Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Study and implementation of a real time online football game for mobile devices

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Marc Fernández Vanaclocha

DIRECTOR: José Yúfera Gómez

DATE: 02-08-2012

Title: Study and implementation of a real time online football game for mobile devices

Author: Marc Fernández Vanaclocha

Director: José Yúfera Gómez

Date: 02-08-2012

Overview

The main goal of this project is extracting an optimized system to do a sensible application to delay loses and jitter in a wireless environment.

This system will be used in a football game for mobile devices with maximum two players per game.

To all those who gave me the opportunity to do it.

Master Thesis

INDEX

INTRODUCTION

This memory reflects the theory and the work made to do a commercial application for mobile devices.

The goal is to offer a 2 players football online game with 3d graphics and physics in a mobile OS (Operating System) like iOS and Android; in the process, it will be necessary to resolve the network problem of how to manage the data to give a good service to the final user in mobile networks.

With this problematic in mind, there is an analysis of how to synchronize, avoid delay, jitter, loses, resolve the matching players and take empirical statistics and verify the results to evaluate the system.

At the end, the result will be a fun football game for mobile devices that will rock the market.

The document is divided into:

-Problems to solve: an enumeration of all target problems to solve.

-Game Description: a description of the game to introduce.

-Transport layer analysis: a basic vision of UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) and the role of each one in the result.

-Types of architecture: a basic view of three basic architectures and the selected one.

-NAT transversal techniques: a description of NAT (Network address translation), the problem produced because of it and a description of how to avoid this problem.

-Synchronization techniques: a description of why it is needed synchronization and a view of different techniques with advantages and drawbacks.

-Own network system: a complete description of the resulting work.

- Statistics: description of the test, the results and an evaluation.

-Conclusions: at last, a global view of all the work and some critics.

CHAPTER 1. PROBLEMS TO RESOLVE

The base of this project is a java 2D offline game football and the result, a 3D game based on C/C++ with online mode.

The points to resolve in this project are:

- Translate the logic and remake the structure if it is needed.

- Make a new 3D system.

- Resolve input system for different SO.

- Make a new GUI (Graphic User Interface).

- Resolve audio system.

- Analyze and select the online architecture.

- Analyze the game data, the transport layer options, select one option and give reasons.

- Describe how to resolve the NAT transversal techniques to be implemented with the transport layer selected and the architecture.

- Analyze how to synchronize the application and what the acceptable clock resolution is.

- Analyze what signaling is needed and how to resolve it.

- Describe the protocol used.

- Describe the application life flow for online system.

- Analyze the main security problems and what options we can take to resolve it.

- Analyze the application and extract statistics to make conclusions.

At the end, this document reflects all network points.

CHAPTER 2. GAME DESCRIPTION

It is a single and multiplayer soccer game; where there are in screen 11 players for team, 3 referees and 1 ball.

The game logic is based on a java game for Android called Striker Soccer, simple 2D game with some physics and tactic control.

The interfaces and controls must be really simple because the target can be kids smashing phones or a simple player that wants to play right now.

Illustration 1 - In game screen shot

CHAPTER 3. TRANSPORT LAYER ANALYSIS

This section will analyze the two main options that can access developing a mobile application, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

3.1. TCP

There are too many TCP versions.

To explain how it works we can assume that mobile devices work with TCP Westwood [12] version that takes into account the random loses from the radio environment.

TCP offers a congestion control [11], assures that the data stream will be read in the original sort, ensures retransmissions if there are any loses and is connection oriented with only 20 bytes overhead.

Congestion control can be defined as reactive or predictive: reactive is based on doing a reduction of the sender window when a RTO expires; this means that TCP is reducing its data flow when the network has congestion and in consequence, it needs time to recover the normal function and maybe will be congested again.

Predictive are more complicated but they are based on the increasing delay of router queues. If a RTT increase is detected, TCP will decrease his window; in the other hand, if RTT decreases the window will increase.

With this technique the network has less loses by queue overflow and has more stable throughput.

In wireless connections, those using TCP Westwood inclusive, the channel can present many loses and this will decrease the efficiency of throughput and increase the delay in some data. This data, that arrives really late, can be useless in case of a real time application.

3.2. UDP

This transport protocol only has 8 bytes of overhead and does not ensure anything.

UDP can be useful when a periodic emission is needed and the service can manage loses like video or voice over IP.

If the network is congested, it will continue congested but other TCP connections will reduce their flows because of its congestion control.

In wireless access, UDP can be an excellent option because the channel is shared along all users and if your application congests the link, the other TCP connections will decrease their flows and liberate resources.

To end, because the game has an interactive level with a hard dead line, messages that come later are useless.

All critical data will be sent using UDP datagram and data sensible to loses, like signaling using TCP to ensure that they arrive to the destination.

CHAPTER 4. TYPES OF ARCHITECTURE

The main goal of this chapter is to analyze and select the best architecture for the project.

There are 3 basic types to choose: pure peer to peer, client server structure and hybrid architecture.

4.1. P2P(peer to peer)

This architecture, based on the peer to peer one, does not have any infrastructure that supports it. See figure 1.

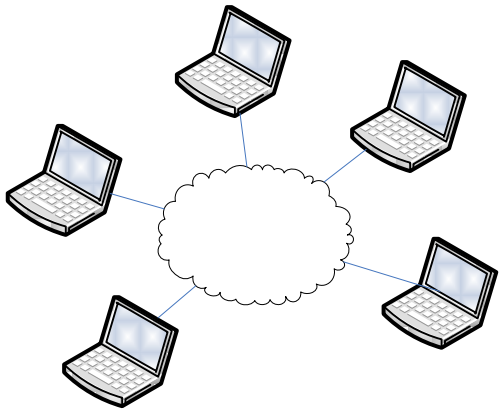

Figure 1 - Peer to peer architecture

There are some hybrid solutions to solve the problems but the result is a network overlay to do all with high resistance to crashes.

Advantages:

-Robust to crash: there is not any essential computer that holds all the service, then, if any peer closes the application the service will run (if there are other peers in the network).

-Low delay (once connected): it is a point to point connection. For applications with high interactivity like videogames this is basic; other structures (not point to point connections) would be really hard to develop correctly and have a good commercial result.

Drawbacks:

-Search: without any server that holds information to start a connection and search other peers, the solutions are given by an IP multicast direction or using broadcast communication to search peers. Multicast direction is not a cheap or possible solution and broadcast messages are limited to LANs.

-Volatile information: the data held in the P2P networks can disappear quickly. If the service wants to hold some data like the football match results or/and associate them to an account, this drawback is really hard.

-Cheating access: all logic is inside the peers, then, if there is any advanced user, it can extract important data from the software and network and make their own programs to cheat.

4.2. Client-server

Client-Server architecture means that each client is connected to a server and this one gives all data to the client; each action from the client will pass through the server.

An example of client server architecture is shown on figure 2.

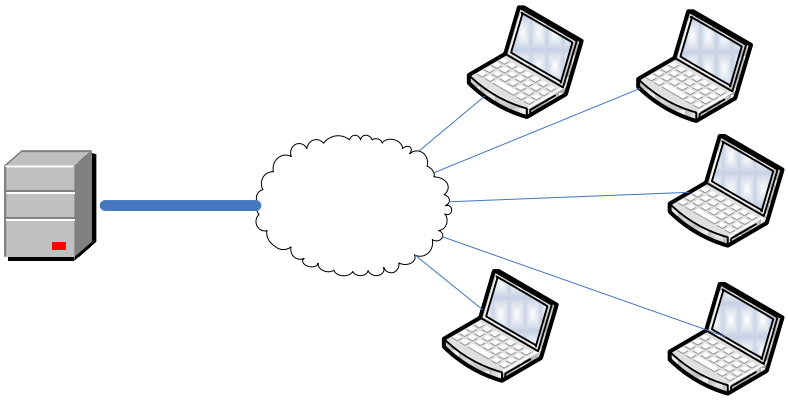

Figure 2 - Client server architecture

Typical problems are that the whole service is held by one side only and can be target of attacks.

