TRAFFIC SHAPER FOR VBR VIDEO SOURCE

by Anders Johansson, Dragos Ilie and Ola Nilsson

Abstract

The evolution of data communications towards ATM networks has forced the development of new techniques aimed to improve the flows of data (i.e. ATM cells) throughout the network. Traffic policing, traffic shaping and statistical multiplexing are some techniques worth mentioning.

The requirement to keep these techniques up to date and nevertheless stable and highly effective is one of the main reasons for repeatedly simulating the events occurring inside a network or simulating the network as a whole. Simulation is also a way to test new ideas without causing damage upon the physical network, something well appreciated by the network users.

Why shaping the traffic

One part of the ATM specification is the "Traffic Management Specification" found on the web at http://www.atmforum.org. The document specifies how two ATM networks negotiate the parameters for one or more connections and the methods used to ensure and/or enforce the terms of the negotiation (also called traffic contract). Some of the parameters used for the traffic contract include Peak Cell Rate (PCR), Sustainable Cell Rate (SCR), Maximum Burst Size (MBS), Minimum Cell Rate (MCR) and Cell Delay Variation Tolerance (CDVT).

After the networks set up the traffic contract they start monitoring the traffic flow to check if the terms negotiated are being respected. Sources found to disrespect the contract are punished by the means of cell deletion, often without any notification. These actions are part of the traffic policing. The Generic Cell Rate Algorithm (GCRA) is used to define conformance with respect to the traffic contract.

A good way to avoid punishment for non-conforming cells is to shape the traffic before sending it to the other side. Results of traffic shaping can be burst length limitation, peak cell rate reduction, cell scheduling policy and cell delay variation limitation. These results can occur by themselves or in some combination depending mostly on the traffic shaping algorithm implementation.

The assignment

Part of the course "Modern Techniques of Networking" was to simulate some traffic and to analyze different aspects attached to it. In our case we had to simulate a traffic shaper for a VBR (Variable Bit Rate) video source.

The source is sending data as frames, at a rate of **f** frames per second where **f** usually is 25 when a flicker-free image is desired. The frame size is variable, an effect of the MPEG-coding. When a frame enters the shaper it is divided into ATM cells that are placed into an internal buffer. The buffer size is **B** ATM cells and when the buffer is full the cells that that cannot be stored are lost. This brings up the parameter of Cell Loss Rate (CLR).

The shaper outputs cells from the buffer at a peak rate, PCR, of **r** cells per second. In other words a cell is outputted each 1/**r** seconds and the rest of the cells in the buffer age by 1/**r**. The age of each cell is a measure for the delay between the arrival and departure of that cell. This second parameter is the Cell Delay Rate (CDR).

Several video traces were used to simulate the shaper. The goal was to find some relation between **B** and **r** and the traffic characteristics in order to maximize the QoS with respect to CLR and CDR.

Matlab was chosen to do the simulation, mostly because of its ease of use. We divided the source code into two parts: the shaper driver (kernel) and the user interface.

Software: requirements, installation and usage

In order to use the simulation you need access to the MATLAB 5 software installed on a computer. Note that the simulation won't run out-the-box for previous versions of MATLAB 5. You'll probably need to reprogram the user-interface and most of the for-loops as they rely on new features.

Unpack the two zip files, *sim3d.zip* and *simtime.zip*, into one directory each. We suggest sim3d and simtime respectively.

Start MATLAB in the usual way and set the path to **one** the two created directories as following:

>path('*your_path*\sim3d', path)
or
>path('*your_path*\simtime, path)

Change *your_path* with the path to the directory and sim3d and simtime with the names you have chosen for your directories.

Please note that you can't have both programs into the path at the same time. This is a limitation we are well aware of and will probably be corrected in future releases.

The simtime program let you try one set of values at one time. The sim3d will run all the values in an interval with the step length of your choice and show the result as a cool 3d-graph.

You can start the simtime or respectively sim3d program by entering *run_time* and respectively *run_3d* in the MATLAB command window. Most options and buttons should be self-explanatory.

