

# Síťová komunikace v počítačových hrách

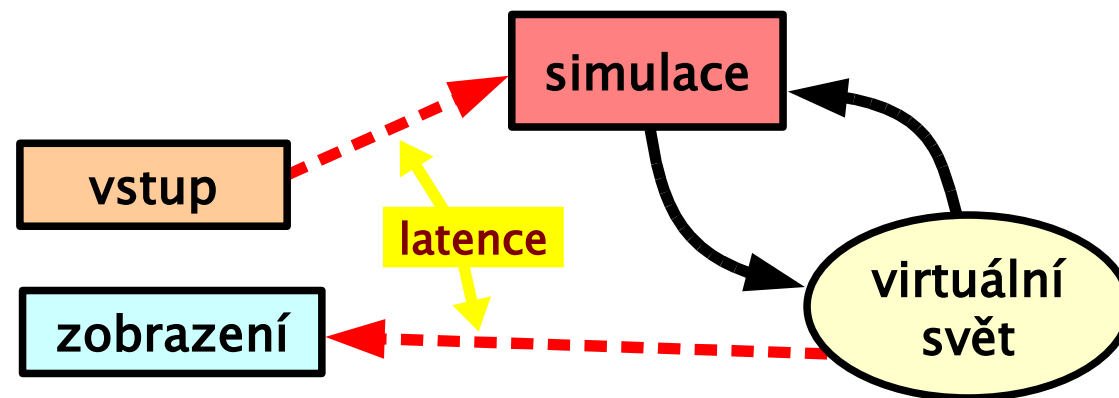
© 2009 Josef Pelikán, CGG MFF UK Praha

<http://cgg.mff.cuni.cz/~pepca/>  
[pepca@cgg.mff.cuni.cz](mailto:pepca@cgg.mff.cuni.cz)

# Obsah

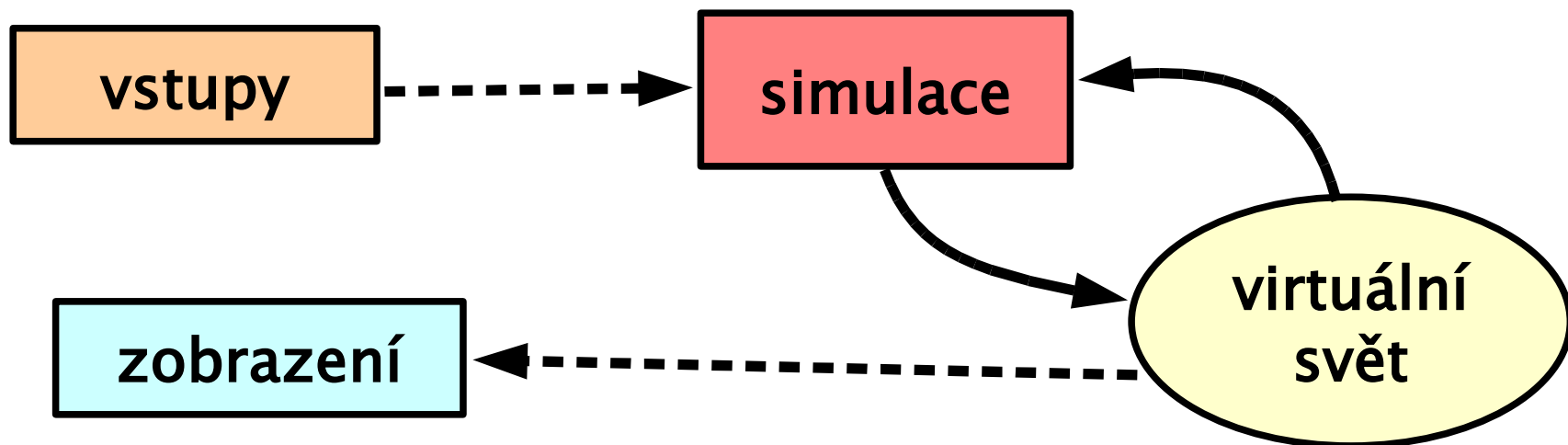
- ◆ logika návrhu síťové vrstvy (high-level)
  - ◆ server-client model vs. peer2peer
  - ◆ čas, identita, autorita, stav simulace, latence, ..
- ◆ spodní vrstvy síťové komunikace (low-level)
  - ◆ TCP/UDP, NAT, sockets, ..
- ◆ síťové knihovny (middleware)
- ◆ další aspekty síťového programování
  - ◆ podvádění (cheating), šifrování
  - ◆ kvalita fyzické vrstvy
  - ◆ časově kritické přenosy dat (Voice-over-Net)

# High-level architektura



# Simulační logika hry

- ♦ obecně („single-player“ i „multi-player“ hra):
  - ♦ **stav** simulovaného světa
  - ♦ **vstupy** od živých i „umělých“ hráčů
  - ♦ **simulace** světa (reakce na vstupy, [přírodní] zákony)
  - ♦ **zobrazení** výsledku (aktuálního stavu)



# Supertenký klient – server

- ♦ simulaci i „rendering“ dělá server
- ♦ klient jen sbírá vstupy a zobrazuje **bitmapu** výsledku
- ♦ technologicky velmi náročné (minimalizace lagu, komprese paketů..)

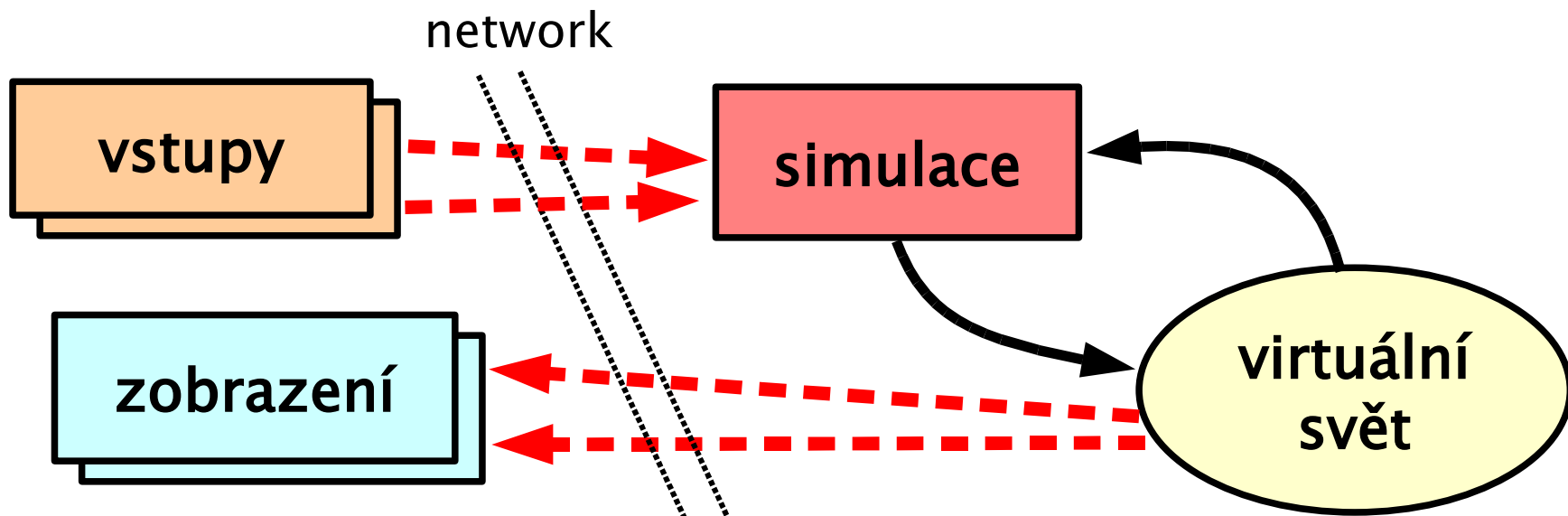
Up to 4 wireless controllers  
and 4 bluetooth™ headsets

