Software Engineering

Screen Share - Spec Document

Aditya Agarwal (111901005), Team Member

OVERVIEW

In this subject, we are trying to make a lab monitoring system that has the following modules:

1. UX
2. Dashboard
3. Screen Sharing
4. White Board Sharing
5. Content
6. Network

We are implementing the screen sharing feature, enabling students (clients) to share their screen with the professor (server), which includes different functionalities and specifications one must follow to satisfy various constraints.

The screen sharing feature is one the most important feature of the project as it ensures fairness during any lab/test session among students and facilitates the professor to take the session smoothly.

OBJECTIVE

The screen sharing feature includes different functionalities on the client side and server side so the following are the objectives on the client side:

1. Capturing the screen of the student (client):
    a. We took the screenshot of the client screen and then push it to a queue for further processing.

2. Process the captured screen:
    a. We take the last captured screen from the queue and then process the image according to the specification requested by the server.
    b. The server states if the client should share a screen (send packets) or not, it also states the desired resolution of the screen that the client should share.
    c. We first compare the new screen and the last one captured, we send only the updated part of the screen to save the bandwidth.
3. Processing on the server side:
    a. The server is subscribed to each client so that it gets notified as it receives the screen image (the changed part).
    b. On receiving the image, the server pushes that inside the queue of that particular user.
    c. It patches the updated part of the image and sends it to the viewModel of screen sharing.

PSEUDO CODE AND CLIENT CLASS EXPLANATION:

```
/* This class has all the methods and functions that a client needs to share his
screen*/

Class screenSharingClient(){


Networking communicator object
/* we instantiate a communicator object to get the details from the server like
resolution, targeted IP, and to send back the screen share images  */

function Capture() {

	/* utility function to capture the image */

	}

}

function process(){

	/* utility function to process the image */

}

function Compress(req_sz, curr_fr) {

	/* utility function to compress the image */

}

/* the class which maintains the information about each client separately */
Class SharedScreen {
	IP
	/* IP of the client machine */

	Name
	/* Name of student (client) sharing screen */

	FrameQueue
	/* queue that have to be presented*/

	FinalImageQueue
	/*queue which has all processed images*/

	CurrentImage
	/*the latest image we received from the client */

	Bool Pinned
```

```
        /*bool variable which says if the user is pinned or not*/

        Timer timer
        /* some threshold timer after we stop receiving the packets which we assume
        the client is disconnect*/
}

/*class of server to receive the shared screen from different clients and process
them accordingly*/
Class ScreenShareServer:

        /*network class object to communicate over network*/
        Networking communicator;

        /*map to maintain all the subscribers with their IP as key */
        Map<IP, SharedScreen> subscribers;

        Constructor()
                /* instantiate networking object*/
                /* subscribe to the networking for OnDataReceive()*/

        OnDataReceive(data)
                /* Receive packet from the client via networking */
                /* Based on the packet header, do further processing */
                        /* SUBSCRIBE → SubscribeUser(ip) */
                        /* UNSUBSCRIBE  → UnsubscribeUser(ip)*/
                        /* IMAGE → Stitch(ip, converted array) */
                        /* CONFIRMATION → UpdateTimer(ip) */

        SubscribeUser(ip)
                /* add this ip to the map */

        UnSubscribe(ip)
                /* remove this ip from the map */

        BroadcastClientsInfoOfSS(currentWindowUsers)
                /* resolution of the image to send to each client */
                /* whether to send the packet or not */

        Stitch(ip, Bitmap array)
        /* Image Processing Team  */

        UpdateTimer(ip)
                 /* start/reset timer for the ip with the OnTimeOut() */

        OnTimeOut(ip)
                 /* callback for timer */
}
```

PSEUDO CODE FOR STITCHING IMAGE

```
Stitch(ip, new_res, list of : {x, y, (R,G,B)}):
      screenShareObj = subscribers.get(ip)
      ImageFrameList = screenShareObj.FrameQueue.pop()
      for (auto {x,y,(R,G,B)}: list){
            currentImage[x][y] = (R,G,B);
}
```

ANALYSIS

Here we are sending only those pixels which got changed in the next frame and their coordinates. Using these pixel values at the server end we can iterate over the current image frame and then update the pixel value of the image to get the next frame. The complexity of this algorithm will be on the order of size of the list (i.e O(Image Size)) sent as an argument to the stitch function.

