shibpur
India
sdasbit@yahoo.co.in

ABSTRACT

The Opportunistic Network Environment (ONE) simulator is a Java based simulator intended for research in Delay Tolerant Networks (DTNs). Apart from allowing users to simulate different DTN scenarios rapidly and in a versatile manner, it also offers an easy way to generate results from the simulations performed. Although the ONE simulator is widely accepted among the DTN research community as a standard DTN simulator, it supports only some limited features of DTN. The current release of the ONE simulator (version 1.6.0) only supports dissemination of messages with textual content over unicast mode of communication. The propagation of messages containing multimedia content and support for multicasting and broadcasting mode of communication are still not included in it. In this paper, we propose to implement the multicast and broadcast modes of communication and dissemination of multimedia content in the ONE simulator. We augment the ONE simulator version 1.5.1 RC-2 to incorporate the proposed features. Two sets of experiments are conducted to validate the implementation of the proposed features. The experimental results demonstrate successful implementation of both the features in ONE simulator.

CCS CONCEPTS

• **Networks** → **Network simulations**; • **Networks** → Network performance evaluation; • **Software and its engineering** → Simulator / interpreter

KEYWORDS

Delay Tolerant Networks, Simulations, ONE Simulator, Multicasting, Broadcasting, Multimedia content processing

1 INTRODUCTION

Delay tolerant networks (DTNs) are special types of mobile ad hoc networks which allow communication among intermittently connected mobile devices where end to end connectivity between the sources and destinations are non-existent. Apart from that, DTNs are suitable for implementation in environments which exhibit long or variable delay during message dissemination, asymmetric link bandwidth and high error rates due to frequent network disconnections. Due to these characteristics, DTNs are considered to be a viable option for communication in both urban scenarios where the cellular network is overloaded, as well as in scenarios where cellular infrastructure is incapacitated, such as in sparsely populated rural areas, battlefields and during disasters [1-4].

Multicast and broadcast mode of communication facilitates the dissemination of information to a group of users. Majority of contemporary DTN applications are designed to operate in a group-based manner and require support for group communication. For example, in rural areas circulations of the weather forecast and sophisticated cultivation techniques to farmers are essential. In a battlefield, soldiers need to inform each other about their surrounding environment. In a disaster recovery scenario, dissemination of situational information about victims and potential hazards among relief workers is crucial. Although group communication can be accomplished by disseminating separate unicast messages to each user, this approach suffers from elevated network overhead. The situation is especially acute in DTNs where resources such as connectivity among nodes, available bandwidth and storage are usually scarce. Moreover, in such application domains information exchange is necessary in heterogeneous file formats. Heterogeneity of file formats regarding information exchange implies dissemination of multimedia content (images, audio and video etc.) along with textual content over DTNs. Thus efficient implementations of multicast and broadcast mode of communications, as well as dissemination of multimedia content are indispensable in contemporary DTNs research.

Simulations play a crucial role in analyzing the behavior of DTN routing and application protocols. One hindrance for research on DTNs is the deficiency of appropriate simulators. Many simulators exist for MANETs (e.g. *ns2* [5], *ns3* [6] and *OMNeT++* [7]) and also for DTN routing (e.g. *dtnsim* [8] and *dtnsim2* [9]). The MANETs simulators offer limited DTN support whereas the *dtnsim* concentrate solely on routing simulation. Further, the performance of DTN routing and application protocols fluctuate significantly, depending on the mobility pattern of the nodes, density of the node population, and the

SIMUTOOLS '17, September 11–13, 2017, Hong Kong, China

ACM ISBN 978-1-4503-6388-4/17/09…$15.00
https://doi.org/10.1145/3173519.3173540

distance between the sender and the receiver. Therefore, there is an acute need for a simulator that has realistic mobility modelling capabilities, integrated support for DTN routing, and simultaneous visualization of the simulation progress and results. ONE simulator [10] is equipped with all these necessary functionalities. In spite of the aforesaid advantages, ONE simulator supports the dissemination of messages with textual content over the unicast mode of communication only. In this work, we implement the multicasting and broadcasting of multimedia content in ONE simulator version 1.5.1 RC-2.

The rest of the paper is organized as follows. Section 2, summarizes related work in this field. A brief overview of the ONE simulator is presented in Section 3. The implementation of multicasting and broadcasting is elaborated in Section 4. Section 5 describes the processing of multimedia content. Section 6 provides experimental results to validate our proposed implementation. We conclude the paper with a direction towards future work in Section 7.

2 RELATED WORK

Substantial amount of modifications has been implemented in ONE simulator till date. However, the majority of those modifications are aimed at implementing different routing protocols, mobility models or content distribution mechanism. To the best of our knowledge, the implementation of multicasting and broadcasting of multimedia data in ONE simulator has not been reported. Some of the important works which report modifications in the ONE simulator are discussed below.

Lindgren et al. [11] implement their proposed probabilistic routing protocol (PRoPHET). The proposed protocol is able to increase message delivery probability by selecting suitable forwarder node based on the history of prior encounters. Nelson et al. [12] implement encounter based routing (EBR) which maximizes delivery ratio and minimizes network overhead and delay and offers security against the black hole and DoS attack. Garsic et al. [13] has modified the PRoPHET routing protocol and introduces PRoPHETv2 which has improved the performance of the PRoPHET especially in heterogeneous mobility scenario in terms of message delivery ratio and network overhead. Moreira et al. [14] implement a Social-aware Content-based Opportunistic Routing Protocol (SCORP) that considers the users' social interaction and their interests to improve data delivery in urban, dense scenarios. All the aforesaid routing protocols are designed to disseminate textual contents in the unicast mode of communication.