OnLive (březen 2009)



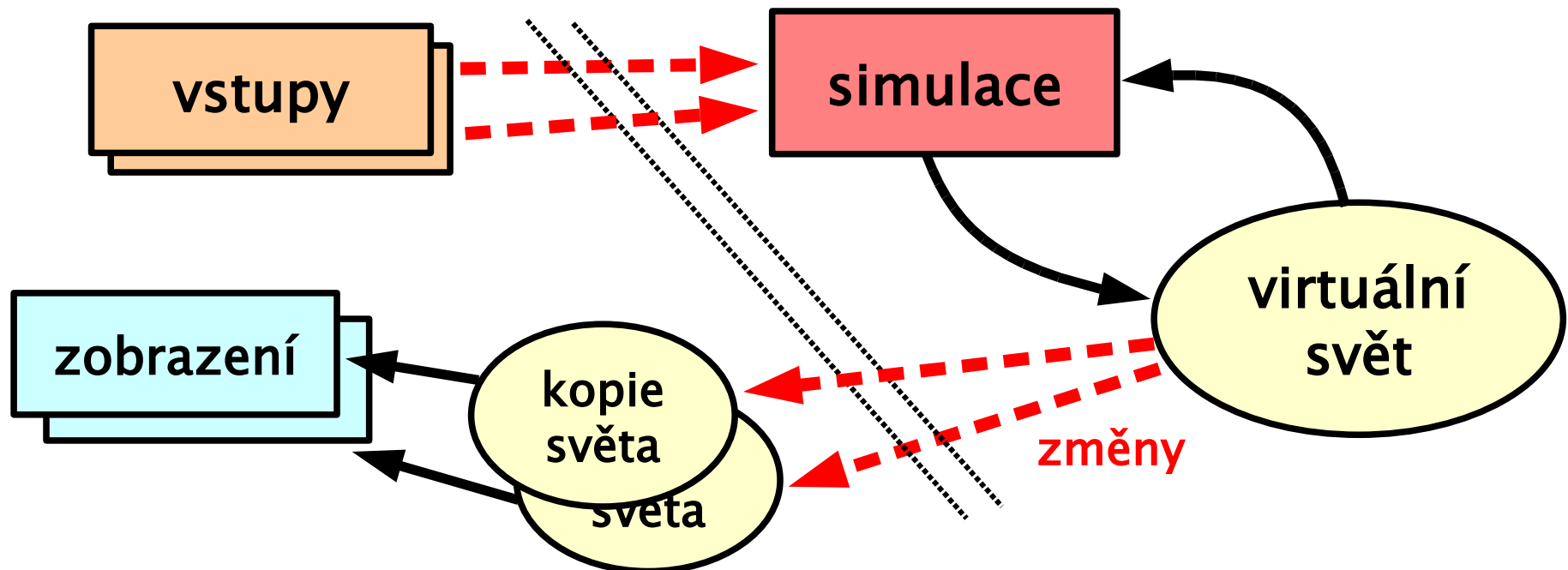
# Tenký klient – server

- ♦ simulace se odehrává na serveru
  - ♦ server má **jedinou instanci** stavu hry
  - ♦ klient jen sbírá vstupy a zobrazuje výsledek
  - ♦ textové hry typu MUD (multi-user dungeons)



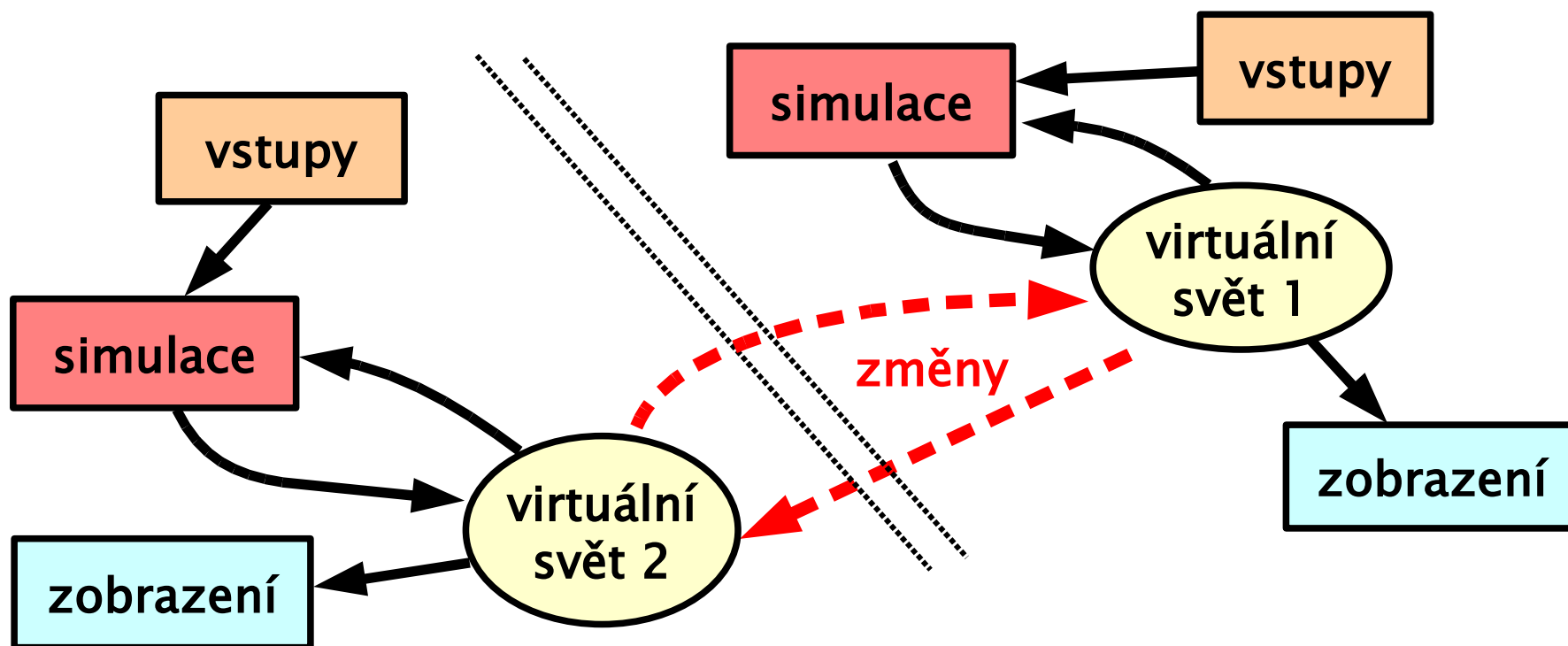
# Klient – server

- ♦ simulace se odehrává na serveru (**jediná autorita**)
  - ♦ i klienti mají své instance stavu světa
  - ♦ server posílá klientům **změny stavu** světa



