Katedra softwarového inženýrství,
Matematicko-fyzikální fakulta,
Univerzita Karlova, Praha

Rodina protokolů TCP/IP,
verze 2.3

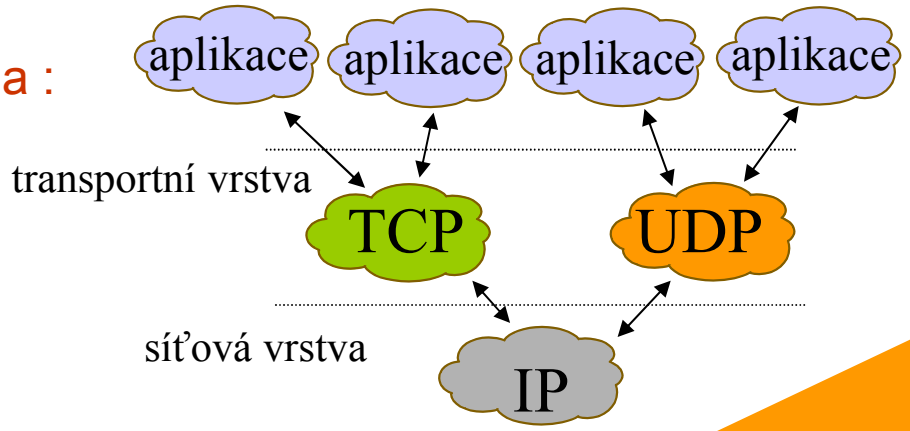
Část 7: Transportní protokoly

Jiří Peterka, 2006

role transportní vrstvy

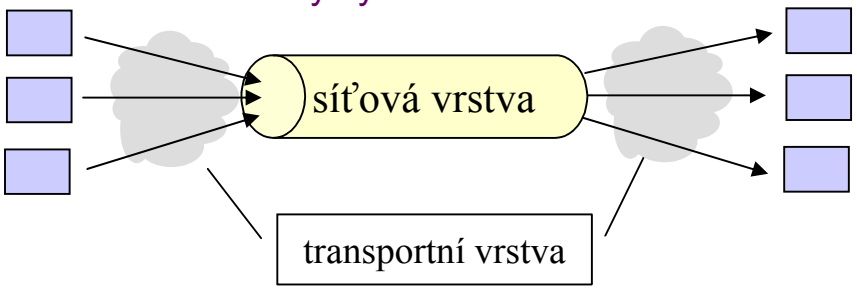
- obecně:
  - přizpůsobuje možnosti nižších vrstev požadavkům vyšších vrstev
  - požadavky se mohou týkat:
    - spojovaného/nespojovaného charakteru komunikace
    - spolehlivosti
    - kvality služeb (QoS)
- v TCP/IP:
  - přenosové mechanismy síťové vrstvy (protokol IP) jsou nespolehlivé a nespojované, bez podpory QoS
  - transportní vrstva řeší požadavky na :
    - spojovanou komunikaci
    - spolehlivost
  - transportní vrstva (dosud) neřeší:
    - požadavky aplikací na QoS

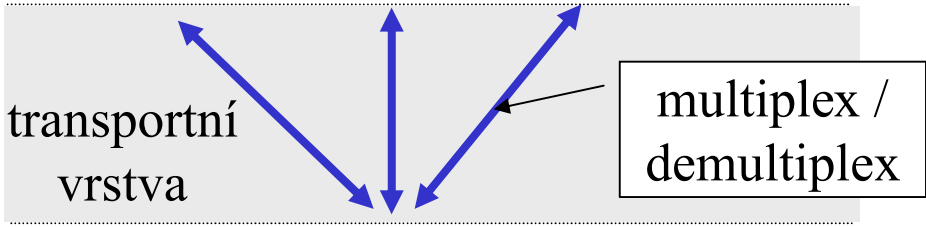
- princip řešení v TCP/IP:
  - podpora spojovanosti a spolehlivosti je volitelná, aplikace si mohou svobodně vybrat
  - existují dva transportní protokoly:
    - **TCP** – funguje spojovaně a spolehlivě
      - mění to, co nabízí protokol IP
    - **UDP** – funguje nespojovaně a nespolehlivě
      - nemění to, co nabízí protokol IP



další úkol transportní vrstvy

- síťová vrstva:
  - dívá se na každý uzel jako celek
    - síťové adresy (např. IP adresy) označují uzly jako takové
      - přesněji: jejich rozhraní
    - nerozlišují konkrétní entity v rámci uzlů
      - zejména na aplikační úrovni
- transportní vrstva:
  - má za úkol rozlišovat různé entity (procesy, úlohy, démony, …) na úrovni vyšších vrstev
  - musí provádět tzv. **multiplex**
    - "sběr" dat od více entit vyšších vrstev a jejich další přenos
  - a tzv. **demultiplex**
    - rozdělování přijatých dat mezi různé entity vyšších vrstev

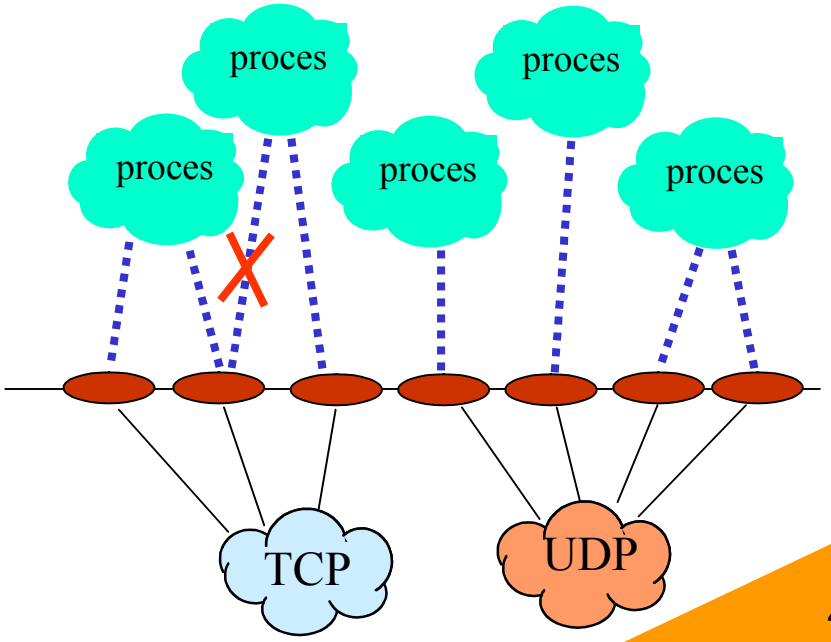




porty, čísla portů

- entity aplikační vrstvy nejsou identifikovány přímo
  - ale pouze nepřímo, prostřednictvím tzv. portů
- port
  - je přechodovým bodem mezi transportní a aplikační vrstvou
  - je identifikován **číslem**
    - to představuje relativní adresu v rámci uzlu
  - port si lze představit jako (obousměrnou) datovou strukturu typu fronta
    - z jedné strany se do ní vkládají data, z druhé strany odebírají
  - porty existují apriorně
    - nevznikají ani nezanikají, pouze se může měnit jejich využití (přidělení)

- entity aplikační vrstvy (procesy, démoni, úlohy, …) se dynamicky "**připojují**" (**asociují**) k portům …..
  - jedna entita může být asociována s více porty
  - s jedním portem NESMÍ být asociováno více entit



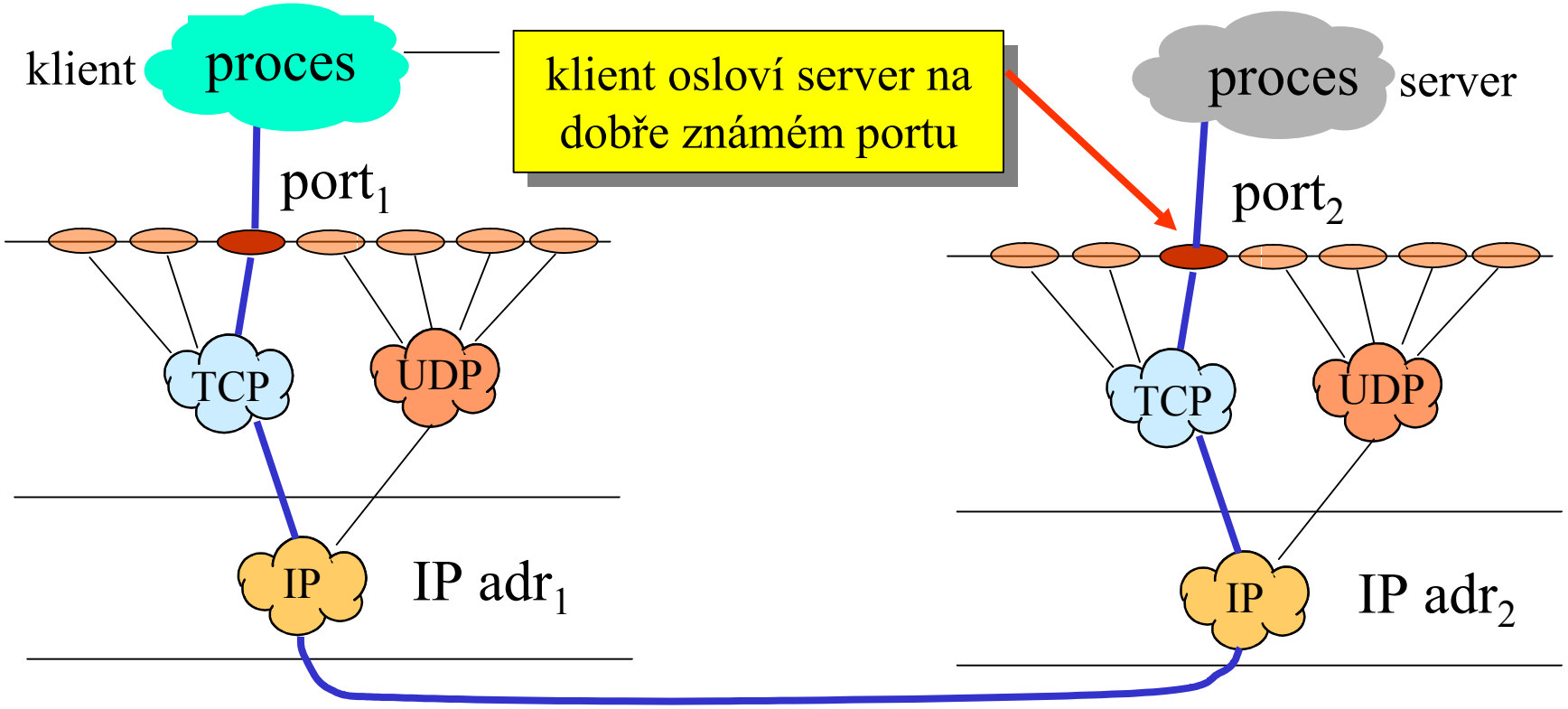
dobře známé porty

- význam některých portů je fixován
  - je apriorně dán (přidělen)
    - přiděluje IANA (ICANN)
  - je to všeobecně známo
    - dříve se zveřejňovalo v RFC (naposledy RFC 1700)
    - dnes pouze on-line, na adrese http://www.iana.org/assignments/port-numbers
- jde o tzv. **dobře známé porty** (well-known ports)
  - jsou to porty 0-1023
  - **smysl: na těchto portech jsou poskytovány služby**
- existují též tzv. registrované porty
  - porty 1024-49151
  - IANA nepřiděluje, pouze registruje jejich použití
- ostatní porty (Dynamic, Private)
  - 49152 - 65535
  - jsou používány volně, nejsou ani registrovány

Port #	Popis
21	FTP
23	Telnet
25	SMTP
69	TFTP
70	Gopher
80	HTTP
88	Kerberos
110	POP3
119	NNTP
143	IMAP
161	SNMP