Advantages:

-Control of data: this architecture can control all data from the clients. In a game scenario, this data can be the match results from each account to make some rankings and print it in a web page.

-Control of statistics: all communication between clients pass into the server. Then, the server can measure delay, cadency, jitter, charge of the system, hours played, places with most clicks to put a banner for example and all the information that can be extracted from all movements done by the users, or detect if any of the servers is not enough to the number of users.

-Anti cheating: if the logic is inside the server, it cannot be manipulated without an attack and the service can assure a quality of service detecting users that try to cheat and ban it.

Drawbacks:

-Maximum delay: in this scenario the data travels to the server and it sends the new events to the other clients. In the best case this delay can be approximated to the point to point connection if the ISP (Internet Service Provider) does not supply variations given by congestion, losses, access control or multipath.

-Maintain the servers: this point is expensive because any crash in the server side implies a downtime of the service and, in consequence, losing money and getting frustrated users.

-Crash or inaccessible servers: this can occur because the server cannot manage more clients (this architecture has lot of load), because of being the target of a DoS attack (Denny of Service) or because of sudden problems like the NIC (Network Interface Card) being broken.

4.3. Hybrid

This architecture mixes P2P and client server architectures with more or less influence of each type (figure 3 tries to show all components of this architecture).

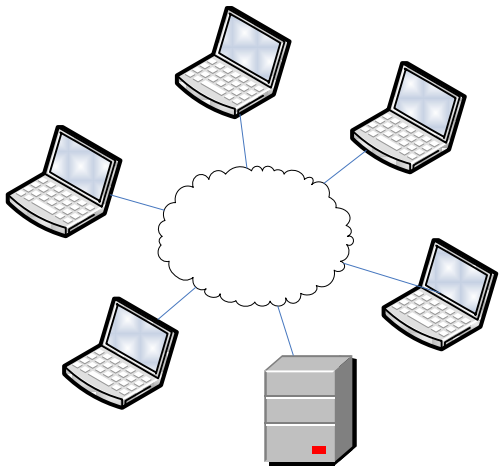

Figure 3 - Hybrid architecture

In result this network architecture ensures service and has low load at server side.

Advantages:

-Lower delay: the connections from peers are point to point; this means that there is not any middle point that adds delay.

-Easy search: a big problem of a pure P2P is to search the first peer to enter to the network overlay. This problem is solved because the server knows some or all peers connected and if any problem occurs like a crash of these, the server can become a fulcrum to the peers to rejoin to the overlay network.

Drawbacks:

-Cheating access: same that P2P architecture but, the peers can report any incongruence of the other peers. This report has to be managed carefully to do not become an attack spot to the server.

-Susceptible to crash: in case of server crash, the current served peers will continue with their own work but new petitions to the server will not be served.

-Susceptible to fake data: if at the end of any activity the peers upload some data to associate it to their own account, the server can accept some fake data like a match lost by 4-0 becoming a favourable 4-0 at their own account.

-Maintenance cost: like client server but the server does not have a big load, then the maintenance is cheaper or for the same machine can serve more users or other functionalities.

4.4. Conclusions

Some drawbacks can be avoided with solutions like using more servers, comparing results from peers and comparing with expected results, or doing periodic actualizations to avoid the appeal of cheating programs of easy use for everyone, among others.

These architectures have their own targets. In the scenario of an online game (application with critical interactivity) the best solution might be a hybrid solution if the match is for a few players, or a client server architecture if it is pretended that game has more than N players in the same match (number of players depend on target device, expected throughput for each peer and expected problems in network to chose hybrid or client server).

For a two player game, like a soccer game, hybrid architecture will maximize the player experience because the reduction of delay. This option is the selected one to design the online system taking into account that actual mobile devices have enough process capacity.

CHAPTER 5. NAT TRANSVERSAL TECHNIQUES

Some problems in the current networks are the large amount of machines connected and the limited IP directions. This is a problem solved with IPv6 but it is not used globally. For this reason and to reduce the use of public IP address the NATs (Network Address Translator) were created.

NAT basically uses a single public IP for a private IP range, which saves a lot of directions but increases the number of problems to do a direct connection between two machines that are inside other NATs.

There are 3 IP ranges defined with IP and subnetting that become for private use: 10.0.0.0/8, 172.16.0.0/12 and 192.168/16, defined at RFC 1918 [8].

These ranges are used inside the private network behind the NAT and all messages with private address are translated to the same public address.

All this is a good solution but adds a problem for connections between users inside private networks and the applications affected are the ones that try a peer to peer connection.

The main problem is, without any previous configuration to NAT, that translating incoming IP packets correctly to the private address becomes impossible.

To do a successful communication there are some tricks that normally include a mid point with a public IP address to be accessible from every place.

The tricks used depend of the NAT type. Once it is learned how it works, the process to cross it can be found easily.

There is not any NAT transversal technique without using an accessible midpoint and without doing any interaction with NAT or trying a port mapping.

There are some projects like UPnP but it is not needed to activate this option in the house routers, since it could become a big disadvantage for mobile users that do not want to know anything about technical procedures.

5.1. Types of NAT

There are 4 types of NAT defined at RFC 3489 [9] (definitions extracted from RFC):

> *Full Cone*: A full cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address

and port. Furthermore, any external host can send a packet to the internal host, by sending a packet to the mapped external address.

Restricted Cone: A restricted cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Unlike a full cone NAT, an external host (with IP address X) can send a packet to the internal host only if the internal host had previously sent a packet to IP address X.

Port Restricted Cone: A port restricted cone NAT is like a restricted cone NAT, but the restriction includes port numbers. Specifically, an external host can send a packet, with source IP address X and source port P, to the internal host only if the internal host had previously sent a packet to IP address X and port P.

Symmetric: A symmetric NAT is one where all requests from the same internal IP address and port, to a specific destination IP address and port, are mapped to the same external IP address and port. If the same host sends a packet with the same source address and port, but to a different destination, a different mapping is used. Furthermore, only the external host that receives a packet can send a UDP packet back to the internal host.

With these definitions, all cone type maintains a mapping if the internal host sends a previous packet.

This can be a problem for unidirectional messages but an online game has bidirectional communication and the mapping at NAT can be created from the two sides.

At last, symmetric NAT can increase problems. However, the patterns that use the NAT to change the output ports can be extracted with more logic and multiple public midpoints, not without a painful development.

5.2. UDP and TCP hole punching

Described at [2], this technique describes how to connect two peers with the four types of NAT using TCP and UDP and one or two midpoints.

The typical scenario will be two users behind one NAT, not with multiple NAT.

The worst case scenario will be a port restricted cone NAT.

The scheme of UDP hole punching technique in Figure 4:

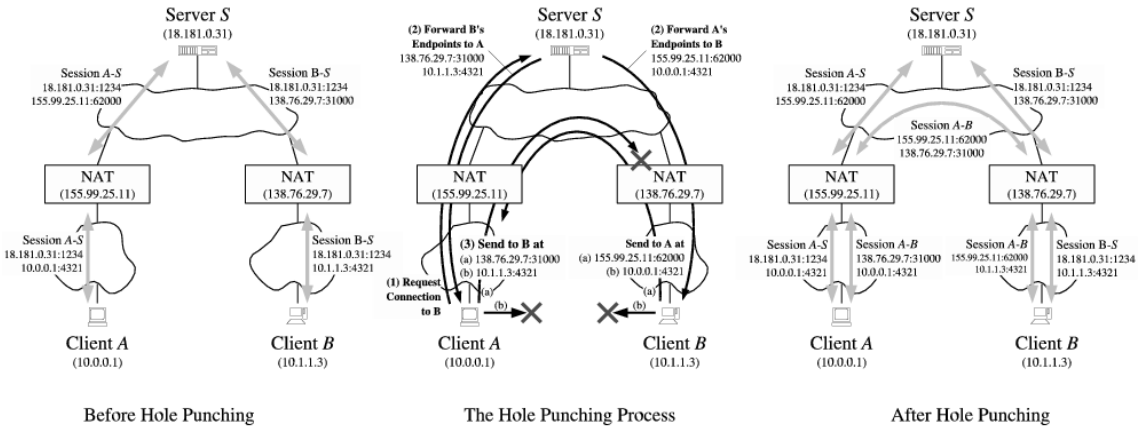

Figure 4 – UDP Hole punching

Suppose two clients A and B behind their own different NATs. A and B have initiated their own bidirectional communications with server S.

