CMPT 885-3: SPECIAL TOPICS: HIGH-PERFORMANCE NETWORKS

Investigation and Implementation of Reliable Multicast Transport Protocols using ns-2

Spring 2001

FINAL PROJECT

Chao Li, Cheng Lu, and Thomas Su

WWW page (not available)

{clij, clu, tmsu}@sfu.ca

ABSTRACT

Widespread availability of IP multicast has substantially increased the geographic span and portability of collaborative multimedia applications. Examples of such applications include distributed shared whiteboards, group editors, and distributed games or simulations. Such applications often involve many participants and typically require a specific form of multicast communication in which a single sender must reliably transmit data to multiple receivers. IP multicast provides scalable and efficient routing and delivery of IP packets to multiple receivers. However, it does not provide the reliability needed by these types of application.

Our goal is to exploit the highly efficient best-effort delivery mechanism of IP multicast to simulate several scalable and efficient transport protocols for reliable multicast. In this project, we implement and compare different flavors of multicast transport protocols, including Reliable Multicast Protocol, Tree-based Multicast Transport Protocol, and Scalable Reliable Multicast, using ns-2 simulator. We examine the performance for each chosen protocols in campus network model. We use carefully chosen web and FTP background traffic to capture the characteristics of a real network environment. In addition, we use multimedia traffic traces to evaluate each chosen multicast transport protocol. Based on the simulation, we show the advantages and trade-offs for each of the multicast protocols.

1. INTRODUCTION

1.1 Multicasting Overview

There are three fundamental types of communication between computers, unicast, broadcast, and multicast. Unicast refers to a host computer talks directly to another computer. Broadcast allows one computer on the network simultaneously to talk to all devices contained in the same broadcasts domain. In a multicast scenario, a node makes a single call on a transport service that delivers a copy of that message to each destination nodes of the multicast transmission. The number of destination nodes can be zero, all, or any number in between. We can think of unicast as a form of multicast in which there is exactly one destination node. Similarly, we can think of broadcast as a

form of multicasting in which messages are sent to all reachable nodes. By implementing multicasting, we can reduce the load on the network. Suppose we have an application that needs periodically transmitting packets to large number of hosts within the company. Periodic unicast transmission of these packets would require many of the packets to traverse the same links. Broadcast transmission is not a sound solution for this case since every end station has to check the packet and it wastes bandwidth. Multicast transmission, on the other hand, would require only a single packet transmission by the source. As a result, considerably amount of traffic can be reduced.

Multicast networks are networks that have been extended to support multipoint message delivery services. The Internet protocol (IP) and asynchronous transfer mode (ATM) standards are important classes of multicast networks nowadays. Currently, network managers and planners are strongly interested in IP multicast because it resolves problems associated with delivering rich multimedia content across the Internet and within their private intranets. One of the main uses of IP multicasting will be for multimedia functions.

However, IP and ATM multicast networks provide best-effort multicast service for data applications. In other words, the network does not guarantee message delivery, delay, duplication, ordering, or throughput. Therefore, we need end-system protocols (multicast transport protocols) to provide reliable services. Receiver feedback and error control services are fundamental for many group-based applications. For instance, file and data distribution applications need complete delivery of messages from the sources to each member in the group in order. In this case, the transport protocol must detect and recover packets dropped by the network as well as report to the source when each receiver has successfully obtained the messages. On the other hand, some applications, such as multimedia applications, do not require any error control service. Instead, multimedia applications require the transport protocol to provide timely and accurate network drop and delay feedback so that the source can determine an appropriate transmission rate.

1.2 Project goal and structure

In this project, we study and evaluate how different reliable multicast transport protocols, including Scalable Reliable Multicast (SRM) [5], Multicast File Transfer Protocol (MFTP) [13], Tree-Based Multicast Transport Protocol (TMTP) [15], and

On-Tree Efficient Recovery using Subcasting (OTERS) [16].

We set up a campus network environment suggested in [8, 9] and background traffic implemented in [14] using ns-2 simulator [11]. We focus our ns-2 simulation on implementing a number of multicast transport protocols, and we measure the delay, throughput, packet loss rate, and protocol overhead in such campus network. In this project, we try to simulate a realistic network and compare the performance, strength, drawback, and tradeoff of each multicast protocol. We also plan to measure the scalability of each protocol by constructing a scenario that distributes video traffic trace over a wide-area network and study how the packets are dropped. However, due to resource limitations, we only study the performance in a relatively small campus network model, and we set the multicast group size less than 100 members. The simulation occasionally uses up our disk quota (200 MB) even for this campus network model.

The reminder of this paper is structured as follows. Section 2 covers the descriptions of each chosen multicast transport protocol and the overall simulation setups. It also shows the results of performance comparison. Section 3 gives our conclusion and discusses future work.

2. MAIN SECTIONS

In section, we briefly introduce the idea of reliable multicast protocol and each of the chosen multicast transport protocols, including SRM, MFTP, TMTP, and OTERS. Then, we provide description of our simulation model, including carefully chosen topology and background traffic model. Finally, we provide simulation experimental results and implementation details.

2.1 Reliable Multicast Protocols

Unlike unicast data delivery, multicast data delivery permits efficient use of the available bandwidth by having at most one copy of each packet sent over each link. Reliable multicast protocols provide a mechanism to ensure that every site receives that data. However, multicast also requires a more sophisticated method to provide reliable data transmission in comparison to reliable unicast. Suppose a sender-based approach (i.e. TCP) is applied to multicast distribution, a number of problems occur. First, each

of the received data packets triggers an acknowledgement from all the receivers, and the sender will suffer the ACK implosion effect. Also, it is a huge burden for the sender to keep track of the states of all members especially when the group size increases. Floyd et al. [5] have provided a more detail explanation on the design requirements for reliable multicast.