Desta et al. [15] describe the implementation of content distribution mechanism among fixed nodes as well as mobile nodes in ONE simulator. Uddin et al. [16] has implemented a post disaster mobility model in ONE simulator.

Popular network simulators for MANETs (e.g. *ns2* and *OMNeT++*) inherently support the multicast and broadcast mode of communication. For instance, *ns2* supports three types of multicast route computation strategies (e.g. Centralized, Dense and Shared Tree Mode). It uses a configurable parameter viz. the *multicast* mode during simulation setup in order to enable multicasting/broadcasting. Veselý et al. [17-18], describe the implementation details of IPv4 multicasting in *OMNeT++*. The implementation uses IGMP to manage membership in multicast groups. DTN does not support IPv4 addressing. Therefore, the popular IP based multicast group management protocols like IGMP remain infeasible for implementation in DTN.

The abovementioned discussion reveals that, the majority of the modification works in the ONE simulator are aimed at implementing routing protocols for disseminating textual content in the unicast mode of communication. These limitations motivate us to implement multicasting and broadcasting of multimedia content in the ONE simulator. The main contributions of our work are outlined as follows:

- Enhance the ONE simulator to implement multicast and broadcast mode of communication.
- Enable dissemination of multimedia content over the simulator.
- Validate the implementation of the proposed features using realistic simulation.

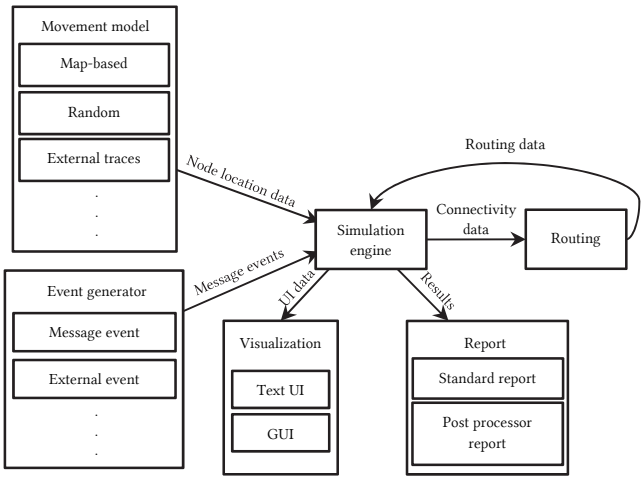

Figure 1: ONE simulator modules and their interactions.

3 OVERVIEW OF ONE SIMULATOR

In this section, we describe a brief overview of the ONE simulator along with its architecture. The ONE is an agent based discrete event simulator written in Java. Figure 1 depicts the interaction among different modules within it. It consists of six major modules viz. movement model, event generator, simulation engine, routing, visualization and report.

The movement model module contains ten classes which correspond to ten different mobility models that dictate how the nodes move during simulation. The node mobility can be controlled either by using the integrated movement models or by importing movement data from external sources. This module produces node location data and forwards this data to the simulation engine. The node location data produced by the movement model is used for determining if two nodes can

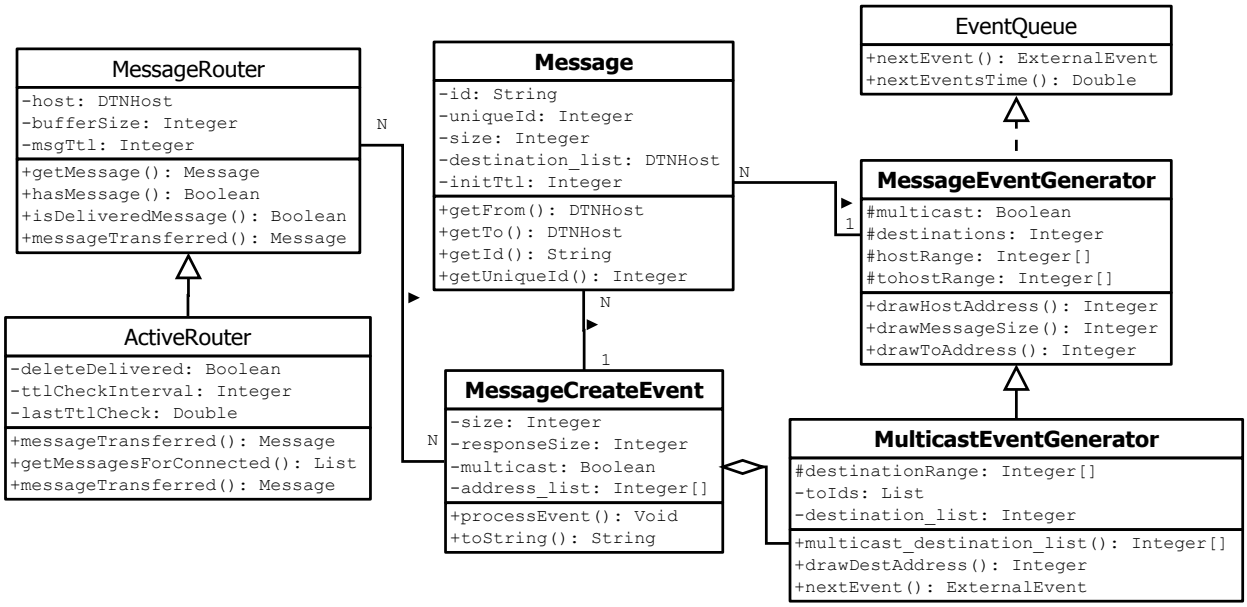

Figure 2: Class diagram of multicast/broadcast implementation.

