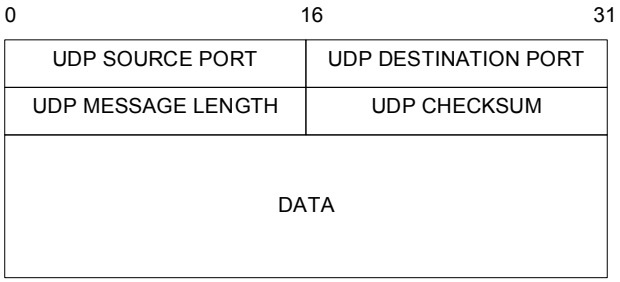
UDP – User Datagram Protocol

Protokol UDP patrí do rodiny TCP/IP. Je to veľmi jednoduchý transportný protokol. Poskytuje základný mechanizmus, akým môžu aplikácie posielať datagramy. Je definovaný nad protokolom IP, ktorý využíva na posielanie datagramov. Poskytuje **nespoľahlivý, connectionless prenos datagramov**. UDP protokol pridáva adresovanie na transportnej vrstve tak, že definuje prístupové miesta v rámci jedného hosta (fyzického interfejsu stanice) pre viaceré aplikácie. Tieto prístupové miesta nazývame *protocol ports.* Hovoríme, že aplikácie komunikujú pomocou User datagramov, ktoré sú prístupné cez *porty*. UDP k funkcionalite IP pridáva len možnosť rozlíšiť viacero destinácií na jednej stanice, so samotným zabezpečením prenosu (duplikáty paketov, straty, poruchy, poprehadzovanie poradia, oneskorenie, ...) sa musí vysporiadať vrstva nad UDP (obyčajne priamo aplikácia).

Tak, ako na smerovanie na tretej vrstve sa používajú IP adresy, tak na štvrtej-transpornej vrstve je potrebné datagramy smerovať na konkrétne aplikácie. Na to pri UDP datagrame slúžia tzv **porty** (*protocol ports*). Každá aplikácia musí od operačného systému dostať tzv. **číslo portu**, ktorý je ďalej uvádzaný v odchádzajúcich datagramoch ako *source port*. Pri príjme daná stanica, resp. UDP proces transportnej vrstvy prijíma datagramy a buffruje ich na jednotlivé porty (do front). Pokiaľ stanica dostane datagram s cieľovým portom, ktorý nie je používaný (nemá ho kam zaradiť), tak ho zahodí a pošle ICMP správu *port unreachable*. Keď je port plný (fronta daného portu), tak datagram zahodí tiež. Ako čísla zdrojových portov možno použiť tzv. ***well-known porty***, ktoré sú dohodnuté štandardom a sú všeobecne známe (napr. HTTP má 80), prípadne pre ostatné aplikácie ich možno získať od operačného systému, ktorý ich prideľuje. Pri cieľových portoch je rovnako zvykom využívať známe *well-known porty*, v prípade neštandardných aplikácií je možné využiť službu tzv. *daemona*. Daemon je proces bežiaci na cieľovej stanice na známom well-known porte. **Daemon** proces vie o konkrétnych aplikáciách a číslach portov, na akých sú dostupné a preto vie odpovedať na prípadný dotaz *„Na akom porte beží u teba aplikácia XY?“*.

formát UDP správy



- polia *source port* a *destination port* obsahujú číslo portu
- pole *source port* je nepovinné (ak je nepoužité, tak sa vyplnení nulami)
- pole *Length* obsahuje dĺžku UDP datagramu, vrátane Headra v oktetoch (minimum je 8 – práve dĺžka hlavičky bez dát)
- pole *checksum* je nepovinné – ak je nepoužité, tak sa vyplní nulami
  - ak je použitý a výsledkom je nula, tak sa pole vyplní samými jednotkami – lebo je použitý doplnkový kód, ten má dve reprezentácie pre hodnotu nula – samé jednotky a samé nuly.
  - vypočítava sa z celého datagramu, pričom sa pred neho ešte predradí tzv. **pseudoheader**, ktorý okrem iného obsahuje zdrojovú a cieľovú IP adresu (pozri Pseudoheader pri TCP protokole). Je to síce porušenie čistého vrstvového modelu, ale je to funkčné a zvyšuje to bezpečnosť prenosu.

Protokol TCP
(Transmission Control Protocol)

všeobecne

Protokol TCP zabezpečuje spoľahlivú, **stream**-orientovanú transportnú službu nad protokolom IP. (UDP poskytuje nespoľahlivú, packet-delivery službu bez spojovej orientácie). TCP je popri IP protokole najdôležitejší protokol z rodiny TCP/IP, vo všeobecnosti je však nezávislý od IP protokolu (môže rovnako dobre fungovať aj na inom sieťovom protokole). Stream-orientovaná služba znamená, že tento protokol umožňuje prenos toku Bajtov (stream). Pre vyššie vrstvy, často priamo aplikácie vytvára predstavu, že medzi ním a cieľovou IP stanicou existuje kanál, ktorý umožňuje prenášať tok Bajtov.