In our research, we found a number of proposed reliable multicast protocols, including Scalable Reliable Multicast (SRM) [5], Reliable Multicast Transport Protocol II (RMTP2) [7], MESH [8], Multicast File Transfer Protocol (MFTP) [13], Tree-based Multicast Transport Protocol (TMTP) [15], and On-Tree Efficient Recovery using Subcasting (OTERS) [16].

Unfortunately, not all reliable multicast protocols we have investigated have source code available for integrations into ns-2. Currently, there are three multicast transport protocols implemented in ns-2, Real Time Protocol (RTP), Scalable Reliable Multicast (SRM), and Multicast File Transfer Protocol (MFTP). Among these three protocols, RTP is not a reliable multicast protocol because it only performs best effort delivery and has no mechanism for error retransmission and recovery. As a result, we decided not to include RTP in our simulation even though we did study how to use RTP in ns-2. SRM are successfully installed and a complete set of documentation and example files are available in ns-2. MFTP is contributed by Hanle [13] and has already been ported to ns-2 in 1998. However, there is no document and example available for MFTP. The author has kindly provided us a demo file after we email him for help, but we realize that MFTP does not function due to version incompatibility problem. We finally solve the problem after we download entire ns-2 to our local directory, fix the bugs, and recompile the program. We would like to provide the MFTP demo file and debug experience to ns-2 in school if necessary. In addition, we also try to implement Tree-based Multicast Protocol (TMTP) and integrate On-Tree Efficient Recovery using Subcasting (OTERS) source code contributed by [16].

In this section, we discuss each of the chosen reliable multicast protocols in brief. More details on these protocols can be found in our references.

2.1.1 Scalable Reliable Multicast (SRM)

SRM [5] provides reliable many-to-many multicast packet delivery. Each member in the multicast group could be a sender and a receiver simultaneously. In this scheme, each receiver is individually responsible for detecting loss, and the closest neighbor

performs the retransmission in an ideal case. In general, lost packets are detected when there are gaps in the sequence number space. Though SRM does not offer timing guarantees, the design aims at quick packet loss recovery.

This loss recovery mechanism works well unless the last object of a sequence is dropped. SRM solves this problem by having all members periodically disseminate low-rate *session messages* that contain the highest sequence number received from every member so far. Session messages are also needed for estimating one-way distance between nodes; these host-to-host distances are needed when packet repair is necessary. When a receiver detects a loss, it multicasts a *request message* (a NACK with group address) to entire group. Every member who receives a request message is able to perform loss recovery by multicasting a *repair message* to the group. As with the original data, repair requests and retransmissions are always multicast to the whole group. Therefore, it needs some method to avoid implosion of request and repair messages. A host waits a random time before sending a request or repair message, and it stops sending the message if it sees a message from another receiver with the same information. Lucas [9] named this method unstructured approach because any (neighbor) host that has a copy of the requested data can perform loss recovery.

This mechanism is reliable as long as each data item is available from at least one member. Ideally, a lost packet triggers only a single request from a host just downstream of the point of failure and a single repair from a host just upstream of the point of failure. From our simulation, however, we found that SRM cause substantial network overhead. In addition to the periodical session messages generated by each group member, request and repair messages are also multicoated to the whole group. This means most participants receive unwanted request or repair messages whenever a loss-recovery takes place. It is possible that more than one neighbor send out repair messages simultaneously, in which case more overhead is generated.

2.1.2 Multicast File Transfer Protocol (MFTP)

MFTP [13] is a protocol for the reliable file distribution to a large number of receivers simultaneously over a multicast group. The protocol divides the file into packets and builds equally sized blocks of packets. The block size is the total number of bits of one maximum sized IP-packet. For instance, if the maximum picket size in the network is 1500 byte, a block contains roughly 12000 packets. Receivers report the receive status of data messages a block at a time.

Initially, the sender transmits (multicasts) all packets to every receiver in the first *pass*. Receivers write all data to a file and leave appropriate space whenever they detect a packet loss. Later, the sender makes another pass to retransmit data that was not received during the first pass, and the receivers fill the gaps when the appropriate repair packet arrives. Receivers report sender what packets need to be retransmitted by unicast a NACK-bitmap. For each block with at least one missing packet, receivers send back in unicast mode a NACK-bitmap, reflecting the status (received/missed) of each data packet within the block. In our example, the receivers keep gather packets until 12000 packets are received. Among the 12000 packets, if any of them is lost, the receiver replies (unicast mode) a NACK-bitmap indicating which packets are lost. Receivers randomly delay the NACK packet to avoid the implosion problem. In the end of a pass (after first 12000 packets are transmitted), the sender collects all NACK packets, determines loss packets, and retransmits those in a second pass. At the end of the second pass, again, receivers send back NACK bitmaps. This procedure continues until all receivers have completely received the file.
In this protocol, the sender is responsible for error/loss recovery for all members. Lucas [9] categorized this mechanism into centralized approach. In this approach, a central site has to keep track and process all the control messages of all the receivers in the group. Eventually, as the group size grows, the increasing protocol overhead will use up the central site's resources. MFTP has addressed this scalability problem because error recovery is aggregated at the end of each pass. However, timely error/loss recovery is impossible for MFTP, and it can only be used in non real-time applications.