communicate and exchange messages. This connectivity data produced by simulation engine is forwarded to the routing module. The message events are generated through event generator module. This module controls different configurable parameters during message creation like creation time, message size, destination range, frequency of message creation etc. Themessages are always unicast, having a single source and destination inside the simulation world. The routing module consists of fourteen classes corresponding to different DTN routing protocols e.g. Epidemic, PRoPHET, MaxProp, Spray and Wait, etc. The routing module performs operations on the messages on their own, but they can also be commanded using event generator modules or external traces. The movement modelling and routing simulation are displayed in the simulator's visualization module. The report module assembles results of the simulation for further analysis or interaction with other programs.

The simulation scenarios contain a variable number of wireless nodes having heterogeneous movement speed and transmission range. Based on the movement speeds and transmission range the nodes are assembled in node groups. Each group shares a set of common configurable parameters such as message buffer size, routing protocol, mobility model, number of nodes, etc. Since different node groups can have different configurations, creating a simulation scenario with node groups e.g. pedestrians, cars, public transportation, etc. is possible. All movement models, report modules, routing protocols and event generators are dynamically loaded into the simulator. This feature enables the extension and configuration the simulator with different types of plugins much easier. The implementation details for multicasting, broadcasting and processing of multimedia data is described in detail in the subsequent sections.

4 IMPLEMENTING MULTICASTING AND BROADCASTING

In this section, we illustrate the implementation details of multicasting and broadcasting modes of communication in the ONE simulator. In general multicasting/broadcasting refers to the forwarding of messages to a group of nodes/all nodes in the network. However, due to the frequent network partitioning, long transmission delay and node mobility in DTN, group membership of nodes changes during data transfer regularly. Hence, instead of making groups as destinations, node addresses are treated as destinations for multicasting/broadcasting in our implementation. Our proposed implementation consists of six classes. Out of these six classes, three classes (*MulticastEventGenerator*, *MessageEventGenerator* and *MessageCreateEvent*) are associated with the event generator module. Two classes (*MessageRouter* and *ActiveRouter*) are linked with the routing module. The *Message* class is connected with the simulation engine module. Figure 2 presents the class diagram of our proposed implementation. The implementation of multicasting and broadcasting in the ONE simulator is organized as follows: developing multicast event generator, modifying routing module and modifying the settings file.

4.1 Developing multicast event generator

As discussed in section 3, messages generated in the ONE simulator are always propagated using unicast mode of communication. Therefore, implementation of multicasting and broadcasting requires developing a new message event generator which is capable of creating messages with multiple destinations. We have developed the *MulticastEventGenerator* class to generate message events from a single source to a predefined set

of destinations. Figure 3 describes the *MulticastEventGenerator* constructor. The *MulticastEventGenerator* class is a subclass of *MessageEventGenerator* class which is responsible for creating uniformly distributed message events. The subclass has various configurable parameters such as message size, first messages time, message source, message destinations and the intervals

```
public MulticastEventGenerator(Settings s) {
                  super(s);
                  this.destinationRange =
s.getCsvInts(MULTICAST_DEST_RANGE, 2);
                  this.toIds = new ArrayList<Integer>();
                  if (toHostRange == null) {
                           throw new
SettingsError("Destination host (" + TO_HOST_RANGE_S + ")
must be defined");
                  }

                  for (int i = toHostRange[0]; i <
toHostRange[1]; i++)
                           toIds.add(i);
                  if ((destinationRange[1]-
destinationRange[0]) != destinations){
                           throw new
SettingsError("Destination address range (" +
MULTICAST_DEST_RANGE + ") and number of destination (" +
MULTICAST_DEST_S + ") must be identical");
                  }
                           destination_list = new
int[]{destinationRange[0], destinationRange[1]};
                  Collections.shuffle(toIds, rng);
         }
```

Figure 3: *MulticastEventGenerator* constructor.

between creating messages, etc. We have introduced a new parameter, *MULTICAST_DEST_RANGE* to hold the multicast/broadcast destination addresses list. Further, it has two methods viz. *drawDestAddress()* and *multicast_destination_list()*. These two methods are designed for picking up the user defined destination range from the settings file and verification of destination address range respectively. The *MessageCreateEvent* class is designed to generate message events based on the parameters supplied by the user. This class accepts eight configurable parameters from the settings file to generate message events such as *from*, *to*, *id*, *size*, *responseSize*, *time*, *multicast*, *destination_list*.

```
public MessageCreateEvent(int from, int to, String
id, int size, int responseSize, double time, boolean
multicast,int destination_list[]) {
          super(from,to, id, time);
          this.size = size;
          this.responseSize = responseSize;
          this.multicast = multicast;
          this.destination_list = destination_list; }
```

Figure 4: Modified *MessageCreateEvent* constructor.

