



SVEUČILIŠTE U ZAGREBU



Fakultet  
elektrotehnike i  
računarstva

Diplomski studij

Računarstvo

Akademska godina  
2022/2023

# Umrežene igre

Programski koncepti umreženih igara



# Sadržaj

- Podsjetnik iz raspodijeljenih sustava
- Serijalizacija i replikacija
- Prikљučnice
- Poziv udaljenje procedure
- Sučelja niske i visoke razine

# Podsjetnik iz raspodijeljenih sustava

# Tipovi promjenjivih informacija u igrama

- **Promjenjive informacije** su one koje se tijekom izvođenja igre mogu promijeniti
  - Tko treba dobiti informaciju koja se promijenila?
    - Specifični klijent?
    - Poslužitelj?
    - Svi klijenti?
  - Koliko često se informacija mijenja (i/ili šalje)?
    - Svaki okvir?
    - Svakih par sekundi?
    - Ovisno o stanju igre?
    - Primjeri:
      - Položaj dinamičkog objekta – mijenja se svaki okvir te ga možemo staviti u osvježavanje stanja
      - Podaci o stanju udaljenog drugog igrača koji ne utječe na našeg igrača – svakih par sekundi
      - Kad je igrač pokupio unaprjeđenje ili municiju – ovisno o stanju igre
  - Koliko je velika informacija?
    - Mala – inkrementalni položaj objekta?
    - Veća – podaci o karakteristikama objekta koji je stvoren u nekoj kutiji?
- O svim ovim karakteristikama ovisi koja i kakva metoda koja će biti korištena za prijenos informacija preko mreže!

# Raspodijeljeni sustavi (podsjetnik)

- “Sustav u kojem programske i sklopovske komponente umreženih računala komuniciraju i uskladjuju svoje aktivnosti isključivo razmjenom poruka”
- Više računala
  - Spojeno na mrežu za razmjenu poruka
  - Izvršava svoj program
  - Neki program se izvršava na jednom (A,X), a neki i na više računala (B)
  - Ako je potrebno za neku zadaću da komuniciraju programi A i B treba uspostaviti komunikaciju između računala 1 i računala 2 ili nekog računala *i*
- Sloj raspodijeljenog sustava ima ulogu programskog posredničkog sloja, tj. međuopreme (engl. middleware) koji omogućuje povezivanje, komunikaciju i suradnju aplikacija, sustava i uređaja te omogućuje interakciju programa na aplikacijskoj razini

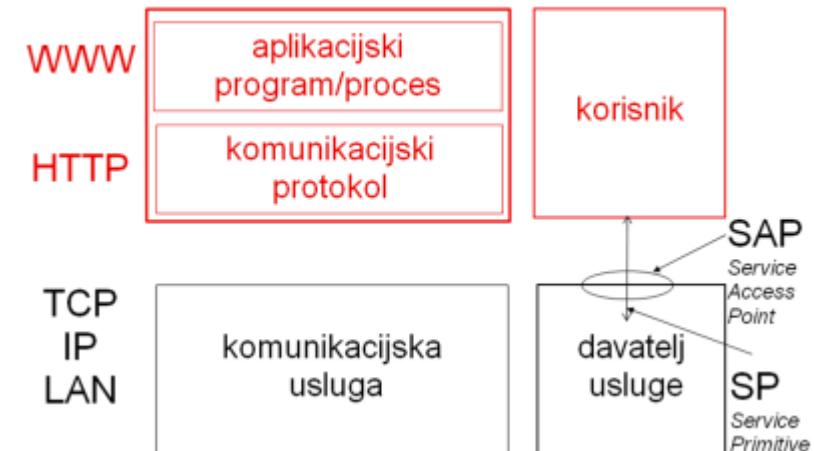


Izvor: Raspodijeljeni sustavi Radna inačica udžbenika v.1.3

[https://www.fer.unizg.hr/\\_download/repository/Rassus-2016\\_udzbenik\\_v\\_1\\_3.pdf](https://www.fer.unizg.hr/_download/repository/Rassus-2016_udzbenik_v_1_3.pdf)

# Slojeviti arhitekturni model

- Sustav se vertikalno dekomponira na slojeve, a svakome se sloju dodjeljuju funkcije i specificiraju sučelja sa susjednim slojevima, kako bi viši sloj mogao koristiti usluge nižeg sloja
  - Open System Interconnection Reference Model, OSI RM,
  - TCP/IP Reference Model
- Točka međudjelovanja između dva susjedna sloja naziva se točkom pristupa usluzi (engl. Service Access Point, SAP), a opisuje se uslužnim primitivima (engl. Service Primitive, SP) određenih nazivom i skupom parametara.
- Aplikacijski sloj određuje funkcionalnost te sadrži aplikacije i usluge namijenjene korisnicima
  - Primjer Weba
    - Aplikacijski program je web pretraživač
    - Aplikacijski protokol je HTTP
    - Ova struktura koristi komunikacijsku uslugu nižih slojeva (TCP, IP, fizički sloj)
  - Primjer igara
    - Aplikacijski program je igra
    - Aplikacijski protokol je često zatvoren i vezan za sam sadržaj igre (imat ćemo primjere na laboratoriju)
    - Ta struktura koristi komunikacijsku uslugu nižih slojeva (TCP, IP, fizički sloj)
- Povezanost dva procesa u aplikacijskom sloju naziva se asocijacijom (združivanjem), umjesto vezom (konekcijom), kako bi se naglasila slaba povezanost entiteta: pri razmjeni podatak primatelj a-priori ne mora znati tko je pošiljatelj, a raspoloživost i pouzdanost asocijacije mogu varirati



# Slojeviti arhitekturni model

- Dva se procesa u aplikacijskom sloju združuju da bi komunicirali i razmijenili podatke, stvarajući asocijaciju posredovanjem nižih slojeva koji imaju ulogu davatelja usluge, putem kojih se ostvaruje komunikacija između računala na kojima su smješteni
- Procesi su korisnici usluga nižih slojeva, pri čemu se u općem slučaju koriste ovi uslužni primitivi:
  - zahtjev (engl. request) – upućuje ga korisnik kako bi od davatelja usluge zatražio izvršenje određene funkcije
  - indikacija (engl. indication) – upućuje ga davatelj usluge kako bi od korisnika zatražio izvršenje određene funkcije ili ga izvijestio da je od drugog korisnika primio zahtjev.
  - odziv (engl. response) – upućuje ga korisnik kao odgovor na prethodno primljenu indikaciju.
  - potvrda (engl. confirm) – upućuje ga davatelj usluge kao odgovor na zahtjev

