

Computer Networks

ELEC2920

Multimedia Communications

Lab # 3

Aubry Springuel
Oscar Medina Duarte

Abstract: The objective of this laboratory work was to discover the particular constraints related to multimedia data transmissions through computer networks.

- 1) Why don't we use a protocol such as TCP which guarantees the arrival of the packets (and in the correct order) and ensures an equitable division of the band-width?

UDP is commonly used with multimedia applications, such as Internet phone, real-time video-conference, and streaming of stored audio and video... We mention that all of these applications can tolerate a small amount of packet loss, so that reliable data transfer is not absolutely critical for the application's success. Furthermore, real-time applications, like Internet phone and video conferencing, react very poorly to TCP's congestion control. For these reasons, developers of multimedia applications often choose to run their applications over UDP instead of TCP.

- 2) What do we mean by unequal error protection?

In this lab we have done a pictures transmission scheme which combines a picture coder with unequal error protection (UEP) across packets. The picture coder produces a bit stream, decodable at different bit rates. It allows computation-time and memory-limited decoding on less powerful hardware platforms, And it can substantially improve the quality and acceptance of the transmission services. Picture coding has been proposed previously to address the problem of transmitting video over an unknown channel. (From :*Scalable Internet Video Streaming With Unequal Error Protection*)

- 3) Within a multimedia application, what makes it possible to achieve unequal error protection or rate adaptation without having to have various copies of the media on the server? Briefly explain.

The multimedia files are very big and they contain a big amount of not very relevant information. A codestream as these of the JPEG 2000 frames are divided into packets. A JPEG 2000 packet contains the information regarding a given component, a given resolution and a given quality. Thus, for a frame containing 6 levels of resolution, 4 layers of quality and 3 components, there will be 72 JPEG 2000 packets.

We can thus well adapt the protection of the transmitted packets in accordance with the sensibility of the content of these packets. That is the unequal error protection. For that, we can transmit this information more than one time.

We can also adapt the rate to transmit also the important info (some packets) and not these with less of content (the other packets).

- 4) What is the interest of such a method - compared with the first one – to transmit the data packets?

This method consists thus in distributing each packet on a given number of RTP packets, even if this JPEG 2000 packet could be obtained in only one RTP packet.

Forward Error Correction, of which the concepts were explained during the first TP, is a mechanism consisting in inserting redundancy in the packets sent on the networking order to compensate for the packet losses.

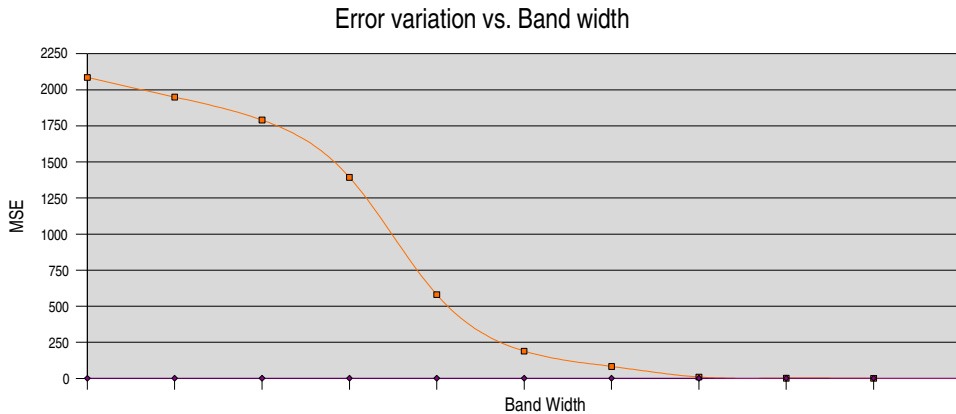
- 5) Calculate several PSNR of JPEG 2000 sequences transmitted with the simulator, exploiting the parameters of the network and visualize the result.

In order to visualize the effect of this parameters on the simulated model, attention to the Mean Square Error between the sent and reconstructed image frames was emphasized. The Band width of the critical path between the sender and receiver was changed several times in order to appreciate the relationship between the MSE and band width as expressed on the following table:

MSE	Band Width	Empirical
2085.21	1	Poor
1949.48	1.05	Poor
1790.28	1.1	Poor
1393.27	1.15	Poor
580.5	1.2	Bad
189.48	1.25	Bad
82.98	1.26	Good
9.58	1.27	Acceptable
1.98	1.28	Acceptable
0	1.29	Excellent
0	1.3	Excellent

The empirical column is related to the visualization quality of the video transmitted, for which the following measures were proposed: Poor, when the image has a lot of flickering and is non pleasant to view. Bad, when there is some flickering but the general idea of the video can still be captured. Good, when there is few flickering and most of the video is understandable. Acceptable, very little of null flickering and/or some visual defects can be detected. Excellent, all the video is smooth and no visual defects are detected.