CODE FOR BENCHMARKING :

```
/*** Naive Code ***/

using System.Drawing;
using System.Diagnostics;
using System.Threading;

Bitmap img = new
Bitmap(@"C:\\Users\\aditya\\Pictures\\Screenshots\\ss1.png");
Bitmap img1 = new
Bitmap(@"C:\\Users\\aditya\\Pictures\\Screenshots\\ss2.png");
int count = 0;

Stopwatch stopwatch = new Stopwatch();
stopwatch.Start();
for (int i = 0; i < img.Width; i++)  {
    for (int j = 0; j < img.Height; j++)  {
        Color pixel1 = img.GetPixel(i, j);
        Color pixel2 = img1.GetPixel(i, j);
```

```
        if (pixel1 == pixel2)
            count++;
    }
}
stopwatch.Stop();
Console.WriteLine("Difference b/w 2 images is {0} pixels", count);
Console.WriteLine("Elapsed Time is {0} ms", stopwatch.ElapsedMilliseconds);
```

Output :

Difference b/w 2 images is 1944923 pixels

Elapsed Time is 5372 ms

/*** OPTIMISED CODE BENCHMARK CODE***/

On using the following code :

```
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;

Bitmap img = new
Bitmap(@"C:\\Users\\amish\\Pictures\\Screenshots\\ss1.png");
Bitmap img1 = new
Bitmap(@"C:\\Users\\amish\\Pictures\\Screenshots\\ss2.png");

int count = 0;

Stopwatch stopwatch = new Stopwatch();
stopwatch.Start();

bool ProcessUsingLockbits(Bitmap processedBitmap, Bitmap processedBitmap1)
{
    BitmapData bitmapData = processedBitmap.LockBits(new Rectangle(0, 0,
processedBitmap.Width, processedBitmap.Height), ImageLockMode.ReadWrite,
processedBitmap.PixelFormat);

    int bytesPerPixel =
Bitmap.GetPixelFormatSize(processedBitmap.PixelFormat) / 8;
```

```
    int byteCount = bitmapData.Stride * processedBitmap.Height;
    byte[] pixels = new byte[byteCount];
    IntPtr ptrFirstPixel = bitmapData.Scan0;
    Marshal.Copy(ptrFirstPixel, pixels, 0, pixels.Length);
    int heightInPixels = bitmapData.Height;
    int widthInBytes = bitmapData.Width * bytesPerPixel;
    processedBitmap.UnlockBits(bitmapData);

    BitmapData bitmapData1 = processedBitmap1.LockBits(new Rectangle(0, 0,
processedBitmap1.Width, processedBitmap1.Height), ImageLockMode.ReadWrite,
processedBitmap1.PixelFormat);

    int bytesPerPixel1 =
Bitmap.GetPixelFormatSize(processedBitmap1.PixelFormat) / 8;
    int byteCount1 = bitmapData1.Stride * processedBitmap1.Height;
    byte[] pixels1 = new byte[byteCount1];
    IntPtr ptrFirstPixel1 = bitmapData1.Scan0;
    Marshal.Copy(ptrFirstPixel1, pixels1, 0, pixels1.Length);
    int heightInPixels1 = bitmapData1.Height;
    int widthInBytes1 = bitmapData1.Width * bytesPerPixel1;

    processedBitmap1.UnlockBits(bitmapData1);


    for (int y = 0; y < heightInPixels; y++)
    {
        int currentLine = y * bitmapData.Stride;
        int currentLine1 = y * bitmapData1.Stride;
        for (int x = 0; x < widthInBytes; x = x + bytesPerPixel)
        {
            int oldBlue = pixels[currentLine + x];
            int oldGreen = pixels[currentLine + x + 1];
            int oldRed = pixels[currentLine + x + 2];

            int newBlue = pixels1[currentLine1 + x];
            int newGreen = pixels1[currentLine1 + x + 1];
            int newRed = pixels1[currentLine1 + x + 2];

            if (oldBlue != newBlue || oldGreen != newGreen || oldRed !=
newRed)
                count++;
        }
```

```
    }
    return true;
}

ProcessUsingLockbits(img, img1);
stopwatch.Stop();
Console.WriteLine("Difference b/w 2 images is {0} pixels", count);
Console.WriteLine("Elapsed Time is {0} ms", stopwatch.ElapsedMilliseconds);
```

Output:

Difference b/w 2 images is 128677 pixels

Elapsed Time is 90 ms

ANALYSIS

Capture Screen BenchMarking:

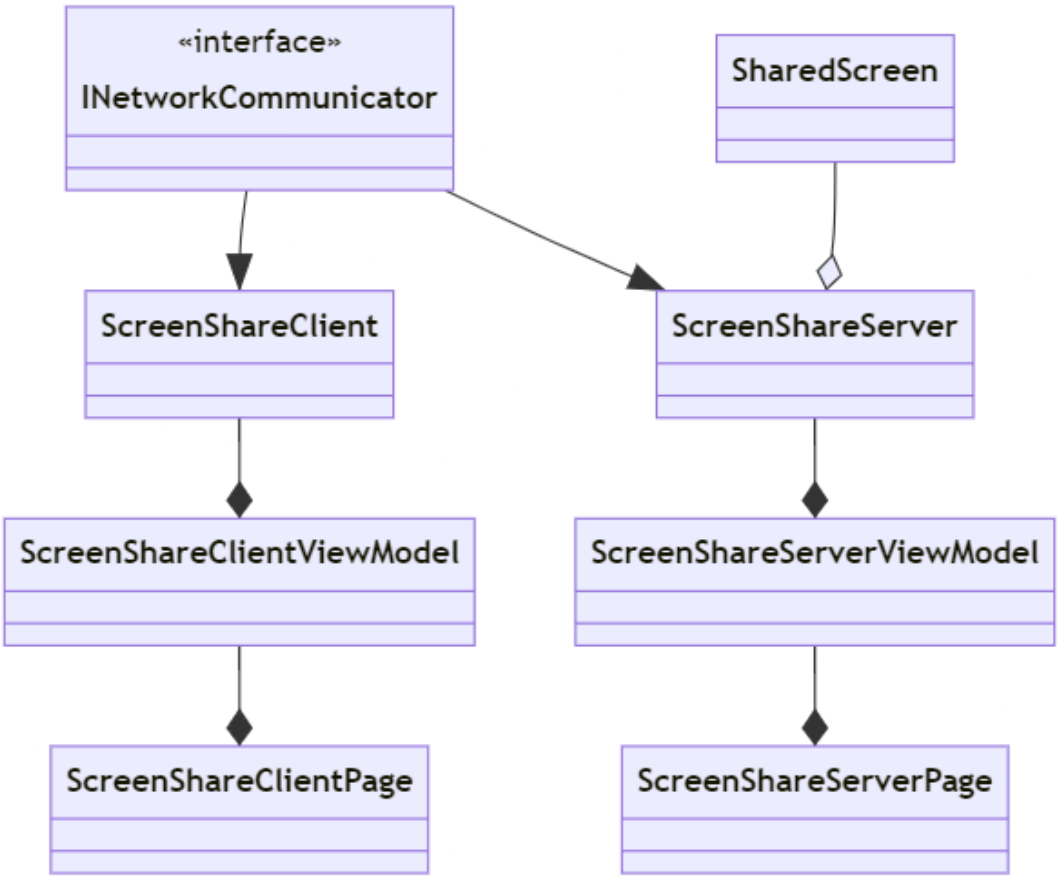
We have written two approaches above, and compared the output of those. The second approach is more optimized as evident from the time elapsed in output. .

First we compared two images with nested for loop and count the changed pixels which is very slow and not feasible to share screeen

We then tried using the inbuilt BitMapdata datatype in C# and we see that we can compare 2 images of 1080p in about 70 - 95ms.

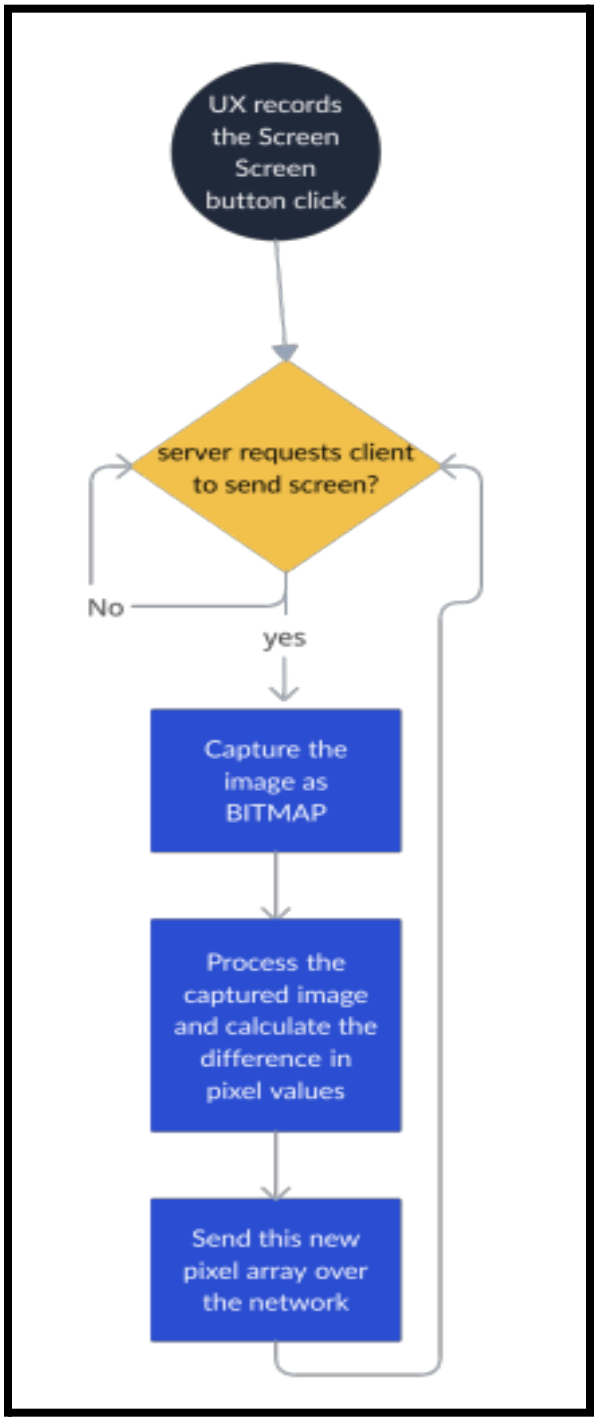
Also instead of comparing two 1080px images, we compared 720px images which give a further boost to our algorithm resulting in sharing more frames at the same time, and give a more enhanced user experience.

