INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Vector-Field Consistency para Mobihoc.Net

Integração de Modelo de Consistência Adaptado a Jogos Multi-utilizador na .Net Framework

Vector-Field Consistency for .NET Multiplayer Games

DINIS DE MORAIS TIAGO LAGE

Dissertação para obtenção do Grau de Mestre em

ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Abstract

Multiplayer games are currently one of the major sources of entertainment worldwide. With the widespread use of the Internet it becomes essential to use the available communication resources effectively allowing the best interaction possible for the players. There already are many techniques and models dedicated to improve the gameplay experience and increase the maximum number of participants supported in each session.

The problems of consistency and scalability in terms of computational and communication cost are addressed in this work. There are several solutions that essentially try to balance between flexibility, consistency and performance of distributed systems and simulations.

The vector-field consistency model and its applicability to multiplayer games are thoroughly examined in this work.

Keywords: multiplayer games, locality-awareness, scalability, playability, consistency management

Resumo

Os jogos multijogador são actualmente uma das principais fontes de entretenimento em todo o mundo. Com a generalização da utilização da Internet torna-se imprescindível a utilização dos recursos de comunicação disponíveis efectivamente de forma a permitir a melhor interacção possível para os jogadores. Já existem muitas técnicas e modelos dedicados a melhorar a experiência de jogo e aumentar o número máximo de participantes suportados em cada sessão.

Os problemas de consistência e escalabilidade são abordados neste trabalho em termos de custo computacional e de comunicação. Há várias soluções que, essencialmente, tentar equilibrar entre a consistência, flexibilidade e desempenho de sistemas distribuídos e simulações.

O modelo de consistência VFC e sua aplicabilidade para jogos multijogador são analisados neste trabalho.

Palavras-chave: jogos multijogador, localidade de interesse, escalabilidade, jogabilidade, gestão de consistência

Table of Contents

List of Tables

List of Figures

1 Introduction

Multiplayer games currently represent one of the most popular uses of group interaction on the Internet. The real-time pace of many of these games has special interest because these require short response and communication periods. By controlling the quantity of information transmitted and adapting the consistency, depending on the interest level of each player, it is possible to increase the number of supported participants and improve the way in which they interact in the virtual world.

Multiplayer games require some consistency to be enforced in order to allow a smooth interaction of the players. The requirements vary on each type of game and the game in itself.

Another important aspect of multiplayer gaming that motivates this work is the urge for efficient methods of consistency enforcement due to its usual real-time pace.

Understandably, to accommodate multiplayer games consistency and performance requirements, an effective optimistic approach to the consistency enforcement should be used.

1.1 Objectives and contributions

The work presented in this thesis has the following objectives:

1. Make an assessment of the work related to networking in multiplayer games, consistency enforcement mechanisms and the use of locality-awareness in those mechanisms;
2. Adapt and apply the VFC (*Vector-Field Consistency* [1]) model to a multiplayer shooter game and analyse its impact on the gameplay and performance in a distributed environment of personal computers;
3. Implement a prototype XNA game that uses VFC and evaluate its performance;
4. Evaluate the complexity required in order to use the VFC in the game communication.

1.2 Document Roadmap

In chapter 2 we discuss the work related to consistency enforcement and communication mechanisms used in multiplayer online games. Chapter 3 describes the architecture for consistency enforcement in multiplayer games, used on the adaptation of Vector-Field Consistency to a multiplayer action game. In chapter 4 we describe the implementation details of the prototype game. Chapter 5 exposes the evaluation methodology and the results that were obtained while evaluating the prototype. Finally, in chapter 6 we summarize the achievements and make some observations for the forthcoming work.

2 Related Work

In this chapter, some relevant consistency mechanisms, with possible application in multiplayer games, are discussed. This section begins with an introduction to some general concepts that can be applied to these games and then expose some models that use locality-awareness in multiplayer games. Section 2.3 presents some general requirements depending on the game genre. In 2.4 an analysis is made of multiplayer games' architecture and some examples are given of actual implementations in prototype and commercial games. Section 2.5 makes a comparison of the usual network architecture in multiplayer games. In section 2.6 an assessment of the work related with networking in multiplayer games is made.

2.1 Consistency Mechanisms

In any distributed system there is a need for mechanism to enforce some type of consistency of the shared data. These mechanisms are typically separated in two groups: pessimistic and optimistic. Consistency mechanisms explore the span shown in Figure 1 regarding each system's need for availability, performance and tolerance for access to inconsistent data.

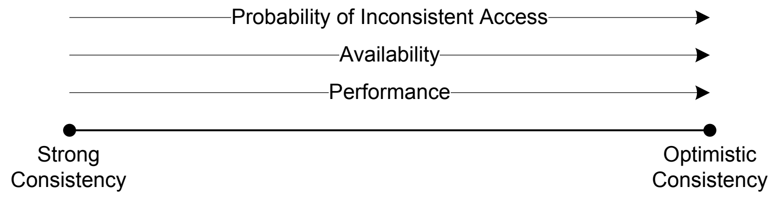


Figure 1 – Relation between consistency, availability and performance

Following we present an analysis of mechanisms used to enforce consistency among replicas.

2.1.1 Pessimistic Replication

Traditional replication techniques try to maintain an approximation to a single-copy of data, giving users the illusion that there is only one highly available copy. There are many ways of accomplishing this objective but, in general, the mechanism blocks access to data when it is not possible to prove it is up to date. This is the reason why these are so-called pessimistic techniques. The performance obtained by these techniques in a local area network is good due to the low latency and high availability but the same cannot be expected in a large scale network, like the Internet, for three main reasons:

First, even with the technology evolution, the Internet is still slow and unreliable. Besides that, the use of mobile devices, with intermittent connections, is becoming increasingly popular.

Second, pessimistic algorithms scale poorly on the wide area. It is difficult to build a large scale pessimistically replicated system, with frequent updates to data, because the increase on the number of nodes damages the throughput and availability of the system.

Third, there are activities in which the user requires the data sharing to be made asynchronously. Some require the users to be in relatively isolated environments. It is better to allow concurrent access and repair occasional conflicts when they happen, than to completely lock out the data while someone might be editing it.

In many problems in which some data inconsistency can be tolerated it is better to use an optimistic mechanism. Optimistic replication is a group of techniques that allow data to be shared efficiently in wide area networks and mobile environments. Contrarily to the pessimistic techniques, optimistic algorithms let the data be read and written asynchronously based on the assumption that conflicts will occur rarely, if at all. Updates are propagated in the background and conflicts are fixed after they happen.

2.1.2 Optimistic Replication

The algorithms of optimistic replication offer many advantages over the pessimistic ones: improved availability of data; flexibility with respect to networking, even with variable and/or unknown communication channels; should be able to scale to a large number of replicas because they require little synchronization among them; replicas and users are highly autonomous allowing replicas to be added with no change to the existing ones and also asynchronous collaboration between users like in version control systems as CVS [2] and SVN [3]; finally, optimistic algorithms provide quick feedback since they can apply updates tentatively as soon as they are submitted.

However, there is a big challenge in the optimistic techniques which is the possibility of diverging replicas and concurrent operations. Thus, it is only applicable to applications that can tolerate occasional conflicts and inconsistent data.

A more detailed analysis about optimistic replication is made in [4].

2.1.2.1 Availability vs. Consistency

In applications that can tolerate relaxed consistency there were different optimistic consistency models proposed [5-8]. Unfortunately, typically, optimistic models do not impose limits on the consistency of data that is made available to users. To solve this and limit the inconsistency of the objects some models emerged that tried to balance between availability and consistency.

Real-time guarantees [4] allow obsolete object replicas to be used for a specified time without confirming its freshness. During those periods the system allows the data to be used even if it is stale, reducing considerably the cost of managing that replicated data and improving the availability.

Order bounding [9] may be used to limit the number of uncommitted changes that may be applied to a replica. This allows transactions to proceed faster because a bounded number of preceding transactions can be ignored.

Numeric bounding is based on the definition of acceptable limits for the difference of the value for each replica. When the quota is exhausted in a given replica, it must be made consistent with the others. This concept was first introduced with *Tunable Availability and Consistency Tradeoffs* (TACT).

2.1.2.2 Tunable Availability and Consistency Tradeoffs

TACT (Tunable Availability and Consistency Tradeoffs) [10],[11] is a multi-dimensional model that proposes the use of numeric bounding together with order bounding. In this way it is possible to explore the existing space between strong and optimistic consistency. Since there is a limit on the inconsistency for each replica, it is now possible to calculate, with an expected probability, for example, if there will be a conflict from an update or if a read will observe an update that will be later reverted.

The relation between availability, performance and the probability of occurrence of inconsistent accesses by relaxing the requirements from strong to optimistic consistency is represented in Figure 1. Moving from strong to optimistic consistency, system availability and performance increase, but so does the probability of an inconsistent access. With TACT it becomes possible for each application to tune the level of consistency as needed.

Still, this model does not take into account locality awareness which can be crucial, for instance, in multiplayer games. The next section describes some techniques that were proposed that take into account different consistency degrees that relate players' positions and corresponding sensing and acting ranges.

2.2 Locality Awareness in multiplayer games

The notions found in locality awareness are fundamentally related with the concept of interest management, used to filter a large number of data in large-scale distributed simulations [12]. By communicating only relevant data to other nodes it is possible to optimize the use of the communication channel.

2.2.1 Dynamic load management

The authors of [13] propose the use of locality awareness to make load balance in MMG (Massive Multiplayer Games). Based on localized information the proposed algorithm makes the load balancing and reduces the communication between servers, while trying to avoid changes of responsibility for each zone. Given that, with this approach, the existence of hotspots (concentration of a lot of players in the same zone) is possible, a heuristic is implemented to calculate when the load should be reduced in a server (splitting highly populated zones) or aggregated on servers with low load (grouping sparsely populated zones).

Locality aware dynamic partitioning uses two dimensions for its heuristics: i) communication based on network proximity in the game; and ii) region clustering based on adjacency on the game map. The first translates in a load balancing algorithm that favours localized communication between servers hosting neighbouring partitions for both load shedding and aggregation. The later favours an attempt to keep adjacent regions together on one server.

Each server periodically communicates its load information, along with data representing the area of interest, to its neighbours. This information is maintained for all neighbours and when overloaded with clients, the server favours dropping load to its neighbours rather than any other server. On the other way around, when under loaded, the servers try to correct locality disruptions that might have been caused by load shedding.

2.2.2 Visibility-driven communication

In [14] it is proposed an approximation turned to the players' visibility. In that architecture a division of the virtual world is made based on layers. That partition is done only when the game is loaded so that there is no need to redraw the zones during the game. The idea behind this type of partition is that by increasing the granularity of the zones it is possible to reduce the size of each zone. By reducing the size of the zone it is possible to reduce the probability of finding a big number of players in that zone as well. Figure 2 presents an example of the subdivision of the zones with two layers. Considering zone 1 represents a zone with an overload of players, each server responsible for a zone contiguous to zone 1 would take over the responsibility of a corresponding subarea of the zone 1. This way, locality is preserved since the load is shared only with contiguous zones. The algorithm is also capable of dealing with multiple contiguous zones of overload. The hexagonal division allows regularity when the partitioning of multiple zones with overload is needed. The authors were able to verify a substantial reduction on the number of messages that was needed during the in-game interaction.

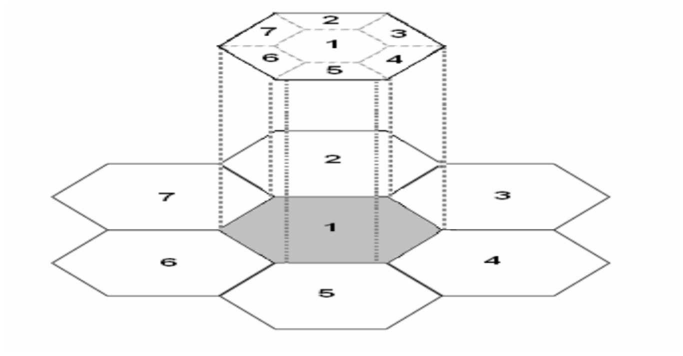


Figure 2 – Partition with two layers (from [14])

2.2.3 Network infrastructure

In [15] an infra-structure at the network level is discussed, in which some of the servers' functionalities are executed on computational platforms (booster boxes) that are co-allocated with the routers and are network state aware.

Booster boxes acquire network awareness by monitoring traffic, measuring network parameters such as delay. Each booster box serves a range of clients and performs caching, filtering, forwarding and redirection of game events in a game-specific manner. The main advantage of this approach when compared to distributing the servers across the network is that booster boxes combine network and application awareness in a single entity. For example, forwarding decisions can be based on application logic as well as current network conditions.

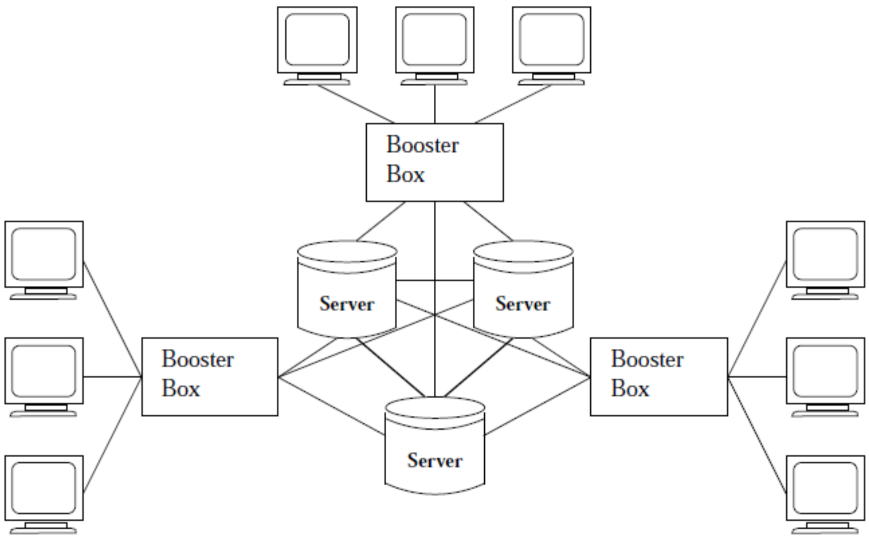


Figure 3 – Distributed servers with booster boxes

Booster boxes are co-located with routers and are based on programmable network processors in order to achieve adequate performance.

This way it is possible to reduce the load on the servers and they argue that it is possible to do some tasks otherwise intractable.

2.2.4 Peer-to-peer

Another interesting solution is the implementation of a support platform for MMGs over peer-to-peer presented in [16], where it has taken advantage of the players' locality of interest. It is then possible to arrange the players in self-organized groups based on their location in the virtual world. Their application was built on top of Pastry [17] as a peer-to-peer overlay and used Scribe [18], a multicast infrastructure built on top of Pastry, to propagate game state.

Pastry is a generic, decentralized peer-to-peer content location and routing system for very large overlay networks of nodes, which reliably locates a copy of the requested data in a decentralized, fault-resilient and scalable approach. In this system Pastry maps the participating nodes and the application objects to random, uniformly distributed IDs and implements a distributed hash table to support object insertion and lookup.

Scribe is a scalable application level multicast infrastructure built on top of Pastry. It uses the same Pastry identifiers to map the multicast groups. In the proposed peer-to-peer system for MMGs, Scribe maps the multicast groups to the same ring of identifiers of Pastry. Each group has an associated multicast tree formed by the union of the Pastry routes from each group member to the group ID's root. The root of the group ID acts as the root of the multicast tree as well. Scribe can efficiently tolerate a large number of groups, random numbers of group members and frequent changes in group's memberships.

2.2.5 Vector-Field Consistency

VFC [1] is an optimistic model designed to allow bounded divergence of object replicas. In this model replica consistency is selectively and dynamically strengthened or weakened based on the on-going game state and simultaneously it manages: how the consistency degree is changed throughout the execution; and how the consistency requirements are specified.

Locality-awareness techniques are employed to deal with the consistency degree required. To calculate the consistency degree required it uses the distance to certain 'observation points', called *pivots* (e.g. player's position). The consistency degree weakens with the increase in the distance to a pivot. Since pivots can change with time (e.g. a player moves), consistency requirements of the objects can change with time as well. Figure 4 shows a pivot (P) and its zones with objects belonging to the virtual world.

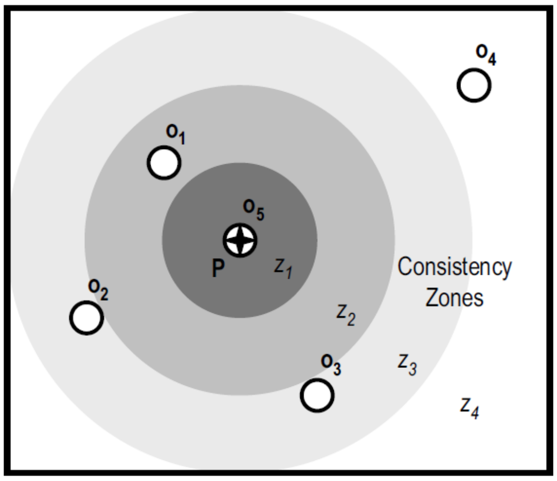


Figure 4 – Consistency zones centred on a pivot within the virtual world

The specification of the consistency requirements is done by providing a three-dimensional vector which represents the consistency degrees. Each of the dimensions bounds the replica divergence in *time* (delay), *sequence* (number of operations) and *value* (magnitude of modifications).

The *time* dimension specifies the maximum period allowed for a replica not to be refreshed with its current value, regardless of the number of updates performed during that interval.

Sequence represents the maximum number of replica updates that can be lost, that is, updates not applied to a replica.

Value represents the maximum relative difference between the replica contents against: the actual value (e.g. the difference between a player's position and the replica's value for it cannot exceed a certain value); or a constant (e.g. a player's score is approaching the top value).