2.1.3 Tree-based Multicast Transport Protocol (TMTP)

TMTP [15] exploits the efficient delivery mechanism of IP multicast for packet routing and delivery. However, for the purpose of scalable flow and error control, it dynamically organizes the participants into a hierarchical control tree. The following are essential TMTP features.

TMTP takes advantage of IP multicast for efficient packet routing and delivery. It uses an *expanding ring search* to dynamically organize the multicast group members into a hierarchical control tree as members join and leave a group. Expanding ring search means that when a node want to join the multicast control tree, expanding ring search is employed a potential connection point for the node into the control tree. A new node begins an expanding ring search by multicasting a SEARCH_FOR_PARENT request message with a small time-to-live value (TTL). The small TTL value restricts the scope

of search to nearby control nodes by limiting the propagation of the multicast message. If the node does not receive a response within some fixed timeout period, the manager resends the SEARCH_FOR_PARENT using a larger TTL value. This process repeats until the manager receives a response from one or more nodes in the control tree.

TMTP achieves scalable reliable multicast via the hierarchical control tree used for flow and error control. The control tree takes the flow and error control duties normally placed at the sender and distributes them across several nodes. This distribution of control also allows error recovery to proceed independently and concurrently in different portions of the network.

Error recovery is primarily driven by receivers who use a combination of restricted NACK with *NACK suppression*. In addition, the tree structure is exploited to restrict the scope of retransmissions to the region where packet loss occurs, thereby insulating the rest of the network from additional traffic. When a receiver notices a missing packet, the receiver generates a negative acknowledgment that is multicast to the parent and siblings using a restricted (small) TTL value. To avoid multiple receivers generating a NACK for the same packet, each receiver delays a random amount of time before transmitting its NACK. If the receiver hears a NACK for another sibling during the delay period, it suppresses its own NACK.

Currently, ns-2 does not provide this protocol. We try to implement this protocol in this project and we provide more implementation details below.

2.1.4 On-Tree Efficient Recovery using Subcasting (OTERS)

Another multicast transport protocol is OTERS (On-Tree Efficient Recovery using Subcasting)[14]. This tree-based reliable multicast protocol organizes receivers into a fusion tree that matches the multicast delivery tree of the source and uses this tree to fuse NAKs and subcast retransmissions.

OTERS consists of the *loss recovery protocol* (LRP) and the *fusion free formation protocol* (FTFP). Through FTFP, receivers can be organized into a tree rooted at the multicast source in which each subtree is a subtree of the multicast routing tree and for each subtree there is a designated receiver. FTFP builds an OTERS fusion tree by: 1) incrementally identifying subroots on the multicast routing tree using multicast route backtracing, and 2) selecting a *designated receiver* (DR). LRP efficiently repairs correlated losses using the fusion tree and subcast retransmission.

A key technique of OTERS is the use of the multicast route backtracing facility to dynamically trace the multicast delivery tree in order to identify subroots for building a fusion tree and models the multicast delivery tree and adapts to network and membership changes. Another key technique is the exchange of subcast FTFP packets among group members to distributively select a well positioned designated receiver for each subtree that handle NAK fusion and data retransmission for receivers in the subtree. A final key technique of OTERS is using the subcasting facility to retransmit packets for the DR to its subtree to fast and efficiently repair packet losses in the entire subtree.

Some other multicast protocols also use some form of hierarchical structure among group members. But it constructs subgroups on the closeness between group members. However, TTL or RTT –wise closeness does not necessarily reflect their relative position on the multicast routing tree.

In TMTP, requests are multicast with DA. Requests and repairs re-multicast to the host group, with the TTL equal to the subgroup's radius, to restrict the delivery scope. This incurs both DA delay and suffers from the coarse granularity of TTL-scoping in the current Internet. Besides, OTERS improves on SRM because it avoids duplicate avoidance (DA) delay which is a major part of the recovery latency in SRM. And it also will achieve higher overhead because repair packets are sent by subcasting not sent to the whole multicast group.

2.2 Simulation Scenario

In this project, we try to capture the characteristics of real networks into our simulation model. This simulation model contains two major components, the network model and the background traffic model The network model consists of the topology, the transmission infrastructure, routers, end-systems (servers and clients), and underlying protocols that define how messages are delivered from source to destinations end-systems. The traffic model creates background load within the network, which occasionally causes transmission delays, queuing delays, and packet drops in the network. We use ns-2 simulator to establish the network model, and then we implement the reliable multicast protocols mentioned above to the end-systems. We provide more detail information of the chosen model in the following.

2.2.1 Network model

In order to make our simulations realistic, we choose a fairly complex network model based on some real network, suggested in [8, 9]. Figure 1 shows the network topology. It is closely based on the SURAnet backbone, a contemporary WAN interconnecting research institutions in the Southeastern US. We use this model to represent a large-scale Intranet or ISP network using low-speed T1 infrastructure. Each node in the topology represents a campus local area network. We simplify the LAN by attaching five end-systems to each local network, including one web server and four clients. The simplified campus network is shown in Figure 2. The numbe r of end-systems attached to each campus network can be adjusted in our script (tcl) file.