Each NAT has assigned to privatePort-A and privateIP-A a publicIP-A and publicPort-A. The same for B for the UDP channel. These public values are known by S.

A and B want to communicate between them, then S will pass the public values from the other to each client.

If the two peers start to send data to the other public IP and port, the mapping will be correct in any cone NAT and it will be the same with the server. At last, some datagrams will cross creating the complete connection.

This system requires that a peer does not change of port origin. With UDP this is not a problem, but using TCP there are some complications because not all OS support it.

In this case, there is a sequential method that needs a little schedule to sort out who listens and who tries the connection.

In order to do multiple TCP sockets with the same values and do the same like UDP, though listening and trying connections, the SO needs to accept the SO_REUSEADDR flag.

The scheme of the needed sockets to do the same as UDP is represented at figure 5.

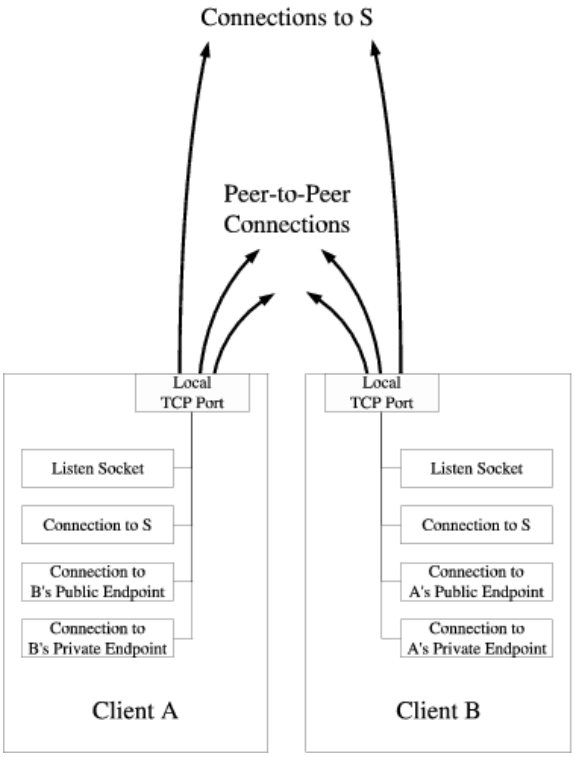

Figure 5 - TCP Socket scheme

The sequential scheme needs to reuse the socket closing connections with the server (another socket can be opened to manage new incoming messages), decide who will listen and whom to send. Although the first try will crash, it will do a correct mapping on one side.

Then the server schedules another connection, though, in the other way, the mapping is completed and the TCP connection is created.

5.3. Stun (Simple Transversal UDP over NATs) technique

Defined with RFC [10], the technique is basically the same as hole punching but with standards to communicate with everyone. This case is used with VoIP (Voice over IP) to create a channel to use SIP (Session Initiation Protocol) and SDP (Session Description Protocol) but with an online game there is no need to implement all the requirements.

5.4. Conclusion

A mid implementation of STUN is useless since this one only ensures UDP communication with cone NATs.

Because mobile devices have their own problems, using an open source implementation can be a bad solution if this one does not work well or using different options such as threads or blocking sockets.

From other side, the hole punching technique describes how to cross a NAT to make an UDP and TCP connection between peers using a midpoint. Since it is not needed to follow any standard, this technique will be used to solve NAT transversal in this project.

CHAPTER 6. SYNCHRONIZATION TECHNIQUES

A basic thing of online real time games is the synchronization of each player [2][3][4][5].

What seems a good game when each user sees something different becomes a bad game experience, for example, in shooters that have critical interactivity. If a player shoots another one and kills him but in other computer the latter is still alive and kills the killer, there is an incongruence. It causes angry and confused players that become less money for the company.

There are some techniques to ensure the interactivity and coherence between users.

These concepts are:

-Interactivity: it is based on time. It is better when less time is spent on visualizing the actions of the user.

This implies that all tasks needed to the normal game function do not have to overpass interactivity time limit, about 150 – 200 ms in games.

-Coherence: it is the other aim of synchronization techniques, none of the players have to perceive any difference between screens, and this can be obtained using the same clock and reproducing the state in the same time.

Synchronization systems are evaluated with human perception and then there are fewer restrictions in the implementation.

There are two kinds of approaches: optimistic and conservative implementations.

Conservative approaches are based on retaining the maximum coherence possible. A simple implementation is the lock step technique. This technique is based on not advancing the game state until all players acknowledge that they have processed it; the global experience is the worst possible for all users, so this approach only works in good such as playing in the same LAN.

Another conservative system is the time-bucket synchronization, also known as local lag. It is based on queuing the entrance events and processing them at the same time. This gives time to reorder events, ensure coherence between players and put a limit on interaction. This one works for a maximum delay. If one user surpasses it, this user will work doing constant death reckoning because all events come later.

The time warp mechanism is an optimistic approach. It is based on detecting incoherence in the game state and in this case, correcting it with a roll back to a

coherent game state. This roll backs need to be really infrequent because they create a disappointing user experience losing all actions previously done.

This system is used for large scale multiplayer games and normally with client server architecture with multiple servers that hold the game state. In this case can maximize the interactivity.

At the end, for the game system, time-bucket synchronization can be easy to implement and with good results for a hybrid architecture.

CHAPTER 7. OWN NETWORK SYSTEM

The network method designed in this master thesis is a mixture of some of the techniques described before. It is a good result but only for our aim because the game is player restricted (in this case 2 players, though the system can work for more but not massive players).

The design is based on 3 columns:

1- Limited number of players: the mobile phone goes really short with 3D graphics, networking, AI (Artificial Intelligence) and managing inputs; then it becomes nonsense to design an online system to hold more than 4 or 8 players at the same time.
2- Maintain the normal speed of the game without incoherence.
3- The game has to be delay resistant; it has to be playable with delays up to 150ms and high loss like half second without communication because the radio links are not the best scenario.

7.1. Connection layer

The final architecture is a hybrid peer to peer with a tracker, and clients organized with master and slave role (master holds all logic and slave is a simple client) once connected, as explained in previous chapters.

This architecture reduces the tracker load and delays between peers, though it might be the target of cheats easily.

For example, 500 concurrent matches with client server architecture, a unique server will manage 1000 TCP connections, 1000 UDP sockets, 500 match logic…

In throughput data, if it is only considered the upload traffic, with an expected datagram size of 350bytes (all data from 25 puppets at screen, ball and time stamp), the load at server will be like the figure 6:

$$Throughput_{user} = (expected\ datagram\ size + UDP\ and\ IP\ overhead) * events/second$$

$$Throughput_{user} = (350bytes + 28\ bytes) * 30events/s = 11340bytes/s = 90'72Kb/s$$

$$Throughput_{500\ matches} = 90'72Kb/s\ user * 2\ user/match * 500\ match = 90'72Mb/s$$

Figure 6 – Throughput in case of client server architecture at server side

These values can be reduced accepting more loss effect, reducing events per second.

A normal value is 15 events / s reducing it to half but 45Mb/s is a big value if there is not a big infrastructure at back.

With a tracker this load does not exist. With the first implementations the tracker will hold several TCP connections with low activity. The throughput charge resides on clients.