2.3 Main Genres of Multiplayer Games

In order to improve the network performance in interactive multiplayer games it is necessary to understand the types of traffic generated in each game genre [19]. The communication network requirements of most of the games can be inserted in one of the following categories or, in some cases, might have a group of elements from each game genre:

- First Person Shooters (FPS) – this game genre involves a big part of combat that requires rapid response times. It is necessary to have a great number of messages, such as position updates or players' actions. In [20] a study was made that shows that the latency starts being noticed at about 100ms and that at approximately 200ms the game playability becomes impossible. This game genre has the highest latency requirements.
- Role Playing Games (RPG) – visually, this game genre is similar to FPS but the speed of the interactions of the players is less intense. Online RPGs usually aim at supporting a large number of players, and so, require good scalability. These games require the most in terms of concurrent game flows.
- Real-Time Strategy (RTS) – in this game genre the effect of latency, even if easily detectable by the player, does not interfere in the playability nor in the performance of the player due to its nature that clearly promotes the strategy over real-time aspects as studied in [21] and [22].

In Table 1 presents a group of characteristics of the different multiplayer game genres. Nowadays, there are MMORPG (Massively Multiplayer Online Role-Playing Games) games that are able to support a large number of players (up to thousands) using parallelization per zones, splitting the virtual world in various independent zones to process it concurrently in several servers. In FPS and RTS games, in which scenarios are usually smaller, it is still not possible to support a massive number of participants because the zone approximation is not adequate to these. Both in FPS and RTS players tend to concentrate their units in certain zones where the action takes place.

Game Genre	Number of Servers	Maximum players	Maximum Latency (ms)	Virtual World Size	Number of controlled entities
FPS	1	12 to 32	100	small	one
RPG (MMORPG)	1 (hundreds)	2 to 12 (thousands)	500	big	one
RTS	1	6 to 12	1000	medium	tens to hundreds

Table 1 – Major characteristics of the main multiplayer games genres

2.4 Multiplayer Games Architecture

Nowadays, most of the commercial games are implemented with client-server architecture, using only one or a cluster of servers. Each client has an independent connection to the server and all the communication is made through it. This way it is easier to reduce the possibilities of cheating, assure the anonymity of the players and the administration is simplified. In this architecture the way we can deal with the generated traffic differs between server to client and client to server, for the following aspects:

- Server to client: servers identify groups of clients to which it sends the same information. Servers can use multicast as a way to improve the efficiency in the network use or, even if that is not possible, it is possible to prioritize or group the messages.
- Client to server: clients may generate events periodically or because of user interaction. Usually clients only communicate with a server and not directly between them.

In almost every commercial FPS the architecture used is simple client-server, where there is only one server node. Because of that, each server has a reduced player capacity. Consequently, games like *Quake*, *Half-Life* and *Unreal Tournament*, have thousands of available servers worldwide. This worldwide distribution also helps reduce the latency which intensively affects games of this genre. Other techniques, like client-side prediction, are frequently used to compensate for latency. Using this technique the users commands can be interpreted immediately (e.g. movement, weapon firing), assuming these commands will be accepted by the server and, if there is some inconsistency it is subsequently corrected with the servers response [23].

In FPS games, players tend to converge to a rather small virtual space where more action takes place. In order to address that problem in [24] the use of a cluster of servers was studied in order to increase the number of supported players in FPS games. Though it is possible for more players to join the same session, its scalability is not similar to the massive participation possible in RPG games.

2.4.1 World of Warcraft and Everquest

The use of a server network is frequently used in MMORPG games. The vast scenarios, the relaxed latency requirements for good playability (approximately five times more than FPS), controlling a single entity (compared to tens or hundreds in RTS), make it possible for games of this genre to implement replication techniques with clusters that share the responsibility for different zones of the virtual world. Many times, in order to explore a certain section of the virtual world, players have to explicitly connect to a certain server who is responsible for that section. Examples of great success that implement such techniques in MMORPGs are *World of Warcraft*[1] and *Everquest*[2].

In other genres, there are multiplayer games that use decentralized models, like *MiMaze* [25] and *Age of Empires* [21], whose design has high limitations in terms of scalability because the game state is transmitted to all players.

2.4.2 MiMaze

MiMaze is a game in which each player, represented by a 3D avatar similar to *Pacman*, moves along a maze trying to “kill” the maximum number possible of opponents. This game has a completely distributed architecture and uses multicast for the communication between the players. All actions must arrive, be processed and shown to all players with the shortest delay possible.

[1] http://www.worldofwarcraft.com/

[2] http://everquest.station.sony.com/

2.4.3 Age of Empires

Age of Empires uses a different approach where each participant executes an identical simulation of the game. Only the players commands are transmitted, which reduces the quantity of information that would be needed because of the large number of entities each player controls. The communication is made in an all-to-all mode. It also uses a mechanism to control the possible speed of the game. Since all terminals must execute the same simulation the game speed is dependent of not only the connection between them but also on the processing capacity of each terminal. This model is robust against cheating (e.g. try to falsify resources or number of units) because the player would be considered as desynchronized.

2.4.4 Mercury

An alternative to the behaviour in *MiMaze* and *Age of Empires* includes *Mercury* [26], an FPS game implementation that is based on group communication depending on the position of the players in the virtual world. *NPSNET 3D* [27] is a vehicle simulator (requirements are similar to FPS games) that also uses group communication based on the interest in the virtual space.

In *Mercury* a distributed architecture that uses publish-subscribe based on area of interest (e.g. visibility, teamwork) was implemented. The responsibility of the interest zones is distributed uniformly among the participating nodes. It uses a routing protocol similar to Chord [28]. As in Chord, the members are organized in a logical ring. *Mercury* was designed to also allow search based on content rather than just identifying the responsible node for a specific resource identifier. During the course of the game, each node subscribes the content that has interest for the player's actions.

2.4.5 NPSNET 3D

NPSNET 3D, being a vehicle simulator, shares many similarities with FPS games, also having strict requirements in terms of maximum latency that allows good playability. It is based on a peer-to-peer architecture in which the virtual space is segmented in sets of hexagons that represent different groups of interaction. The set of groups (at least one) of each player changes with its movement.

Game	Genre	Architecture	Communication
Quake	FPS	Client-Server	TCP/UDP
World of Warcraft	RPG	Server network	TCP/UDP
MiMaze	FPS	Peer-to-Peer	Multicast
Age of Empires	RTS	Peer-to-Peer	All-to-all
Mercury	FPS	Peer-to-Peer	Publish-subscribe
NPSNET 3D	Simulator (FPS)	Peer-to-Peer	Multicast Groups

Table 2 – Comparison of the presented games

2.5 Network Architecture in Multiplayer Games

Multiplayer games generally use one of the following network architecture: client-server; or peer-to-peer.

2.5.1 Client-server Architecture

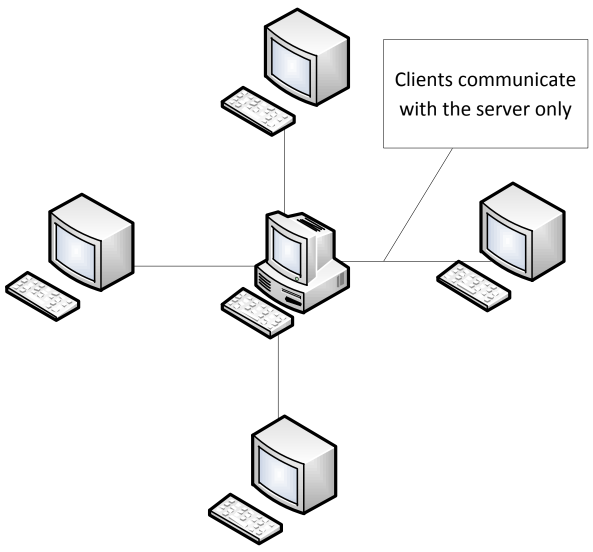


Figure 5 – Client Server Architecture

Typically, in a multiplayer game with client-server architecture players interact with each other through a server. Each client sends requests to the server, who maintains the game state and coordinates the propagation of the updates to clients.

As illustrated in Figure 5, each terminal running a game client only communicates with the server. Frequently, the game server also acts as a client, allowing all nodes to participate in the game. The server has a coordinating role regarding the globally accepted game state.

2.5.2 Peer-to-peer Architecture

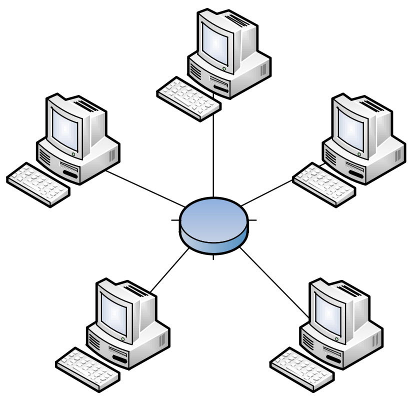


Figure 6 – Peer-to-peer Architecture

When a game uses a peer-to-peer architecture, clients also act as servers for part of the game state. In this approach the ‘correct state’ in each instant of the game is given by merging the player’s position on their own terminals. Each terminal is executing a parallel simulation and so there are easily and frequently discrepancies between the real state of an object and the state observed by each of the other participants. To prevent the various simulations to continuously diverge from each other some state packets are sent with given fixed interval.

2.6 Assessment

Optimistic consistency mechanisms favour significant improvements in the availability and performance of a system by relaxing the requirement for an exact match between replicas.

Client-server architecture provides a single location for the coordination of the state of shared data, including coordination of the consistency level that must be applied to each replica. When choosing peer-to-peer for the system architecture many other concerns must be accounted for (e.g. synchronization of the permitted actions).

Optimist consistency mechanisms typically apply divergence limits on the data by limiting: the interval in which a replica must be updated; the number of updates before a refresh must be made; the difference in value allowed for a replica. These limits are usually enforced to the data in general disregarding the importance of the locality and importance of the data to each user. By employing some awareness of the locality of interest of each participant, it is possible to prioritize and reserve resources where they are most needed.

Multiplayer games represent systems which allow for an optimistic approach to be made since these can tolerate some inconsistencies. Particularly, for FPS games, we propose a mechanism for consistency enforcement that is: **optimistic; centralized;** and **location-aware** (inside the virtual world).

3 Architecture

This chapter describes the architecture for consistency enforcement in multiplayer games, used on the adaptation of Vector-Field Consistency to a multiplayer action game.

Our solution was based on a XNA starter kit game called *Net Rumble*, modified to use client-server architecture, which would use a VFC model.

To analyse the VFC model, originally proposed in [1], it is now proposed to implement it on a multiplayer game in an environment of distributed personal computers. In this chapter we describe the design of our solution.

In Section 3.1 we detail some important considerations of the VFC model. In 3.2 we describe possible approaches and considerations of the system architecture. Finally, in 3.3 the main aspects of the software architecture are explained, describing the platform used in this work and denoting some considerations about the integration of the specific layers.

3.1 VFC Model

In VFC, each object is positioned in an N-dimensional space. Each node of the network has a local replica of the virtual world, its view. VFC delineates how the bounded inconsistencies between views are managed.

Within each view, object consistency depends on their distance to a *pivot*. Pivots are characterized by a position in the virtual world and can move over time. Figure 7 illustrates a virtual world populated with o_1, o_2, o_3, o_4 and o_5, with the consistency zones z_1, z_2, z_3 and z_4. Depending on the object's corresponding consistency zone, different requirements may apply.

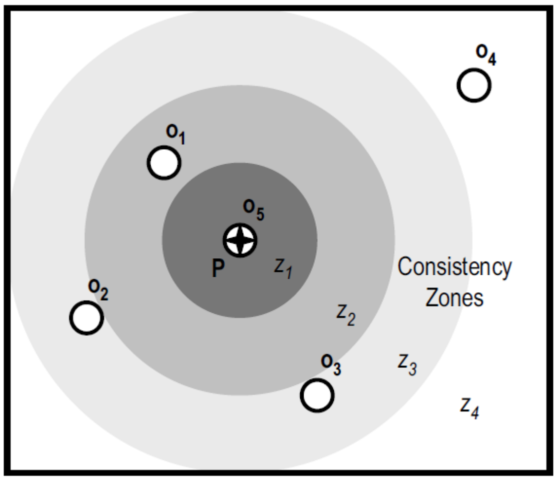


Figure 7 – Consistency zones centred on a pivot within the virtual world

Each zone maps to a consistency degree of a consistency scale. This scale is an ordered set of consistency degrees, which specify the consistency to be enforced for each zone. Determining the consistency degree of an object depends on its relative position with respect to the pivots.

VFC describes the consistency degrees as a 3-dimensional vector. Each dimension is a numerical scalar that defines the maximum divergence of the constraints of *time*, *sequence* and *value*.

The *time* dimension specifies the maximum period allowed for a replica not to be refreshed with its latest value, regardless of the number of updates performed during that interval.

Sequence represents the maximum number of replica updates that can be lost, that is, updates not applied to a replica.

Value represents the maximum relative difference between the replica contents against: the actual value (e.g. the difference between a player's position and the replica's value for it cannot exceed a certain value); or a constant (e.g. a player's score is approaching the top value).

To enforce this consistency model a coordinator is required. Its role is to control whether or not the consistency requirements for each pair of objects and pivots are being complied and act accordingly.

Figure 8 illustrates the architecture of the VFC Coordinator, the interfaces it needs and its relations to other VFC concepts.

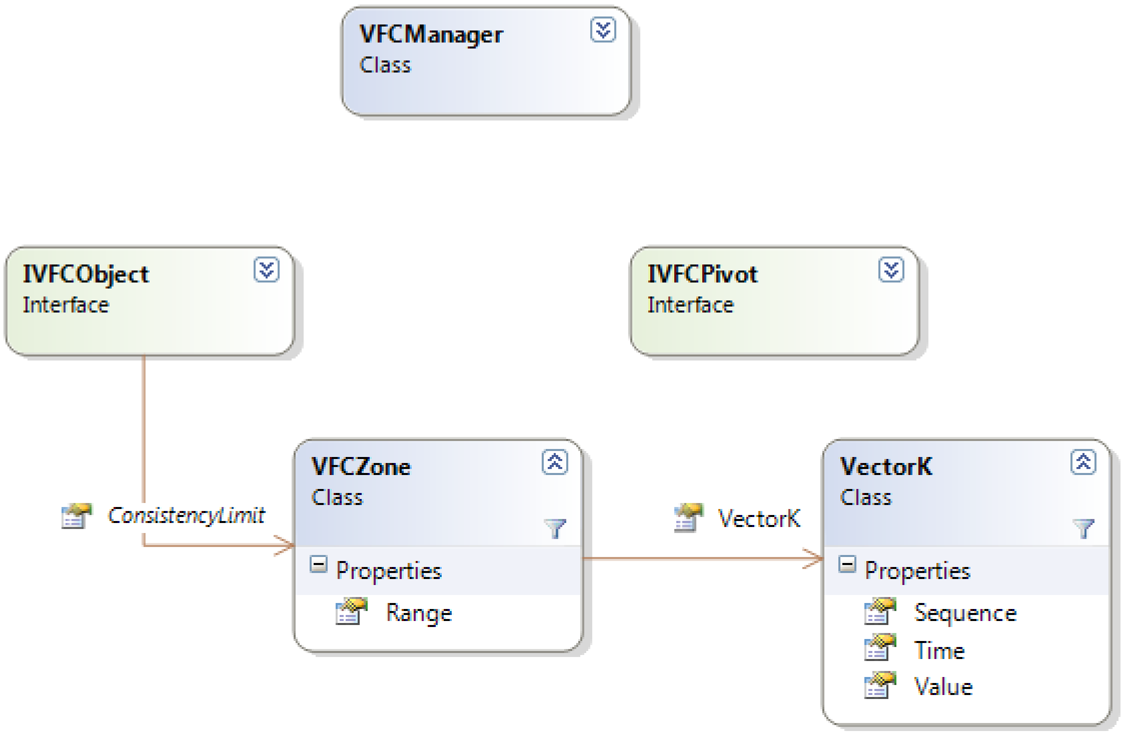


Figure 8 – VFC Coordinator and related entities

In Figure 9 the logic to be applied in the VFC manager is presented. When an object is updated the trigger mechanism will cycle the pivots and verify if the consistency is still complied with this update and selects whether to refresh the corresponding replica or not.

```
UpdateObjectTrigger(object)
  FOR each pivot
    IncreaseUpdates(object, pivot)
    IF VFCRequirementsComplied(object, pivot) DO
      SkipToNextPivot
    ELSE
      RefreshReplica(object, pivot)
      MarkRefresh(object, pivot)
    END
  END
```

Figure 9 – Pseudo-code of VFC Coordinator

3.2 System Architecture

To make a comprehensive study of the effects of the use of VFC in the communication it is important to use multiple methods in the communication and compare the impact it has on all of them. Following we compare the approaches to the system architectures of client-server and peer-to-peer.

Using client-server communication, the determination of the interest areas around each pivot is simplified because the server has the correct data for all players. Given this, the server can decide where to send the information needed and ensure that the consistency requirements are enforced.

Besides security issues, that are out of the scope of this work, a peer-to-peer implementation also aggravates the work required to comply with the game rules. In this type of architecture, when allowing some divergence between the replicas, situations may happen where a certain player performs an action in a moment when it would be allowed by some of the participants and disallowed by other. This would require an agreement mechanism which would increase the complexity of the coordination process.

Considering that, it is better to implement the game in a client-server structure in which all clients send the desired actions to the server and only consider updates to the game state sent by the server. To minimize the delay between the player's actions and its application to the controlled entity in the game, it can suppose that his action will be accepted by the server and correct possible differences in the execution (e.g. a packet sent to the server is lost).