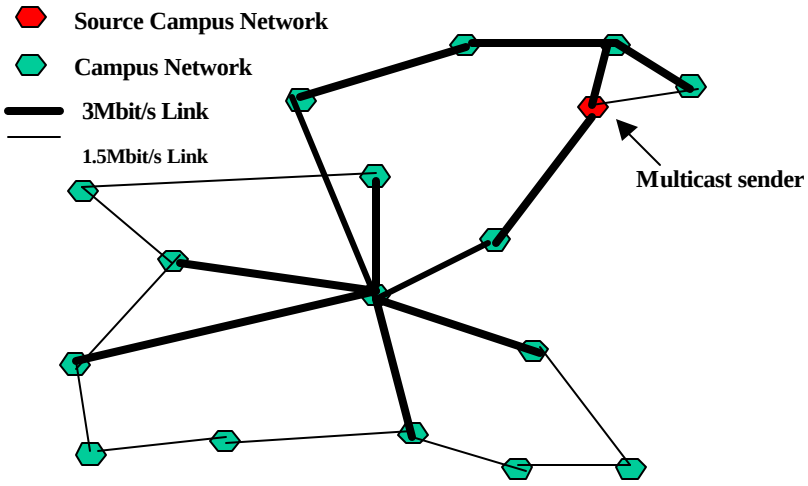

Figure 1: network model infrastructure and topology.

In a simplified local campus network, each end-system connects to the router by a 10Mbps link. The number of end-systems and servers can be determined in our ns-2 source code. The propagation delay is 5 milliseconds. The router employs a First In First Out (FIFO) buffer with a DropTail queue management policy. We use the default ns-2 queue size, which is capable of holding 50 packets. The multicast sender is located in a campus network marked in Figure 1. The multicast group members could be located in any end-system in any campus network.

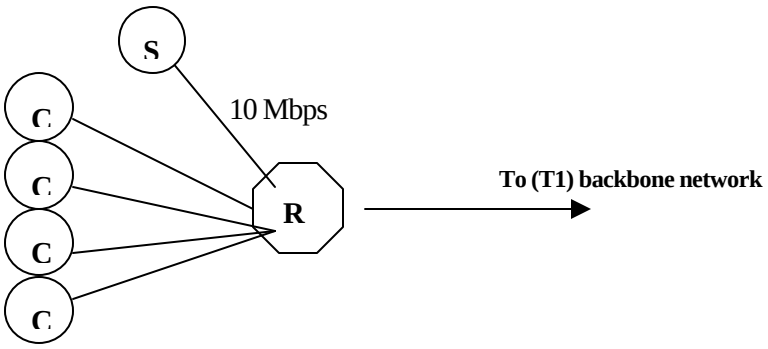

Figure 2: Simplified local campus network

2.2.2 Background traffic model

We define background traffic for all end-systems in the model. In order to simulate a realistic network environment, we have to create representative traffic flow among these end-systems. Study [14] has shown that most network traffic (including Ethernet, wide-are, and World Wide Web) exhibits self-similar behavior, and that TCP and UDP are the most dominant protocols in the Internet. These research results help us to build up the traffic model. Markovski [14] has given a well-defined background traffic model in his thesis. He also provides help and suggestions for our background traffic and other ns-2 related questions.

We implement three major types of background traffic in our model, Web traffic, FTP traffic, and Video traffic. The web traffic is generated according to the SURGE model. We also set the parameters for the user web sessions based on [14], and they are summarized in Table 1.

Web page parameter	Mean value
Inactive OFF time (Inter-page time)	10 sec
Number of page components (Number of objects per page)	3
Active OFF time (Inter-object time)	0.5 sec
Page component size (Object size)	12KB

Table 1: Parameters for Web sessions.

Interactive OFF time (think time) is the time between two consecutive web pages

requested by the user. *Number of page components* refers to the number of objects, including images and associated data, that are downloaded with the initially requested we page. *Active OFF time* is the time between the transfer of two objects. *Page component size* means the size of the downloaded web object. All the parameters refer to the Pareto distribution of the second kind. The Web related parameters set the generation of the web traffic in our simulation. Currently, ns-2 provides HTTP 1.0 (Hypertext Transfer Protocol). There are two types of source agents in Web traffic, www client and www server. At configuration time, a client connects to a server. A client makes a short request to the server, which then generates a random number of connection responses, each with random length. Once the server finishes sending data to the client, the client waits for a random time before makes another request from server. One-way TCP are used for connection.

Markovski [14] choose the FTP parameters based on the statistical analysis of wide-area traffic and some Internet backbone measurements. FTP enables file distribution between two end-systems. In our simulation, we create separate TCP connections for control and data information. Generation of ftp sessions is based on an exponential distribution. The mean value of the ftp sessions is equal to the total number of sessions assigned in the simulation divided by the simulation time. Some UDP connections are also established in our background traffic model in order to simulate Video traffic. A video session, initiated between two random end-systems, has random starting points, and it lasts for a random amount of time. The minimum duration of a video session is 60 seconds (for short video clips) and the maximum duration is restricted by the simulation time. Since more information regarding to FTP and Video traffic is presented nicely in [14], we will not go into how FTP and video traffic parameters are determined in detail.

In our simulation, various numbers of web, FTP, and video sessions are assigned in the tcl files to reflect different background traffic load in our scenario. We set the TCP packet size to 1500 bytes because it is the default MTU (Message Transmission Unit) size and it also corresponds with the size of the Ethernet frame. An example trace file provided by ns-2 is used to determine the packet size of video traffic. We use constant bit rate (CBR) traffic with 512 bytes packet size and 5ms interval for our multicast traffic. 2000 packets (about 1MB file) are sent during the simulation, and we set the simulation to be 200 seconds. We provide the topology and multicast traffic summary in Table 2.

