TORCS V2

TORCS Networking Mode

Author: Eric Espie

Creation Date: April 25, 2003

Date: December 25, 2003

1 Introduction

TORCS (The Open Racing Car Simulator) is publicly available on the web at the following address:

http://torcs.org

The goal of this document is to describe the networking mode of TORCS in terms of architecture modifications and also to describe the protocols used.

The following list describes the features and constraints:

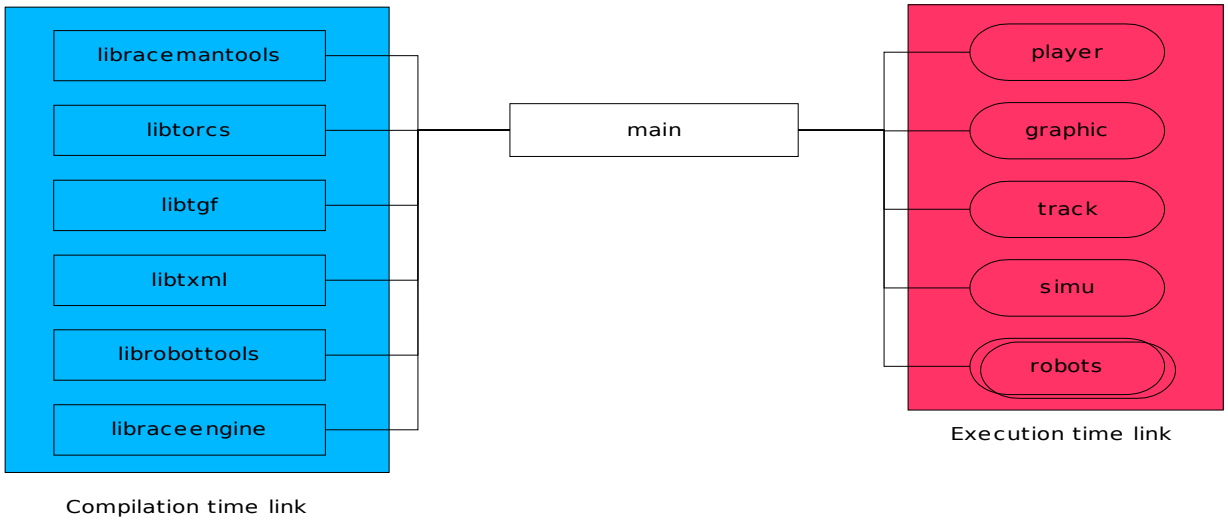
- The network mode is reserved to human players
- The target network is a LAN (tests on WAN can be done later)
- The client executable contains also the server code
- Pure server without graphic interface can be launched alone
- Separate client and server executable can be launched on the same system

2 Architecture

This chapter describes the general architecture.

2.1 Old architecture (version 1.2.x)

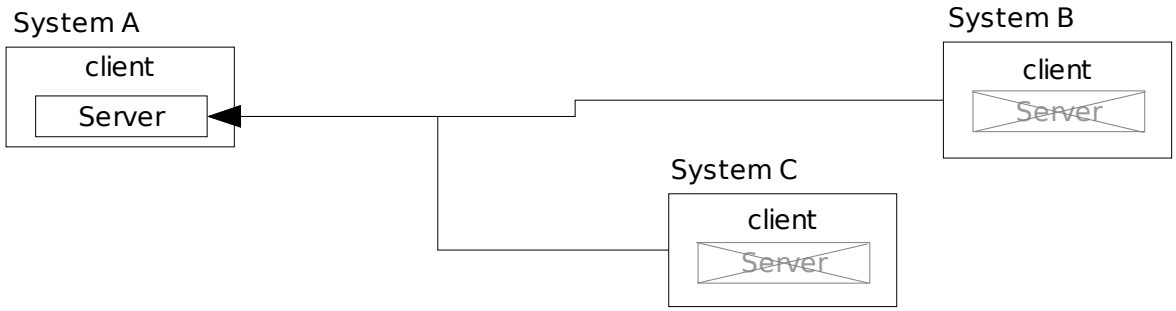
TORCS is organized in shared libraries, some are linked at compilation time and others are linked on demand at execution time (the robots for example).



