Enhancing an ATM Simulator

Joel Clawson
Division of Science and Mathematics
University of Minnesota, Morris
clawsonj@mrs.umn.edu

Rebecca Christen
Division of Science and Mathematics
University of Minnesota, Morris
ruegemr@mrs.umn.edu

A. A. Lopez
Division of Science and Mathematics
University of Minnesota, Morris
alopez@mrs.umn.edu

Abstract

We are studying congestion control algorithms on an ATM network. Our studies are based on computer simulations. We are fortunate that the National Institute of Standards and Technology (NIST) has developed (and made freely available) a simulator that has facilitated our studies of ATM networks.

In our studies we need to track cells as individual entities. We need to know where this cell has originated, what type of traffic does it represent, how long has this cell been waiting on a queue, the cell's priority, etc. The simulator as distributed does not permit us to discriminate between the cells based on a priority. We found it necessary to add an additional field to each cell for priorities and replace FIFO queues with priority queues. The switches along a network can select higher priority cells for processing before lower priority cells, using the priority queues and cell priorities.

Introduction

The authors of this paper are interested in studying Asynchronous Transfer Mode (ATM) computer networks. There are at least two aspects of these networks that one may be interested in studying: the physical devices that make up such a network and the programming and routing of packets over such a network. Our interest is in the routing of packets over such a network. The study of the physical devices, their design and operation is a very costly endeavor.

The study of the routing of packets, congestion control, throughput rates, etc. can be conducted over a physical network or it can be simulated using a suitable software network simulator. Most commercial network simulators [1,2] cost thousands of dollars and require special contract programming in order to simulate any new congestion control algorithm that we may wish to study. Fortunately for us, there are public domain network simulators [3,4] that can be used to simulate ATM networks. The developers of many of these simulators make available the source code for the simulator. Researchers like us can then modify the simulator to investigate various congestion control algorithms, queuing techniques, rate of packet loss, average packet delay and throughput.

For our investigation, we chose to use the NIST ATM/HFC Network Simulator version 4.1. This simulator is produced and distributed by the National Institute of Standards and Technology, which is part of the U.S. Department of Commerce. Source code for this simulator is available from http://w3.antd.nist.gov/Hsntg/products/atm-sim-doc.html. We have been using this simulator or its previous versions for about 5 years. In each case, we have downloaded the C source code, compiled it under Linux and used it with X Windows.

For this investigation, we decided that the simulator would need to be modified to support a specific priority for each ATM cell that the simulator handles. ATM cells consist of a 48-byte payload and a 5-byte header. This paper describes how we modified the NIST simulator. The goal of the research is to be able to investigate different congestion control algorithms for a variety of traffic regimes.

Statement of the Problem

The ATM protocol does not support the existence of a priority field in each cell. We wish to introduce such a field, so we can improve traffic congestion and throughput.

The ATM protocol has defined 4 different types of traffic, Available Bit Rate (ABR), Constant Bit Rate (CBR), Unspecified Bit Rate (UBR) and Variable Bit Rate (VBR). Typically, CBR traffic has the highest priority in an ATM switch and ABR traffic has the lowest priority. We are interested in seeing if we can improve the responsiveness of a network by adding a priority field to cells within a given type of traffic. For example, we may have two applications sharing a VBR stream in which we wish to have the first

application assign a higher priority to its cells than the second application does. In order for us to be able to simulate this situation, we would need to add a 'priority' field to each cell sent by any of the applications.

The designers of the ATM protocol did not make provisions for such a priority value in the cell header. The only differentiation that occurs in cells is through the Cell Loss Priority (CLP) bit in the header. This bit is a way for the application to decide that one cell (CLP=1) is more important than another (CLP=0). But all CLP=1 cells, for the same type of traffic, will be treated the same. The designers did provide for the different types of traffic (for example, ABR and UBR) to be treated differently.

Proposed Solution

Our solution to the problem is to increase the sophistication of the ATM switch's scheduling algorithm by giving each cell a priority. This involves changing how the switch handles cells by adding a priority field, which is set by the application. To upgrade the simulator so it supports 'priorities' within a particular class of service, we needed to implement priority queues instead of First-In-First-Out (FIFO) queues. The purpose of the priority queues is to hold the cells that are received by the switch in a prioritized way. The priority queues are arranged in an array of queues which are added to according to the priority field.