You will need also at least one MPEG time stamp file. You should find some at the following Internet addresses:

ftp://ftp-info3.informatik.uni-wuerzburg.de/pub/MPEG/
ftp://tenet.berkeley.edu/pub/dbind/traces/

The *run_time* program displays the following graphs:

- Buffer occupancy in percent versus time
- Cell loss in percent per frame versus time
- Cell delay in seconds per frame versus time
- Cell and bit rate for outgoing data and bit rate for incoming data
- Frame input sizes versus time. This is just a plot of the data trace fed to the simulation

The *run_3d* program displays the following 3-d graphs:

- Mean error in percent per frame versus buffer size and peak output rate
- Mean frame delay versus buffer size and peak output rate

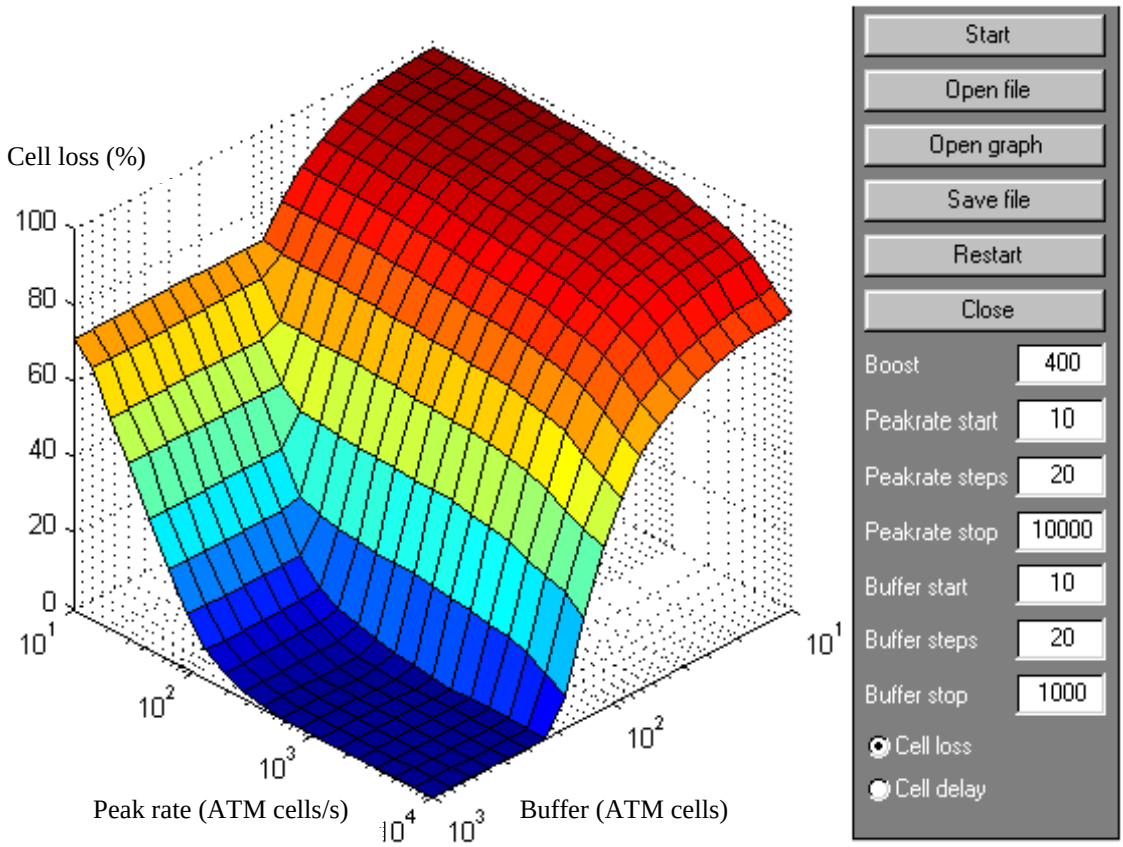
Conclusions

The time plots show a bursty structure for the buffer occupancy, cell loss and cell delay variables. The three of them follow each other in time as one might expect. The input and output rate show very weak correlation, if any at all, mainly due to cell loss and cell delay inside the buffer.

