МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних систем та мереж

"ДОПУСТИТИ ДО ЗАХИСТУ"
Завідувач кафедри
_____ Жуков І.А.
"_____" _____ 2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

випускника освітнього ступеня "МАГІСТР"
спеціальності 123 «Комп'ютерна інженерія»
освітньо-професійної програми «Комп'ютерні системи та мережі»

на тему: **«Комп'ютерні мережі як апаратна платформа для проектування онлайн-ігор»**

Виконав: _____ Зуй Д.Л.
Керівник: _____ Проценко М.М.
Нормоконтролер: _____ Надточій В.І.

Засвідчую, що у дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань
_____ Зуй Д.Л.

Київ 2020

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL AVIATION UNIVERSITY

Faculty of Cybersecurity, Computer and Software Engineering

Computer Systems and Networks Department

"PERMISSION TO DEFEND GRANTED"
The Head of the Department
_____ Zhukov I.A.
"_____" _____ 2020

MASTER'S DEGREE THESIS

(EXPLANATORY NOTE)

Specialty: 123 Computer Engineering

Educational-Professional Program: Computer Systems and Networks

Topic: **"Computer networks as a platform for designing of online games"**

Completed by: _____ Zui D.L.

Supervisor: _____ Protsenko M.M..

Standards Inspector: _____ Nadtochii V.I.

Kyiv 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії

Кафедра Комп'ютерних систем та мереж

Освітньо-кваліфікаційний рівень магістр

Спеціальність 123 "Комп'ютерна інженерія"

Освітньо-професійна програма: «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____Жуков І.А.

« ____ » _____ 2020 р.

З А В Д А Н Н Я
на виконання дипломної роботи

Зуя Дмитра Леонідовича
(прізвище, ім'я, по батькові)

1.Тема проекту (роботи): " Комп'ютерні мережі як апаратна платформа для проектування онлайн-ігор "
затверджена наказом ректора від " 25 " вересня 2020 року № 1793/ст.
2.Термін виконання проекту (роботи): з 05.10.2020 до 12.12.2020
3. Вхідні дані до роботи (проекту*)*:.функції та мережеві потреби для розробки онлайн ігр
4. Зміст пояснювальної записки:
Вступ, огляд теми, огляд існуючих мережевих технологій, розробка мережової частини онлайн гри, висновки по роботі
5. Перелік обов'язкового графічного (ілюстративного) матеріалу:
Матеріали представленні у вигляді презентації в *Power Point.*

NATIONAL AVIATION UNIVERSITY

Faculty of Cybersecurity, Computer and Software engineering

Department: Computer Systems and Networks

Educational and Qualifications level: Master Degree

The Specialty 123 "Computer Engineering"

The Specialization 123.01 "Computer Systems and Networks"

APPROVED BY

The Head of the Department

_____ Zhukov I.A.

"_____"_____2020

Graduate Student's Degree Project Assignment

Zui Dmytro Leonidovych

1. The Project topic: "Computer networks as a platform for designing of online games"

Approved by the Rector's order of 25.09. 2020 № 1793/st.

2. The Thesis to be completed between 05.10.2020 and 12.12.2020

3. Initial data for the project (thesis): functions network requirements for the online game development

4. The content of the explanatory note (the list of problems to be considered):

Introduction, review of the topic, review of existing network technologies, development of the online part of the online game, conclusions on the work

5. The list of mandatory graphic materials:

Materials representation in the form of Power Point Presentation

6. Календарний план-графік

№ п/п	Етапи виконання дипломного проекту	Термін виконання етапів	Підпис керівника
1	Узгодити технічне завдання з керівником проекту	05.10.20	
2	Виконати пошук та вивчення науково-технічної літератури за темою роботи	06.10.20 – 07.10.20	
3	Опрацювати теоретичний матеріал щодо стану науково-технічної проблеми	08.10.20 – 15.10.20	
4	Проаналізувати мережеві технологіх в ігрових проетах	16.10.20 – 25.10.20	
5	Спроектувати мережеву ігрову систему та протестувати рішення ігрового сервера	26.10.20 – 25.11.20	
6	Оформити пояснювальну записку	26.11.20 – 05.12.20	
7	Оформити графічні матеріали проекту та представити роботи на кафедру	06.12.20 – 12.12.20	

7. Дата видачі завдання «05» жовтня 2020 р.

Керівник дипломної роботи _____ Проценко М.М.
(підпис)

Завдання прийняв до виконання _____ Зуй Д.Л.
(підпис студента)

6. Timetable

#	Completion Stages of Degree Project	Stage Completion Dates	Remarks
1	Coordinate the technical task with the supervisor	05.10.20	
2	Perform research and study of scientific literature on the topic	06.10.20 – 07.10.20	
3	Elaboration of theoretical material	08.10.20 – 15.10.20	
4	Analyze game networking technologies technologies in the work of the analytical department	16.10.20 – 25.10.20	
5	Design networked game system and game server solution testing	26.10.20 – 25.11.20	
6	Perform the analysis of results, examine and evaluate efficiency parameters of recognition, provide recommendations for accuracy improvement and write the explanatory note.	26.11.20 – 05.12.20	
7	Receiving review from the supervisor. Deliver materials to the department.	06.12.20 – 12.12.20	

7. Assignment issue date: 05.10.20

Diploma Thesis Supervisor _____ Protsenko M.M.
(Signature)

Assignment accepted for completion _____ Zui D.L.
(Signature)

ABSTRACT

The Explanatory Note to the Master's Degree Thesis "Computer networks as a platform for designing of online games": 89 pages, 49 figures, 20 references.

Object of research: computer networks as a platform of the development of multiplayer game, network performance metrics, sources of jitter, loss and latency, methods of solving the problem of client-server communication, implementation of game server as the tool of game data varification.

Purpose: to solve the problem of game networking, specify latancy, jitter and loss using client – server architecrute.

Research methods: processing of scientific and technical literature sources, comparative analysis, configuring and testing game servers by means of Netty and Unreal Engine.

The results of the master's work are recommended to be used during scientific research in the sphere of game networking, as well as in the practical activity of developers dealing with networking tasks.

CLIENT-SERVER ARCHITECTURE, NETWORKING, PREDICTION SYSTEMS, JITTER, LOSS, LAG COMPENSATION.

CONTENT

LIST OF SYMBOLS, ABBREVEATIONS, TERMS

UDP	–	User Datagram Protocol
TCP	–	Transmission Control Protocol
LAN	–	Local Area Network
WAN	–	Wide Area Network
RTT	–	Round Trip Time
ISPs	–	Internet Service Providers
AI	–	Artificial Intelligence
BBSes	–	Bulletin-Board-Systems
P2P	–	Peer-to-peer
VoIP	–	Voice over Internet Protocol
PPS	–	Packets Per Second
MTU	–	Maximum Transition Unit
PPP	–	Point-to-Point Protocol
MAC	–	Media Access Control
CRC	–	Cyclic Redundancy Check
ATM	–	Asynchronous Transfer Mode
DOCSIS	–	Data-over-Cable Interface Specification
CSMA/CA	–	Carrier Sense Multiple Access/Collision Avoidance
FEC	–	Forward error correction
QoS	–	Quality of Service
WFQ	–	Weighted Fair Queuing
TOS	–	Type of Service
SLA	–	Service Layer Agreement
CIR	–	Committed Information Rate
EIR	–	Excess Information Rate

INTRODUCTION

Actuality of theme. Computer networking is a branch of computer science, telecommunication and computer engineering and its was influenced by a wide range of the technologies and historical events.

A computer network provides new ways of communications by means of diverse technologies, such as online chats and messengers, electronic mail, voice and video calls. Network users are able to share resources such as data, files and other types of information. Authorized participants of the network can gain access to the stored information on other computers, resources provided by the devices or perform some tasks with the help of distributed computing across the network.

Despite the fact that networked games are rather fresh development field in comparison to the single-player games, computer networks are rather old and well-studied subject, some first computer network systems date back to the 1950s.

In 1958 William Alfred Higinbotham created one of the first electronic games that used an oscilloscope to simulate a virtual interactive game “Tennis for Two” (fig.1.1). Since that time computing technology has made huge steps forward in power, speed, sophistication and miniaturization. Internet, as a collection of interconnected high speed international data networks, has become a part of our modern everyday life. Human desire for fun and entertainment has pushed the development of both computing and networking technologies. Nowadays, computer games market is significantly big and its annual profit exceeds that of the movie industry.

Even though lots of technique have improved over the years, but the basic idea behind networked game is the same: two are more computer machines are interconnected to the establish communication between them. Communication is a data exchange, i.e. sending messages from one computer to another and getting feedback, or one machine sends some information and the other only receives and processes it.

Multi -player games increasingly use the internet and are driving demand for the better that dial-up access services in the consumer area. Nevertheless, networking engineers are not aware of games that use their networks and game developers are not

quite sure how exactly the internet behaves. Multiplayer games are different kind of software product, that utilizes Internet, influences traffic patterns on the Internet and puts a strain on the Internet Service Providers (ISPs) than in a different way than email, web surfing or content streaming.

The need for realistic interactivity leads to unique demands at the network level for reliable and timely communication over the Internet and it is something that Internet can rarely provide because it is originally used as a “best effort” service. Best-effort delivery means that a network service does not guarantee that data will be delivered, i.e. packet loss can take place, or that delivery meets any quality of service, i.e. network performance characteristics depend on traffic load. Because of the Internet instability game designers have created some techniques to trick user and maintain a game’s integrity and illusion of a shared experience even when network data loss can occur and general misbehavior takes place. As all computer systems and applications, multi-player games improved along with the capabilities of the hardware, it became cheaper and more powerful.

This brief description of the networking and multi-player games is enough to know what is required to network a game — make two or more computers exchange data with each other as close to the real time as possible.

The purpose of the thesis is designing and testing of the computer network as a platform of multiplayer game development with a client-server architecture.

Research methods. The thesis covers analysis of the network as a hardware component of the multiplayer game. The main attenction is paid to the network performance metric such as jitter, loss, propagation delay and its impact on the overall system.

Lost of lag compensation techinques, particularly, prediction system for the multiplaer game and action verification on the server side, is described in the Part 3, it is devoted to the development of client-sever architecture computer system to solve the player and mainfraime server communication and solve prediction problem. Also recommendation and method for further improvement of the created system are proposed.

Scientific novelty of the obtained results. The techniques and methods for the game networking were deleoped and improved that allow fast communication between client and server using User Datagram Protocol and prediction and action verification mechanisms on the server side to speed-work of the networked system and reduce error rate.

The practical significance of the thethis results allow to implement the fast and efficient game network to solve the lag issues by means of prediction system and server-side action verification. Also, it is worth to be menatione that proposed solution can be scaled and improved to receive better result even in high load systems.

PART 1

PRINCIPLES OF THE GAME NETWORK ENGINEERING

1.1. The Concept of Networked and Multiplayer Game

Network game by its definition is a type of game that include a network is a digital connection between two or more machines. It allows users, who are connected via network, to share one game world and influence on other user's behavior, decisions they make and their payoffs.

Multiplayer games are mostly a network games in that the players and the machines are physically separated and personal computers or consoles are connected by means of network. Yet, not every multiplayer, especially early ones, were network games. Generally, those multiplayer games made users to play on the same physical machine, for example one of the players take turns by marking a cell on the Tic-Tac-Toe game board, while the second player is watching. Once the first player marked a game board, the second player would have a turn. Scoring for each player is kept separately. For contemporaneously gameplay, either head-to-head or cooperative, each of the players would have its own avatar on the same screen or the screen would be split into several separate parts for each player in the game. Consequently, multiplayer games also include non-network games.

On the other side, not all network games can be considered as multiplayer games. A network can be used to connect player's machine to the remote servers that handles different aspects of the game. The game can be totally single play, it runs locally on PC and there is not interaction with other player's avatars, but a player into a server to obtain map content or interact with AI (Artificial Intelligence) units controlled by a remote server.

Accordingly, network and multiplayer games overlap (fig.1.1), but neither fully contains the other.

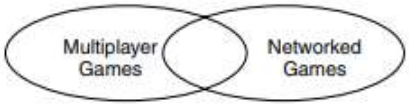

Fig. 1.1. The sets of multiplayer games and network games are overlapping, but not subsets or supersets of each other

1.2. The Overview of Early Networked and Multiplayer Game

Examining early online and multiplayer games can help to understand the context of the modern network games and it will show the importance of early multiplayer games that for the modern networking games.

The evolution of computer games, starting with first multiplayer and networked games and going to early multiplayer networked games is represented in the list below:

- 1960s is era of early multiplayer games (1958 Tennis for Two);
- 1970s is era of arcade multiplayer games (1961 Space War, Pong, Atari Football);
- 1990s and beyond is era of online multiplayer games (1993 Doom).

1.2.1. PLATO

PLATO stands for Programmed Logic for Automatic Teaching Operations and it was the first generalized computer-assisted instruction system. PLATO can be considered as the first network community that had users log into the servers and communicate or interact with other network participants from their terminals.

PLATO included many modern concepts of multi-user computer, it provided various communication mechanisms such as e-mail, forums, chat rooms, screen sharing and, of course, online games (the most popular games were Airfight and Emprire). First online games were networked in the way they connected to the mainframe, such as email client or remote login shell. Thereby, the game architecture had client (user machine) and server, that performed all communication and computation tasks.

The network performance of early systems was defined by the PLATO terminal communication with the server through the network protocol user by the Telnet program i.e. Transmission Control Protocol (TCP) that performs transmission of data users type with control information. Generally, client sends a user's character input and waits for the confirmation to display it on the screen. From the users point of view a performance measurement is the echo delay, it is a time that in takes to approve the segment previously sent. Using a TCP connection to echo the characters sometimes can lead to unpredictable response times to the user input.

1.2.2 Multi User Dungeons

Multi user dungeons is a virtual environment for the users, that provides interaction with the world and with other players with some gameplay elements and structure. Multi user dungeon is an online chat session that gives player multiple places to move and interact in like adventure game, and may comprise various elements such as combat, puzzles and traps, magical spells and some kind of simple economics. Early multi user dungeons had text-based interface (fig.1.2) that made users be able to enter some basic commands e.g. "cast a spell", "move north".

Fig.1.2 Screen shot of the Multi user dungeon interface

Multi user dungeon used a client–server architecture (fig.1.3), it means that administrator would run the server and player from the terminal would establish connection with mainframe with the help of Telnet program.

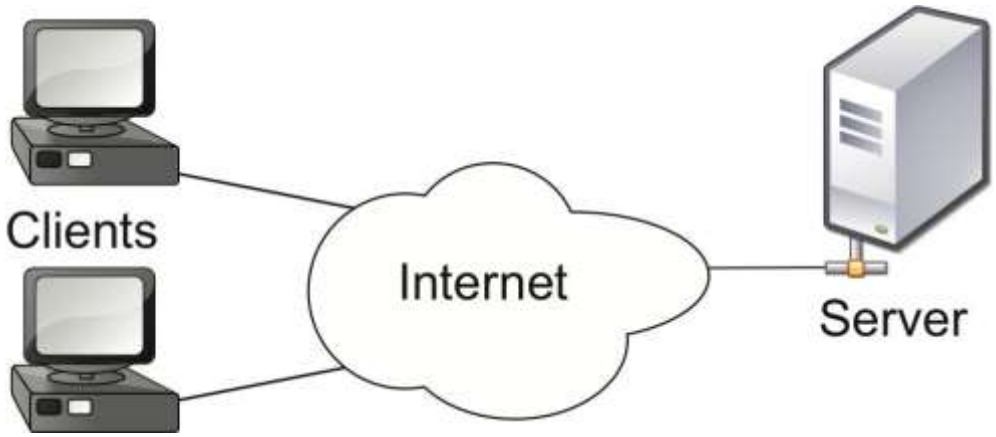

Fig.1.3 Client–server architecture used by early Multi user dungeon games

One of the major disadvantages of the Telnet was sometimes not quite effectively wrapping text lines and incoming messages could be printed in the middle of the command the user was typing. On the other hand, Telnet shortcomings led to the development of the specialized Multi user dungeons client applications that fixed some interface issues and provided some extra features such as text highlighting and providing various fonts.

1.2.3 Hosted Online Games

In the 1980s, the is idea of "pay to play" first appeared, with several game companies charging a monthly fee to play the games hosted on their servers. Companies decided to use the idle computer-cycles during nonbusiness hours by charging some payment to gain access to their computers to play games.

Such companies mostly provided text-based games that were rather popular, some of them were multiplayer e.g. Compuserv's Mega Wars I, a space battle that could handle up to 100 players simultaneously. The prices for the gaming were rather steep, in range from 5$ per hour up to 22$ per hour [5].

The high change for online game play has led to the emergence of the group of enthusiasts who hosted individual Bulletin-Board-Systems (BBSes). A bulletin board system is a computer an application or computer devoted to files and messages exchange. BBSes granted play by email or play by bulletin board system versions of famous table-top games such as Dungeon and Dragons or chess. Players connected to the BBSes by modem (typically by simply making a local phone call). There were

people who provided better gaming experience and charged money for playing, but at much lower prices.

Although commercial successes, the early computer games were fundamentally different from today's modern computer games. Players did not have anything near human-like avatars, or game-world environment, or character interaction. Players just moved simple geometrical shapes, maybe pushed buttons to shoot and something would happen it shapes collide. First games had poor graphics, the gameplay was totally different. Game-world did not have vehicles, weapons and even different levels. Immersivness is crucial for the modern game and, it is achieved by stunning graphics and fascinating gameplay mechanics, but in early games it was out of the question, it was coming only from the player's imagination.

Early computer games were relatively easy to produce both in terms of time and cost. This is in striking contrast with today's popular computer games, which take 18 to 24 months to produce and often have budgets in millions of dollars.

1.2.4 Multiplayer Network Games

By the early mid-nineties, computational power has significantly increased, allowing computers to create more realistic graphics and sound effects. Computer game players were no longer forced to user their imagination to obtain proper gaming experience. Instead of slowly moving a geometrical shape on the four color screen, players were able to move swiftly in the 256-colour environment and contemplate more realistic and detailed artificial world. Moreover, it was common for a computer to have network connection leading to the new area of networked games.

In 1993, the Doom game was released by id Software, that took a First-Person Shooter (FPS) genre to the next level, providing a brand new powerful engine that gave a violent and fast-paced shot-them-up gameplay with more realistic characters and levels that had been seen in any previous shooter games.

As a multiplayer game, Doom gave the players ability to play cooperatively in groups up to four people using the IPX protocol (fig.1.4), an early internetworking protocol from Novell) on a LAN, or playing in a competitive mode against each other trying to get more kills then their opponents.

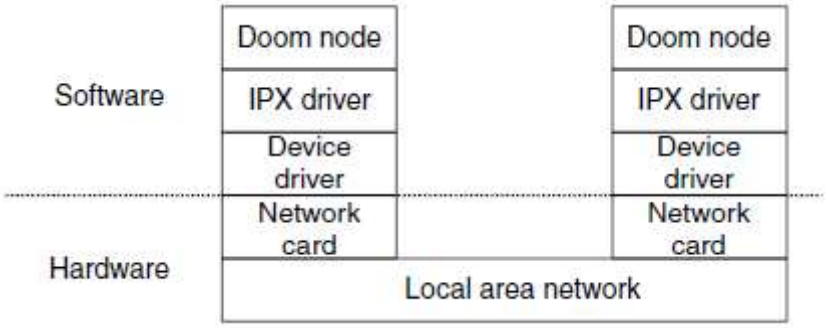

Fig.1.4 Hardware and software layers required to run multiplayer Doom

IPX packet was an internetworking protocol essentially for interconnecting LANs (fig. 1.4). It was often combined with Novell's Sequence Packet Exchange (SPX) forming the SPX/IPX stack – functionally equivalent to the TCP/IP stack, which is a basis of the Internet. SPX/IPX could not compete with TCP/IP for wide area performance, and has since all but was forgotten.

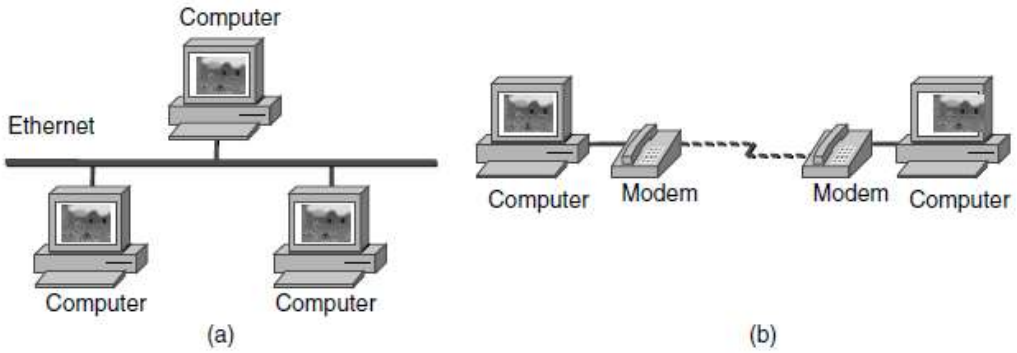

Fig.1.4 Network topologies used by Doom, computers connected to an Ethernet LAN acted as 'peers' (a), or computers connected by a modem acted as 'peers' (b)

Doom networking was based on the peer-to-peer topology. All players in the game are considered as independent "peers" running their own copy of the game and communicating with other Doom peers directly. Every $1/35^{th}$ of a second, each peer gathered the input data (such as move right, left, shoot) from each player and transferred it to all other peers in the game. After all data from all other players for that time period had been obtained, the game updated and its timeline advanced. Doom used sequence numbers to define if a data packet was lost. If a doom peer received a packer with unexpected number, meaning that data lost took place, it decides to resend request to sender.