Client-server architecture is simpler and, although theoretically it supports less players than a peer-to-peer architecture, in the scope of VFC, the gain in terms of avoided communication are potentially higher and it can be an advantage in terms of network traffic reduction.

3.3 Software Architecture

When designing an implementation of VFC in a multiplayer game it is advantageous to approach it as an independent module that abstracts from the game logic and grants the possibility to use it in different games with minimal effort.

Figure 10 illustrates the different software layers involved in this game design in the server and the client. As seen, clients abstract the VFC layer of the server.

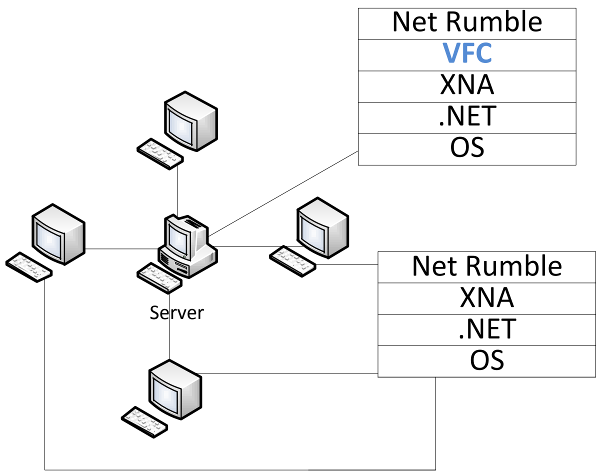


Figure 10 – Software layers run on clients and server

Besides the advantages exposed in the previous section, in client-server architecture, clients do not need to be aware of the use of VFC. For clients, the implementation of VFC is then transparent. In a peer-to-peer architecture all nodes would need to have the VFC logic and, in order to enforce a certain consistency, would need to settle the parameters among themselves.

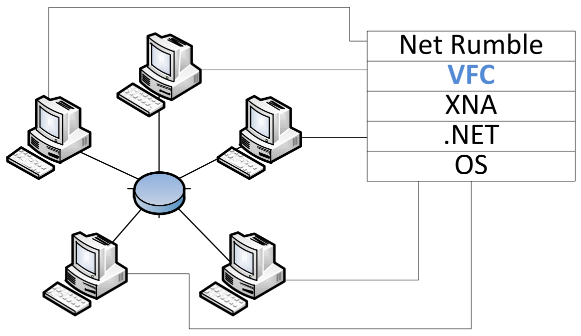


Figure 11 – Software layers run on peers

Next, we will make a brief introduction to XNA, followed by an integration of VFC as an extension of XNA in this context. Finally, some considerations specific to the game used in this work are made.

3.3.1 XNA

XNA™ is a set of tools with a managed runtime environment provided by Microsoft® that facilitates video game development and management.

The XNA Framework is the .NET-based framework for development of video games and simulations for deployment on Windows PC, Xbox 360 or Zune. It includes an extensive set of class libraries, specific to game development, to promote maximum code reuse across target platforms.

As seen in Figure 12, XNA has a very modular structure, and it allows the developer to choose which libraries from the framework are to be used.

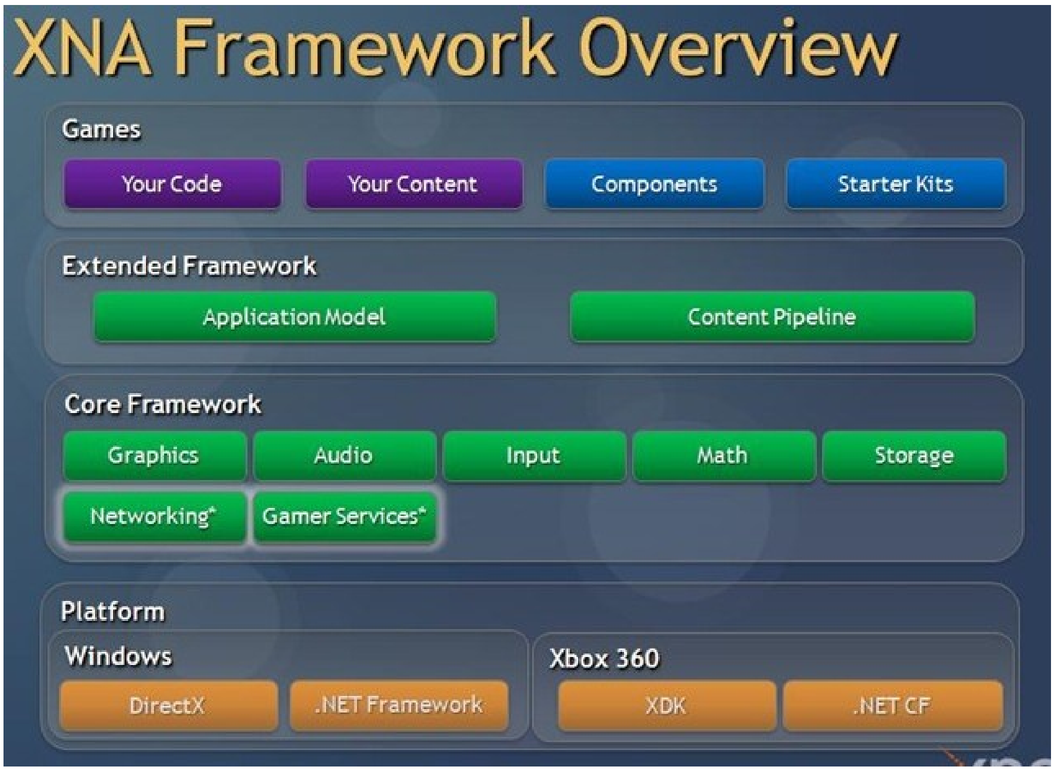


Figure 12 – XNA overview

The XNA Framework was developed with two primary goals: to enable cross-platform game development; and to simplify game development. To describe the XNA Framework it is appropriate to think of it as a series of layers. In a bottom-up approach these are:

- Platform – the lowest layer which consists of the low-level native and managed APIs that the framework is built on top of;
- Core Framework – provides the core functionality extended by the other layers;
- Extended Framework – group of components focused on easing game development;
- Games – the highest layer which groups the game specific content

3.3.2 VFC Integration

Typically, in XNA and multiplayer games in general, a common method to update players' actions and game state is to use packets containing relevant information. The selection of dispensable packets can be placed in an alternate method that, instead of updating all clients, enforces VFC and only sends the update to the relevant participants. Figure 13 illustrates the layer in which VFC is enforced. This minimizes the effort needed to be done in order to adapt a game to use the VFC model. Furthermore, this eases the process of enabling and disabling the use of VFC.

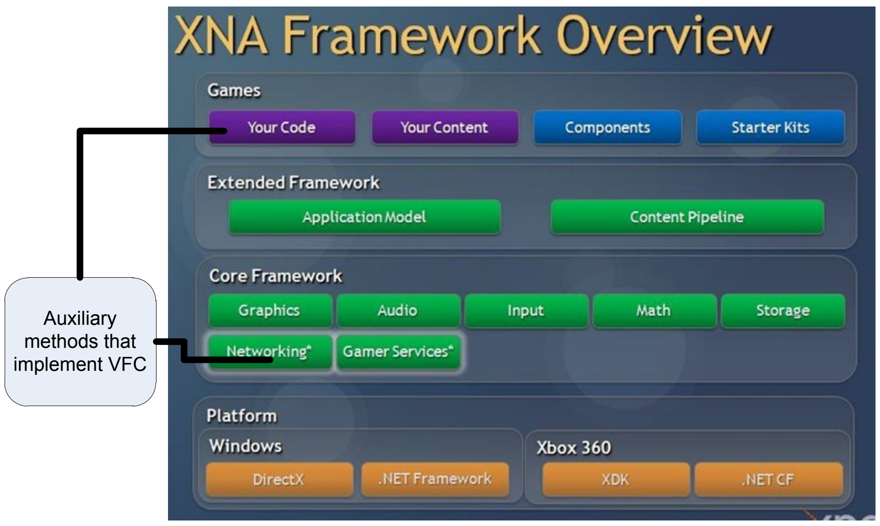


Figure 13 – VFC integration with XNA

Figure 14 shows the adaptation of the trigger mechanism of the VFC coordinator to the context of sending data packets to other nodes.

```
SendData(object, packet)
  FOR each pivot
    IncreaseUpdates(object, pivot)
    IF VFCRequirementsComplied(object, pivot) DO
      SkipToNextPivot
    ELSE
      SendData(pivot, packet)
      MarkRefresh(object, pivot)
    END
  END
```

Figure 14 – Pseudo-code of VFC Coordinator adapted to send packets

3.3.3 Net Rumble Considerations

Net Rumble[3] is a complete XNA Game Studio game available under the Microsoft Permissive License. It consists of a two-dimensional space shooter where the players compete inside an arena filled with asteroids and power-ups. This game can be included in the FPS genre by its network communication requirements as well as type of gameplay. So, it makes it very suitable for the study of the effects of the implementation of VFC since it belongs to the kind of game with the most strict requirements, as seen in 2.3.

Figure 15 illustrates the main class diagram of the classes involved in the gameplay. There are several other blocks of logic in this game, like audio manager, screen manager, which were omitted from the diagram since those are, as should be, independent parts that are not affected by

[3] http://creators.xna.com/en-US/starterkit/netrumble

the consistency model discussed in this work. Some elements, like specific weapons and power-ups were omitted as well for a matter of simplicity since those are only specifications of particular cases of the gameplay elements already present in the diagram.

The network communication in the original game uses a type of communication more common in the RTS genre since it is an all-to-all method. Each player sends all other players its actions and is responsible for the effects of others' actions on him (e.g. each player's responsible for his ship's death). Some parts of the game were already centralized, such as the session control, the asteroids location and movement, and power-up appearance.

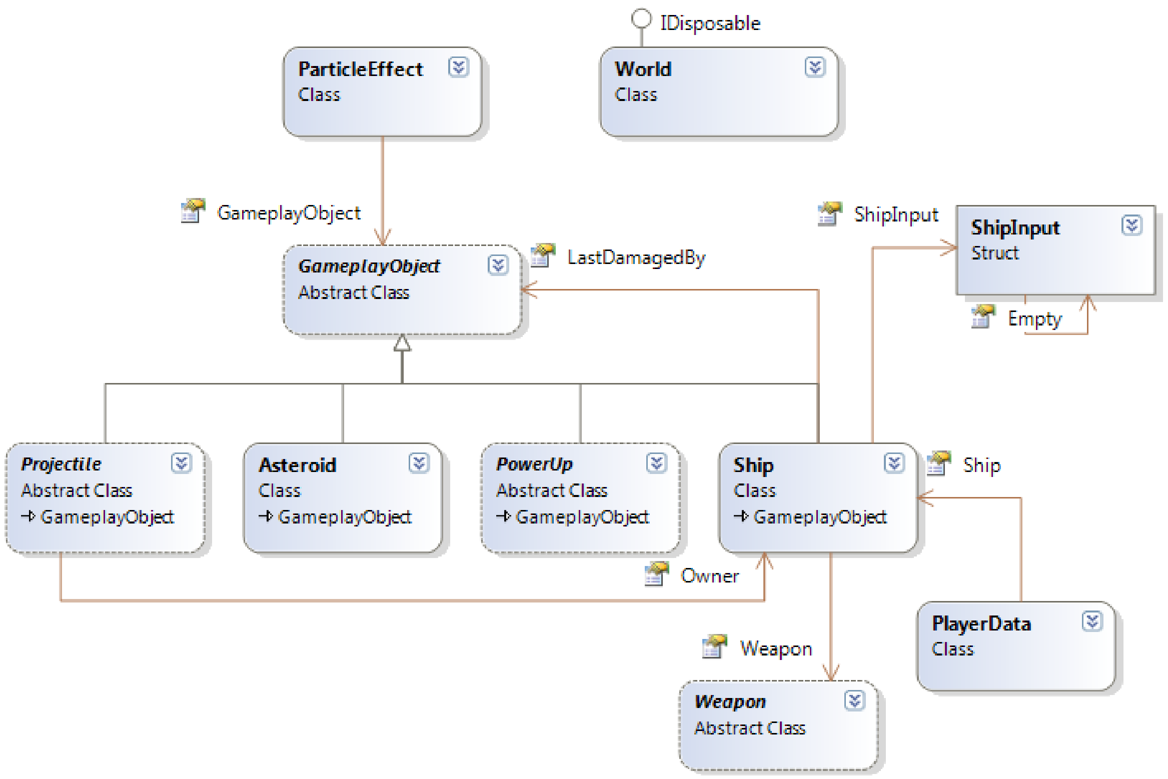


Figure 15 – Main Gameplay Class Diagram of Net Rumble

4 Implementation

In this chapter we present the relevant difficulties and issues that arose during the development, explaining how these were overcome.

Besides the necessary modifications to use VFC as the communication model, several features were added to help or improve the study of the impact of the model.

Section 4.1 describes the details related with the changes that were made in the communication used in the game. Next, in section 4.2, the implementation of the VFC model applied to the Net Rumble prototype are explained. Section 4.3 exposes some cosmetic changes implemented in the game and finally, in section 4.4 we explain the automation and testing process of the prototype achieved.

4.1 Communication

The communication in Net Rumble is done by sending update messages between participants in the game session. For VFC to be enforced some changes in the game communication system were required.

4.1.1 Network Architecture

Since a centralized architecture was desirable we changed *Net Rumble* to a client-server approach. The first issue we ran into here was the need to create IDs for the players because the packets that corresponded to the sender would now all come from the server and previously there was no way to identify players apart from their position in the collection containing all participants. Since this was later implemented by the core XNA framework, we dropped the ID generation process and used the ones provided by the framework.

Then, new message types needed to be added in order for the players to identify the packets containing different types of requests and information. The host needed to receive all players' inputs and, in turn, send it to all participants. These changes were mostly made in the *World* class since that is the class that contains the game-specific logic and code. Some classes like *ShipInput* and *PlayerData* that serialized themselves into packets (to be sent over the network) also needed to be changed.

One issue noted at this stage was that, due to the introduced delay by having to send the input first to the server and only when the message was received back from the server to apply the input given by a certain player, client participants would not accelerate to the maximum velocity. To overcome this issue, a previous player's input was kept in the following steps until a different input was received.

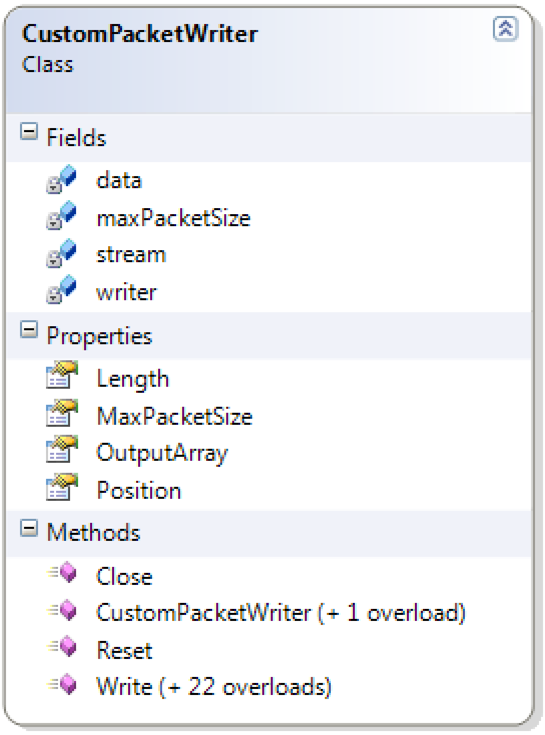
By centralizing the responsibility for the correct state of the game it'd be possible to later make accurate decisions related to locality interest.

4.1.2 Packet Writer

The XNA framework provides a packet writer class, which was used by all communication in this game. This class provides common functionality for efficiently formatting outgoing network packets. One downside in this formatter is that it only allows for it to be sent once: to all participants or to a specific participant because it automatically clears itself after being sent.

To prevent the need to re-construct the full packet for each participant a custom packet writer was created. This class mirrors the behaviour of the packet writer included in the XNA framework without clearing itself after being sent.

Using the packet writer in Figure 16 it was possible to use the rest of the networking methods provided by the XNA framework and support a middle layer that uses group communication.



CustomPacketWriter
Class

Fields
- data
- maxPacketSize
- stream
- writer

Properties
- Length
- MaxPacketSize
- OutputArray
- Position

Methods
- Close
- CustomPacketWriter (+ 1 overload)
- Reset
- Write (+ 22 overloads)

Figure 16 – Custom Packet Writer

4.2 VFC Model

The implementation of the VFC model is made as a module that can be easily integrated and detached of a similar game. It abstracts from the specific game logic in which it was implemented.

Figure 17 depicts the integration of the VFC implementation for the game Net Rumble.