We have added two parameters (*multicast* and *destination_list*) in the *MessageCreateEvent* constructor to support multicasting/broadcasting. Figure 4 presents the modification applied on the *MessageCreateEvent* constructor.

The *Message* class in simulation engine module is responsible for creating messages with different attributes provided by the users in the settings file. In order to facilitate multicasting/broadcasting, the destination address field of the

```
public Message(DTNHost from, DTNHost to, String id,
int size,ArrayList<DTNHost> destination_list) {
this.from = from;
this.to = to;
this.id = id;
this.size = size;
this.path = new ArrayList<DTNHost>();
this.uniqueId = nextUniqueId;
this.timeCreated = SimClock.getTime();
this.timeReceived = this.timeCreated;
this.initTtl = INFINITE_TTL;
this.responseSize = 0;
this.requestMsg = null;
this.properties = null;
this.appID = null;
this.address_list = destination_list;
Message.nextUniqueId++;
addNodeOnPath(from);}
```

Figure 5: Modified *Message* constructor.

message header must contain a set of addresses instead of a single address. Therefore, we have modified the Message class constructor and included the destination address list in it. Figure 5 presents the modified *Message* constructor.

4.2 Modifying routing module

The ONE simulator traditionally support unicast mode of communication. Therefore, the messages are intended for a single destination and when the messages reach to those destinations they are not propagated further. The situation is different for multicasting/broadcasting. Here, propagation of messages is not terminated upon reaching to a single destination. This implies, the messages are copied to one destination node and then disseminated to multiple other destinations in parallel. Hence, modification of the routing module is indispensable. The routing module in ONE simulator consists of two abstract classes (*MessageRouter* and *ActiveRouter*). The *MessageRouter* class contains *messageTransferred()* method responsible for terminating messages once reached to the intended destinations.

We have modified the *messageTransferred()* method and introduce a destination list in it. Hence, the messages intended for multicasting/broadcasting are not terminated upon reaching to a single destination.

The *ActiveRouter* class inherits the *MessageRouter* class. It includes *getMessagesForConnected()* method responsible for returning a list of message-connections tuples at any instant of time. Modifications are made in this method in order to implement multicasting/broadcasting.

```
Events2.class =MulticastEventGenerator
Events2.interval = 1
Events2.size = 50k, 250k
Events2.hosts = 2,2
Events2.prefix = INV_MW2_
Events2.multicast = true
Events2.nrofDestination = 23
Events2.destinationRange = 1,24
```

Figure 6: Modified settings file.

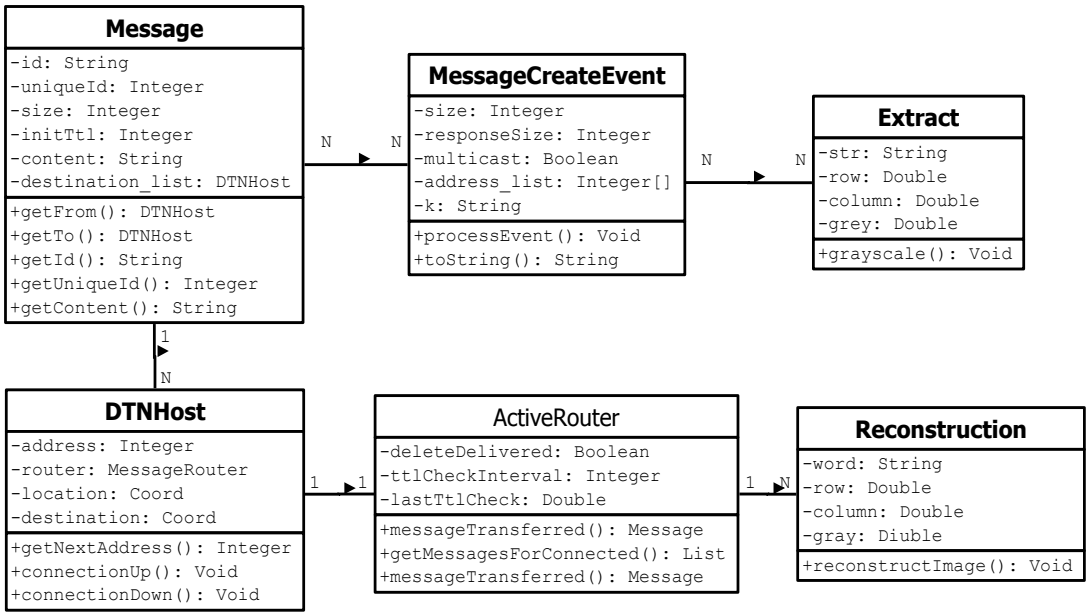

Figure 7: Class diagram of processing multimedia data.

4.3 Modifying the settings file

Figure 6 describes the snapshot of a sample settings file for implementing multicasting/broadcasting in the ONE simulator. This file stores the values of different configurable parameters.

We have included three parameters (*multicast*, *nrofDestination* and *destinationRange*) in the settings file. The *multicast* is a Boolean parameter which can take either true or false values. If the value of *multicast* is set as false the simulator works in the unicast mode. The *nrofDestination* is used to identify the number of multicast destinations. If the *nrofDestination* parameter contains the count of all nodes in the simulation environment the simulator works in the broadcasting mode. The *destinationRange* parameter indicates the range of the destination addresses.