# Peer-to-Peer

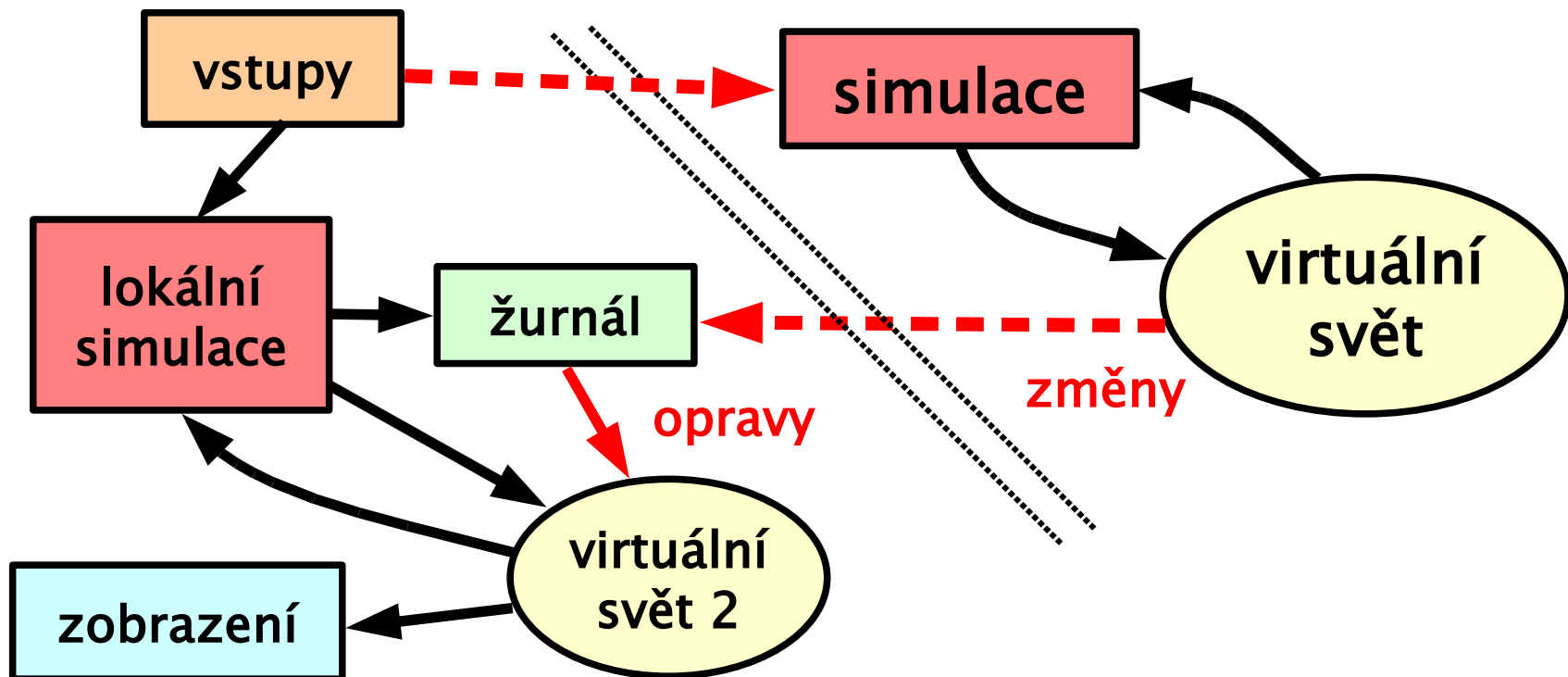
- každý klient má svou reprezentaci světa
  - každý je zodpovědný (je **autoritou**) za jistou část světa
  - navzájem si posílají **změny stavu** světa





# Hybridní architektura

- každý klient má **svou kopii** reprezentace světa
  - server je oficiální **autoritou**, ale klient predikuje vývoj
  - klient zpětně opravuje svoje data,...



# Síťové architektury

- předchozí členění bylo výhradně podle **simulační autority**
  - ❖ **autorita** (vlastník, zodpovědná entita) rozhoduje o všech důležitých změnách stavu virtuálního světa
  - aktuální **směrování paketů** již není tak důležité..
- **peer-to-peer simulace** může být síťově implementována jako **klient-server**, kde server:
  - ❖ shromažďuje pakety
  - ❖ akumuluje změny
  - ❖ rozesílá relevantní informace klientům
- příklady: Operation Flashpoint, ArMA



# Reprezentace světa

- ◆ společný **system souřadnic** (world-space)
- ◆ společný **čas** \*
  - ◆ neustále se musí lokální časy synchronizovat
- ◆ jednoznačné **identifikátory** objektů světa
  - ◆ pozor na lokálně vznikající entity..
- ◆ jednoznačně definovaná **autorita**
  - ◆ každý objekt je spravován **jedním správcem**
    - ostatní mohou jeho stav jen **predikovat**
  - ◆ musí být určeno, kdo dělá důležitá **rozhodnutí**
    - např. je-li cíl/hráč zasažen nebo ne

# Autority ve hrách OFP, ArmA

- ◆ autorita = „**vlastník objektu**“
  - ◆ klient vždy vlastní **svou postavu**
  - ◆ vlastníkem **AI skupiny** je **server**
- ◆ **přenos vlastnictví:**
  - ◆ vlastníkem **vozidla** je řidič
  - ◆ vlastníkem **podřízeného** ve skupině je vlastník velitele skupiny..
  - ◆ vlastníkem **střely** je vlastník toho, kdo vystřelil
- ◆ autorita rozhodující **zásah zbraní**
  - ➔ **vlastník střely** = ten, kdo vystřelil