Topology	-102 nodes and 107 links - 17 routers - 5 end-systems for each router
Multicast traffic	-10 members in multicast group (1 sender and 9 receivers) -512 byte packet size -2000 packets (~ 1MB) -Simulation time: 200 sec

Table 2: Topology and multicast traffic details

2.3 Experimental Results

We evaluate the performance of SRM and MFTP based on the number of packet losses, protocol overhead, and transmission time of fixed size data (throughput).

2.3.1 Packet Loss

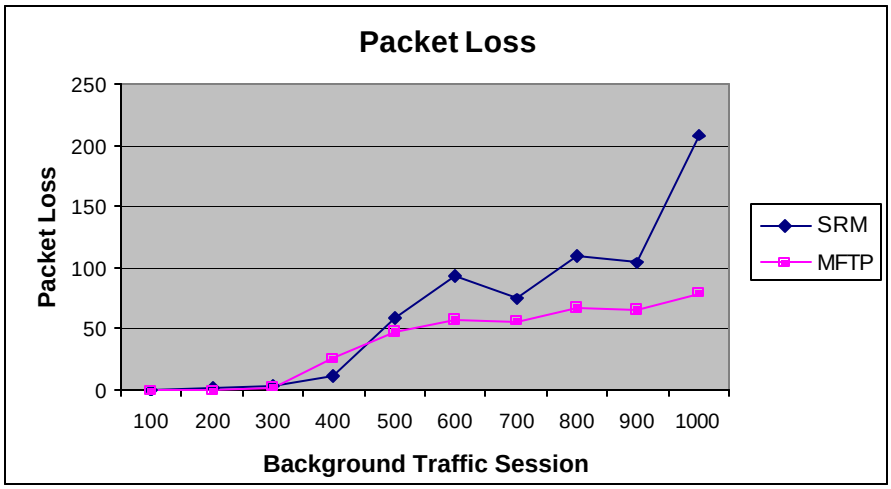

Figure 3: Number of packet losses

The number of packet losses is similar for both protocols when the background traffic load is moderate. However, as the network load increases, SRM causes more packet loss than MFTP. This is due to the higher protocol overhead in SRM. When the network background load is high, SRM tends to generate a substantial number of repair packets that flood the network (Figure 4). These repair messages further occupy the available bandwidth, which cause more packets to be dropped. The maximum number web sessions are set to 1000 in our simulation, because it has already caused a 10% packet loss rate for SRM. We believe that SRM will cause even more packet losses compared to MFTP if we further increase the background traffic load.

2.3.2 Protocol overhead

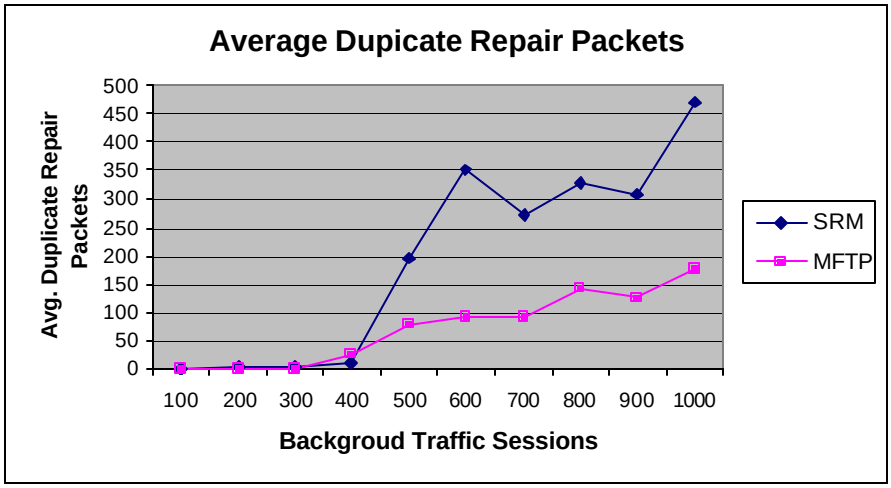

Figure 4: Average Duplicate Repair Packets for each Receiver

SRM generates a considerable amount of protocol overhead when the background traffic load is heavy. In SRM, there are three types of messages, session, request, and repair messages. Each member periodically multicast session messages to all members, and every member tends to receive duplicate request and repair messages. Repair message has the largest packet size among these three types of messages (it is the retransmission packet) and therefore dominate the protocol overhead. In Figure 4, we show that the each member receives much more unless messages when SRM is implemented. This result is consistent with our observations in Figure 3. More packet loss immediately triggers more repair messages, which may further cause more packet drop in the buffer. When the background traffic load is heavy (i.e. 1000 sessions), every member picks up, in average, about 450 unwanted repair messages in addition to the 2000 data packets.

MFTP, on the other hand, only creates moderate protocol overhead. This outcome is reasonable because MFTP sender collect and aggregate the NAKs from the receivers and perform retransmission in the end. In this matter, most receiver obtain "useful" retransmission packets if the packet losses is roughly evenly distributed over the receivers. However, in the worst case, MFTP could also generate a great number of duplicate retransmission messages. Consider the case that 100 packets are lost in only one receivers, while all other multicast members obtain compete data. In the end of transmission, the sender has to multicast the 100 lost packets over the entire group. All members will receive the 100 retransmission packets, but only one receiver needs those packets. Though, this is a extreme case, and we never see this in our simulation.