5 PROCESSING MULTIMEDIA DATA

In this section we illustrate the implementation details of multimedia content (image) processing in the ONE simulator. This simulator can only handle messages containing textual data. It does not support the propagation of image files. Therefore, to address this limitation we have transformed the image files into text files. The conversion has been performed by transforming the image into grayscale and storing the grayscale luminance values in a text file. We have integrated *OpenCV* (Open Source Computer Vision Library version 2.4.13) in the ONE simulator for this purpose. *OpenCV* is an open source computer vision and machine learning software library written in C++.

Our proposed implementation for multimedia data processing is organized as follows: extracting text from image, modifying simulation engine module and reconstructing image at the destinations. Figure 7 depicts the class diagram of processing multimedia data in ONE simulator. The proposed implementation consists of six classes. Out of these six classes, two classes (*ActiveRouter* and *Reconstruct*) are associated with the routing module of the ONE simulator. Two classes (*DTNHost* and *Message*) are associated with the simulation engine module and two classes (*MessageCreateEvent* and *Extract*) are linked with the event generator module.

```
public static void grayscale()throws IOException{
     System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
     Mat frame=new Mat();int i=0,j=0;
     String strr,str,strrr;double row,col,gray;
     String k;
     strr="input/content/image.jpg";
     frame = Highgui.imread(strr);
     Size size =frame.size();
     row=size.height;col=size.width;
     FileOutputStream fos = new
FileOutputStream("input/content/image_data.txt");
     OutputStreamWriter osw=new OutputStreamWriter(fos);
     BufferedWriter bw=new BufferedWriter(osw);
     for (i = 0; i <row; i++) {
        for (j = 0; j <col; j++) {
           double[] data = frame.get(i, j);
           gray=(data[0]+data[1]+data[2])/3;
           k=Double.toString(gray);
           bw.write(k+" ");
        }
     }
     bw.close();
     osw.close();
     fos.close();
}
```

Figure 8: The *grayscale()* method of Extract class.

5.1 Extracting Text from Image

In this section we present the procedure to extract text from image file and generate messages containing those text data. We have developed *Extract* class for this purpose. This class contains *grayscale()* method responsible for extracting grayscale luminance values from red, green and blue (RGB) luminance values and storing the grayscale luminance values in a text file. The transformation of RGB to grayscale is achieved by capturing the arithmetic mean of the RGB luminance values for each pixel. Figure 8 describes the *grayscale()* method. As the figure explains, the *gray* variable stores the arithmetic mean of the RGB luminance values for each pixel and writes them in the *image_data.txt* file.

As mentioned in section 4.1 the *MessageCreateEvent* class is designed to generate message events based on the parameters supplied by the user through the settings file.

We have modified the *MessageCreateEvent* constructor to facilitate the creation of messages containing data from the *image_data.txt* file. Figure 9 depicts the modified *MessageCreateEvent* constructor. The *MessageCreateEvent* class creates text files extracted form images for each message generating nodes. The contents of those text files are appended in the corresponding message body during generation of the messages.

```
public MessageCreateEvent(int from, int to, String id, int
size, int responseSize, double time, boolean multicast, int
destination_list[]) {
         super(from,to, id, time);
         this.size = size;
         this.responseSize = responseSize;
         this.multicast = multicast;
         this.destination_list = destination_list;
         String k;
         if(id.startsWith("M")){
                  try {
                           extract.grayscale();
                           s =  new Scanner(new
File("input/content/image_data.txt"));
                           content = s.nextLine();
                  }
                  catch(Exception e){
                           System.out.println("ERROR");
                  }
                  try{
                           k=Integer.toString(from);
                           FileOutputStream fos = new
FileOutputStream("Game/"+k+"/image_data.txt");
                           OutputStreamWriter osw=new
OutputStreamWriter(fos);
                           BufferedWriter bw=new
BufferedWriter(osw);
                           bw.write(content);
                           bw.close();
                  }
                  catch(Exception e){
                           System.out.println("ERROR");
                  }
         }
}
```

Figure 9: Modified *MessageCreateEvent* constructor.

5.2 Modifying simulation engine module

In this section we describe the modifications made in the simulation engine module to facilitate propagation of the extracted text files through the ONE simulator. The *DTNHost* class is responsible for creating new DTN nodes in the ONE simulator. We have modified the *DTNHost* constructor to facilitate the creation of folders having names of the DTN nodes in order to track the message propagation. The folders having the names of the message generator nodes and relay nodes contain text files extracted from images. The folders having the names of destination nodes consist of text files and reconstructed image files.

```
public Message(DTNHost from, DTNHost to, String id, int
size,ArrayList<DTNHost> destination_list, String content){
                  this.content=content;
                  this.address_list = destination_list;
                  Message.nextUniqueId++;
                  addNodeOnPath(from);
}
```

Figure 10: Modified *Message* constructor.