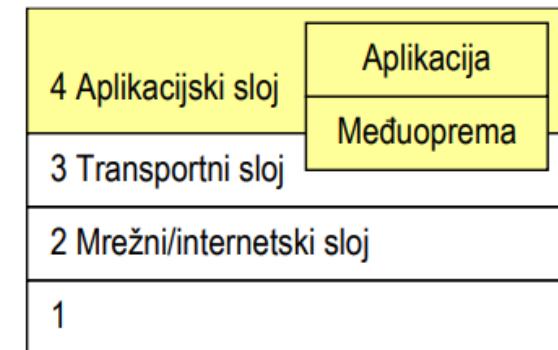


# Osnovni modeli raspodijeljene obrade

- **Klijent poslužitelj** – klijent traži uslugu od poslužitelja, a poslužitelj je pruža za više, u pravilu mnogo klijenata uz svojstvo konkurenčijske transparentnosti
- **Ravnopravni sudionici** (engl. peer to peer – P2P) – su takvi programi koji mogu djelovati i kao klijent i kao poslužitelj. Ravnopravni sudionici tvore sustav u kojem međusobno komuniciraju (engl. Peer-to-Peer, P2P) tako da se na aplikacijskom sloju povezuju u “prekrivajuću mrežu” (engl. overlay network) čvorova – ravnopravnih sudionika
- **Premještanje programa i programskih agenata** – premještanje programa s jednog na drugo umreženo računalo zahtijeva posebna rješenja za migraciju programskog kôda (engl. code migration)

# Programski posrednički sloj

- Programski posrednički sloj ili međuoprema (engl. middleware) je programski sustav na aplikacijskom sloju koji se nalazi između transportnog sloja i aplikacije
  - Koristi usluge transportnog sloja
  - Pojednostavljuje razvoj aplikacija
  - Najčešće u obliku programskih knjižnica za programere aplikacija
- Za klijent poslužitelj ključne tri tehnike
  - **Prikљučnice** (engl. socket) – pristupne točke preko kojih aplikacija šalje podatke u mrežu i prima podatke s mreže
  - **Poziv udaljene procedure** (engl. Remote Procedure Call – RPC) – omogućuje procesima pozivanje i izvođenje procedura na udaljenom računalu
  - **Poziv udaljene metode** (engl. Remote Method Invocation – RMI) – je nasljednik poziva udaljene procedure razvijen za objektne jezike jer se poziva metoda udaljenog objekta



# Serijalizacija i replikacija

# Serijalizacija

- **Serijalizacija** je zapravo pretvaranje određenog objekta iz njegovog opisa u memoriji u linearan set bitova koji se može zapakirati u paket transportnog sloja
- **Deserijalizacija** je obrnuti proces u kojem se iz seta bitova rekreira objekt
- Bitna kako bi se poslali cijeli objekti i kako bi se repliciralo i stanje njihovih spremnika
- Serijalizacija može imati problema kada obrađujemo složenije podatkovne oblike
- Primjerice ako bi se slao pokazivač na određeno memorijsko mjesto, on bi se serijalizirao na dobar način, ali na tom memorijskom mjestu na drugom računalu neće biti željena vrijednost
- Zbog ovih problema koristi se koncept podatkovnog strujanja

# Primjer

- Primjer klase s pohranom korisnikovog zdravlja
- Informacije o zdravlju su pohranjene u varijabli koje pokazuju na određeno mjesto u memoriji
- Kada se objekt u cijelosti serijalizira prenose se i vrijednosti koje su pohranjene za varijablu „healthPoints“
- Umnogome je lakše serijalizirati objekt i poslati ga u cijelosti nego raditi kompletno kloniranje objekta na drugoj strani.
- Serijalizacija može imati problema kada obrađujemo složenije podatkovne oblike
- Primjerice ako bi se slao pokazivač na određeno memorijsko mjesto, on bi se serijalizirao na dobar način, ali na tom memorijskom mjestu na drugom računalu neće biti željena vrijednost
- Dakle, bitno je da se u robusnoj serijalizaciji podataka prenese i vrijednost na koju pokazuje pokazivač.

```
using UnityEngine;
using System.Collections;

public class Health : MonoBehaviour {

    public float healthPoints = 10f; // količina zdravlja

    void Update ()
    {
        if (healthPoints <= 0) {
            Destroy(gameObject);
        }
    }

    public void ApplyDamage(float amount) // uništi objekt
    {
        healthPoints = healthPoints - amount;
    }

    public void ApplyHeal(float amount) // uništi objekt
    {
        healthPoints = healthPoints + amount;
    }
}
```

# Podatkovno strujanje

- **Podatkovno strujanje** je u računalnoj znanosti podatkovna struktura koja u potpunosti obuhvaća uređeni set podatkovnih elemenata te omogućuje korisniku da upisuje podatke u isti ili ga čita
- Strujanje pruža jednostavan način za stvaranje međuspremnika, ispunjavanja međuspremnika vrijednostima iz pojedinačnih polja izvornog objekta, slanje tog međuspremnika na udaljeno računalo, izdvajanje vrijednosti po redu i umetanje istih u odgovarajuća polja odredišnog objekta
- Strujanje koristi metode ugrađivanja (engl. embedding) i povezivanja (engl. linking)
- **Ugrađivanje** omogućuje da se referencirani podaci ugrađuju direktno u sam set podataka strujanja kroz posebnu funkciju za serijalizaciju (primjerice kod pokazivača)
- **Povezivanje** se koristi kada više objekata treba imati isti pokazivač, a u kom slučaju bi ugrađivanje stvorilo nove kopije objekta na koji se pokazuje.
- Povezivanje radi na način da se objektu na koji pokazuje više pokazivača dodaje identifikator na strani pošiljatelja te se onda na strani primatelja taj identifikator u posebnoj funkciji povezuje s kreiranim objektima, odnosno kreira se jedan objekt na kojem pokazuju svi pokazivači s odgovarajućim indikatorom

# Serijalizacija u C# i Unityu

- U C# programskom jeziku klasa Network Stream iz paketa System.Net.Sockets se koristi za čitanje i pisanje iz određene priključnice
- Klase TcpClient i UdpClient imaju metodu iz kojih se mogu izvaditi objekti mrežnog strujanja te se iste mogu koristit za jednostavnu implementaciju strujanja (implementacija u laboratorijskim vježbama)
- U Unity pogonskom sustavu u biblioteci Netcode for GameObjects koristi se INetworkSerializable sučelje za serijalizaciju naprednih tipova podataka
- Osnovni tipovi podataka kao što su bool, int, string i ostalih automatski se serijaliziraju.
- Automatska serijalizacija je omogućena za osnovne Unity primitive poput Vector2, Quaternion, Color32 itd.
- Serijalizacija naprednih klasa mora implementirati sučelje INetworkSerializable.