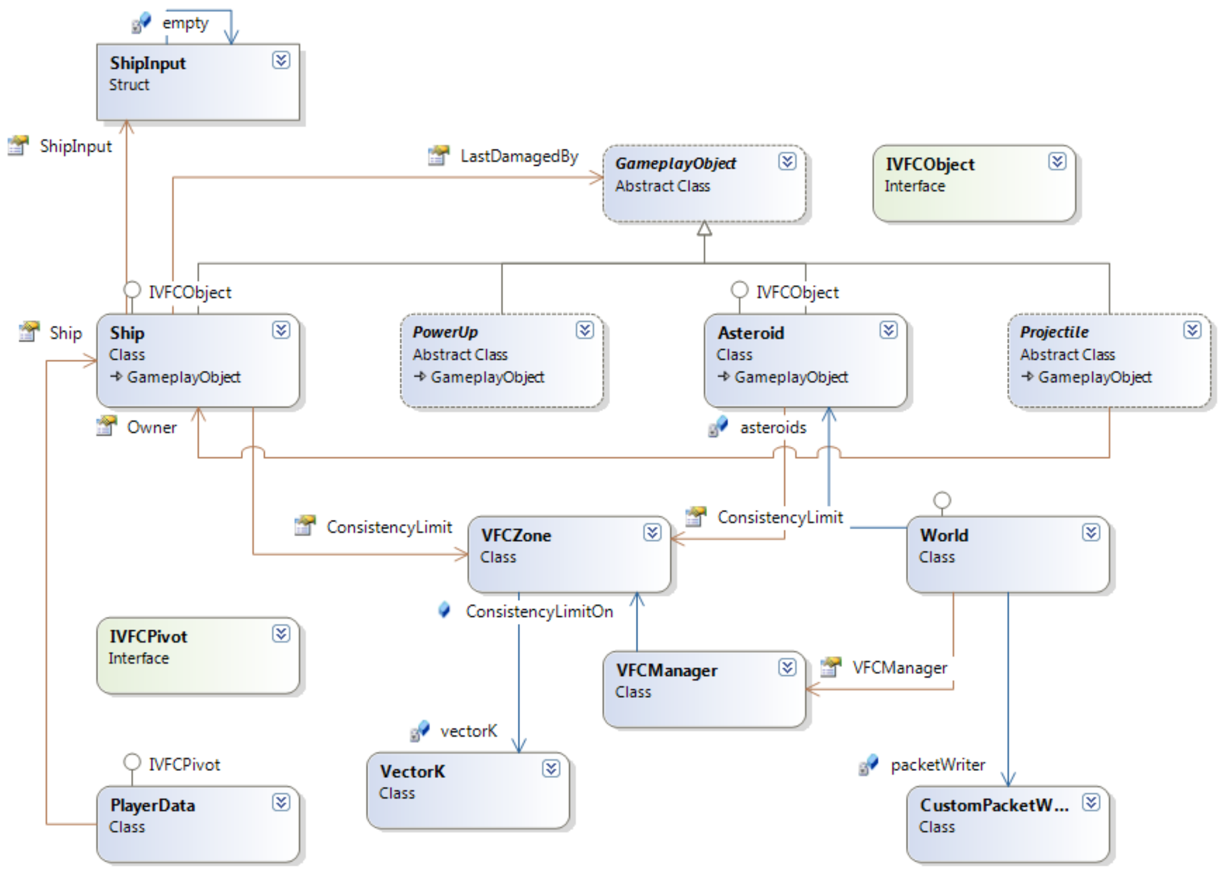


Figure 17 – VFC integrated in Net Rumble

4.2.1 Pivot

Pivots are the centre of the consistency zones for each player. Since each player only controls one ship the pivot is given by the ship's position. In XNA's Gamer Services system, each gamer has a *tag* object to which the programmer can add required information (e.g. ship colour). The class that represents that information (*PlayerData*) then implements the *IVFCPivot* interface for the middleware to be able to extract its position.

In addition to the position the interface also requires an ID to be defined so it can later identify each pivot in the lists of maintained consistencies.

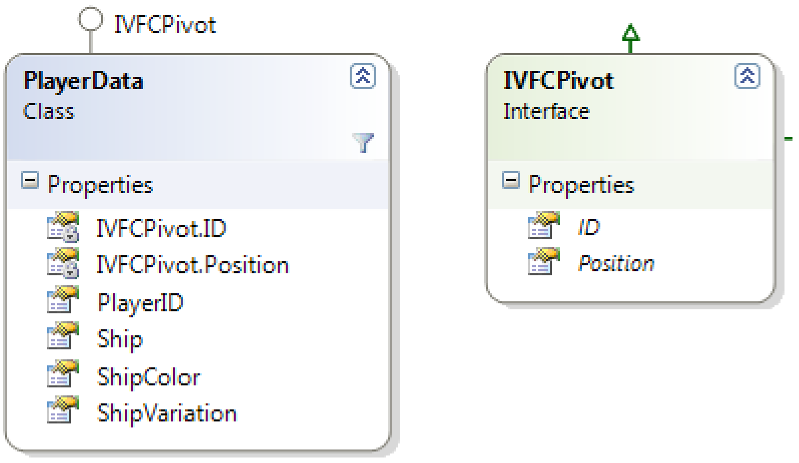


Figure 18 – Implementation of VFC pivots

4.2.2 Objects

Each object, for which updates are to be managed using VFC, implements *IVFCObject* interface. The object has a variable that specifies the consistency limits, made of an array of VFC zones. Furthermore, it contains a dictionary with the status vector for each player which is managed by the VFC middleware. This was done to encapsulate the necessary information, avoiding unique IDs for all objects. The logic that gives the current *value* VFC dimension is defined in the implementation of this interface as well.

As shown in Figure 19, in this game the VFC objects are ships and asteroids.

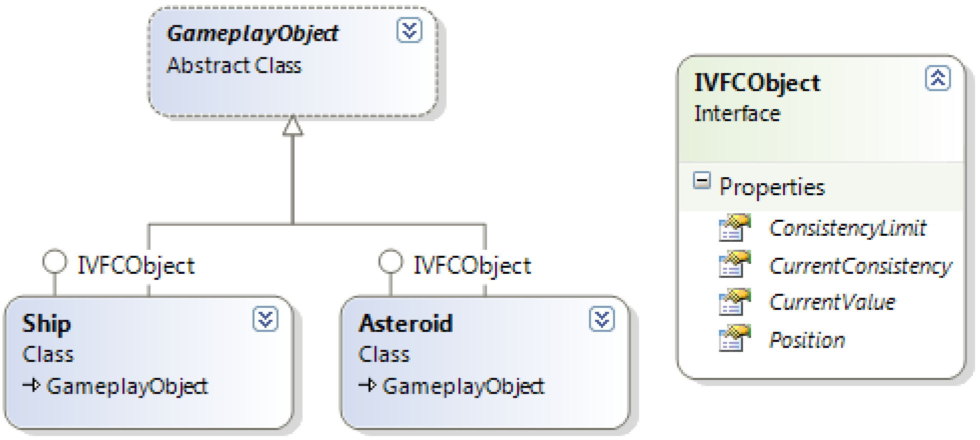


Figure 19 – Classes related to VFC objects

4.2.3 Dimensions

An essential procedure to use the VFC model is to define the logical dimensions to be used by the algorithm.

To calculate the distance of objects and pivots, its positions in the virtual world were used. The distance then translates into a specified consistency zone.

Time and sequence dimensions of the consistency vector correspond to the number of updates of each object and the time it was last updated. For the value dimension, which is a qualitative dimension, the length of the velocity of each object was used.

4.2.4 Consistency Enforcement

In this implementation the consistency is enforced in the method used to send packets to all players. This is encapsulated in the class *VFCManager*. The logic implemented in this class is the flow presented in Figure 14. There is a particular case worth noting in this implementation, which is that when a player's pivot has no position (e.g. ship is dead), the zone with the most tolerant consistency requirements for the object currently considered. Since a two-dimensional space is being considered here, the distance between pivots and other objects is given by $\sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$.

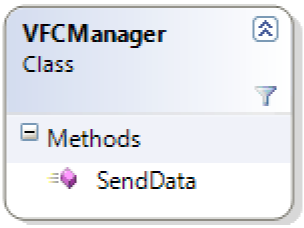


Figure 20 – Class that encapsulates consistency coordination

4.3 User Interface

In order to simplify testing and improve the capacities of the game some functionalities were added. The most relevant ones are described next. Some minor errors in the game were detected and corrected, e.g. a bug in the collision mechanism.

4.3.1 Zoom

For testing purposes, to be able to analyse the effect of the different consistency zones of VFC it was important to allow the game camera to be zoomed so, a menu option was added to allow the user to choose the desired scale in which the game was to be played. Figure 21 shows the game with the original appearance and Figure 22 shows an example drawing the world at a 40% scale.

This also allowed us to observe the effects of the VFC model in real time as the objects moved across different consistency zones.

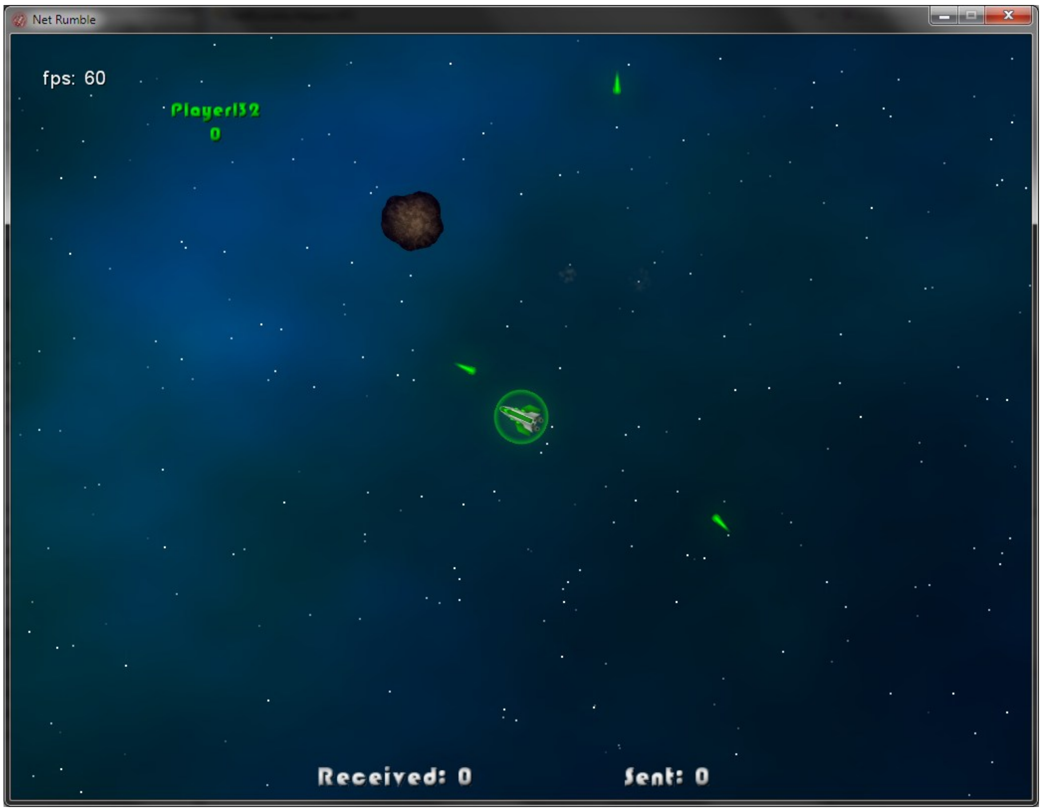


Figure 21 – Net Rumble with original size

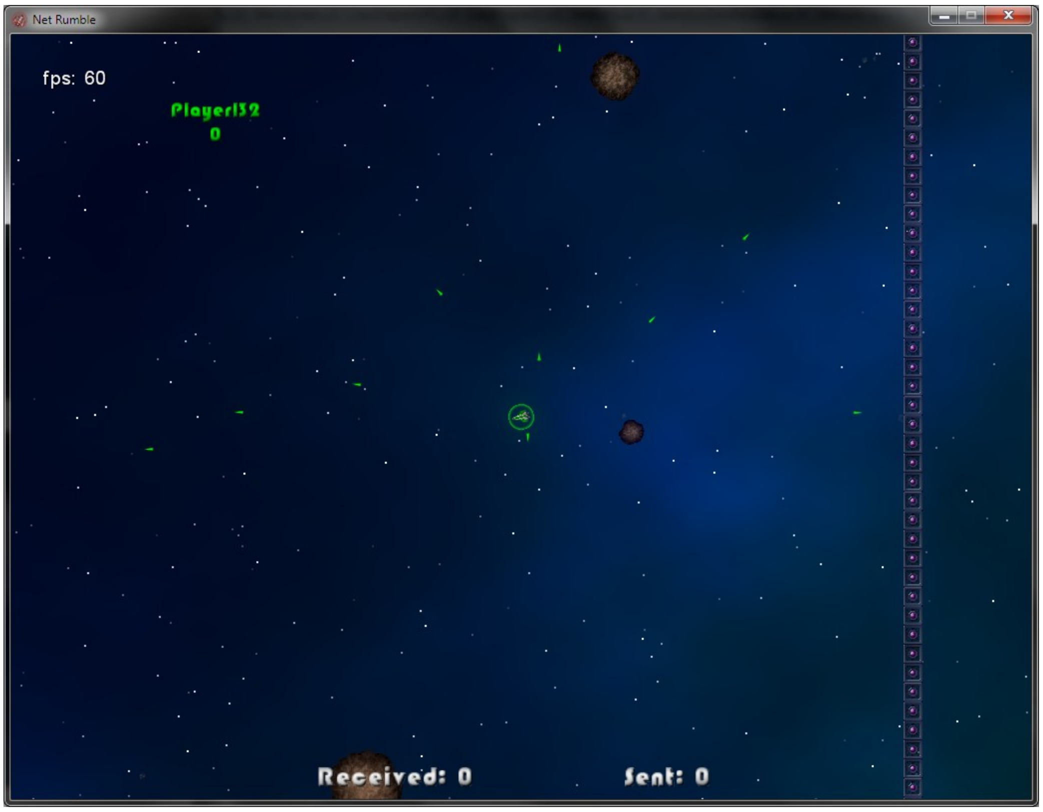


Figure 22 – Net Rumble zoomed out at 40%

4.3.2 Consistency zones

Another interesting visual addition was the identification of the zone fields, done by drawing the circles around the player's ship according to the zones' radius. With this component added to the game it was very easy to track when a certain object in the game (enemy ship or asteroid) changed its consistency requirements and then monitor the difference in the behaviour.

Figure 23 shows a large portion of the arena showing several players interacting located in different zones with distinct consistency requirements.

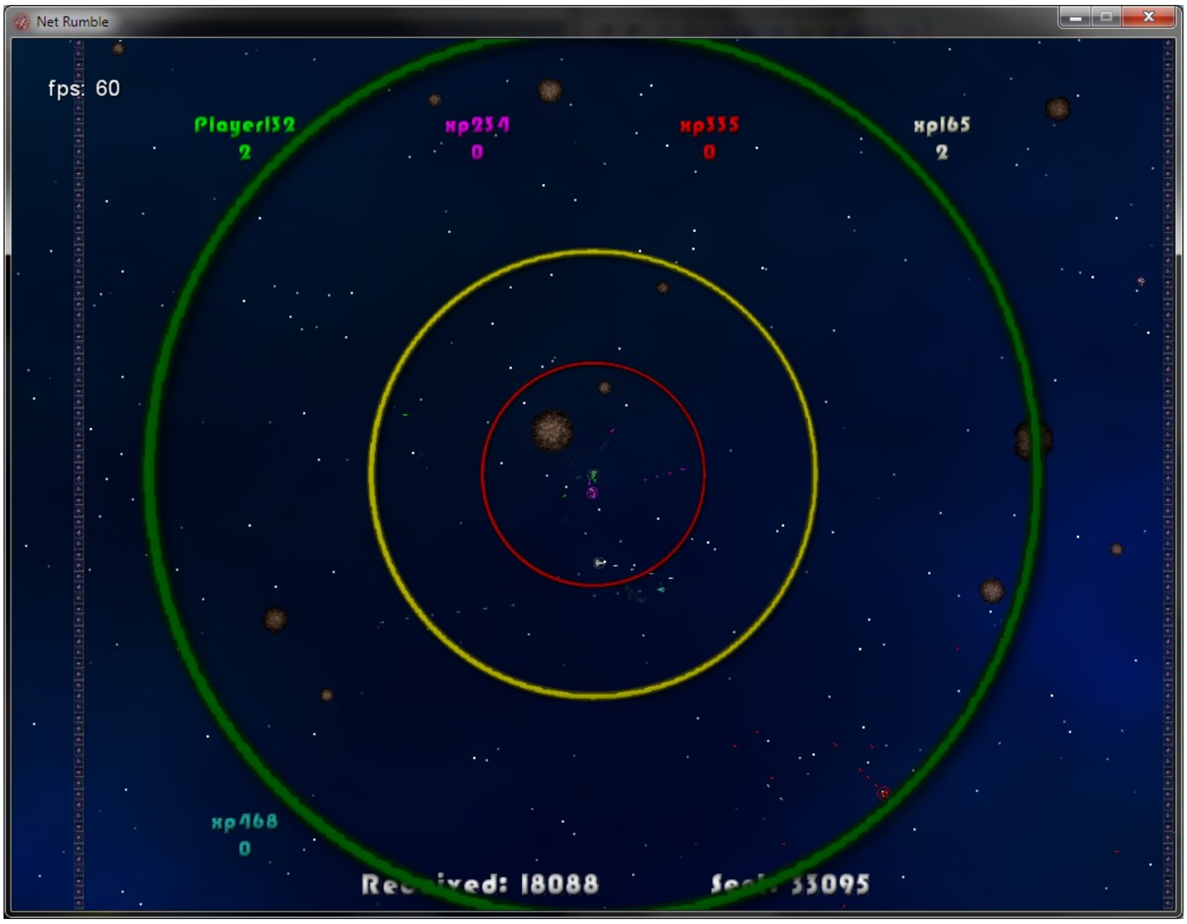


Figure 23 – Consistency zones drawn around the player

4.3.3 Frame-rate

A component to measure and show the frame-rate during the gameplay was added to the game. The extra weight of the VFC model implemented should not degrade performance of the game. The current frame-rate calculated is shown on the top left corner throughout the game.

4.3.4 Options Menu

To aid in the testing of different conditions of the network an options menu was created so we can choose among a variety of configurations to run the game with. Most of these options only needed to be changed on the server to create different conditions for the game to be run with. Figure 24 presents a diagram with this options menu integrated with other menu screens of Net Rumble.