The *Message* class is responsible for generating messages. We have modified the *Message* constructor to append data from the text files to the message body. The original ONE simulator does not support the propagation of message content. It only uses a configurable parameter *size* to represent the existence of textual content. Figure 10 presents the modifications performed in the *Message* constructor. As the figure explains, we have introduced a string variable *content* in the constructor. This variable retrieves the textual representation of the image from *image_data.txt*. We have further developed a method *getContent()* in the *Message* class to fetch the content from the messages.

5.3 Reconstructing image file at destinations

Here, we illustrate the procedure of reconstructing the image files at the destinations. The routing module is responsible for propagation and delivery of messages to the destination. In our proposed implementation, reconstructions of images form text are carried out at destinations. Hence, we have incorporated the reconstruction mechanism in the routing module of the ONE simulator.

The *ActiveRouter* class of the routing module contains the *messageTransferred()* method responsible for terminating messages once reached to the intended destinations. Figure 11 indicates the modification performed in the *messageTransferred()* method to facilitate reconstruction of the image file at the destination. The figure indicates that *reconstructImage()* method is invoked once the message reaches to the destination.

We have developed the *Reconstruction* class which reconstructs image file from a text file. The *Reconstruction* class contains the *reconstructImage()* method. This method accepts the data from the extracted text file and reconstructs the corresponding image file. The reconstructed image file is stored in the folders having names of the destination nodes.

```
public Message messageTransferred(String id, DTNHost from) {
     Message m = super.messageTransferred(id, from);
     String data=m.content;
     String to=getHost().toString();
     System.out.println(to);
     try{
         FileOutputStream fos = new
FileOutputStream("Game/"+to+"/image_data.txt");
         OutputStreamWriter osw=new OutputStreamWriter(fos);
         BufferedWriter bw=new BufferedWriter(osw);
         bw.write(data);
         bw.close();
         if(getHost()== m.getTo()){
             System.out.println("Reconstruction Started");
             reconstruction.reconstructImage(to);
         }
     }
     catch(Exception e){
         System.out.println("Message delivery error.");
     }
     if (m.getTo() == getHost() && m.getResponseSize() > 0) {
         Message res = new
Message(this.getHost(),m.getFrom(),
RESPONSE_PREFIX+m.getId(), m.getResponseSize(), m.content);
                          this.createNewMessage(res);
         this.getMessage(RESPONSE_PREFIX+m.getId()).setReques
t(m);
         }

     return m;
}
```

Figure 11: Modified *messageTransferred()* method.

6 PERFORMANCE EVALUATION

We conduct two sets of experiments to evaluate the performance of the proposed implementation of the multicasting and broadcasting of multimedia data in the ONE simulator. In the first set of experiments, the performance of multicasting and broadcasting of textual content in the ONE simulator is evaluated. In the second set of experiments, we evaluate the performance of multimedia content forwarding (multicasting/broadcasting) through the simulator.

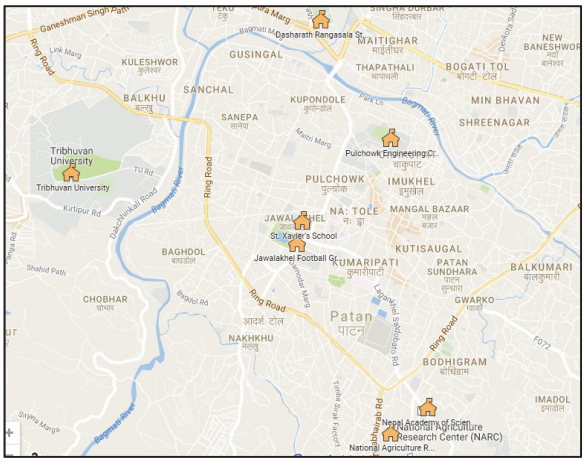

Figure 12: Unofficial crisis map of Nepal earthquake. [19]

6.1 Simulation setup

The simulation environment is configured based on a snapshot of the earthquake that occurred in Nepal on April, 2015. We create our simulation set-up in ONE simulator [10], version 1.5.1-RC2, based on information provided by the unofficial crisis map [19] regarding post-disaster relief operation carried out in Nepal, at the Kathmandu area. Figure 12 presents a snapshot of the unofficial crisis map of Nepal earthquake. On the map of the disaster affected area of Kathmandu, seven camps are set up within an area of 8.5 Square KM. Five volunteers are associated with each camp. So, there are two types of nodes (fixed and mobile) in our simulation scenario. Camps are fixed nodes and volunteers are mobile modes. We consider PRoPHET routing protocol [11] and post-disaster mobility model (PDM) [16], for the simulation. PDM proposed by Uddin et al. describes the movement pattern of victims and rescue as well as relief workers in a post-disaster situation. Other important simulation parameters are provided in Table 1.

Table 1: Parameter values used in simulation

Parameters	Values
Simulation Time	8 Hrs.
Number of nodes	42
Transmission Range	100 meters (High-speed interface)
Buffer Size	20 MB
Message TTL	720 min
Message Generation rate	One in every 12 hours (multicasting/broadcasting of textual content) One in every 2.5 hours (multicasting/broadcasting of multimedia content)
Mode of communication	Unicast, Multicast and Broadcast

6.2 Results and discussion

In the first set of experiments, messages (textual content) are generated from each camp nodes periodically (say once a day). Disaster relief and rescue operations are generally carried out for 12 hours in a day during 6 a.m. to 6 p.m. So, once a day actually implies a message in every 12 hours. The experiments are conducted in three modes viz. unicast, multicast and broadcast mode. We assume the existence of seven camp nodes in our simulation scenario. Therefore, the number of messages generated from the camp nodes for each of the three modes of communication is seven. We assume messages are disseminated to seven random nodes and twenty two specified nodes in unicast and multicast mode respectively.

