
Egosoft

**Network Integration
Networking Background Information
For X4**

Version 1.4D

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

Revision History

Date	Version	Description	Author
15/11/07	0.8	Initial Draft.	Stefan Hett
16/11/07	0.9	Moved the integration part to a separate document.	Stefan Hett
19/11/07	1.0	Proper document layout.	Stefan Hett
19/11/07	1.1	Added numbers to the chapters, marked the document to be confidential and added the missing TOC.	Stefan Hett
27/11/07	1.2	Added two new network architectures.	Stefan Hett
03/12/07	1.3	Client Prediction -> Client Side Prediction and layout fix	Stefan Hett
03/02/08	1.4	Some minor corrections.	Stefan Hett
03/02/08	1.4D	Adjusted version to be included in the diploma thesis document	Stefan Hett

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

Table of Contents

1 Preface.....	3
2 Facts - where we are at the moment.....	4
2.1 Internet Environment of the Present.....	4
2.1.1 Available Bandwidth.....	4
2.1.2 Lag and Packet Loss.....	4
2.2 Network Architectures.....	5
2.2.1 Peer-to-Peer.....	5
2.2.2 Client-Server.....	5
2.2.3 Client-Server Hybrid.....	6
2.2.4 Peer-To-Peer-Client-Server Hybrid.....	6
2.2.5 Network Of Servers.....	6
2.2.6 Distributed Servers.....	7
2.3 Network Relevant Systems.....	8
2.3.1 Object Replication.....	8
2.3.2 Object Synchronization.....	8
2.3.3 Authority Server and Object Roles.....	8
2.3.4 Object Prioritization.....	9
2.3.5 Dead Reckoning.....	9
2.3.6 Client Side Prediction.....	10
2.3.7 Remote Procedure Calls (RPC)s.....	11
2.3.8 Multicasting.....	12
2.4 Current State of the Game Engine.....	13
2.4.1 Game Graph.....	13
2.4.2 Attention Level System.....	13
2.4.3 Event system.....	14
2.4.4 Movement Controller.....	14
2.4.5 AI.....	14
2.4.6 Mission Designer.....	14
2.4.7 UI.....	14
2.4.8 Save game system.....	15
2.4.9 Network Framework.....	15
2.5 Current Network Implementation.....	16
2.5.1 Overview.....	16
2.5.2 XU Layer.....	17
2.5.3 Network Base Layer.....	18
2.5.4 Framework Layer.....	19
3 References.....	20
4 Glossary.....	21

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

1 Preface

The Internet has undergone several changes since the early approaches of the first multiplayer FPS-like games about 10 years ago (Doom (1993), Duke Nukem 3D (1996), X-Wing vs. Tie-Fighter (1996), Quake (1996), etc.[3]). A lot of problems those games had to face are nowadays no longer of any concern. 300-500 ms lags and a lot of package loss became the exception rather than being the norm meanwhile. Analogue Modems or even ISDN has been replaced by broadband which offers us data transfer rates of several kb/s.

Lag and limited transfer rates were the main concerns of the early network solutions of games like Doom and Duke Nukem 3D. [4] Their simple network topology (which in the beginning was based on a peer-to-peer network) together with the requirement of all peers running absolutely synchronously limited the number of players being able to play outside a LAN. [5]

The question one might ask himself now is: “With all these advantages in data transmission would it not be sufficient to use the older more easy approaches, rather than building on network code based on later developments which main purposes are to fight the lag and low data rates of the past?”.

Providing a comprehensive overview of all the different systems which can be used for network implementations, their advantages and disadvantages as well as their requirements and limitations is the goal of this document.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2 Facts - where we are at the moment

This section provides an overview and background of everything which is necessary to understand the statements and ideas presented later to discuss the network integration.

2.1 Internet Environment of the Present

This section describes the current environment we'd have to face when providing internet capable multiplayer support for X4.

2.1.1 Available Bandwidth

Nowadays the available bandwidth is most likely be limited by the upstream rather than a peer's downstream. In case of DSL 1000 [1] provided by the “Deutsche Telekom”, the upstream is only 128 kbit/s (about 16 kb/s of raw data). Depending on the provider and type of DSL-package, the available data throughput might go up to about 128 kb/s (in case of DSL 16000 [2]) though we should not assume that the majority of the players will have that much upstream available.

Furthermore we also have to take into account that 16 kb/s is only the raw throughput. About 11% [6] are acquired by the header information of the used transfer protocols. This leaves us with approximately 14 kb/s.

2.1.2 Lag and Packet Loss

Based on reports in the Internet and my own experience with Counter Strike and Team Fortress 2 servers, the current lag we should expect is about 100 ms - 200 ms. There are already servers which kick players with pings above 100 ms.