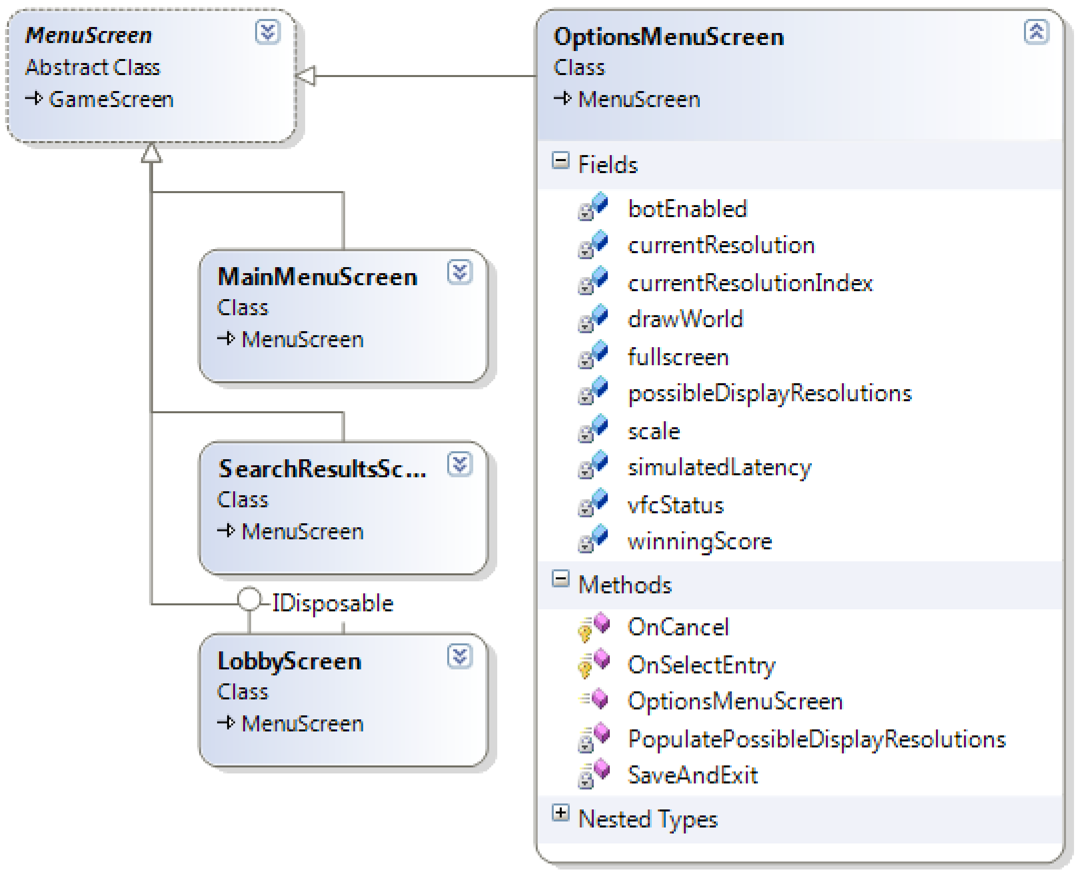


Figure 24 – Class diagram with the added Options Menu Screen

4.4 Automation and Testing

To conduct the tests and be able to automate players' and gather data about the gameplay in order to further study the behaviour of the VFC model along the game.

4.4.1 Automated Player Simulation

The need for automation of the ships controller was evident when trying to test the game. This led to the development of some basic artificial intelligence used to control the ships. To simulate the input the terminal would just follow some basic logic to keep wandering inside the arena and shooting randomly until an enemy ship approached. Then, it enters a pursuit mode in which it tries to eliminate the opponent.

Each bot may be in one of three states, in reverse order of priority:

1. Wandering – randomly navigating the arena and shooting in random directions;
2. Getting a Power-up – when the ship gets near a power-up item it tries to get it;
3. Pursuing an enemy – when another ship gets in a certain radius of the ship it enters pursuit mode in which firing and movement are directed at the opponent;

By ensuring the ships would constantly move and change their actions this certainly stresses the communication as well, or better, than a human player.

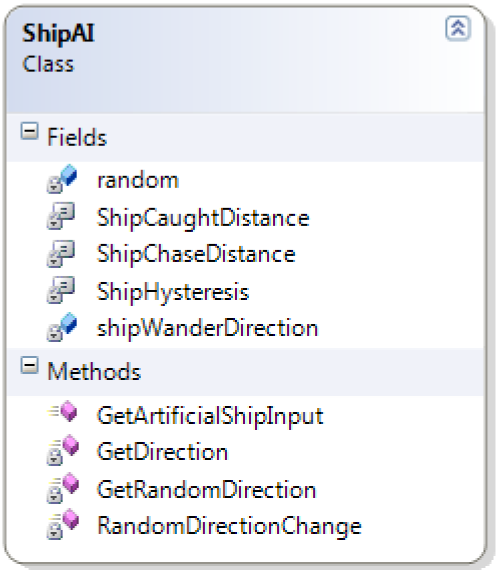


Figure 25 – Class that simulates player input

4.4.2 Record game data

During the gameplay, the necessary data is recorded in CSV format in order to later process the meaningful information to elaborate on the impact of VFC on the game communication mechanism.

The data recorded include elapsed time, players' positions, and average amount of data being sent and received. This enables a further study of the divergence that occurred. It also allows a deeper analysis of the effect of the VFC model on the performance and consistency.

Regarding the elapsed time, a discrepancy in the time between participants was naturally noticed due to clock synchronization and network delay. When comparing data recorded in different clients there would be offsets between events. This was studied further and concluded that it introduced an error in the divergence calculated between clients and server. To minimize the effects of this error the data was adjusted and re-synchronized on the first recording each time a player re-spawned. Since this error happens with and without the use of VFC the data is still perfectly valuable for a comparison of the effect of VFC in the communication mechanism of the game.

4.4.3 Deployment

There is a XNA limitation to run only one game at a time on each terminal. This diminishes considerably the number of participants for testing purposes. To circumvent this, the VMware Workstation is used to increase the number of participants in each session. With this virtualization and simulating adverse network conditions it is then possible to examine, or at least speculate, the behaviour of the game in actual conditions (e.g. over the Internet). The network connection for the virtual machines was bridged to its host. With this connection type VMware uses the physical interface of the host to directly emulate the virtual interface in the virtual machine. This allows virtual machines to communicate with other nodes on the network and is transparent to the applications running on the host. All have to use the network as if they were different physical machines.

5 Evaluation

This chapter presents the methods we used in the evaluation process and the results obtained in the experiments. The evaluation is done both qualitatively and quantitatively. In section 5.1 we explain the method and elements of evaluation of the system. Section 5.2 presents the results that were obtained in the evaluation of the system.

5.1 Evaluation Method

There are several methods for evaluating the effect of the implementation of VFC on a game. In this chapter it is discussed what the relevant criteria are, and how they should or should not be affected. These range from accordance to the game rules to the bandwidth consumed in the course of the game.

5.1.1 Bandwidth

Given that the comparison made to a model that does no filtering of messages and sends everything to all players, the use of VFC should greatly reduce the bandwidth used in the course of the game.

5.1.2 Latency

The latency has an important role on the gameplay experience of players. For a given pivot, the latency grows with the virtual distance to it. It is important to measure the latency in each zone and compare it with the latency on without using VFC. It would also be interesting to possibly reduce the average latency.

5.1.3 Frame-rate

The calculations needed to use the VFC model should be as light as possible and should not have a negative impact on the frame-rate possible. Preferably, by reducing the number of messages that need to be processed by each terminal, it should be possible to increase the frame-rate to improve user experience.

5.1.4 User experience (playability)

User appreciation is an important factor on the evaluation of the work since all attempts to improve game communication have an essential goal: to improve user experience. The impact that the use of VFC has on the gameplay is an important study object.

An experiment related to this is to simulate network conditions and compare the latency and packet loss that can be sustained by VFC.

5.1.5 Compliance with game rules

It is crucial for the game logic not to be modified and the rules must still be obeyed, even if some divergence is *temporarily* allowed.

A major factor is that each player has, at least, enough information that does not affect negatively his decisions. It should as well have no effect on the score in the game, not favouring even the host player.

5.1.6 Simulated conditions

Since the tests take place on a local area network, with high throughput, there is a need to simulate adverse network conditions. The XNA Framework is able to simulate latency and packet loss which can be very useful in order to study the behaviour of the game with the changes made in the communication layer.

5.2 Results

The tests were run on two machines: a desktop computer with a quad-core AMD Phenom™ II X4 945 Processor 3.00 GHz, with 8 GB of RAM, running Windows 7 64-bit OS; a laptop computer with a Intel® Core™2 Duo T6400 at 2.00 GHz with 4 GB of RAM. Each virtual machine had 512MB of RAM allocated and access to one processor and had Windows XP 32-bit OS. All virtual machines had to have Microsoft® Visual Studio® 2008 and XNA Game Studio 3.1 installed, both needed for the use of networking provided by the XNA framework. The host player was run on the desktop machine and clients on virtual machines both on the desktop and on the laptop. The machines, even the virtual machines (which was bridged to the physical computers), are connected through a 100 Mbps network.

Multiple tests were run with each configuration to verify the integrity of the data acquired during gameplay.

The data regarding network and players' positions was recorded at a rate of about 5 samples per second (0.2s intervals).

The tests were run without the use of VFC, transmitting all packets, and with a VFC configuration that uses the definitions on Table 3 for the consistency vectors.

	Range	Time (s)	Sequence	ValueDiff
Zone1	500	0	0	0
Zone2	1000	0.3	5	2
Zone3	2000	0.5	10	5
Zone4	5000	1	20	10
Zone5	∞	3	40	20

Table 3 – VFC Zones with strict constraints

5.2.1 Bandwidth

The relevant data for the study of bandwidth usage are:

- Data received by each client
- Data sent by server

The data presented here refers to specific cases that can be used as examples for typical game progress. Surely there are variations in the traffic observed depending on the actual game session but these negligible in the comparisons made.

Since there is no filtering on the messages that the client sends to the server there is no interest in analysing neither the data sent by clients nor data received by the server. These, as expected, are not affected by the usage of VFC. The maximum data sent by each client is around 4,5kBps.

With only two participants in the game, the host and a client, the data sent by the server and the data received by the server is approximately the same, as expected. This can by comparing Figure 26, showing the amount of data sent by the server, and Figure 27, which illustrates the amount of data received by the client throughout the game. For the server the maximum is 7338 Bps with an average of 6758 Bps sent, whereas for the client the maximum is 7243 Bps and the average 6750 Bps received. The small difference in the amount of data sent versus the received may be explained by out of order packets dropped on the client. Another factor that may influence this slight difference is the underlying performance counter provided by XNA framework. As seen in both figures, the data received by a client (and sent by the server) throughout the game, without the use of VFC, is practically constant.

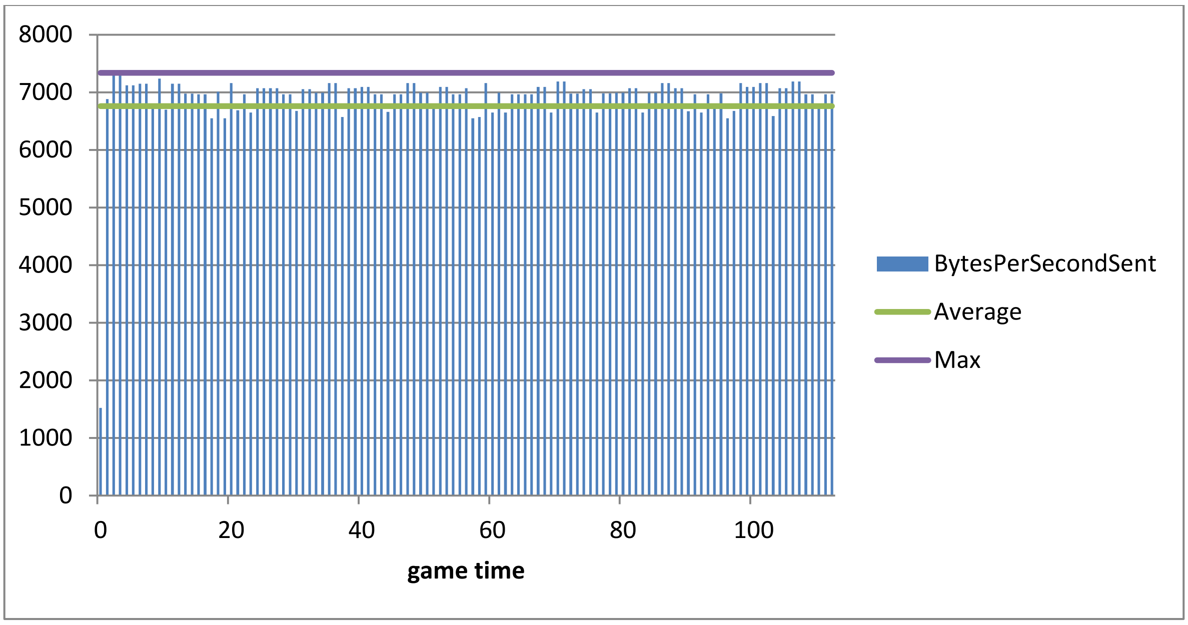


Figure 26 – Bps sent by the server without VFC – 2 players

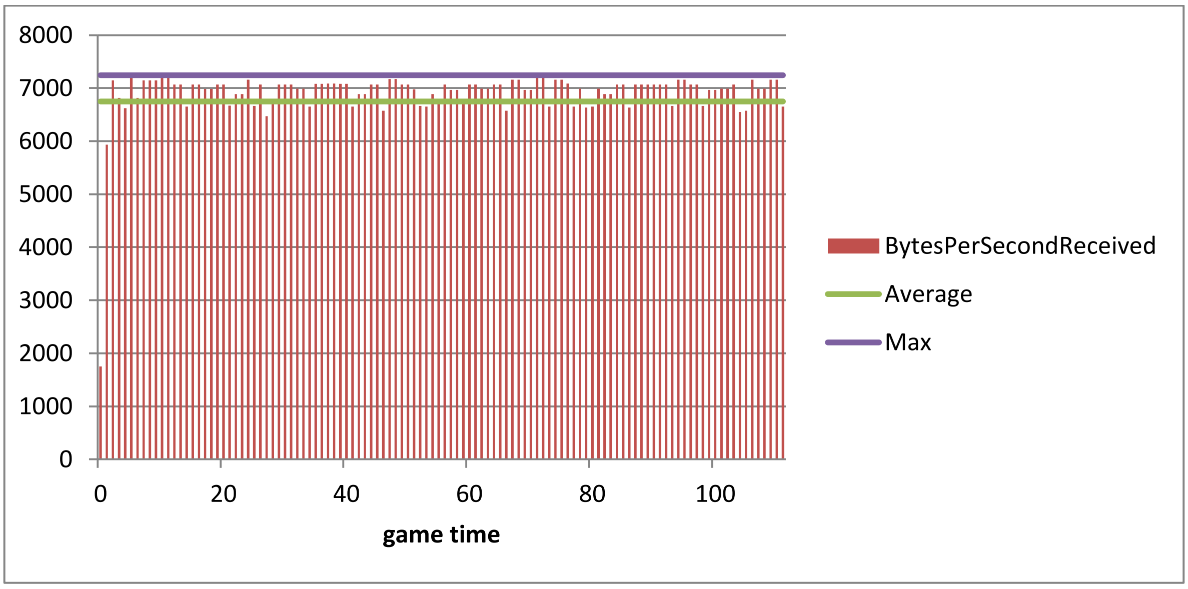


Figure 27 – Bps received by the client without VFC – 2 players

Still with 2 players, but now using our implementation of VFC with the strict approach (see Table 3), Figure 28 shows the amount of data being sent by the server. A clear difference is noticeable. There is a slight decrease in the maximum amount of data being sent to 7142 Bps (2,7%) but the difference in the average data sent is considerable. There is a decrease of 23% to 5046 Bps. Since when the players' ships come close enough (inside the first consistency zone) the amount of data sent is the same as when not using VFC. The only difference in this scenario is the data that regards the asteroids which are in different consistency zones.

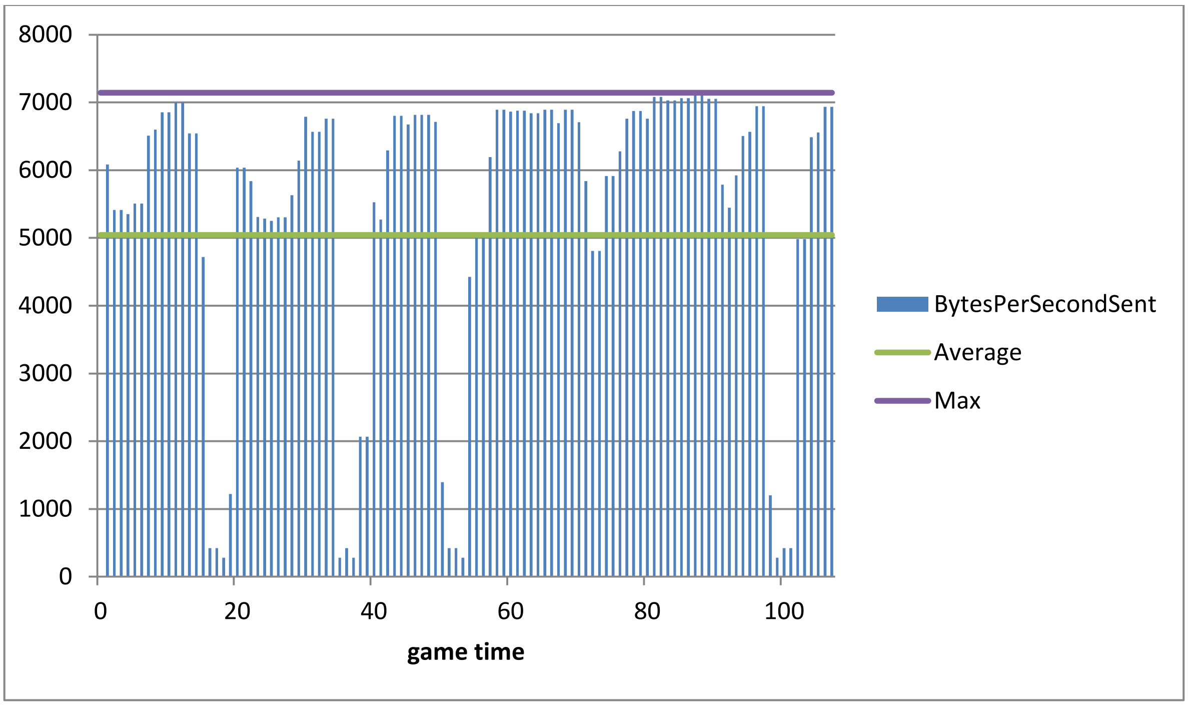


Figure 28 – Bps sent by the server with VFC – 2 players

When considering three players in the game, without the use of VFC, the server sends an average of 16802 Bps with a maximum of 17914 Bps, throughout the game. Each client receives an average of 8391 Bps and a maximum of 8953 Bps.