Port #	Popis
1433	MS SQL
1527	ORACLE
6000-63	X Window

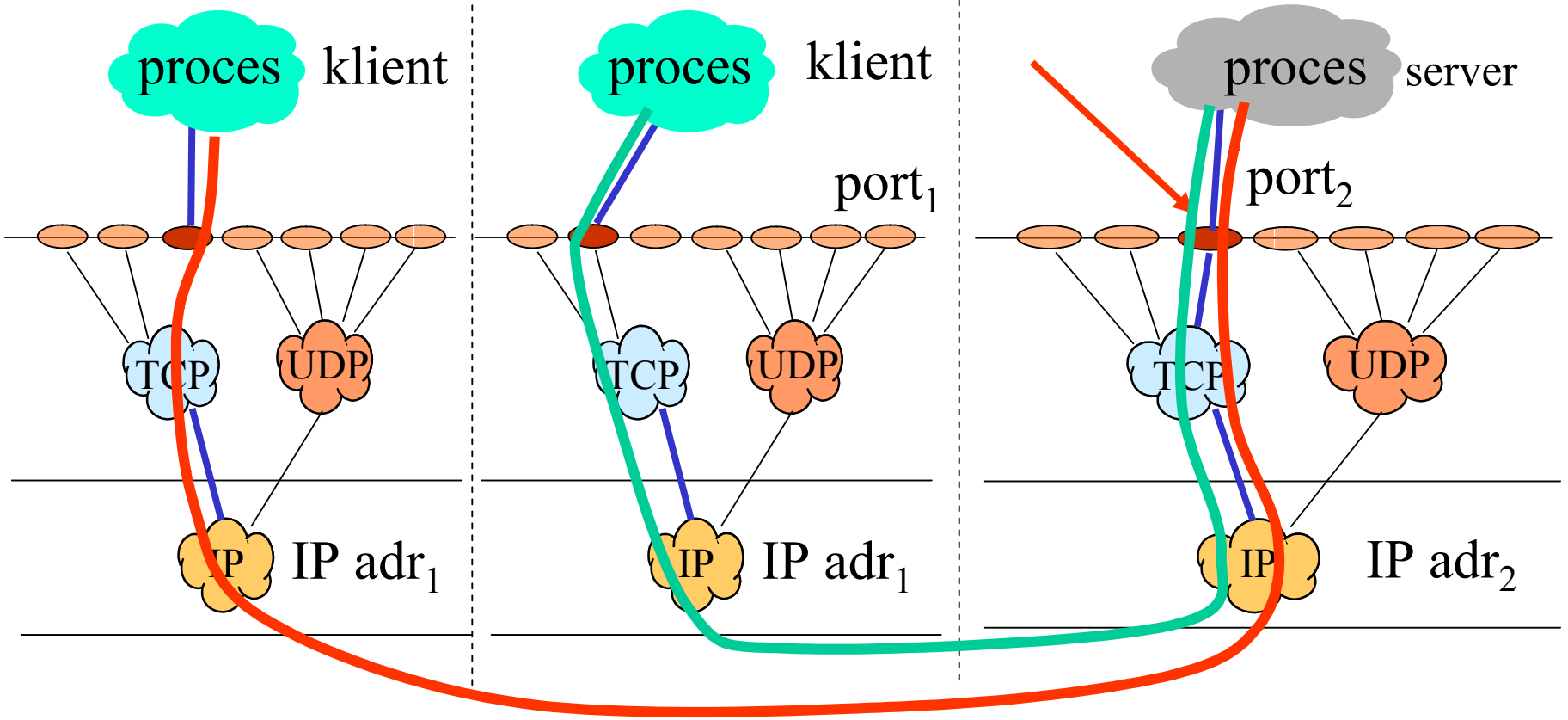
představa (aplikačního) spojení



- port je "logickým zakončením" spojení (entita je "fyzickým zakončením")
- (aplikační) spojení je jednoznačně určeno pěticí

(transportní protokol, IP adr_1, $port_1$, IP adr_2, $port_2$)

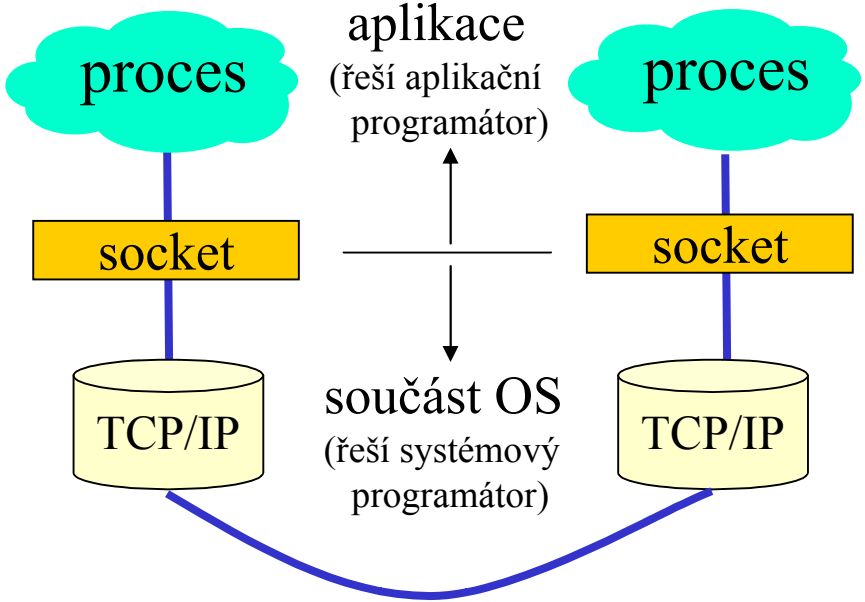
představa (aplikačního) spojení



- "na" stejný uzel a port (např. na port 80 WWW serveru) může být vedeno více spojení
  - přesto je lze rozlišit podle originující IP adresy a portu

porty vs. sockety

- porty jsou logickou záležitostí
  - na všech platformách jsou stejné
    - identifikované svými čísly
  - jejich konkrétní implementace je závislá na platformě
- aplikace (entity aplikační vrstvy) obvykle pracují s porty skrze API
  - API může být součástí operačního systému, nebo může mít formu knihoven linkovaných k aplikaci



- podstatný je také "styl" práce s porty
  - dnes převažuje "styl" (paradigma) zavedený v BSD Unixu (verze 4.2), založený na tzv. socketech
- socket vznikl jako abstrakce souboru v BSD Unixu
  - pro potřeby práce se soubory (a také pro vstupu a výstupy)
  - pracuje se s ním style "open-read-write-close"
- sockety byly použity i pro potřeby síťování
  - byly rozšířeny o další možnosti/operace
- "socketové API"
  - takové API, které procesům vytváří iluzi že pracují se sockety
    - např. rozhraní WINSOCK
- socket si lze představit jako analogii brány
  - vedoucí k síťovým službám

socket =

práce se sockety

- sockety existují nezávisle na portech
- vznik socketu:
  - voláním funkce pro vytvoření nového socketu: **SOCKET (…)**
    - parametry:
      - rodina protokolů (TCP/IP)
      - typ služby (STREAM, DATAGRAM, RAW)
      - protokol (TCP, UDP)
  - nově vytvořený socket není asociován (sdružen) s žádným portem
    - **vzniká "sám o sobě"**
- asociování socketu s konkrétním portem
  - voláním funkce: **BIND (…)**
- po skončení práce se socketem je nutné jej zavřít/zrušit
  - **CLOSE(...)**

- se sockety lze provádět další "primitivní operace"
  - SENDTO(socket,data,…,adresa ….)
    - pošle data zdanému příjemci
    - určeno pro nespojovaný způsob komunikace, bez navazování spojení
  - RECVFROM(socket,…,adresa, …)
    - přijme data nespojovaným způsobem

příklad: činnost serveru při nespojované komunikaci:

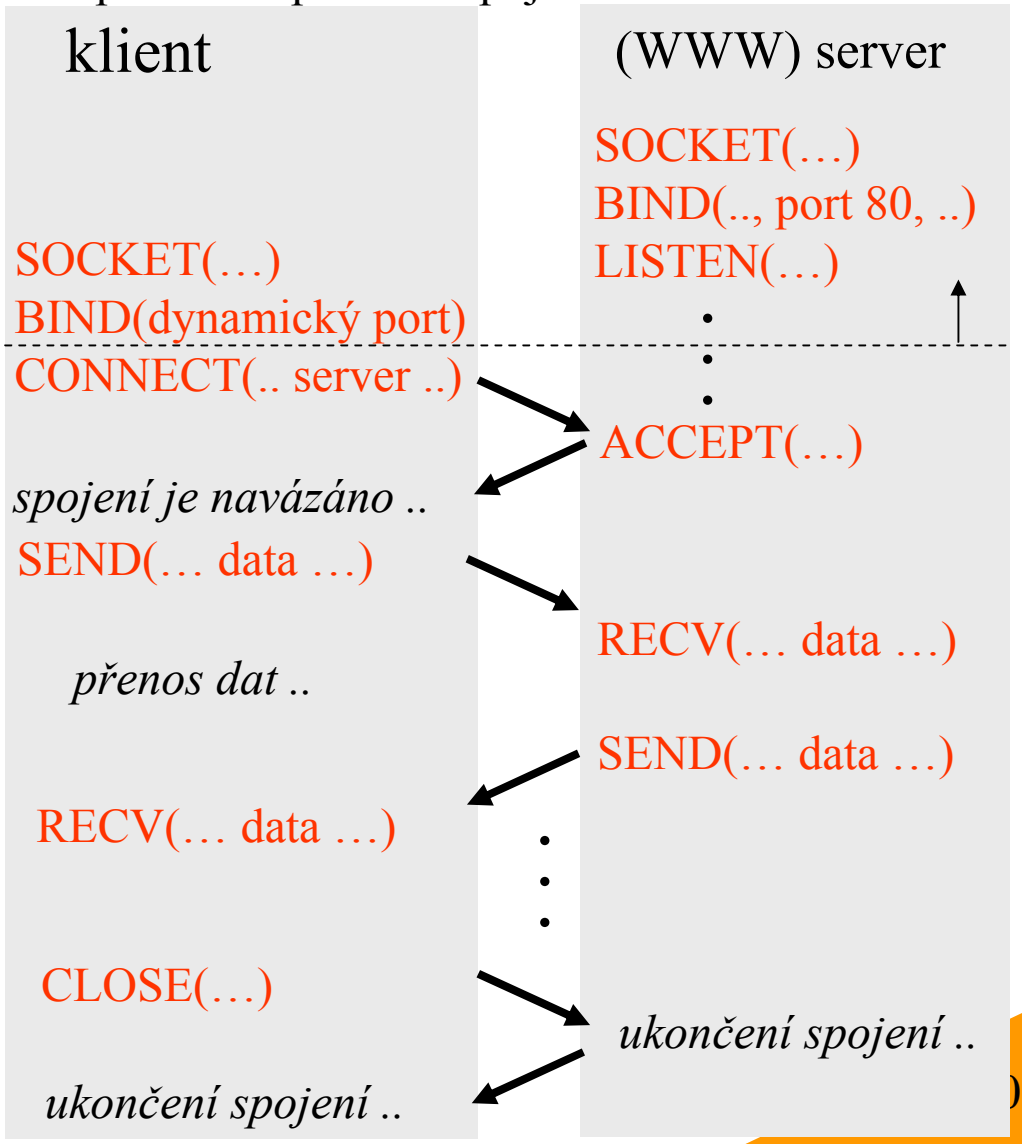
```
newsock=SOCKET(…);
BIND(newsock,číslo portu);

repeat
    RECVFROM(newsock, adresa..);
    zpracování požadavku z "adresa"
until ....
```

práce se sockety – spojovaná komunikace

- primitivní operace, určené pro spojovanou komunikaci
  - LISTEN(socket, …)
    - server chce přijímat požadavky z určitého socketu – počáteční akce, čekání na žádost o navázání spojení
  - ACCEPT(socket, ….)
    - přijetí požadavku na navázání spojení (na straně serveru)
  - CONNECT(socket,adresa_serveru …)
    - požadavek (klienta) na navázání spojení (ze zadaného socketu) se serverem na zadané adrese (IP adresa, port)
  - SEND(socket,data, ….)
    - pošle data skrz navázané spojení
  - RECV(socket,buffer,délka,flags)
    - pro příjem ze zadaného socketu při navázaném spojení

představa průběhu spojované komunikace



typ transportní služby

- STREAM
  - transportní protokol (služba) služba vytváří iluzi bytové roury
    - data jsou přijímána/vydávána po jednotlivých bytech
      - neexistuje jiné členění, např. na bloky, zprávy, …
    - přenos je spolehlivý, pořadí dat se nemění, nejsou duplicity ani ztráty
    - je zabudováno řízení toku
  - takto funguje protokol TCP
    - členění na bloky pro potřeby přenosu je realizováno interně a transparentně
- RAW
  - speciální režim
    - pro "přímý přístup" k nižším vrstvám
      - pro testování, PING, OSPF ….
- DATAGRAM
  - transportní protokol (služba) vytváří iluzi blokového přenosu
    - data jsou přijímána/přenášena/vydávána již členěná na bloky (datagramy)
    - přenos je nespolehlivý, nespojovaný, ztráty a duplicity nejsou ošetřeny, pořadí doručování není zajištěno
    - není zabudováno řízení toku
  - takto funguje protokol UDP