Packet Loss is no longer a big concern. It ranges between 0% and 2% (based on location, ISP and the current time). [7, 8]

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.2 Network Architectures

The following subchapters describe the different network architectures.

2.2.1 Peer-to-Peer



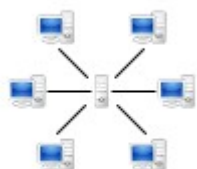
A peer-to-peer network is a network solution used by early multiplayer games like Duke3D or Doom. In this kind of network each peer normally runs exactly the same simulation sending informations about its own player inputs to all other peers.

In case of Duke3D and Doom all clients were completely synchronized meaning that each computer waited for the information package of all connected clients, processed it and then rendered the next frame.

The big advantage of this system is its relative simplicity, the absence of any synchronization issues and the straight forward network implementation.

The disadvantages however are that the frame rate depends on the slowest machine, the high latency because all clients need to send their package reliably to all other clients and the inability of fine tuning the traffic (meaning that all participants receive all the client updates, even if the information is not important to all of them).

2.2.2 Client-Server



The most widespread network architecture nowadays when it comes to non-MMOGs is the client-server architecture. It consists of a dedicated server, which means that the server is running the authoritative version of the game which multiple clients are connected to.

Normally, those clients only keep a subset of the complete game state (that part, which they are interested in – i.e. are currently displaying).

The server controls the complete data transmission. Clients do not transmit data between one another but instead solely communicate with the server.

This way it can be assured that the transmission can be easily adjusted. The server can decide what the most important information for each client is and send it right away without having to bother about missing packets from a client (up to a given point, of course). It also provides the possibility for cheat prevention (since the server could have the authority over the complete game). It also reduces the bandwidth required by connected clients, because they now only need to communicate with one peer (the server) rather than with all connected clients. From a design perspective this architecture could be used as a basis for a clean game engine, since it requires the server side code from being separated from the client's one.

This of course is also a downside. It really requires the game to be specifically designed for such a system. A strict client-server architecture also prohibits the user at the server from taking part at the

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

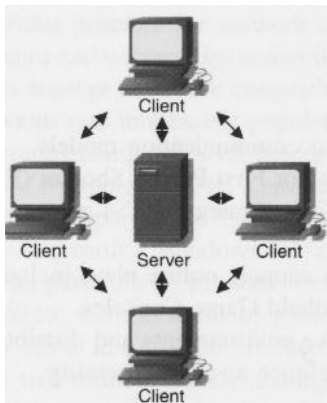
game unless he runs a client process, too (which causes an overhead and therefore requires an even more powerful server machine). Additionally the server can easily be the bottle-neck in such a system. If its upstream does not cope with the required bandwidth, the number of connected clients will easily be limited.

Speaking about cheat prevention: There is normally no way in a strict client-server architecture to prevent the server itself from cheating.

2.2.3 Client-Server Hybrid

The Client-Server Hybrid architecture looks quite the same compared to the Client-Server approach. The main difference is that in this case the server machine combines the server and client in a single process. That way the overhead for the server compared to the client-server architecture is reduced, with the cost of increasing the complexity of the game engine at the same time.

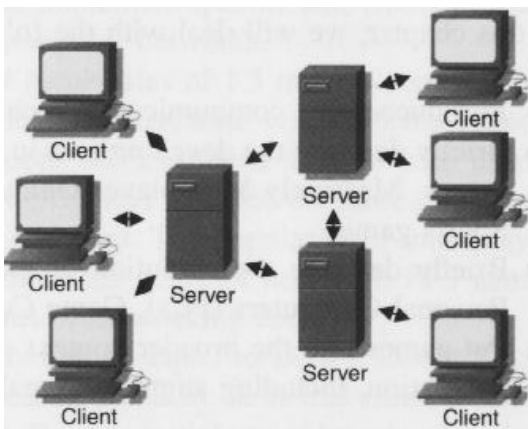
2.2.4 Peer-To-Peer-Client-Server Hybrid



To further reduce the problem of the server being the bottleneck in the client-server architecture, in this architecture clients themselves can communicate to one another without having to use the server for basic data relying procedures. For instance VoIP in a game can be implemented by allowing clients to directly transmit the voice data packages to the receiving client while the server is still in control of game relevant communications.

A slight variation of this architecture could be one which combines the server machine with a client at the same time. The advantages/disadvantages would be the same as the ones described in the Client-Server-Hybrid chapter.

2.2.5 Network Of Servers



Most commonly known from MMOGs, this architecture splits the single server machine up into several ones to reduce the workload for a single machine. This kind of architecture is also known to be used for Grid Computing (Boinc, [Seti@Home](#), ...) or database systems.

While splitting up the necessary processing power and bandwidth between several servers, such an architecture increases the complexity of any server application, especially when allowing clients to “move” between servers and or communicate with clients currently connected to another server.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.2.6 Distributed Servers

Based on the “Network of Servers” architecture, distributed servers architectures allow connected peers to act as servers and clients at the same time. Each peer simulates part of the game world and communicates with all peers (though, not necessarily with all at the same time).