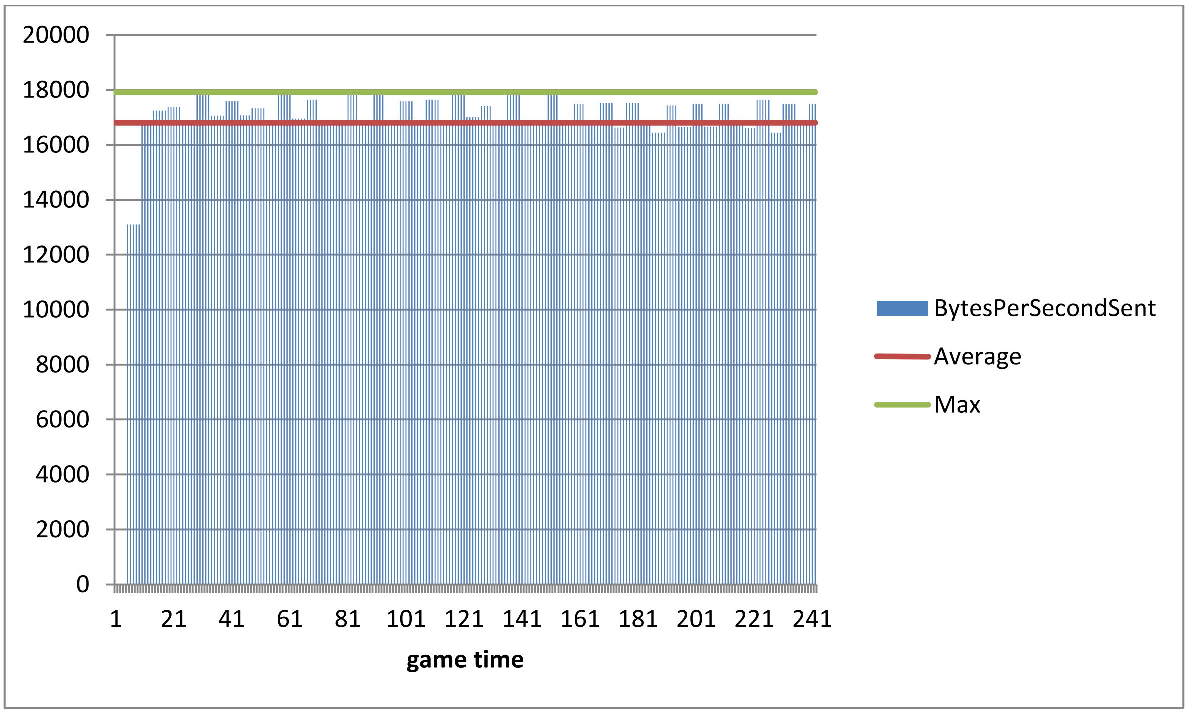


Figure 29 – Bps sent by the server without VFC – 3 players

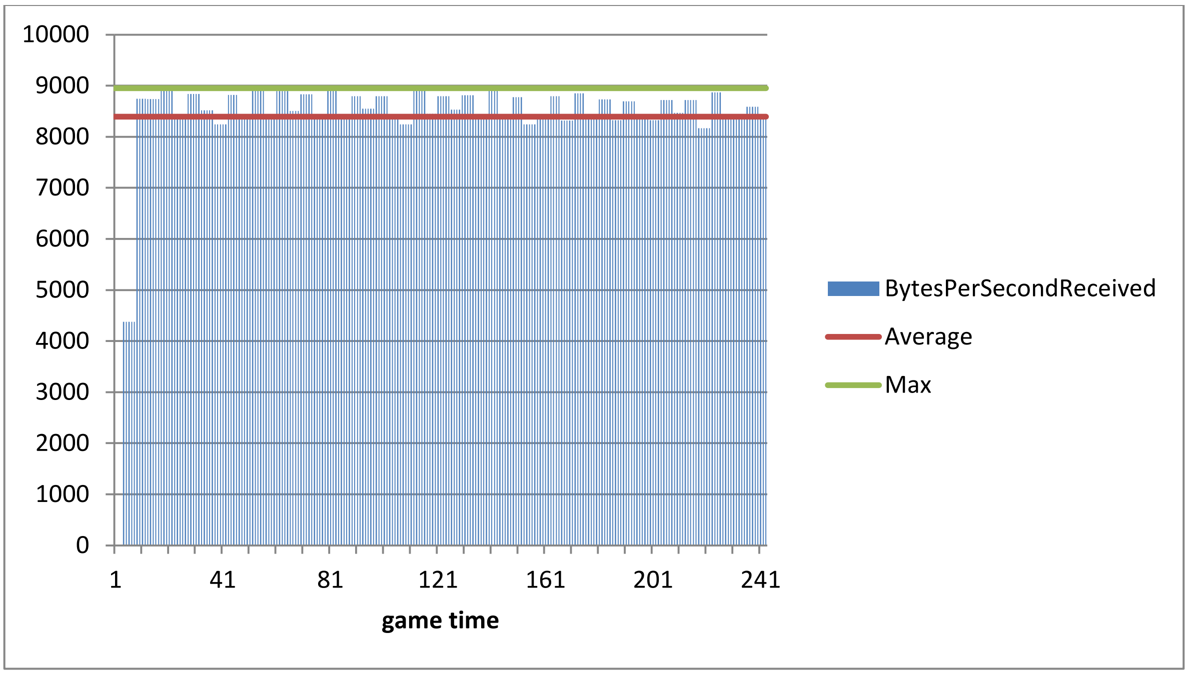


Figure 30 – Bps received by a client without VFC – 3 players

Using VFC, the average amount of data sent by the server is decreased to 10086 Bps with a maximum of 15242 Bps. This denotes a reduction of 40% of the traffic generated by the server.

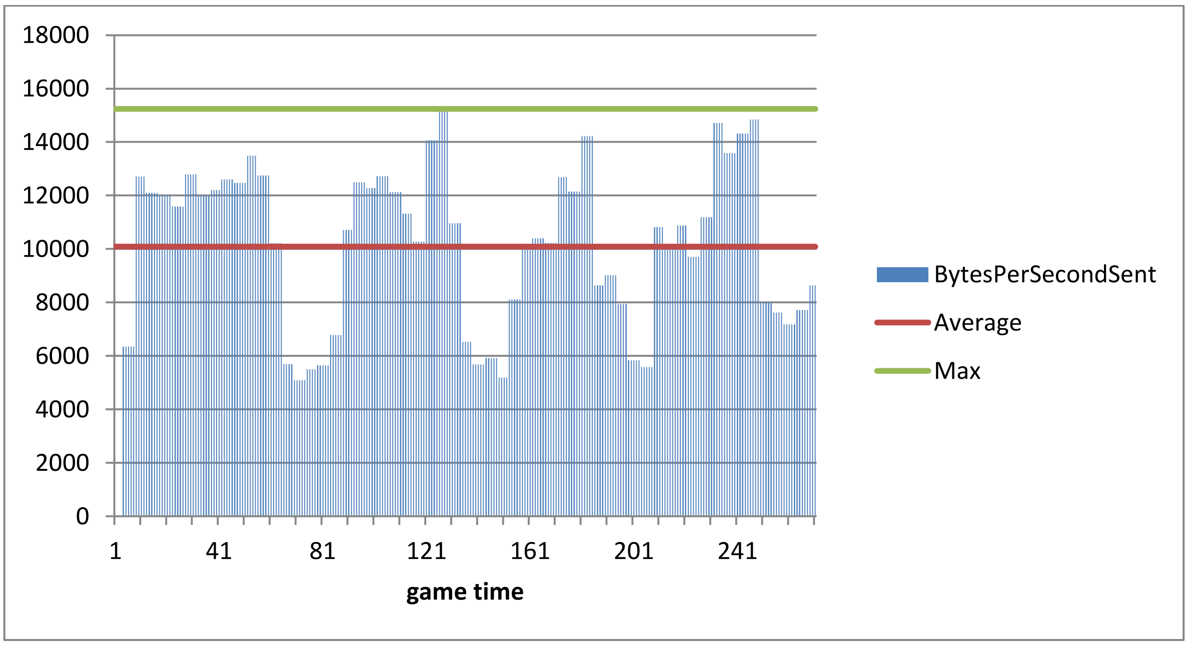


Figure 31 – Bps sent by the server with VFC – 3 players

When considering the client side of the communication channel, Figure 32 shows an example of the amount of data received by a client. Analyzing the graph it is clearly suggested that this player's ship was never killed since there is never a plunge usually noted when that happens, and justified by the fact that its pivot is considered to be in the most tolerant consistency zone. The average traffic in this case was 5803 Bps with a maximum of 8787 Bps. This can be considered somewhat a worst-case scenario and even so the decrease in the average amount of data received was about 31%. The maximum dropped only 2% because it is still very likely for all three players to be inside the most strict consistency zone at the same time frame.

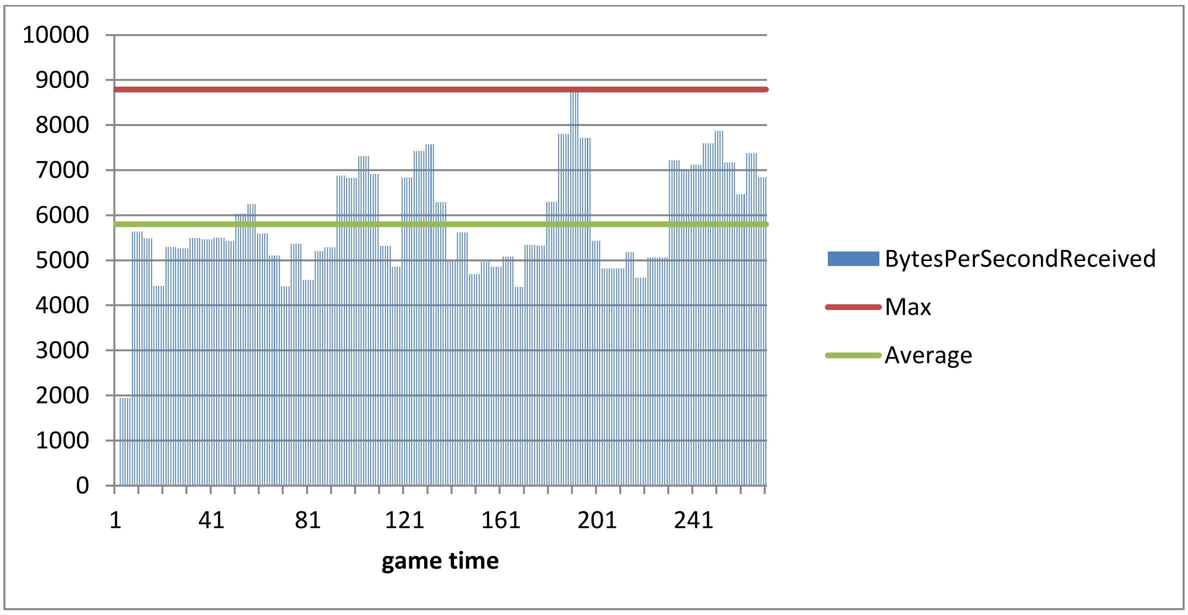


Figure 32 – Bps received by a client when using VFC – 3 players

The following graphs, besides showing the amount of data exchanged during the game, also provide a mean to identify in-game conditions like proximity of the players, number of players in the re-spawn period and the number of deaths of the ship represented, in the case of graphs representing the amount of data received by a client.

Without using VFC, when the game involves four participants, the average amount of data being sent by the server is 29869 Bps, with a maximum of 32247 Bps. Each client receives about 9997 Bps in average and a maximum of 10919 Bps.

Using VFC, the server sends an average 16205 Bps (46% less) and a maximum of 26643 Bps (17% less) throughout the game. The values for the client are dropped to an average of 5656 Bps and a maximum of 10165 Bps. This represents a decrease of 43% and a 7% lower maximum of the amount of data received.

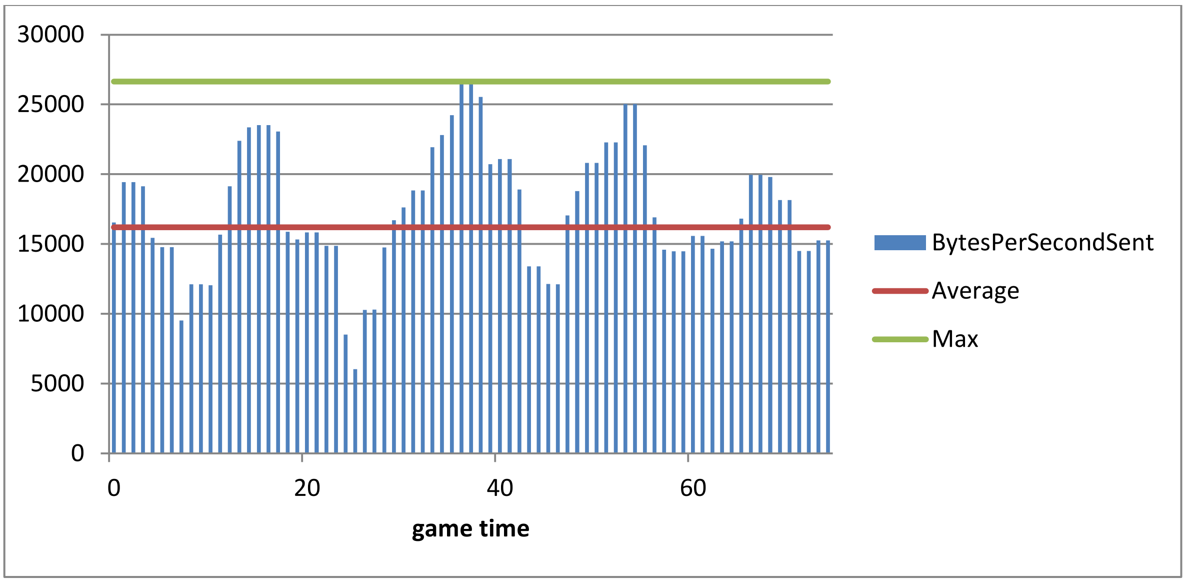


Figure 33 – Bps sent by the server with VFC – 4 players

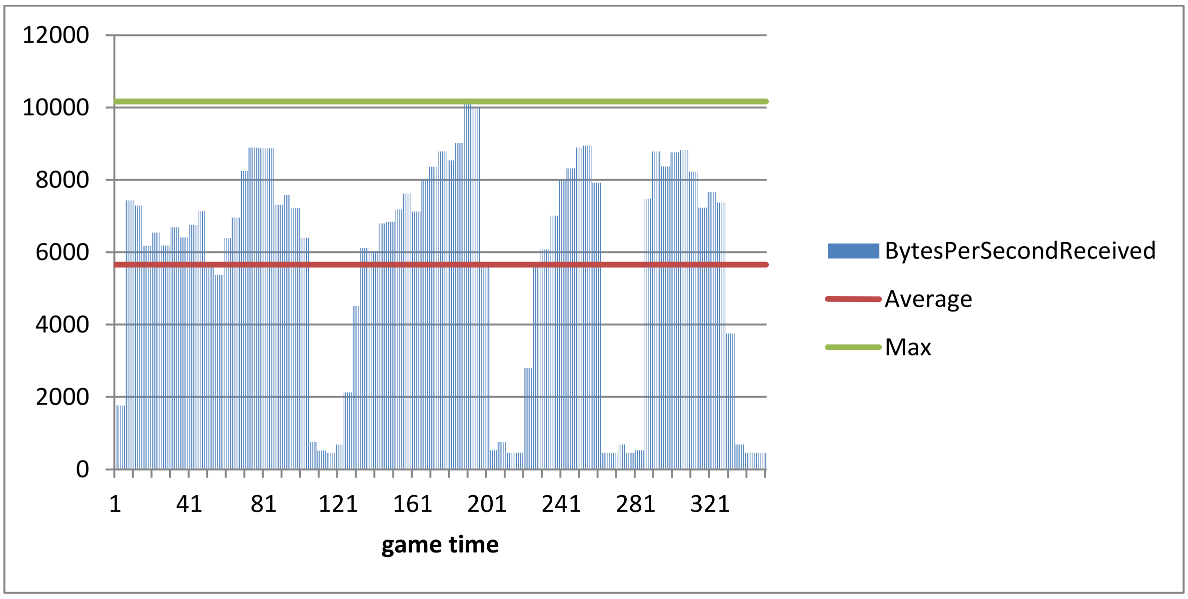


Figure 34 – Bps received by a client using VFC – 4 players

Without using VFC, with five participants in the game, the average amount of data being sent by the server is 49238 Bps, with a maximum of 51440 Bps. Each client receives about 12687 Bps in average and a maximum of 12687 Bps.