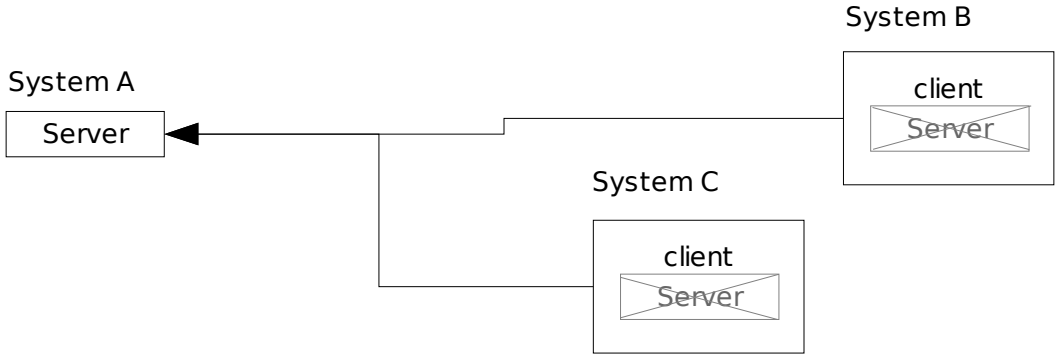
2.2 Architecture for version 2.x

The distribution will contain two executables, a client with an embedded server and a pure server. The clients are graphic applications while the pure server is a console based application.

If all the systems are used by players, one must be used as client and server and the others as client.



When a system is a dedicated server, a pure server executable can be run on it and players will run clients.

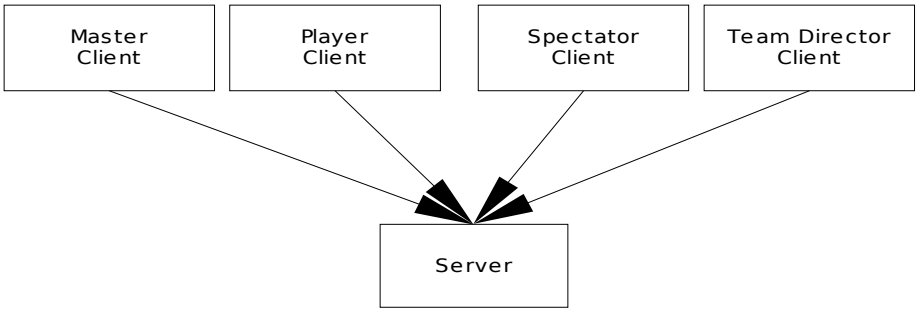


The server will be a console application (i.e. no graphic), and all the controls will be done from a master client.

2.3 Roles

Entities can be defined with different roles:

- **Server**: The host of the simulation.
- **Master Client**: The client controlling the simulation and optionally used to play.
- **Player Client**: Used to play.
- **Spectator Client**: Used to watch a race.
- **Team Director Client**: Used to play remotely with a robot.