Though this approach has not yet been completely implemented in any computer game (to the knowledge of the author), part of the idea has already been taken up by games which allow the server to switch (for instance, if the server loses the connection, one of the still remaining clients will take over).

Of course implementing a network system based on this architecture requires a lot of additional work and can be seen as the most difficult one compared to the all others.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.3 Network Relevant Systems

This section provides an overview over the most relevant systems and techniques used for the network engine design.

2.3.1 Object Replication

Object replication describes a system which autonomously informs peers of the creation and destruction of objects. For instance when the server constructs a new ship, it automatically sends this information to all clients.

The idea behind this system is that clients would then create a corresponding object to synchronize their local representations of the game with the server.

2.3.2 Object Synchronization

In combination with object replication, these two systems provide an easy way to keep objects synchronized in a network game. While object replication takes care about the creation and destruction of objects, object synchronization informs objects about any change which occurred to their corresponding “master” object.

One of the advantages of such a system is that it supports the object oriented approach, which is that functionality related to an object should be handled by the object itself.

2.3.3 Authority Server and Object Roles

Imagine two clients are playing against one another in an FPS. The first one ducks behind a crate. Caused by latency the second player can still see the first one before he ducks and lands a headshot. Which one of the two clients is right? Answering such questions is the task of an authority server.

In accordance with the authority server comes the object's role. Network objects are distinguished by the role they have on a peer. The basic different roles are:

Authority – When the peer has the authority over an object, he is the one who does the final decisions on what happens with the object (on an authority server, all objects have this role).

Owner – This role indicates that the object is being owned by this peer (for instance the player object on a client might keep this role). The client will usually send updates to the server for that object.

Proxy – All objects on clients which are not owners, are proxies. Proxy objects normally used to indicate that the client uses simple prediction algorithms until being informed of updated states for that object by the server.

Even though the object roles' names already imply their meaning, roles are basically used to differentiate between different kind of objects on a peer. They can be used in methods to execute different code based on the current role, for instance. Roles are also not limited to these three,

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

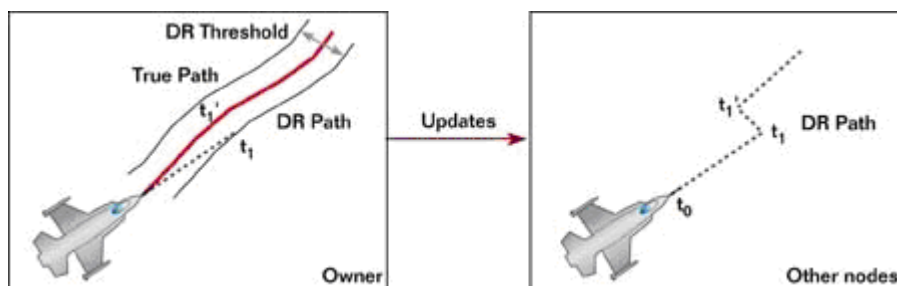
though these are the most common ones. For example the Unreal 1 engine introduces three proxy roles (DumpProxy, SimulatedProxy, AutonomousProxy) [5].

2.3.4 Object Prioritization

We normally won't have all the bandwidth available we'd like to have to transmit all the detailed information to all our clients. Even if we limit the number of objects being transmitted to the clients we might sooner or later run into a bottleneck. In such a situation we can prioritize the data which needs to be sent. That way we ensure that the data which is more important (let's say the current player's position) than other (for instance the heading of the nearby fish swarm) is still being sent.

2.3.5 Dead Reckoning

Dead Reckoning is used to reduce the amount of data which needs to be transmitted. Assume a player is flying in a straight line towards a space station. Why should we need to send the player's updated position all the time, if we very well could extrapolate the correct flight path from it's current position, velocity and acceleration?



That's what Dead Reckoning is used for. The basic idea behind it is that the server and all clients run the same simulation of an object. If the object changes it's path the server checks, if the aberration between the simulated and the object's real position is important enough for the connected clients to be informed of the changed path. If so, the server sends so called PDUs (Protocol Data Units) which contain the object's updated information.

The PDUs are then utilized by the clients and the object's path is adjusted. [10]

The question here is what to do when the client receives a PDU. Of course he could simply put the object at the corrected position and then let it fly in a straight line again. It's quite obvious that for nearby objects which move quite fast this results in a visible jitter. Much better would be the possibility to smoothly extrapolate the object's movement and include the data received in the PDU.