Using VFC, the server sends an average 23895 Bps (51% less) and a maximum of 39838 Bps (17% lower) throughout the game. The values for the client are dropped to an average of 4639 Bps and a maximum of 10383 Bps. This represents a decrease of 62% and an 18% lower maximum of the amount of data being received.

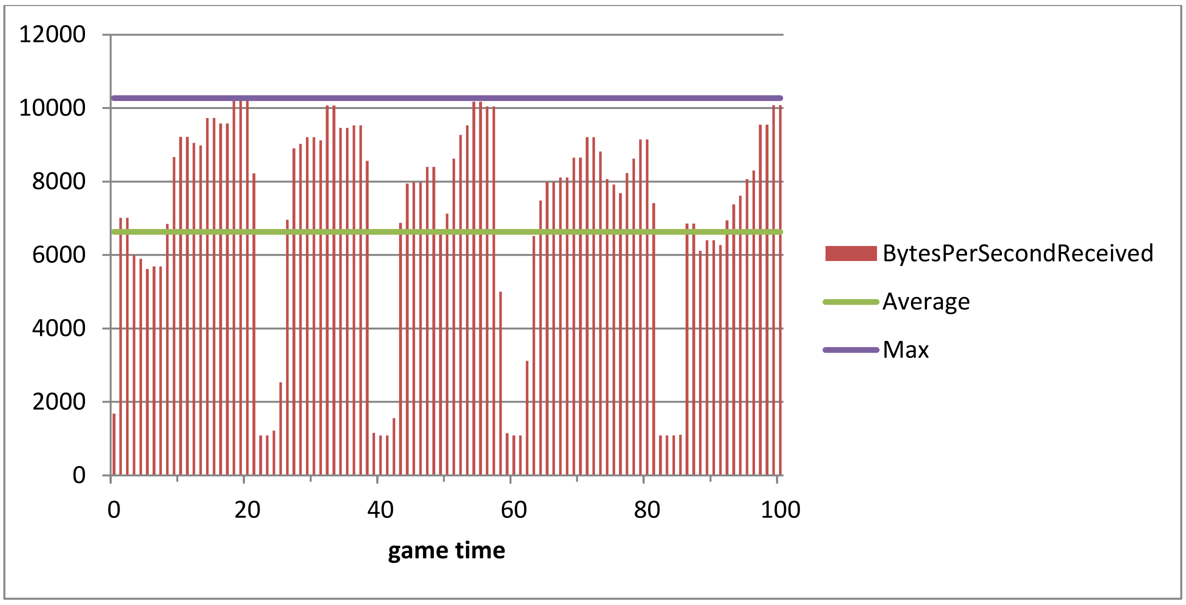


Figure 35 – Bps received by a client with VFC enabled – 5 players

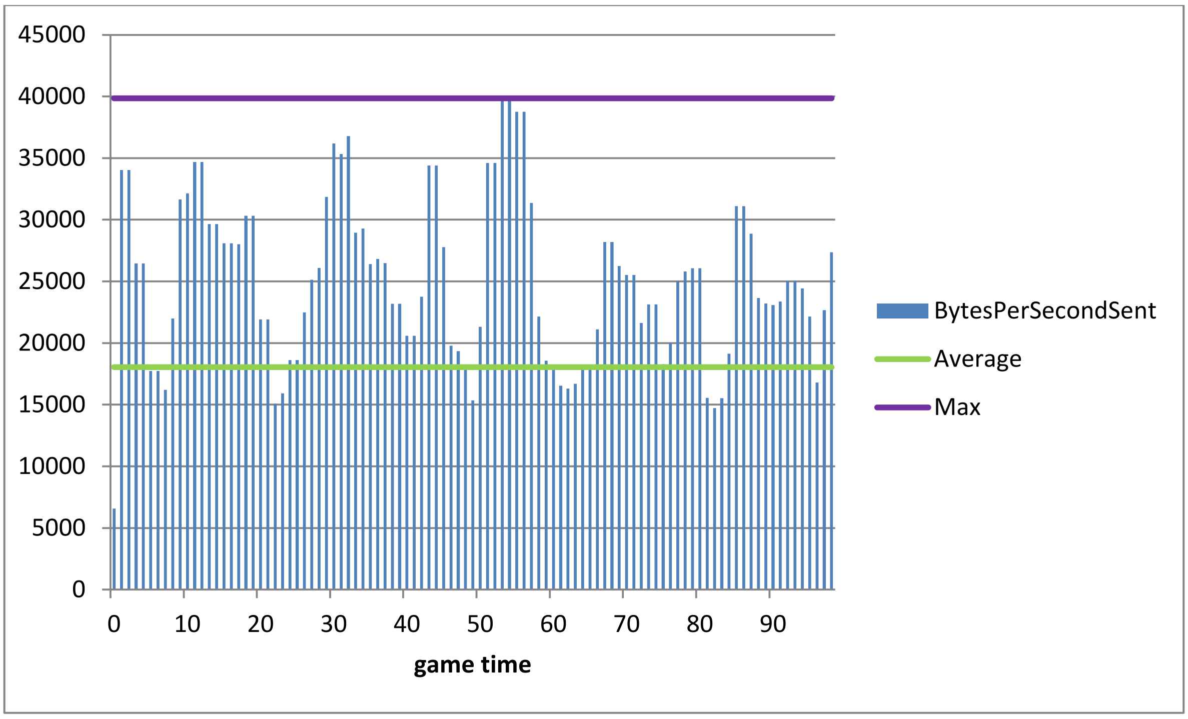


Figure 36 – Bps sent by the server with VFC – 5 players

For a study case with ten participants, the server sent an average 180 kBps and a maximum of 190 kBps, without using VFC. When using VFC, there was a 64% reduction to an average 65 kBps and a 46% drop of the maximum rate to 103kBps. On the client side there was 65% reduction of the average rate (from 20kBps to 7kBps) and a 30% lower maximum (from 22kBps to 15kBps).

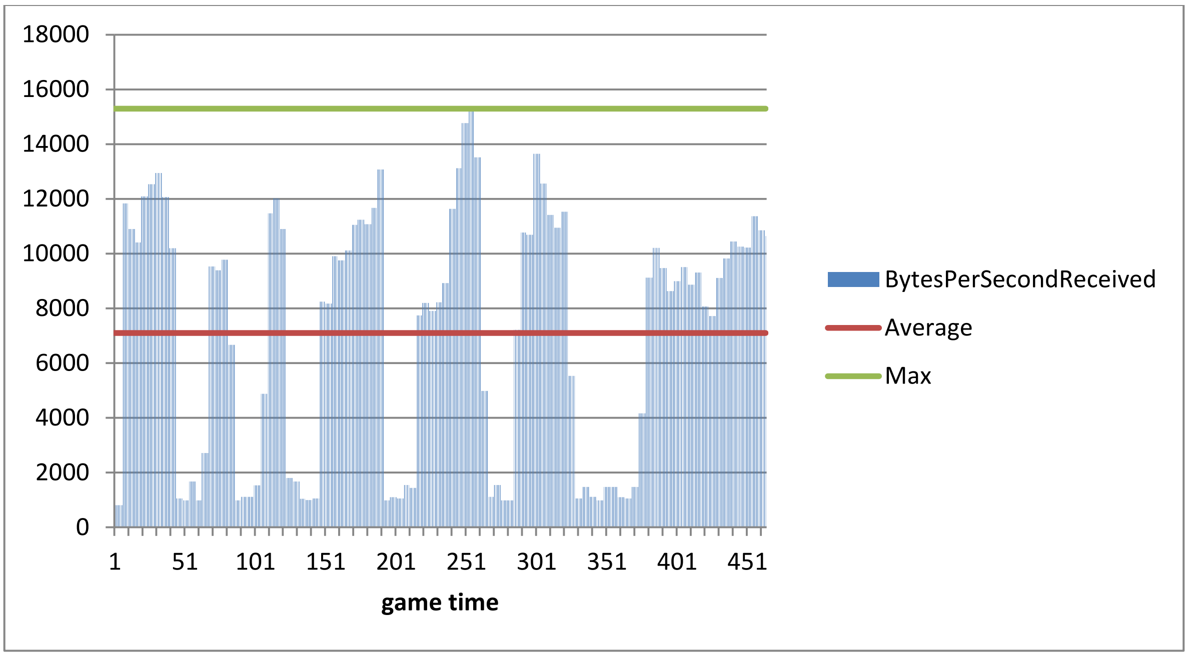


Figure 37 – Bps received by a client using VFC – 10 players

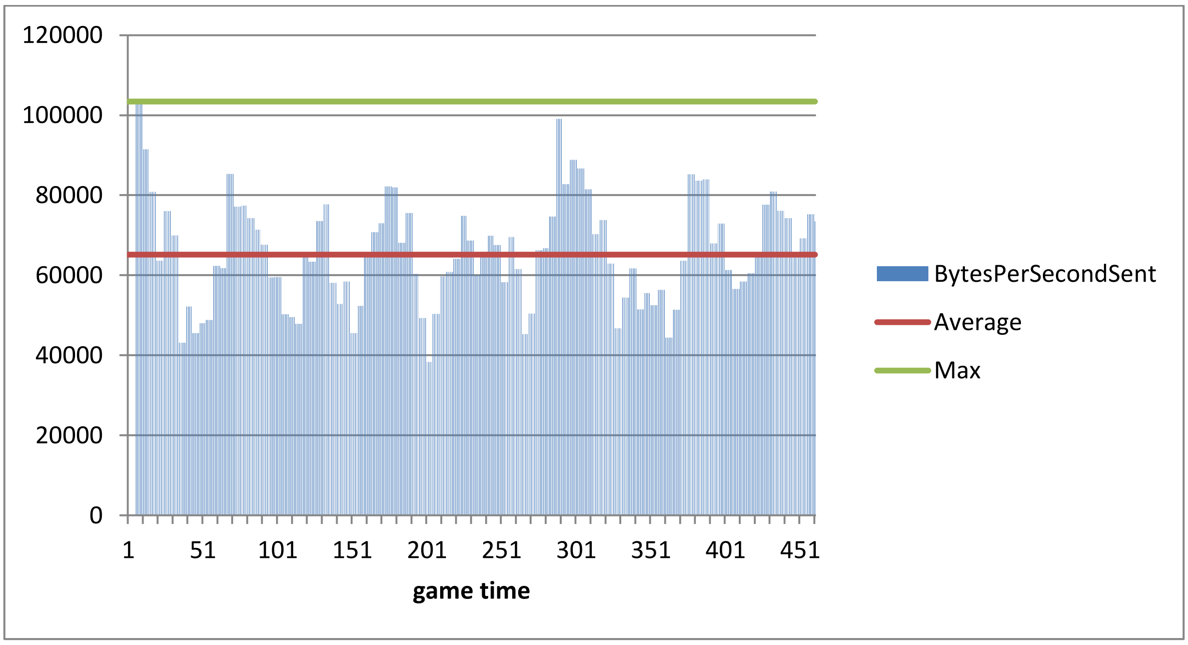


Figure 38 – Bps sent by the server with VFC – 10 players

As summarized in Table 4 for the server and in Table 5 for clients, the reduction of the amount of data transferred throughout the game is significantly decreased with the use of VFC.

# players	2		3		4		5		6		10	
	Avg	Peak	Avg	Peak	Avg	Peak	Avg	Peak	Avg	Peak	Avg	Peak
All (Bps)	6758	7338	16802	17914	29869	32247	49238	51440	66663	71440	180k	190k
VFC (Bps)	5046	7142	10086	15242	16205	26643	23895	39838	30227	43132	65k	103k
% of traffic	75%	97%	60%	85%	54%	83%	49%	77%	45%	60%	36%	54%
Reduction (%)	25%	3%	40%	15%	46%	17%	51%	23%	55%	40%	64%	46%

Table 4 – Comparison of amount of data sent by the server

# players	2		3		4		5		6		10	
	Avg	Peak	Avg	Peak	Avg	Peak	Avg	Peak	Avg	Peak	Avg	Peak
All (Bps)	6750	7243	8391	8953	9997	10919	12303	12687	13316	14486	20085	21784
VFC (Bps)	5049	7142	5802	8787	5656	10165	4639	10383	5677	12112	7097	15296
% of traffic	75%	99%	69%	98%	57%	93%	38%	82%	43%	84%	35%	70%
Reduction (%)	25%	1%	31%	2%	43%	7%	62%	18%	57%	16%	65%	30%

Table 5 – Comparison of amount of data received by a client

Table 6 presents approximate values for the amount of data received by a server depending on the number of participants in the game. An estimate of the amount of data received by a server in this game, depending on the number of players is given by: $(N - 1) * 4500$. Note that the server (i.e. host) counts as a player but does not generate network traffic to the server. This applies to all tests since the logic run on the client side is always the same.

# players	2	3	4	5	6	10
Average (Bps)	4350	8797	13151	18038	22285	40436

Table 6 – Amount of data received by the server depending on the number of participants

5.2.2 Divergence

Measuring the divergence on the clients turned out to be very complex. The game time has an intrinsic error among the participants in the game. This introduces a divergence in the positions between the data recorded in the host and other participants. The divergence due to this lack of synchronization of the data recorded during the game was too significant, so, it did not allow us to measure the real difference of the positions at a certain time frame during the game.

Nevertheless, with that data it was possible to reconstruct the paths that the players' ships had taken throughout the game. Figure 39 is a reconstruction of the paths of the ship controlled by the host player (Player1) as seen on the server and the client, without the use of VFC. This was recorded in a game involving only two players. It is noted that in both points of observation the paths are the same although recorded at slightly different times, resulting in different positions since the player automation always makes the ship move.

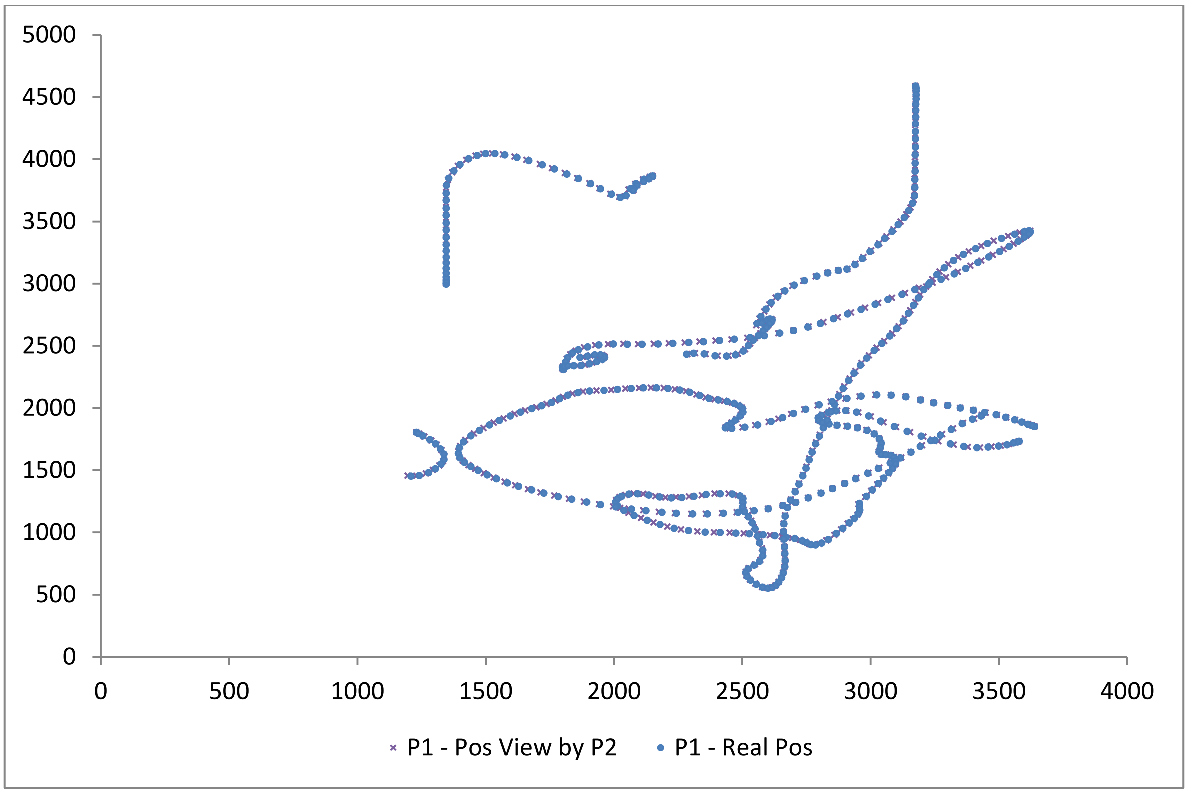


Figure 39 – Positions of Player1 observed in Player2 and the real position without VFC

Figure 40 is a similar example but now using VFC. In general, the consistency enforced by VFC is very similar to the results in Figure 39, without VFC. There are some periods in the game where there is a significant increase in the divergence of the position of Player1 observed by Player2. An example of this is illustrated in Figure 41, in which, after a period when the position was very precise, the difference between the positions observed by the client and the real position increases suddenly. This is explained by the fact that during that time frame the player2's ship was killed, changing suddenly from a very strict to the most relaxed consistency zone. Note that when a ship is killed it is treated as if it was in the consistency zone with minimum requirements, and that there is a period of around five seconds for a ship to be re-spawned somewhere in the map.