CLASS DIAGRAM

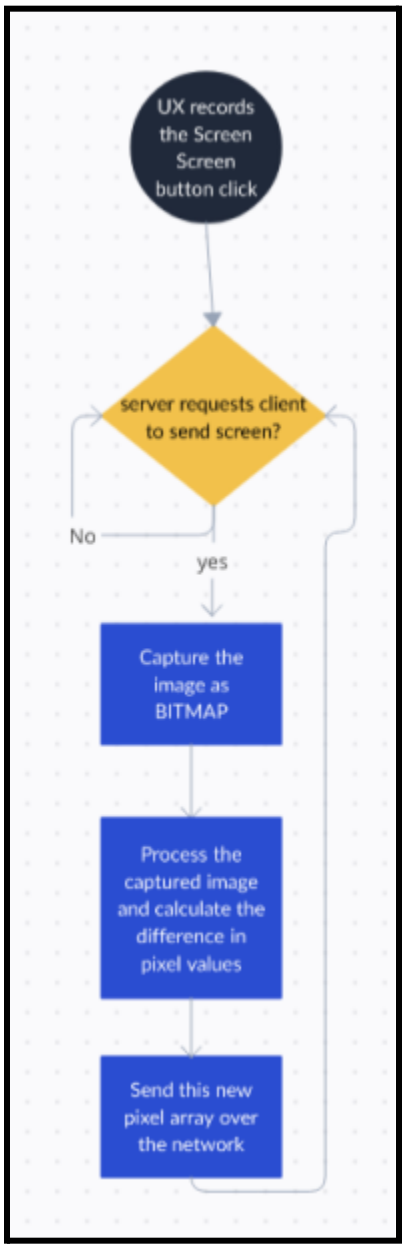


The class diagram for our share screen module.

ACTIVITY DIAGRAM FOR CLIENT



ACTIVITY DIAGRAM FOR SERVER



PERFORMANCE

We have thought of a couple of techniques in order to increase the performance of our code.

1. **Multithreading** - we tried to implement the functions which are independent of each other in different threads, in order to exploit the multicore CPU processing power. This smoothens the process for each function and didn't hinder the processing of a function if another function stops/hangs due to some reason.
2. **Patching** - after capturing the image, instead of sending the whole image we first compare it with the previous screenshot if they both are in the same resolution, and only send the updated part of the new screen. Then on the server side, we stitch it with the old screen and make the new updated screen. This saves bandwidth and resources.
3. **Latency Handling** - As stated above that we have introduced multithreading in our program, so it may be the case that different functions different times to complete due to some latency. Hence we have introduced a Queue data structure in our program which makes them independent as one pushes into it and the other only pops out of the element if the queue is not empty (also there are no race conditions involved as one writing at the end of the queue and other is only reading from the beginning).

SUMMARY AND CONCLUSION

This document discusses the different approaches to capturing and processing the screen shared and compares them on basis of complexity. Also, we did an analysis of all the approaches which helps us to pick the more optimized approach. Then we discussed how the client side and server side coordinate between themselves in order to show images.

FUTURE WORK

1. We may introduce that server can also share its screen, right now only clients (students) can share them.

2. We can enhance the patching algorithm, in which instead of traversing all the pixels if we can store multiple pixels' information in the same pixels which is a kind of compression that will save more time.
3. Right now when the server (professor) switches the screen or changes the tab, there may be some latency in showing the screen of a new user on screen. We can improve that if we cache some screens.

References

- https://learn.microsoft.com/en-us/dotnet/api/system.drawing.image?view=dotnet-plat-ext-6.0
- https://csharpexamples.com/fast-image-processing-c/
- https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics?view=net-7.0