Cubic splines are one possible extrapolation method. They not only rely on the objects current and corrected position but also take into account its velocity. To even further improve the extrapolated spline, we include quadratic movement when calculating p3 and p4. That way we also take into account the object's acceleration.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

The 4 points which define the cubic spline are calculated as follows [11]:

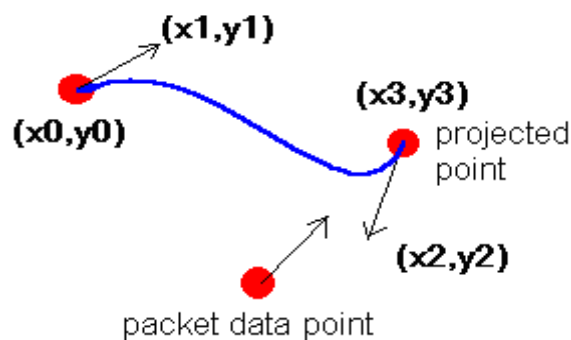
$p1$ = Current position of the object.

$p2 = p1 + \text{current velocity of the player}$

$p3 = p4 - \text{PDU's velocity} - \text{PDU's acceleration} * t$

$p4 = \text{Position of the object given by the PDU} + \text{PDU's velocity} * t + .5 * \text{PDU's acceleration} * t * t$

The resulting spline could look like this:



2.3.6 Client Side Prediction

Client prediction is one of the most efficient ways to reduce (and in most cases completely remove) any recognizable lag for players. Old systems relied on sending the player input to the server prior to applying the changes to themselves. The advantage of this solution was that all the clients in a game were informed of the player input as soon as possible and thereby the resulting latency for waiting for those information was as small as possible.

On the downside the clients themselves needed to wait for the server acknowledgment telling them that the package was received. Especially in the Internet where packages might travel several ms from one computer to another this resulted in a noticeable lag. Quake 1 was one of the games which suffered from such a system.

QuakeWorld tried to improve these drawbacks by introducing client prediction. Rather than waiting for the acknowledgment package from the server, clients at once process the player input. Additionally to informing the server about the player input, they store all the player inputs in a buffer.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

The server still sends its normal update packages to the player. When the player receives a package it:

- discards all stored input data prior to the package's timestamp
- sets the state of the object (normally the position, acceleration, orientation, etc.) according to the received PDU
- applies the stored player input on the current object

This results in a smooth player control while at the same time ensures that the player's state on the client does not vary too much from the one displayed on other clients. [12]

2.3.7 Remote Procedure Calls (RPC)s

RPCs provide the means of running methods/functions on another machines while, from a programmers point of view, it looks like he simply calls a local method.

This can be used to easily access functionality on a server from the client. For instance it might show useful to perform non-time-critical but transaction-critical operations directly on the server instead of creating network messages which then would have to be interpreted by the receiver.

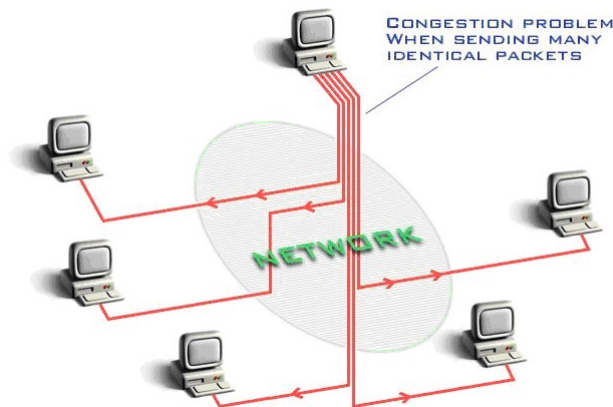
Imagine the player interacts with a merchant to buy a new engine for his ship. We could of course create a network package ourselves, send it to the server, wait for the server's reply and then make the appropriate changes, while the server does the same on his side.

It might be easier though to simply call a method: BuyShield(playerid, shieldid). This method would return at once but instead of performing any local changes calling this method would result in the appropriate server side call of the BuyShield() method. It then checks, if the shield is still available, whether or not the player has enough cash and ensures that the player's ship is allowed to have the selected shield. If so, the server could adjust the gamegraph and make the appropriate changes. Using object replication and synchronization the client would then receive the shield.

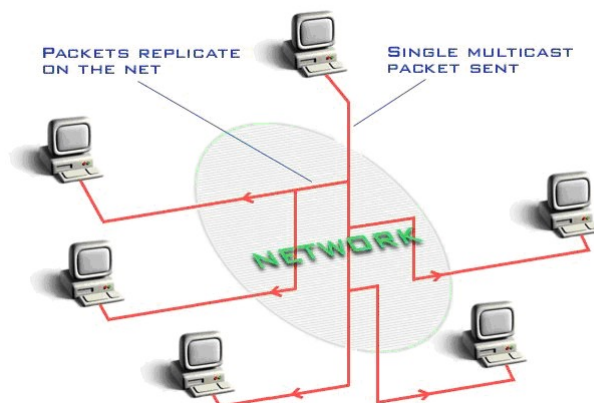
Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.3.8 Multicasting

One of the main limitations in a client/server topology is the required upstream by the server. Even if he sends the same data to multiple clients a separate package for each client is being created which needs to be transmitted through the server's bottleneck.