# Replikacija objekata

- **Replikacijom** se naziva prijenos stanja pojedinog objekta iz jedne instance virtualnog svijeta u drugu instancu koja je odijeljena mrežom (najčešće se poslužitelja na klijent).
- Replikacija objekata je više od samo serijalizacije podataka i slanja s jednog računala na drugo
- Kako bi se stanje objekta uspješno replicirano preko mreže potrebno je napraviti i pripremne mjere
  1. Označiti paket kao paket koji sadrži stanje objekta.
  2. Jedinstveno identificirati replicirani objekt.
  3. Navesti klasu objekta koji se replicira.
- Registrar kreiranih objekata mapira identifikator klase na funkciju koja kreira objekt dane klase
- Kod osvježavanja distribuiranog stanja virtualnog svijeta ne šalje se kompletno novo stanje svijeta već samo **razlika između prošlog stanja i novog stanja (engl. delta)**, a samo periodički kompletno novo stanje
- **Parcijalna replikacija** omogućava da se replicira samo onaj dio stanja objekta koje se promijenio
- Primjerice ako se objektu promijenila lokacija, ali nije orijentacija u razlici se šalje samo razlika u lokacijskim koordinatama objekta
- Često potreban je i poziv udaljenih metoda odnosno izvođenje koda, a ne samo replikacija stanja objekata
- Replikacija je ključni element sučelja za višekorisničke igre niže razine