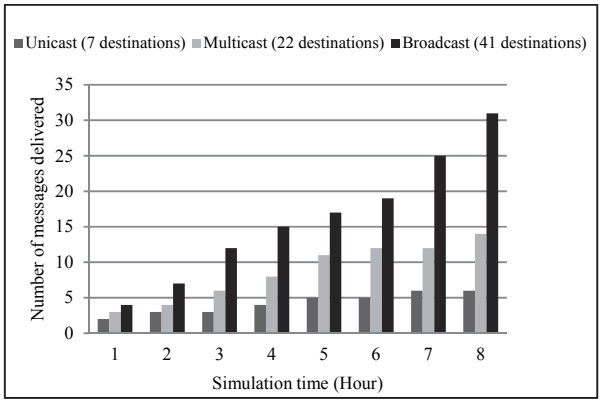

Figure 13: Comparative performance of three modes of communication.

Figure 13 compares the performance of unicast, multicast and broadcast mode of communication in terms of the number of messages delivered. As the figure explains, the number of messages delivered to the destination through multicast and broadcast mode in a stipulated time period is much higher compared to the unicast mode of communication. We further observe that the number of delivered messages is much higher than the number of created messages in multicast and broadcast modes of communication. The aforesaid outcome is only possible if a single message is delivered to multiple destinations. This implies the successful implementation of multicasting/broadcasting in the ONE simulator.

In the second set of experiments, messages having multimedia content (image files) are generated from seven arbitrarily chosen volunteer nodes in every 2.5 hours. Hence, the total number of generated messages within a simulation interval of 8 hours is 21. Each message is disseminated to a randomly chosen volunteer node in unicast mode, a group of fifteen volunteer nodes in multicast mode and in broadcast mode.

Figure 14(a) and 14(b) demonstrates the comparison of message stats reports during the dissemination of the textual and multimedia content after eight hours of simulation in all three modes. The figures clearly indicate identical phenomena during both the textual and multimedia content dissemination (i.e. the number of delivered messages is much greater than the number of created messages in multicast and broadcast modes of communication). This result is a clear indicator of successful dissemination of multimedia content through multicast and broadcast mode. Hence, the figure 14(a) and 14(b) undoubtedly validate our claim of successful dissemination of multimedia content in three modes (unicast, multicast and broadcast) of communication.

```
sim_time: 28800.1000
created: 7
started: 105
relayed: 92
aborted: 7
dropped: 0
removed: 0
delivered: 6
latency_avg: 6218.4513
latency_med: 4716.6000
hopcount_avg: 2.4882
hopcount_med: 2
```

Unicast mode

```
sim_time: 28800.0000
created: 7
started: 293
relayed: 254
aborted: 23
dropped: 0
removed: 0
delivered: 14
latency_avg: 2342.7524
latency_med: 1396.1000
hopcount_avg: 2.2451
hopcount_med: 2
```

Multicast mode

```
sim_time: 28800.0000
created: 7
started: 313
relayed: 277
aborted: 36
dropped: 0
removed: 0
delivered: 31
latency_avg: 2355.7386
latency_med: 1436.1000
hopcount_avg: 2.2362
hopcount_med: 2
```

Broadcast mode

(a)

```
sim_time: 28800.0000
created: 21
started: 424
relayed: 679
aborted: 56
dropped: 0
removed: 0
delivered: 16
latency_avg: 3245.2374
latency_med: 2987.5000
hopcount_avg: 1.4823
hopcount_med: 1
```

Unicast mode

```
sim_time: 28800.0000
created: 21
started: 306
relayed: 299
aborted: 7
dropped: 0
removed: 0
delivered: 131
latency_avg: 4280.1154
latency_med: 3115.6000
hopcount_avg: 1.4615
hopcount_med: 1
```

Multicast mode

```
sim_time: 28800.0000
created: 21
started: 352
relayed: 303
aborted: 15
dropped: 0
removed: 0
delivered: 459
latency_avg: 3678.3072
latency_med: 3274.4000
hopcount_avg: 1.3829
hopcount_med: 1
```

Broadcast mode

(b)

Figure 14: Message stats report of three modes of communication after eight hours of simulation for disseminating (a) textual content and (b) multimedia content.

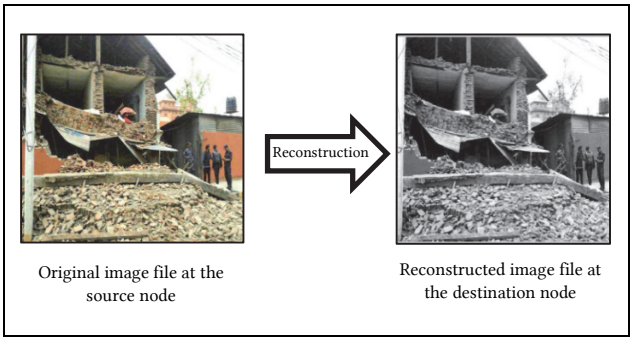

Figure 15: Snapshots of original and reconstructed image files.