Multicasting is one way to improve this situation. It allows the server to send packages to multiple recipients rather than only a single one. Replicating the package is done inside the network. This can reduce the necessary upstream by the client.



If multicasting is such a great tool, why is it not widely used? The main reason beside some additional work for the programmers is that it needs to be supported by the underlying infrastructure. There are still many ISPs which do not provide multicasting support in their network. Referring to Martin Brenner's own research, this is still the case in the present, though with the improvements which come together with Web 2.0 it should be supported on a wider range in the near future (video streaming heavily relies on the capability of multicasting).

Another limitation of the expected gain for an application is that multicasting only helps when exactly the same data is being replicated to multiple clients. [13]

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.4 Current State of the Game Engine

The state of the game engine is comparable to what Peter Lincroft was facing when starting development on X-Wing vs. Tie-Fighter. [4]

We already have a powerful game engine which supports several aspects of a single player space simulation (graphic engine, UI, AI, event system, mission designer, etc.). However, beside some nice features like the design of our movement controller and some of the basic concepts behind the engine (for instance our attention levels), the game is not strictly designed with the goal of adding a multiplayer mode. The following is a list of systems which we think are most relevant to a possible network implementation.

2.4.1 Game Graph

Being the core of the game, the game graph represents the current state of the complete universe. In its hierarchical structure, it contains all entities in X4 including their relations (connections). Almost all the network relevant data can be found in the game graph, although there are some game elements which are not directly attached to the game graph. There can be up to hundred of thousands of objects within the hierarchy (depending on the game environment).

At the moment the game graph already provides the capability to save/load its current state to/from an XML stream/file but lacks the possibility to resynchronize itself and is not thread safe.

2.4.2 Attention Level System

The strongest concern about having to manage a huge amount of objects is performance. It would be almost impossible to update the states of hundreds of thousands of objects several times per second and still have enough CPU time left to render the graphics, play the sound, update the AI, etc.

That's where attention levels come into play. By separating all the objects and assigning them to different attention levels, we are able to finetune which part of the game graph is updated how often and even handle special sections totally differently. For instance the system allows us to only calculate physics for objects which are in the high attention level (ships which are close to the player, for example) while objects being far away in a totally distinct part of the universe can use faster methods to compute their actions.

The system is flexible enough to be able to run more than just one single part of the game universe in a specific attention level. This might be helpful since we could run each sector which a player is currently positioned in the same high attention level. It's also imaginable that we could fall back to a lower attention level, if the server is lacking the necessary power to compute all the different players' sectors.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.4.3 Event system

This system is used as a means of internal communication between objects. The observer pattern is used to inform objects whenever anything interesting happens. Events can initiate the construction and/or destruction of objects, be the reason for AI objects to change their current state and/or cause the UI to display new information, for instance.

The event system is rather powerful and at the same time quite big (by means of the communication it produces). We expect it to generate thousands of events each second.

2.4.4 Movement Controller

To be able to fly around the universe objects need so called movement controllers. These controllers are attached to a ship, for instance which then can navigate through the sectors. One feature of the movement controllers is that they don't need to update the ships position iteratively but instead can directly calculate the final ships position/velocity/orientation based on the given time offset. That might come in handy when/if client prediction is to be implemented.

However the movement controllers currently don't directly support dead reckoning or client prediction, for instance.

2.4.5 AI

The current implementation in the AI sector already allows us to create NPC ships. Those ships can follow calculated paths which are based on bezier curves. The actions for NPCs are computed in advance, which is a great advantage when it comes to networking, because those precalculated behaviors could be transmitted to the clients before they are executed.

On the downside the AI does not yet support receiving updates which define the NPC behavior.

2.4.6 Mission Designer

The mission designer allows the creation of all kind of missions. The current design however is limited to single player missions. Functionality which would be required for multiplayer missions has yet to be implemented.

2.4.7 UI

The UI contains all the functionality which takes care of user in- and output. It handles the different HUD elements and allows the user to control his ship and access different menus. The system heavily relies on the event system at the moment. Events are being caught by so called message sources (for instance the range scanner), which then uses the event to create corresponding UI messages.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.4.8 Save game system

Our current XML based save game system is designed to create a minimalistic save game which only contains as little information as necessary to regenerate the stored gamestate. As a first rough estimate we expect the size of a complete savegame to be about 5 MB (this is a minimal estimate and will likely to be much higher in the release version).

2.4.9 Network Framework

We currently have a basic network implementation which allows us to establish a connection between several clients and one server. Simple data transmission has also been tested.