Following, the way to start a new match is explained: a player connects to tracker, he/she waits for another player, when there are two players, starts a connection with the other one (between them) and starts: they do the signal to solve all game prerequisites, play the match and the game ends and uploads statistics to tracker.

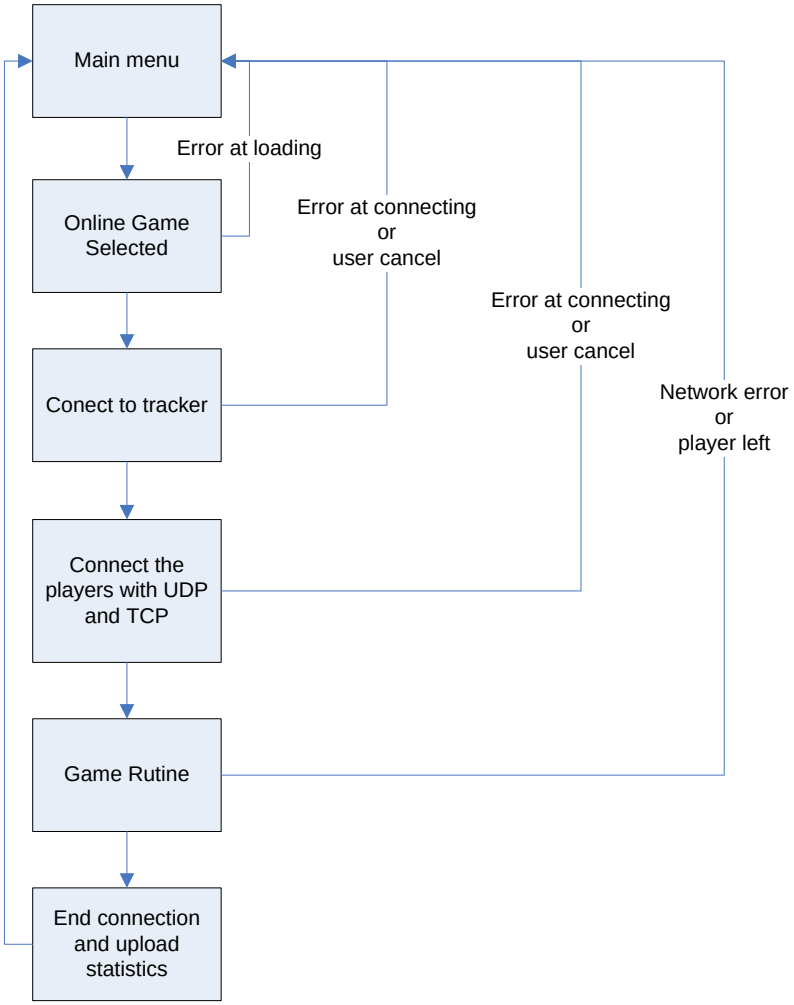

Figure 7 - Basic network game states

The connection between peers is made using TCP and UDP hole punching.

The TCP channel is for signaling, and the UDP channel is for synchronising and sending game events (the events that can be lost).

If there is any problem with the connection, an error will pop up and the user will be noticed.

Because the peers can be located inside a private network, the reconnect option is not contemplated right now.

7.2. Signaling layer

Signaling is the logic behind the online game and it is necessary to choose the roles (master or slave), negotiate with tracker, know certain values to use algorithms like the hole punching technique, or advertise important events to slave like half time or end time.

There are some standards like SIP using SDP (with their own RFCs) or H.323 defined by ITU. However, using those protocols to made the required signalling is a waste of resources.

Signalling data is a really critic data. Because of that and to ensure no losses the transport layer selected is TCP for all messages. However, it is not important if it is received 500 ms later.

In this case, signalling can be divided into 3 points: tracker, master and slave.

Tracker

Its function is to match players and be a NAT traversal server; it will take statistics from players, search for cheats, and manage users by authenticating, searching for friends and giving the option to create matches.

Once the players are chosen and connected, they need to select their role.

Master

The master will be the one with better hardware or, in case of draw, the one selected by tracker.

Become master implies more CPU usage, better game response and waste of uploading throughput.

Master role will signal each game event and there will be some acknowledges sent from slave to ensure an application keep alive.

Slave

Slave role have less CPU usage than a single player game but it is the target of network problems. If there are any losses or jitters or high delays the user will be affected directly. For this reason all the efforts are focused on ensuring delay and losses resistance.

7.3. Synchronism

In our game there are two types of synchronization: clock and game synchronism.

7.3.1. Clock synchronization

To do something coherent, the same reference is needed, in this case the clock.

Because the result has an empirical evaluation, the clocks does not need to be synchronized at high resolution, with an error of +-5ms the game experience is not affected. To solve this problem a simple UDP system considering symmetric delay is used.

The clock synchronism used, consist on given moments that the game can be hold (start, half part or some faults). Slave sends his clock to master, this one marks with its own clock and sends back to slave; at the end, the slave knows its RTT and marked clock; to use the same clock of master only need to use the formula from figure 8.

$$Master\ clock\ on\ slave = Master\ clock\ mark + RTT/2$$

Figure 8 – How a slave calculates the clock

To ensure some precision there will be 5 measures and in the worst case, this measures will hold the player 2’5 seconds until a game message notifies him/her that synchronism is not available (more than 500ms RTT).

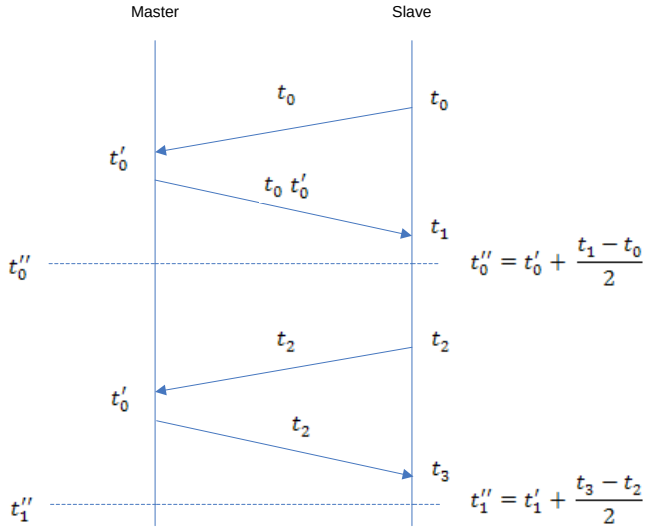

Figure 9 - Clock synchronization scheme

Once synchronized, slave sends the expected delay to master to use it later.

7.3.2. Game synchronization

Once the peers are using the same clock it is time to send events. To ensure a good experience the methods used are the following:

A dead reckoning technique is applied on slave to ensure the perception of interactivity but really simple (using first order equation and taking to account gravity on ball) to safe CPU.

Another measure is adding delay at master. An event is sent out and queued to delay its visualization delay (milliseconds). This reduces the effects of dead reckoning on slave and gives some fairness because both players have the same delay in actions.

At the end, if there is not so much variations at the delay between peers, the slave will do a simple use of dead reckoning with little time intervals.

In the other side, to become more loss resistance, every event is independent from the others because each one has absolute data and not relative, and in consequence there is a major use of throughput.

