Timecritical network syncronization and signaling

Jenny Olsson

Stockholm 2013

KTH

Abstract.

This bachelor thesis is about synchronizing rendering over an network. Since projectors have a limited resolution several projectors needs to be used to get a specific high resolution. If these projectors are connected to different computers the rendering needs to be synchronized.
add stuff on how i did this

Summary

This thesis was written in the spring of 2013. It is an investigation into different solutions on how to sychronize rendering, on different computers, over an network. A possible use case is to manage projections that needs to combine a number of projectors to achieve a desired resolution, since projectors have a limited resolution.

Different solutions on how to achieve synchronizations has been investigated, starting with the solution implemented in the NTP-protocol.

Resumé

Preface

This thesis was written at 23c in Stockholm as the fulfilling of my bachelors degree in computer engineering from The Royal Institute of Technology in Kista. It was started in March of 2013 and consists of 15 ECTS points.

It presents a possible solution on how to synchronize the rendering processes on different computers over an network.

This thesis and the code presented has been written by me and in the cases where other sources has been used this is clearly citated.

Stockholm 2013

Jenny Olsson

Glossary

NTP

Network Time Protocol, used to sychronize computer clocks over an network.

Frameskipping

Skipping a certain amount of frames forward.

Acknowledgements

I would like to thank 23 Critters, especially Mathias Tervo and Kristoffer Smedlund, for presenting me with the problem and having great patiece in discusing it with me.

Finally i would like to thank Sanna Barsk for the extensive proof-reading of this thesis.

Contents

Chapter 1

About synchronization

You may delay, but time will not.

- Benjamin Franklin

1.1 Background

Because projectors and screens have a limited resolution, achieving a specific higher resolution requires combining several screens or projectors. This leads to a need to synchronize the graphics-rendering of an arbitrary number of computers, connected to these screens or projectors. The work in this thesis will ultimately be used to synchronize the rendering in a platform for graphics visualisation written in C++, OpenGL3 and GLSL. According to the specification, specific events also needs to be able to be sent to the clients, and the clients should perform specific animations on recieving these events. This thesis will explore the possibility of synchronizing rendering in python, using pygame for animations.

1.1.1 When is the rendering synchronized?

Research has been made into the maximum amount of delay allowed(for the human eye) between Audio and Video[1], as this is important in the broadcasting of television. No research into what could be "acceptable" asychronization for the human eye in video to video synchronization will be made in this thesis, it will be assumed that it is a soft Real Time system and that the delay between the video should be as short as possible, an best effort system. For the sake of clarity we discuss delays in milliseconds.

1.1.2 What needs to be synchronized?

Because the clients will be distributed on different computers it cannot be assumed that the clocks on these computers are in sync with each other, this needs to be solved, the clients needs to have the same perception of time. The time to start rendering a specific animation needs to be abla to vary so that the clients can adapt the time to start this animation to their own latency (the slow clients will set the pace for the faster clients). If a client is to slow this client will instead skip forward in the animation (frameskipping).

1.2 Synchronizing clocks over an network

The problem of synchronizing computer clocks over an network has been investigated since the early days of networks. There are a few well known algoritms.

1.2.1 The NTP protocol algorithm

The NTP protocol was originally devlopeded in 1985 by David L.Mills It is used to synchronize clocks over an network by calculating the master offset using the round trip delay.

$$\delta = (\mathrm{t3} - \mathrm{t0}) - (\mathrm{t2} - \mathrm{t1})$$

Figure 1.1: Calculating the round trip delay

[1] http://tech.ebu.ch/docs/r/r037.pdf, EBU

t0 is the time of the request packet transmission

t1 is the time of the request packet reception

t2 is the time of the response packet transmission

t3 is the time of the response packet reception.

Figure 1.2: Variables for calculating round-trip delay time

$$\delta = \frac{(\mathrm{t3} - \mathrm{t0}) - (\mathrm{t2} - \mathrm{t1})}{2}$$

Figure 1.3: Calculating the master offset

NTP is one of the oldest protocols still in use, thus the algorithm take into account the processing between reception and transmisson of a package. This difference would be significant in computers in the 80:s and 90:s but on the computer tested on this will be sometime around 1 millisecond.