Furthermore the two tested network frameworks (namely RakNet and ZoidCom) provide build-in capability of replication support.

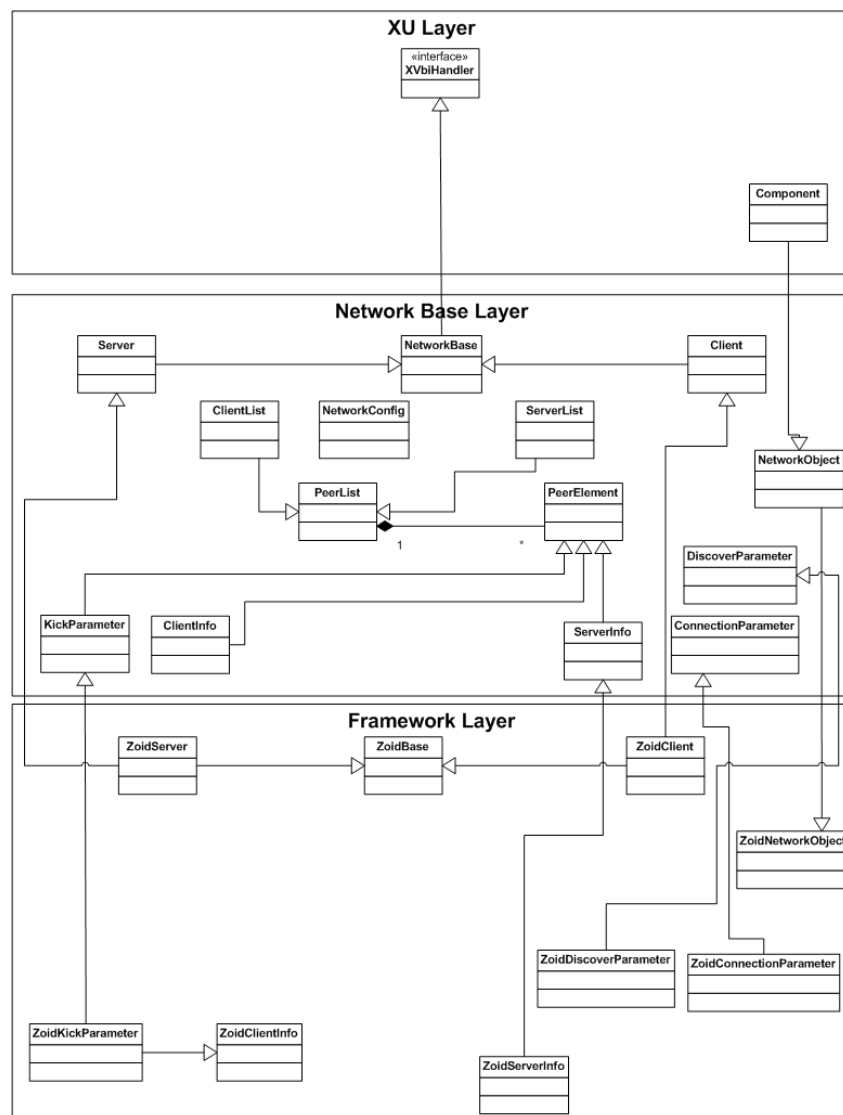
Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.5 Current Network Implementation

This chapter covers the current implementation of the network engine.

2.5.1 Overview

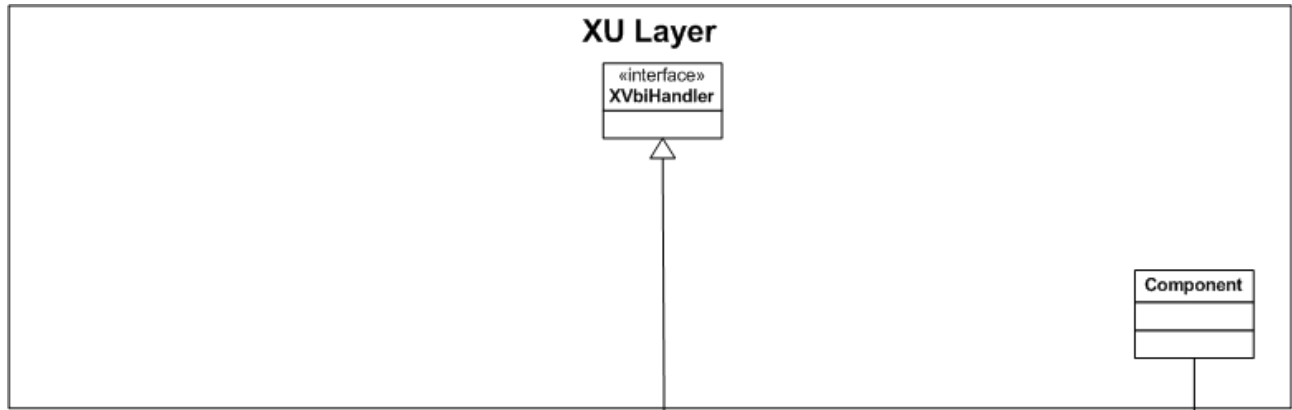
The following figure depicts the current state of the X4 network implementation.