STREAM
DATAGRAM
RAW

volání:

descriptor=SOCKET(pf,type,protocol)

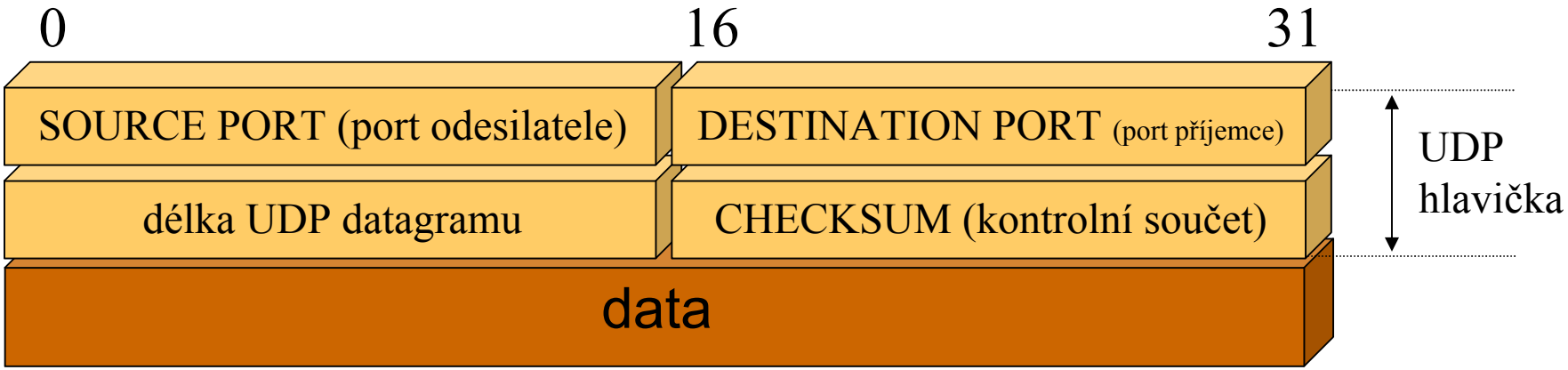
kde:

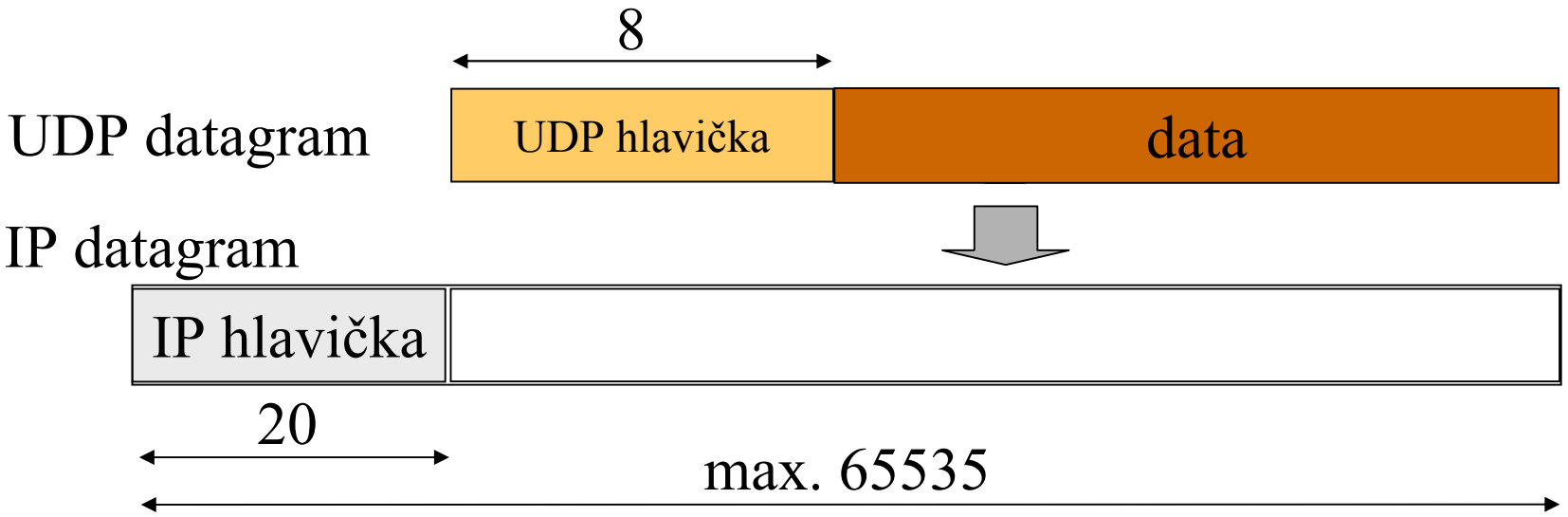
type je požadovaný typ služby

protokol UDP (User Datagram Protocol)

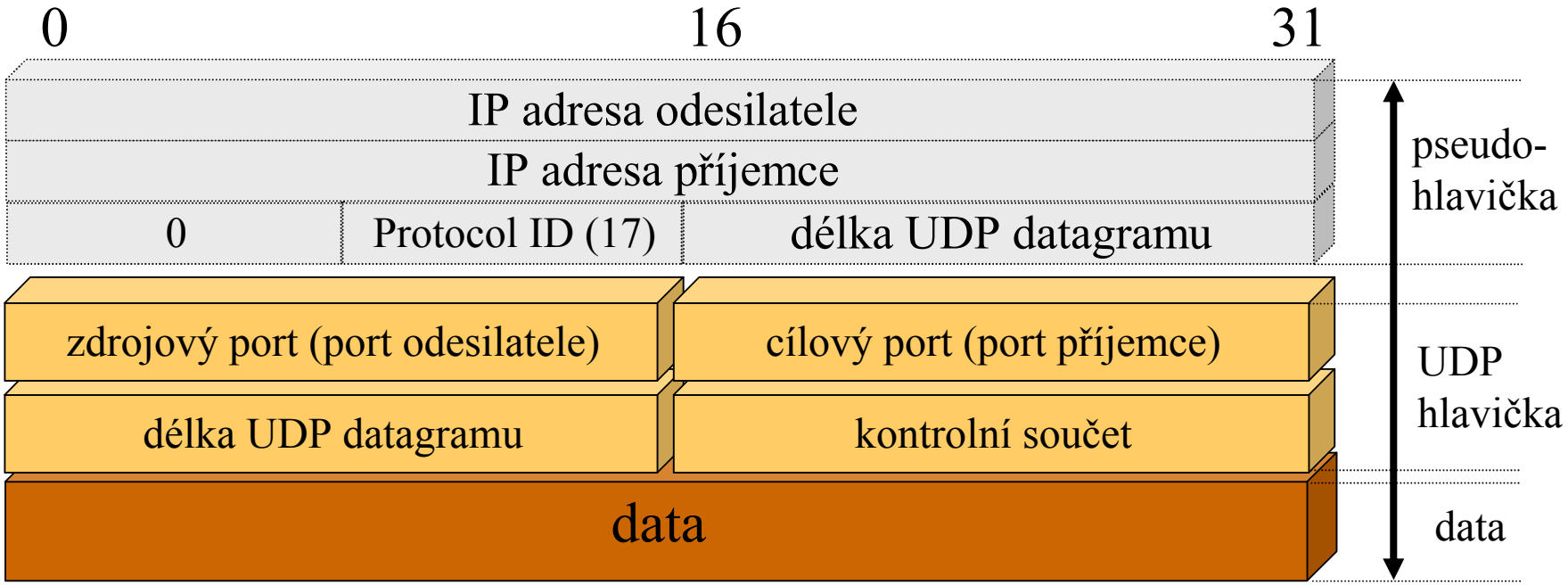
- je maximálně jednoduchou nadstavbou nad protokolem IP
  - nemění základní vlastnosti protokolu IP
  - navíc poskytuje jen multiplexing/demultiplexing
  - má kontrolní součet který pokrývá hlavičku i data
    - kontrolní součet IP datagramu pokrývá pouze hlavičku
    - kontrolní součet UDP datagramu lze vypnout
- protokol UDP používají takové aplikace, které potřebují co nejrychlejší a nejefektivnější komunikaci
  - UDP není zatížen velkou režií jako protokol TCP
- vlastnosti UDP:
  - poskytuje nespolehlivé přenosové služby
  - funguje nespojovaně
  - vytváří iluzi blokového přenosu
    - přenáší UDP Datagramy
    - velikost bloku (datagramu):
      - taková, aby se vešla do IP datagramu (2^{16} – 20 - 8)
      - v praxi se posílají velmi malé bloky, např. do 512 bytů
  - může být použit pro rozesílání
    - broadcast i multicast
      - u spojovaného protokolu to nejde)
  - komunikace je bezestavová
    - u TCP je stavová

formát UDP Datagramu





kontrolní součet UDP datagramu



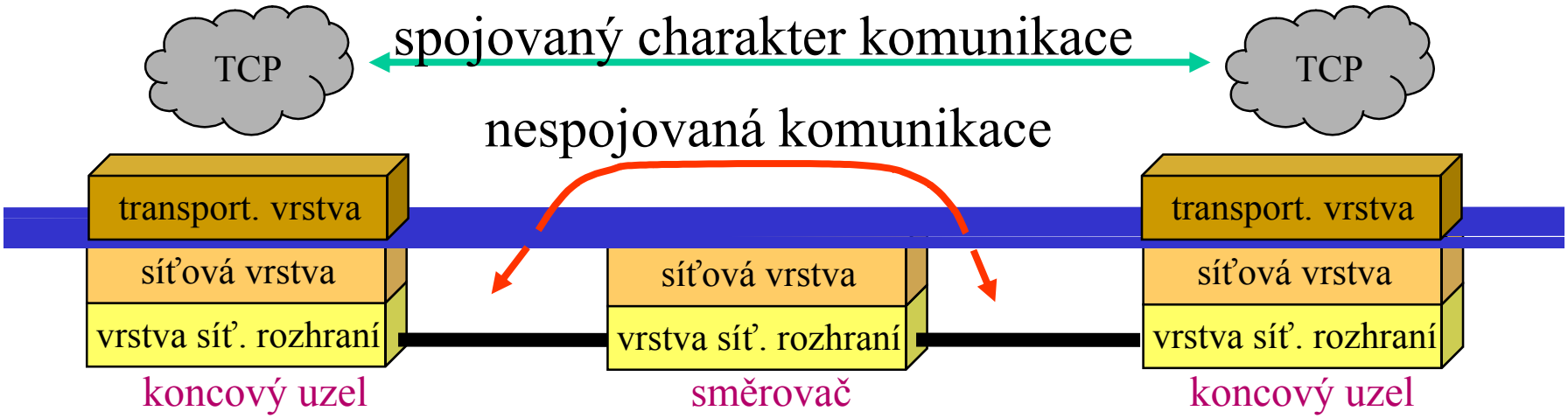
- kontrolní součet se počítá z UDP datagramu doplněného o pseudohlavičku
  - ale tato pseudohlavička se nepřenáší, existuje jen pro potřebu výpočtu kontrolního součtu
    - smyslem je ochrana proti nesprávně doručeným datagramům
- kontrolní součet se počítá v jedničkovém doplňku
  - nulový kontrolní součet = samé jedničky (tzv. záporná nula)
  - žádný kontrolní součet = samé nuly (tzv. kladná nula)

když UDP protokol přijme datagram se špatným kontrolním součtem, zahodí jej (a není generována žádná ICMP zpráva)

protocol TCP (Transmission Control Protocol)