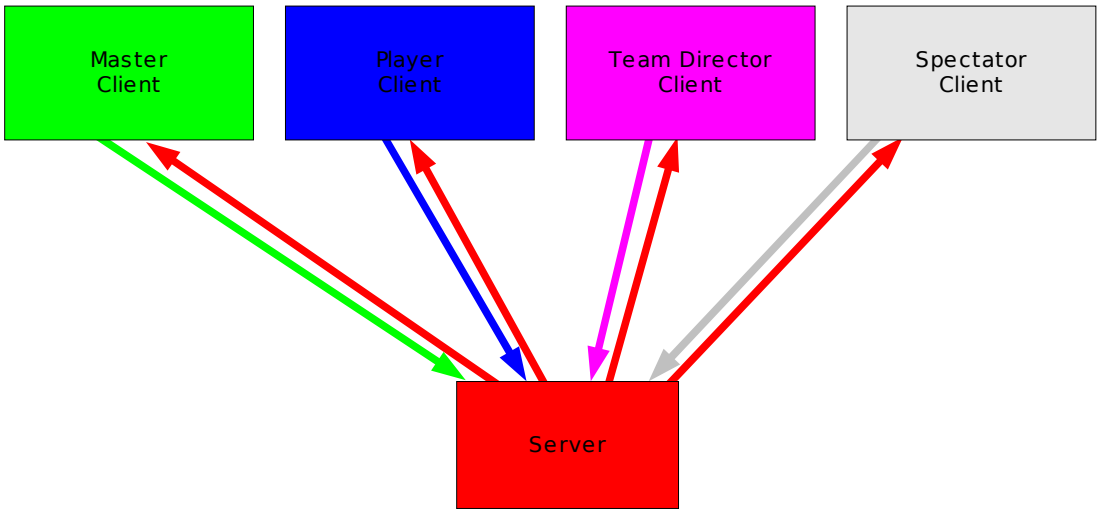


3 Operations

3.1 Network data flow

The following network flows can be described:

- Server control from the **Master Client** to the **Server**.
- Display information from the **Server** to the **clients.**
- Input device information from the **Player Client** to the **Server.**
- Control information from the **Team Director Client** to the **Server**.
- Connection information from the **Spectator Client**.



3.2 Race Initialization process

The initialization process will follow the steps:

1. Select the “Multiplayer” menu on the master client
2. Select “Launch Local Server” in the “Multiplayer” menu, this will enable the local server, or
3. Select “Master a Remote Server” in the “Multiplayer” menu, to enable the connection between the master client and the remote server
4. The server send the available tracks and robots list to the master client
5. Configure the race on the master client:

   race type, track(s), robots, number of remote players/spectators/team directors, ...

6. The master client send the race configuration to the server
7. The incoming connections are initiated on the server
8. The master client auto-connect the player if necessary (players have to be selected before the multiplayer process)
9. The remote clients connect themselves to the server and receive the track information.
10. The connections of the remote clients are signaled to the master client and to the already connected remote clients by the server
11. The master client send the start race order to the server
12. The server wait for all the (active) clients to be ready (graphic initialization) and start the race

Note that the spectator clients are allowed to connect at any time during a race providing the maximum number of clients is not reached. The other clients are not allowed to connect during a race.

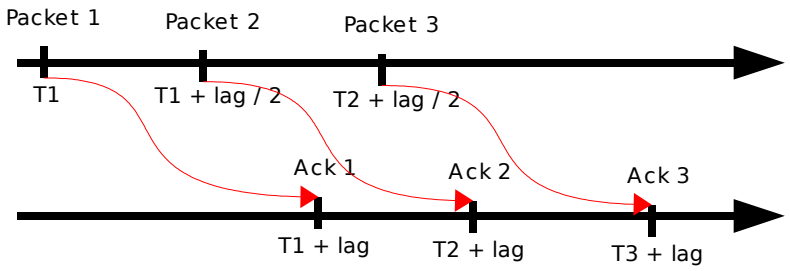
3.3 In Race Data Flow

3.3.1 Player Client to Server

The player robot is split into two parts. The part on the client side is used to collect the player's input (keyboard, mouse or wheel) and to communicate them to the part on the server which perform the computation of the response to the race engine.

The flow control can be achieved with acknowledgment messages. The lag can be computed and the packets can be synchronized on fractions of the lag.

For example:



The packets are small and the lag is constantly measured and can be displayed on the screen with a graphic indicator. The acknowledgment can contain information about the race engine state.

Minimum and maximum rates will be configurable.

3.3.2 Server to Clients

The server send to the clients only the information necessary to display the race.

A mechanism similar to the player flow control can be done plus a configurable maximum data rate depending on the network bandwidth because the data sent are much bigger and can saturate the link. Optimizations can be done to reduce the data transmitted depending on the power of the systems and the link bandwidth. Simple compression with the zlib can be done on WAN for example, but sending raw data on LAN will be faster.