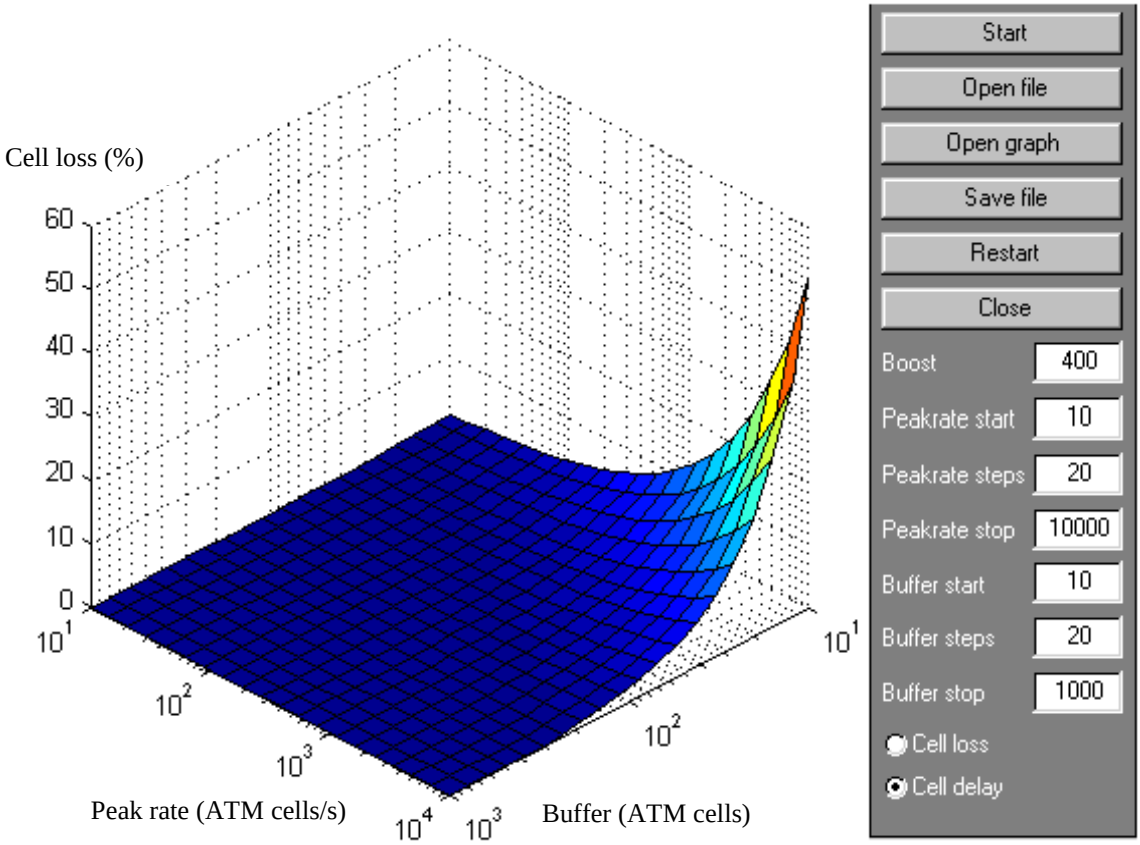
The three-dimensional plots show an increase in cell error as the buffer size and output peak rate decreases. However, if the buffer size or peak rate is increased, or both of them at the same time, one will

generally experience increasing cell delay. Best overall performance is achieved if the buffer size is around 300 ATM cells big at the same time as the peak rate is 600. Higher values won't necessarily result in proportionally high performance.

Together with this document we supply also a number of graphics created using the file *atp_.ipb*.



Cell loss graph
The peak rate is increased from 10 to 10000 ATM cells and the buffer from 10 to 1000 ATM cells. Both with step length 20



Cell delay graph
The peak rate is increased from 10 to 10000 ATM cells and the buffer from 10 to 1000 ATM cells. Both with step length 20

The shaper driver

The whole driver is build around a single Matlab function called vbr_video. The parameters passed to the function are:

- B – the buffer size given in ATM cells where an ATM cell size is 53 bits with an effective data size of 48 bits
- r – the output peak rate also in ATM cells
- f – the incoming frame rate in frames per second
- data – vector containing the video data as frame size stamps
- bf – boost factor used to speed up the simulation at the expense of accuracy, meant mostly for debugging purposes. Should be 1 at all times
- suppress – 1 to show some test graphs, 0 otherwise.

The function returns the following parameters:

- meanSpeed – vector containing the mean speed in ATM cells per second
- buff_ocup – vector for buffer occupancy in percent for each observation. Observations are not necessarily equally spaced in time.
- cell_delay – cell delay vector with each element corresponding to a cell outputted from the buffer
- cell_loss – vector containing the number of lost cells per frame

The shaper is a simplified version of the leaky-bucket algorithm. It uses a token buffer of size one token and a token arrival rate of 1/r. In other words we are forcing the shaper to send always at peak rate, assuming there are cells waiting in the buffer.

The variables **next_frame** and **next_token** are the event time for receiving the next frame or the next token.

If the next event to occur is an incoming frame then the next frame is read from the data vector and divided into ATM cells. Observe that there is no guarantee that the frame data will divide exactly into ATM cells, but because ATM cells have fixed size the last cell we assume the last cell is padded with zeros to cover for the full size.