7.3.3. Master description

The next scheme (figure 10) reflects how the system works for master events:

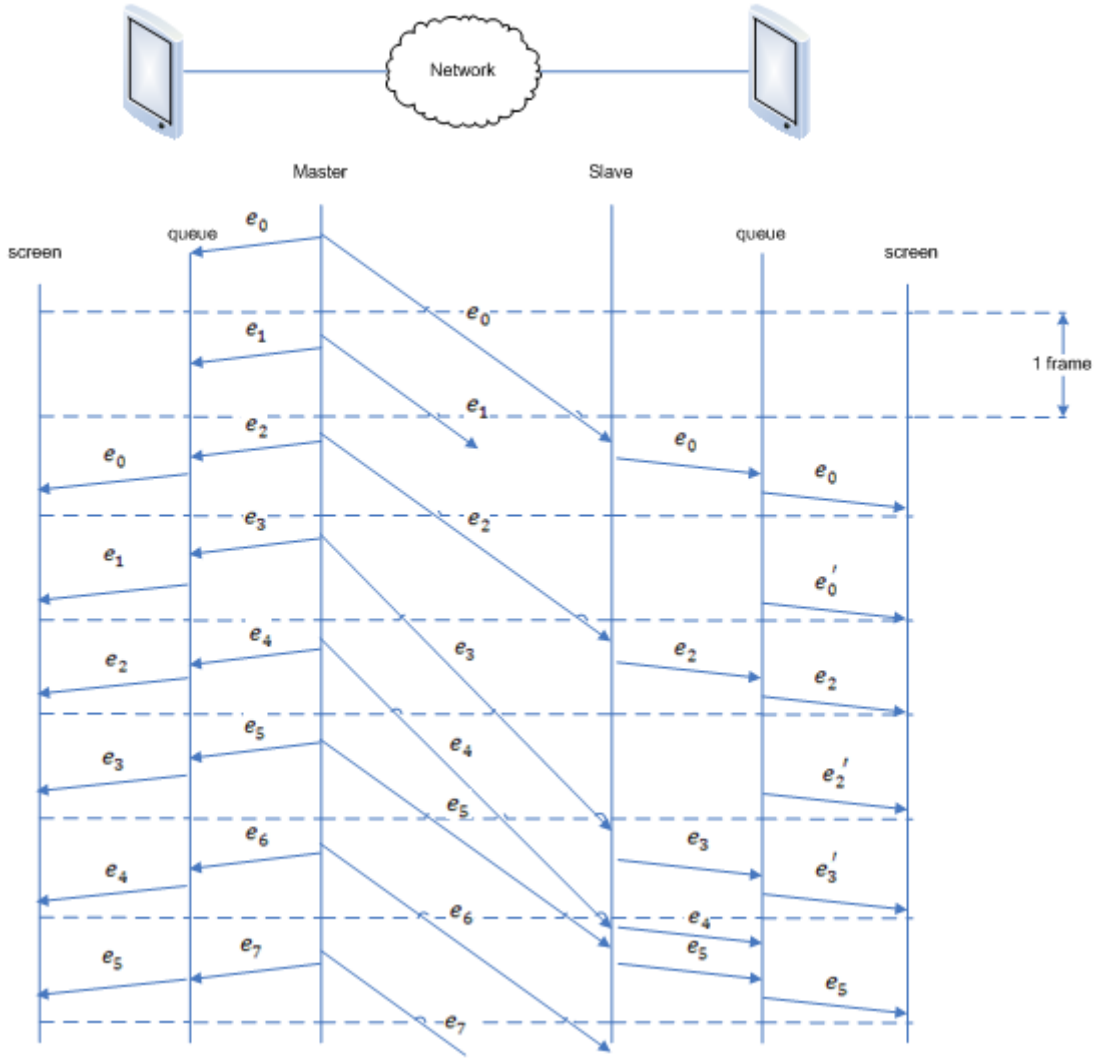

Figure 10 - Online scheme for events generated by master

- **Event 0 is delayed 2 frames** at master queue because delay point to point is near 2 frame seconds.
  Slave when receives event 0. This one has a timestamp that points present or past clock, and in this case, the program will visualize it. In case of the actual time – time stamp ≥ 1frame a dead reckoning will be applied.

- **Event 1 is lost**. There isn't any problem for master but slave will predict his next event taking into account the previous correct event. In that case event 0.

- **Event 3 and 4 suffered some delay variation**, again for master role there isn't any problem.

For slave, to print the event number 3 use a prediction from 2 and for the event number 4 use a prediction of event number 3. Event 4 is discarded because event 5 comes before the game logic look if it is in the queue.

7.3.4. Slave description

Slave sends his control data with the current time stamp + delay ms, and server will process it and correct the possible jitter.

Slave will receive the master event with the actual control at RTT ms; for example the figure 11, a shoot (c0) at t0 will be received at t2 (t0+RTT) processed inside an event. The shoot will be displayed correctly but not from start.

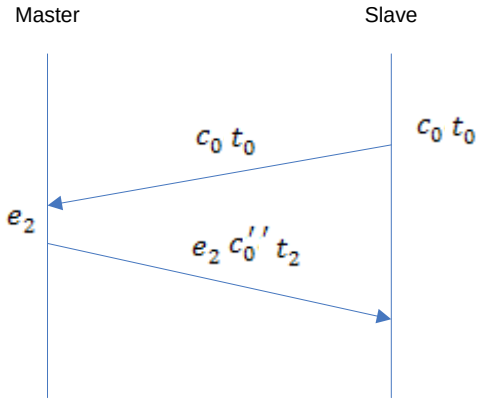

Figure 11 - Slave control message scheme

This scheme reflects that control event 0, that is a state variation (running, shooting, tackle or waiting to other one), will come back with more delay. The user will not appreciate any difference if the delay is less than 150ms. When this happens, it will be the slave who visualizes the transition before the event arrives.

7.4. Complete flow scheme

At this point the entire system is near to be described and, to do a global vision of all the system, there is a flow diagram of all the process followed to make, play and end a match (figure 12).