On LAN another optimization can be done by using multicast for the systems on the same segment that the server.

It is obvious that the number of clients will be limited by the server bandwidth.

The data are split into two parts, one static sent only once by the server to the clients, and on dynamic sent during the race .

The data can also be customized to be sent only when necessary, for example when a car achieve a new lap more information is sent than during the lap (lap time, last lap time, best lap time, ...) these information can be stored in an optional part of the structure.

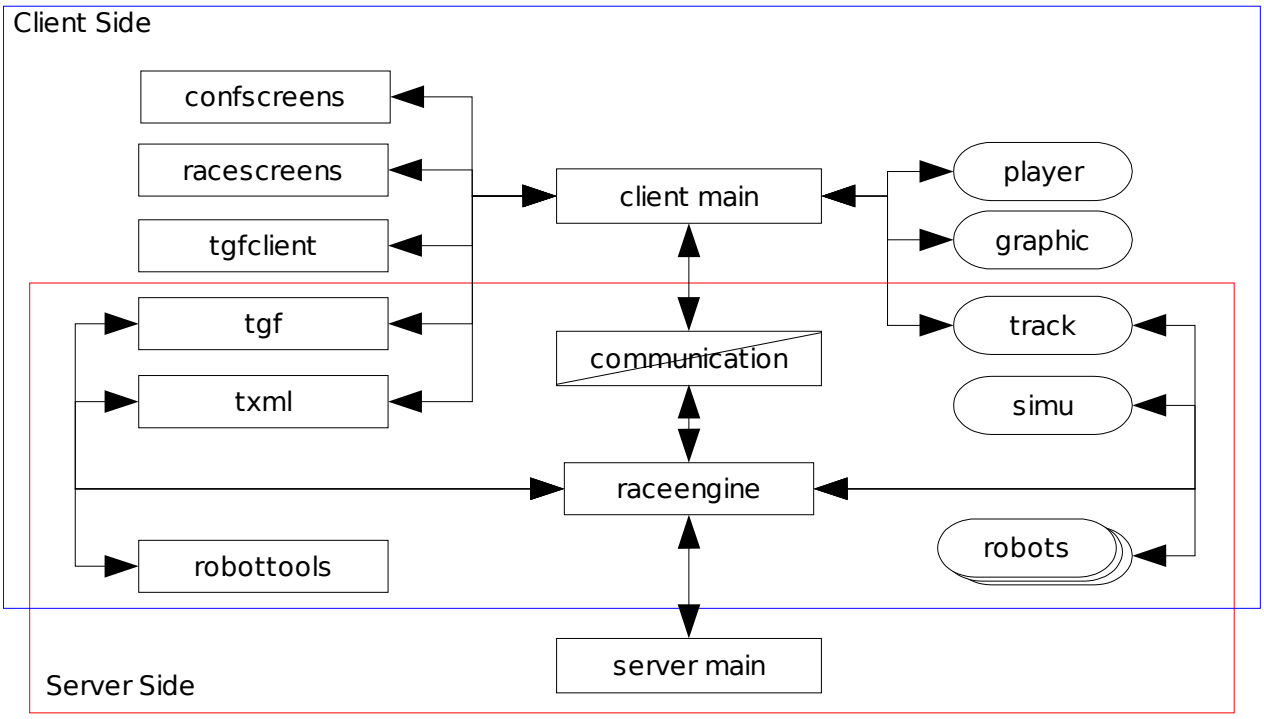
3.4 End of Race Process

The server send the race results to all the clients.

4 Detailed Architecture

4.1 Modules

In order to deal with all the constraints exposed above, the following architecture will be used for the modules.



4.2 The Tgf and Tgfclient Modules

The 1.2.x **tgf** module is split into two parts a **tgfclient** module dealing with all the screen related stuff and the **tgf** module keeping all the basic features.

4.3 The Screens Modules

The **confscreens** and **racescreens** modules will be a merge of the 1.2.x modules dealing with screen management. The modules impacted are, **torcs**, **racemantools** and parts of **raceengine**.