# Priklučnice (engl. socket)

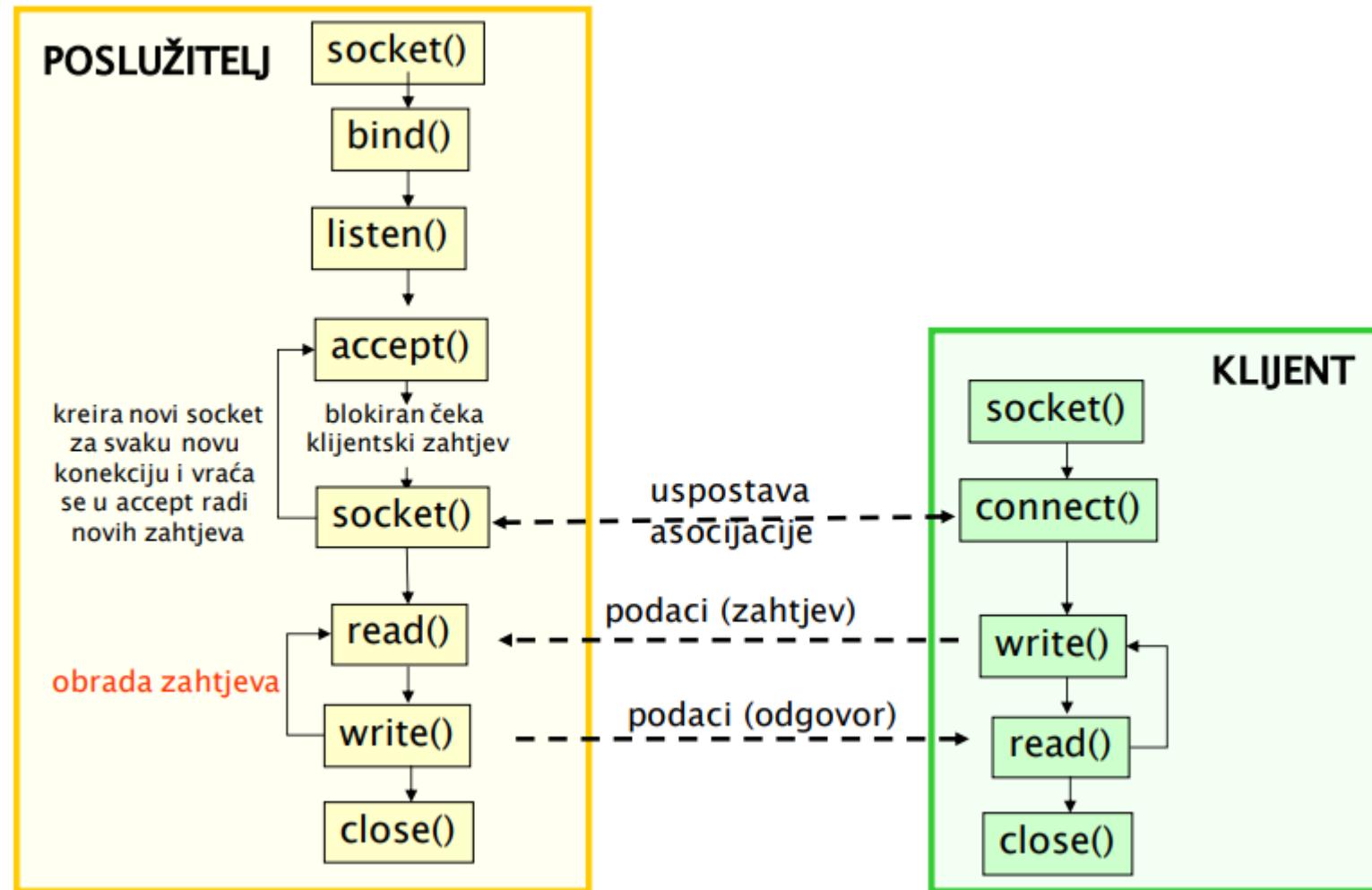
# Priklučnice

- **Priklučnice ili punim imenom Berkeley sockets** su programsko sučelje za spajanje računala putem mreže
  - Programsko sučelje priključnica je kreirano početkom 80-ti kreirano početkom 80-tih kao dio originalne Berkeley distribucije Unixa
  - Temelji se na programskom modelu klijent – poslužitelj
  - Potrebno je poznavati IP adresu i vrata na koja se spaja na udaljenom računalu
  - Korisničko sučelje definira sljedeće metode za upravljanje priključnicama
    - SOCKET() za kreiranje priključnice,
    - BIND() za povezivanje lokalne adrese s priključnicom,
    - LISTEN() koja drugim uređajima javlja da je priključnica spremna primiti zahtjev za povezivanje te koja moguće operacijskom sustavu rezerviranje sredstva (spremnika) za specificirani maksimalni broj konekcija
    - ACCEPT() koja prihvaca zahtjev za povezivanje i kreira novu priključnicu za svaki zahtjeva. Stvara se nova Poslužitelj stvara novi priključnica koji se koristi za komunikaciju s klijentom. Originalna se koristi za primanje novih zahtjeva.
    - CONNECT() za uspostavljanje direktne komunikacije preko priključnica
    - SEND() za slanje podataka preko uspostavljene veze
    - RECEIVE() za primanje podataka preko uspostavljene veze
    - CLOSE() za zatvaranje uspostavljene veze.
  - Napomena: Mogu se koristiti i READ() i WRITE() metode umjesto RECEIVE() I SEND(), kao generički deskriptori u UNIX sustavima, ali onda nemaju određene zastavice koje se mogu podesiti

# Priklučnice

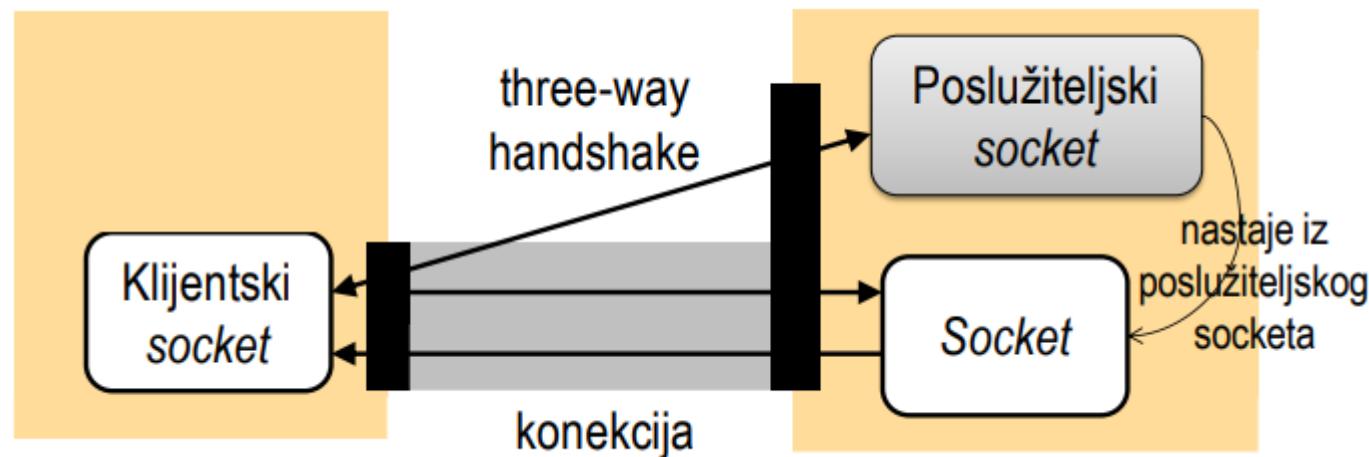
- Sama priključnica je apstraktna reprezentacija komunikacijske krajnje točke te za aplikaciju predstavlja datoteku koja joj omogućava čitanje i pisanje podataka s mreže i na mrežu (na Unixu su svi I/O uređaji, pa i mreža, modelirani kao datoteke)
- Klijenti i poslužitelji komuniciraju čitanjem i pisanjem podataka u „datoteku“ priključnice
- Osnovna razlika između regularnih datoteka i priključnica je način „otvaranja“ deskriptora:
  - uspostavljanje konekcije i
  - specifikacija adresa krajnjih točaka komunikacije

# Dijagram toka uspostave veze



# Kreiranje novog socketa

- Metoda accept je blokirajuća jer osluškuje zahtjeve iz mreže, a za svaki primljeni korisnički zahtjev kreira se novi socket koji je kopija poslužiteljskog socketa
- Novi socket komunicira s klijentskim socketom (stvoren na istim vratima kao originalni poslužiteljski socketi)
- Operacijski sustav zna kojem socketu proslijediti podatke m s obzirom da je konekcija identificirana pomoću para transportnih adresa između klijenta i poslužitelja



# Priključnice u C#

- Unutar pogonskog sustava videoigara Unity koristi se programski jezik C#
- Implementacija priključnica u C# definirana je u klasi Socket iz paketa System.Net.Sockets.
- Priključnica se u navedenoj klasi kreira pomoću četiri različita konstruktora. Primjer jednog konstruktora je:
- `Socket(AddressFamily, SocketType, ProtocolType)`
  - Familija adresa se za protokol IPv4 postavlja na vrijednost InterNetwrk
  - Parametar ProtocolType može poprimiti vrijednosti za TCP ili UDP protokol.
  - Vrijednosti parametra SocketType definiraju obilježja socketa
- Ova tri parametra nisu neovisna jedna od drugom, već mogu jedan definirati drugi - primjerice, ponekad je tip priključnice vezan za određeni protokol