Doom peers communicated was held by Ethernet broadcasts for all its network traffic. This led to a side effect that when some event happened e.g. player strafed, the packet sent by the game peer was received by all members of the network, but not only

by Doom nodes and all other machines on the same LAN were interrupted. The computer that are not Doom nodes would ignore this packet, further transfer it to the main memory and then operating system will decide that it does not need it. Normally, LAN traffic is addressed directly to a specific computer and it is whether not received by other machines or it is discarded by the network card before interrupting the processor.

The huge part of the processor time is wasted by the machines not taking part in a Doom game, but still processing network packets sent by Doom nodes. It is crucial for the slower computers; it could even cause some keystrokes drops. This was a heavy problem for the network managers, prompting companies such as Inter or AMD and many universities/colleges across the USA to provide some kind of ati-Doom policies in order to turn down the overload of the local computer networks.

The popularity of multiplayer mode of Doom, specially the death-match mode, significantly influenced the genre of nearly all FPS games to follow, both in terms of game play and in terms of networking code.

In 1994, id Software produced Doom 2, an impressive sequel to Doom. The game could support eight players and, more importantly, Doom's initial use of broadcast packets was removed, and this change brought with it a marked change in the acceptability of networked games on LANs and wide area links [6].

1.3 Networked Games Architectures

Investigation of the recent gaming software products brings up more detailed information about commutation architectures and their communication models. The overview of the possible gaming platforms and history of its development will explain improvements of gaming experience from the networking perspective.

In view of the communication architectures, it is need to explain the difference between game system level communication and network level communication. The network-level communication defines the way communication is conducted when the date is send over the Internet; it can be client-server or peer-to-peer. A game system

level communication describes the way how game pieces feel themselves to share game-state information; it can also be client-server or peer-to-peer.

1.3.1 Peer-to-Peer Architecture

Peer-to-peer architecture (P2P) (fig.1.5) is a computer network, where each node has the same capabilities and responsibilities, it means that each gaming node has equal control over the gaming process. In goes in contrast with commonly used client-server network architecture, in which a mainframe is devoted to serve other computers. There is no one intermediate node to route the game messages or control a game state. Peer-to-peer architecture is used in multiplayer games played on the Local Area Networks (LAN) and support rather small amount of players to take part in a game because of its limited broadcast. Peer-to-peer architecture can be applied to the Wide Area Network (WAN), in order to scale it additional hierarchical structure is required.

Primary advantages of peer-to-peer architecture are:

- It is easy to setup;
- All nodes are independent;
- More stable;
- Provides better distribution of network traffic;
- Does not require central administrator;
- requires cheaper hardware.

Also it has some drawbacks and they are the following:

- Typically, is less secure and is more difficult to secure;
- Harder to administer;
- Difficult to backup;
- Harder to locate information.
- Difficult to scale.

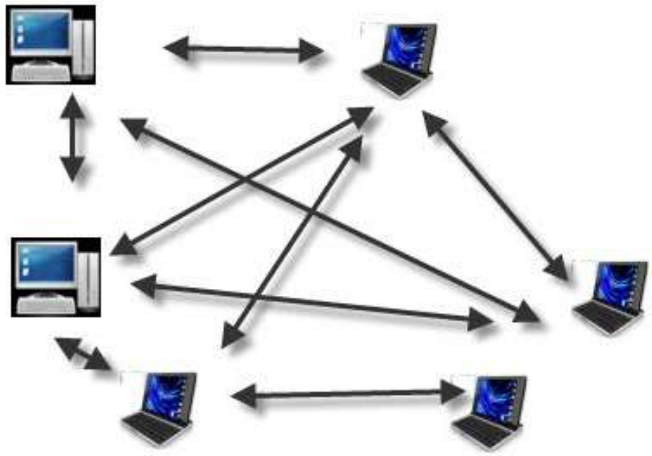

Fig.1.5 Peer-to-peer architecture

1.3.2. Client-server Architecture

Client-server architecture (fig.1.6) provides a models of multiple components, that work according to a strict communication rules. The server plays the role of the host, which manages most part of the resources and computation used by a client. Clients are not able to communicate with each other directly but rather user the sever to transfer messages from one to another. The server is a major component in of communication process; if a client is not able to communicate with a server, so the game cannot be played; is a server experiences difficulty with communication with clients it results in gameplay degradation for all clients connected to it. Server can become a bottleneck to the game performance, it means that it cannot handle receiving, processing and sending the game-state updated for all users fast enough.

Fig.1.6 Client-server architecture representation

Client-server architecture is particularly advantageous at the network level when mainframe server provides minimal handling of the game-state data and does not parse or modify the messages, leaving this to the clients.

As a pluses of client server architecture it worth to be mentioned:

- Easy to located resources thus they are located on mainframe server;
- Easily secured;
- Easy do administer;
- Are much easier to backup.

Client server architecture has a list of shortages:

- Servers is the only one point of failure;
- More expensive hardware required in comparison with peer-to-peer architecture;
- Network traffic can be overloaded.

Nevertheless, client-server architecture is the most commonly used architecture in online games, it was used be the early online Multi uses Dungeon games previously described in this part.

1.3.3. Peer-to-peer Client-server Architecture

Peer-to-peer Client Server architecture (fig.1.6) possesses that server receives and processes some intermediate game states sent by the clients as in typical client-server architecture and are also able to communicate with each other as in the regular peer-to-peer network. As an example, generally in multiplayer network games a player voice communication is done with peer-to-peer by means of Voice over Internet Protocol (VoIP) and processing the commands to control client's avatar goes to the server side.

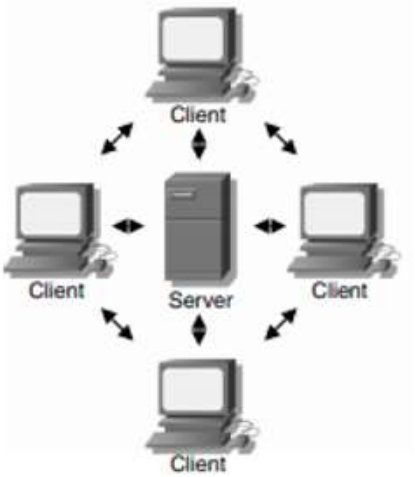

Fig. 1.6 Hybrid network architecture

1.3.4. Network of Servers Architecture

Network of servers' architecture (fig.1.7) may be considered as a way of solving an inability of a single machine to keep up with processing of the incoming traffic from the client's side. A single server can become a pool of interconnected servers with a peer-to-peer connection between them, all the servers have equal, or implemented in a client-server way, where servers communicate with master servers, getting hierarchal game architecture. By balancing the load coming from the client's side, can decrease the requirements for a single server. Implementing network of servers significantly increases the game architecture, but can require a specific and more complex communication system between servers with some additional difficulties in maintaining game-state information consistent.

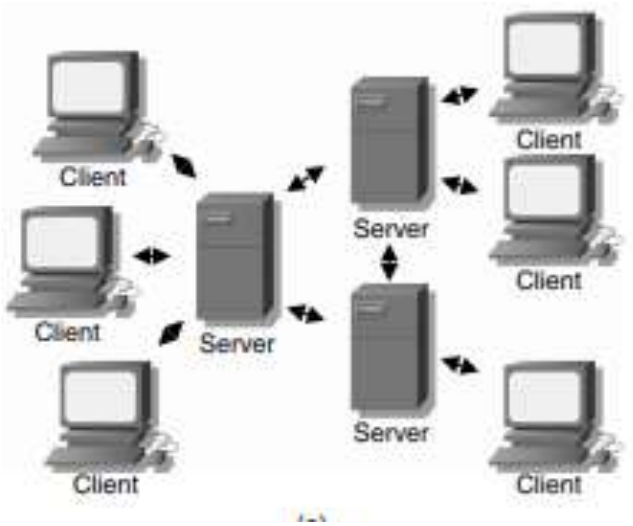

Fig. 1.7 network of servers

1.4. Network Protocol as a way of networked game communication

Transferring and receiving data is one of the basic concepts of networking engineering and It is a major task for the network developers.

Typically, a mainframe server is running on a specific computer and has a socket, that is devoted to a specific port number. The server is constantly waiting, listening to the socket for a client to make a connection request. Also a client needs to identify themselves to the server, so the server can bind a specific local port number that will be used during the connection process (fig.1.8).

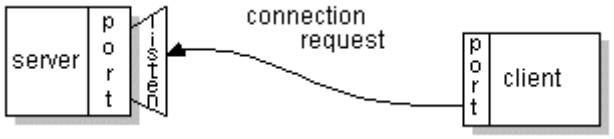

Fig.1.8 Connection request from client to the server

In case of successful connection between server and client (fig.1.9), a server receives a new socket related to the same local port and has its remote endpoint set to the address and port of client's side. If the connection was accepted on client's side a socket will be created and client is able to use the socket to communicate with the remote server.

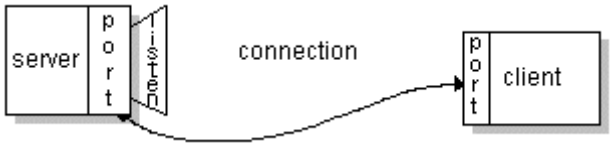

Fig.1.9 Established connection between server and client

Now the client and server can communicate with each other by reading and writing to their sockets. So socket is one endpoint in two-way communication link between two application running via a network; it is devoted to a specific port number in order to give TCP layer ability to identify the desired application that data need to be sent to [5].

Most game developers will use an IP in combination with either the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). UDP and TCP are transport protocols, created to provide another layer of abstraction on top of the IP layer's network service, supporting the concurrent multiplexing of data from multiple applications onto a single stream of packets between two hosts. The type of socket entirely depends on the type of game that is developed.

1.4.1 Transmission Control Protocol

Transmission Control Protocol is a protocol that utilizes the principles of the reliable connection, meaning that connection between two remote machines is established and the data transfer takes place; it can be compared to the writing some information on one computer in a file and reading its content from another computer from this file. The connection is reliable and consistent at the same time, all the data is

delivered to the endpoint and in the exact same order as it was sent. Also TCP can be described as continuous data stream, it handles data splitting into packets and its transfer over the network on its own.

TCP goes above the Internet Protocol (fig.1.10) and creates bidirectional routes between two endpoints. TCP uses the abstraction of the data stream and allows to write bytes of data in this flow, then data is spited in a TCP data frames, which is a part of IP packet, that is transferred over the network right to its destination point.

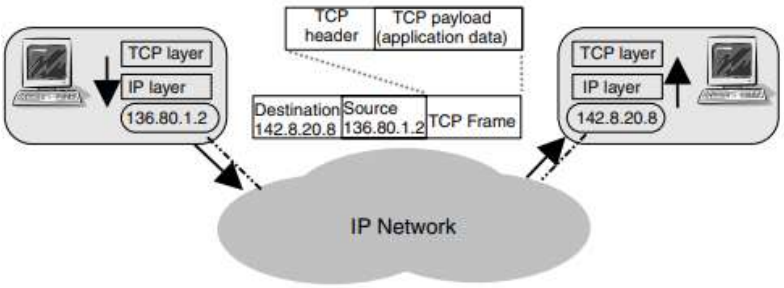

Figure 1.10 TCP runs across the IP network

The destination point's TCP layer explicitly approves received packets, allowing the transmitting TCP layer to detect data loss. In case of data loss, TCP frames will be retransmitted until acknowledgement from the client's side will not be received; it shows that data is sent with a high degree of reliability. TCP gathers some amount of data in queue and if some specific limit is reached, it forms a data frame and immediately sends it. This approach could lead to a problem in a multiplayer networked game if a transfer of small packets is required, it could happen that TCP will refuse to send data until some amount of information e.g. more than 100 bytes is gathered. For example, the game needs to send information about each pressed keyboard button to the server as fast as possible, but delays due to data buffering will occur and a player on the client's side will fell gameplay degradations it will ruins gaming experience. Update of the game world state will take place rarely and with some delay. TCP has an option "TCP_NODELAY", that tells the protocol to ignore accumulation of data and send it immediately, but unfortunately even with this option enabled it brings a lot of problems in networked multiplayer games.

The problem is in how TCP synchronizes the data flow, if a packet is lost, transmission stops until the lost packet is resent and received by the destination. If new data arrives while waiting, it will be queued and you will not be able to read it until the

same lost packet arrives. All this operation takes time; it is equal to the roundtrip time (fig.1.11) of the packet plus the time re-deliver the lost package. The delay varies from 125ms up to 500ms; those are enormous indicators for multiplayer networked game.

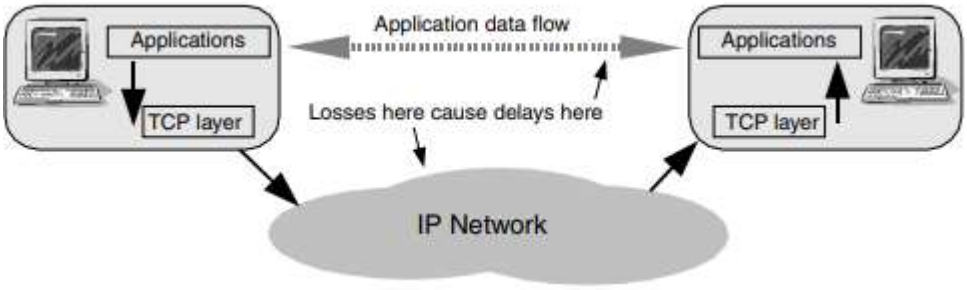

Figure 1.11 TCP converts IP layer packet loss into application layer delays

Let's see a simple example of multiplayer networked game as FPS. The network part of the game work in this way: each game tick a client sends a data to the server, containing all player's actions (mouse position, keys pressed and soon on) and after this a server processes all the information and updated the game world state and gives back this information to the clients and gives a new picture of the virtual environment. If the data packet was lost, while transferring over the network, the game freezes and waits until the needed packet will be received. On the client's side nothing happens in this time intervals. When a lost data frame arrives, it contains outdated information. Also, retransferring of the packet created a queue of other packet, that have reached the client and all they need to be processed in one game tick, it turns out to be a complete mess.

To sum up: TPC uses reliable connection, guarantees the order and delivery of the data packet to its destination, automatically breaks information into packets, controls the dataflow (prevents excessive data sending) and is easy to use.

1.4.2 User Datagram Protocol

User Datagram protocol is a simpler version of TCP that provides unreliable, connectionless datagram oriented transport service, it imposes no flow control on the packet transmission and does not implement packet loss or recovery mechanisms.

By means of UDP we can sent a data packet to a specific IP address and port and will transfer over the network until it reaches the endpoint or will get lost. From the client's side perspective, we wait and listen to a port, if the packet arrives we receive a notification with and address and port of a sender, size of a packet and after that

process it. As mentioned above, UDP does not ensure data delivery, almost all data frames reach its destination, but 1-5% of them are lost. This protocol does not engage for the order of packets delivery, five packets can be sent in order 1, 2, 3, 4, 5 and the actual result will be 3, 2, 1, 5, 4. Generally the order will be right, but you can't rely on it.

One thing should be mentioned, UDP guarantees that if a package is forwarded, it will be either arrive to the endpoint or will not arrive at all. So, if a packet of 256 bytes is sent to another machine, then it cannot receive only 50 bytes from a data frame – it must obtain all 256 bytes.

To sum up: UDP does not use reliable connection, does not guarantee delivery of the data packet to its destination, they can reach endpoint in a wrong order or with duplicates, information should be broken into packets manually, control of the dataflow (prevents excessive data sending) is also should be implemented by engineers, if the packet is lost it won’t be detected and resending of the data relays on the developers.

1.5.2 TCP or UDP

For the real-time networked multiplayer games, the world state data, such as user’s clicks and so on, only up-to-date information if is relevant, but for other types of data, such as sets of instructions sent from one client to another, the consistency and reliability of the channel can be crucial.

Of course, the possibility of using both UDP and TCP exists, transfer user input and world state data over the first one, and the second one for the data the that definitely needs to be delivered. The multiple streams of commands can be created, for example one for the performing instructions of the Artificial Intelligence, another for the loading levels. AI instances don’t need to wait in case data packet for level loading was lost, because they are completely separate. In this case, creating of a separate TCP socket for each command steam makes sense.

At first sight, the idea of using both TCP and UDP is good, but the problem is that both this protocols run on top of the Internet Protocol, and the packets of both protocols with affect each other on the IP level. It is hard to say in which manner this

negative effect will show up, and it relates to the TCP reliability mechanism. Also usage of the TCP leads to the increase in UDP packets loss.

A proprietary UDP-based protocol can be more efficient than TCP for various reasons. For example, it can mark some packages as trusted and others as unreliable. Therefore, he does not care if the unreliable packet reached the recipient. Or, it can handle multiple streams of data so that a packet lost in one stream does not slow down the backup streams. For example, there might be a thread for player input and another thread for chat messages. If a chat message that is not urgent data is lost, then it is not an urgent input trigger that is urgent. Alternatively, a proprietary protocol might implement robustness differently from TCP to be more efficient in a video game environment [7].

Even though TCP is almost suboptimal for gaming network systems, it can work quite well for your specific game and save you valuable time. For example, latency might not be an issue for turn-based play or a game that can only be played on LANs, where latency and packet loss are much less than on the Internet.

It is highly recommended not just using UDP, but using only it and nothing else. It is better to implement needed TCP tricks by means of UDP.

Conclusions on the First Part

The development of the technologies brings up a huge impovement in the game engineering field. Computer networking is a branch of computer science, telecommunication and computer engineering and its was influenced by a wide range of the technologies and historical events.

Overview of the early networked products sheds light on the basic network architectues, that are used till the modern days:

- Peer to peer;
- Client-server.

Refrences the problems ecountered by the pioneers of the field. Defines the basic communication protocols:

- TCP;

- UDP.

Taking into account their pros and cons it is recommended not just using UDP, but using UDP only and nothing else, better not to use TCP and UDP together — instead implementation of the needed TCP tweaks is more preferable using UDP.

The choise of the communication protocol is not the only problem faced by the multiplayer game developers. Research and analysis of the potentional network problems, sush jitter, loss, lag, serialization delays are need to be taken into account and are covered in the Part 2.

PART 2

THE ANALYSIS OF THE GAME NETWORKING PROBLEMS

2.1 Network Systems Characteristics Overview

The realism of multiplayer game depends on how well network part allows all game clients to communicate in timely and predictable way. Considering the field of networks, essential features and properties need to be defined and summarized by the concept of metrics. Those metrics of communication network are properties that allow to accurately estimate a communication network as a whole.

Let's consider major metrics; the first one is bandwidth, it determines the maximum amount of data that can be sent per one unit of time, it also can be characterized as network, data or digital bandwidth. The next one is Packets Per Second (PPS) says how many data frames could be transferred over a second. Frame loss – amount of packets that don't reach the endpoint, generally is 1-5%. Delay or latency describes what time does a packet need to reach its destination point; there is one-trip it is a travel time from point A to point B, and round-trip – time travel from A to B and back A. Also a latency can vary on jitter, it shows the deviation of latency of the first data fame relatively to the second one. Maximum Transition Unit (MTU) – this metric is crucial for high loaded gaming products. And the last one is normalized maximum bit rate.

Among all this list latency, jitter and loss – have a significant effect on the game design and overall gameplay experience. Brief look at technical methods of Internet Service Providers (ISPs) to control this characteristics control of their network services will be done.

2.2 Relevance of Latency, Los and Jitter

As it was previously mentioned, IP packets transfer information between two points, source and destination, on the network. Latency is a time needed to carry a data packet from a sender to a receiver. Round Trip Time (RTT) refers to the delay from the source to the endpoint and back again. In lots of cases, RTT is twice as latency, but it's not always true, some network routes show asymmetric latencies, it means that the latency in one direction is higher than in another. In an online game developer's community, the term lag is used to describe RTT.

There are multiple ways to describe jitter, but the main is a rapid phase change of a signal. In computer networks it is called "changing ping", the latency changes and the direction in which it goes over the time. Let's take an example, all the packets reach destination over 2 milliseconds or 200 milliseconds, meaning there is no any jitter. Let's have a look at another situation, some network path shows average latency of 90ms, but it could be different for each individual frame (80ms and 100ms), so it means there is a jitter in the network it can be viewed in the short term, but the overall latency is rather constant [8]. For the online game development, the slow or instant latency oscillation from one to the next packet can occur. The difference between jitter and loss is represented on the figure (2.1).