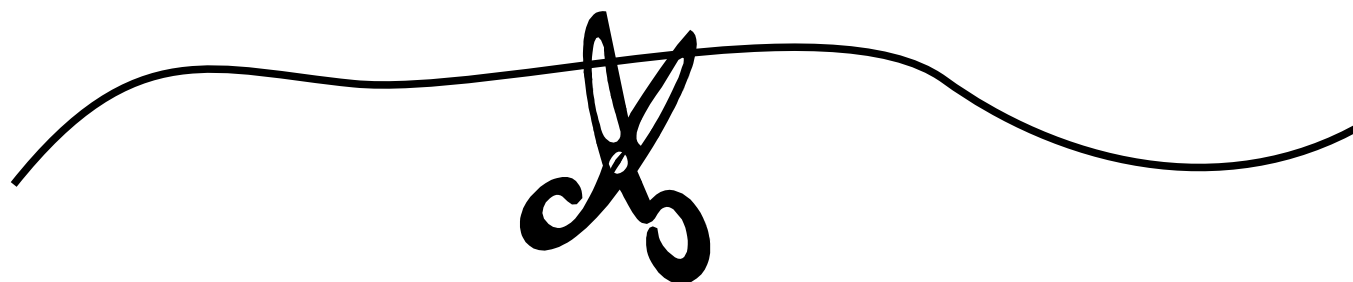


# Správa (multi-player) hry I

- ▶ každý hráč (klient) má jednoznačný **identifikátor**
  - ♦ může být odvozen z low-level síťových parametrů..
  - ♦ centrální autorita pro správu hry (hráčů)
- ▶ připojování hráčů **na začátku hry**
  - ♦ vznikají identifikátory hráčů
  - ♦ kontrolují se jejich primární data (distribuce, DVD..)
  - ♦ již během domlouvání: instant messaging nebo VoN
- ▶ připojování **uprostřed hry** (Join-in-Progress)
  - ♦ nutnost přenést stav světa (rozdíl proti DVD)

# Správa hry II

- ▶ **odpojování hráčů** od běžící hry
  - ♦ bezproblémové u architektur klient – server
  - ♦ nutné ošetřit u **peer2peer**
- ▶ odpojení z **technických příčin**
  - ♦ rozumné nastavení parametrů v low-level síťové knihovně
  - ♦ time-out: umožnit re-dial nebo re-boot síťového modemu
  - ♦ změna IP adresy (spolupráce s low-level?)



# Co posílat?

- ◆ každý klient má svoji **oblast zájmu**
  - ◆ poloha hráče ve virtuálním světě, směr pohledu..
  - ◆ pozor na dalekohled!
  
- ◆ **priority** jednotlivých typů změn
  1. zásah hráče, poškození, smrt, .. **primární cíle hry** ..
  2. změny v **zorném poli** hráče
  3. vzdálenější změny ve světě
    - mohou ovlivnit přesnost lokální simulace/predikce
  
- ◆ časově-kritická data (VoN = zvuk)
  - ◆ trvale vyhrazená část pásma ?

# Jak posílat?

- ◆ **úplná stavová informace** o objektu
  - ◆ snadné kódování, ale plýtvání pásmem
- ◆ **vybrané atributy** objektu
  - ◆ jen nejdůležitější atributy, ale musím je pojmenovat
  - ◆ bitové masky, předem připravené „úrovně“
- ◆ posílání **rozdílové informace**
  - ◆ musím vědět přesně, co ví protější strana.. (garantované zprávy)
- **cyklická obsluha objektů**
  - ◆ každý objekt přenesu jednou za delší čas (např. 200ms)



# Úsporné kódování dat

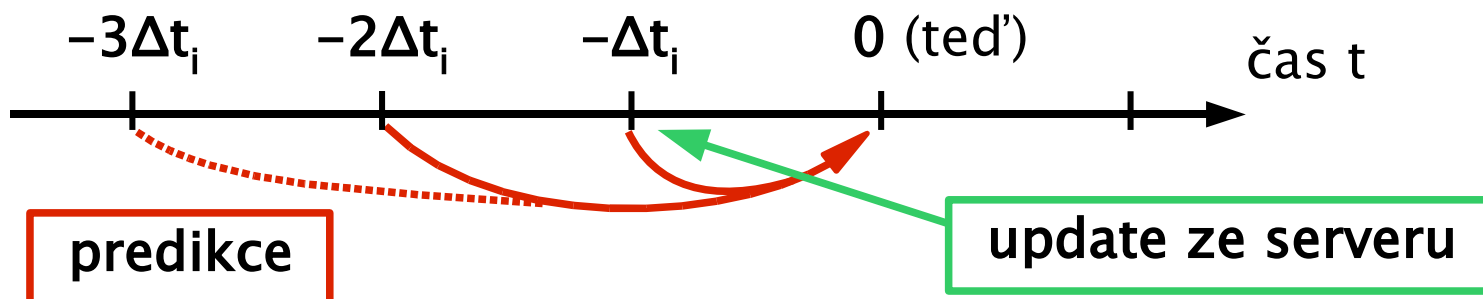
- ♦ síťové přenosy mají **ms** zpoždění, cykly CPU jsou **ns** !
  - ➡ vyplatí se obětovat pár taktů CPU na úspornější kódování
- ♦ příklad 1: **časový údaj** (musí být v každém paketu)
  - ♦ float: **32 bitů**
  - ♦ chytřejší: **16-bitový fixed-point**
    - 4 bity celá část (16 sec ... starší zprávy se stejně zahazují)
    - 12 bitů zlomky sekundy (přesnost ve stovkách  $\mu$ s)
- ♦ příklad 2: **orientace**
  - ♦ **kvaternion** (float[4]) je úspornější než matice orientace (float[9]) !

# Zpoždění síťového přenosu

- přenos zprávy pomocí low-level vrstev
  - **zpoždění**: SW (UDP stack) i vlastní síťové médium
  - síťová knihovna by měla poskytovat **odhad zpoždění** ke každému individuálnímu klientovi ( $\Delta t_i$ )
- „**ping-time**“, RTT (Round-Trip-Time)
  - cca *dvojnásobek* jednosměrného zpoždění
  - podle kvality síťového spojení může být mezi 1ms a ~2s
  - u nekvalitních připojení navíc *kolísá* (jitter)
- možnost **kompensace zpoždění** v simulacích

# Extrapolace

- ♦ **extrapolace změn** přicházejících ze serveru (peeru)
  - ♦ musím znát odhad svého síťového zpoždění  $\Delta t_i$ 
    - já se dozvím o globálních změnách o  $\Delta t_i$  později
    - server (ostatní) se dozví o mých změnách o  $\Delta t_i$  později
  - ♦ uchovávám několik **starších stavů** světa
    - ♦ historie stará  $\Delta t_i$  (predikce jen podle rychlosti) až  $2-3\Delta t_i$  (odhady založené na zrychlení)



# Kompenzace zpoždění I


- ◆ pohyb **vlastního hráče** (avatara) ve hrách klient-srv.
  - ◆ není únosné čekat  $RTT + \tau$  na potvrzení od serveru
  - ◆ nutná *okamžitá odezva* na uživatelský vstup
- ◆ **oprava** podle (autoritativního) stavu ze serveru
  - ➔ opravím položku historie + *přepočítám predikci*
  - ➔ *velké rozpory*  $\Rightarrow$  poskakování (**snapping**)
    - ➔ mohou mírnit **exponenciálním průměrováním**
  - ➔ snapping je nevyhnutelný, pokud se v simulaci používají data neznámá klientovi
    - ➔ výbuch neznámé nálože, interakce se soupeřem ..