- je velmi úspěšný:
  - dobře řeší poměrně složitý problém
    - funguje efektivně v sítích, které se významně liší svými vlastnostmi, např. přenosovým zpožděním
- vlastnosti poskytovaných služeb:
  - spojovaný charakter
    - práce stylem: navaž spojení, posílej/přijímej, ukonči spojení
  - dvoubodové spojení
    - vždy jen jeden příjemce a jeden odesilatel
  - plně duplexní spojení
    - obousměrný přenos, současně
  - řízení toku
    - přizpůsobuje se schopnostem příjemce
  - efektivní fungování přenosů
    - používá kontinuální potvrzování
    - snaží se předcházet zahlcení
  - "plná" spolehlivost
    - protokol ošetřuje chyby při přenosech, duplicity, ztráty, garantuje pořadí doručování dat
  - "stream interface"
    - vůči vyšším vrstvám vytváří iluzi bytové roury, přijímá i vydává data po bytech, nikoli po blocích
  - korektní navazování spojení
    - zajišťuje že obě strany souhlasí s navázáním spojení a že nedojde k deadlocku ani "ztrátám" pokusů o navázání
  - korektní ukončení spojení
    - protokol zajistí že před ukončením spojení jsou přenesena všechna odeslaná data

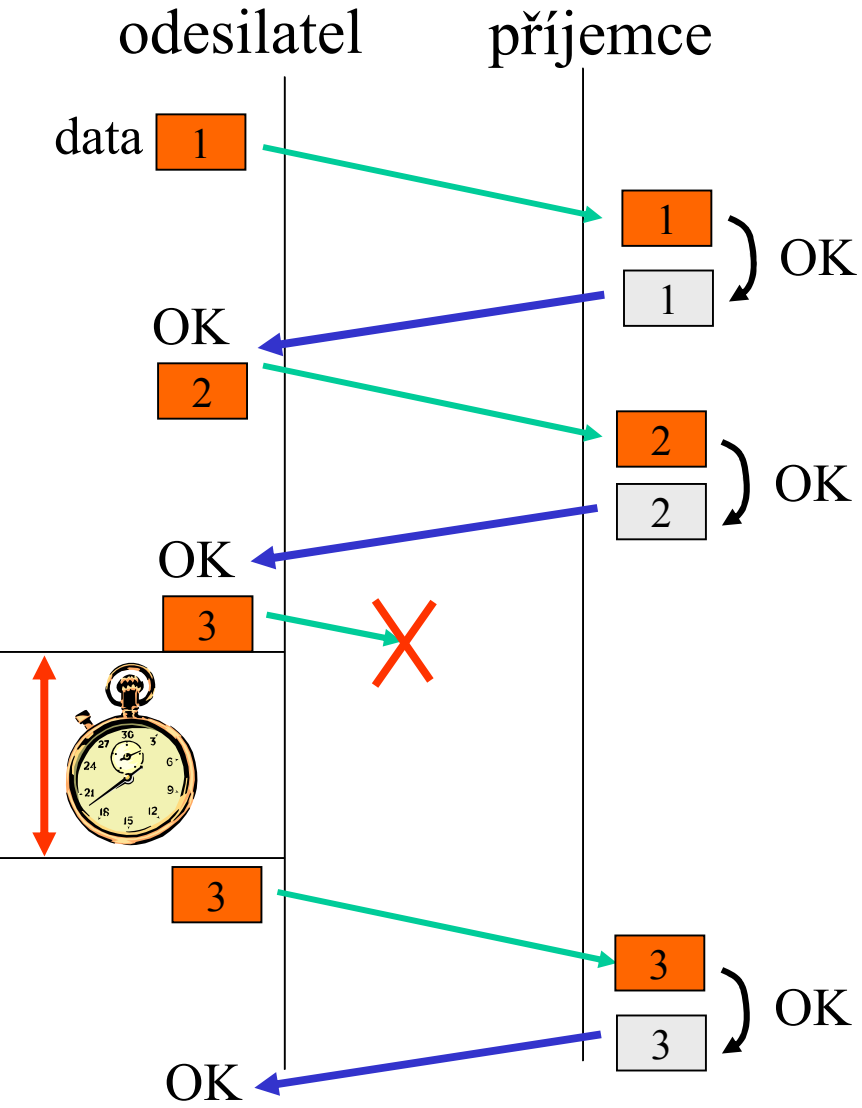
spojovaný charakter komunikace



- nejde o "skutečný" spojovaný přenos na principu virtuálních okruhů
  - přenosová infrastruktura (protokol IP) funguje nespojovaně
- jde o "softwarovou emulaci" v koncových uzlech
  - mezilehlé uzly o tom nevědí, fungují nespojovaně (a také nespolehlivě)

- co všechno musí být ošetřeno:
  - nespolehlivost přenosové infrastruktury
    - ztrácí data, mění pořadí, duplicity, …
      - ztratit se může i žádost o navázání spojení, potvrzení, ..
  - reboot uzlů
    - uzel ztratí historii, je třeba ošetřit původně existující spojení, "minulá data", ….

zajištění spolehlivosti (obecně)



- je použita celá řada technik
- základem je potvrzování
  - příjemce generuje kladná potvrzení
  - odesilatel po každém odeslání spustí časovač
    - pokud nedostane potvrzení včas (do vynulování časovače), posílá data znovu
- jak dlouho má odesilatel čekat?
  - krátká doba: zbytečně se posílá znovu
    - data nebo potvrzení se mohly jen trochu zdržet
  - dlouhá doba: neefektivní přenosy

v lokálních sítích je malé přenosové zpoždění, v rozlehlých je větší

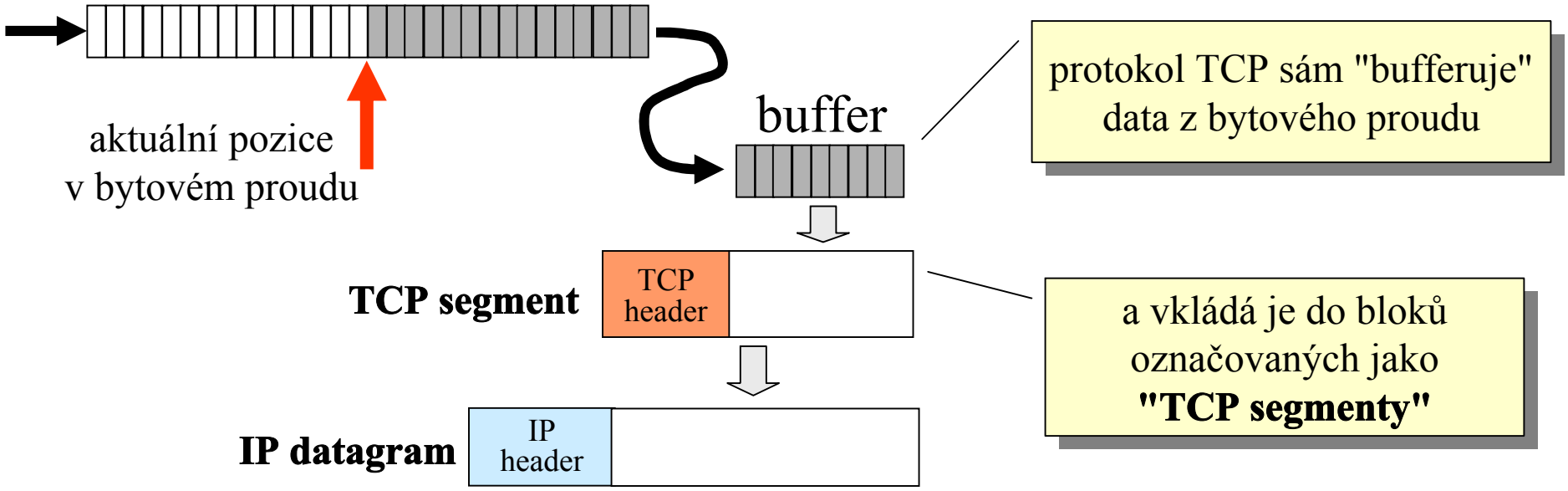
- TCP konkrétně:
  - nepoužívá jednotlivé potvrzování, ale kontinuální.
  - nečísluje přenášené pakety jako takové, ale je číslována pozice v bytovém proudu !!!

adaptivní opakování

- idea:
  - TCP průběžně monitoruje přenosové zpoždění a podle něj mění délku časového intervalu, po který čeká na potvrzení
- ve skutečnosti:
  - TCP monitoruje "dobu obrátky"
    - nepozná přenosové zpoždění, ale sleduje za jak dlouho dostává odpovědi
  - TCP vyhodnocuje:
    - vážený průměr dob obrátky
    - rozptyl dob obrátky
  - "čekací dobu" TCP vypočítává jako funkci váženého průměru a rozptylu
- výsledný efekt:
  - "čekací doba" vychází "těsně nad" střední dobou obrátky
    - je-li doba obrátky konstantní, čekací doba se jí více přibližuje
    - jakmile se doba obrátky začíná měnit, čekací doba se zvětšuje
  - dobře to reaguje na:
    - prodlužování doby obrátky při "dávkách paketů"
    - zkrácení doby obrátky po odeslání dávky paketů

potvrzování je nesamostatné, vkládá se do paketů cestujících opačným směrem
(tzv. piggybacking)