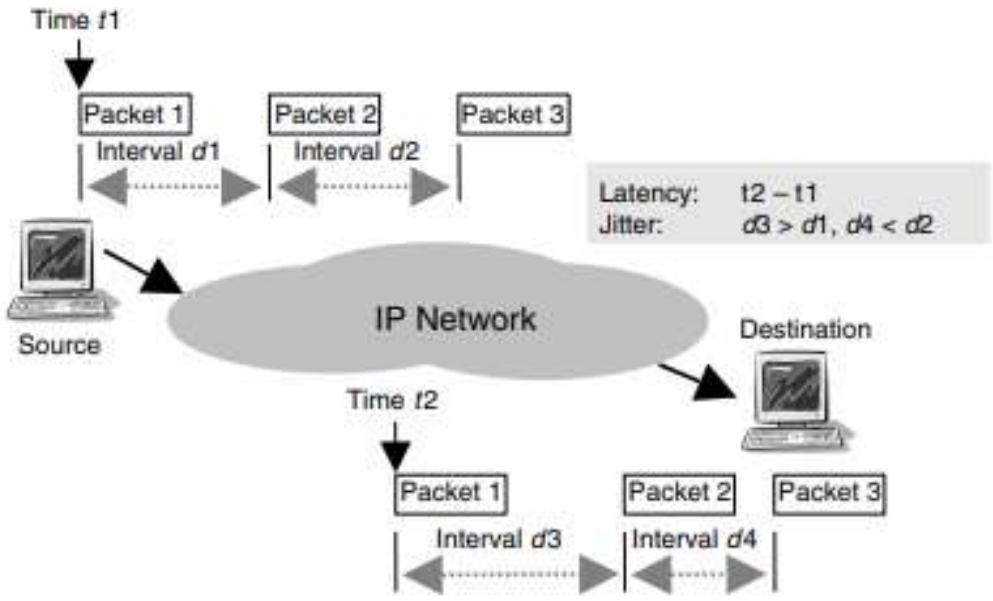

Figure 2.1 Key difference between latency and jitter

Interval d1 and d2 represent the time when those two packets were sent, d3 and d4 stand for the arrival time of each individual packet, t1 and t2 show the overall time when the packets were sent a reached the endpoint. The expression t2 – t1 gives us a latency, and difference between d1, d3 and d2, d4 represents a jitter.

Packet loss (fig.2.2) stands for the situation when one or more data packets don't reach its destination while travelling over the network. It is caused by data transmission errors over the wireless networks or because of the network overload. Ratio between number of lost packets and amount of packets sent gives a packet loss rate or packet loss probability.

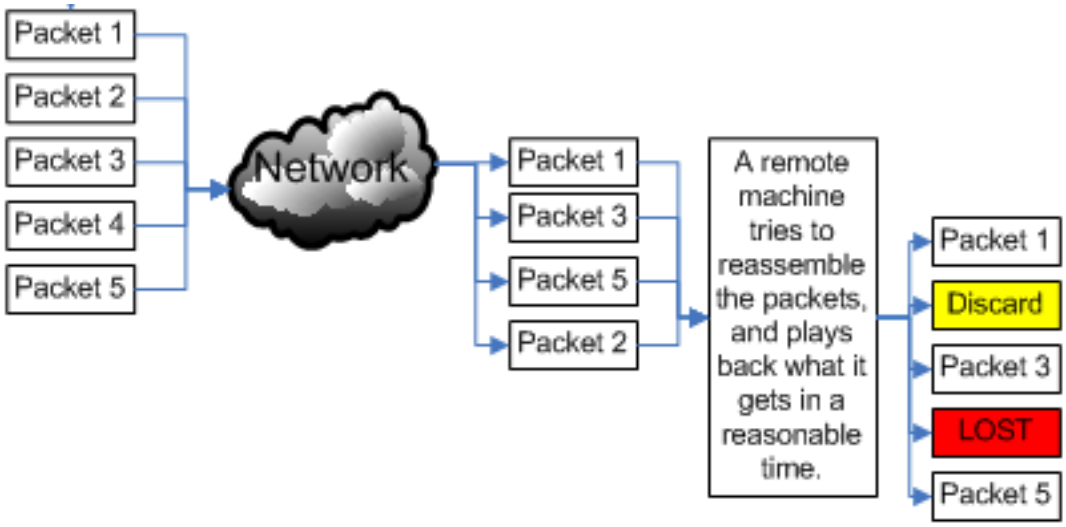

Fig. 2.2 Semantical representation of a packet loss

All this listed aspects have a negative effect on the online gaming. Latency can downgrade the sense of real-time presence in the game, it sets a lower bound of how fast a player can obtain update information about game state and restricts player's ability to react to changes in the virtual world. Jitter makes it difficult for the game and engine and a player itself to overcome and compensate form the average network latency, it should be kept as the lowest possible point. The result of a packet lost a quite obvious, the game engine, client and server need to get the game world update and fix the loss as fast as it can.

2.3 Sources of Network's Latency, Jitter and Loss

In addition to the three main delay sources the following list adds on some additional delays to the packet delivery process:

- Limited propagation delays over huge distances (laws of physics should be taken into account);
- Serialization delays (especially over low bit rate links);
- Queuing delays caused by network congestions;
- Changes in path length;
- Packet size variations;
- Transient congestion, typical cause of packet losses;
- Excess temporal network congestion causing queues overflow;
- Packet corruption;

Routing transients temporarily disrupting the path.

2.3.1 Laws of Physics and Propagation Delays

The speed of light determines the maximum speed at which any information can be transferred in a particular medium e.g. fiber, air. The transfer speed of copper cables and wirer is usually lower than the speed of light in the air, depending on the specific physical design of the cables e.g. dielectric characteristics of the insulation affects it. Thus, the speed of light is finite, the laws of physics set lower limit on the delay between demographically two far located points on the Internet, it is called propagation delay.

Since the speed of light is circa $3*10^8$ kilometers per second, propagation delay of data frames could be noticed over the lines that cover thousands of kilometers or the path goes through numbers of router that are separated by thousands of kilometers. Let's see on the example, a 10,000 km trace, will show at least 35-ms delay and 70-ms RTT (we double a latency value), because of the light transfer limitations. It makes a problem for the players that connect to the servers, that are located in a different state or county, or the use high latency path between two close sites, because they are connected to the Internet by means of different IPSs.

A simple formula can be applied to calculate the latency (2.1):

$$latency\ (ms) = \frac{\text{distance of link (km)}}{300\ (\text{km/sec})} \tag{2.1}$$

If the speed of light in the medium will be lower than 300km/sec, so the latency will be higher, as an example the speed of light in optical fiber is 30% slower that in a vacuum.

2.3.2 Serialization

Serialization is a process of converting the data or some data structure into a sequence of bits and is used to transfer those objects over the network and later reconstructed in a different computing environment. Serialization takes place on link layer and is another source of the delay in IP networks.

Data frames are broken into sequence of bites and those bytes are sent one bit a time; the time period needed to feed one bit to the IP packet at a time is serialization latency. Length of the packet and speed of link affect the time needed to send of bit. The serialization latency specific latency in addition to the propagation delay.

The overall serialization delay of the data packets also depends on the framing protocol implemented in a specific link layer. Some extra bits at the beginning and end of the byte could be added (typically serial port) or the beginning and the end of the frame, depending on the link layer technology. Let's have a look at the time needed to convey a 1500-byte IP packet on a 100-Mbps Fast Ethernet LAN and a nominally '56-Kbps' V.90 dial-up connection via Point-to-Point Protocol (PPP).

On a V.90 dial-up link, the uplink limit is 33.6 Kbps and the downlink rarely goes beyond 51 Kbps. If we further assume PPP encapsulation of 8 bytes, the 1500-byte Internet Protocol packet demands 1508 × 8 = 12,064 bits to transmit. Yet, a 1500-byte IP packet takes 359ms to send towards the ISP (upstream) and 237ms towards the client (downstream).

On the Ethernet link, a 1500-byte IP packet turns into 1526 bytes long, additional 8 bytes of Ethernet preamble, 12 bytes for source and destination MAC address, 2 bytes' protocol type and 4 trailing bytes Cyclic Redundancy Check (CRC), or 12,208 bits. At 100 Mbps, it takes 122 microseconds to transmit the frame along with this packet [9].

Serialization delay becomes an issue with low-speed links; taking into account previously calculated number, even a small 40-byte packet takes 11.4ms on a V.90 upstream and 7.5ms on a V.90 downstream.

The same situation occurs with high-speed links when your ISP sets temporary speed limits. For example, imagine an ISP is using ADSL2 + to offer a 4 Mbps downlink service to a customer who has reached their download limit per month. The ISP applies the 64 Kbps speed set at the IP packet level until the end of the month. While each packet is still individually transmitted at 4 Mbps, the ISP achieves a long-term 64 Kbps rate limit, limiting the number of packets per second that can be sent. The effective serialization delay is as if the ADSL2 + link was literally 64Kbps [10].

Serialization delay needs to be calculated only once, since the endpoint strips bits from the channel at the same rate as the transmitting side sends them. Except for a slight time offset due to propagation delay, transmission and reception are performed almost simultaneously. The formula for serialization delay calculation (2.2):

$$latency\ (ms) = \frac{8 * (\text{fame length in bytes})}{\text{link speed in Kbps}} \qquad (2.2)$$

Worth to be mentioned that some communication technologies such as Asynchronous Transfer Mode (ATM) and Data-over-Cable Interface Specification (DOCSIS) the ration between link-layer frame is IP packet length in non-liner and hard to compute.

2.3.3 Queuing Delays

In a computer networking a queuing delay is a time that passes until the job can be executed and it is a key part of a network delay. Queueing delays can considerably the latency and also can make those delays unpredictable as well, it makes it hard to define the size of the buffer that the network interface should carry. One of key concepts of the Internet's best effort approach is that users traffic is unregulated and bursty, that gives us an opportunity to use a notion of statistical multiplexing. Is happens when several incoming packages come together at the same outbound channel in a specific router or switch, those packets are mixed up in terms of time. IP routers of an ordinary telecommunication companies do not provide guaranteed time intervals on

the unbound links for the competing incoming data streams. Generally, statistical multiplexing assumes that average bit rate will not be more than a capacity of the inbound link and everything will be fine, most of the time packages will not arrive to the endpoint at the same time. But sometimes it could happen and in this case all the packets are queued up and are transferred one after another.

Queue begins to emerge when the traffic is incoming faster that it can be processed and the number of delays increases, the average delay in processing of the queue is expressed by the formula (2.3):

$$avetage\ latency\ (ms) = \frac{1}{(\mu - \lambda)} \tag{2.3}$$

Where μ stands for the number of packets per second and λ is an average speed of the incoming packages. This formula can be applied to the case when there are no packets dropped for the queue.

As a result, every packet in the queue will experience the additional delay due to the serialization latency of each packet, that goes before it. Let's take a look on some real example, several computers on the LAN are trying to transfer packages via the same ADSL or cable modem, thus a queuing delays may occur, packets will have to wait for their turn to be sent on the upstream link.

Another example, queuing delay can take place on a shared links, that allow to send data only for one host at a time, and the link access protocol works to separate transmission capabilities between connected hosts. Let's have a look at 802.11 b / g wireless LANs or WiFi networks. Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) and 4-way handshaking protocol create variable access delays that depend on the traffic load on the wireless network (the number of clients and / or the number of packets per second heading off). For example, 802.11 b networks have been shown experimentally to add 50 to 100ms to RTT under heavy load due to bulk TCP file transfers [11].

2.3.4 Jitter Sources in the Network

Technically, jitter is also called packet delay variation. It refers to the difference in latency in milliseconds (ms) between data packets over the network. This is usually a

violation of the normal sequence of sending data packets. It also means that the latency of packet transmission over the network fluctuates.

Too big jitter sometimes causes additional digital signal errors or even out-of-sync. Possible reasons are:

- Parasitic phase modulation in clock generators;
- Noise effect and interference on the synchronization circuit in the receiver;
- Changing the length of the transmission path;
- Change in propagation speed;
- Doppler shift from moving objects;
- Irregular receipt of timing information, etc.

The level of latency will fluctuate throughout the transit and may result in a packet transfer delay of 50 milliseconds. The result is network congestion due to the way devices compete for the same bandwidth. Therefore, the more congested it is, the more likely it is that packet loss will occur (fig.2.3).

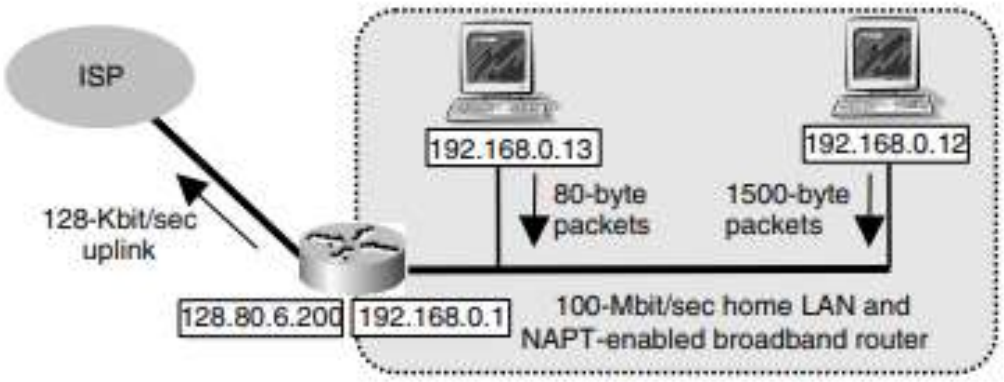

Figure 2.3 A congested uplink can introduce jitter through queuing streams of different sized packets

Let’s have a look at two machines on one 100-Mbps Ethernet LAN that are connected to the Internet via broadband 128 Kbps link. First computer produced a data stream of the 80 bytes IP packet each 40ms, let’s suppose that link layer adds fixed 10 bytes to each frame and exceeding it to the 90 bytes. It takes 5.6ms to transfer these 90 byte frames over 128 Kbps channel. Suddenly the second machine produces a 1500-byte data stream each 1000ms. Those packets need circa 188.8ms to be sent. Now the data flow from the first computer will suffer from random jitter, most of the time going through the link immediately, but sometimes with a 188.8ms delay, in addition to its

average sending time. At the moment when the packet from second machine arrive to the outbound link, the packages from the first machine will be queued, until the 1500-byte IP packet will be sent and the queue will be empty.

There is one additional source of jitter that should be mentioned, that only affects Internet Protocols over PPP/High-level Data Link Control (HDLC) over serial synchronous and asynchronous communication links. Those links area unable to set the boundaries of the beginning and ending of the frame, HDLC technique ensures safe identification of frames boundaries over serial links. The unique sequence if bits is added as a flag, e.g. 01111110 in binary or 0x7E in hexadecimal, ensuring that those bits were never seen in the frame sequence. In synchronous framing, because a flag consists of six 1-bits, all other contain no more than five 1-bits in row, it is achieved by bit stuffing (adding of bits that don't carry information in the data flow), the stream is stopped and 0-bit is sent. The receiving side known what happened and after facing five consecutive 1-bits, 0-bit is deleted from the data. If the last bit is set to 1, then it is an indicator in case if the seventh bit is set to 0, or it could be an error in data stream and receiving procedure stops and will be restarted when next flag will be spotted. Bit-stuffing technique guarantees the sufficient number of signal striations, 0 indicates the signal change, 1 as no changes occurred.

As an example, we transfer an UDP traffic containing the value 0x7E two hundred times and it is 238-byte UDP/IP/PP data frame. As a result, two hundred 0x7E bytes will double and form a 438-byte frame, and a serialization latency over the link can be randomly affected by HDLC control bytes in the packets [12].

2.3.5 Packet Loss Sources in the Network

Packet can be lost during its sending by many reasons, it can reveal itself in slowing down of services or disconnected network connections, and complete loss of network connectivity. Packet loss can affect any application, but is more likely to disrupt those that rely on real-time data transmission, such as multiplayer networked games, that require instant updates. Loss of packets necessarily happens from time to time due to constant use and high demand, packages get confused or lost along the way, and here are some of the most common reasons.

A network can reach its maximum bandwidth and become overloaded and they are more likely to experience packet loss because of the increased traffic income. Thus the packet transfer process follows specific steps, connection failures lead to some packet loss so that the network can handle the incoming load. However, as modern technology advances, many applications and programs are now able to process rejected data using a different method, which involves slowing down the transfer rate or automatically forwarding lost data packets.

Physical layer of the links is non error-prominent and extremely low rate of data corruption exists. Bad signal-to-noise ratios in the digital-to-analogue to-digital conversion process, leading to the inappropriate encoding or decoding of data. Simple electrical glitches in the hardware can be a reason. Forward error correction technique (FEC) can be applied to add some information within each data frame to provide partial reconstruction after one-two bit errors were found.

Software bug is another reason for the packet loss on the network, not properly tested applications are more likely to cause network problems and affect their packet transmission. Restarting the software will resolve this issue, but program updates or full application patching is required to solve the problem completely.

The traffic on your network may be stable, but the CPUs of the individual devices are heavily loaded and unable to process all the incoming data. The result is packet loss. The inability of the CPU to fully process information becomes a bottleneck for the entire network and negatively affects performance. Devices and their components sometimes crash. Errors in the operation of the RAM or network card may occur. With high traffic, even small hardware errors can have significant consequences.

There are several potential hardware or software issues that can significantly affect incoming traffic to the network. When legacy hardware devices are used to start the system, packets can be lost due to slow data transfer. Companies and individuals are encouraged to continually update or upgrade their hardware to optimize the performance of network processes. This is necessary to avoid network delays, packet loss, or even complete loss of connection to the system.

Security leaks and network threats can also cause packet loss. Recently, cyber-attacks known as packet drop attack have become popular among cybercriminals. Some

people send commands that send packets of data into the data stream. These malicious users can do this by gaining access to the network router. These types of attacks can be identified by monitoring the packet loss rate on the network. A sudden jump in these statistics could be a sign of an online attack.

The last one reason is changes in a dynamic routing, that do not always find a not high loaded and fully functional path to the endpoint. Some delays can take place after route changes (starting from tens of seconds up to minutes) until a valid shortest path between source and destination will be found.

2.4 Control of Network Lag, Jitter and Loss

Multiplayer networked games are mostly a real time-time in the interaction meaning, they are more demanding to the jitter, loss and latency control over the network than chats, email and web-surfing applications. Let’s cover up the existing mechanisms used by Internet Service Providers (ISPs) to control network state on behalf of the player and challenges faced by ISP in using those techniques effectively.

One way for the ISPs is to make their router and links bandwidth much bigger than the traffic that they get, and use creative routing technique to make sure that any single router will not be overloaded. This approach sounds kind of easy to implement, but in reality it could be implemented by huge ISPs companies, who have access to the physical channels infrastructure, typically a telephone company who owns underlying optical fiber between cities.

Deploying more bandwidth will not work out in the territories where they could not be deployed because of cost-effective factor. Marking some packages as first priority, that require special service could be a problem solver, but how exactly those packages will be different and what the special service should be thought over.

2.4.1 Preferential IP Layer Queuing and Scheduling Techniques

Queuing mechanisms are used in any network device where packet switching is used - a router, a LAN switch, an end node. The need for a queue arises during periods

of temporary congestion, when the network device does not have time to transmit incoming packets to the output interface. If the reason of overload is processor unit of the network device, then unprocessed packets are temporarily placed in the input queue, that is, in the queue on the input interface. In the case when the cause of the overload lies in the limited speed of the output interface (and it cannot exceed the speed of the supported protocol), then the packets are temporarily stored in the output queue.

Estimating the possible length of queues in network devices would make it possible to determine the parameters of the quality of service with the given characteristics of the traffic. However, changing queues is an unpredictable process that is influenced by many factors, especially when complex algorithms for processing queues in accordance with given priorities or through weighted service of different flows. Queuing analysis deals with a special area of applied mathematics - queuing theory. Therefore, the Quality of Service (QoS) uses a complex model that solves the problem in an integrated manner to maintain a guaranteed level of quality of service. This is done using the following methods:

- Reservation of bandwidth for traffic with known parameters (for example, for average values of intensity and size of a packet block);
- forced profiling of input traffic to keep the device load factor at the desired level;
- Usage of complex queue management algorithms.

A specific set of rules defines the combinations of IP addresses, port numbers, that require special treatment. According to this rules a packer can be recognized as high-priority or normal packet on get into corresponding queue.

The most common queuing algorithms used in routers and switches are:

- First In - First Out;
- Priority Queuing or overwhelming;
- Custom queues algorithm;
- Weighted Fair Queuing (WFQ).

Each of the listed algorithms was created to solve specific problems and as a result they affect the quality of service of different traffic types in a different manner. Simultaneous application of those algorithms is possible.

2.4.2 First In – First out

The principle of the FIFO algorithm (fig.2.4) is as follows. In case of congestion, packets are placed in a queue, and if congestion is eliminated or reduced, packets are sent out in the order in which they arrived (First In - First Out). This queuing algorithm is the default for all packet-switched devices. It is distinguished by its simplicity of implementation and the absence of the need for configuration, but it has a fundamental disadvantage - differentiated processing of packets of different streams is impossible. FIFOs are essential for network devices to function properly, but they fail to support differentiated QoS.

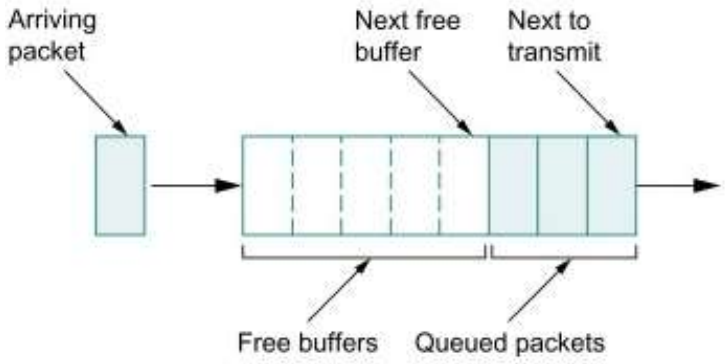

Fig.2.4 FIFO algorithms adds packet to the queue

If the queuing buffer will be overloaded, the last incoming packet will be dropped due to the tail drop policy (fig.2.5).