2.3.3 Transmission Time

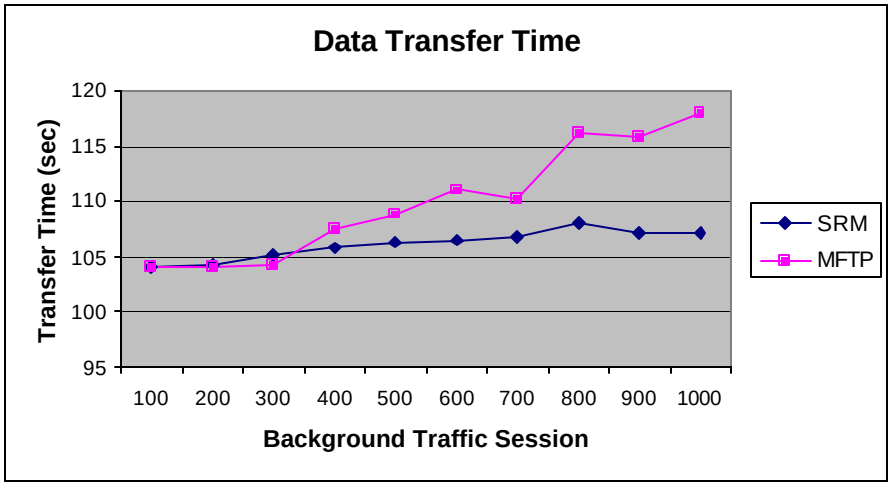

Figure 5: Data Transmission Time

Although SRM brings substantial amount of protocol overhead in a congested network environment, it provides timely loss recovery. Figure 5 shows the time required to transmit 1 MB data. As the background traffic load grows, SRM only needs a small amount of additional transmission time while MFTP requires more extra time to complete the transmission. In our simulation scenario, the transmission time for MFTP boost more than 10% (SRM increased 2%-3%) when the background traffic session is set to 1000. This phenomenon is due to the diverse loss recovery design behind two protocols. As we mentioned above, SRM provide almost immediate loss recovery, therefore, it is not surprising that it has better throughput. MFTP performs retransmission after the sender finishing sending a large number of packets. Therefore, MFTP certainly request more extra to complete the transmission, unless there were no packet loss. Moreover, it is also possible for retransmission packets to be dropped in which case further transmission time is necessary.

We also found that SRM has repair delay no more than 0.5 second in our simulation, even when the background traffic load is heavy. MFTP, in contrast, could suffer from 10 second repair delay. This advantage makes SRM a good candidate for real-time applications in which it is feasible to buffer incoming data for several seconds. For instance, radio over the Internet is a possible application for SRM. In this case, by buffering incoming packets, the receivers can have enough time to recover from lost packets before the packet's contents are used to play the sound.

2.4 Tree-based Multicast Transport Protocol (TMTP) implementation

Because ns-2 does not provide any tree-based or sub-group multicast protocol, we strive to implement this protocol. We provide implementation details in the following.

2.4.1 Data Structures of the TMTP agent

The implementation of TMTP involves two data structures: the *inter data structure* in TMTP agent and the *intra data structure* in entire ns system.

Inter data structure :

There are three types of messages in TMTP: *SessInfo* class, *DMInfo* class and *DataInfo* class.

SessInfo class store and process the attribute values of sessions that are essentially used to manage the control tree.

- *type* is a basic value in SessInfo that is used to distinguish which type the session is: SEARCH_FOR_DM, ANNOUNCE_DM, JOIN_DM, ACCEPT_JOIN, ANNOUNCE_SOURCE or REFUSE_JOIN.
- *ring* contains the time value from *expand ring search*, which controls the scope of the session traffic multicast.
- *level* value can be –1, 0 or 1, which represents the level of the sender in the tree: it is not a node in the tree, a leaf node or an internal node.
- *ttl* contains the value of time-to-live.
- *delay* store the delay time.

DMInfo class store and process a set of values of a domain manager (DM), typically it is stored in a node.

- *DM* is the address of the DM.
- *ring* saves the *ttl* value after session stage.
- *delaytome* is the attribute, which contains the distance to its parent DM in the tree.

DataInfo class includes the information of data transport.

Based on the above message classes, we build the TMTPAgent class. It basically includes the following component:

- *SessInfo* object, *DMInfo* object (here we need to instance two objects for a node because every node may act as two roles in the tree: child and parent.) and *DataInfo* object.
- Control messages.
- Functions perform the tree-based multicast tasks. It includes:

  void recv_data(int msgid, double delay);
  // to receive the data
  void recv_sess(SessInfo *dr, nsaddr_t sender, int ring, double delay);
  // to receive a session packet
  void recv_repr(int msgid, double delay);
  // to receive a repair packet
  void recv_rqst(int requestor, int msgid, double delay);
  // to receive a request packet
  void send_sess(int type, int ttl, nsaddr_t receiver);
  // to send a session packet
  void send_rqst(int msgid, int uni);
  // to send request packet
  void send_repr(nsaddr_t requestor, int msgid, int uni);
  // to send a repair packet.
  void becomeImDR(int ring, double delay) ;
  // to have children
  void updateMyDR(DRInfo *dr, double delay);
  // to update parent DM
  void badDR(int msgid);
  // to handle the bad DR

Intra data structure

ns-2 simulator supports a class hierarchy in C++(called the compiled hierarchy), and a similar class hierarchy within the Otcl interpreter(called the interpreted hierarchy). The two hierarchies are related closely related to each other.