bytový proud v TCP

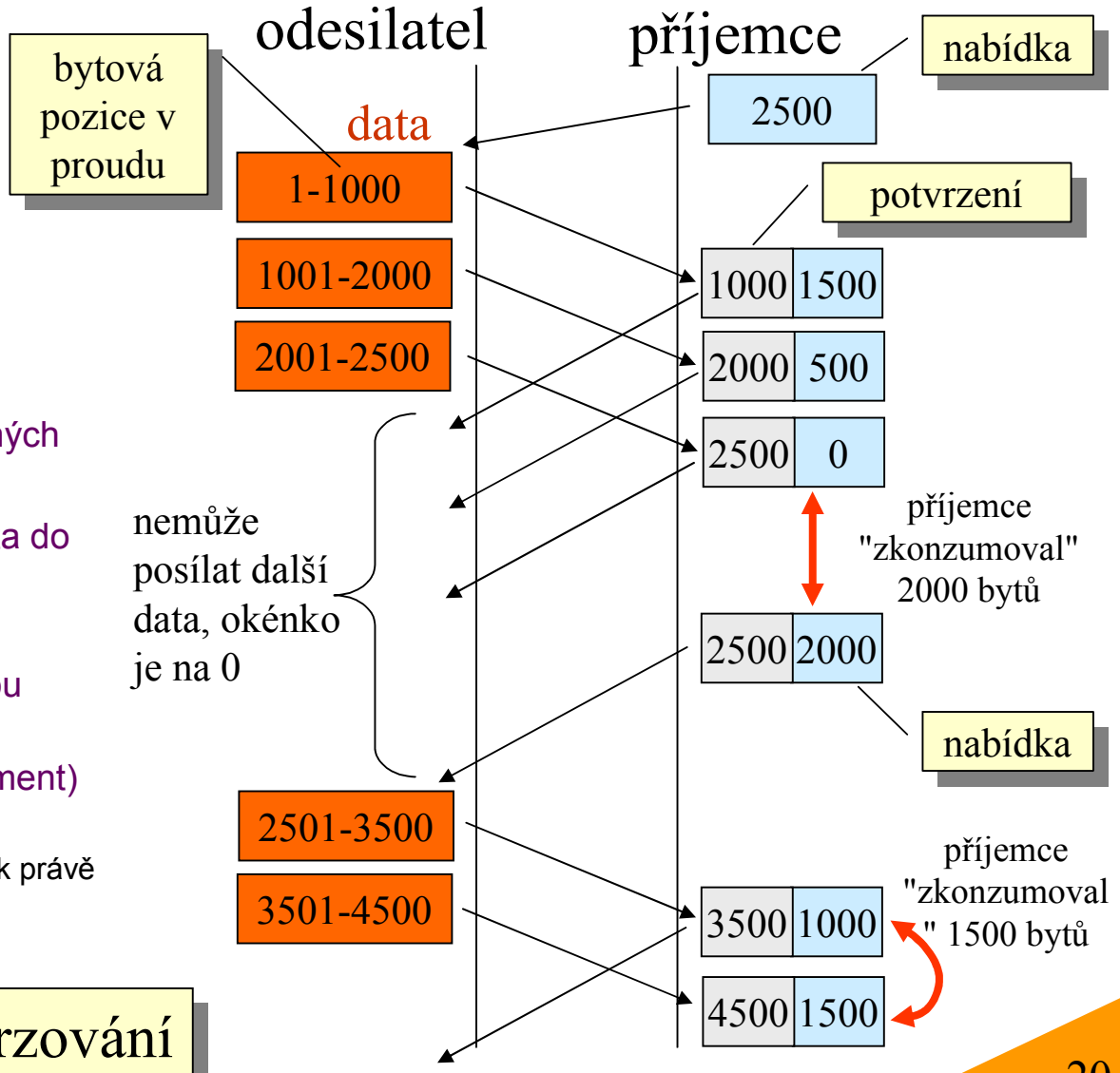


- protokol TCP přijímá/vydává data po jednotlivých bytech (pracuje s bytovým proudem, byte stream)
  - ve skutečnosti data bufferuje (ukládá do bufferu, jehož velikost volí podle parametru MTU)
- obsah bufferu je odesílán až po jeho naplnění
  - aplikace má možnost vyžádat okamžité odeslání obsahu bufferu (operace PUSH)

- TCP potřebuje označovat jednotlivé byty v rámci proudu
  - jelikož nepracuje s bloky
  - potřebuje to například pro potvrzování
    - aby vyjádřil "kam až" se data přenesla
  - používá k tomu (32-bitovou) pozici v bytovém proudu
    - začíná od náhodně zvoleného čísla

buffery a řízení toku

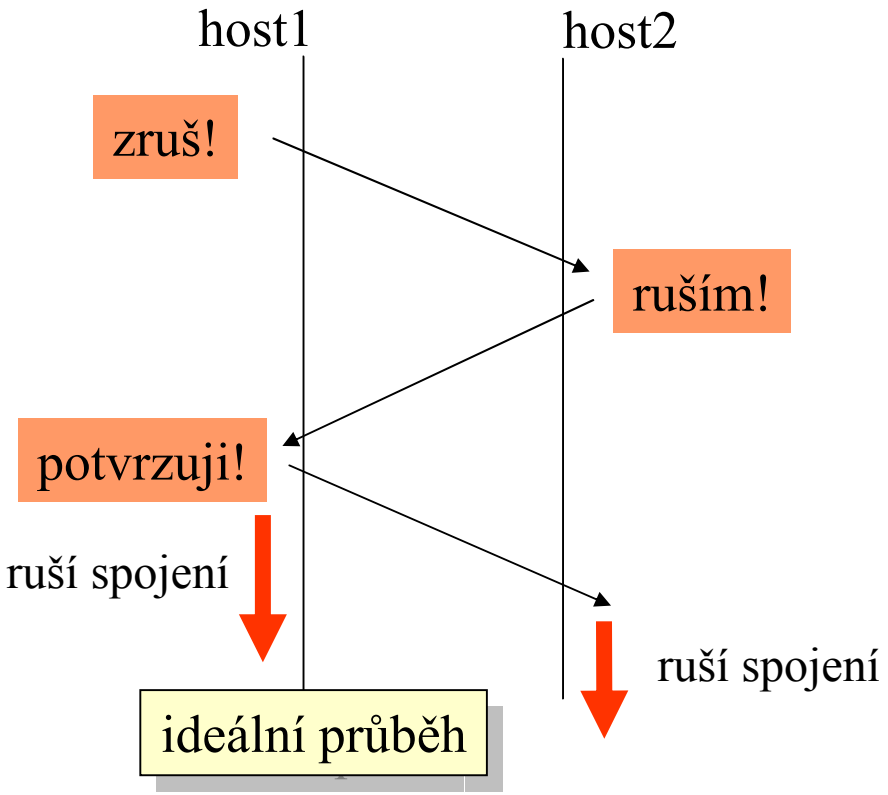
- TCP se snaží řídit tok dat
  - aby odesilatel nezahlcoval příjemce a kvůli tomu nedocházelo ke ztrátám dat
- podstata řešení:
  - používá se metoda okénka
    - okénko udává velikost volných bufferů na straně příjemce
    - odesilatel může posílat data do "zaplnění" okénka
    - příjemce spolu s každým potvrzením posílá také svou "nabídku" (údaj o velikosti okénka, window advertisement)
      - tím říká, kolik dat je ještě schopen přijmout (navíc k právě potvrzovaným)



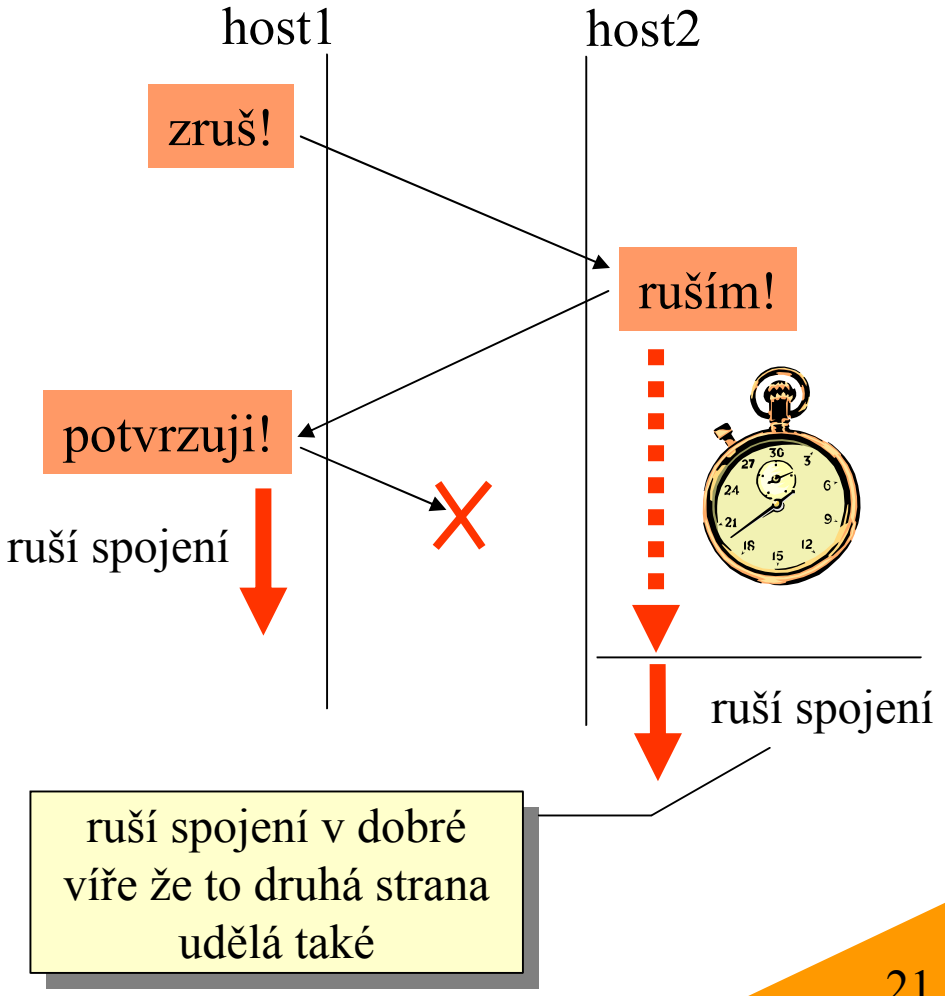
používá kontinuální potvrzování

navazování a rušení spojení

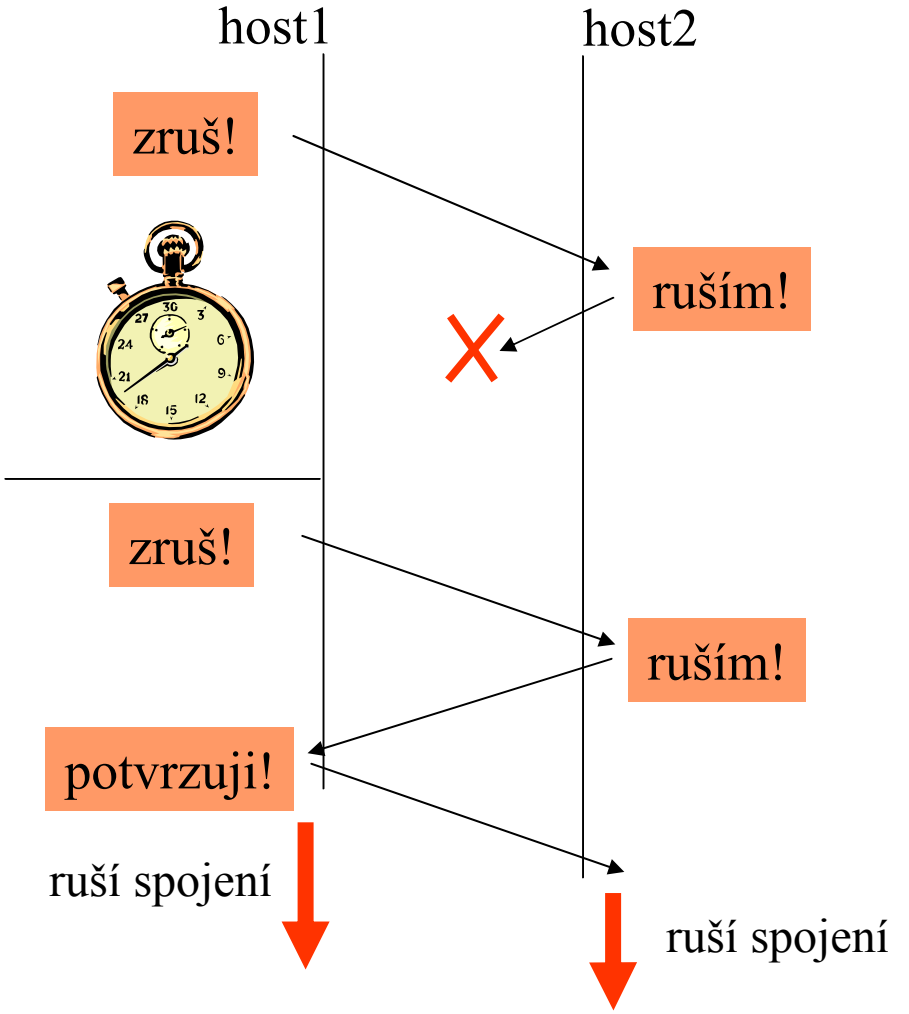
- TCP používá 3-fázový handshake
  - je to nutné ke korektnímu navázání/zrušení spojení v prostředí, kde může dojít ke zpomalení, ztrátě, duplicitě, ….