We can see that the amount of error decreases significantly fast as the band width decreases, as we can better appreciate at the following figure:



From this graphic its interesting to see that there exists a critical range of MSE change that occurs for values of the band width near 1.2, that is that for this simulation, the amount of error is very sensible around 1.2Mb of band width.

For the second TP, part of an actual error correcting code mechanism was introduced to the sender and receiver, and the requested task was to complete and verify the functionality of such by altering its main parameter, the FEC, which indicates the amount of redundant information is going to be included into the packets in order to protect them better from packet losses. A FEC of 1, means that no redundant information is going to be sent, and a FEC of 2 indicates that 100% of redundancy will be used (ie. Send two times each packet).

On the sender side, we received a code with a FEC fixed to 1 wich means that no protection is used, the idea here was to create a more clever approach than just fixing it to a higher number. The most intuitive approach was to take into consideration the number of layer and resolution into a linear formula like this:

$$FEC = ((MaxOfLayers / (MaxOfLayers + CurrentLayerIndex)) + (MaxResolution / (MaxResolution + CurrentResolutionIndex)))/2$$

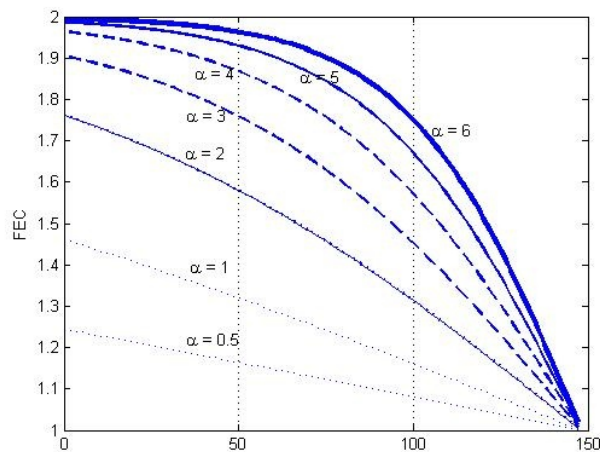
Which lead to a significant improvement of the received image, giving a MSE of 78.172 and an empirical rate of Bad, but with no flickering. But still, the variation of the FEC doesn't seem very wise, so the next not so intuitive step was to try a decaying exponential function like:

$$FEC = (2 * (1/(1 + e^{(\alpha * (((MaxOfLayers - CurrentLayerIndex)) / (MaxOfLayers)))))) * \omega_1$$

$$FEC += (2 * (1/(1 + e^{(\alpha * (((MaxResolution - CurrentLayerIndex)) / (MaxResolution)))))) * \omega_2$$

$$FEC += (2 * (1/(1 + e^{(\alpha * (((MaxComponent - CurrentCompIndex)) / (MaxComponent)))))) * \omega_3$$

Where ω_n is the weight of each frame component such that $\omega_1 + \omega_2 + \omega_3 = 1$ and the greatest weight corresponds to the most important to preserve part of the frame. For this experiment, the best error was found for the values .7, .3, .1 when $\alpha = 3$ the plot of this function shows the effect of α for the change rate over the layer.



As we can see in the graphic, the larger FEC is obtained for the lower layers, or components.

On the receiver side, the only goal was to actually take advantage of the FEC by implementing part of an algorithm to decide which packets to drop. The policy to decide which packets to drop and which packets to preserve is related to the ability of the receiver to decode successfully each BOP. In more concrete words, the packets received that contain a higher layer for frames that were lost, are not useful because its most significant bits are required to decode. The second condition to decode correctly the frames from the BOP, is that there exists enough information to decode the rest of frames, this is explicitly expressed by the line:

```
minimum_Packets = N_RS / bop->Ppack[kk]->fec; /* Compute minimum required pkts */
```

For the case that there is not enough information to decode or there exists not significant enough layers, the whole BOP is the rejected and all of the information contained by it. The key piece of code that was modified can be found at the annex 2.

The goal of the third and last TP about the streaming video was to set up the mechanism of the rate adaptation. For the test, we had a script with a bandwidth that change during the simulation. We had four different layers 1,5Mb/s, 900kb/s, 500kb/s and 200kb/s.

We had Process: Layerchange, a value: interval_ and five variables: layer_incr_; last_seq; recv_; lost_ and rtt_.