4.4 The Raceengine Module

The **raceengine** module will have to be heavily modified from the 1.2.x branch to separate all the GUI features from the race management.

4.5 The Communication Module

The communication module is separated into three parts:

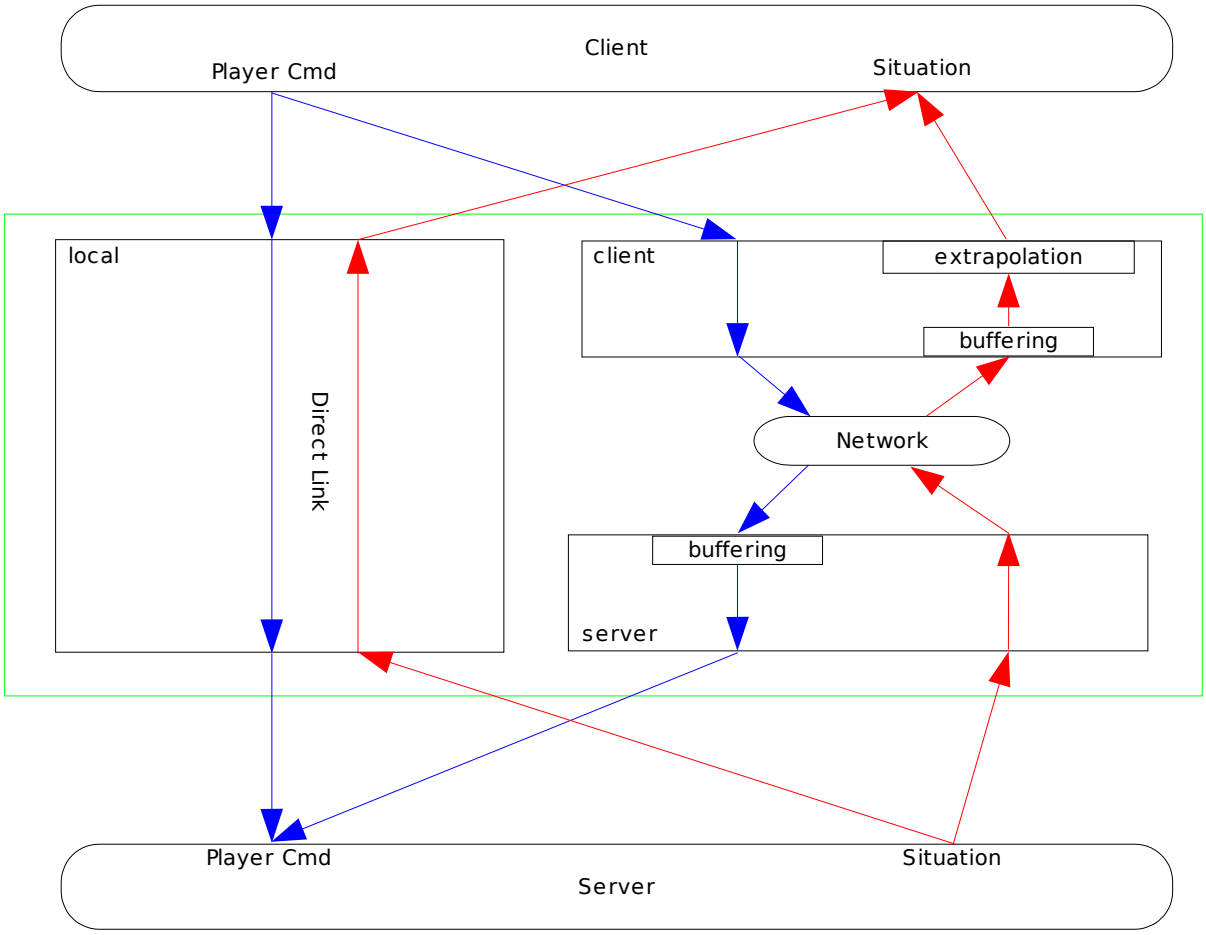
- Local communication when the client and server are in the same executable
- Client communication to manage the interface between the client and the network
- Server communication to manage the interface between the server and the network

The extrapolation module is used to provide updated situations to the client part at a higher frequency than the network. It will use a polynomial extrapolation function of order 1 or 2 depending on the variables. The buffering will have to save the three last positions.