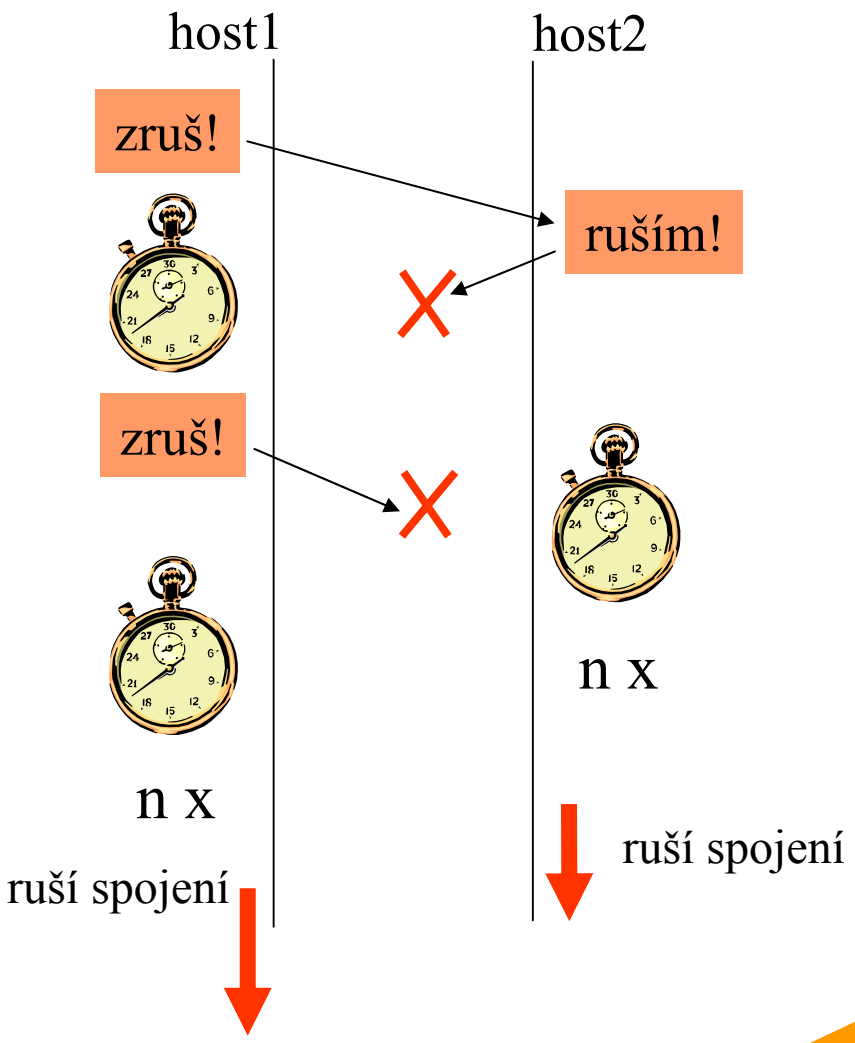


- ošetřuje to většinu nežádoucích situací



navazování a rušení spojení (příklady nestandardního průběhu)

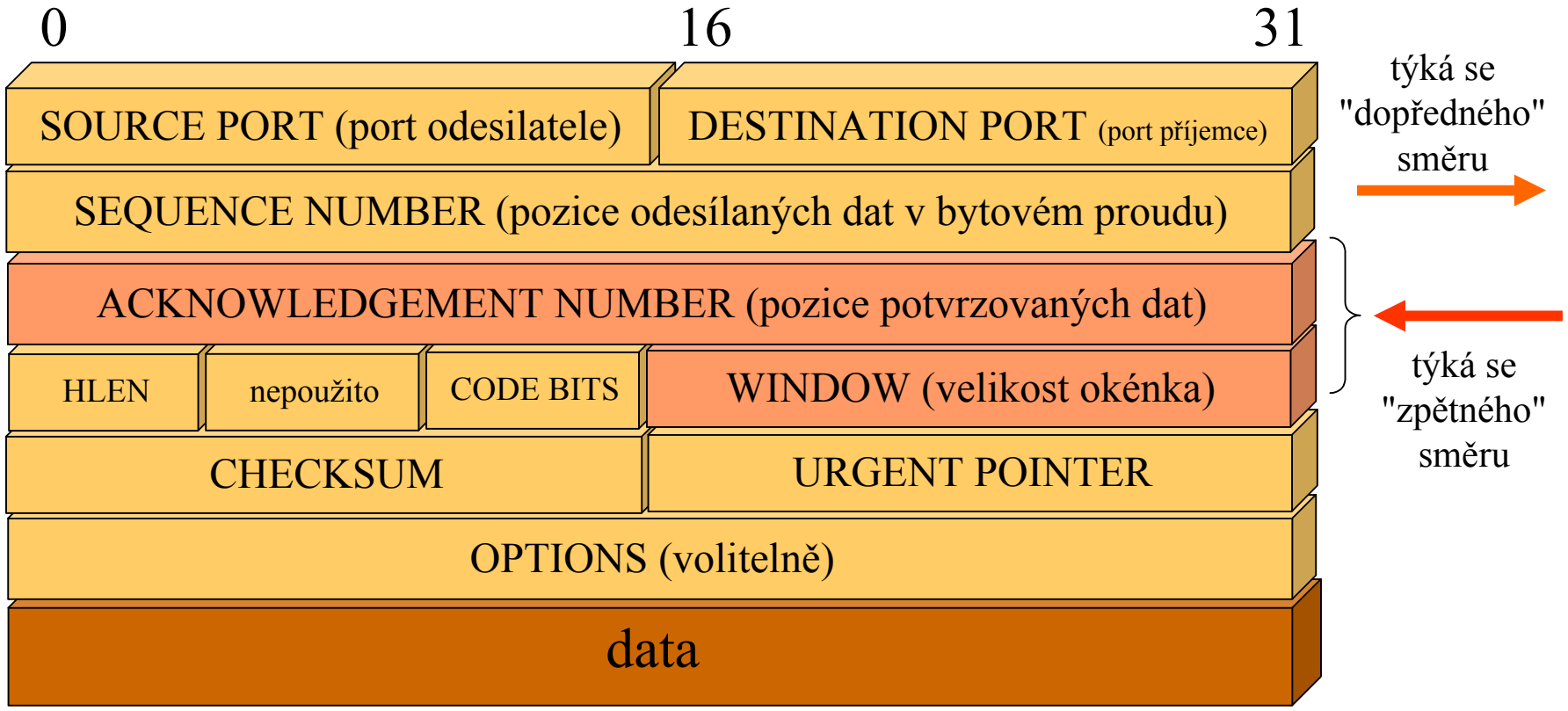




ochrana před zahlcením

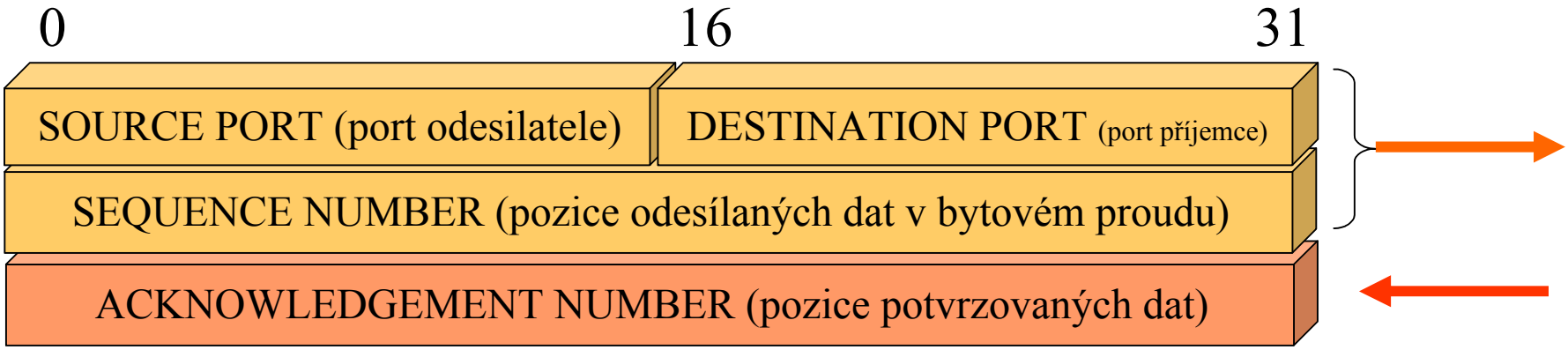
- pozorování:
  - většina ztrát přenášených dat jde spíše na vrub zahlcení než chybám HW
  - transportní protokoly mohou nevhodným chováním důsledky ještě zhoršovat
    - tím že se snaží odesílat další data
- přístup TCP
  - každou ztrátu dat chápe jako důsledek zahlcení
    - a nasazuje protiopatření (congestion control)
  - po ztrátě paketu jej pošle znovu, ale neposílá další a čeká na potvrzení
    - tj. přechází z kontinuálního potvrzování na jednotlivé !!
      - nevysílá tolik dat, kolik mu umožňuje okénko!!
  - přijde-li potvrzení včas, odešle dvojnásobek dat a čeká na potvrzení
    - odešle dva pakety
  - takto postupuje dokud se nenarazí na omezení dané aktuální velikostí okénka
    - postupně se tak vrací na kontinuální potvrzování

Formát TCP segmentu



kontrolní součet se počítá s použitím stejné pseudohlavičky jako u protokolu UDP !!