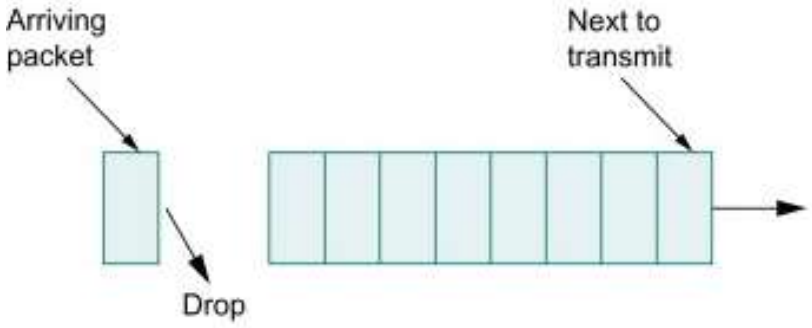

Fig.2.5 FIFO tail drop policy

2.4.2 Preferential Queuing

The mechanism of priority traffic processing provides the division of all network traffic into a small number of classes with a certain numerical attribute its identification, so that all subsequent network devices can use it to process traffic along its route. This field is found in the header of many protocols. For example, an IP packet provides a three-bit IP Precedence subfield in the Type of Service (TOS) field for this purpose. If a special priority field is not provided, a new header with such a field is added by means of specially designed protocol. In particular, for the Ethernet protocol (and other protocols of the 802 family), the IEEE 802.1Q / p specifications have been adopted, which define an additional three-bit priority field.

The rules for classifying packets into priority classes are an integral part of the network management policy. Most part of the modern routers are ably to distinct IP packets, utilizing five pieces of information from its inside information, this rule is called flow classification or 5-tuple classification. Those pieces of information are the following:

- Packet’s source address;
- Packet’s destination address;
- Protocols type, whether it is UDP or TCP;
- Source port number;
- Destination port number.

Regardless of the chosen method of traffic classification, the network device has several queues, according to the number of classes. The packet arriving during the congestion period is placed in the queue according to its priority. Figure 2.6 shows an example of using a non-preferential queuing and preferential for the packets transmission. In the second case, until all packets are selected from the higher priority queue, the device does not proceed to processing the next, normal priority.

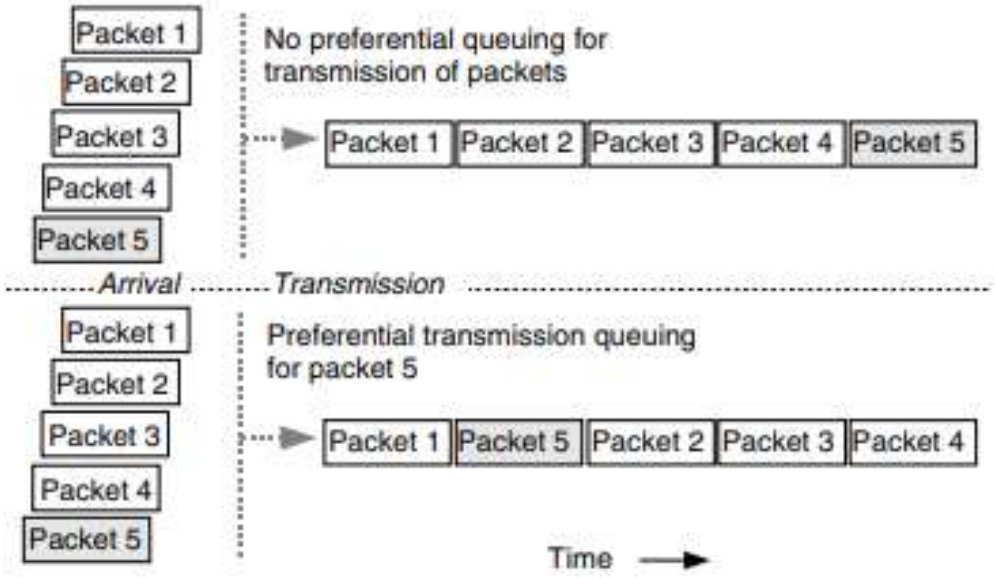

Fig.2.6 Preferential queuing allows priority packets to 'jump the queue'

The finite size of the buffer memory on a network device assumes some limit on the length of each queue. Typically, by default, all priority queues are allocated the same size buffers, but many devices allow the administrator to allocate a separate buffer for each queue. Its maximum length determines the limit on the number of packets that can be stored in the queue of this priority. A packet arriving while the buffer is full is simply discarded.

Priority queuing ensures high quality of service for packets from the highest priority queue. If the average rate of their arrival at the device does not exceed the bandwidth of the output interface (and the performance of the internal blocks of the device itself involved in packet forwarding), then the packets with the highest priority always receive the bandwidth they need. As for the rest of the priority classes, their quality of service is lower than that of the packets with the highest priority, and it is difficult to predict the level of reduction. It can be quite significant if high priority data is transmitted at high rates.

Therefore, priority service is usually applied when there is delay-sensitive traffic on the network. So for the multiplayer networked game such a preferential system will allow to send and receive all the needed game state data in time and will not degrade the overall gaming experience.

2.4.3 Weighted Custom Queuing

Weighted Queuing is designed to provide a minimum bandwidth or latency requirement for all traffic classes. The weight of a class means the part of the output interface bandwidth allocated to this type of traffic. Custom Queuing algorithm allows an administrator to assign the weight of the traffic classes. Weighted Fair Queuing (WFQ) algorithm is implemented to assign the weights automatically, based on some adaptive strategy.

In both weighted and priority service, traffic is divided into several classes, and a separate packet queue is created for each class. Each queue is associated with a fraction of the output interface bandwidth guaranteed to a given traffic class when this interface is overloaded. As an example, the device supports five queues for several traffic classes, queues correspond to 0.1, 0.1, 0.3, 0.2, 0.1 weight, so that corresponds to the percent of the bandwidth of the output interface during congestion (fig.2.7).

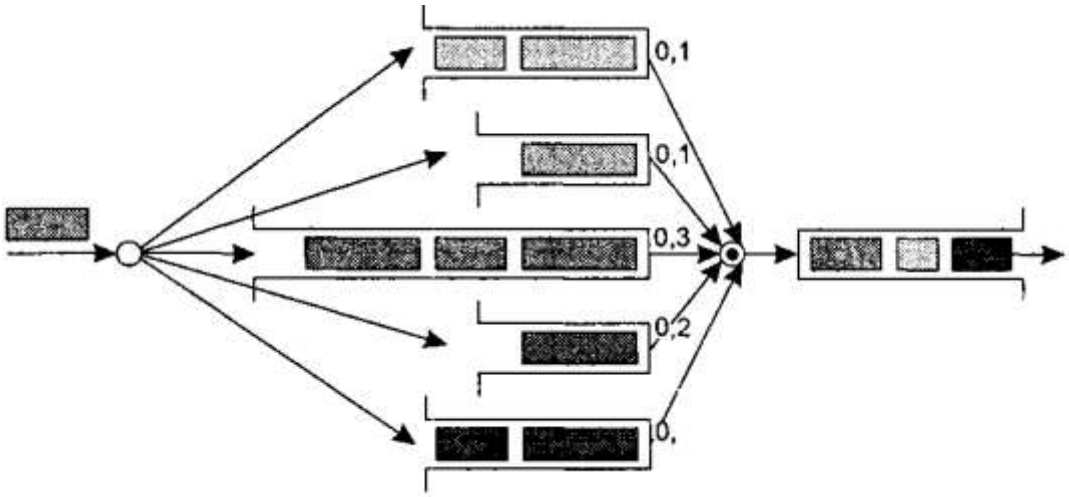

Fig.2.7 Weighted Queuing

Queues are processed sequentially and cyclically, meaning each time frame from the queue specific number of bytes is taken, which corresponds to the weight of the queue. As a result, each traffic class gets a guaranteed minimum throughput, which in many cases is a more desirable result than suppressing low-priority classes with high-priority ones.

The exact QoS parameter values for a weighted service algorithm are difficult to predict. They essentially depend on the dynamically changing parameters of the network device load - the intensity of packets of all classes and the variations in the time intervals between the arrival of packets. In general, weighted service leads to

greater delays and delays than priority service for the highest priority class, even if the allocated bandwidth is significantly exceeded over the input flow rate of this class. But for lower priority classes, weighted fair service is often more acceptable in terms of creating a favorable service environment for all traffic classes.

2.4.4 Weighted Fair Queuing

Weighted Fair Queuing (WFQ) is a combination of weighted and priority mechanisms (fig.2.8). Network equipment manufacturers offer numerous proprietary WFQ implementations that differ in the way they assign weights and support different modes of operation, so in each case, you must carefully study all the details of the supported WFQ.

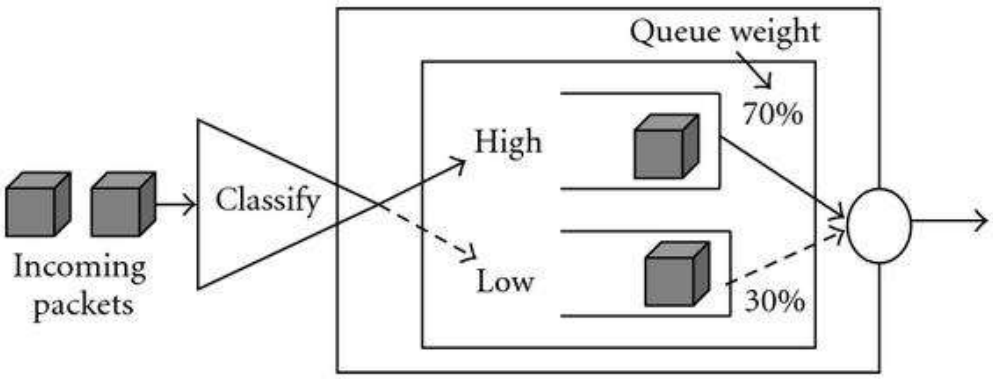

Fig.2.8 Weighted Fair Queuing

The most common implementation provides special queues, that are processed according to the priority, until all the requests from them will not be handled. This queues are intended for system messages, network management messages, and possibly the most critical and demanding application packages. In any case, it is assumed that its traffic is of low intensity, so a significant part of the bandwidth of the outgoing interface remains for other traffic classes. Its aim is to regulate usage of one transmission channel with the help of several concurrent threads. Thread is a stream of data. Fair Queuing schedulers have separate FIFO queues for each data flow. So that if a channel with speed R utilizes N threads, so the processing speed for each of them will be define by formula (2.4):

$$average\ speed = \frac{R}{N} \tag{2.4}$$

Fair Weighted scheduler allows to regulate the share of bandwidth for each individual thread. If there are N threads with different weights, so the processing speed can be calculated by formula (2.5):

$$avetage\ latency\ (ms) = \frac{R\ w_i}{(w_1 + w_2 + \cdots w_N)} \quad (2.5)$$

Every incoming package p_i^k gets its own virtual start time S_i^k and the end of processing F_i^k, where *k* is a package number, *i* is a thread number. The start and end time is calculated by the formulas (2.6, 2.7):

$$S(k,i) = max(F(k-1,i), V(a(k,i)) \quad (2.6)$$

$$F(k,i) = S(k,i) + \frac{L(k,i)}{r(i)}, F(0,i) = 0 \quad (2.7)$$

where *a(k, i) and L(k, i)* are the arrival time and packet length correspondently.

V(t) is a virtual function of time, that is defined by the formula (2.8):

$$\frac{dV(t)}{dt} = \frac{1}{\sum r_i} \quad (2.8)$$

where *j* represents all active sessions, *r* is a speed of *j-th* channel.

Let's have a look at the example (fig.2.9) with three queues: two of them have priority set to 1 and the third one has a priority 2. From the very beginning 1 packet in firsts queue, 2 packets in the second and 5 packets in the third one. It is assumed that all the packages are of the same size.

The device scans the rest of the queues in accordance with the weighted service algorithm. The administrator can set the weight for each traffic class in the same way as in the case of weighted service. The default operation option provides for all other traffic classes equal shares of the output interface.

Hardware manufacturers add some useful modes to the WFQ engine. For example, Cisco routers provide several flavors of WFQ:

- Flow-based WFQ mode (FWFQ);
- Class-based WFQ (CWFQ) mode.

$V(t)$	$dV(t)$	N_1	S_1	F_1	N_2	S_2	F_2	N_3	S_3	F_3
0	1/4	1	0	1	2	0	1	**5**	0	1/2
1/4	1/4	1	0	1	2	0	1	**4**	0	1
1/2	1/4	1	0	1	**2**	0	1	3	0	1.5
3/4	1/4	**1**	0	1	1	0	1	3	0	1.5
1	1/3	0	—	—	1	1	2	**3**	1	1.5
1 1/3	1/3	0	—	—	1	1	2	**2**	1	2
1 2/3	1/3	0	—	—	**1**	1	2	1	1	2.5
1 2/3	1/3	0	—	—	0	—	—	**1**	1	2.5

Fig.2.9 Representation of three queues characteristics

For FWFQ, the number of queues in the router is created depending on the number of flows in the traffic. In this case, a stream is formed by packets with specific values of the source and destination IP addresses and / or the source and destination TCP / UDP ports, as well as the same values of the Type of Service (ToS) field. In other words, a stream is a sequence of packets from one application with certain quality of service parameters specified in the ToS field.

Each stream has its own output queue, for which WFQ allocates equal partitions of the port bandwidth during periods of high load. Therefore, sometimes the FWFQ algorithm is called Fair Queuing (FQ).

2.5 Trust Traffic Classification of Game Traffic

techniques are applied on practice nowadays. The main problem challenge for ISPs is how exactly to prioritize incoming packages at any given time, or how to determine game traffic and give it a preferential treatment. And how does ISPs guarantee that only authorized traffic will get a special treatment.

Only server and a game client are aware of what IP packets are a game traffic. For the preferential service ISPs router need to have a 5-tuple rules set up in their configuration. But the question still remains, how ISPs discovers proper 5-tuple values,

by asking a client or it just gives an opportunity to a game client to mark Differentiated Services Code Point as well-known?

Generally, ISPs don't trust hosts that it does not control to specify the DSCP values. In case if it will be stated that DSCP value will be always set as '1', so every application will mark all their out coming packets with this value, and the problem with game traffic classification remains present, because all IP packets will get to the game traffic queue.

As a result, the assignment process is usually done by the IPS routers at the edge of the provider's network. After the 5-tuple classification by a router is finished, DSCP values are assigned. But the question is still open, which exactly 5-tuples define a game traffic?

The ISP could know the address and port number of the game server in case if its own, so this information may be enough to conduct a classification at the network's boundaries and give to the game packets trusted DSCP values on their way via a network. Without taking into account the address and port number of client all the packets will be served as a game traffic. As a possible approach is a signaling protocol used by the game client can be used inform the ISP about game traffic, when a new game traffic flow begins.

IPS can use another approach, analyze statistical properties to detect game traffic instead of looking at 5-tuple values or well-known port numbers. As a flow was recognized as incoming game traffic, its 5-tuple can be transferred along the preferential paths right to the router.

Implementation of 5-tuple classification technique makes it hard for fraudsters to create a game-like traffic and get benefit of preferential treatment on the network, the source and destination IP address should be the same, and the game-like packets should be actually going from the game server. It will not be useful for the non-game software products at all.

Hundreds and thousands of game-traffic ports and as many IP addresses that represent a game server exist. Game and ISPs developers run into issues to properly, securely and safely define a game traffic among other data stream and provide it needed preferential service.

2.6 Network Conditions Measurement

Network diagnostics as measuring the characteristics of the network during its operation (without stopping the operators of work). Network diagnostics is, in particular, measuring the number of data transmission errors, the utilization rate (utilization) of network resources or the response time of the application software, which the network administrator must perform daily. Gathering information about the network conditions is a crucial process and there are exists several ways to determine the latency, RTT, packet loss rate and others.

2.6.1 Ping and Traceroute

Ping is a tool for checking connection integrity in TCP / IP-based networks, traceroute or tracert is a program for determining the routes of data in TCP / IP networks. To check the ping, use the ping command of the same name, which must be entered on the command line. You can run the command line instruction to ping a specific server: ping <IP or domain> 9 (fig.2.10).

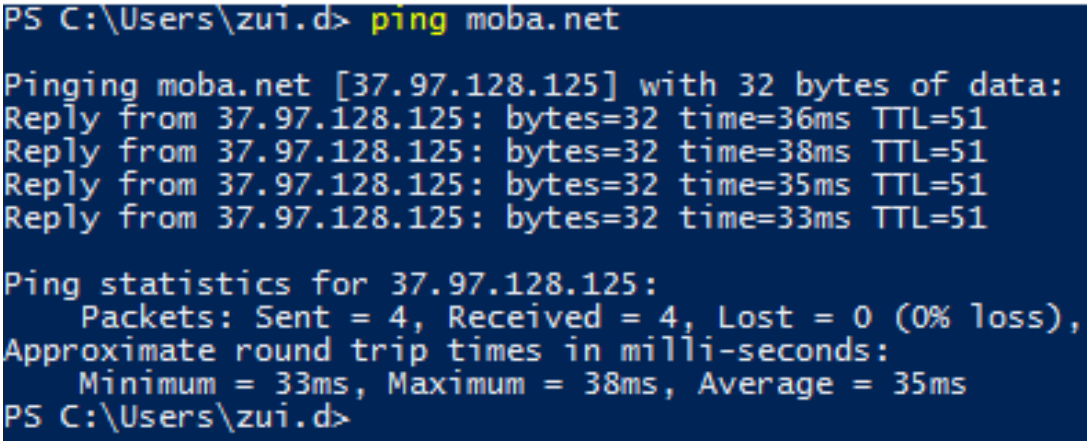

Fig.2.10 Result of ping command usage

As a result, 4 packages of 32 bytes of data were transmitted and received and the average exchange time is 35ms. The number of packages and its size could be changed by adding parameters to the command call.

Route tracing can show the rate at which packets travel between routers that connect the requesting PC and the destination server (fig.2.11). Command tracert <IP or domain> should be used to obtain results. It should be noted that by default, the trace also performs a DNS request to resolve the IP address to the domain name for each

router passing through. This option can be disabled, thus reducing the time it takes to get trace results.

```
PS C:\Users\zui.d> TRACERT.EXE moba.net

Tracing route to moba.net [37.97.128.125]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  192.168.0.1
  2    <1 ms    <1 ms    <1 ms  10.128.208.1
  3     1 ms    <1 ms    <1 ms  10.128.254.249
  4     1 ms     1 ms    <1 ms  kyiv1-ae0-1107.ett.ua [78.154.171.157]
  5    26 ms    26 ms    26 ms  fft1-fft0.ett.ua [80.93.127.194]
  6    35 ms    35 ms    35 ms  m6.e1.ams0.transip.net [80.249.208.244]
  7    33 ms    33 ms    33 ms  e1-a0.r1.ams0.transip.net [157.97.168.8]
  8    44 ms    47 ms   113 ms  r1.f2.ams4.transip.net [77.72.151.123]
  9    79 ms    58 ms    64 ms  f2.l2.ams4.transip.net [77.72.151.63]
 10    33 ms    33 ms    33 ms  37-97-128-125.colo.transip.net [37.97.128.125]

Trace complete.
PS C:\Users\zui.d>
```

Fig.2.11. Result of the tracert command

A full-fledged channel testing cannot be done by ping or traceroute, it will never tell you what the bandwidth is, since with the simultaneous use of the network and testing, the software utilities do not know the amount of user data that is in the channel at the moment, also a number of inaccuracies are possible due to the presence of packet headers, depending from the frame size, the headers remain the standard length, and the body with data increases or decreases, the software utilities determine the channel bandwidth without taking into account the size of the headers, which at different packet sizes introduces a certain confusion into such testing.

Since the quality of a network channels is a combination of many factors, therefore, correct testing should cover all these combinations as much as possible. There are many aspects to consider when testing and it would be useful to have advanced features such as BER Test, packet jitter, MPLS support, QoS, load testing of application layer protocols (http, ftp, etc ...).

Today there are two main techniques for testing throughput: the old one, RFC-2544, and the slightly younger, Y.1564. The ITU-T Y.1564 methodology is more relevant today, it has descriptions for testing modern, high-speed communication channels with modern concepts of SLA (Service Layer Agreement).

2.6.2 RFC-2544 and How It Works

Initially, the methodology was developed directly for testing network devices, for example, when developing switches, but the set of functions was adapted to measure the quality of links. The technique was approved in 1999 by ISOC.

The RFC-2544 methodology recommends measuring different frame sizes: for Ethernet traffic, frames are 64, 128, 256, 512, 1024, 1280, 1518 octets, for each frame size, a separate serial test is required. If necessary, you can test for the Jumbo frame (frames of 4096 or 9000 octets). Different frame sizes are needed to simulate different types of traffic. The technique offers a set of 6 tests, they will be described in more detail.

