INF5090/9090
State Exchange in Distributed Applications

Magnus Evensberget
Institute of Informatics
University of Oslo
magnusev@ifi.uio.no

Vinay Setty
Institute of Informatics
University of Oslo
vinay@ifi.uio.no

***Abstract*—This assignment is part of the Real-time Application Mobility Platform (TRAMP) project with focus on distributed multimedia applications with real-time requirements. The goal of the assignment is to design and implement at least two components (producer and consumer) of a real-time application (such as a media player) and evaluate the provided distribution framework with respect to delay, throughput and other relevant metrics. Above is an overview of the provided framework. Your objective is to design and implement the producer and consumer application parts (the two gray components). The producer/consumer duo can run locally on one machine or be distributed over a network. In addition, multiple consumers can subscribe to the same produced data and receive identical copies.**

I. Introduction

In our assignment we were to use the Real-time Application Mobility Platform (TRAMP). This is a project developed by the Distributed Multimedia Systems (DMMS) group at the University of Oslo.

This is a assignment given to us in the course INF5090 - Advanced Topics in Distributed Systems which is a course given by the University of Oslo, as well as Lancaster University in England and University of Mannheim in Germany. The teachers are Thomas Plagemann and Vera Goebel with their teaching assistants Piotr Kaminski and Hans Vatne Hansen. These are also the people that developed the TRAMP framework together with some other developers all from the University of Oslo.

The TRAMP project was started in early 2010, the goal of the framework is to "make a migration system for real-time applications, such as Spotify and Skype, where users can take applications with them when they move." [2]

Lets say you sit on your computer talking to a friend on skype. You need to go, but you got more to say to him/her. TRAMP will let you stream the conversation to your phone seamlessly and when you get to work, you can then transfer the conversation to your work computer.

II. Related work

A. Skype

As we've mentioned earlier in the report Skype is one of the applications the developers of the TRAMP framework reference to. Skype was the first peer-to-peer voice over IP (VoIP) network, and requires minimal infrastructure in order to function. Skype came on market in late 2003 developed by Niklas Zennstrm and Janus Friis, the two main developer behind the filesharing system Kazaa. this was bought by eBay in late 2005 and again by Microsoft in May 2011. Since Skype is propriatory, the code is not available and this is therefore based on observations and partial reverse engineering of the protocol.

The skype arcitecture is designed around three entities: Supernodes, regular nodes and the login server (See Figure 1). Each regular node has a cache of the IP address and portnumber to all reachable Supernodes. The Supernodes are a subset of the regular nodes. If a node has high uptime, good bandwidth and is not restricted by firewalls or Network Address translation (NAT) it can be chosen as a Supernode[6]. This will ofcourse put some extra stress on the nodes not

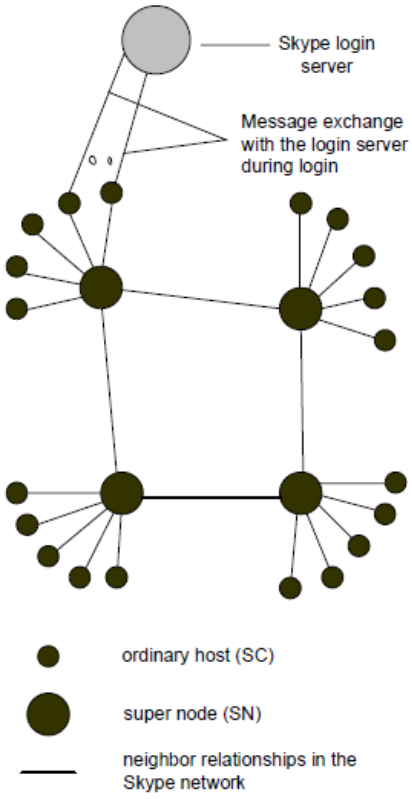

Figure 1. Picture of the skype arcitecture [6]

behind NAT, but also make Skype possible to work for free due to the little infrastructure needed in order to make the network work. Another issue for these Supernodes is that they are used as third party for UDP hole punching to connect clients behind NAT. UDP hole punching is done by letting the regular nodes behind NAT connect to the third party (the Supernode) thereby opening ports that they can use for direct traffic between the two regular nodes. This is only open as long as there is communication traffic going. If there is a prolonged absence of traffic Skype sends "keep alive" packets instead of closing the connection and then having to use the Supernode as a third party again redo the connection.[3]

This is something that might be implemented in TRAMP to scale up the network.

B. P2P streaming

P2P streaming works by having all nodes in the network relay the stream to other nodes, thereby reducing the upload link needed by the sourcenode, as it does not need to upload the stream to all connected nodes. SopCast is such an application.

SopCast is a P2P streaming application developed as a student project at Fundan University in China in 2005 and has become a very popular way to watch live video streams. The name SopCast is short for "Streaming over P2P Cast", and the protocol used is built by the SopCast team.[5] The SopCast source code is propriatory, so as with Skype this information is based on observations and mesurements.