1.2.2 The Berkley algorithm

The Berkley algorithm was written by Gusella and Zatti in 1989.

1. A master polls slaves, the slaves replies with their time.
2. The master uses the round-trip time of the messages to estimate the time of each slave and the masters own time.
3. The master calculates an average of the clock times, ignoring extreme values.
4. The master sends the slaves their delta, which can be positive or negative.

Figure 1.4: Simplification of the Berkley algorithm

The delta values that the slaves recieves can be used to create an local time

1.2.3 Cristians algorithm

Cristians algoritm, developed in 1989 by Flaviu Cristian, is used for synchronizing clocks by calculating the roud trip time.

1.3 Problem definition

- Can we synchronize the rendering on an arbitrary number of computers?
  - How can we solve the messaging of synchronization-events?
  - When and how often do we need to synchronize?
  - What techniques are available?
  - Which way gives the tightest synchronization?
  - Which way is the most efficient?
  - How can this be optimized?

Chapter 2

The Demo

This section aims to explain the overall architecture and choices made for the demo, specific solutions for synchronization will be presented in the next chapter.

All code is available at github: https://github.com/Rymdsnigel/thesis-demo

2.1 Running the programs

The demo program is written in python 2.7.2. It requires pygame 1.9.1, simplejson 2.1.6, docopt 0.6.1 and gevent 0.13.0.

2.2 Overall architecture

The demo program consists of a server that continously accept connecting clients.

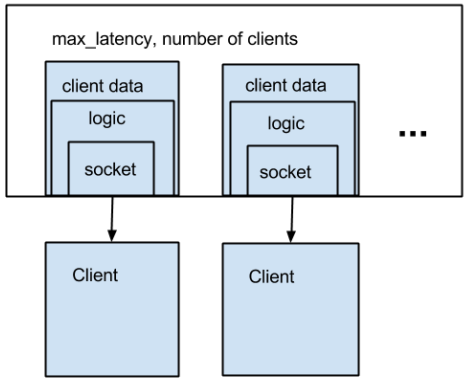

Figure 2.1: Overall architecture of the system.

2.2.1 Why the Client-Server approach?

Because there was a need to both generate and distribute specific events the client-server approach were chosen, rather than choosing a master among the clients, which might have been a more flexible choice if the generation of events had not been in the specification. The downside of this is that if the server should crash, the clients will loose their connection, the connection will not be reestablished by restarting the server.

2.3 Design choises

2.3.1 Communication

2.3.1.1 Protocols

The choice of TCP-sockets as opposed to UDP-sockets was made at an early stage. Although UDP is typically the ideal choice for time critical applications, it was belived that customizing the needed controlmechanisms would take to much time. This choise presented itself as an issue when it was discovered that TCP:s message buffering, using Nagels algoritm, generated a general delay of 20 ms both from the server as from the client. This made the calculated latancy for the client 20 ms longer than it needed to be. It also ment that the more than one json object could be put on the queue of recieved events, which lead to a json-parsing error when trying to load the objects. These errors where removed by disabling Nagels algoritm.

```
self.s.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
```

Figure 2.2: Disabling Nagels algorithm on a socket

2.3.1.2 Sockets

The demo server communicates with the clients using gevent sockets. The recieving needed to be threaded to not block the other processes, and for this gevent greenlets was used

2.3.1.3 Replacing the networklayer

Functional cohesion has been strived for so that the transport-part of the code could easily be replaced by for example an implementation using UDP or implementation of a ready solution such as redis. Though this has not been entirely accomplished