Determining the throughput of the device means that a small volume of packets specially formed by the tester is sent at a certain speed to the input port of the device, on the output port the number is counted, if more is transmitted than received, the speed decreases and the test starts again.

After determining the throughput, for each frame size, at the corresponding maximum rate, a stream of packets is sent to a specific address. The stream must be not shorter than 120 seconds long. After 60 second passed, in one packet a specific mark is inserted, the format is determined by hardware manufacturer. he transmitting side records the time by which the tagged packet was completely sent. On the receiving side, a mark is determined and the time of complete reception of a packet with a mark is recorded. Latency is the difference between sending time and receiving time. This test, according to the methodology, must be repeated at least 20 times. The average delay is calculated from the results of 20 measurements. The test should be carried out by sending the entire test stream to one address and sending each frame to a new address.

Determination of the frame loss rate is carried out in the following step. A certain number of frames are sent to the input port of the device at a certain rate and the number of packets received from the output port of the device is counted. The frame loss rate is calculated as ration between number of frames transmitted minus number of frames received multiplied by hundred to the number of frames transmitted. Quick diagnose with the free MTR utility (fig.2.12)

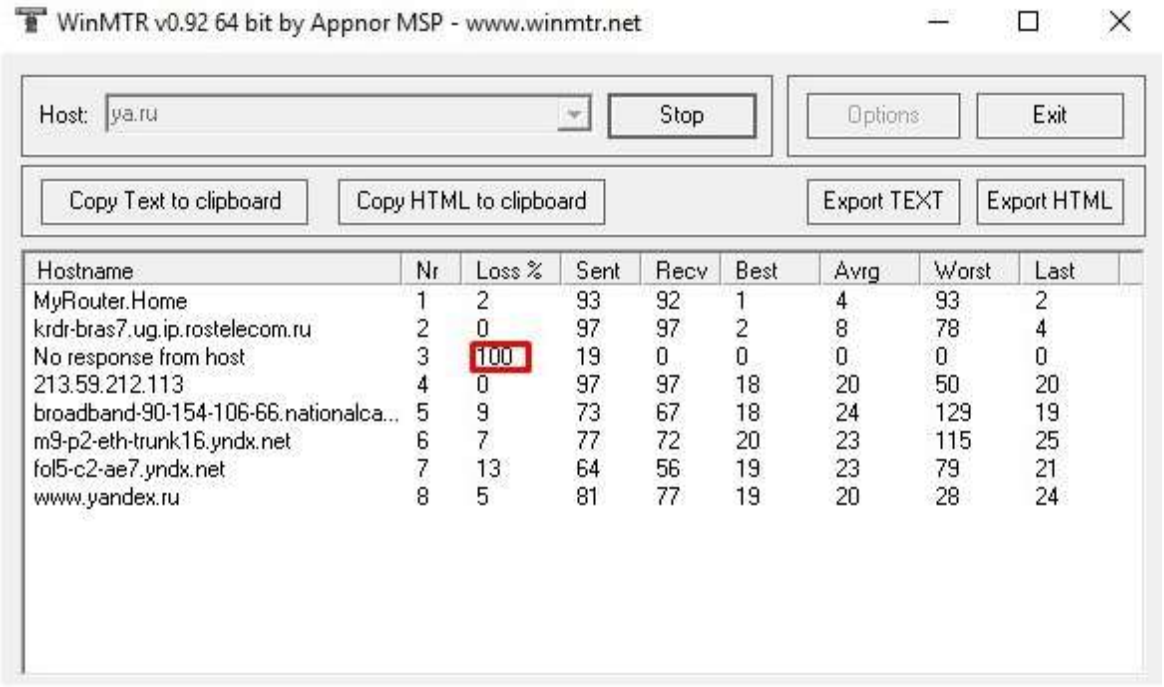

Fig.2.12 Result of the MTR utility usage to determine package loss rate

The first sending occurs at the maximum possible speed, then the sending speed is reduced with a maximum step of 10%, according to the method, decreasing the % step will give the most accurate results. The decrease in speed must be continued until the last two sends are error-free, namely, we find out the maximum data transfer rate at which the frame loss rate becomes 0.

The back-to-back frame test implies sending a certain number of frames with a minimum delay between frames to the input port of the device under test and counting frames from the output port of the device. If the number of frames sent and received is equal, then the volume of frames sent increases and the test is repeated, if the received packets are less than the sent ones, the volume of sent frames decreases and the test is repeated. As a result, the maximum number of packets sent and received without loss for each packet size is obtained, this will be the value of the back-to-back test. According to the methodology, the duration of sending frames to the device port should not be less than two seconds, and the minimum number should not be less than 50 times. The final number is the average of 50 performed tests.

System recovery tests send a frame stream to device during 60 seconds at the 110% rate relative to the measured throughput test. If the throughput test showed ideal results, then the maximum speed of this connection is selected. At the moment of

overload, the flow rate is halved and the difference between the time the flow rate decreases and the time when the last frame was lost is recorded.

Recovery time of the device under test after reset is applied to the device under testing. A continuous stream of frames is sent to the input of the device at a rate determined as a result of the throughput test with a minimum frame size. The device is reset. The reset recovery time is the difference between the time the last packet was received before the drop and the time the first packet was received after the drop. Both hardware and software reset types are tested.

2.6.3 Y.1564 technique updates

Jitter is added to the recommendations already outlined in RFC 2544, namely the ability to calculate the time difference when receiving a number of sequential data packets belonging to the same stream, in an ideal world it should not exist, but in problem networks there can be violated, which may affect the speed of data processing. RFC2544 allows you to make checks exclusively at the maximum channel speed at which there will be no packet loss, which is usually higher than the Committed Information Rate (CIR). Y.1564 was created specifically for SLA, assessing the speed and quality of the provided channel according to key performance indicators [13].

Y.1564 allows you to check the guaranteed bandwidth, the maximum allowable, as well as overload the bandwidth, for example, to review the shaper settings.

There are a few more differences between the methods (fig.2.13), RFC2544 does not verify the correctness of the service configuration (compliance with the specified KPIs, and the rate limiting above the Excess Information Rate (EIR), in order to avoid network congestion). The original version of RFC2544 does not measure jitter. According to RFC2544, each test is triggered by a separate thread, which does not allow measuring the quality of the services provided in the aggregate and increases the testing time, another disadvantage of RFC2544 is that there is no possibility of profiling to test different types of traffic in one channel, for example, if the network uses QoS, in Y.1564 the shortcomings are taken into account and the functionality is slightly expanded.

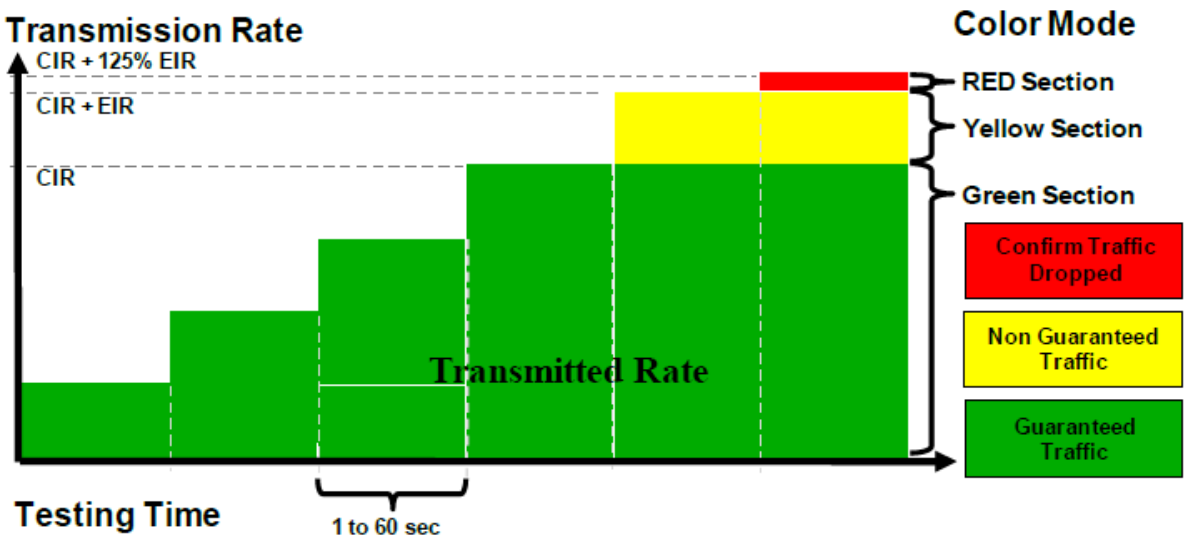

Fig.2.13 Additions to network testing brought with Y.1564

To date, network testing strives for complete systematization and constant control of channels, earlier versions of the RFC2544 methodology were created for testing channels / equipment in Out Of Service mode, and were used mainly for testing equipment, but today all manufacturers of test devices are switching to newer ones. testing standards allowing continuous monitoring of the network in In Service mode. Such testing allows you to check the bandwidth speed without disconnecting clients, which is important for telecom operators.

2.7 Server Information Gathering Approach

Another approach is to use information gathered by a modified game server. Figure 2.14 shows the experiment where RTT samples from an active game server were used to plot the average jitter per map played against the average latency per map played.

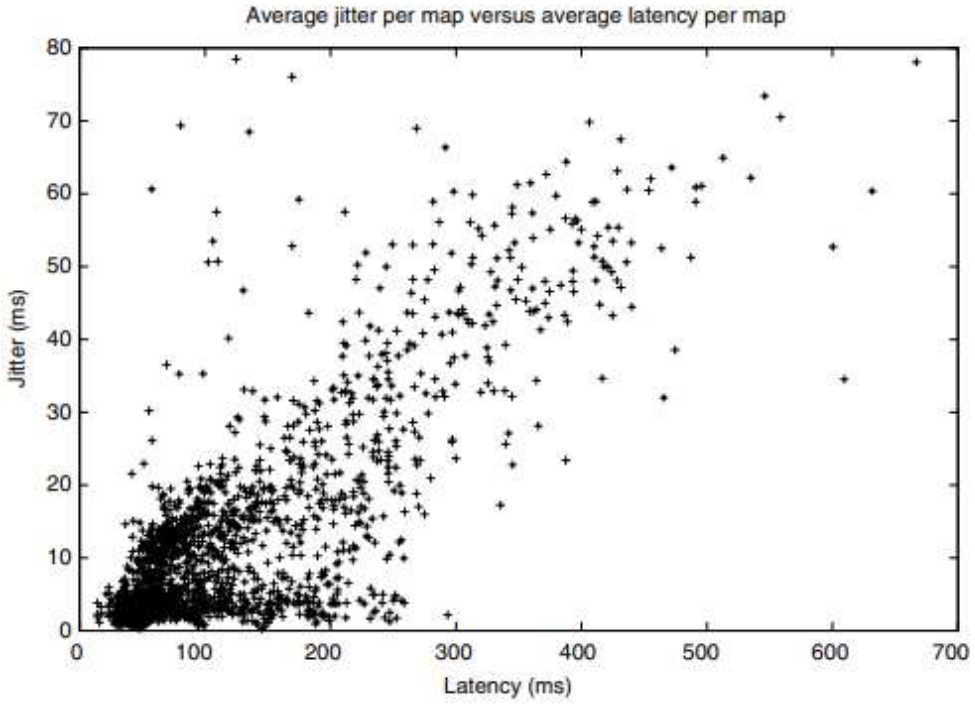

Fig.2.14 Jitter and latency measures from an active game server

After the experiment was conducted it is seen that jitter typically never exceeded 20% of the average latency. Another observation is that there are two broad clusters – one where jitter is low regardless of latency out to 300ms latency and another where jitter is roughly proportional to latency. The former is devoted to paths where most latency is propagation delay. The latter is attributed to paths where delay accumulates from many router and link hops – also places more likely to contribute to jitter.

A simple script written with programming language php can be used to obtain some information from the server (fig.2.15). The obtained data after the script execution return the information about the map, players, maximum players, status and IP address of the server and the most importantly latency. This data could help to monitor the server status and other characteristic and alarm about some unpredictable changes in the server work.

```php
<?php

function serverInfo($server) {
    list($ip,$port) = explode(":", $server);
    $timeStart = getmicrotime();
    $fp = @fsockopen('udp://'.$ip, $port);
    if($fp) {
        stream_set_timeout($fp, 1);
        fwrite($fp,"\xFF\xFF\xFF\xFFTSource Engine Query\0\r");
        $temp = fread($fp, 4);
        $status = socket_get_status($fp);
        if($status['unread_bytes']>0) {
            $temp = fread($fp, $status['unread_bytes']);
            $version = ord(getChar($temp));
            $array = array();
            $array['ping'] = (int)((getmicrotime() - $timeStart)*1000);
            $array['status'] = "on";

   if($version == 109) {
        $array['game'] = "cstrike";
        $array['ip'] = getString($temp);
        $temp = substr($temp, 1);
        $array['name'] = getString($temp);
        $temp = substr($temp, 1);
        $array['map'] = getString($temp);
        $temp = substr($temp, 1);
        getString($temp);
        $temp = substr($temp, 1);
        getString($temp);
        $temp = substr($temp, 1);
        $array['players'] = ord(getChar($temp));
        $array['max_players'] = ord(getChar($temp));
    } else
        $array['status'] = 'off';
    }
    return $array;
}
```

Fig.2.15 Screenshot of the code for getting information from the server

Conclusion on the Second Part

The realism of multiplayer game depends on how well network part allows all game clients to communicate in timely and predictable way. Considering the field of networks, essential features and properties need to be defined and summarized by the concept of metrics.

The analysis of the networking problems allows to define those metrics of communication network:

- Lag is a sort of downtime and delays that arise during the transmission of data packets, as well as in the process of processing the data received or prepared for dispatch, are determined.;
- Loss is a loss of data during the transfer process to the final destination.
- Jitter is a latency changes and the direction in which it goes over the time, a jitter in the network can be viewed in the short term, but the overall latency is rather constant.

Lots of ways of the network condition measurement tools allows to check the values of the listed metric and help to define the overall picture of the used network.

Mornitoring and checking network condiditons is a part of the network game developer. Implementation of the networked game acritecture, bulding prediction systems and solving peeker’s advanatage problem by means of interpolation technique is covered in Part 3.

PART 3

THE DEVELOPMENT OF MULTIPLAYER CLIENT SERVER GAME

3.1 Basic Client-Server Architecture of the Game Server

Multiplayer networked game development is a rather complicated process, but the presence of online component of the game will guarantee the success of the final product. Pc game development is known for the wide range of the supported client's devices, in terms of different specifications and network connectivity.

Broadband networks were created and brought a possible solution for the networked games, but it does not mean that latency and other network factors should be ignored by developers. Some clients may have significant latency and packet loss rate, because of the provided bad broadband solutions.

The game should be ready to face real-world problems. Let's discuss how exactly client-server architecture works in online games, some intelligent modeling techniques for the latency compensation.

In general, approaches to creating multiplayer games have not changed much over the past 20 years. Most part of the current games have one authoritarian sever, that runs all main game logic and there are several clients connected to it. Clients are considered as a tool for command input and sending gathered information to the server, the latter executed received commands, updated the game object state and sent to the client list of the objects for rendering (fig.3.1). The real systems are more complicated and have more components in it, but a simpler definition is useful in later discussion of the prediction and lag compensation techniques.

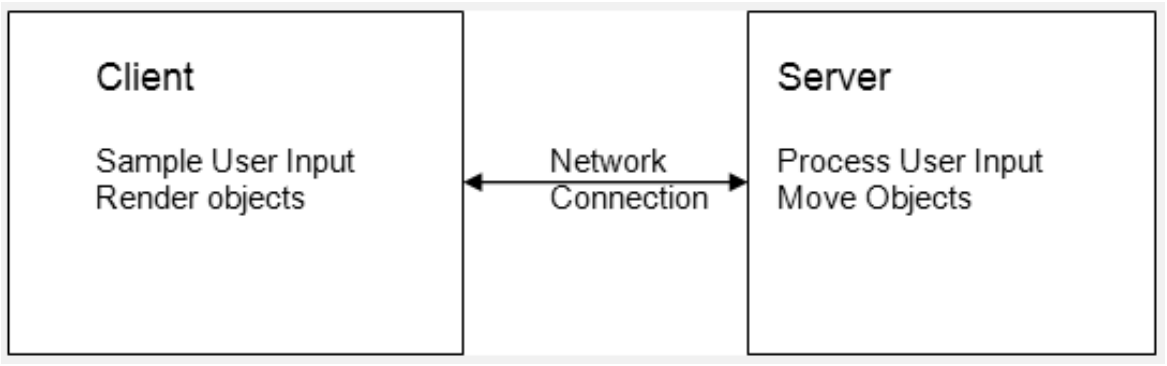

Fig.3.1 Typical client-server game architecture

The game cycle from the client's perspective looks like this:

- Save start time;
- Process user input;
- Update and send a packets, that contains move commands using the last frame simulation;
- Reading of the received from server packets;
- Using of packets to determine visible objects and its state;
- Scene rendering;
- Save finish time;
- Calculating of the next frame calculation (finish time extract start time).

After a client finishes a cycle, time cycle needed for one tick execution is used for the calculation of the time needed for the world modeling on the next step. If the client has a constant frames frequency, then cycle time will be a right measure. Otherwise, the time frames are incorrect and there is no solution of the current problem, it is not possible to define how much time it is needed for the next iteration of the game cycle.

The game cycle from the server's point of view looks like this:

- Save start time;
- Reading user's input from received packet;
- Execution of the client's input commands;
- Modeling of the server objects using the simulation time from the last completed cycle;
- Pack for each client visible objects, game-state and send it to the client;

- Save finish time;
- Calculating of frame modeling time (finish time extract start time).

This models non client related objects are processed only on the server side, while player's objects are controlled by movement based on the incoming packets.

An example of user commands representation, containing crucial data can be viewed on the figure 3.2.

```
struct USER_DATA {
    // Interpolation time on client
    uint8       lerpMsec;
    // Duration in ms of command
    byte        msec;
    // Command view angles.
    FVector     viewAngles;
    // Forward velocity.
    float       forwardMove;
    // Sideways velocity.
    float       sideMove;
    // Upward velocity.
    float       upMove;
    // Attack buttons
    uint16 buttons;
};
```

Fig.3.2 User command encapsulated in the C++ structure

The main components of the structure are 'mces', that represents the execution time of the command or cycle time; 'viewAngles' is stands for the direction vector of the player at the current frame; 'upMove', 'sideMove', 'forwardMove' are the impulses defined by the keyboard, mouse or controller input; 'buttons' includes the information about hold keyboard, a specific bit for the corresponding button.

Taking into account the presented structure the modeling core will be following. Client forms and sends user command to the server, after that server processes all the information and executes commands and sends back to the client updated game state data. At the end clients renders a scene with all the objects. The major problem is a simplicity of the current system, it could not provide a real time reaction on the user input, if the client experiences some huge network connectivity latencies. In this scenario a client performs a very simple tasks, it is reading/sending the messages from/to the server. If the client's latency is 700ms, so each client's action, for example,

a client pressed a button to perform a move right on one unit, will take 700ms for server to validate and perform update of the game world states on the client's side (fig.3.3).

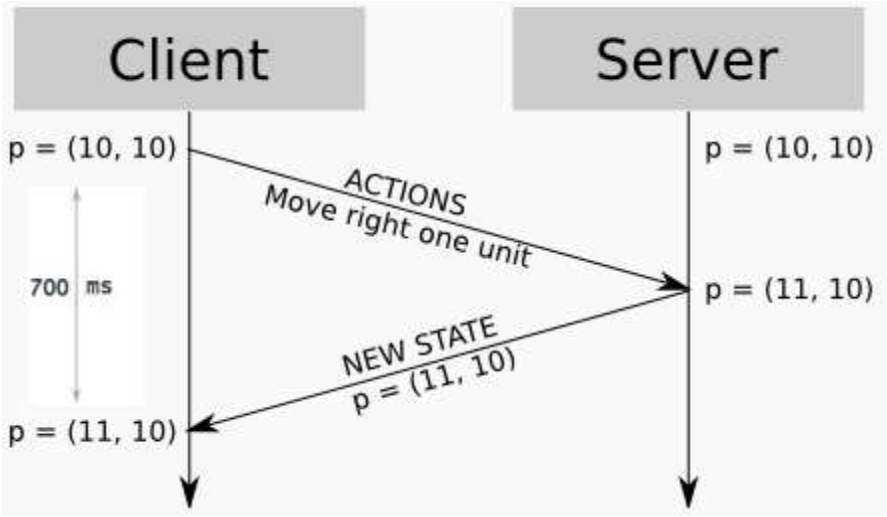

Fig.3.3 Delay between sending of the information to the server and client update

This latency could be acceptable for the Local Area Networks, but not for the Internet. The gameplay can be not responsive or become not playable at all.

3.2 Prediction system for the FPS game

Responsive control is a crucial component the networked FPS games, clients need to see the updated on the screen immediately the button or stick was pressed. First of all because the game state of the world in this type of games changes very quickly and it is necessary to instantly react to the situation. One of the possible solutions is a local execution of the commands and a client suggests that server will receive and immediately validate user's instructions. In this way a client predicts the game world state and the prediction usually will be right, since the game world is deterministic, the state is determined by the commands and a previous state. But sometimes the client's prediction can give non-realistic results, so the obtained data from the server will be used to correct the mistakes. Network latency the correction will not take place, until the time of the game cycle will pass. For example, it could lead to the significant shift in the player's position due to previously accumulated prediction errors.