# Tipovi priključnica u C#

| Ime       | Vrijednost | Objašnjenje  |
|-----------|------------|--|
| Stream    | 1          | Podržava pouzdane, dvosmjerne tokove okteta koji se temelje na povezivanju bez dupliciranja podataka i bez očuvanja granica. Utičnica ovog tipa komunicira s jednim klijentom i zahtijeva uspostavu veze s udaljenim klijentom prije nego što komunikacija može započeti. Stream koristi protokol kontrole prijenosa (ProtocolType.Tcp) i obitelj adresa AddressFamily.InterNetwork.                               |
| Dgram     | 2          | Podržava datagrame, koji su nepouzdane poruke bez veze fiksne (obično male) maksimalne duljine. Poruke bi se mogle izgubiti ili duplicirati te bi mogle doći van reda. Utičnica tipa Dgram ne zahtijeva vezu prije slanja i primanja podataka i može komunicirati s više kolega. Dgram koristi Datagram protokol (ProtocolType.Udp) i obitelj adresa AddressFamily.InterNetwork.                                   |
| Raw       | 3          | Podržava pristup temeljnog transportnom protokolu. Koristeći Raw, može se komunicirati pomoću protokola kao što su Internet Control Message Protocol (ProtocolType.Icmp) i Internet Group Management Protocol (ProtocolType.Igmp). Aplikacija mora osigurati potpuno IP zaglavje prilikom slanja. Primljeni datagrami se vraćaju s netaknutim IP zaglavljem i opcijama   |
| Rdm       | 4          | Podržava poruke bez uspostave veze, orientirane na poruke, pouzdano isporučene poruke i čuva granice poruka u podacima. Rdm (pouzdano isporučene poruke) poruke stižu neduplicirane i u redu. Nadalje, pošiljatelj je obaviješten ako se poruke izgube. Ako inicijalizirate Socket pomoću Rdm-a, nije vam potrebna uspostava veze prije slanja i primanja podataka. S Rdm-om možete komunicirati s više klijenata. |
| Seqpacket | 5          | Omogućuje koneksijski orientiran i pouzdan dvosmjerni prijenos podataka. Seqpacket ne duplicira podatke i čuva granice unutar toka podataka. Utičnica tipa Seqpacket komunicira s jednim ravnopravnim klijentom i zahtijeva uspostavu vezu prije nego što komunikacija može započeti.  |

# Implementacija u C#

- Klasa TcpListener
  - Ima jednostavne metode za prihvatanje nadolazećih konekcija
  - Sinkroni i blokirajući način rada
  - <https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.tcplistener?view=net-6.0>
- Klasa TcpClient
  - Ima jednostavne metode za spajanje na TCP socket, slanje i primanje podataka
  - Sinkroni i blokirajući način rada
  - <https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.tcpclient?view=net-6.0>
- Klasa UdpClient
  - Jednostavne metode za spajanje, primanje i slanje datagrama putem UDP socketa
  - Nema poslužitelja na drugoj strani je opet UDP client
  - Sinkroni i blokirajući način rada
  - Potrebno se ipak „spojiti“ na udaljeni UDP klijent
    - Kroz konstruktor gdje se pružaju ime i port udaljenog klijenta
    - Kroz metodu Connect
  - <https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.udpclient?view=net-6.0>
- Dodatna literatura oko priključnica s puno više detalja – Miljenko Mikuc: Mrežno programiranje:  
[https://www.fer.unizg.hr/\\_download/repository/MrePro-2022.pdf](https://www.fer.unizg.hr/_download/repository/MrePro-2022.pdf)

# Laboratorij

- Izrada igre križić – kružić
- Dva računala u lokalnoj mreži
- Potrebno napraviti spajanje priključnicama na poslužitelj
- Za spajanje potrebno detektirati IP adresu poslužitelja
  - Klijent preplavljuje mrežu (broadcast) s paketom adresiranim na dobro znanim vratima
  - Poslužitelj odgovara na broadcast
  - Iz toga se saznaje IP adresa
  - Temeljeno na UDP socketu
- Uspostava veze te prijenos informacija putem TCP socketa
- Jednostavan aplikacijski protokol SYNC, TURN, MOVE, EXIT poruke

# Primjer iz laboratorija – UDP broadcast

- Server

```
void ListenForBroadcast()
{
    var responseData = Encoding.ASCII.GetBytes("TIC");
    var clientEp = new IPEndPoint(IPAddress.Any, 0);

    udpClient = new UdpClient(Constants.PORT);

    while (true)
    {
        var requestData = udpClient.Receive(ref clientEp);

        if (Encoding.ASCII.GetString(requestData) == "TACTOE")
        {
            tcpServer.Start();
            udpClient.Send(responseData, responseData.Length, clientEp);
            tcpClient = tcpServer.AcceptTcpClient();
            break;
        }
    }

    udpClient.Close();
}
```

- Klijent

```
udpClient.Send(requestData, requestData.Length, new IPEndPoint(IPAddress.Broadcast, Constants.PORT));

Task.Run(() =>
{
    try
    {
        Debug.Log("Sending Broadcast");
        responseData = udpClient.Receive(ref serverEp);
    }
    catch (SocketException exc)
    {
        Debug.Log("SocketException caught in SendBroadcast, udpClient was closed or timed out: " + exc.Message);
    }
})
.Wait(5000);

if (responseData != null && Encoding.ASCII.GetString(responseData) == "TIC")
{
    udpClient.Close();
}
```

# Primjer iz laboratoriјa – spajanje i slanje podataka

Spajanje

```
public async void ConnectClient()
{
    byte[] buffer = new byte[256];

    try
    {
        IPAddress serverAddress = await Task.Run(() => SendBroadcast());

        if (serverAddress != null)
        {
            tcpClient = new TcpClient();
            await tcpClient.ConnectAsync(serverAddress, Constants.PORT);

            NetworkStream networkStream = tcpClient.GetStream();
            int read;
            SendSync(networkStream, buffer);
        }
    }

    private void SendSync(NetworkStream networkStream, byte[] buffer)
    {
        ProtocolData reqHeader = new() { };
        reqHeader.messageCode = ProtocolData.MessageCode.SYNC;
        reqHeader.space = ProtocolData.MoveSpace.NULL_SPACE;

        Buffer.BlockCopy(BitConverter.GetBytes((int)reqHeader.messageCode), 0, buffer, 0, 4);
        Buffer.BlockCopy(BitConverter.GetBytes((int)reqHeader.space), 0, buffer, 4, 4);

        networkStream.Write(buffer, 0, 8);
    }
}
```