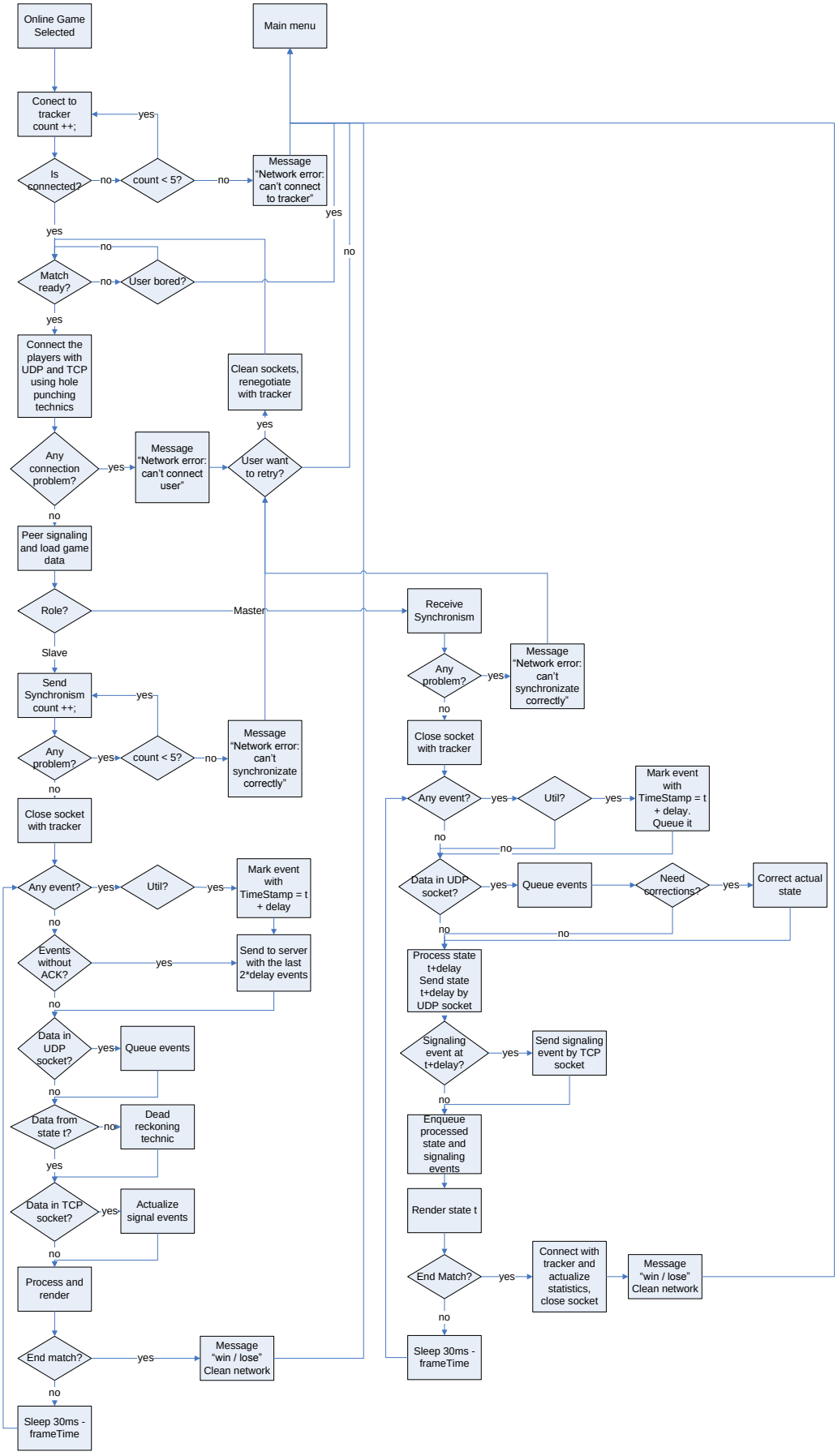

Figure 12 – Complete flow scheme

This scheme can be explained in seven points:

1- Tracker runs, is listening for a known port and known IP address.
2- A peer connects to tracker searching for a match; the connection is maintained until there is another user to start a match (figure 13).

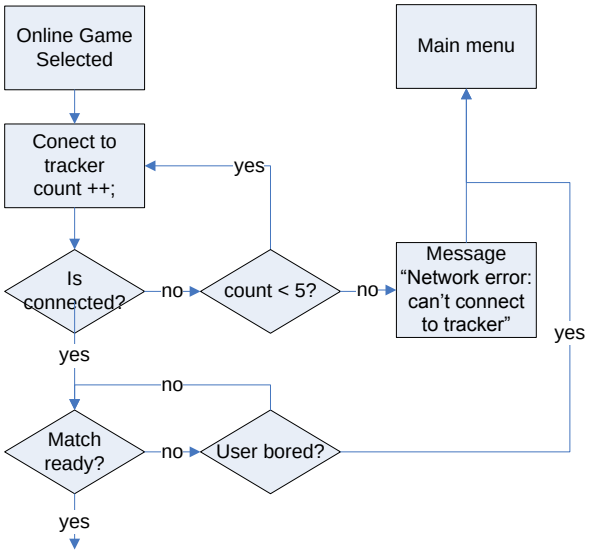

Figure 13 – Point 2 flow scheme

3- A second peer connects to tracker. Now there are 2 peer connected, tracker starts a new match, sends data to peers to start with P2P connection and helps them to connect and chose roles (figure 14).

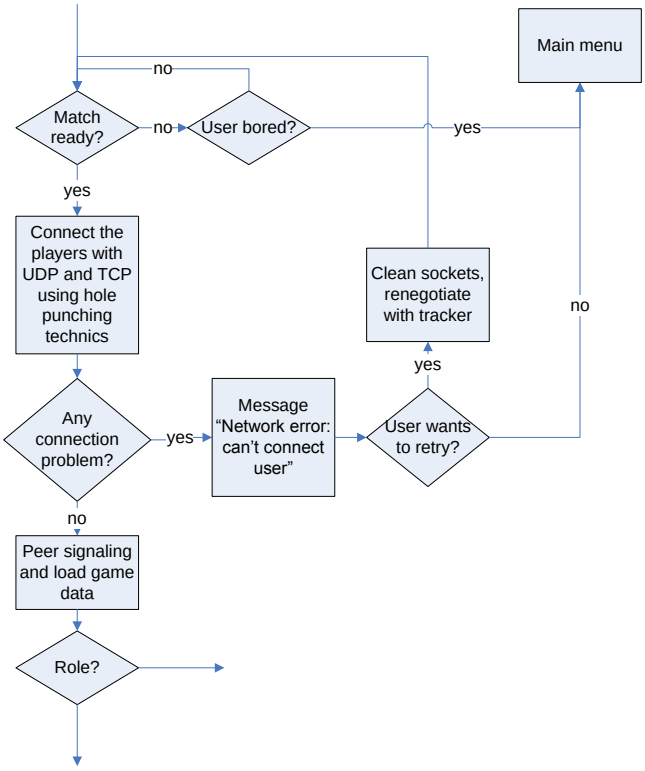

Figure 14 – Point 3 flow scheme

4- Once peers are connected, they do the signalling and clock synchronism (figure 15).

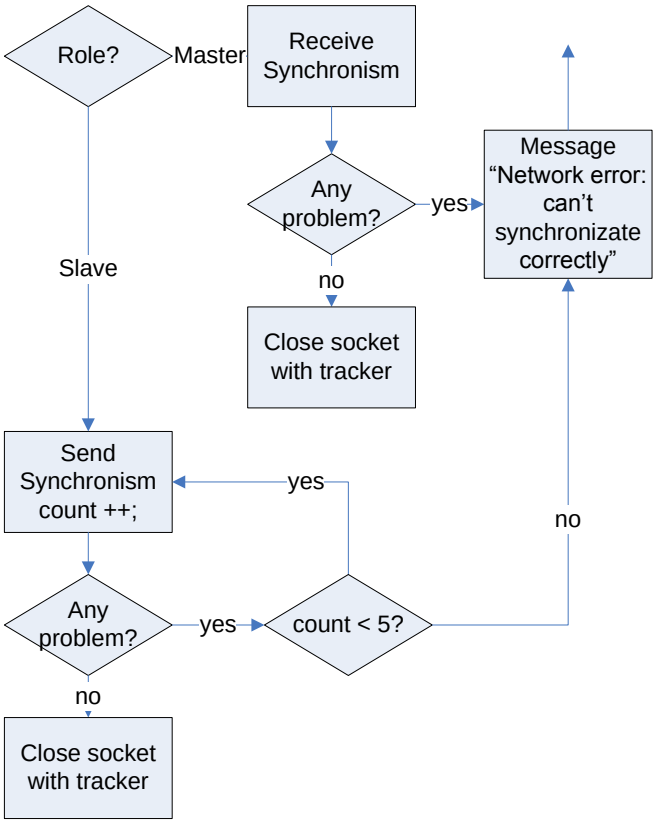

Figure 15 – Point 4 flow scheme

5- If all works fine, the match starts and ends in a few minutes. All signalling events go with the TCP channel to avoid losses (figure 16).