Figure 15 presents a visual comparison of two image files (original and reconstructed) at the source and destination node respectively.

Thus, the two sets of simulation results clearly validate our claim of implementing multicasting and broadcasting of multimedia data in the ONE simulator.

7 CONCLUSION

In this work, we present the implementation details of multicasting and broadcasting mode of communication as well as the dissemination of multimedia data in the ONE simulator. We enhance the ONE simulator version 1.5.1 RC-2 in order to incorporate the proposed features. The enhancement has been made primarily in three modules (i.e. event generator, routing and simulation engine) of the simulator. Two sets of experiments are conducted to validate the success of the proposed implementation. While evaluating the performance, we focus on two aspects i) evaluating the performance of multicasting and broadcasting of textual content, ii) validating multimedia content dissemination through the multicasting and broadcasting mode of communication. The experimental results clearly indicate the successful implementation of multicasting and broadcasting and dissemination of multimedia data in the ONE simulator.

In spite of the proposed enhancements, the simulator has several limitations. The propagation of video and audio files is still not supported in the ONE simulator. Hence, processing audio and video files in the ONE simulator may be implemented as a part of future work.

ACKNOWLEDGEMENT

This work is supported by Information Technology Research Academy (ITRA), Government of India under, ITRA-Mobile grant [*ITRA/15(58)/Mobile/DISARM/01/Rev/2015*].

We acknowledge all support rendered by Mr. Sahil Gupta through his website that has been instrumental in this work. (Link: https://sites.google.com/site/sahilgupta221231/file-cabinet)

REFERENCES

[1] Fall, K. 2003. A delay-tolerant network architecture for challenged internets. In *Proceedings of the SIGCOMM*, pp.27-34.

[2] Fall, K., Iannaccone, G., Kannan, J., Silveira, F. and Taft, N. 2010. A Disruption-Tolerant Architecture for Secure and Efficient Disaster Response Communications. In *Proceedings of the ISCRAM*.

[3] Bhattacharjee, S., Roy, S. and Bandyopadhyay, S., 2015. Exploring an energy-efficient DTN framework supporting disaster management services in post disaster relief operation. *Wireless Networks*, 21(3), pp.1033-1046.

[4] Trifunovic, S., Kouyoumdjieva, S.T., Distl, B., Pajevic, L., Karlsson, G. and Plattner, B., 2017. A Decade of Research in Opportunistic Networks: Challenges, Relevance, and Future Directions. *IEEE Communications Magazine*, 55(1), pp.168-173.

[5] The Network Simulator NS-2. http://www.isi.edu/nsnam/ns/. Last Accessed: 2017-05-14.

[6] The Network Simulator NS-3. https://www.nsnam.org/. Last Accessed: 2017-05-14.

[7] Varga, A., 2001. Discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, pp 319-324.

[8] Jain, S., Fall, K. and Patra, R., 2004. Routing in a delay tolerant network, Vol. 34, No. 4, pp. 145-158.

[9] University of Waterloo. DTNSim2 Delay-tolerant Network Simulator. https://ex001slvr.wordpress.com/category/dtnsim2/. Last Accessed: 2017-05-14.

[10] Keranen, A., Ott, J. and Karkkainen, T., 2009. The ONE simulator for DTN protocol evaluation. In *Proceedings of the SIMUTOOLS*, paper id. 55.

[11] Lindgren, A., Doria, A. and Schelen, O., 2004. Probabilistic routing in intermittently connected networks. In *Proceedings of the Service assurance with partial and intermittent resources (SAPIR)*, pp. 239-254.

[12] Nelson, S.C., Bakht, M. and Kravets, R., 2009. Encounter-based routing in DTNs. In *Proceedings of the INFOCOM 2009*, pp. 846-854.

[13] Grasic, S., Davies, E., Lindgren, A. and Doria, A., 2011. The evolution of a DTN routing protocol-PRoPHETv2. *In Proceedings of the 6th ACM workshop on Challenged networks*, pp. 27-30.

[14] Moreira, W., Mendes, P. and Sargento, S., 2013. Social-aware opportunistic routing protocol based on user's interactions and interests. In *Proceedings of the International Conference on Ad Hoc Networks*, pp. 100-115.

[15] Desta, M.S., Hyytiä, E., Keränen, A., Kärkkäinen, T. and Ott, J., 2013. Evaluating (Geo) content sharing with the ONE simulator. In *Proceedings of the 11th ACM international symposium on Mobility management and wireless access*, pp. 37-40.

[16] Uddin, M.Y.S., Nicol, D.M., Abdelzaher, T.F. and Kravets, R.H., 2009, December. A post-disaster mobility model for delay tolerant networking. In *Proceedings of the Winter Simulation Conference*, pp. 2785-2796.

[17] Veselý, V., Ryšavý, O. and Švéda, M., 2013. IPv6 unicast and IPv4 multicast routing in OMNeT++. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pp. 346-349.

[18] Veselý, V., Matoušek, P. and Švéda, M., 2013. Multicast simulation and modeling in OMNeT++. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pp. 142-145.

[19] Google unofficial crisis map. https://www.google.com/maps/d/viewer?mid=zjU92k9XcdHk.kGerGuVhu2R0. Last Accessed: 2017-05-14.