The simulation module used for the extrapolation is optional. It will be used to refine the extrapolation if the lag is too important. The principle will be to simulate the player's car with the commands issued locally and the last available car position.

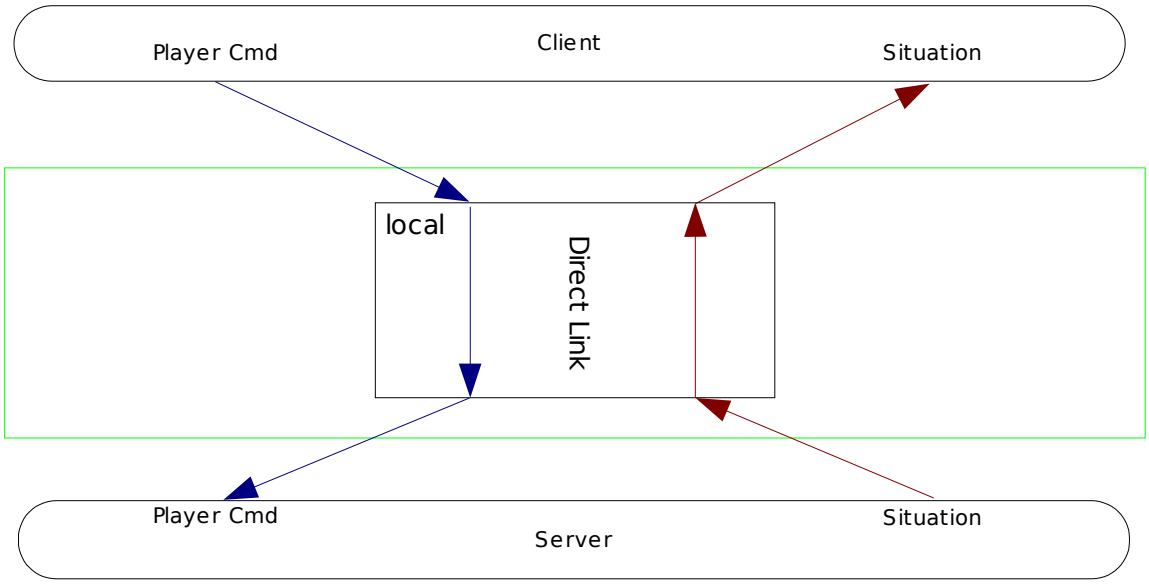
The data flow between clients and server is described below:

TORCS Networking Mode



5 Development steps

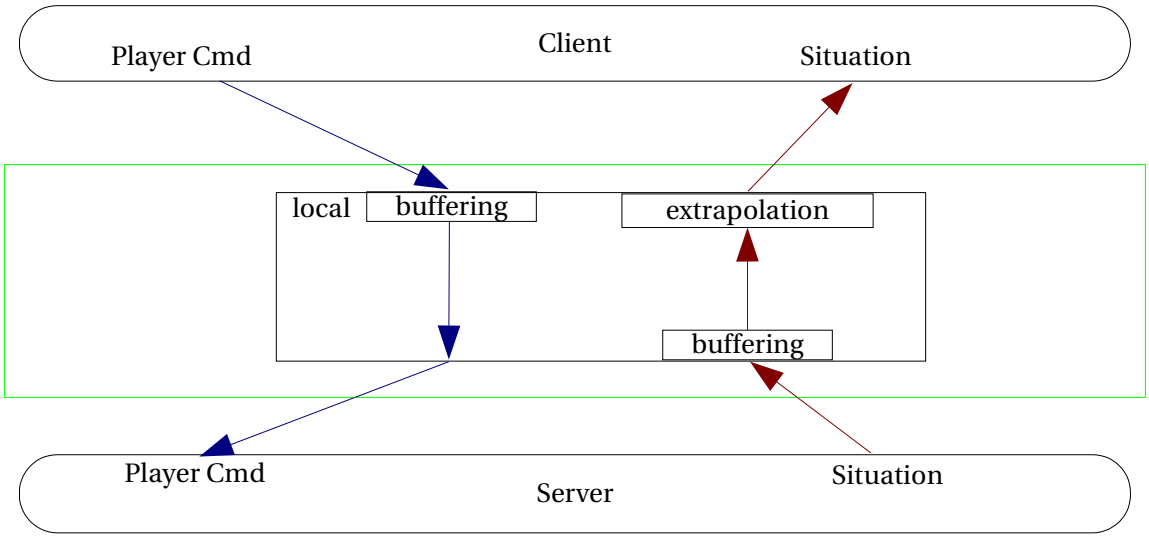
5.1 Client /server separation



The first step to be done is to separate the client and the server parts. An API as to be defined to exchange the player control commands and the game situation between the client and server parts.

5.2 Lag Simulation and Correction

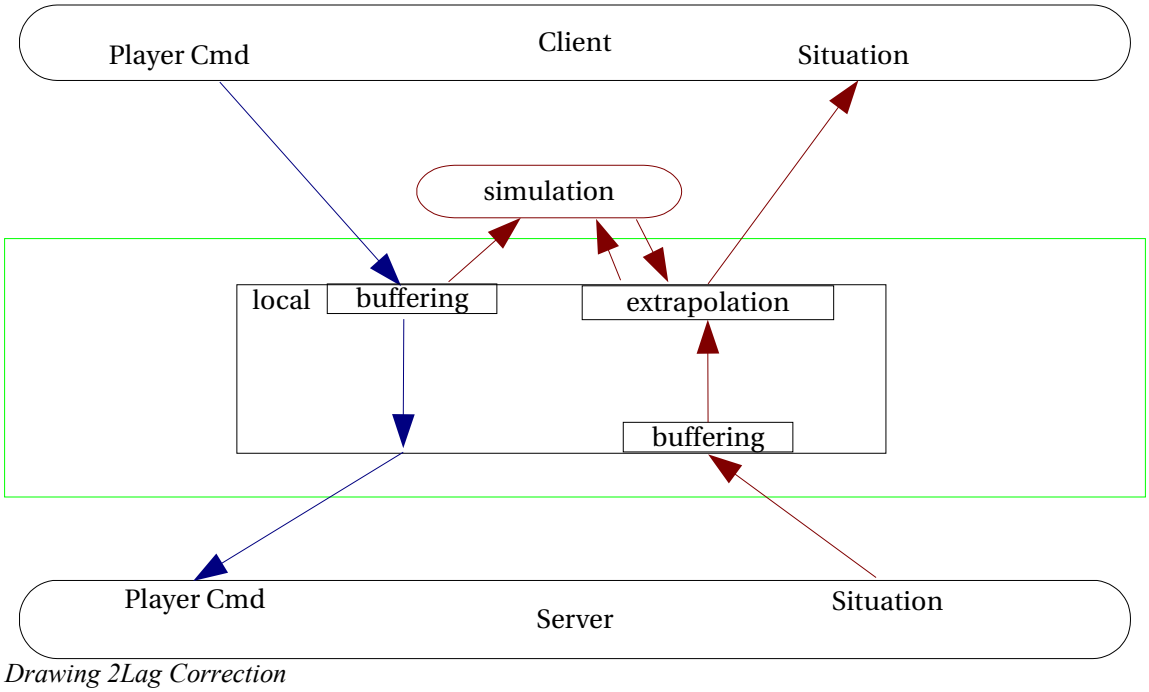
In order to verify the extrapolation function the communication module can be used to add artificial lag by adding a buffering function for both the commands and the situation.