formát TCP segmentu

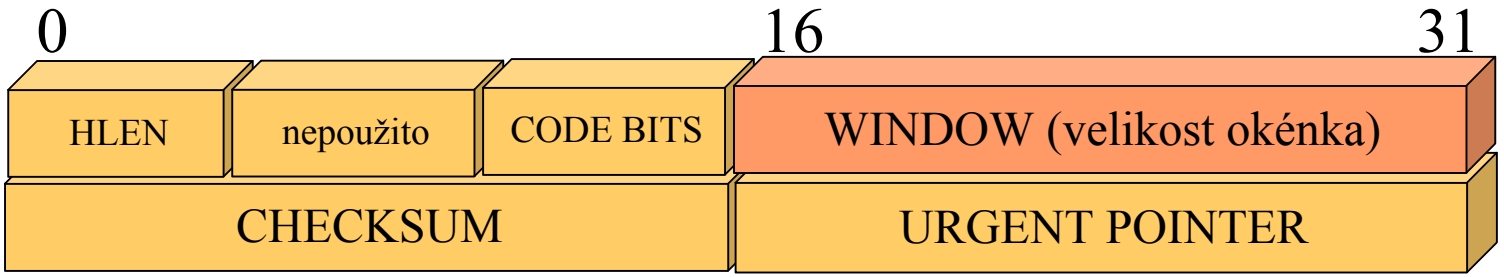


- připomenutí: TCP nečísluje segmenty které posílá
  - místo toho čísluje bytové pozice dat v bytovém proudu
  - jde o 32-bitové číslo
    - začíná na náhodné hodnotě

týká se opačného směru přenosu než SEQUENCE NUMBER

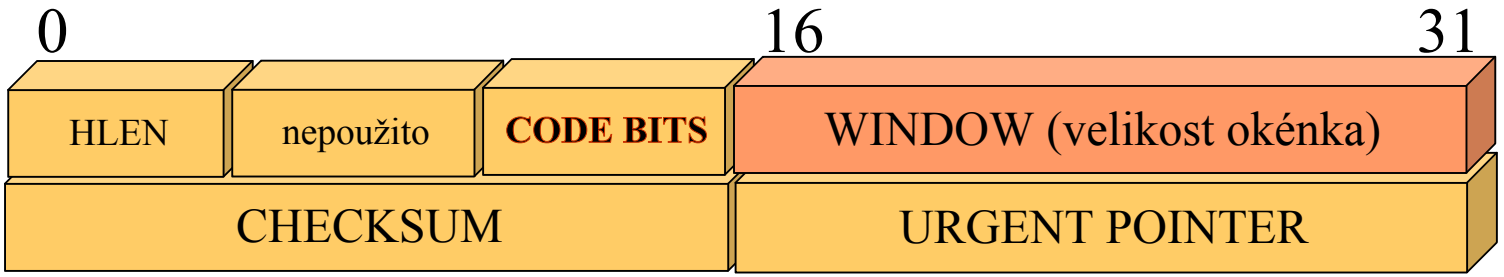
- **SEQUENCE NUMBER**
  - pozice prvního odesílaného bytu v proudu
    - týká se "odesílajícího" směru
- **ACKNOWLEDGEMENT NUMBER**
  - pozice následujícího očekávaného bytu v bytovém proudu
  - má to význam potvrzení úspěšného doručení všech bytů na nižších pozicích !!!
  - potvrzení nemusí být generováno pro každý přijatý segment
    - následující potvrzení potvrzuje i předchozí segmenty

formát TCP segmentu



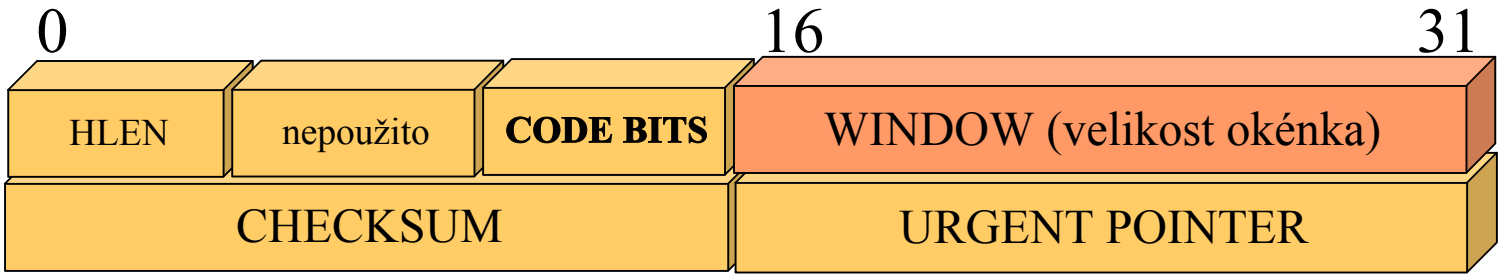
- HLEN
  - Header LENgth, velikost hlavičky v násobcích 32 bitů
    - kvůli volitelným položkám
- URGENT POINTER
  - část přenesených dat je možné prohlásit za "přednostní data"
    - musí být nastaven příznak URG v CODE BITS
- Window
  - "nabídka" velikosti okna
    - počet bytů které je příjemce schopen přijmout, navíc k právě přijatým
  - týká se "opačného směru" přenosu
    - než jsou přenášená data
  - max. velikost je 64 KB, po dohodě obou stran lze zvětšit
    - řeší se pomocí volitelných položek v záhlaví TCP segmentu

formát TCP segmentu



- CODE BITS
  - jednotlivé příznaky, ovlivňující přenos
  - **SYN**: pokud je nastaven, v poli "SEQUENCE NUMBER" je počáteční hodnota pro nové spojení
    - používá se při navazování spojení ("synchronizace čítačů")
  - **ACK**: pokud je nastaven, v poli "ACKNOWLEDGEMENT NUMBER" je platná hodnota (pořadové číslo dalšího očekávaného bytu)
  - **FIN**: pokud je nastaven, v poli "SEQUENCE NUMBER" je pořadové číslo posledního přeneseného bytu
    - používá se při ukončování spojení, aby příjemce věděl kde je konec dat
  - **RST**: spojení má být okamžitě zrušeno (rozvázáno, ukončeno)

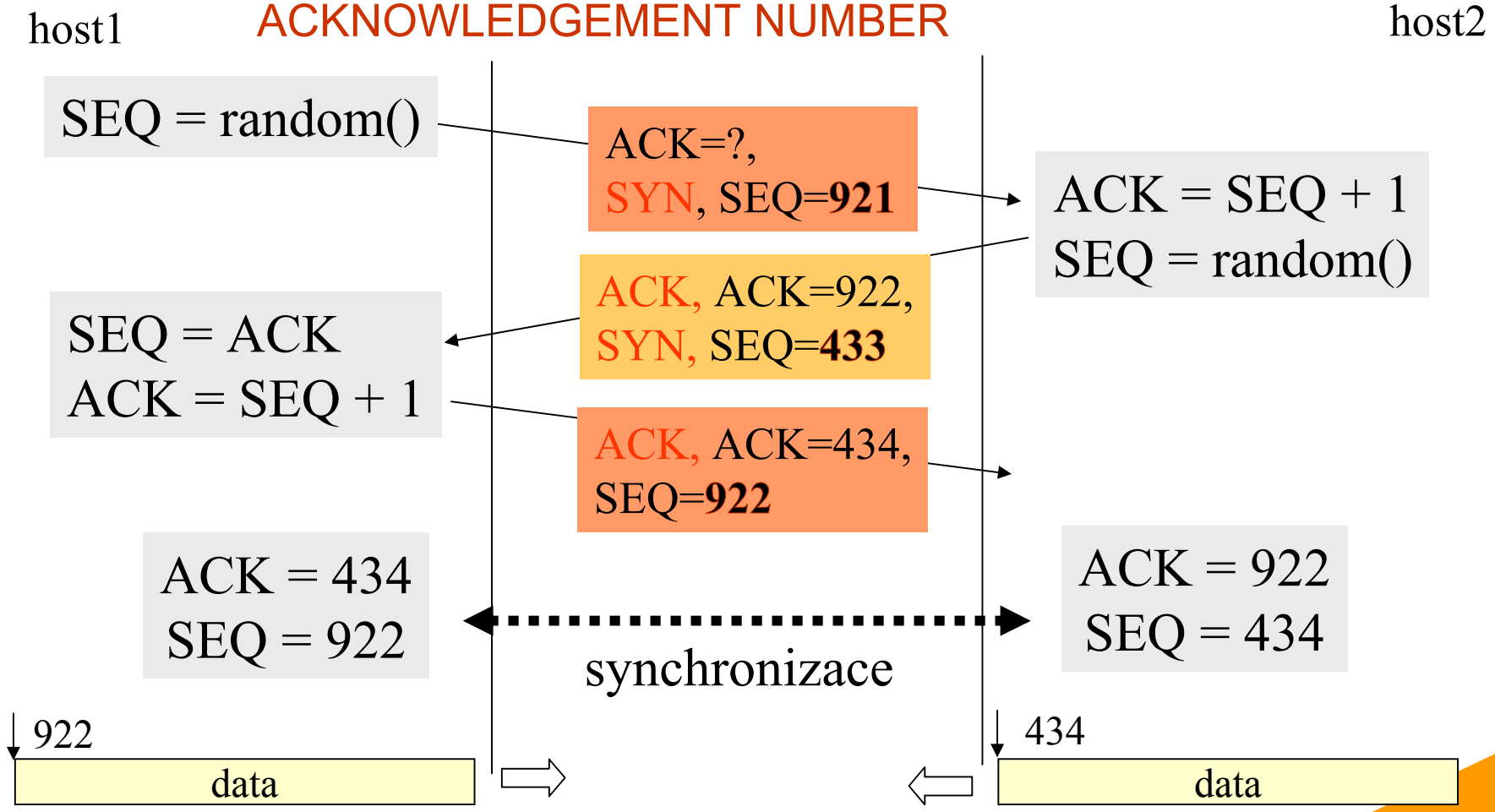
formát TCP segmentu



- CODE BITS
  - jednotlivé příznaky, ovlivňující přenos
  - URG: prioritní (urgentní) data: SEQUENCE NUMBER + URGENT POINTER ukazují na poslední byte "urgentních" dat
    - není způsob jak vyznačit začátek urgentních dat
    - TCP příjemce by mělo upozornit aplikaci na příchod segmentu s nastaveným příznakem URG, a pak také na konec urgentních dat
  - PSH: příznak pro "push", data by měla být předána sousední vrstvě bez meškání (a bufferování)
    - je na implementaci, jak to konkrétně zařídí

navazování spojení - detailněji

- TCP volí ISN (Initial Sequence Number) **náhodně**
  - jako první hodnotu pro SEQUENCE NUMBER a ACKNOWLEDGEMENT NUMBER



QoS - zajištění kvality služeb

- pozorování:
  - různé druhy přenosů mají různé nároky
  - ale standardní způsob fungování přenosových sítí (best effort) měří všem stejně!!
- řešení:
  - žádné / "hrubou silou"
    - tzv. overprovisioning
    - zvýší se přenosová i další kapacita, tak aby k problémům nedocházelo tak často
  - podpora kvality služeb (QoS, Quality of Service)
    - s různými druhy přenosů bude "nakládáno různě"
- možnosti implementace QoS:
  - přímo na úrovni síťové vrstvy
    - kde jinak vzniká "best effort"
    - např. protokol IP má v halvičce položku (ToS) pro vyjádření požadavků paketu na QoS – ale standardně se ignoruje
  - "předpoklady" na síťové vrstvě, hlavní část řešení na transportní vrstvě
    - příklad: na úrovni síťové vrstvy se rezervují určité kapacity, které se pak využívají na úrovni transportní vrstvy
  - řešení na transportní nebo aplikační vrstvě
    - bez "předpokladů" na síťové vrstvě
- možné přístupy a techniky
  - traffic conditioning,
    - úpravy datového provozu tak, aby "lépe prošel"
  - "Integrated Services"
    - zásadnější změny v přenosové části sítě, tak aby bylo možné rezervovat potřebné zdroje a dát "pevné záruky"
  - "Differentiated Services"
    - menší zásahy do přenosové části sítě, snaha diferencovat provoz a poskytnout alespoň "záruku rozdílu"

QoS – požadavky aplikací

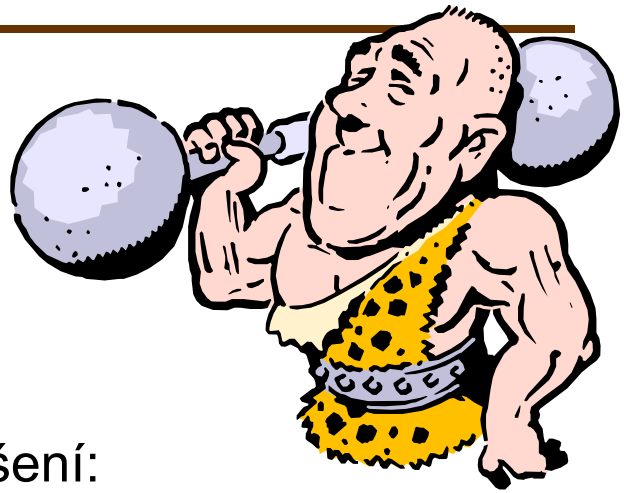
- problém multimediálních aplikací:
  - potřebují dostávat svá data
    - **včas** (s malým přenosovým zpožděním - latence)
    - **pravidelně** (s rovnoměrnými odstupy mezi jednotlivými částmi - jitter)
  - příklad: přenos hlasu (latence)
    - business kvalita: max zpoždění 150 milisekund
    - při 250 msec. zpoždění znatelná a vadí
    - nad 500 msec. nepoužitelné pro přenos hlasu
  - příklad: přenos obrazu (jitter)
    - nepravidelnost lze přirovnat k nerovnoměrné rychlosti posunu filmového pásu při promítání
      - když jsou nerovnoměrnosti moc velké, nelze se na to dívat

	Požadavek na			
	spolehlivost	nízké zpoždění (latence)	pravidelnost (nízký jitter)	přenosovou kapacitu
email	Max.	Min.	Min.	Min.
přenos souborů	Max.	Min.	Min.	Medium
www	Max.	Medium	Min.	Medium
remote login	Max.	Medium	Medium	Min.
Audio on demand	Min.	Min.	Max.	Medium
Video on demand	Min.	Min.	Max.	Max.
IP telefonie	Min.	Max.	Max.	Min.
Videokonference	Min.	Max.	Max.	Max.