And we have achieved the code in annex. The goal was to compare the layer. Us strategy compares thus the number of packets lost since the last call of the procedure with the relative max redundancy estimated per layer. If the layer is 0, the redundancy is estimated at 0.75, if 1 estimated at 0.5, if 2 estimated at 0.25, if 3 estimated at 0.

If the lost_ value is smaller than the relative max redundancy estimated, you must let up the layer. If the lost_ value is greater than the relative max redundancy estimated for the faster layer, you must let down the layer.

For the layer 0 you can not have faster rate. For the greater layer (3), you can not let down the rate. For layer 3, you must thus increase the rate if the packets lost a certain level to use the faculty adaptation of the FEC.

We use the interval_ to increase the time if the rate is not the same to react rapidly.

Annex:

```
set oldinterval_ 0.5
set oldlayer_ 3
```

```
Agent/UDP/BOPreceiver instproc layerchange {} {
```

```
    #This procedure is recursively called at the receiver
    #At the end of each call, the receiver requests the sender to
    #increase the number of quality layers by a value layer_incr_
    #(or decrease if layer_incr_ is negative)

    global ns oldinterval_ oldlayer_

    $self instvar last_seq_ lost_ recv_ interval_ layer_incr_ rtt_

    #last_seq_ : sequence number of the last packet received
    #lost_ : number of packets lost since the last call of the procedure
    #recv_ : number of packets received since the last call of the procedure
    #interval_ : time interval_ between each call of the procedure
    #layer_incr_ : variation of quality asked by the receiver
    #rtt_ : estimation of the Round Trip Time between the sender and receiver

    set security_ 0.9

    set layer_incr_ 0
    if {$oldlayer_ == 0} {
        if {$lost_ < [expr $security_ * 0.75 / 1.75]} {
            set layer_incr_ 1
            puts "At time [$ns now], Switching from layer 0 to layer 1."
        }
    } else if {$oldlayer_ == 1} {
        if {$lost_ > [expr $security_ * 0.75 / 1.75]} {
            set layer_incr_ -1
            puts "At time [$ns now], Switching from layer 1 to layer 0."
        } else if {$lost_ < [expr $security_ * 0.5 / 1.5]} {
            set layer_incr_ 1
            puts "At time [$ns now], Switching from layer 1 to layer 2."
        }
    } else if {$oldlayer_ == 2} {
        if {$lost_ > [expr $security_ * 0.5 / 1.5]} {
            set layer_incr_ -1
            puts "At time [$ns now], Switching from layer 2 to layer 1."
        } else if {$lost_ < [expr $security_ * 0.25 / 1.25]} {
            set layer_incr_ 1
            puts "At time [$ns now], Switching from layer 2 to layer 3."
        }
    } else if {$oldlayer_ == 3} {
        if {$lost_ > [expr $security_ * 0.25 / 1.25]} {
            set layer_incr_ -1
            puts "At time [$ns now], Switching from layer 3 to layer 2."
        }
    }
}

if {$layer_incr_ != 0} {
```

```

        set interval_ [expr $rtt_ * 4.0]
    } else {
        set interval_ [expr $oldinterval_ *1.5]
    }

    set oldlayer_ [expr $oldlayer_ + $layer_incr_]

    # memory hack
    set oldinterval_ $interval_
}

```

Annex 2:

(Fragment from receiver.c)

```

...
    if (bop->numreceived<N_RS) {
        int minimum_Packets = 99999; /* Infinity... */
        int kk2 = 0;

        for (kk=0;kk<bop->numJ2Kpack;kk++) {

            minimum_Packets = N_RS / bop->Ppack[kk]->fec; /* Compute minimum required pkts */
            unsigned char tmp=0;

            if (bop->numreceived < minimum_Packets){ /* If fec condition fails */
                /* Then remove */
                rescomp[(bop->Ppack[kk]->id)%numrescomp]=1;
                fwrite(&tmp,1,1,fframeOUT);
            }else{
                /* Include it */
                if (rescomp[(bop->Ppack[kk]->id)%numrescomp]==1) {
                    unsigned char tmp=0;
                    fwrite(&tmp,1,1,fframeOUT);
                } else {
                    fseek(fframeIN,bop->Ppack[kk]->start_pos,SEEK_SET);
                    for (kk2=bop->Ppack[kk]->start_pos;kk2<bop->Ppack[kk]->end_pos+1;kk2++)
                {

                    unsigned char tmp;
                    fread(&tmp,1,1,fframeIN);
                    fwrite(&tmp,1,1,fframeOUT);
                }
            }
        }
    }
}
...

```