The current network integration is based on 3 separate layers. The main reason for implementing it that way, beside achieving easier maintainability and a clearer structure, is the requirement of being able to replace the underlying network framework. That way we don't essentially rely on the framework which is being used and can replace it by any other without causing any impact on other parts of the game code.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.5.2 XU Layer



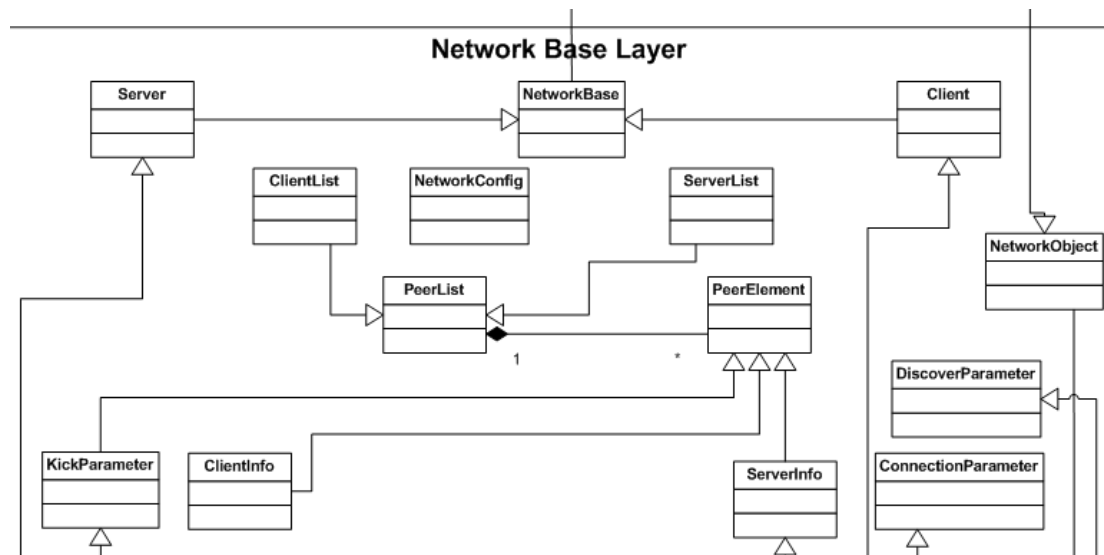
This is the layer which directly integrates the network functionality into X4. At the moment it only contains two interaction points which are necessary to get things started. The final design of this layer depends on the decisions made after this document has been reviewed.

The connection points so far:

In this context the Component class which derives from the NetworkObject class represents all game objects which are being accessed by the network code. The XVBIHandler interface is inherited by the NetworkBase class to allow the Network classes being attached to the game cycle handler.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.5.3 Network Base Layer



The network base layer provides the core network functionality and directly builds on the Framework Layer. Its main purpose is to define the general interface which the XU layer uses to perform the network tasks without having to know anything about the underlying network framework.

Currently it provides the following features:

Server side:

- Listen for incoming discovery requests.
- Reply to incoming connection/discovery requests.
- Keep a list of connected clients.
- Kick one specific or all clients.