- In C++ compiled hierarchy, we inherited a TMTPAgent class from a based class Agent in ns:

```
static class TMTPAgentClass : public TclClass {
public:
  TMTPAgentClass() : TclClass("Agent/TMTP") {}
```

```
  TclObject* create(int, const char*const*) {
    return (new TMTPAgent());
  }
} class_tmtp_agent;
```

Also we inherited a TMTP packet header class from a based class:

```
PacketHeaderClass
static class TMTPHeaderClass : public PacketHeaderClass {
public: TMTPHeaderClass():
    PacketHeaderClass("PacketHeader/TMTP", sizeof(hdr_tmtp)) {}
} class_tmtp_hdr;
```

- There is a hierarchy of Tcl classes corresponding to the compiled classes. The root of the hierarchy is the class TclObject. We build a TMTPAgent interpreted class in tmtp.tcl file.
  The following example illustrates the Agent/TMTP class constructor:

```
Agent/TMTP instproc init {} {
      $self next
      $self instvar ns_ nid_ node_ unwanted_
      set ns_ [Simulator instance]  …
```

The class Agent/TMTP is a subclass of Agent, is a subclass of TclObject.

- *ns* needs to establish bi-directional bindings such that both the interpreted member variable and the compiled member variable access the same data, and changing the value of either variable changes the value of the corresponding paired variable to the same value.
  The following example shows the bindings in TMTPAgent constructor:

```
      TMTPAgent::TMTPAgent()
      {   …
        bind("off_tmtp_", &off_tmtp_);
        bind("off_cmn_", &off_cmn_);
        bind("packetSize_", &packetSize_);
        bind("src_", &src_);
          …
```

Note that all the binding functions above take two arguments, the name of an Otcl variable, and the address of the corresponding compiled member variable that is linked.

2.4.2 Algorithm of the TMTP agent

Overview

Packet transmission in TMTP proceeds as follows: when a sender wishes to send data, TMTP uses IP multicast to transmit packets to the entire group. The control tree ensures reliable delivery to each member. Each node of the control tree is only responsible for handling the errors that arise in its immediate children. When a child detects a missing packet, the child multicasts a NACK in combination with *nack suppression*. On receipt of the NACK, its parent in the control tree multicasts the missing packets. To limit the scope of the multicast NACK and the ensuing multicast retransmission, TMTP uses the Time-to-Live(TTL) field to restrict the transmission radius of the message. As a result, error recover is completely localized.

Control tree management

Over the lifetime of the multicast group, the control tree grows and shrinks dynamically in response to additions and deletions to and from the multicast group membership.

There are only two operations associated with control tree management: *JoinTree* and *LeaveTree*. The *JoinTree* operation employs an expanding ring search to locate potential connection points into the control tree. Figure 3 outlines the procession of *JoinTree*.

```
While (NotDone){
  Multicast a SEARCH-PARENT msg
  Collect respondses
  If (no responses)
     Increment TTL   // try again
  Else
  select closet respondent as parent
  send JOIN-REQ to parent
  wait for JOIN-CONFIRM reply
  if (JOIN-CONFIRM received)
     NotDone = False
```

```
     Else  //try again
}
```

Figure 4: *JoinTree* procedure

For *LeaveTree* operation, the operation for leaf DM in the tree is straightforward. However, the algorithm for internal DM is complicated by the fact that internal DMs are a crucial link in the control tree, continuously providing reliable service to their children. The algorithm for *LeaveTree* is shown in figure 5.
We implemented the *LeaveTree* operation in this way: firstly, a departing DM has to fulfill all obligations to its children, it then instructs its children to find a new parent. The children then begin the process of joining the tree all over again. After the children finish their *JoinTree* operation successfully, the DM can leave from the control tree.

```
If ( I am a leaf manager)
     send LEAVE-TREE to parent
     receive LEAVE-CONFIRM
     terminate
Else   // I am an internal manager
     Fullfill all pending obligations
     send FIND-NEW-PARENT to children
      receive FIND-NEW-PARENT from all children
      send LEAVE-TREE to parent
```

Figure 5: *LeaveTree* procedure

Error recovery

TMTP basically employs error recovery from both sender and receiver initiated approaches. A DM relies on both periodic positive ACKs and Negative ACKs (NACK) from its children. In our implementation, we only depended on the NACK message for error recovery. Here, we uses restricted NACKs with *nack suppression* to respond quickly to the packet losses.

Implementation difficulties

It may result in each receiver is a DM. Any multicast receiver in TMTP protocol acts as either a Domain manager in the control tree or a Group member in a local domain. However this protocol did not provide any schema to identify multicast participants

between these two roles. Because of the heterogeneity of real work network such as our simulation topology, this protocol would probably be confused how to organize the receivers.

TMTP cannot handle DM failure. To solve this problem, we provided a new function BadDM() that allows the failed DM's children to update their parent.

In TMTP protocol, a TMTP traffic source (sender) requires both periodic positive acknowledgements from a receiver and NACK when the receiver notices an error. However, the TMTP protocol did not provide a measure about how to select between ACK and NACK.
In our implementation, we only use NACK with *nack suppression* to invoke the error recovery procedure.