To implement motion prediction system on the client's side, the following algorithm is used. The user input is still processed and formed from the player's side,

then the collected user data is sent to the server, but every instruction and its creation time is stored on the client. This time is then used by the prediction algorithm.

The last approval from the server is used as a starting point for the prediction process. It defines the last player command run on the game server and the last position and other properties/states of the player, since the last movement command was modeled. The last confirmed instruction will be in the past in case if there any connection lag exists. As an example, client runs at 60 frames per second (fps) and has 90ms latency, so client will store three user commands then before the last confirmed by the server. This three commands are modeled on the game client as a part of the prediction; performing of this commands gives a precise final state for the client, that is needed for the for the rendering of the current scene.

To reduce the discrepancies between server and client, the shared usage of the same movement code on both sides. The input data for the common procedures can be encapsulated in the command ‘from player state’, the result is a new players state, after issuing the user's command. Outgoing information about the final state represented in ‘to player state’, being the result of prediction and used for the rendering purposes. The part where the command is executed is just the part where all the player's state data is copied into a common data structure, custom commands are processed and the resulting data is copied back to the ‘to player state’ (fig.3.4).

```
"from player state" // <- state after last user command acknowledged by the server;
"command" // <- first command after last user command acknowledged by server
while (true)
{
    // run "command" on "from state" to generate "to state";
    if (this was the most up to date "command"
        break;

    "from player state" = "to player state";
    "command" = next "command";
}
```

Fig.3.4 Common algorithm for the player prediction system

However, not all systems run on the client, but only those that are responsible for the local player and do not require up-to-date data on other players (fig.3.5).

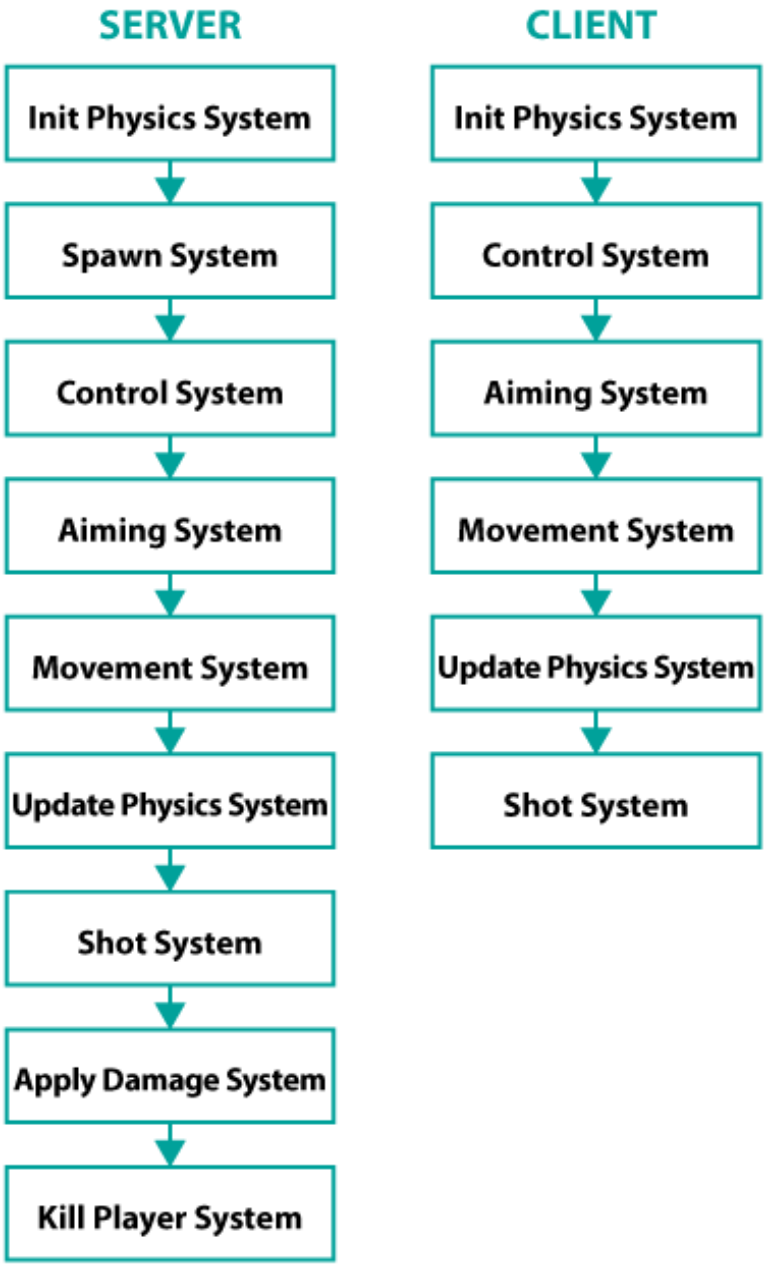

Fig.3.5 Example of systems running of the client and server sides

Implementation of shooting prediction is rather easy to do. Of course, the local player needs additional information about the client's state, including which weapons are stored, which one is active, and how much ammo each of these weapons has left. The state of the pressed ‘shoot’ button is added to the overall structure, which is both used by client and server. The shooting system is marked as a common code, working on the client’s and server side. All variables that contribute to determining the state of the weapon are also part of the main server state and are copied on the client side, so they can be used to predict the weapon’s state. Client-side firing prediction solves the issue with weapon switching, deploy and retract. The user will feel that the game is

fully responsive to. This goes a long way towards reducing the clanking that players could experience using the internet.

Predictions for things like damage dealing, using abilities, or healing teammates. There are two main problems with these mechanics:

- Client does not know about other player’s input and prediction of such things will always confront with the server data;
- The creation of new entities locally (shots, projectiles, unique abilities) generated by one player carries the problem of matching with entities created on the server.

Those kind of mechanics use other lag compensation techniques. For example, render the effect of the hit from the shot immediately, and update the enemy's life only after we receive confirmation of the hit from the server.

Replication of the properties can be implemented by means of mapping the entities on the client and server sides using integer identifier (fig.3.6). Each entity uses the id of the previous number plus one. This approach is very easy to implement, but it has one significant drawback: the order of creating new entities on the client and server can be different.

Property replication technique is rather easy network synchronization technique between client and server, but is it rather slow. Scanning through all replicated variables is a time consuming process, packing and sending will experience additional latency in this case. It is utilizing strategy of random access across the memory and caching slowing down the overall system.

The client and server synchronize the time by ticks. Because of time needed for transferring data over the network requires some time, the client is always ahead of the servers by half of the RTT plus the size of input buffer on the server. The figure 3.7 shows that client sends an input for the tick number 20 and at the same time server processes tick number 15, at the moment when client’s input will reach the server, tick number 20 will be processed. The overall process is the following: client sends input to the server, then is processed after HRTT plus input buffer size, after that the resulting state is sent by the serve, then client receives updated game state after RTT plus input buffer size adding game state interpolation buffer size time is passed.

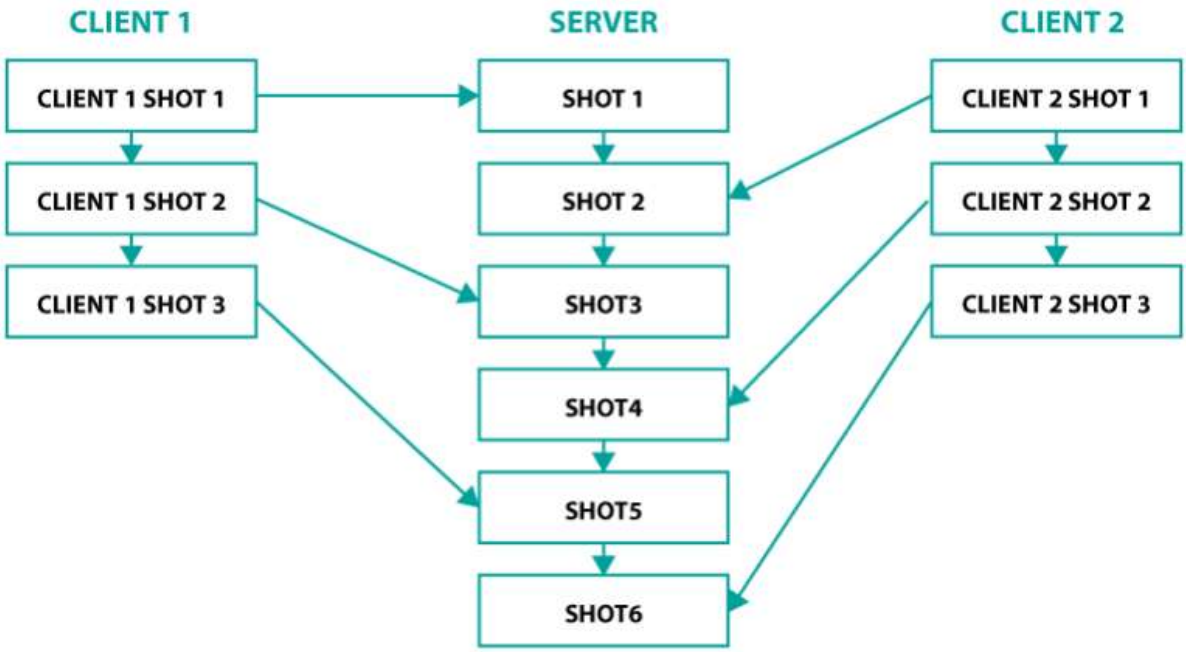

Fig.3.6 Shoot entities on the client and server side

After the client received valid game world state, the process of reconciliation needs to be conducted. Prediction client system does not take into account the state of other players on the server, so the resulting predicted game state could be different from the real one.

Reconciliation process consists of comparison of the predicted game state for the tick N with the obtained result from the server. Only the data for the local player is taken into account, the rest data is taken from the server. There are two possible outcomes of the comparison process. In first case if the result of the prediction corresponds to the server data, the client uses the predicted data for the local player and new data from the server to do simulation. If the predicted and server do not coincide, the client uses all the game state received from the server to recalculate new game world state for the player.

Complexity of the reconciliation technique is in need of re-simulation of all client predicted states that were not validated by the server up to the current tick, in case if desynchronization of the server and client takes place. It could take 5 and more server ticks depending of the client's latency [15].

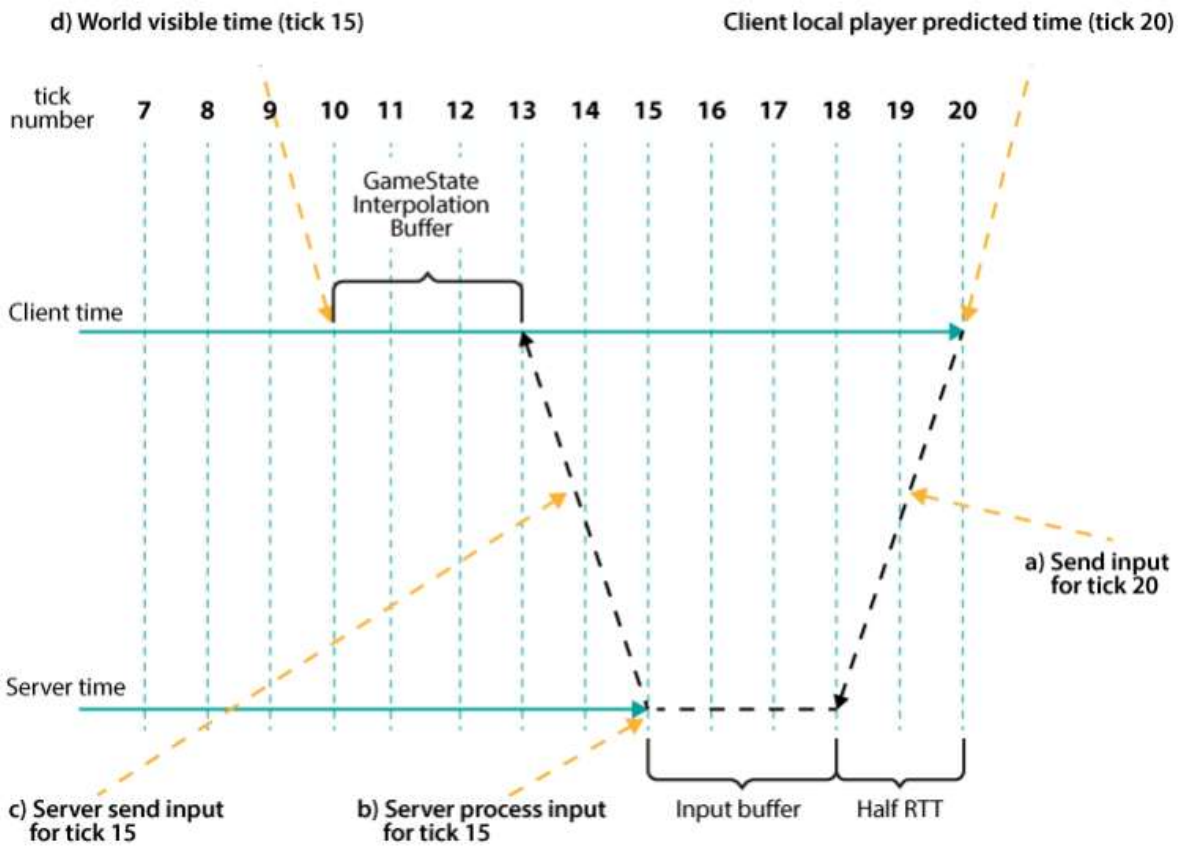

Fig.3.7 Overall server work scheme

3.3 Lag Compensation and Peeking Problem

As it was previously discussed the server processes user input, updated the game state and send it back to the client. The number of clients can be huge and the commands will be sent with high frequency. Calculating of the game state and notifying all the clients about updates will increase the load on the network and server CPU. The good way is to accumulate the commands without further execution and the world will be updated 20 times per second for example, but fast paced shooters require high frequency of update (128 times per second) for comfortable gaming experience (fig.3.8).

Let’s consider the situation with a low update frequency of the server with 10 times per second. The player does not feel any issues with world simulation, prediction system works, but game world updated about other players are received rarely and as a result the game seems to be unstable. Even though the server sends the updates 60 times per second it does not guarantee that client will get them 60 times second because of the network problems.

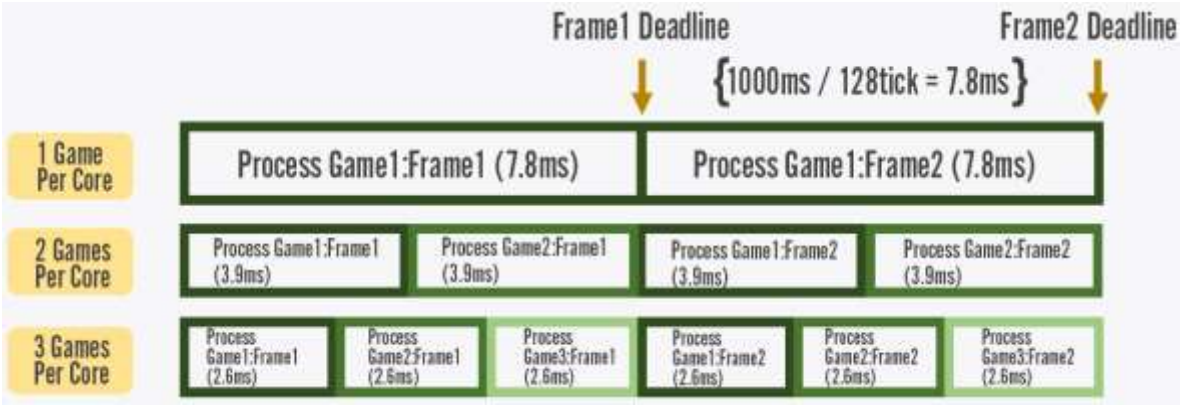

Fig.3.8 128 Tick rate server working process

The extrapolation and interpolation methods can be used to determine the position of other players on the client side. For example, an object's position in modeled ahead of the time, taking into account its previous position, velocity, direction and acceleration, having a 100ms delay a client having this last updated data can predict the resulting position of the object and just render a new position on the fly. It will work for the prediction for the objects with high inertia such as cars, ships etc. Extrapolation will give unsatisfied results in cases when the direction, speed and acceleration could be changed in a matter of time. In the FPS games player can turn around and stop, or apply some forces that produce acceleration in the arbitrary direction and the extrapolation results will be incorrect. Client cannot rely on the updates from the server, thus other players will simply teleport each time frame with a specific latency. The main goal is to simulate all the action between each 100ms delays.

Interpolation could be described as a moving objects in the past with respect to last valid position. In case any of the packets was lost on the way, extrapolation of the player's position can be applied with some potential errors, or the object's position can be updated after the 100ms delay.

The general interpolation algorithm includes the following steps:

- Every server update has start time mark;
- The end time is calculated with respect to the start time using interpolation delta value.
- The timestamp between the start and end time defines how much time has passed since last update.
- The resulting ration is used for the variables interpolation.

The algorithm requires a fixed delta between server updates to work properly, but it can face a problem with simulation of some objects behavior. The bouncing ball problem can be faced; The ball can be in the air or on the ground, but most of the time it is somewhere in the middle; the interpolation of the last position can produce undesirable result of hanging the ball in the air and the ball does not touch the ground at all. This issue can be fixed by increasing the update frequency of the world, but some artifacts can be present.

Storing a history of object positions is another approach of interpolation. Each incoming server updated is saved with a timestamp. To interpolate, we compute the end time as above, then we search backward through the position history looking for a couple of updates that span the end time. We then use them to interpolate and calculate the end position for that block. This allows us to smoothly follow a curve that includes all of our anchor points. If we are running at a higher frame rate than the incoming refresh rate, we are almost sure of smoothly moving through the sample points, thereby minimizing the anti-aliasing problem.

The interpolation can be viewed as an additional 100ms buffering on the client's side, thus simulating all the player's positions in the past. For example, the position values were obtained at t = 700, the information on the time point t = 600 was obtained earlier, so the within the time period from t = 700 and t = 800 the actual enemy movement is simulated, with 100ms delay (fig.3.9).

Good understanding of interpolation and extrapolation is crucial for the lag compensation techniques development, because it also can become a reason of the lag. Lag compensation is a normalizing of the game world state for each client as player's commands are performed on the server. It can be viewed as time travel in the past on the server with the purpose to take look at the game world state at the moment when the action was performed.

The following algorithm is applied:

- Calculate the precise latency, look thought the player's history for world updates that were sent and received by the player just before the command was sent;

- According to the update move each player back in time to the moment when the client's command was created.

This approach is crucial for the peeler's advantage problem solving, that is a bottleneck of lots networked games. This means the player coming out of the corner spots the opponent, but the opponents have no clue about the upcoming player. It is an example of the network desynchronization, even couple of frames difference makes a big deal (fig.3.10).

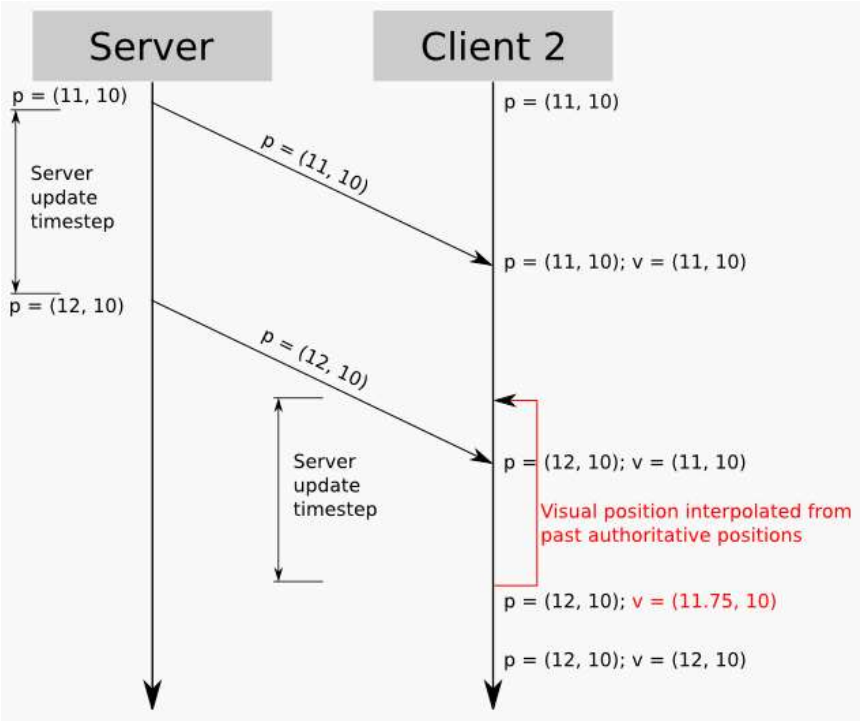

Fig.3.9 Interpolation algorithm with 100ms delta

Fig.3.10 Peeker's advantage example

Let’s brake down the whole situation and figure out what is happening when player leaves the corner and shots in the opponent as soon he spots him (fig.3.11). The peeking player is able to spot enemy as soon the movement commands will be processed and executed [16].