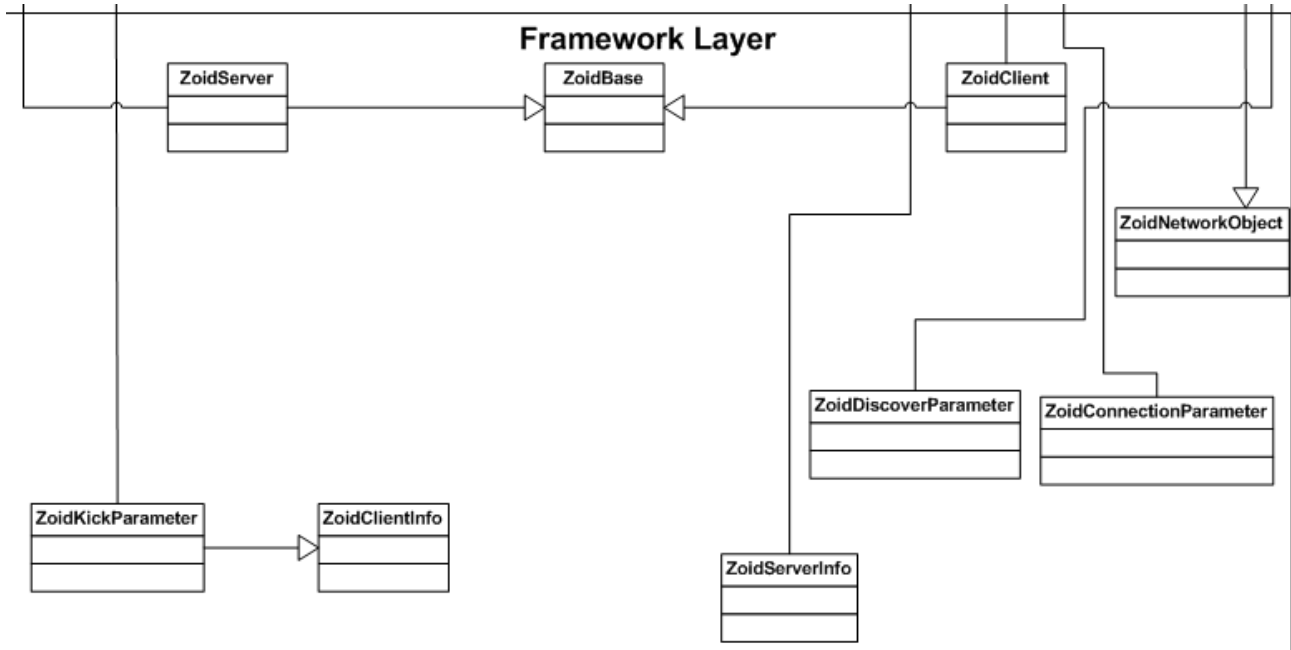
Client side:

- Discover servers.
- Connect to a specific server.
- Keep a list of discovered servers.
- Keep track of the connected server.
- Disconnect from the server.

Depending on the network solution we are about to implement, many more features need to be added to this list. The goal is that the XU layer only accesses the network using methods/functions provided by this layer (i.e. there is no need to bypass the Network Base Layer and directly communicate with the Framework Layer).

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

2.5.4 Framework Layer



The Framework Layer is the bottom most layer used for X4's network implementation. The figure above depicts the current test implementation using ZoidCom. Depending on the network framework in use the class structure in this layer might completely change.

Keeping a separate layer for the concrete framework implementation reduces its impact on any other parts of the code. That way we ensure that replacing the current framework can be done with a modicum of effort.

The task of this part of the network code is to provide the connection between the Network Base Layer and the OS' network interface, providing the necessary functionality to connect clients to servers, send data over the Internet, etc.

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

3 References

- [1] Deutsche Telekom homepage – information about DSL 1000: http://www.t-home.de/Produkt_PK_Keyword;sid=PWuThnc9IvKThTANtUiZHIY1IcBQ7HY4tF2v49e7DYYALVb217E=?ProductRef=0302001000714%40EKI-PK&KeywordPath=Katalog%2FTelefonieren_surfen%2FWeitere_Produkte%2FDSL%2FAnschluss
- [2] Deutsche Telekom homepage – information about DSL 16000: http://www.t-home.de/Produkt_PK_Keyword;sid=PWuThnc9IvKThTANtUiZHIY1IcBQ7HY4tF2v49e7DYYALVb217E=?ProductRef=0302001000711%40EKI-PK&KeywordPath=Katalog%2FTelefonieren_surfen%2FWeitere_Produkte%2FDSL%2FAnschluss
- [3] IMDB – release dates of several multiplayer capable games: <http://www.imdb.com>
- [4] Designing Fast-Action Games For the Internet:
http://www.gamasutra.com/features/19970905/ng_01.htm
- [5] Unreal Networking Architecture: [F:\Stefan Working Folder]
- [6] DSL Network Guide – Chapter 18: Overhead:
<http://maxolasersquad.com/network/chapter18.html>
- [7] Internet Traffic Report: <http://www.internettrafficreport.com/>
- [8] Internet Pulse: <http://www.internetpulse.net/>
- [9] The Internet Sucks: http://www.gamasutra.com/features/19990903/lincroft_01.htm
- [10] Defeating Lag with Cubic Splines: <http://www.gamedev.net/reference/articles/article914.asp>
- [11] Dead Reckoning – Latency Hiding for Network Games:
http://www.gamasutra.com/features/19970919/aronson_01.htm
- [12] Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization
[F:\Stefan Working Folder]
- [13] Advanced WinSock Multiplayer Game Programming: Multicasting:
<http://www.gamedev.net/reference/articles/article1587.asp>

Network Integration	Version: 1.4D
Background Information	Date: 04/02/08
X4-NET-BACKGROUND	

4 Glossary

AI	– Artificial Intelligence
FPS	– First Person Shooter
host	– a computer connected to a network
MMOG	– Massive Multiplayer Online Game
PDU	– Protocol Data Unit
peer	– a computer connected to a peer-2-peer network / also used as a synonym for host
UI	– User Interface