2.3.1.4 Emulating network delays

In the beginning of development the emulated network-delays where inserted using an gevent.sleep for a variable time in the code, this variable could then be specified from command line. This was then replaced by using netem (http://www.linuxfoundation.org/ a more flexible solution. With netem, delays can be bound to specific ports. This required the clients to be bound to these ports, the port to bind the client to must be specified when starting the client, using the –port flag.

```
[]
tc qdisc add dev lo handle 1: root htb

tc class add dev lo parent 1: classid 1:1 htb rate 1000Mbps

tc class add dev lo parent 1:1 classid 1:11 htb rate 100Mbps
tc class add dev lo parent 1:1 classid 1:12 htb rate 100Mbps
tc class add dev lo parent 1:1 classid 1:13 htb rate 100Mbps

tc qdisc add dev lo parent 1:11 handle 10: netem delay 40ms
tc qdisc add dev lo parent 1:12 handle 20: netem delay 20ms
tc qdisc add dev lo parent 1:13 handle 30: netem delay 0ms

tc filter add dev lo protocol ip prio 1 u32 match ip dport \\ 10001 0xffff flowid 1:1
tc filter add dev lo protocol ip prio 1 u32 match ip dport 10002 0xffff flowid 1:12
tc filter add dev lo protocol ip prio 1 u32 match ip dport 10003 0xffff flowid 1:13

tc filter add dev lo protocol ip prio 1 u32 match ip sport 10001 0xffff flowid 1:11
tc filter add dev lo protocol ip prio 1 u32 match ip sport 10002 0xffff flowid 1:12
tc filter add dev lo protocol ip prio 1 u32 match ip sport 10003 0xffff flowid 1:13
```

Figure 2.3: Setting delays on port 10001, 10002 and 10003

```
tc qdisc del dev lo root
```

Figure 2.4: Removing delays set on dev

2.3.2 Threading

Both the server and the client has to achieve concurrency. This is achieved by letting both the TransportServers and the TransportClients inherit from gevent Greenlets Greenlets are in fact pseudothreads that share the same OS-thread so

to release the control the threads must sleep on time critical operations to not block all other threads.

2.3.3 Messaging

The server and clients sends and recieves json, the json is created using simplejson-library functions (dumps() and loads()), creating json from dicts. The functions for generating the dicts that will be the messages are specified in event.py.

The choice of json for messaging, instead of using pickle or cpickle to read and write messages was made due to jsons speed of reading and writing [1] and to support future flexibility in language since pickle and cpickle is python-specific.

2.3.4 Pygame

Pygame, which is built on SDL(ref:http://www.pygame.org/wiki/about) was choosen for graphics due to its simplicity to work with.

[1] http://kovshenin.com/2010/pickle-vs-json-which-is-faster/, Kovshenin

Chapter 3

Synchronizing the clients

3.1 Implementation of the NTP Protocol

Describing how we implemented the NTP protocol for syncing the clocks, and why we choose NTP for this.

3.2 Client latency

3.2.1 Setting latency of client

The latency of each client is calculated on the server and sent to the client in an latency_update_event, along with the maximum latecy. The maximum latency is the latency of the client with the greatest latency. The client then calculates the latency it should apply by subtracting its own latency from the maximum latency.

$$\text{applied\_latency} = \text{maximum\_latency} - \text{latency}$$

Figure 3.1: Calculating the applied latency

3.2.2 Delaying animation start based on latency

Before placing a new render event on the queue the client sleeps, using a greenlet.sleep, for the number of milliseconds specified by its applied latency.

3.2.3 Skipping frames if delay is to long

If the client latency is to long we should skip ahead instead.

Chapter 4

Conclusion

The goal of this thesis was to implent some kind of algoritm for synchronizing rendering on different computers over an network.

Some important problems faced in development:

1. Making sure that the client and the server agreed on a timestamp
2.
3.

Outlook – suggested improvements

The demo is just an proof of concept.

Bibliography

[1] Nicolas T. Courtois. The dark side of security by obscurity and cloning mifare classic rail and building passes anywhere, anytime. Cryptology ePrint Archive, Report 2009/137, 2009. http://eprint.iacr.org/.

[2] Flavio D. Garcia, Gerhard Koning Gans, Ruben Muijrers, Peter Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling mifare classic. In *ESORICS '08: Proceedings of the 13th European Symposium on Research in Computer Security*, pages 97–114, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Wirelessly pickpocketing a mifare classic card. In *SP '09: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, pages 3–15, Washington, DC, USA, 2009. IEEE Computer Society.

[4] M. Hellman. A cryptanalytic time-memory trade-off. *Information Theory, IEEE Transactions on*, 26(4):401 – 406, jul 1980.

[5] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology - CRYPTO 2003*, pages 617–630, 2003.

[6] David L. Mills NTP Architecture, Protocol and Algorithms http://www.eecis.udel.edu/ mills/database/brief/arch/arch.pdf [Online; accessed 7th April 2013].