# Kompenzace zpoždění II

- ◆ **zjednodušení** sdíleného kódu – přenesení některých zodpovědností na klienta
  - ◆ jen *důvěryhodné* prostředí (letecký, vojenský simulátor)
- ◆ míření, **střelba**
  - pozici *cizích hráčů* (AI) nemohu tak přesně předvídat

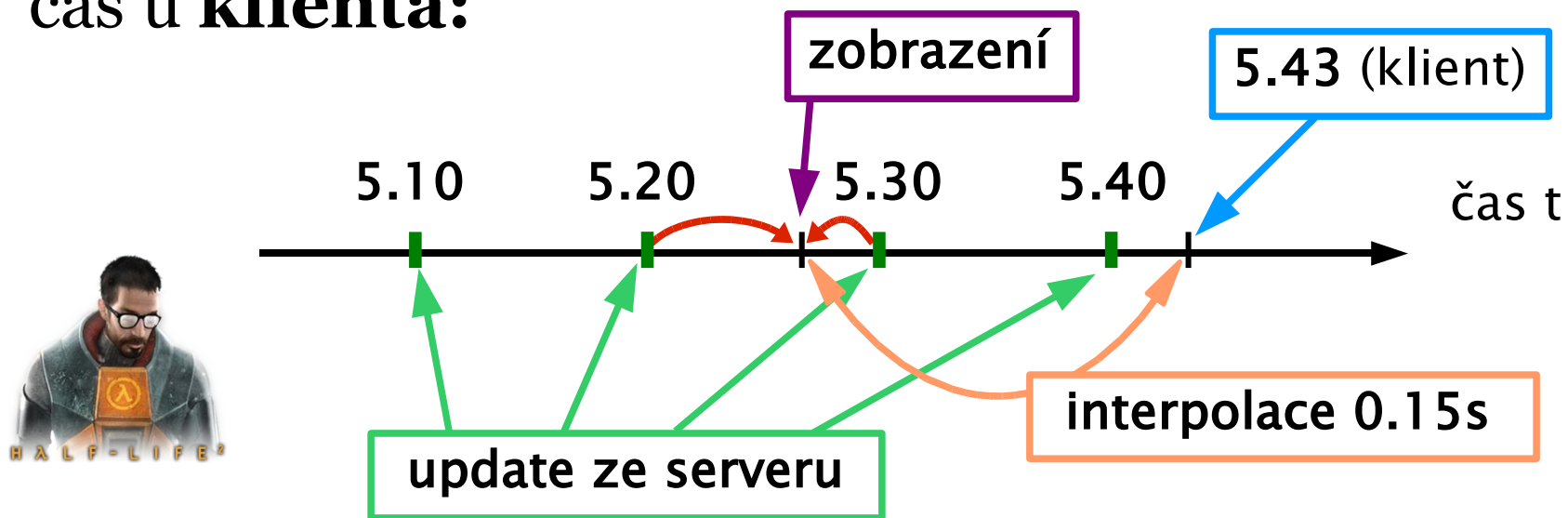


# Interpolace místo extrapolace

- ◆ **klientská predikce soupeře** může být nespolehlivá
  - ◆ „**first-person**“ střílečky
    - je důležité spolehlivě mířit a střílet 
  - ◆ neomezené zrychlení, síla a dynamika, rychlé úskoky
- ◆ posunutí vykreslovacího času **do minulosti**
  - ➔ **interpoluji** mezi dvěma potvrzenými stavy
  - ◆ zavádím tím **zpoždění navíc** !  
.. ale zobrazuji soupeře tam, kde **skutečně byl**
- **kompence zpoždění klientů na serveru !**

# Interpolace a kompenzace

## čas u klienta:



- server musí při simulaci „vracet zpátky čas“ a simulovat důležité akce **pro každého hráče zvlášť**
  - střelba (potvrzení zásahu)

# Implicitní vs. dedikovaný server

## ◆ implicitní server

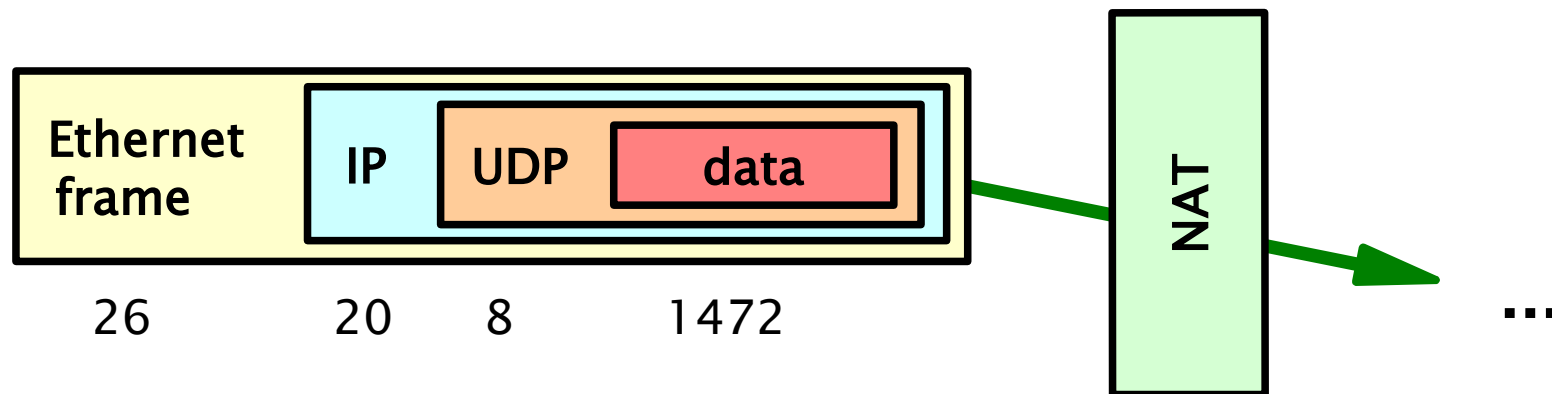
- ◆ pustí se na stroji jednoho z hráčů
- ◆ konzumuje **výkon CPU** a **síťové pásmo**
- ◆ je lepší, když hostitel není za firewallem (NAT)

## ◆ dedikovaný server

- ◆ **zvláštní binárka** běžící na **zvláštním stroji** (veřejný herní server)
- ◆ nemusí být vůbec přítomny grafické a UI vrstvy
- ◆ musí zůstat: herní data, síť, simulace, fyzika, AI, ..
- ◆ může běžet pod **Linuxem** (server-hosting friendly), i když původní klient používá Direct3D ..