If there are more cells than free size in the buffer, the buffer will accept just enough cells to fill itself and discard all the rest. The number of lost cells is stored in the **cell_loss** vector.

The cells accepted in the buffer are marked with a time stamp in the **cell_delay** vector and the **next_frame** variable is updated with the time for the next arrival of a frame.

If the next event is the arrival of a token then we first do flow speed calculation. The variable **cellTimer** is incremented by 1/r. If cellTimer is equal to or greater than one then it means one second of simulation elapsed. The variable **counter**, which is the number of cells that have been outputted under one second, is saved in the vector **meanSpeed**. Then if there are cells in the buffer, the first cell in the buffer is outputted and its delay time is computed and saved in the vector **cell_delay**. Also the buffer size variable, **buffer**, is increased together with **counter**.

The main while-loop does a check for frames left to compute or cells left in the buffer. The program tries to approximate the duration of the simulation by the number of frames in the **data** vector. A graphical bar is used to show the amount of simulation done, as percentage. When huge buffers are simulated the latency of the simulation increases greatly and the bar reaches hundred percent long before the simulation is done.

```
function [meanSpeed, buff_occup, cell_delay, cell_loss] = vbr_video(B, r, f, data, bf,
supress)
%TRAFFIC SHAPER FOR VBR VIDEO SOURCE
%            Usage: [meanSpeed, buff_occup, delay, loss] = vbr_video(B, r, f, data)
%            B       - size of the shaper's internal buffer given in # of ATM cells
%            r       - PEAK RATE for the shaper's output process given in # of ATM cells
%            f       - frame rate for the VBR video traffic
%            data    - vector containing the video data as frame size stamps
%            bf      - boost factor for the outer for-loop. Increases speed at the
%                      expense of accuracy. The loop uses every bf-th frame. Set to
%                      1 for best accuracy, higher integer for boosting
%            supress - turns off screen output
%
%            meanSpeed   - mean speed per second
%            buff_occup  - buffer occupancy rate
%            delay       - cell delay vector
%            loss        - cell loss vector
%
%            Neet graphs will be shown if the shaper doesn't bail out!
%

count = length(data);
p2 = count / 10;

if supress > 0
   figure;
   subplot(2,1,1), plot(frame), ylabel('Kbit/s');
   title('Video frames size stamp');
   subplot(2,1,2), plot(frame .* f), ylabel('Mbit/s'),;
            title('Bitrate for video VBR data');
end

next_frame = 0;            %time for next incoming frame
next_token = 1/r;          %time for next token
cell_in  = 1;              %cell index for cells entering the shaper
first_cell = 1;            %first_cell in the buffer
buffer = B;                %free space in buffer
ATM_data = 48 * 8;         %amount data in one ATM cell
i = bf;
w = 0;
counter = 0;               %count no. of cells passing in one second
cellTimer = 0;             %count up to 1 sec.
index = 1;


h=waitbar(0,'Performing simulation...');
while (i <= count | buffer < B)                        %all frames and cells must be processed
   w = w + 1;
   buff_occup(w) = (B - buffer) .* 100 ./ B;
   if (next_frame < next_token & i <= count)
      this_event = next_frame;
      next_frame = this_event + 1/f;
      cells = ceil(data(i) / ATM_data);                %fragment frame
      if cells > buffer
         cell_loss(i/bf) = cells - buffer;             %buffer full! Discard some cells
         cells = buffer;
              %compute amount remaining cells
         buffer = 0;
                                                       %buffer full!
      else
         cell_loss(i/bf) = 0;                          %no cell loss. Good!
         buffer = buffer - cells;
      end
      cell_delay(cell_in : cell_in + cells - 1) = this_event;
      cell_in = cell_in + cells;
      i = i + bf;

   else
      this_event = next_token;
      next_token = this_event + 1/r;                   %schedule next_token
      cellTimer = cellTimer + 1/r;
      if cellTimer >= 1
         meanSpeed(index) = counter;
         counter = 0;
         index = index + 1;
         cellTimer = 0;
      end
```

```
      if buffer < B                                    %cells in buffer
         cell_delay(first_cell) = this_event - cell_delay(first_cell); %compute cell delay
         buffer = buffer + 1;                          %increase space in buffer
         counter = counter + 1;                        %count one cell
         first_cell = first_cell + 1;                  %point to the next cell
         if first_cell > cell_in
            first_cell = cell_in;
         end
      end
   end
   p1 = i ./ 10;
   waitbar(p1 ./ p2);
end

close(h);
```