Drawing 1Lag Simulation

5.3 Lag Correction Enhancement

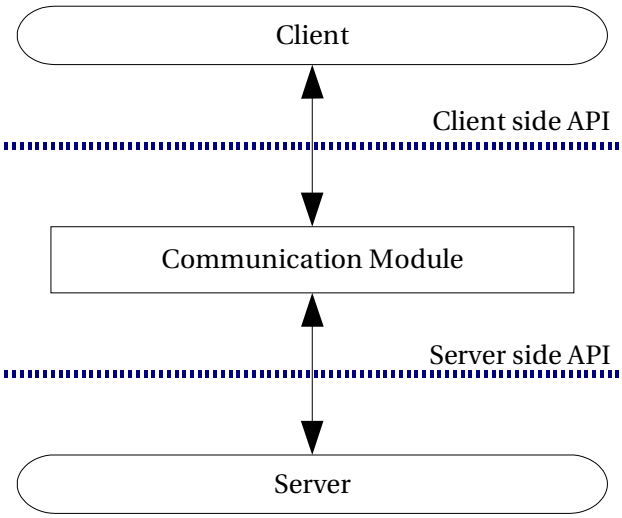
If the extrapolation is not good enough when the lag is high (simulation of WAN) then the local simulation module can be added to help having a better extrapolation for the situation.



Drawing 2Lag Correction

6 Communication APIs

There are two APIs, the client side and the server side.



Drawing 3API

The APIs are separated into the following parts:

- Server management
- Race management
- Race data

6.1 Server Management

This is the part dealing with the connection between the client and the server.

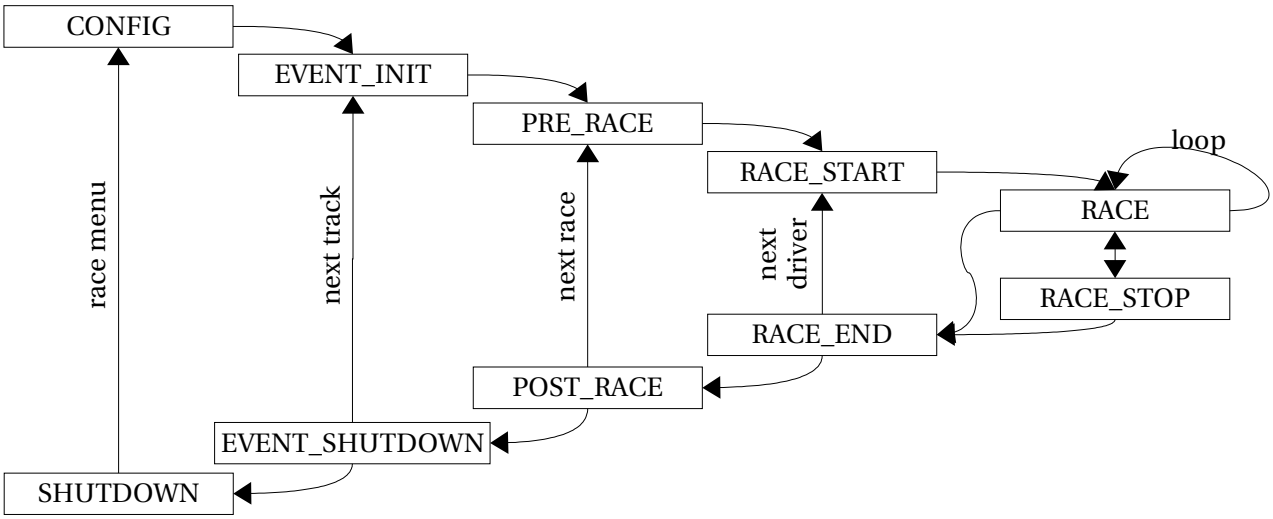
- Server discovery (network mode)
- login procedures (network mode)
- Server configuration (network and local modes)

6.2 Race Management

The functions for the race management are:

- Race configuration
- Results retrieving

The client race states are the following ones:



Drawing 4Race States