- Slanje

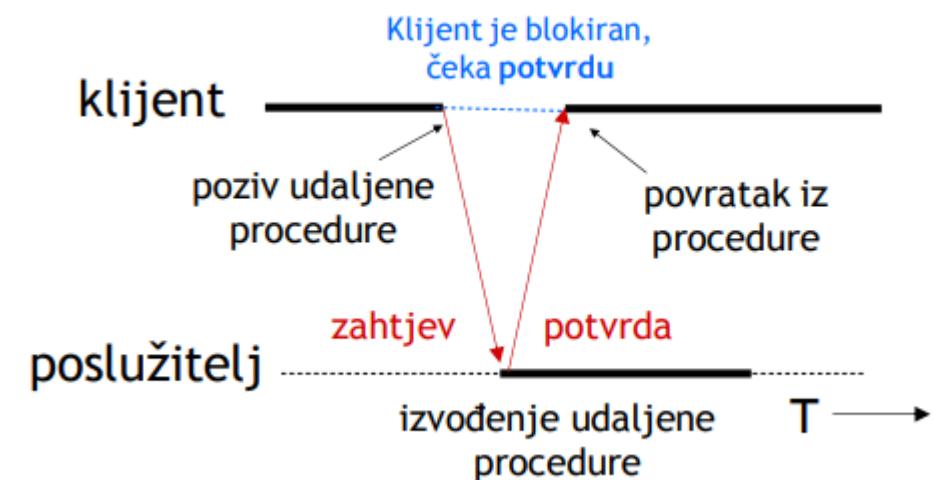
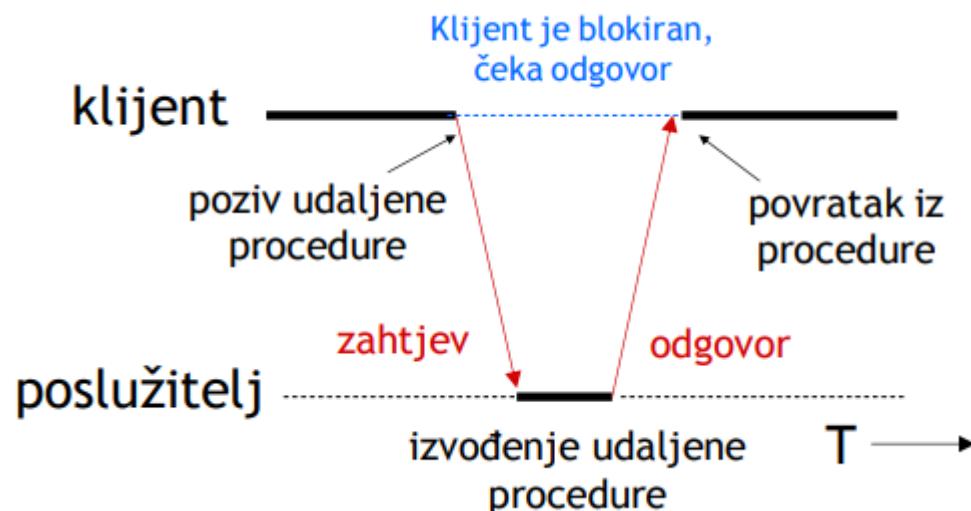
# Poziv udaljene procedure (engl. Remote Procedure Call – RPC)

# Poziv udaljene procedure (engl. Remote Procedure Call – RPC)

- **Pozivi udaljenih procedura (engl. Remote Procedure Call skr. RPC)** omogućuju pozivanje funkcija na udaljenom stroju
- Kada proces koji se izvodi na računalu A poziva proceduru na računalu B, pozivajući proces na računalu A šalje parametre za izvođenje procedure na računalo B i čeka rezultate izvođenja procedure (može biti blokiran ili ne)
- Računalo B izvodi proceduru koristeći primljene parametre i šalje odgovor računalu A
- RPC pokušava pokazati da je poziv udaljene procedure istovjetan kao i poziv lokane procedure (izvedbeno nije tako jednostavno)

# RPC

- Klijent poziva udaljenu proceduru putem posebne komponente nazvane stub koja šalje zahtjev za izvođenje procedure na udaljenom računalu
- Na poslužitelju će zahtjev primiti odgovarajući stub te pozvati proceduru i nakon njenog izvođenja vratiti odgovor klijentu
- Sinkroni RPC – čeka na odgovor, odnosno klijent je blokiran
- Asinkrni RPC – čeka se samo na potvrdu

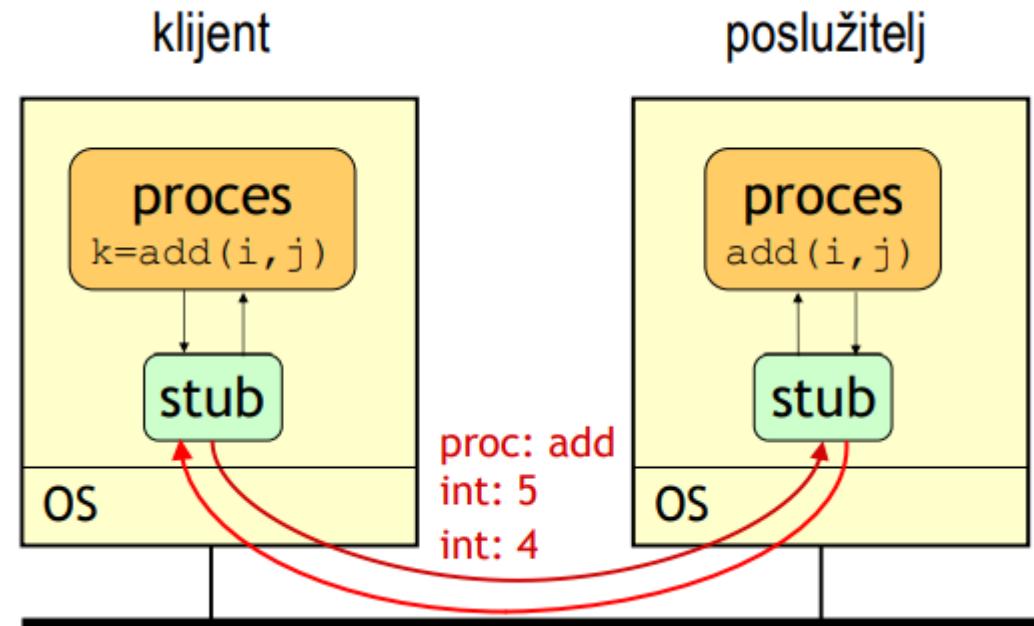


# RPC u igrama

- Poziv udaljene procedure može imati onoliko parametara koliko je potrebno, ali potrebna propusnost mreže povećavat će se s brojem i veličinom parametara
- Poziv udaljene procedure treba dodatni parametar za označavanje primatelja zahtjeva za izvođenjem udaljene procedure
- Primatelji mogu biti:
  - Svi sudionici
  - Svi drugi klijenti
  - Poslužitelj
  - Specifični klijent
- U igrama se najčešće koriste za pojedinačne poruke (a ne kontinuirane poruke osvježenja) koje se moraju poslati kad je ispunjen određeni uvjet najčešće temeljen na igračevoj interakciji
  - Chat poruke
  - Ispaljivanje metka
  - ...