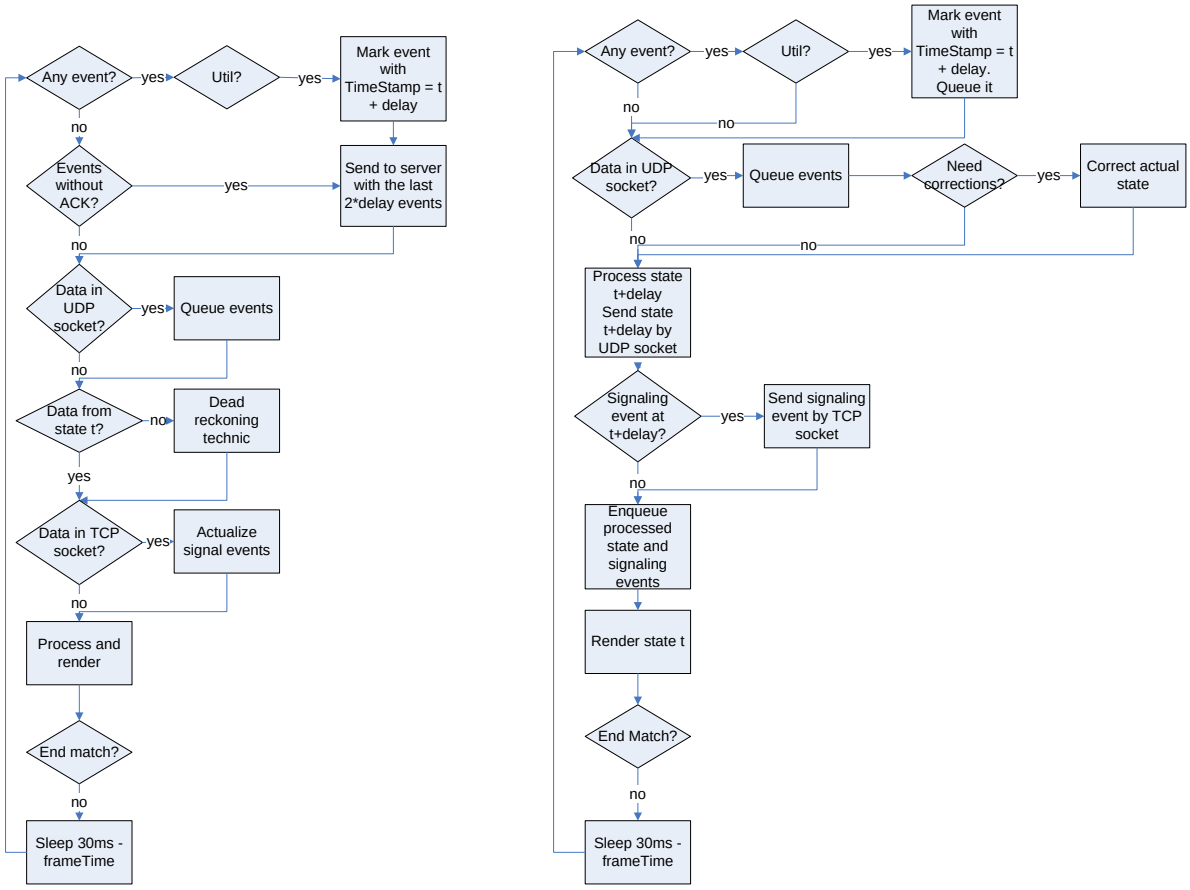

Figure 16 – Point 5 flow scheme (left slave role, right master role)

6- At the end, the tracker will know the result to extract statistics (figure 17).

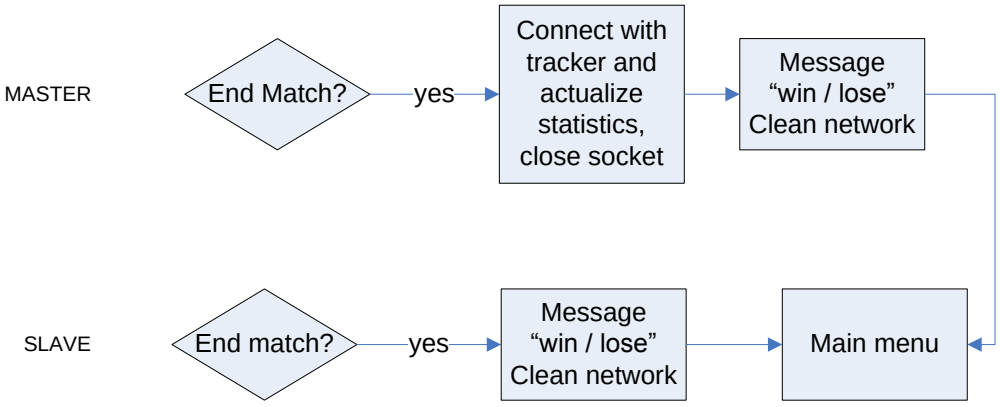

Figure 17 – Point 6 flow scheme

7- If there is any error in network or application, the live peer will notify the tracker. The peers will close connections and print an error message on the screen before any error.

This is useful to use to develop it and into take to account the scheme to search all vulnerabilities. That is the next section.

7.5. Analysis of vulnerabilities and cheating

What are software vulnerabilities? The short answer is all the things that a user (expert or not) can do to change the normal function of software to take an advantage from the normal using. If there is not any advantage on doing this, the options changed by the user can be considered a bug, for example, in a calculator putting 1/0 and crashing it.

To do an analysis of any software and search and get protections is needed to know every part of the code. In addition, using the previous scheme, the analysis will be divided into sections trying to expose how to solve it.

A constant work is needed to actually avoid vulnerabilities because these kinds of projects are in continuous development and giving new functionalities: a simple new formulary to search friends can be a big security hole giving access to users to access to databases, for example.

In a game context some vulnerabilities might be used to cheat and take advantage in front of other players.

System vulnerabilities can be explained in 7 points that represent 7 states to make a complete online match: tracker normal runtime, peer log in, match signalling, peer synchronization, game, game ending and, at last, network errors.

Point 1:

- Tracker application crash. It can happen because:

  Segmentation fault: error because of any null pointer in the program

  Memory leak: it needs a good memory management

  Buffer overflow: it can be located with backtraces

- Tracker suffers DoS (Denial of Service):

  Add a previous login server to avoid ghost players and to kick inactive players.

  1.1. To avoid attacks on login server and hijacking accounts, the number of login tries can be limited to 5 each 5 min per IP.
  1.2. To avoid man in the middle attacks to extract user and password, the channel will be encrypted using SSL for example.
    1.2.1. This is not enough to avoid all attacks like ARP spoofing.
  1.3. The password sent and stored will be a hash to avoid losing important data in case of any successful attack.

1.4. All data will be parsed searching any special character to avoid SQL injection. This includes all client inputs, formularies or getting/posting parameters

Point 2:

- When peer does five fail logins, the server will refuses all connections until the penalty expires. In case of trying it more times, the tracker will trigger a warning to administrators.
- If peer does not do anything in 1 min, the connection will be reset to save connections.
- Once logged:
    - If there is any signalling fake, the tracker will close the connection and send warnings to administrator.

Point 3:

- Two users are selected by tracker to play; the server generates a symmetric code to encrypt the communication between peers. This code is sent by the secure channel between users and server.
- The tracker saves the match data and puts a time mark to avoid data manipulation out of time.
- Before create a match, it is seen if any one of the players has any match already started or in penalty because he left, in this case, it is communicated to the user who will not be able to start the match.
- In case of data manipulation in the client side, the result will be an error message to the other player and the end of the match.
- To ensure the channel between peers, the symmetric code is generated randomly.

Point 4:

- At this phase the critical values are the clock synchronism values. For example, in case of master with a freeBSD proxy that adds delay once the game starts, the game for slave player is lost; this attack cannot be avoided, because of the extra delay, so the slave will play without a correct delay in actions and will have inconsistencies. If the player does clock synchronism with a certain frequency the problem can be solved, though mobile devices cannot do these measures without adding delay if the measure does not block the game.
- In case of signalling manipulation, the tracker will apply a penalty to user that depends on the number of manipulations done.

Point 5:

Now the game is running.

- A modification of game results might happen because master is sending goals to its own team but they are not real. To avoid this, slave will see if any signalling event is coherent with the game state.
- Referring about coherence between game events, again, the master can try to cheat the slave doing a change position favourable to it; in this case, slave will measure if the different distances are coherent with the time between events.

Point 6:

The game is ended.

- Players connect with tracker with his own session id (to avoid log in again). This action might become a real problem since the session will have a caducity of 10 min of inactivity, for instance.
- When uploading their statistics, the tracker will compare the 2 reports to avoid different values. In case of differences the match will have a warning to be revised later. If there is any report of player this one will have more priority to be resolved.
- Again, all the data is referred to a match identifier. If this match does not exist or it is over, a warning will be made to erase cheaters.

Point 7:

Network errors.