# Low-level architektura



# Low-level síťové vrstvy

## ● TCP

- spolehlivý transparentní přenos dat à la „stream“
- spojovaný protokol (navázané spojení, 2 sockety)
- ➕ **nehodí se** pro realtime hry !



Quake 1, Carmack, 1995

## ● UDP

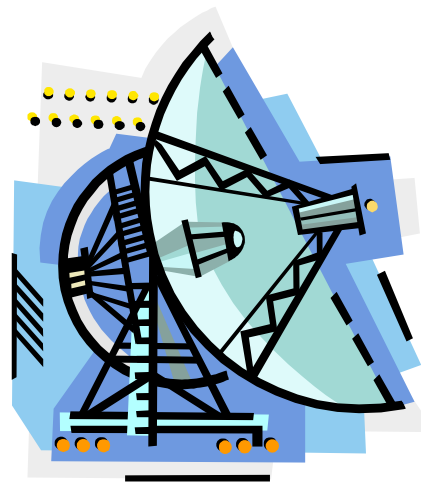
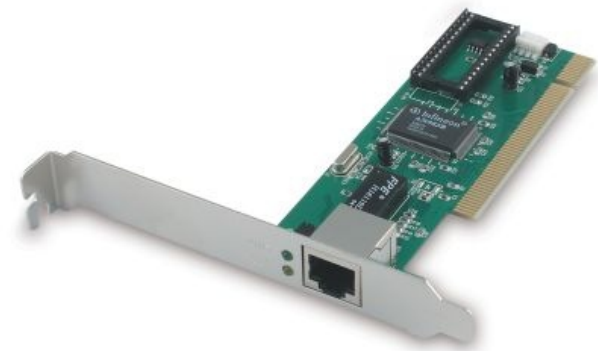
- efektivnější přenos, ale nespolehlivý:  
není zaručeno **doručení** paketů ani jejich **pořadí**
- nespojovaný (bezestavový) protokol
- ➕ **spolehlivé doručování** (potvrzování) si musíme implementovat sami !

# UDP/IP protokol

- ◆ posílá/přijímá se **jednotlivá zpráva** (datagram)
- ▶ komunikace probíhá přes **socket** (BSD sockets, viz)
  - ◆ **vysílání**: posílám do socketu jednotlivé datagramy
    - mohu specifikovat **adresu příjemce** nebo **broadcast**
  - ◆ **příjem**: socket přijímá datagramy (recvfrom, select)
    - každý přijatý datagram obsahuje **adresu odesilatele..**
- ▶ **maximální velikost** datagramu je omezena (**MTU**)
- ▶ UDP hlavička je dlouhá jen 8 bytů  
(celkem včetně IPv4 hlavičky 28 bytů)

# Příklady MTU

◆	dial-up:	576
◆	AOL DSL:	1400
◆	PPPoE:	1492
◆	Ethernet:	<b>1500</b>
◆	WiFi:	2304
◆	FDDI:	4479



# Adresování

## ● IP adresa ↔ DNS jméno

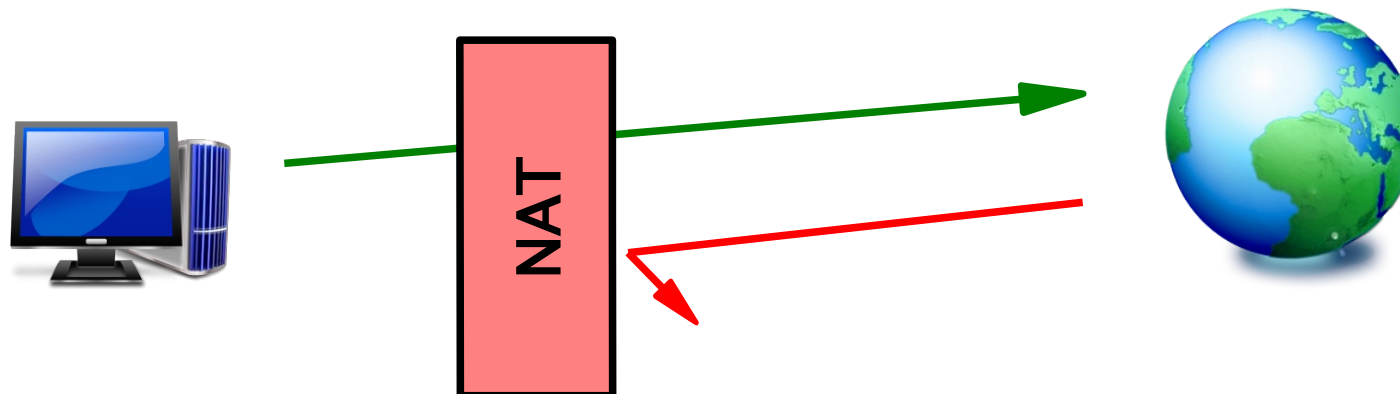
- gethostbyname(), gethostbyaddr()
- IP verze 4 (potřebuje NAT), IP verze 6
- network-endian = big-endian !

## ● UDP port

- slouží primárně ke směrování příchozích paketů jednotlivým zaregistrovaným **službám / aplikacím**
- hry používají dočasnou (dynamickou) registraci portu
  - čísla portů mezi 49152 a 65535
  - strategie výběru neobsazených portů ..

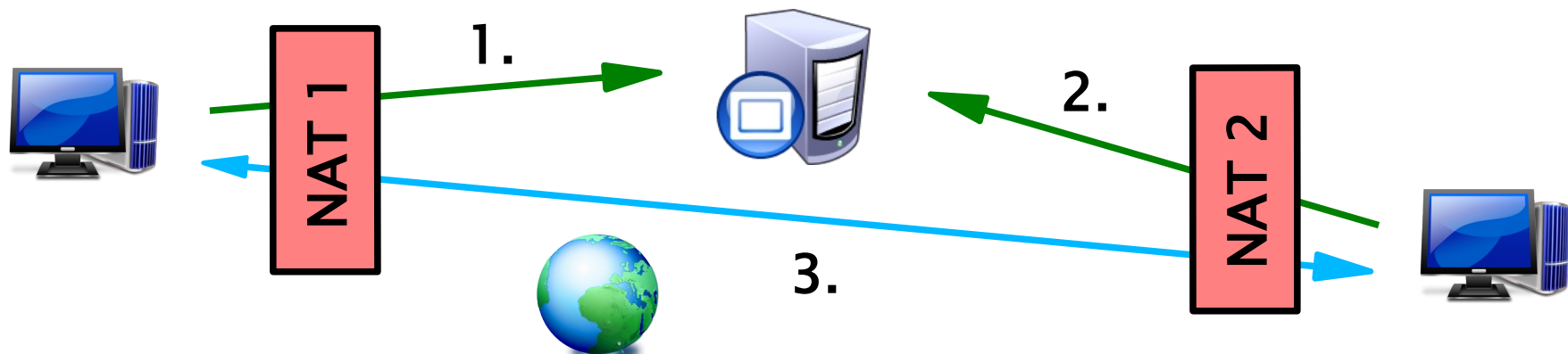
# Překlad adres – NAT

- ◆ **realita** dnešního připojení k Internetu
  - ◆ doma i ve firmách (symetrický NAT)
- ◆ neumožňuje jednoduše **navázat spojení** z veřejného Internetu do privátní sítě
  - ◆ OK, je-li iniciátorem lokální počítač