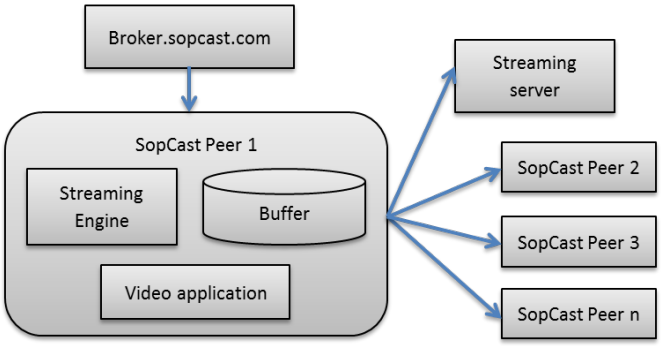

Figure 2. SopCast Arcitecture

Sopcast works by having a tracker with all public channels allowing each node to have a list of available channels, it also keeps track of the chunk information created by the Streaming server. Each node consists of a Streaming engine, a Buffer and a Video application. The Streaming server is the special node that is recording the stream. this node is converting the video into segments of equal size, then pushing it out.

SopCast uses a mesh-pull topology, this means that all peers has to pull the missing segments of the stream from their neighbours. Peers are adverticing what segments they have available via Buffer Maps.

There are two buffers that SopCast maintains; the reciever buffer and the client buffer. The reciever buffer is a buffer for all the chunks pulled from neighbours, when this buffer is full SopCast launch a media player that uses the clients buffer to play the stream.

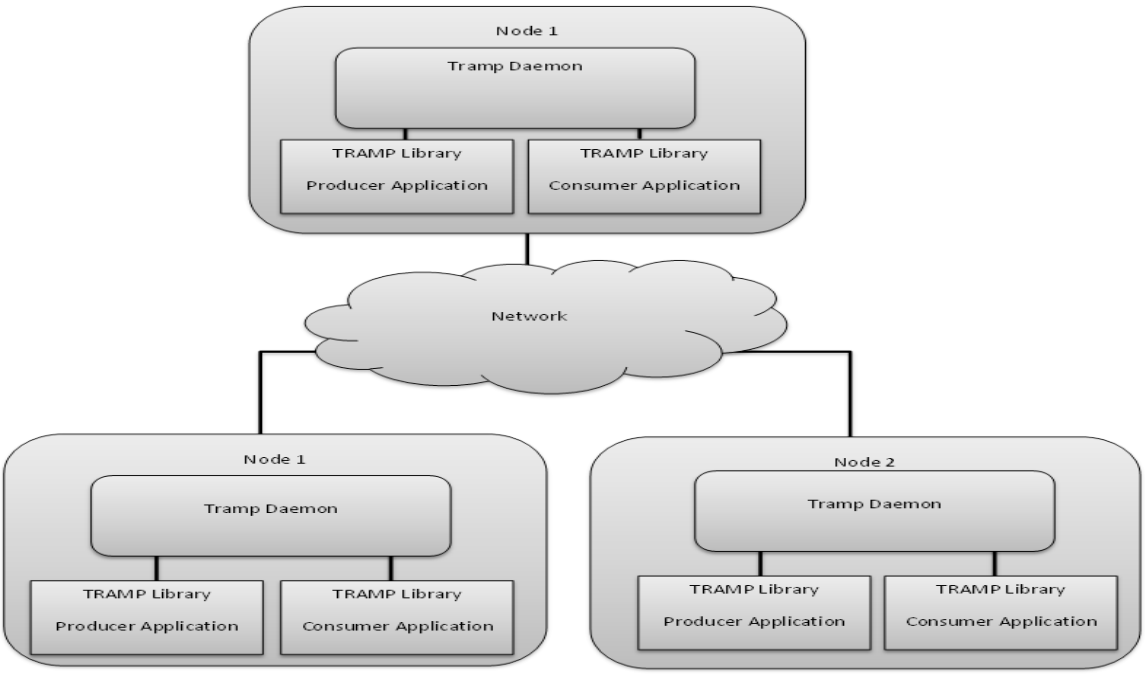

Figure 3. The TRAMP Architecture

III. System Design

The TRAMP platform is a platform designed to stream real-time data from a device to other devices connected to the session, mobility is the keyword. Lets take the example given in the introduction; You are on a skype call with some of your friends when you find out you need to go out of the house. Traditionally you then had to end the skypecall and call them up using your phone. The goal for TRAMP is to be able to seamlessly enter the conversation without your friend noticing you just changed device.

In TRAMP this is done by having a TRAMP daemon running on each device, with a consumer and a producer. The producer produces data, putting it into the shared memory, the TRAMP daemon then pushes the data to all other daemons connected to this session. When a TRAMP daemon gets data it puts it into the shared memory so the consumer can read it. All nodes has a TRAMP daemon, a producer and a consumer running. The shared memory of all nodes in the session are in synch.