Currently, we have successfully built our TMTP agent into the c++ compiled hierarchy of ns. Unfortunately, we are unable to attach any traffic to the agent. We believe that this problem may result from ns-2's very complex configuration. During our implementation, we also try to integrate anther tree-based protocol, OTERS, introduced by Li [16]. The author has already implement this protocol in ns-2, and source code can be downloaded from [11]. However, we also cannot integrate his source code into our system. We believe this problem also due to complicated configuration and incompatibility among different ns-2 versions. We would like to mention again that Velibor not only provided us help in our background traffic model, he also gave us suggestions when we build up ns-2 system in our local directory.

3. Discussion and Conclusions

In this project, we have study numerous reliable transport protocols. We have implemented a realistic network simulation environment, including the network model and background traffic model, for multicast communication using the network simulator ns-2, and we measure the performance and tradeoffs for different reliable multicast protocols. We believe that it is a meaningful task because it is essential to provide appropriate methods for evaluating emerging multicast protocols. Network simulation is the most promising and cost-efficient approach for assessing protocols if a realistic network model that reflects real network behavior is used.

We use this network model to evaluate SRM and MFTP. These protocols are available in ns-2, but we have to fix some bugs in MFTP before we can successfully run it. We show that MFTP causes less packet loss and protocol overhead than SRM. However, SRM is able to provide time loss recovery and, as a result, reduces the data transmission time. We also tried to compare more protocols by implementing them in ns-2. Unfortunately, we were unsuccessful to do so because adding a new module in ns-2 system is too complicated. However, we was close to achieving the goal because we had the C++ code was compiled successfully and the ns-2 system was able to recognize the new Agent we added in. It would be very interesting if we could simulate sub-group and tree-based multicast protocols.

Alternative approaches for this project would be investigating the packet loss pattern in each multicast receiver, studying the protocol performance in different topology, or examining how the protocols perform when link failure takes place (network dynamics). We believe these are also attractive topics for network researchers.

We did not investigate scalability issues due to limited resource. We might be able to find more characteristics for each protocol if we scale to a much larger multicast group. For example, NACK implosion might be a potential problem for MFTP when the multicast group size is huge. SRM might perform better in a large sparse group since the nearest neighbor would probably be the only one (and is the best candidate) to perform loss recovery. In addition, we really like to implement and evaluate more reliable multicast protocols. There is some contributed source code for sub-group and tree-based multicast protocols for previous version ns-2. We hope these protocols will soon be integrated to ns-2 in the near future.

4. Reference

[1] M. Banikazemi. “IP Multicasting: Concepts, Algorithms, and Protocols, IGMP, RPM, CBT, DVMRP, MOSPF, PIM, MBONE,”; http://charly.kjist.ac.kr/~dwlee/homepage/ipmulitcast.htm

[2] B. Williamson. “Developing IP Multicasting Networks Volume 1 - Distance Vector Multicast Routing Protocol and Multicast Open Shortest Path First,” pp. 106-127 pp.194-211, Cisco Press, 2000.

[3] E. C. Douglas, "Internetworking with TCP/IP Volume 1 - Internet Multicasting (IGMP)," pp. 289-302, Prentice-Hall, Inc., 1995.

[4] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. G. Liu, and L. Wei, "An Architecture for Wide-Area Multicast Routing," *ACM SIGCOMM '94*, vol. 24 no. 4, pp. 126-135.

[5] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne. and L. Zhang. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *ACM SIGCOMM' 95*, Aug. 1995, pp. 342-356.

[6] S. Armstrong, A. Freier, K. Marzullo, "RFC 1301: Multicast Transport Protocol," Feb. 1992, http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1301.html

[7] B. Whetten, G. Taskale. "An Overview of Reliable Multicast Transport Protocol II," *IEEE Network*, Jan/Feb 2000, pp. 37-47; www.komunikasi.org/pdf/multicast/reliable-multicast-transport-protocol-ii.pdf

[8] M. T. Lucas, B. J. Dempsey, and A. C. Weaver. "MESH: Distributed Error Recovery for Multimedia Streams in Wide-Area Multicast Networks," In *Proceedings of IEEE International Conference on Communication (ICC '97)*, pp. 1127-1132, June 1997.

[9] M. T. Lucas. "Efficient Data Distribution in Large-Scale Multicast Networks," Ph.D. Dissertation, Department of Computer Science, University of Virginia, May 1998.

[10] M. Goncalves and K. Niles, "IP Multicasting, Concepts and Applications," New York: McGraw-Hill, 1998, pp. 91-116, pp. 273-287, pp. 305-324.

[11] ns-2 network simulator: http://www.isi.edu/nsnam/ns

[12] Star Wars trace in ns format: http://www.research.att.com/~breslau/vint/trace.html

[13] C. Hanle and M. Hofmann, "Performance Comparison of Reliable Multicast Protocols using the Network Simulator ns-2," *Proceedings of IEEE Conference on Local Computer Networks (LCN)*, Boston, MA, USA, October 11-14, 1998.

[14] V. Markovski, "Simulation and Analysis of Loss in IP Networks – Simulation

scenarios," M. Sci. Thesis, Department of Engineering Science, Simon Fraser University, Oct. 2000, pp. 24-30.

[15] R. Yavatkar, J. Griffioen, and M. Suda. "A Reliable Dissemination Protocol for Interactive Collaborative Application," In *Proceedings of the ACM Multimedia '95 Conference*, Nov. 1995.

[16] D. Li and D. R. Cheriton, "OTERS (On-Tree Efficient Recovery using Subcasting): A Reliable Multicast Protocol," *Proc. of IEEE ICNP '98*, Oct. 1998. 14, pp. 237-245

Appendix:

1. Code Listing