# Obcházení NATu

- ♦ oficiální postupy: **STUN**, TURN
  - ♦ veřejný server, přes který se minimálně spojení **navazuje**
  - ♦ tyto postupy potřebuje i **VoIP**, implementují middlewares
- ♦ **UDP punching**
  - ♦ klienti naváží spojení s veřejným serverem (NAT tabulky)
  - ♦ pak se spojení **předá** (naděje, že tabulky zůstanou)

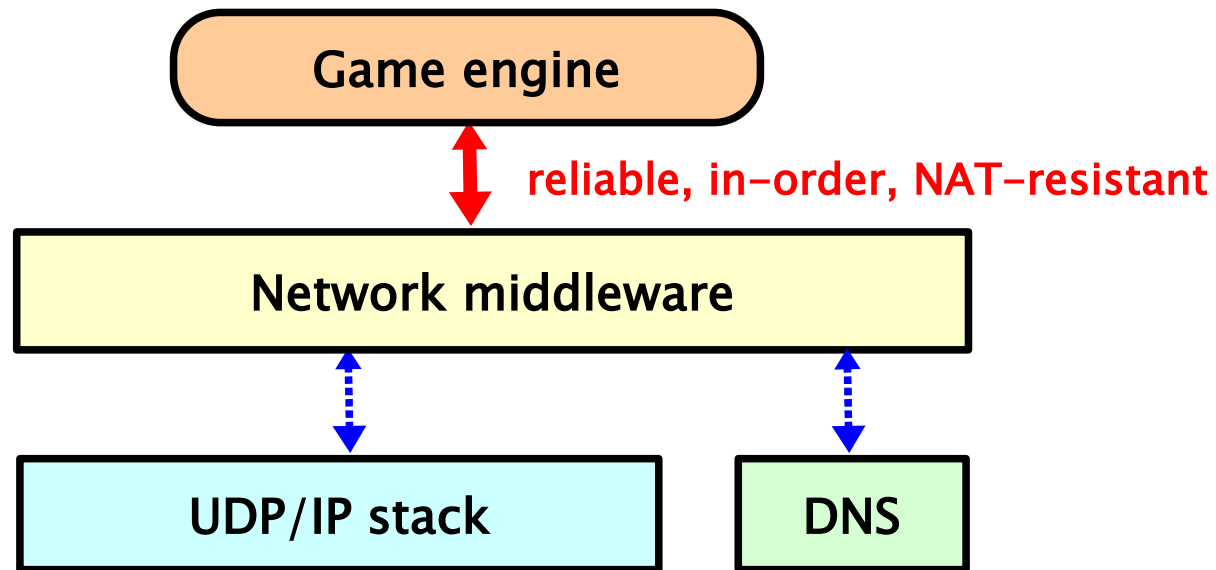


# BSD Sockets API

- přenositelné API pro obsluhu TCP/**UDP**/IP stacku
  - ◆ varianta pro Windows: **Winsock** (dnes verze 2.2.x)
- ▶ **socket()** handle, přes který se komunikuje
- ▶ **bind()** spojení s lokálním portem (je-li volný..)
- ▶ **sendto()** odeslání datagramu
- ▶ **recvfrom()** příjem datagramu
- ▶ **select(), poll()** paralelní čekání na více socketů
- možnost „připojit“ UDP socket
  - pevná adresa protějšší strany
  - nemusí se používat volání s explicitní adresou



# Middleware pro síť



# Funkce

- ◆ **inicializace** a udržení „spojení“
  - ◆ DNS resolving, překonání NAT, heart-beats, ..
- ◆ **spolehlivý přenos** zpráv
  - ◆ na žádost aplikace (u vybraných zpráv)
  - ◆ dva možné typy „spolehlivé“ zprávy
    - garantovaná
    - garantovaná v pořadí (odkaz na předchůdce?)
- ◆ posílání **dlouhých zpráv**
  - ◆ transparentně pro aplikační vrstvu
- ◆ časově kritické přenosy
  - ◆ multi-mediální streaming (VoN)

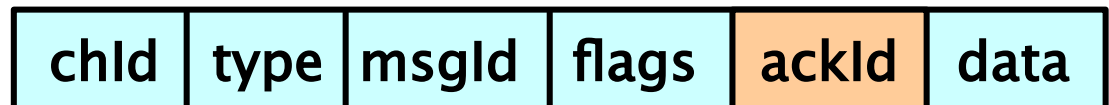
# Přenosový kanál

- ◆ spojení k protějším klientovi/serveru
  - ◆ založení, udržování, uzavření
- ◆ **informace** pro aplikaci:
  - ◆ **MTU** bez fragmentace
  - ◆ odhad **RTT** nebo zpoždění  $\Delta t$
  - ◆ **kdy** naposledy přišel paket?
  - ◆ které zprávy byly **potvrzeny**?
    - a které čekají na příp. přeposlání..
- ◆ možnost provádět **demultiplexing** v knihovně
  - ◆ společný přijímací socket
  - ◆ rozdělování zpráv podle *kanálů*, *typů* (bit-prefix), ..

# Potvrzování zpráv

## ◆ à la TCP

- ◆ posílání zvláštních potvrzovacích paketů



## ◆ **líné potvrzování**

- ◆ využívá přirozené obousměrné komunikace
- ◆ každý datagram obsahuje malou **potvrzovací sekci**
- ◆ jistá redundance potvrzování, v nouzi „heart-beats“

## ◆ **přeposílání** garantovaných zpráv

- čas je odvozen z RTT
- **zahlcení kanálu** (hromadí se zprávy) ⇒ panika

# Udržování spojení

- ◆ UDP spojení by se mělo „**udržovat**“ (nejen kvůli některým aktivním síťovým prvkům)
  - ◆ „... čas od času se pošle paket, i když to aplikační vrstva nepotřebuje ..“
  - ➔ NAT tabulky zůstanou aktivní
  - ➔ přeměří se RTT
  - ➔lepší se informace o doručených zprávách
- ◆ „**heart-beat**“ paket
  - ◆ bez uživatelských dat
  - ◆ může obsahovat rozšířenou **potvrzovací informaci**
  - ◆ maximální **priorita** (měření RTT)

# Časově–kritické přenosy (multimedia)

- ◆ zprávy mají **vysokou prioritu** (mohou předbíhat), ale nejsou garantované
  - ◆ zvláštní vyhražená část pásma ?
  - ◆ zvláštní odesílací fronta ? **P2P** ?
- ◆ **Voice-over-Net (VoN)**
  - ◆ stream rozdělen na nezávislé „frames“ (10-60 ms)
  - ◆ synchronní přehrávání, buffering
  - ◆ **hlasové kodeky** s vysokou účinností
    - založené na CELP (Code-Excited Linear Prediction)
    - dynamická změna kvality ?