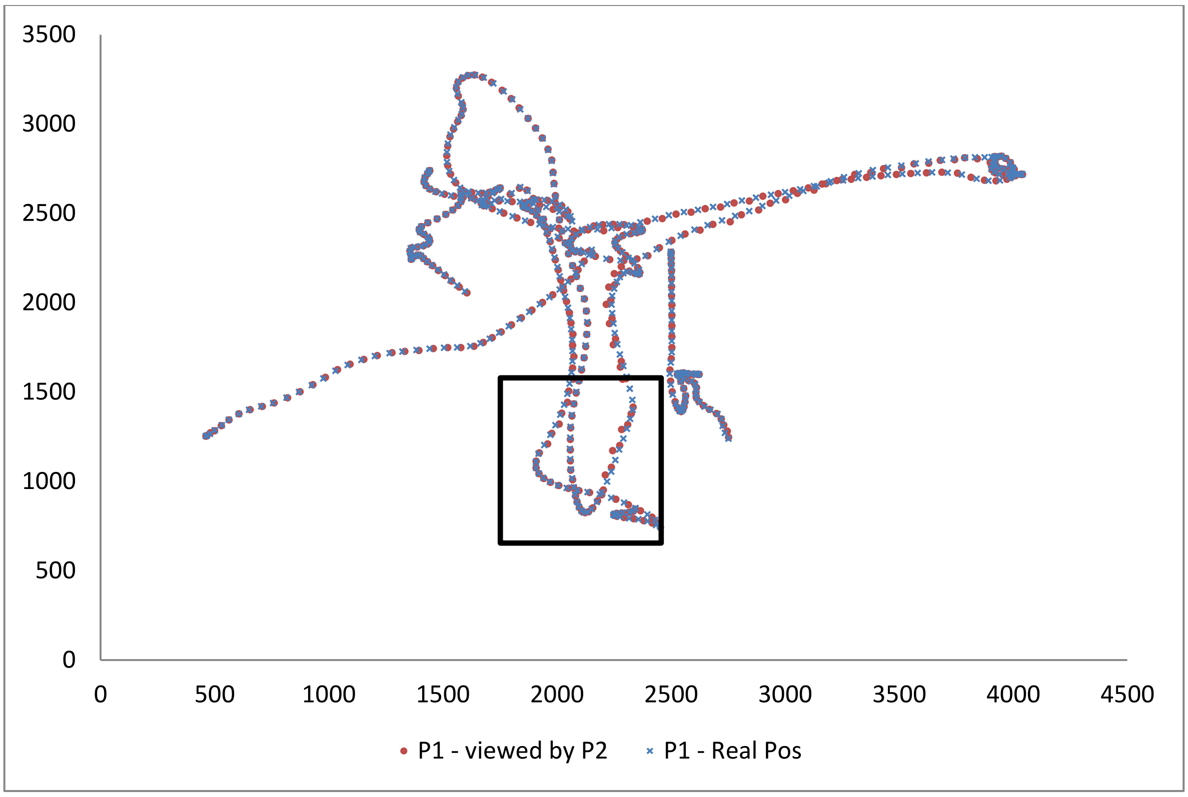


Figure 40 – Positions of Player1 observed by Player2 and the real position with VFC

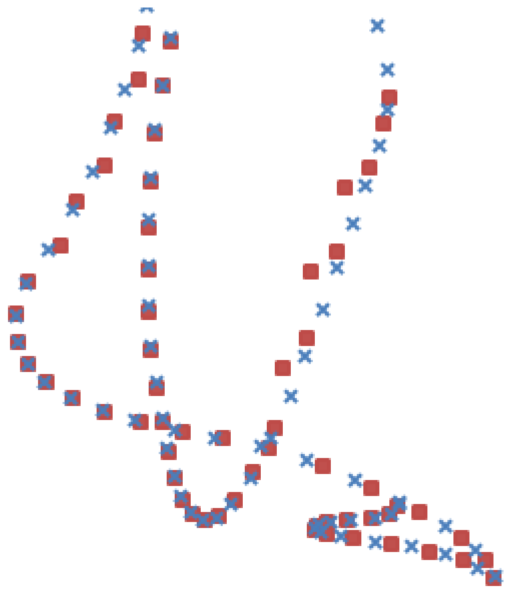


Figure 41 – Detail marked in Figure 40

5.2.3 Latency

Due to the difficulties previously mentioned about the measurement of the divergence of the views of each client there was no precise method to measure the differences when using VFC and not with simulated conditions.

In relation to packet loss simulation it revealed a fine equilibrium. The fact that there are fewer packets being transmitted means that there are also less packets dropped. On the other hand, there is a higher probability of a packet of greater importance (in a tight consistency zone) being dropped since these represent most of the traffic.

5.2.4 User Experience

The playability of the game was maintained with the decrease in network usage provided by the use of VFC. There were occasional controlled inconsistencies observed when viewing a large portion of the game arena (using the zoom feature) but these happened outside the radius of action of the player so had no effect on the players' decisions. The information each player had was accurate enough to make a good decision because the deviation to the real position of other players does not have an impact on, for example, the direction the player had to choose to move towards an opponent or shoot.

If we consider only the original view of the gameplay there was no visible change that allowed a player to even identify whether VFC was being used or not.

An improvement in the playability is expected to happen when the game reaches a saturation point of the network connection. Since VFC reduces substantially the traffic both on the server and the client it can allow more participants in the same conditions without its use.

5.2.5 Frame-rate

Using the component mentioned in 4.3.3, it was possible to observe the frame-rate during the game. It suffered no effect from the use of VFC. This is a rather desirable situation since, if a decrease on the frame-rate was observed, it would possibly mean that the VFC model was unsuitable to be used in multiplayer games.

It could have been possible to improve the frame-rate by increasing the number of participants. With more players and a larger scenario, the reduction of the amount of packets would lead to a decrease of the processing load on clients which could potentially allow a higher frame-rate.

5.2.6 Summary of results

The results obtained with the testing of our solution are very encouraging. We were able to extend the XNA framework in order to implement new communication patterns and a new consistency model. This allows users to play games with larger scenarios and/or with an enlarged view of the playing field, which improves playability. We are able to provide this by reducing the network usage (both in number of messages as well as in total bandwidth). With the obtained results, many games can be played with good frame rates inside LANs and even on wide area networks.

The tests have also shown us good signs of scalability of VFC since the reduction of the amount of data transferred was increasingly higher. This represents a decrease on the processing load both to transfer and to process the packets. Employing scenarios big enough for the playing field not to be saturated, VFC is expected to scale smoothly.

6 Conclusions

The network communication in multiplayer games is an object of extensive study and wide interest. Despite all the work done in the area there is yet to be a solution that can be generalized and applied intuitively. Some previous work incorporates the notions of locality of interest or consistency radius but, an all-or-nothing approximation is usually adopted.

VFC is an intuitive and flexible model that is easily translated to most games' semantics.

Reiterating over the objectives of this thesis:

1. An assessment of the related work was made that lead us to some specific decisions about the architecture of the system that was implemented;
2. The VFC model was successfully applied to a shooter game developed in XNA for personal computers with minimal impact in the gameplay and very encouraging performance results;
3. The XNA game implemented, using the developed VFC module had very good performance and resulted in high bandwidth savings and, consequently, avoided the need to process much of the data previously transferred between server and client;
4. There is a considerable ease in the use of VFC for the game communication due to the modularity of the solution implemented which allowed us to easily switch between methods for prioritizing data used in the communication. It was possible to use VFC with a very small number of changes to the client-server approach of the game.

We were able to implement a prototype of a game provided a case study for the test of the VFC model. VFC proved to be an efficient model that escalates well.

The problems discussed in this work apply to a wide range of practical applications, other than distributed games.

Consistency mechanisms are used in distributed systems in order to manage the divergence of replicas. By allowing controlled divergence among replicas it is possible to considerably increase the availability of systems. When applying these techniques to multiplayer games it is valuable to introduce locality awareness to sophisticated consistency models such as TACT. VFC provides a unified model with simple and flexible abstractions that allow it to be intuitively expressed according to the application semantics.

As multiplayer games vary in genre, its requirements for consistency also change. VFC appears to be fit for any game genre since in all games there is some locality in the players' interest.

There are numerous combinations that can be used in the design of a system for multiplayer games. In terms of network organization of the nodes there are two main architectures: client-server and peer-to-peer. Frequently, in peer-to-peer systems there are portions of the applications' logic that are centralized or grouped and distributed among the nodes (e.g. portions of a game playing field).

The design of a consistency enforcement system for a multiplayer game must take into account numerous factors that may influence the user experience. The number of controlled entities, the importance of the information that may be disregarded, are some of the factors that must be considered.

Implementing this consistency system in a modular manner allows seamless integration with further games and comparison with other alternatives.

It is crucial to have the means to test and study the behaviour of a system when new models are applied and it is useful to automate the testing process.

By evaluating the VFC behaviour we were able to verify its applicability to game semantics and empirically observe the effects it has on the game.

6.1 Future Work

In the future, we intend to pursue the following lines of investigation.

It would be interesting to study the performance of VFC compared to other consistency enforcement models, even if these have an all-or-nothing approach.

The operation in a peer-to-peer architecture requires further investigation in order to comprehend the implications and requirements to enforce VFC.

An interesting exercise would be to extend the VFC model implemented with dynamic consistency requirements. This needs to have a mechanism to either limit the network or enough resources to reach a saturation point of the network used.

The implementation analysed in this work only accounts for one pivot. A study can be made of the behaviour of VFC using a multi-pivot system, e.g. applied to a real-time strategy game. One factor that should be taken into account is that there might be a significant increase in the necessary effort to determine the consistency that needs to be enforced.

7 References

[1] N. Santos, L. Veiga, and P. Ferreira, "Vector-Field Consistency for Ad-Hoc Gaming," *Middleware 2007*, 2007, pp. 80-100.

[2] P. Cederqvist and R. Pesch, *Version Management with CVS*, Network Theory Ltd., 2002.

[3] B. Collins-Sussman, B.W. Fitzpatrick, and C.M. Pilato, *Version Control With Subversion*, O'Reilly Media, Inc., 2004.

[4] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys (CSUR)*, vol. 37, 2005, pp. 42-81.

[5] D.B. Terry, M.M. Theimer, K. Petersen, A.J. Demers, M.J. Spreitzer, and C.H. Hauser, "Managing update conflicts in Bayou, a weakly connected replicated storage system," *Proceedings of the fifteenth ACM symposium on Operating systems principles*, 1995, pp. 172-182.

[6] R.G. Guy, J.S. Heidemann, W. Mak, T.W. Page Jr, G.J. Popek, and D. Rothmeier, "Implementation of the Ficus replicated file system," *USENIX Conference Proceedings*, vol. 74, 1990, pp. 63-71.

[7] J.J. Kistler and M. Satyanarayanan, "Disconnected operation in the Coda file system," *ACM Transactions on Computer Systems*, vol. 10, 1992, pp. 3-25.

[8] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek, "Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication," *Advances in Database Technologies: ER'98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work Support and Spatio-Temporal Data Management, Singapore, November 19-20, 1998: Proceedings*, 1999.

[9] N. Krishnakumar and A.J. Bernstein, "Bounded ignorance: a technique for increasing concurrency in a replicated system," *ACM Transactions on Database Systems (TODS)*, vol. 19, 1994, pp. 586-625.

[10] H. Yu and A. Vahdat, "Design and evaluation of a conit-based continuous consistency model for replicated services," *ACM Transactions on Computer Systems (TOCS)*, 2002, pp. 239-282.

[11] H. Yu, H. Yu, and A. Vahdat, "Building replicated Internet services using TACT: a toolkit for tunable availability and consistency tradeoffs," *Advanced Issues of E-Commerce and Web-Based Information Systems, 2000. WECWIS 2000. Second International Workshop on*, 2000, pp. 75-84.

[12] K.L. Morse, *Interest management in large-scale distributed simulations*, Citeseer, 1996.

[13] J. Chen, B. Wu, M. Delap, B. Knutsson, H. Lu, and C. Amza, "Locality aware dynamic load management for massively multiplayer games," *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, ACM, 2005, pp. 289-300.

[14] I. Kazem, D.T. Ahmed, and S. Shirmohammadi, "A Visibility-Driven Approach to Managing Interest in Distributed Simulations with Dynamic Load Balancing," *Distributed Simulation and Real-Time Applications, 2007. DS-RT 2007. 11th IEEE*

International Symposium, IEEE, 2007, pp. 31-38.

[15] D. Bauer, S. Rooney, and P. Scotton, “Network infrastructure for massively distributed games,” *Proceedings of the 1st workshop on Network and system support for games*, 2002, pp. 36-43.

[16] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, “Peer-to-peer support for massively multiplayer games,” *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, IEEE, 2004.

[17] A. Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Citeseer, 2001, pp. 329-350.

[18] M. Castro, M.B. Jones, A.M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, “An evaluation of scalable application-level multicast built using peer-to-peer overlays,” *IEEE INFOCOM*, INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 2003, pp. 1510-1520.

[19] C. Majewski, C. Griwodz, and P. Halvorsen, “Translating latency requirements into resource requirements for game traffic,” *to appear Proceedings of the International Network Conference (INC’06), Samos*, Citeseer, 2006.

[20] L. Pantel and L.C. Wolf, “On the impact of delay on real-time multiplayer games,” *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, Miami, Florida, USA: ACM, 2002, pp. 23-29.

[21] P. Bettner and M. Terrano, “1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond,” *Presented at GDC2001*, vol. 2, 2001, p. 30p.

[22] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, “The effect of latency on user performance in Warcraft III,” *Proceedings of the 2nd workshop on Network and system support for games*, Redwood City, California: ACM, 2003, pp. 3-14.

[23] Y.W. Bernier, “Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization,” *Game Developers Conference*, 2001.

[24] J. Müller, S. GORLATCH, T. Schröter, and S. Fischer, “Scaling multiplayer online games using proxy-server replication: a case study of Quake 2,” *Proceedings of the 16th international symposium on High performance distributed computing*, Monterey, California, USA: ACM, 2007, pp. 219-220.

[25] C. Diot and L. Gautier, “A distributed architecture for multiplayer interactive applications on the Internet,” *Network, IEEE*, vol. 13, 1999, pp. 6-15.

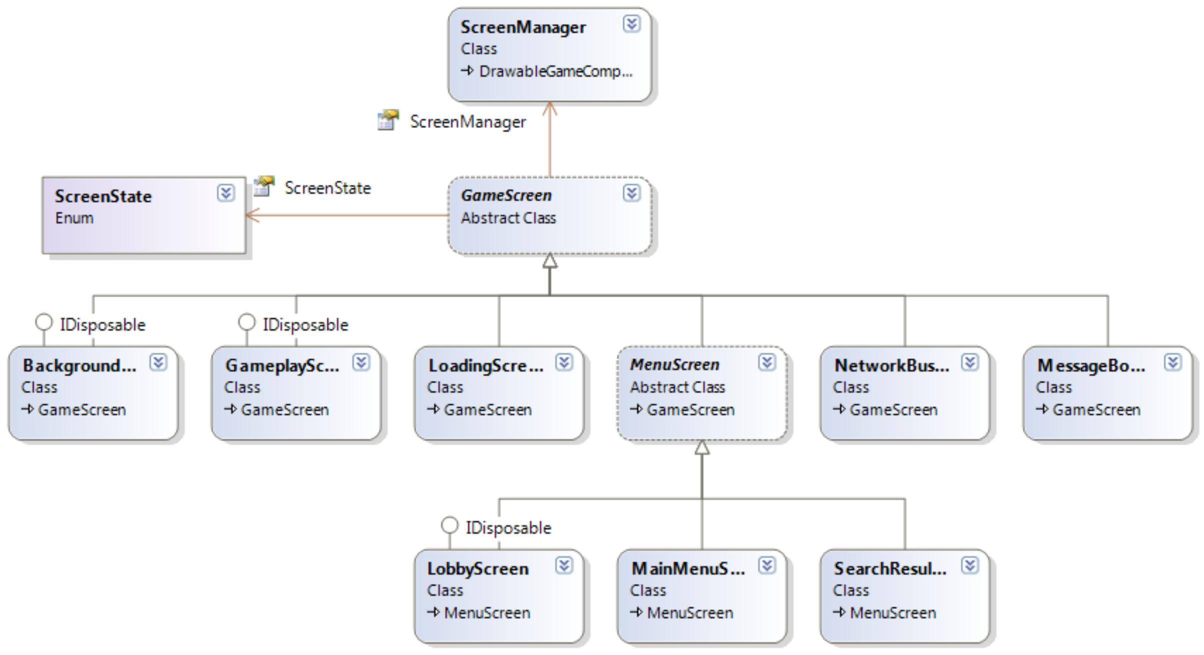
[26] A.R. Bharambe, S. Rao, and S. Seshan, “Mercury: a scalable publish-subscribe system for internet games,” *Proceedings of the 1st workshop on Network and system support for games*, Bruanschweig, Germany: ACM, 2002, pp. 3-9.

[27] M.R. Macedonia, M.J. Zyda, D.R. Pratt, D.P. Brutzman, and P.T. Barham, “Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments,” *IEEE Computer Graphics and Applications*, vol. 15, 1995, p. 3845.

[28] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” *SIGCOMM Comput. Commun. Rev.*, vol. 31, 2001, pp. 149-160.

8 Appendixes

8.1 Net Rumble Screens Diagram


ScreenManager
Class
DrawableGameComp...
ScreenManager
ScreenState
Enum
ScreenState
GameScreen
Abstract Class
IDisposable
Background...
Class
GameScreen
IDisposable
GameplaySc...
Class
GameScreen
LoadingScre...
Class
GameScreen
MenuScreen
Abstract Class
GameScreen
NetworkBus...
Class
GameScreen
MessageBo...
Class
GameScreen
IDisposable
LobbyScreen
Class
MenuScreen
MainMenuS...
Class
MenuScreen
SearchResul...
Class
MenuScreen