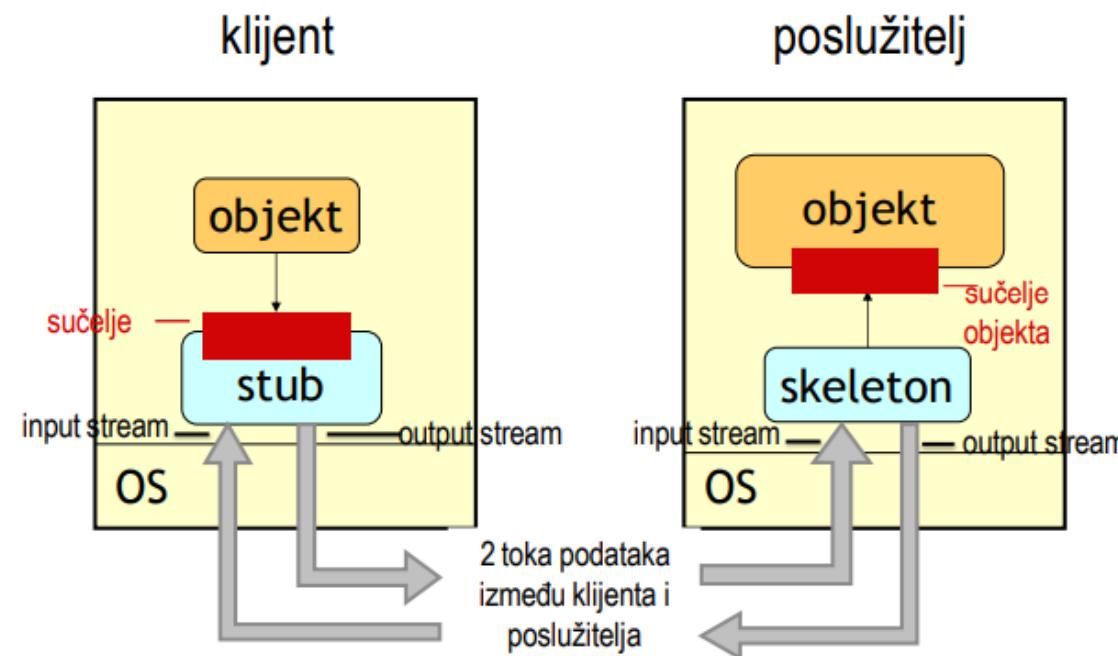
# Primjer RPC-a

- 1. Klijent poziva udaljenu proceduru add(i,j) koristeći lokalni stub.
- 2. Stub "pakira" parametre i identifikator procedure i poziva operacijski sustav (OS) na klijentskom računalu.
- 3. OS klijentskog računala šalje poruku na udaljeno računalo.
- 4. OS udaljenog računala predaje poruku stubu poslužitelja.
- 5. Stub poslužitelja "raspakira" parametre i poziva proceduru add(i,j) koristeći primljene parametre.
- 6. Procedura vraća rezultat izvođenja poslužiteljskom stub-u.
- 7. Stub poslužitelja "pakira" rezultat u poruku i poziva OS.
- 8. OS poslužitelja šalje poruku OS-u klijenta.
- 9. OS klijenta predaje poruku stubu.
- 10. Stub "raspakira" rezultat i predaje ga klijentskom procesu



# Poziv udaljene metode (engl. Remote Method Invocation – RMI)

- Klijentski objekt poziva metodu udaljenog poslužiteljskog objekta na transparentan način tj. identično pozivu metode lokalnog objekta
- Komponenta na strani poslužitelja naziva skeletonom umjesto stubom
- Stub implementira sučelje udaljenog objekta, ali samo kao posrednik koji poziv metode zajedno s parametrima pakira u oblik prikladan za prijenos mrežom i šalje do skeletona koji ga pak preusmjerava do udaljenog objekta
- Koriste 2 toka za prijenos podataka u oba smjera



# Primjer RPC-a u Photonu

- Realizacija smanjenja zdravlja drugim igračima u igri ako ih je određeni igrač pogodio metkom
- RPC metoda koju igrač poziva u lokalnim kopijama drugih igrača kako bi im signalizirao da ih je pogodio
- Provjerava se je li identifikator odgovarajuće Photon komponente taj od lokalnog igrača
- Smanjuje se zdravlje
- Ako je igrač od pogotka umro poziva se funkcija Die

```
[PunRPC]
public void TakeDamage(float _damage, PhotonMessageInfo _info)
{
    if (_info.photonView.IsMine)
    {
        bool _isDead = combatStats.DecreaseHealth(_damage);
        if (_isDead)
        {
            Die(_info.Sender.ActorNumber)
        }
    }
}
```

# Primjer iz laboratorija

- Kreiranje metka na strani svih korisnika kako bi ih vidjeli

```
[PunRPC]
void RPC_CreateBullet()
{
    //Debug.Log("Bullet created! :");
    GameObject bullet = Instantiate(_bulletPrefab, _shootPoint.position, _shootPoint.rotation);
    bullet.GetComponent<Bullet>().SetInfo(((GunInfo)_itemInfo).damage, _photonView.Owner);
    Rigidbody brb = bullet.GetComponent<Rigidbody>();
    brb.AddRelativeForce(Vector3.forward * 3000 * Time.deltaTime, ForceMode.Impulse);
}
```

# Sučelja niske i visoke razine

# Sučelja visoke i niske razine

- Programska sučelja **niže razine** omogućuju **direktno korištenje mrežnih protokola** i specifičnih funkcionalnosti te zahtijevaju dobro razumijevanje funkcioniranja računalne mreže
  - Detaljniji razvoj specifičnih rješenja
  - Moguće više razine optimizacija
- Programska sučelja **više razine** pokušavaju kompleksnost funkcioniranja mreže te izazove s kojima se suočavaju programeri distribuiranih simulacija **pojednostaviti gotovim rješenjima** i omogućiti programerima da se **fokusiraju na izazove u razvoju samih igara**
  - Brži i jednostavniji razvoj
  - Dodatni set usluga (poslužitelji, sustavi za spajanje...)