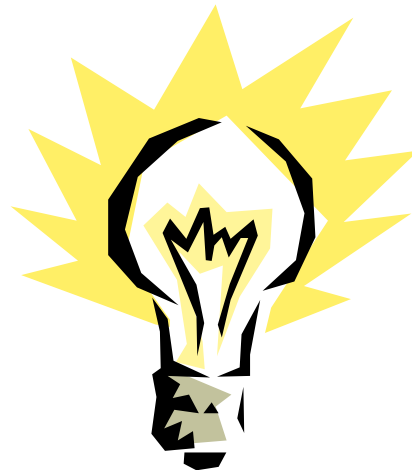


# Middleware knihovny

- ◆ **Open TNL** (Torque Network Library)
  - ◆ původně součástí Torque Game Engine
  - ◆ C++, open-source, GPL i komerční licence
  - ◆ používá UDP, zprávy i RPC, šifrování
  - ◆ typy zpráv: negarant., garant., garant. v pořadí, rychlé
- ◆ **ENet**
  - ◆ původně pro střílečku „Cube“ (Lee Salzman)
  - ◆ C, open-source
  - ◆ používá UDP, zprávy
  - ◆ typy zpráv: negarant., garant., garant. v pořadí
  - ◆ řízení propustnosti kanálu (throttle)

---

# Tipy a triky





# Podvádění (cheating)

- ◆ obzvlášť hrozí u akčních stříleček
- ◆ **aktivní podvody**
  - ◆ modifikace kódu klienta
  - ◆ **nekonečně** životů/zdraví, munice, zbraní, ..
  - ◆ úprava algoritmů (míření, zásahy, vyhledávání nepřátel..)
- ◆ **pasivní podvody**
  - ◆ odchytávač paketů (**sniffer**) je dekóduje a napovídá hráči
  - ◆ na jiném počítači ⇒ nedetekovatelný
  - ◆ dost pomůže již dekódování polohy protihráčů
  - ◆ obrana: šifrování stavových informací ..



# Protiopatření, šifrování

- ◆ ochrana binárního kódu aplikace (např. SHA-1)
  - ◆ nejde udělat dokonale (podvodný program má celou originální binárku k dispozici)
  - ◆ nelze u open-source
- ◆ **šifrování paketů**
  - ◆ asymetrické šifrování (i proti pozměňování dat)
  - ◆ klíče se mohou přidělovat dynamicky, zabudování MAC adresy..
- ◆ **co se nešifruje:**
  - ◆ multimediální přenosy (VoN)
  - ◆ data, jejichž znalost by nikomu nepomohla



# Voice-over-Net (VoN)

- ◆ **časově velmi citlivá** komponenta
  - ◆ výpadky zvuku jsou dost rušivé
  - ➔ real-time techniky („clock-driven replay“)
- ◆ **buffering** na straně příjemce
  - ◆ rezerva kompenzující jitter a změnu pořadí paketů
  - ◆ **celková latence** však nesmí být moc velká (Armstrong)
- ◆ **efektivní hlasové kodeky**
  - ◆ všechny důležité jsou založeny na CELP (1985)
  - ◆ **Speex** od 2 kbps (open-source, umí VBR)
  - ◆ ACELP: EFR, **AMR** od 4.8 kbps, **G.729**: 6.4-8-11.8 kbps



# Kvalita síťového spojení

- ◆ **latence** (delay, RTT) **v ms**
  - ◆ neměla by příliš kolísat (→ parametry síťové knihovny)
  - ◆ *rychlost světla* ! (teorie bez routerů:  $RTT < 133ms$ )
- ◆ **ztrátovost** (packet loss) **v procentech**
  - ◆ UDP nezaručuje doručení datagramu
  - ◆ příliš velká ⇒ až panika síťového kanálu
- ◆ **„těžko na cvičišti ...“**
  - ◆ testovat síťový provoz na *výstředních zapojeních*
    - 1Gbit Ethernet, ADSL 8Mbit proti modemu V.90 (56Kbit)
  - ◆ počítat spíš s horšími podmínkami (ale umět využít dobré)
  - ◆ **emulátor nekvalitní sítě**

# Technické poznámky

- ◆ **určení MTU** pro konkrétní spojení
  - ◆ dnes se již málo používá dial-up (MTU < 600 byte)
  - ◆ rozumné počáteční nastavení = **1400 byte**
    - při fragmentaci postupně zmenšují („pokus-omyl“)
- ◆ **kolik portů** používat pro UDP příjem?
  - ◆ více portů ⇒ mohu zvýšit kapacitu (více bufferů)
  - ◆ méně portů ⇒ menší režie v OS, snadněji se hledají
- ◆ **přijímající smyčka** v middleware
  - ◆ používat **select()** nebo **poll()**
  - ◆ **zvláštní vlákno** (synchronizace) nebo zabudování do simulační smyčky ?

- ◆ ***Good multiplayer game programming tutorials?***  
<http://forums.indiegamer.com/showthread.php?t=8705>
- ◆ ***Multiplayer and Networking,***  
<http://www.gamedev.net/reference/list.asp?categoryid=30>
- ◆ ***Introduction to Multiplayer Game Programming,***  
Brian Hook,  
<http://trac.bookofhook.com/bookofhook/trac.cgi/wiki/IntroductionToMultiplayerGameProgramming>
- ◆ ***The Quake3 Networking Model,*** Brian Hook,  
<http://trac.bookofhook.com/bookofhook/trac.cgi/wiki/Quake3Networking>

- ◆ ***Latency Compensating Methods in Client/Server In-game Protocol Design and Optimisation***, Yahn W. Bernier, Valve Software
- ◆ ***Source Multiplayer Networking***,  
[http://developer.valvesoftware.com/wiki/Source\\_Multiplayer\\_Networking](http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking)
- ◆ ***Beej's Guide to Network Programming***, Using Internet Sockets, <http://beej.us/guide/bgnet/>
- ◆ ***STUN - Simple Traversal of UDP Through NATs***, RFC 3489

# Zdroje

- ◆ ***OpenTNL Library***, <http://www.opentnl.org/>
- ◆ ***ENet Library***, <http://enet.bespin.org/>
- ◆ ***Speex: A Free Codec For Free Speech***,  
<http://speex.org/>
- ◆ ***Massively Multiplayer Game Development 1 & 2***,  
Thor Alexander, Charles River Media, 2003 & 2005