IV. The choice of streamer

When we got this assignement we had to look at what we would implement in order to test the TRAMP platform. In the sample code there was implemented a simple chat program, so we decided to implement something that streamed a lot of data. There are several ways to do this; we could make a file sharing system, an audio streaming application (like Skype) or a video streaming application. We decided on looking into FFMPEG.

FFMPEG is a complete cross-platform solution to record, convert and stream video and audio. It is a free software licensed under the LGPL or GPL (dependent on the configuration), and is the leading multimedia framework today.[1] FFMPEG provides tools for converting, streaming, playing and analysing multimedia, as well as a full developers library with the possibility to create allmoast anything you want. Big projects such as QStream, VLC, GStreamer and Google Chrome has used the framework.

Due to time constraints we decided we did not have enough time to implement this framework. Instead we implemented a streamer that copied over one big file by chunking the big file into small bits, sending them and putting them back together on the other side. This will in our eyes show the capabilities of TRAMP just as well as a more sophisticated streamer like the FFMPEG solution we had planned.

A. Our implementation

To explain what we have done to the code, we must first talk about how the sample chat program work. We decided to use this as a base, as it had a lot of the code we needed anyways. HERE WE MUST TALK ABOUT THE IMPLEMENTATION VINAY DID

V. Proposed Optimizations

A. Shared Memory problem

REWRITE NEEDED Shared memory is memory that can be used to pass data between cores, or to other applications. As the applications or cores can access the data just as any other data in the Random Access Memory (RAM) its a very fast way of passing data between the applications, compared to regular message passing or software implementations such as UNIX domain sockets or CORBA. But it is less powerful, as all the processes has to run on the same machine.[7]

At the moment TRAMP is using shared memory for inter-process communication (IPC). It allocates 32mb of memory, but the problem we've found is that we get a lot of packetloss due to the producer being faster than the consumers. This problem occours when we are trying to send a file over the platform. The problem with the file sending is ofcourse that the platform tries to send it as quickly as possible, where usual real-time applications such as skype sends a video stream that is not the total bandwidth, but 400kbps-1mbps (voice streams will be lower than this).

TODO: major security problems :TODO

B. Data transfer rate

Pushing more data than the recieving nodes can pull

C. Network topology

atm mesh network, should be something else

D. UDP vs TCP

User Datagram Protocol (UDP) was designed by David P- Reed in 1980, and defined in RFC768. UDP allows computer applications to send messages, or datagrams, to other hosts on an Internet Protocol(IP) without a setup phase like in Transmission Control Protocol(TCP). This stateless nature is also good for servers answering small queries from many clients. UDP also suport packet broadcast (sending packets to all hosts on a local network) and multicasting (subscriber based).

UDP does not guarantee that the packet you sent ends up at the host. Neither does it support packet ordering (that packets reach the reciever in the correct order). This is what TCP is for. TCP was specified in RFC675 in December 1974. This protocol uses a three-way handshake in order to set up a connection between two hosts. It also provides reliable transmission, error detection, flow control and congestion control.

1) Real-time Transport Protocol: The Real-time Transport Protocol (RTP) is a protocol developed by the Audio-Video Transport Working Group for the Ineternet Engineering Task Force (IETF). The first published RFC was published in 1996 with the code RFC1889. This was made obselete by the newer RFC3550, published in 2003[4]. This protocol is built on top of UDP and therefore does not provide Quality of Service (QoS) guarantees like delivery or packet order. The RPT standard defines two protocols. RTP and RTP Control Protocol (RTCP). This protocol provides QoS to the RTP protocol.

"RTP provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services."[4]

Due to the real-time nature of this protocol the majority of RTP implementations is build on UDP, but there are some implementations using TCP.

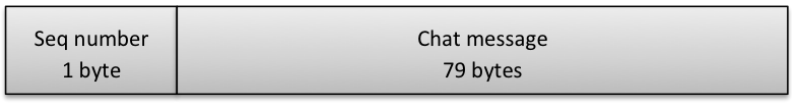

Figure 4. Packet header for the chat application

Figure 5. Packet header for the file sending addon

Figure 6. TRAMP Daemon header

VI. Evaluation

VII. Conclusions

References

[1] Ffmpeg-project homepage. April 2012.

[2] Tramp-project homepage. April 2012.

[3] D. K. Bryan Ford, Pryda Srisuresh. Peer-to-peer communication across network address translators. Technical report, Massachusetts Institute of Technology and Caymas Systems, Inc, 2005.

[4] R. F. V. J. H. Schulzrinne, S. Casner.

[5] R. Kostadinova. Peer-to-peer video streaming. Master's thesis, Royal Institute of Technologies, Stockholm, Sweden, 2008.

[6] H. G. S. Salman A. Baset. An analysis of the skype peer-to-peer internet telephony protocol. Technical report, Department of Computer Science, Columbia University, New York, 2006.

[7] K. G. Sarita V. Adve. Shared memory consistency models: A tutorial. Technical report, Western Research Laboratory, September 1995.