Each ATM switch contains a FIFO queue for each kind of traffic, with CBR traffic sharing a queue with VBR traffic. The switch, once it receives a cell, determines the type of traffic it is by accessing a field in the cell header. It then inserts the cell into the appropriate queue for that cell. The cells remain in the priority queues until it is time for another cell to be moved from the switch towards the receiving application. At this point, the switch can find the cell with the highest priority that has been in the queue for the longest period of time. The switch removes the cell from the highest priority queue and sends it along the Link (Figure 1) towards the next component for the specified cell. Once the cell reaches the receiving application, it handles the cell just like it does in the original version of the simulator.

This setup allows us to investigate various strategies for minimizing congestion and the number of lost cells. For instance, the switch can be programmed to support a "modified" priority queue strategy. The switch could decide that it is processing cells from the higher priority queues a lot faster that it is expected to process them. In such cases, the switch can decide to dedicate some of its processing power to dealing with cells that have been waiting for a long time in a lower priority queue.

The NIST simulator supports many different components and applications. Our work, to date, has focused on optimizing the rate-based ATM switches (that allow us to select a congestion control algorithm, such as ERICA or EPRCA), regular Broadband Terminal Equipment (BTE), and TCP/IP connecting applications (Figure 1).

Current Progress

We began by developing a priority queue data structure for the switches in the simulator to replace the FIFO queues. This priority queue will allow us to increase cell selectivity. The development of the priority queue parallels the FIFO queues and it uses equivalent fields and function names as the FIFO queue component that was supplied with the original simulator. The priority queue is an array of queues. The length of the array is equal to the number of priorities (e.g. 0-7) that a cell is allowed to have. There is also a maximum length for the priority queue, which is set by the user at the instantiation of the ATM switch structure. It doesn't allow any more cells to be added to the queue until cells are removed, thus causing cell drop. The priority queue was thoroughly tested.

Our next problem was to define a priority field. We were fortunate to find that the simulator had a predefined priority field defined in the cell header. This was introduced for the Hybrid Fiber Coax (HFC) Broadband Terminal Equipment (BTE) component. We determined that the priority field in the cell header was only used by the HFC BTE and not by any other component in the simulator. This allowed us to add the functionality of priorities to the simulator without changing the definition of the cell and cell header.

Since the priority field was already defined in the simulator, we searched through the source code for the place where the field should be set in the simulator. We did this by finding where the cells were created before they were passed on to the next component. This was found to be in the TCP/IP application and we added commands for the application to set the priority field of the cells. For the application to know what priority to give each cell, we had to also define a priority field for the application. We added a new field for the priority that gets instantiated at the creation of the component and is subject to user input like other member variables of the application. The application sets the cell's priority field to the priority of the application at the time the cell is created. This allows any component of the simulator to access a cell's priority without knowing anything about all the other components being used in the simulation. For example, a switch accesses a cell's priority through its header and not through accessing the application where the cell was created.

The switch is where we want to handle cell priority. We replaced the ABR queue, UBR queue, VBR queue, and the queue that buffers cell traffic on its way to the trafficking queues with priority queues, and we replaced all the queue functions with the priority queue functions. After the cell enters the switch, the switch determines the cell's priority and handles the cell just as it would in the original simulator, except that it goes into the priority queue instead of a regular queue.

Currently, we have the priority queues fully implemented in the cell trafficking queues and the buffering queue. Traffic flows through the simulation in a similar manner as it does in the original simulator.

Conclusions

Modifying the NIST simulator proved to be more difficult than we first anticipated. The internal documentation of the source code is minimal. The NIST manual is helpful, but it is aimed at those that want to use the simulator in its original distributed form, not at those that want to modify the source code. Finally, there is the difficulty of compiling C source code with the appropriate compiler options and libraries.

On the other hand, modifying the NIST simulator has cost us a fraction of what a commercial simulator would cost us (even with deep university discounts). In addition, it has proven to be a very valuable learning experience for the two students that participated in the project over the past two years. There is no better way to learn about a piece of software than to try to modify it.

The public domain NIST simulator is a very useful piece of software and it is priced right for undergraduate students undertaking network studies. It has a nice, attractive, X-Windows interface with pull down menus for each component and constantly updates information as the simulation progresses.

Future Work

Now that the simulator has been successfully modified, we will return to the task of working on congestion control algorithms that will minimize congestion and improve the throughput of cells through the network.

Bibliography

1. CACI, COMNET-III, http://www.caciasl.com/university/univ_list.html

2. Mil3 Inc, http://www.mil3.com/services/university/univ_use_opnet_list.html

3. SimATM, http://www.mc21.fee.unicamp.br/SimATM/

4. STCP, http://lrcwww.epfl.ch/~manthorp/stcp/stcp.html