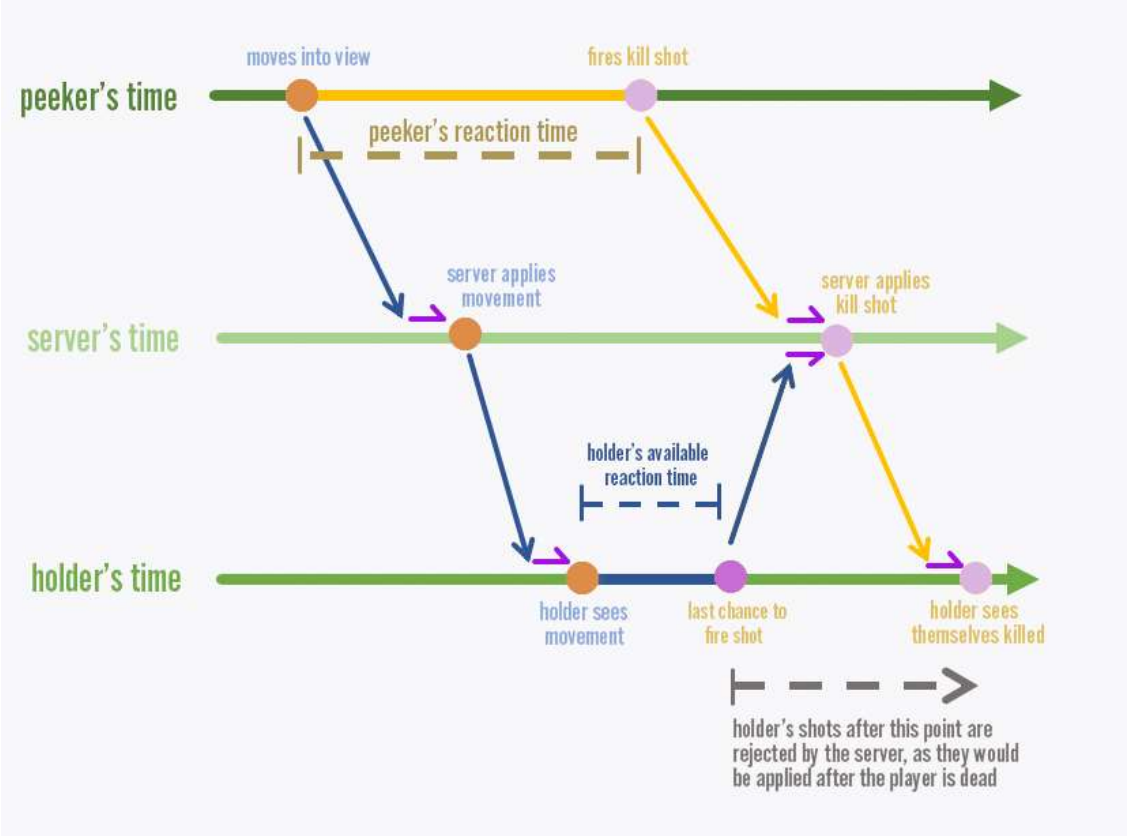

Fig.3.10 Schematic representation of the peeker’s advantage situation

The peeking player can see the holding player as soon as their computer processes their input and predictively moves them around the corner. The holder won’t see the peeker’s movement until the following events happen:

- Peeker’s client sends input movement command to the server and the server processes it;
- Movement data is sent to the holder’s game client;
- After the data was received, the world state is re-rendered and the peeker’s character is in a new position.

After all this steps are done, the holder’s machine will receive information about the made shot, it will be obtained until the same sequence of steps is not finished. In order to become a winner in this situation a holder player should send a shot event to the server, taking into account the time needed for the data to get to the server and be

processed, before the shot from another player will be validated by the server. The small period of time is created when a victim is not killed yet on the client side, but is already marked as dead on the server side, and it could make a some shot that will not be rejected.

In a such simulated environment, the movement information can be applied after the server and client receive it. Regarding to the internet's instability, the packets are transferred with a specific delay or sometimes do not arrive at all. If the simulation was running against the newest available data, it would frequently find itself waiting on late or missing data and be forced to predict what happened instead. Desynchronization of the client and server leads to the visual artifacts where characters pop or slide around in order to resync the simulation.

Those kinds of artifacts are not the part of a FPS game design goal, smooth movement needs to be achieved and make players able to track a moving target without problems. The purple arrows on the figure 3.10 represent the common solution to neglect an unstable incoming data stream by means of buffering the incoming data. The size of buffer can change depending on the incoming data to guarantee a smooth data flow for any downstream systems.

Other types of applications, that aren't sensitive to the delays can increase the buffer size to overcome relatively large network delays or network problems. For real-time games, however, buffering is a delicate balancing act. While buffering incoming movement allows us to smooth out network issues, it delays when the player will see incoming movement and effectively serves as extra network latency.

Buffering involves the full time from a move being received from the network to that move being processed and output (rendered to screen for clients or broadcast out to clients for the server). The duration of buffering is directly dependent on simulation tick rate (and render framerate on clients), because:

- Frames of movement data are buffered accurate to the game ticks;
- Move instruction may arrive in the middle of the frame and needs to wait up to a full tick to be queued or processed;
- Processed moves may take an additional frame to render on the client.

High tick rate server and client optimization of client performance is key to buffering reduction and allows to update game world state more frequently.

To make fighting as fair as possible, minimization of the reaction time advantage that the observer has over the holder is required. In fact, one way of expressing the peeker's advantage is the relationship between the peeker's reaction time and the holder's maximum reaction time before being killed.

Player can win the engagement if the its shot was processed and validated by the server earlier than the other's. An inequality on the figure 3.11 represents the conditions for the holder player's shot command be processed before being killed.

$$ReactionTime_{holder} < ReactionTime_{peeker} - \left(NetworkLatency_{server=>holder=>server} + NetworkBuffering_{holder} + NetworkBuffering_{server}\right)$$

Fig.3.11 Equation for the holder player win the engagement

The reaction time of the holder's player can be defined to survive by taking peeker's reaction time and subtracting the holder's RTT to the server and the network buffering on the both sides. Now, let's take some values to get the real delay numbers to understand how big actually it is. To keep things simple, we define buffering as time between a move being received from the network and being rendered on the client's screen.

Let's assume that two server's buffering frames include:

- 0.5 frames stand for the average time of the incoming data delay until the point in frame when it is added to the queue;
- 0.5 frames mean actual network buffering, representing the time for the data being in the queue before actually being applied;
- 1 frame is a full frame, meaning that movement was applied and transferred to the clients.

Client's buffering takes three buffering frames; it works in the same manner as on the server but:

- One full frame is used for the network buffering, but not the half of the frame;
- Client side renders the result on the screen, thus delaying the output because of the swap chain delay and GPU, it can take from 0 up to 2 frames, taking 0.5 as a frame delay sounds rather reasonable.

Figure 3.12 illustrates the calculation needed to be performed for the peeker reaction time definition.

$$ReactionTime_{holder} < ReactionTime_{peeker} - (60ms + [3 * 16.67ms] + [2 * 15.6ms])$$

$$ReactionTime_{holder} < ReactionTime_{peeker} - 141.25ms$$

Fig.3.12 Peeker reaction time calculation

Resulting value of circa 141ms states that peeker player has additional time to react, taking into account average reaction time of 300ms it is a huge advantage. For the minimization of the peeker's advantage (fig.3.13):

- Separate internet backbone can be created, to minimize processing time and routing over the network;
- Increasing then number of servers to deliver a 35ms delay to each player;
- Perform a server optimization;
- Make a game client to work at minimum 60 frames per second on the machines with different configurations;
- Run both client and server with minimal buffering, targeting one buffered frame of movement data for clients and an average of half a frame of movement data on servers.

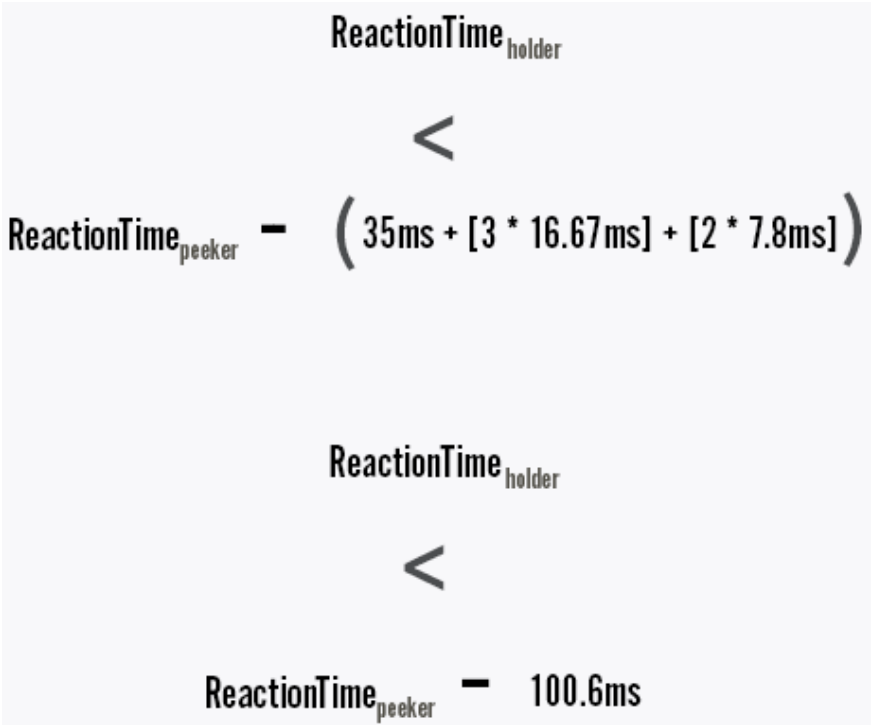

Fig.3.13 Result after minimization techniques are applied

As a result, the of the server's optimization and the required reaction time is reduced by 28% (circa 40ms) comparing to the initial time. Minimizing the raw reaction time advantage of the peeking player is crucial for reaching of consistent holder's advantage. The differences in client's prediction system results and the server's simulation are inevitable, couple of lost packages or some performed actions can the world state the has drifted away from the server's authoritative simulation.

Differences in the server and client framerate does not need to cause some simulation divergence. For example, simulating physics with two times in 10ms (each 5ms) and updating it one time in 10ms will bring different result, regarding to the way to the way forces are numerically integrated during each update. Client and server running with the different frequency, 60Hz physics simulation will be slower than a 128Hz server requiring instant correction of the desynchronized data. The separation of the simulation updated from the game tick will achieve desired result. Movement, physics, shooting and other simulation systems will be updated with a fixed time step, 128 times per second.

Client working with 60 FPS will simulate multiple ticks per frame, but a player with higher frame rate is able to reconcile one simulation update via several frames (fig.3.14). We make a prediction to known the exact position of the player on the frame

edges. After that the game world state is lineally interpolated to render the objects at the exact positions.

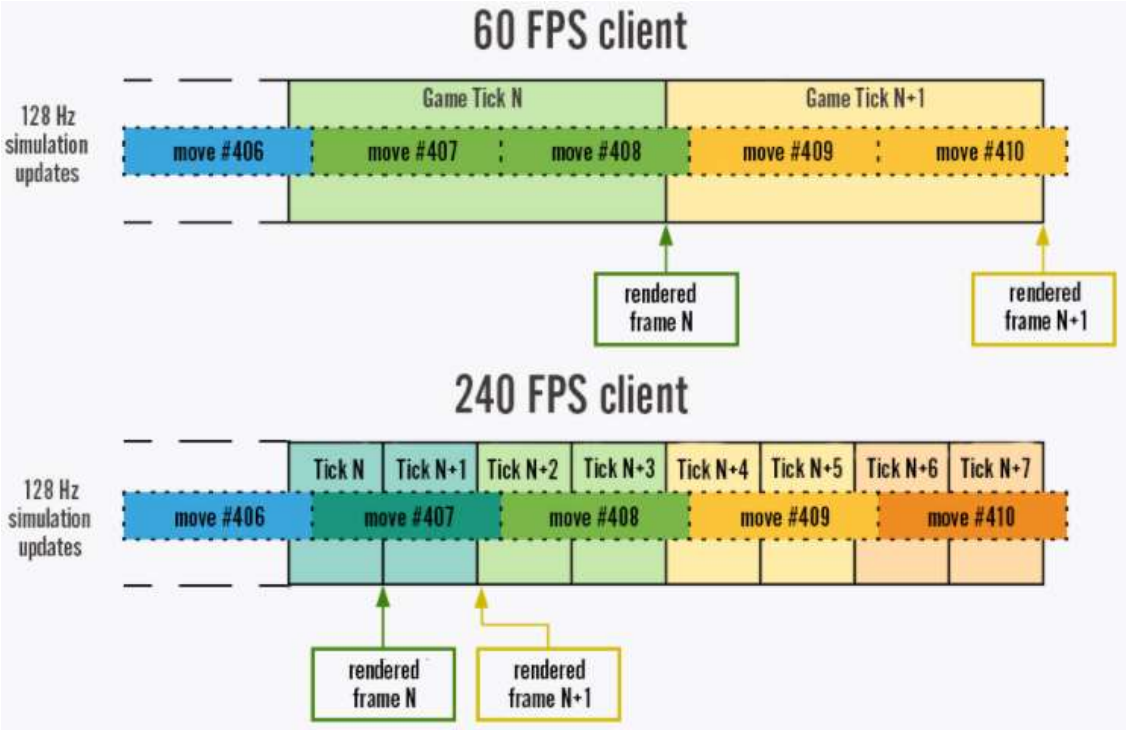

Fig.3.14 The difference in 60 FPS and 240 FPS game client

Dividing the movement into chunks (fig.3.15) of the same size allows easily convey it and reason on different machines. Each client sends input and movement results to the server, which runs its own simulation of the world and sends a fixed version of the game state data to the player in case of disagreement.

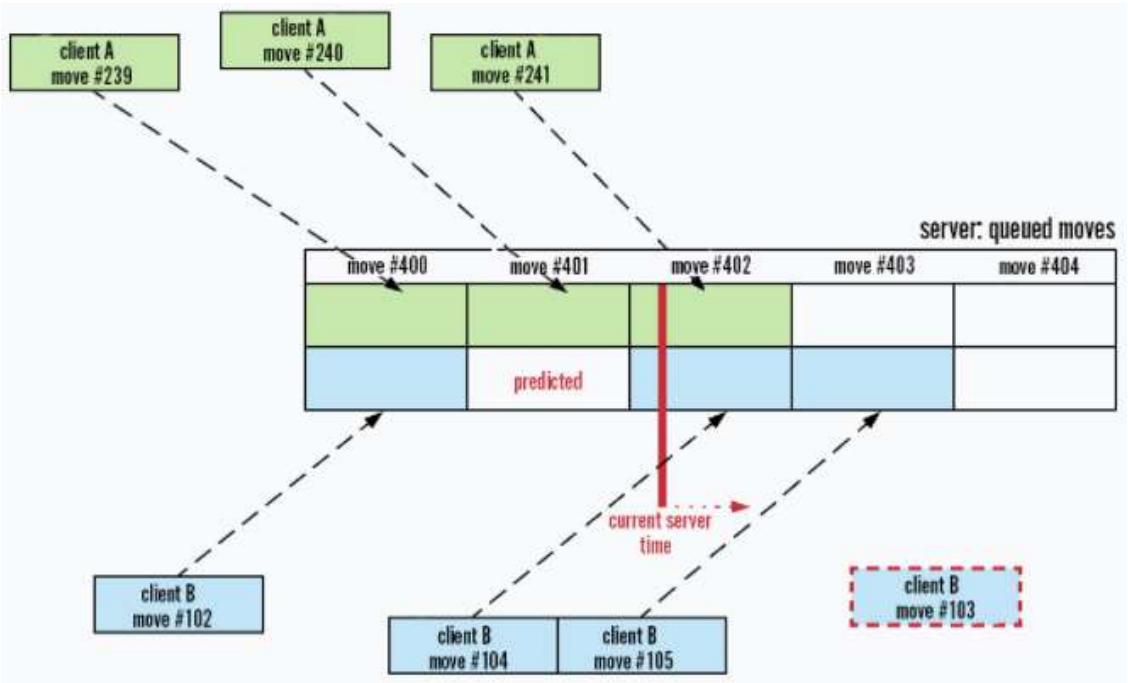

Fig.3.15 Movement chunks division

Receiving data from a client, server creates a baseline between that client's timeline and its own (client A's update number 239 corresponds to server update number 400). When each update comes from a client, the server places it in the appropriate slot in its queue. The network buffering comes in handy. The healthy queue of incoming moves to performed is maintained by the set up by a server and is adjusted if it is necessary. The queue is as short as possible to reduce the latency, but long enough to neglect the different rate at with the updates are delivered [17].

To maintain the simulation server executes the queues and sends results to all the clients, but sometimes the update from the client side is not received when it is needed (step number 401), so the server assumes that the last received pressed key is hold and the, since only a few milliseconds have passed.

3.4 Creation of Game Server with Netty and Communication Mechanism

The architecture of the game implies that server is built on the sockets, Protobuf as structured data serialization protocol defines the format of the message exchange, each in game message is decrypted. Netty is used as asynchronous server, that provides quick and easy way to development scalable network application.

The communication between server and client is implemented in the following way: both of the have message decoding and decryption handling mechanisms, those are entry points for each message, after the packet is received, the message is decrypted, its type is defined according to the type sent to the corresponding handler. Exit point is represented by message encoding and encryption handling mechanisms. When message is sent to Netty it will go through the encryption process and here the decision is made whether to encrypt, and how to wrap [18].

Each message is wrapped into protobuf wrapper (fig.3.16); the receiving side checks the data inside the wrapper for the further choice of a proper handler.

```
message CryptogramWrapper {
    bytes registration = 1;
}

message Wrapper {
    Utility utility = 1;
    CryptogramWrapper cryptogramWrapper = 2;
}
```

Fig.3.16. Example of the message

The message exchange process relies on the decoding-encoding principle, in case if some additional command needs to be added, new additional handler. For example, client performs registration and the message goes to the encoder, wrapped and is sent to the server. On the server side decryption handler kicks in and decrypts message if it is needed, defines the type by the presence of specific fields and starts operation execution (fig.3.17).

```
if(wrapper.hasCryptogramWrapper())
{
    if(wrapper.getCryptogramWrapper().hasField(registration_cw)) {
        // process registration
        byte[] cryptogram = wrapper.getCryptogramWrapper().getRegistration().toByteArray();
        byte[] original = cryptography.Decrypt(cryptogram, cryptography.getSecretKey());

        RegModels.Registration registration = RegModels.Registration.parseFrom(original);
        new Registration().saveUser(ctx, registration);
    }
    else if (wrapper.getCryptogramWrapper().hasField(shoot_cw)) {
        // process shooting
    }
}
```

Fig.3.17. Message processing on the server side

So for the new additional functionality the following steps need to be done:

- New Protobuf message type needs to be created, include it to the Wrapper and CryptogramWrapper;
- Declare the fields that need access in the client and server descriptors;
- Define new class for the logic processing;
- Add new condition in decoding-encoding mechanism in client and server for the further logic processing.

The project utilizes TCP protocol, of course the complementation of the UDP is preferred. TCP is waiting for acknowledgment before continuing to send, this creates delays, and it will be difficult to achieve ping less than 100, if the packet is lost during transmission over the network, the game stops and waits until the packet is re-

delivered. Unfortunately, there is no way to change this behavior of TCP, and it is not necessary, since this is the essence of TCP. The choice of the type of sockets depends entirely on the genre of the game, in action games it is important not what happened a second ago, but the most relevant state of the game world. The data needs to travel from client to server as quickly as possible without any resent delay. This is why TCP should not be used for multiplayer games.

If the reliable UDP is needed the following issues will be faced: implementation of the ordering mechanism, the ability to turn off delivery confirmation, control the channel load, send large messages more than 1400 bytes. For the game data transferring, that requires often update such as player movement UDP is a best option. Simultaneous usage of the TCP and UDP leads to the increased UDP data loss rate, because TCP has more priority [19].

The server is based on Netty, it takes care of working with sockets, implementing a convenient architecture. Multiple data handlers can be connected together; first handler deserializes the incoming message and processes data directly. Flexible management of the library allows to allocate the required number of threads or memory to it. If at some stage of development additional data processing is required, it is enough to add a new handler to pipeline, which greatly simplifies application support. Figure 3.18 depicts overall structure of the server.

```
public class ServerInitializer extends ChannelInitializer<SocketChannel> {
    @Override
    protected void initChannel(SocketChannel ch) throws Exception {
        ChannelPipeline pipeline = ch.pipeline();
        // Decoders protobuf
        pipeline.addLast(new ProtobufVarint32FrameDecoder());
        pipeline.addLast(new ProtobufDecoder(MessageModels.Wrapper.getDefaultInstance()));
        // Encoder protobuf
        pipeline.addLast(new ProtobufVarint32LengthFieldPrepender());
        pipeline.addLast(new ProtobufEncoder());
        /* Close a connection if there were no incomming messages in 30 seconds */
        pipeline.addLast(new IdleStateHandler(30, 0, 0));
        /* encrypt outcomming message  */
        pipeline.addLast(new EncryptHandler());
        /* decrypt incomming message */
        pipeline.addLast(new DecryptHandler());
    }
}
```

Fig.3.18 Netty application architecture

The advantage of this approach is that the server and the handlers can be spread across different machines, having received a cluster for calculating game data,

providing flexible structure. While the loads are small, everything can be processed on one server. As the load increases, the logic can be separated into a separate machine.