- If a disconnection happens, the master will only accept reconnections of the same slaves. In order to do it, the slave needs to encrypt with the same generated code and have the same values (IP addresses, user, MAC).
- In case of total disconnection, master and slave will notify the tracker with his current state; the match will not count.
- If there is a disconnection and it is caused by the match result (since one player is losing), the other player can report it.
- In case of disconnection the match will be a loss in player’s statistics in order not to favour disconnections.

CHAPTER 8. STATISTICS

To evaluate the final work there are some methodical steps to draw objective and subjective conclusions.

The objective results only demonstrate that the scenario is correct to start the subjective test.

The basic points to see are the delay resistance and loss capacity.

8.1. Scenario, configuration and tests

To test the application a private network with 3 pc is used.

The extremes using windows XP without firewall and at the middle a FrenzyBSD based on FreeBSD to add delay and loses (figure 18).

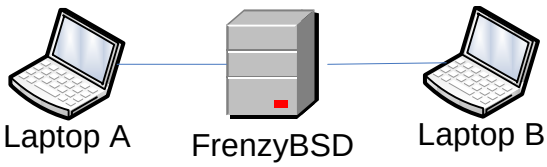

Figure 18 – Test scenario

The hardware used is:

- 2 x crossover cable
- 2 x laptop with Ethernet and windows XP
- 1 x personal computer with 2 NIC interface
- 1 x a CD with FrenzyBSD that includes dummy net software

Software used at server:

- Ipfw commands: these commands are used to set delay and loses in the network. There are a lot of options.
- Ifconfig commands: they can be used to configure network interfaces.
- mii-tool commands: they list the network devices and if there is any physical connection and the throughput negotiated at Ethernet level.
- Ping command: it is used to know if there is any communication between machines.

Software used at client side:

- Iperf commands: it is a software to generate UDP and TCP flows.
- Game implementation.
- Wireshark: it is a network sniffer.
- Network connection options from windows control panel.
- Ping command: it is used to know if there is any communication between machines.

The configuration follows the next steps:

1- Connect one laptop to the server with the cable.
2- At server side, open a console and write mii-tool with root privileges and know what interface is used.
3- Set the IP, netmask and default gateway at the laptop.
4- Set the IP and netmask to the correct NIC at server side.
5- Test using ping from server to laptop and vice versa. If there is any problem only from server to client look if the firewall is deactivated.
6- Repeat the same process for the second laptop.
7- Test connectivity between laptops.

Now there is the work setting.

Server configuration for each test [13]:

- ipfw add 100 pipe 1 ip from any to any → adds internal pipe from any to any with no rules.
- To add delay: ipfw pipe 1 config delay 100ms
- To add random losses: ipfw pipe 1 config plr 0.5

One thing to take into account is that with this configuration, delay and losses are applied at each entry queue.

From one laptop to the other, a single ping will be added 100ms delay four times and in each queue a 50% loose probability.

At the end of the test:

- ipfw pipe 1 delete → deletes the previous created pipe with all the rules.

Once ipfw commands are done, to test if the results are coherent, at one side it is used:

- iperf -c <server address> -u
- iperf -s

Iperf is only used to know if dummy net configuration is correct.

The tests to evaluate the application will be:

- Delay: 32, 64, 80, 100, 120, 150 and 200 ms.
- Looses P(e): 0’02, 0’04, 0’06, 0’08, 0’10, 0’15 and 0’20.

A good result will be that the game supports up to 120-150 ms delay and probability error up to 0’06.

8.2. Results

The results are all empirical evaluated from 0 to 5 (the best) in two terms: response time, since you move and you perceive the move, and screen synchronism, the difference between the screen clients.

Delay tests:

ms	32	64	80	100	150	200
Response time	5	4.5	4.5	4	3.25	2
Screen synchronism	5	5	5	5	5	5

P(e)	0’02	0’04	0’06	0’08	0’10	0’15
Response time	5	0	0	0	0	0
Screen synchronism	5	0	0	0	0	0

The delay results are really good, more than 100 ms delay with good response. They are along the same line as normal market games.

The error test has the worst results. The game does not start, and the problem is in the clock synchronization of the two peers and the politics of dummynet to discard.

To solve these errors we need to ensure a robust system for the signalling. There is a new version pending.

CHAPTER 9. CONCLUSIONS

At the end of it all, there is not any perfect solution but the one chosen out for this scenario to work correctly.

The goals of interactivity, if delay is not to big, can be respected and there is not any appreciable incongruence unless the scenario becomes hell.

For bigger games, a decentralized architecture will be a big problem with solutions that can give a bad interactivity or bad user experience because the game freezes when any super peer falls. Massive online games normally use any load balance system in server side and time warp synchronism techniques.

In terms of security, in mobile phone devices or other ones, the critical data that needs really secure channels are the client server communications, between peers, a simple symmetric but each time different key (symmetric key generated by the server and sent via the secure channel) to avoid easy bots should be ok.

In other cases, the security will be focused on revising all data that comes from users and really important, security inside the enterprise needs to be really high too.

In a few words, this document doesn't does not intend to become a great study, because this needs some years and more empirical tests, but a guide to start to know how to develop an online game for any platform; what needs to be taken into account, what problems might appear and some solutions.

This master thesis doesn't does not reflects all work related, like how to organize the code, more than 10k lines in C++, how to generate 3D graphics, how to manage inputs, non blocking input output system, how to do multiplatform games and much more technical work that have had to be done.

BIBLIOGRAPHY

Secure vulnerability definition

[1] http://technet.microsoft.com/en-us/library/cc751383.aspx

Synchronism:

[2] Shawn Bonham, Daniel Grossman, William Portnoy, & Kenneth Tam: *Quake: An Example Multi-User Network Application | Problems and Solutions in Distributed Interactive Simulations*, University of Washington, May 2000, CSE 561.

[3] Honghui Lu, Björn Knutsson, Margaret Delap ,John Fiore, Baohua Wu: *The Design of Synchronization Mechanisms for Peer-to-Peer Massively Multiplayer Games*, Department of Computer and Information Science, University of Pennsylvania 2009.

[4] Marco Roccetti, Stefano Ferretti, Department of Computer Science, University of Bologna, Claudio E. Palazzi, Department of Pure and Applied Mathematics, University of Padua: *The Brave New World of Multiplayer Online Games: Synchronization Issues with Smart Solutions*, 2008 IEEE.

[5] Haishu Zhang: *The effect of delay on network games,* Master's Thesis at Umea University, May 2006.

Game cheating:

[6] Steve Webb: *A Survey of Cheating Techniques in Online Games*, CS7001.

NAT transversal:

[7] Bryan Ford, Massachussets Institute of Technology, Pyda Srisuresh, Caymas Systems Inc, Dan Kegel: *Peer-to-Peer Communication Across Network Address Translators,* 2005 Proceeding.

[8] Y. Rekhter Cisco, B. Moskowitz Chrysler Corp., D. Karrenberg, G. J. de Groot RIPE NCC, E. Lear Silicon Graphics Inc., RFC 1918: *Address Allocation for Private Internets,* February 1996.

[9] J. Rosenberg, J. Weinberger dynamicsoft, C. Huitema Microsoft, R. Mahy Microsoft, RFC 3489: *Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs),* March 2003.

[10] J. Rosenberg Cisco, R. Mahy, P.Matthews, D. Wing Cisco, RFC 5389: *Session Traversal Utilities for NAT (STUN)*, October 2008.

TCP:

[11] M. Allman NASA, V. Paxson ACIRI, W. Stevens, RFC 2581: *TCP Congestion Control,* April 1999.

[12] Claudio Casetti, Politecnico di Torino, Mario Gerla, UCLA Computer Science Department, Saverio Mascolo, Politecnico di Bari, M.Y. Sanadidi and Ren Wang, UCLA Computer Science Department: *TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks,* 2002.

Dummynet guide:

[13] http://cs.baylor.edu/~donahoo/tools/dummy/tutorial.htm