  - spolehlivost:
    - řada multimediálních aplikací na spolehlivosti netrvá
      - a dávají přednost nízké latenci a pravidelnosti doručování
    - například srozumitelnosti hlasu (zásadněji) nevadí ani ztráta či poškození 20% dat

transportní vrstva a QoS

- standardní způsob řešení transportní vrstvy QoS (Quality of Service) nepodporuje
  - síťová vrstva "měří všem přenosům stejně"
    - pokud funguje na paketovém principu a na bázi "best effort", jako IP v rámci TCP/IP sítích
  - ani TCP ani UDP (v rámci transportní vrstvy TCP/IP) nevychází vstříc potřebám QoS
    - negarantují maximální zpoždění ani pravidelnost v doručování
- díky tomu je přenosová infrastruktura dnešního Internetu tak laciná, dostupná a rychlá
  - problém je ale v tom, že nevychází vstříc multimediálním přenosům, které mají své specifické požadavky na QoS

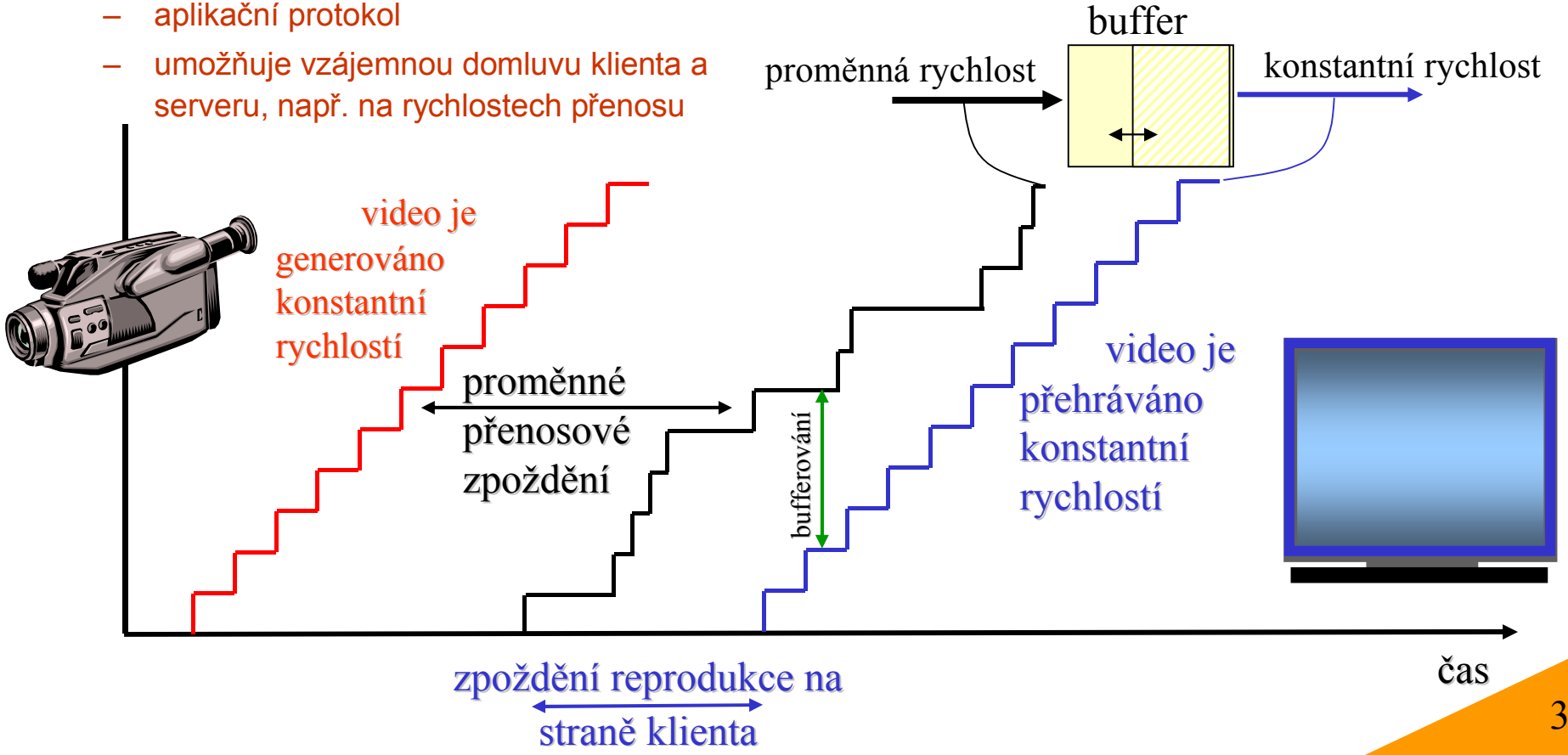


možné řešení:

- přístup "hrubou silou"
  - zvyšuje se disponibilní přenosová kapacita
    - hlavně na úrovni páteřních sítí
  - problém se tím neřeší
    - pouze se statisticky snižuje četnost jeho výskytu
  - dnes je to nejschůdnější cesta
    - relativně laciná
    - ostatní jsou komplikovanější

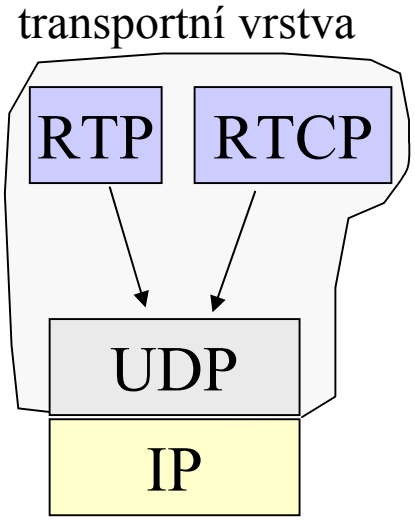
"client buffering" QoS na aplikační úrovni - princip

- u jednosměrných (neinteraktivních) multimédií – například u videa – lze vyrovnávat "jitter" až u klienta vhodným bufferováním
  - u interaktivních přenosů lze využít také, ale celkové zpoždění nesmí být příliš velké!!
- RTSP – Real Time Streaming Protocol
  - aplikační protokol
  - umožňuje vzájemnou domluvu klienta a serveru, např. na rychlostech přenosu



RTP/RTCP

- "čistě transportní" podpora QoS
  - standardizovaný způsob "balení" multimediálních dat do přenášených paketů, s podporou jejich multimediálního charakteru
  - ale bez vlivu na způsob jejich přenosu
    - ten je stále best effort!!!
- RTP (Real Time Protocol)
  - "balí" jednotlivé části multimediálních dat do vlastních bloků (paketů)
    - a ty vkládá do UDP paketů
  - připojuje informace
    - o typu multimediálního obsahu
      - Payload type 0: PCM, 64 kbps
      - Payload type 3, GSM, 13 kbps
      - Payload type 26, Motion JPEG
      - Payload type 33, MPEG2 video
      - ….



- o pořadí paketu
  - jednotlivé pakety čísluje, usnadňuje detekci ztracených paketů
- o čase vzniku dat (timestamp)
  - říká kdy přesně data vznikla
  - tím usnadňuje jejich bufferování na straně klienta
- o konkrétním streamu (proudu)
  - v rámci jednoho RTP přenosu může být přenášeno více samostatných proudů (streamů)
  - podporuje multicast

- RTCP (Real Time Control Protocol)
  - zprostředkovává vzájemné informování zdroje a příjemců
    - např. o procentu ztracených paketů, o jejich zpoždění, o schopnostech příjemce apod.
    - přenáší popis RTP streamu, ….

QoS: Integrated Services

- snaha:
  - garantovat každému přesně to, co potřebuje
- základní princip:
  - při navazování spojení žadatel specifikuje, co bude potřebovat
    - jakou kapacitu, rychlost, zpoždění atd.
  - síť posoudí, zda to dokáže zajistit a garantovat
  - pokud ano:
    - spojení je navázáno
  - pokud ne:
    - žádost o navázání spojení je odmítnuta
- jak lze realizovat?
  - je nutné k tomu vyhradit (rezervovat) potřebný objem zdrojů
    - včetně přenosové kapacity a výpočetní kapacity v přepojovacích uzlech
  - nelze řešit výhradně na úrovni transportní vrstvy !!!!
    - je nutná určitá spolupráce již na úrovni vrstvy síťové
    - nestačí to implementovat v koncových uzlech, **musí být změněny i vnitřní a páteřní části sítě**

- princip realizace:
  - žadatel o navázání spojení uvede své R-spec
    - (**R**equirements), co bude od sítě požadovat
  - žadatel uvede své T-spec
    - jak bude vypadat datový provoz (**Traffic**), který bude generovat
  - musí existovat mechanismus (protokol), který R-spec i T-spec předá jednotlivým směrovačům v síti a "sjedná s nimi" jejich akceptování/odmítnutí
    - takovým protokolem je **RSVP**
      - **ReSerVation Protocol**
    - směrovače pak příslušné zdroje vyhradí pro právě vznikající virtuální okruh, po kterém pak přenos probíhá

v zásadě je to návrat k principu přepojování okruhů

se všemi problémy – například neefektivní využití vyhrazené kapacity

QoS: Differentiated Services

- myšlenka:
  - nesnažit se o "absolutní" naplnění požadavků (jako Integrated Services), ale nabídnout alespoň "relativní" (rozdílové, differentiated) služby
    - jeden druh provozu je upřednostňován na úkor jiného
- princip realizace:
  - zavede se několik tříd provozu
    - a každá třída bude mít jinou přednost/prioritu při přenosu a zpracování
  - každý přenášený paket či každé spojení se může "přihlásit" k určité třídě (prioritě)
    - a podle toho je s ním nakládáno
  - musí být podporováno v celé síti, již na úrovni síťové vrstvy
    - na rozdíl od "integrovaných služeb" mohou být jednotlivé třídy nastaveny dopředu a pevně
- praktické využití:
  - jednotlivé pakety deklarují svou příslušnost k určité třídě provozu skrze vhodnou "nálepku"
    - prefix
    - nastavení údaje ve své hlavičce
  - v IPv4: využívá se byte ToS (Type of Service)
- příklady (TCP/IP):
  - Expedited Forwarding
    - 2 třídy: Expedited a Regular
    - pakety v třídě Expedited mají absolutní přednost při přenosu před pakety z třídy regular
  - Assured Forwarding
    - 4 třídy podle priorit pro přenos
    - k tomu ještě 3 úrovně priorit pro zahazování paketů (při přetížení)
    - celkem 12 kombinací (tříd)