# Programsko sučelje niske razine

- Primjer com.unity.transport biblioteka
- Omogućuje
  - Konfiguriranje
  - Spajanje
  - Slanje podataka
  - Primanje podataka
  - Zatvaranje veze
  - Odspajanje
  - Vremensko odspajanje veze (engl. timeout)
- Radi na „malo višoj“ razini u odnosu na same priključnice

# Primjer

- Jednostavan klijent koji inkrementira dobiveni broj od poslužitelja
- Potrebni isti podaci kao kod stvaranja priključnice

```
void Start () {
    m_Driver = NetworkDriver.Create();
    m_Connection = default(NetworkConnection);

    var endpoint = NetworkEndPoint.LoopbackIpv4;
    endpoint.Port = 9000;
    m_Connection = m_Driver.Connect(endpoint);
}

void Update()
{
    m_Driver.ScheduleUpdate().Complete();

    if (!m_Connection.IsCreated)
    {
        if (!Done)
            Debug.Log("Something went wrong during connect");
        return;
    }
}
```

# Primjer

```
DataStreamReader stream;
NetworkEvent.Type cmd;
while ((cmd = m_Connection.PopEvent(m_Driver, out stream)) != NetworkEvent.Type.Empty)
{
    if (cmd == NetworkEvent.Type.Connect)
    {
        Debug.Log("We are now connected to the server");

        uint value = 1;
        m_Driver.BeginSend(m_Connection, out var writer);
        writer.WriteUInt(value);
        m_Driver.EndSend(writer);
    }

    else if (cmd == NetworkEvent.Type.Data)
    {
        uint value = stream.ReadUInt();
        Debug.Log("Got the value = " + value + " back from the server");
        Done = true;
        m_Connection.Disconnect(m_Driver);
        m_Connection = default(NetworkConnection);
    }
}
```

# Programsko sučelje visoke razine

- U okviru pogonskog sustava Unity podrška za višekorisničke videoigre (UNet) je godinama bila nepotpuna i nije dobro funkcionala
- Zbog toga je Unity već dugo u procesu izrade novih biblioteka za podršku višekorisničkom igranju te biblioteka
- Netcode for GameObjects objavljena 2021. godine te je ista još u razvoju.
- Zbog pomanjkanja podrške za programska sučelja više razine u pogonskom sustavu Unity treće strane su razvile svoje biblioteke i sustave za podršku višekorisničkim igram razvijenima u pogonskom sustavu Unity.
  - MLAPI
  - DarkRift 2
  - Photon PUN
  - Mirror

# Programsko sučelje visoke razine - primjer

- Photon nudi nekoliko osnovnih Proizvoda za izradu višekorisničkih umreženih aplikacija:
  - Quantum
  - Fusion
- Sakrivaju se funkcije mreže od korisnika
- Photon je platforma sadrži mnoge funkcionalnosti potrebne pri izradi višekorisničkih umreženih igara.
- Neke od poznatijih igara koje su razvijene pomoću ovog alata su Shadowgun Legends, LAMO, Thea 2: The Shattering, Pixel Gun 3D i Guns of Icarus: Online.
- Paket Photon Unity Networking je biblioteka namijenjena jednostavnom razvoju višekorisničkih videoigara u pogonskom sustavu Unity.



# Programsko sučelje visoke razine

- Fusion je softverski razvojni okvir (engl. Software Development Kit skr. SDK) koji je namijenjen profesionalnim razvijateljima u pogonskom sustavu Unity i nudi vrlo napredne opcije poput predikcije na strani klijenta, interpolacije snimaka, kompenzacije kašnjenja, razliku snimaka i slično
  - Fusion je najnoviji razvojni okvir koji objedinjuje prethodno razvijene Bolt i PUN sustave te ih i podržava.
  - Sustavi razvijeni u PUN-u i Boltu nastavit će raditi u okviru Fusion sustava
- Quantum je deterministički višekorisnički pogonski sustav za višekorisničke videoigre namijenjen za razvoj u Unity pogonskom sustavu te ga najčešće koriste strategijske igre
- Realtime je mrežni pogonski sustav namijenjen za umrežavanje različitih platformi (primjerice za mobilne i PC platforme).
  - Photon Chat i Voice pružaju podršku za pismenu, odnosno govornu komunikaciju između korisnika
  - Photon Server je još jedan dodatni paket koji nudi funkcionalnosti za uspostavu aplikacije na vlastitom poslužitelju uz korištenje od nekih spomenutih osnovnih paketa
  - Zbog jednostavnosti implementacije detaljnije obrađujemo Photon Unity Networking verzije 2 (PUN 2)

# Photon.Pun

- Osnova za mnoge igre koje koriste Photon je upravo Photon.Pun.
- Neke od značajki tj. funkcionalnosti koje podržava su:
  - Pozivi udaljenih procedura
  - Izvanmrežni (engl. offline) način rada
  - **Komponenta za sinkronizaciju mrežnih objekata (najvažnija komponenta – PhotonView)**
  - Podizanje mrežnih događaja
- Photon.Pun omogućava korištenje sučelja MonoBehaviourPunCallbacks, koje omogućava korištenje mnoštva metoda za spajanje korisnika u sobe i slično
- Omogućava korištenje glavne klase koja omogućava korištenje Photon paketa – **PhotonNetwork**
- Koristeći PhotonNetwork klasu korisnik dobiva pristup mnoštvu metoda

# Zadatak za iduće predavanje

- Pročitati sljedeći rad vezan za utjecaj kašnjenja na iskustvenu kvalitetu igara:
- <https://web.cs.wpi.edu/~claypool/papers/csgo-net-21/paper.pdf>