Special client for the check of hits can be created the help of Unreal Engine, that will accept parameters of the shot, position object in the world based on its movement at the moment of shot, simulate the shot and giving back hit data to the server. The implementation of the connection to the server, connection restore in case it was lost, listening of the incoming messages and checking the server's availability is the first step in client application development. The main object to work with a network on the client side is a Socket, custom created class that derives from UObject. This class holds all the necessary information for the socket creation: the TCP socket itself, TCP server address, connection status, UDP socket and server address, local port numbers and the methods for the initialization and reconnection mechanisms.

As a next step the initialization of the sockets is implemented; allocation of a buffer, local ports assignment (fig.3.19).

```
void USocketObject::InitSocket(FString server_address, int32 tcp_local_p,
                               int32 tcp_port, int32 udp_local_p, int32 udp_port)
{
    int32 buffer_size = 2 * 1024 * 1024;
    // port numbers
    tcp_local_port = tcp_local_p;
    udp_local_port = udp_local_p;
    // tcp
    tcp_socket = ISocketSubsystem::Get(PLATFORM_SOCKETSUBSYSTEM)->CreateSocket(NAME_Stream,
                                                                TEXT("TCP_SOCKET"), false);
    // create a proper FInternetAddr representation
    tcp_address = ISocketSubsystem::Get(PLATFORM_SOCKETSUBSYSTEM)->CreateInternetAddr();
    // parse server address
    FIPv4Address server_ip;
    FIPv4Address::Parse(server_address, server_ip);
    // and set it
    tcp_address->SetIp(server_ip.Value);
    tcp_address->SetPort(tcp_port);
    tcp_socket->Connect(*tcp_address);
    // set the initial connection state
    bIsConnection = Alive();
    // udp
    udp_address = ISocketSubsystem::Get(PLATFORM_SOCKETSUBSYSTEM)->CreateInternetAddr();

    FIPv4Address::Parse(server_address, server_ip);

    udp_address->SetIp(server_ip.Value);
    udp_address->SetPort(udp_port);
    // create socket
    udp_socket = FUdpSocketBuilder("UDP_SOCKET")
        .AsReusable()
        .BoundToPort(udp_local_port)
        .WithBroadcast()
        .WithReceiveBufferSize(buffer_size)
        .WithSendBufferSize(buffer_size)
        .Build();
}
```

Fig.3.19 Socket initialization process

These are basic abstractions of various platform-specific socket interfaces. The server address information is received from the configuration file and brought to the desired form, after that a connection is established and checked if it was properly done. The method for connection check sends empty message to the server and waits for the approval that it was sent. In order to define the structure of serializable data a .proto file with the source code of this structure is created (fig.3.20). Then this data structure is compiled into classes by a special compiler.

```
syntax = "proto3";

message Player {
    string player_name = 1;
    string team = 2;
    int32  health = 3;
    PlayerPosition playerPosition = 4;
}

message PlayerPosition {}
```

Fig.3.20 The structure of the message with player data sent to the server

The initialization and running of the Netty TCP and UDP servers require:

- NioEventLoopGroup is a multi-threaded loop that handles I / O operations. Netty provides different EventLoopGroup implementations for different modes of transport;
- ServerBootstrap is a class that installs the server;
- NioServerSocketChannel class that is used to create a new channel for receiving incoming connections;
- Handler, which provides an easy way to initialize a channel after registering it with EventLoop. It includes incoming message handlers, decoders, encoders and logic;
- Bind and start accepting incoming connections;
- Wait until the server socket is closed.

Pipeline defines what each message goes through and contains a list of handlers that process incoming and outgoing messages. For example, one of the handlers can only accept string data, the other protobuff, if we call write (string), then a handler for strings will be called, in which we decide to process the message further, send it to

another handler corresponding to the new type, or send it to the client. Each handler has a type that determines whether it is for incoming or outgoing messages.

In order to send a message from the client to the server with only one field ‘alive’ that always takes the Boolean value ‘true’, the sending mechanism is implemented (fig.3.21), it determines the format, wraps it up, serializes and sends message via UDP.

```
bool USocketObject::SendByUDP(google::protobuf::Message * message)
{
    Wrapper wrapper;

    if (message->GetTypeName() == "Utility") {
        Utility * mes = static_cast<Utility*>(message);
        wrapper.set_allocated_utility(mes);
    }

    size_t size = wrapper.ByteSize() + 5;
    uint8_t * buffer = new uint8_t[size];

    google::protobuf::io::ArrayOutputStream arr(buffer, size);
    google::protobuf::io::CodedOutputStream output(&arr);

    output.WriteVarint32(wrapper.ByteSize());
    wrapper.SerializeToCodedStream(&output);

    if (wrapper.has_utility()) {
        wrapper.release_utility();
    }

    int32 bytesSent = 0;
    bool sentState = false;

    sentState = udp_socket->SendTo(buffer, output.ByteCount(), bytesSent, *udp_address);

    delete[] buffer;
    return sentState;
}
```

Fig.3.21 Sending message by UDP mechanism implementation

Additional methods for setting up the speed of checking for new messages, reading he size and parsing the data are also provided.

After all the preparation is done, the method for the test UDP message sending is added and is bind to the ‘T’ key button (fig3.22).

```
void ATestGameMode::TestSendUPDMessage()
{
    GLog->Log("send ->>>");
    std::shared_ptr<Utility> utility(new Utility);
    utility->set_alive(true);
    USocketObject::SendByUDP(utility.get());
}
```

Fig.3.22 Test message sending implementation

Figure 3.23 shows the result of starting the server and client clicking the button in the server log.

```
udp:
utility {
alive: true
}

/127.0.0.1 7679
июл 10, 2017 4:42:30 PM io.netty.handler.logging.LoggingHandler channelRead
INFO: [id: 0x89373e1f, L:/0:0:0:0:0:0:0:0:7681] RECEIVED: DatagramPacket(/127.0.0.1:7679 =>
/0:0:0:0:0:0:0:0:7681, PooledUnsafeDirectByteBuf(ridx: 0, widx: 5, cap: 2048)), 5B
+-------------------------------------------------+
| 0 1 2 3 4 5 6 7 8 9 a b c d e f |
+--------+-------------------------------------------------+----------------+
|00000000| 04 0a 02 08 01 |..... |
+--------+-------------------------------------------------+----------------+
июл 10, 2017 4:42:30 PM io.netty.handler.logging.LoggingHandler write
INFO: [id: 0x89373e1f, L:/0:0:0:0:0:0:0:0:7681] WRITE: DatagramPacket(=> /127.0.0.1:7679,
UnpooledUnsafeHeapByteBuf(ridx: 0, widx: 5, cap: 5)), 5B
+-------------------------------------------------+
| 0 1 2 3 4 5 6 7 8 9 a b c d e f |
+--------+-------------------------------------------------+----------------+
|00000000| 04 0a 02 08 01 |..... |
+--------+-------------------------------------------------+----------------+
июл 10, 2017 4:42:30 PM io.netty.handler.logging.LoggingHandler flush
INFO: [id: 0x89373e1f, L:/0:0:0:0:0:0:0:0:7681] FLUSH

B Unreal Engine:
Reveived UDP data
Size of incoming data: 5
```

Fig.3.23 Server log message after key is pressed

All it says that the server is up and running, the information transferring works as predicted and the actual result can be viewed. The next step is to implement shooting validation on the server side.

3.5 Server Shooting Hits Verification

The last thing to do is start the game process, synchronize the movement of the players, make a shot and check on the server it was valid. For the synchronization purposes the message about all player's positions (x, y, z) is send to the to the server 20 times per second. To check the hit, the game needs to remember that the player sees the world in the past, due to a certain delay, in order to compensate for it, the server must store the positions of the targets for a certain time to unwind and place objects as they were at the moment the player made a shot. The player's state class is created to hold the movement is some period of time i.e. position, rotation and location.

The navigable map can be used for the storing time-bound positions, it allows to easily find the closest element to a given time, the player can be static and in this case the updates are not received, on the server all the element of the structure is deleted except the last one, if updates do not come, the last element is stored and time is updated for it for further can access. For example, if the sensitivity, time when a hit is possible to occur, equals to one second, so the data used for the comparison purposes cannot be older than a second.

The firing method of the player casts a ray, that collects a data on the hit. To check if there was a hit to the server, the name of the hit target, the time of the hit, the beginning and end of the shot beam is sent.

For the synchronization purposes let's create a timer in the game logic processor to the send the game updates 20 times per second and destroy it when the game end (fig.3.24).

```
void AGameMode::BeginPlay()
{
    // send 20 times per second a player position
    GetWorld()->GetTimerManager().SetTimer(UpdateLocationTimerHandle,
                        this, &AGameMode::SendLocation, .05f, true);
}

void AGameMode::EndPlay(const EEndPlayReason::Type EndPlayReason)
{
    Super::EndPlay(EndPlayReason);
    GetWorld()->GetTimerManager().ClearAllTimersForObject(this);
}

// send the data to the server if some updated took place
void AGameMode::SendLocation() {...}
```

Fig.3.24 Timer for sending game state update

Also a specific class for the game process handling is created and it provides update positions, computing shooting and handlers is the needed game data is provided (fig.3.25)

```
// gets shot parameters and render it
void UGameProcess::ComputeShot(GameData gData)
// gets player's position and updates it
void UGameProcess::UpdatePositions(PlayerPosition playerPosition)

....

else if (gData.has_playerposition())
{
    UpdatePositions(gData.playerposition());
}
else if (gData.has_shot())
{
    ComputeShot(gData);
}
```

Fig.3.25 Mechanism for the player position update and shot computation

On the server side a check is performed, whether the shot was on target or not. There two scenarios if the hit takes place: the tag of the hit target came, we send it to the test server, while it is empty, if there is no tag, then all the parameters of the shot for simulation is send (fig.3.26).

```
if(gameData.hasShot())
{
    /* find the target */
    GameRoom gameRoom = gameRooms.get(ctx.channel().attr(ROOM_OWNER).get());
    /* check the tag of a hit target, if empty send all the info that shot took place */
    if(gameData.getShot().hasField(requestTo_shot_gd)) {
        gameData = gameData.toBuilder().setShot(gameData.getShot().toBuilder()
                .clearTimeStamp().build()).build();

        MessageModels.CryptogramWrapper cw = MessageModels.CryptogramWrapper.newBuilder()
                .setGameModels(ByteString.copyFrom(gameData.toByteArray())).build();
        MessageModels.Wrapper wrapper = MessageModels.Wrapper.newBuilder().setCryptogramWrapper(cw).build();

        gameRoom.recipients.writeAndFlush(wrapper, ChannelMatchers.isNot(ctx.channel()));
    } else {

    }
}
```

Fig.3.26 Server side shot event processing

The last part of the project concerns the test server. It works like this: transmit the intention of the shot and its parameters to the server, the test server places all game objects at the moment of the shot and checks whether there was a hit, and if there was, it applies damage and updates the state on the clients.

Let's create an empty Unreal Engine project and add the communication with the network from the client, the message handling model MessageDecoder-MessageEncoder, the protobuf model and the instances of the objects under test [20]. Let's add a new VerificationServerConnection class to the TCP package on the server, which will listen and send messages over the socket, add the port of the test server. On

the server we position a player on the map, simulate the shot, if the hit occurs send back a signal for the success (fig.3.27). Next, update the state of the player for everyone.

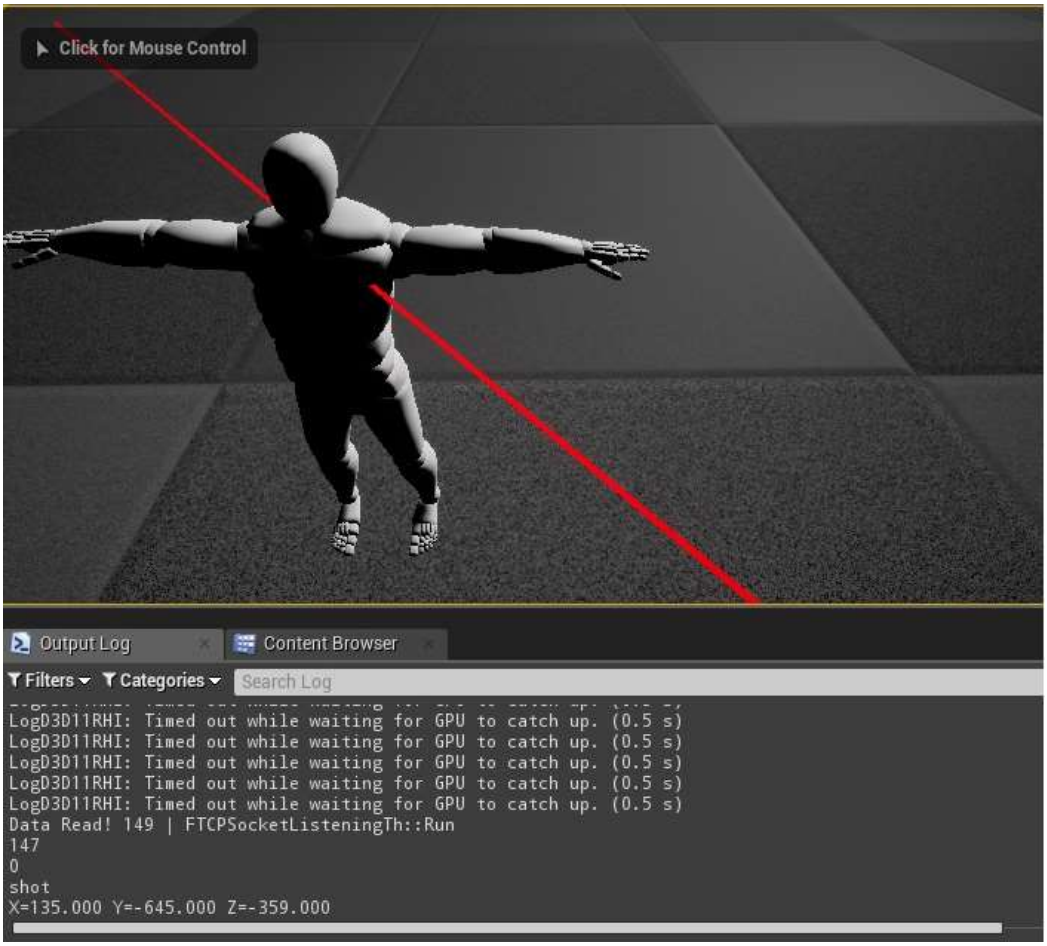

Fig.3.27 Screenshot of the hit result in the Unreal Engine

The server side shot verification system works and brings a desirable result.

Conclusions on the Third Part

Multiplayer networked game development is a rather complicated process, but the presence of online component of the game will guarantee the success of the final product.

The extrapolation and interpolation methods can be used to determine the position of other players on the client side.

The general interpolation algorithm includes the following steps:

- Every server update has start time mark;

- The end time is calculated with respect to the start time using interpolation delta value.
- The timestamp between the start and end time defines how much time has passed since last update.

The resulting ration is used for the variables interpolation.

Good understanding of interpolation and extrapolation is crucial for the lag compensation techniques development, because it also can become a reason of the lag. Lag compensation is a normalizing of the game world state for each client as player's commands are performed on the server.

The architecture of the game implies that server is built on the sockets, Protobuf as structured data serialization protocol defines the format of the message exchange, each in game message is decrypted. Netty is used as asynchronous server, that provides quick and easy way to development scalable network application.

The communication between server and client is implemented in the following way: both of the have message decoding and decryption handling mechanisms, those are entry points for each message, after the packet is received, the message is decrypted, its type is defined according to the type sent to the corresponding handler.

The advantage of this approach is that the server and the handlers can be spread across different machines, having received a cluster for calculating game data, providing flexible structure.

On the server side a check is performed, whether the shot was on target or not. There two scenarios if the hit takes place: the tag of the hit target came, we send it to the test server, while it is empty, if there is no tag, then all the parameters of the shot for simulation is send.

Unreal Engine project and add the communication with the network from the client, the message handling model MessageDecoder-MessageEncoder, the protobuf model and the instances of the objects under test.

CONCLUSIONS

The development of the technologies brings up a huge impovement in the game engineering field. Computer networking is a branch of computer science, telecommunication and computer engineering and its was influenced by a wide range of the technologies and historical events.

Overview of the early networked products sheds light on the basic network architectues, that are used till the modern days:

- Peer to peer;
- Client-server.

Refrences the problems ecountered by the pioneers of the field. Defines the basic communication protocols:

- TCP;
- UDP.

Taking into account their pros and cons it is recommended not just using UDP, but using UDP only and nothing else, better not to use TCP and UDP together — instead implementation of the needed TCP tweaks is more preferable using UDP.

The realism of multiplayer game depends on how well network part allows all game clients to communicate in timely and predictable way. Considering the field of networks, essential features and properties need to be defined and summarized by the concept of metrics.

The analysis of the networking problems allows to define those metrics of communication network:

- Lag is a sort of downtime and delays that arise during the transmission of data packets, as well as in the process of processing the data received or prepared for dispatch, are determined.;
- Loss is a loss of data during the transfer process to the final destination.
- Jitter is a latency changes and the direction in which it goes over the time, a jitter in the network can be viewed in the short term, but the overall latency is rather constant.

Lots of ways of the network condition measurement tools allows to check the values of the listed metric and help to define the overall picture of the used network.

Multiplayer networked game development is a rather complicated process, but the presence of online component of the game will guarantee the success of the final product.

The extrapolation and interpolation methods can be used to determine the position of other players on the client side.

The resulting ration is used for the variables interpolation.

Good understanding of interpolation and extrapolation is crucial for the lag compensation techniques development, because it also can become a reason of the lag. Lag compensation is a normalizing of the game world state for each client as player's commands are performed on the server.

The architecture of the game implies that server is built on the sockets, Protobuf as structured data serialization protocol defines the format of the message exchange, each in game message is decrypted. Netty is used as asynchronous server, that provides quick and easy way to development scalable network application.

The communication between server and client is implemented in the following way: both of the have message decoding and decryption handling mechanisms, those are entry points for each message, after the packet is received, the message is decrypted, its type is defined according to the type sent to the corresponding handler.

The advantage of this approach is that the server and the handlers can be spread across different machines, having received a cluster for calculating game data, providing flexible structure.

On the server side a check is performed, whether the shot was on target or not. There two scenarios if the hit takes place: the tag of the hit target came, we send it to the test server, while it is empty, if there is no tag, then all the parameters of the shot for simulation is send.

The result of the permormed work is a basic client-server multiplayer game with a client created by means of Unreal Engine and Netty server. The created system can be easy expanded to provide wider functionality and process and verify varous actions performed by a client. The server side can be splitted into several machines to perform distinct task and increase overall network performace.

REFERENCES

1. ДСТУ ISO/IEC 15910:2012, IDT. Документування програм. Документація користувача / Київ: «Український науково-дослідний і навчальний центр проблем стандартизації, сертифікації та якості», 2013. – 37 с.

2. ДСТУ ISO 5457:2006. Документація технічна на вироби. Кресленики. Розміри та формати / Київ: «Український науково-дослідний і навчальний центр проблем стандартизації, сертифікації та якості», 2019. – 41 с.

3. Бойченко С.В, Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету / СМЯ НАУ П 03.01(10) – 02 – 2017.

4. Zhukov I.A., Kudrenko S.O., Fomina N.B. Graduation Project Guidelines / Compilers. K.: NAU; 2019

5. “Latency compencation methods in client/server in game protocol desing and optimization” URL: https://developer.valvesoftware.com/ wiki/ Latency_Compensating_Methods_in_Client/Server_In_game_Protocol_Design_ and_Optimization.

6. Grenville Armitage, Networking and Online Games: Understanding and Engineering Multiplayer Internet Games, NY, 1993.

7. Josh Glazer, Multiplayer Game Programming: Architecting Networked Games, Wiley, New York, 2006.

8. Christian Benvenuti, Understanding Linux Network Internals: Guided Tour to Networking on Linux, New York, 2016.

9. “Valorant's 128-tick servers” URL: https://technology.riotgames. com/news/valorants-128-tick-server

10. Глейзер Д., Мадхав С. Многопользовательские игры. Разработка сетевых приложений, Addison-Wesley, Boston, 2017.

11. “MMO с нуля. С помощью Netty и Unreal Engine” URL: https://habr.com/ru/post/333788/

12. “Методы компенсации латентности в разработке и оптим протоколов Клиент-Серверной игры” URL: https://annimon.com/article/3497

13. “Мультиплеер в быстрых играх” URL: https://habr.com/ru/post/302394/

14. “Fast-Paced Multiplayer (Part III): Entity Interpolation” URL: https://www.gabrielgambetta.com/entity-interpolation.html

15. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы, Addison-Wesley, Sydney, 2015.

16. “Как мы писали сетевой код мобильного PvP шутера: синхронизация игрока на клиенте” URL: https://habr.com/ru/company/pixonic/blog/415959/

17. “SQL Server” URL: http://www.microsoft.com/en-us/server-cloud/products/sql-server/

18. “О сетевой модели в играх” URL: https://habr.com/ru/post/467025/

19. Hyvärinen A., Karhunen J., Oja E. Independent Component Analysis / Wiley, 2001

20. Thor Alexander, Massively Multiplayer Game Development, New York, 2013.