Zabezpečiť spoľahlivý prenos údajov vo forme toku dát medzi aplikáciami z bodu A do B použitím protokolu IP, resp. UDP by bolo zložité, pretože by bolo potrebné vyrovnávať sa so stratami, duplikáciami paketov a pod. Preto na transportnej vrstve existuje TCP protokol, ktorý je **spojovo orientovaný** (virtual circuit orientation – analógia telefónneho spojenia) a poskytuje stream-orientovanú službu prenosu toku dát v Bajtoch. Vyrovnáva sa s nárazmi dát (buffruje) a sekanie na IP pakety robí neviditeľné pre komunikujúce aplikácie. Zabezpečuje **full-duplex prenos** (efektívne potvrdzovanie cez piggybacking), samozrejme **aj half-duplex** prenos.

Ďalej implementuje metódy riadenia zabezpečenia (**error-control**). V TCP sú použité metódy ARQ, konkrétne pomocou pozitívneho potvrdzovania a prípadných retransmisií. Každý vyslaný **segment** musí byť potvrdený správou ACK, ak nie, tak vyprší timeout a nasleduje retransmisia. Protokol TCP implementuje aj funkciu **riadenia toku**, pričom je použitá metóda *sliding window s premenlivou dĺžkou okna*.

Protokol TCP bol je definovaný v RFC793 z roku 1981. Jedná sa o relatívne rozsiahlu špecifikáciu, ktorá definuje spoľahlivú transportnú službu na prenos toku dát (4. vrstva v RM OSI). Definuje **vytváranie, rušenie a manažovanie spojení, flow control, error-control**. Samotná špecifikácia špecifikuje protokol, nie služby a teda ani interfejs. Ten je ponechaný na konkrétnu implementáciu (RFC 793 iba dáva odporúčania, čo by mal interfejs poskytovať). Podobne ako UDP sprístupňuje svoje služby cez tzv. porty (protocol ports). Na rozdiel od UDP je potrebné pred samotným prenosom naviazať spojenie, a preto musí byť jedna stanica v stave *passive open* (t.j. čakajúca na spojenie) a jedna v stave *active open* (inicializátor spojenia)
Konkrétne spojenie (virtual connection) je jednoznačne identifikované dvomi koncami, preto môže byť jeden port na jednom hostovi zdieľaný pre viaceré spojenia.

Príklad 1: web-server na jednej stanici, 80-ty port a môže mať viacero nezávislých spojení a tie sú jednoznačne identifikované druhou stranou.
*Príklad 2: tabuľka predstavuje **tri úplne iné spojenia** (aj keď napr. zdroj je rovnaký pre viacero)*

spojenie	zdroj		cieľ	
A	170.20.20.20	2222	10.10.10.10	80
B	170.20.20.20	2222	10.10.10.10	8080
C	170.20.20.21	33333	10.10.10.10	80

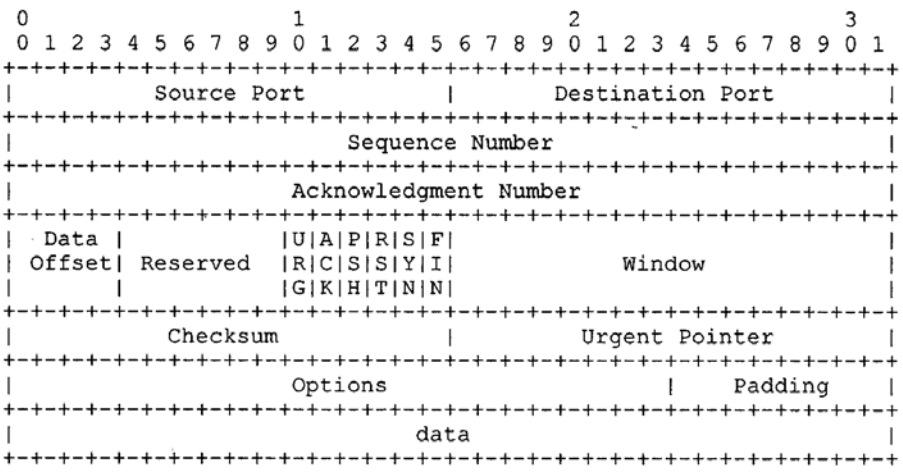
Princíp:
TCP pracuje s dátami ako s tokom Bajtov. Pretože využíva služby protokolu IP, musí tok Bajtov rozsekať na tzv. **segmenty** (obyčajne posiela jeden segment v jednom IP pakete). Metódu *sliding window s premenlivou dlžkou okna* používa jednak na efektívne využitie prenosového pásma pri riadení chybovosti (ARQ error-cotrol) a na end-to-end riadenie toku (flow-control), pričom pracuje na úrovni Bajtov, t.j. aj veľkosť okna pri *sliding window* **je v Bajtoch, nie v segmentoch**, paketoch alebo iných jednotkách.

Pretože používa okno s premenlivou dĺžkou môže riadiť tok, t.j. prispôsobovať rýchlosť posielania dát cieľovej stanici podľa toho, ako ich prijímacia stanica potvrdzuje. Každé ACK, ktoré potvrdí počet prijatých Bajtov zároveň obsahuje aj informácia o veľkosti prijímacieho buffra, t.j. koľko dokáže prijímacia stanica prijať (tzv. *window advertisement*). Na základe toho môže vysielač zväčšit, resp. zmenšiť svoje okno. Metóda *sliding window s premenlivou dĺžkou okna* síce primárne **rieši zahltenie konca**, ale dobre implementovaná sa dokáže **vysporiadať aj so zahltením** vo vnútri **siete**, napr. na routroch.

Pri zahltení v sieti začne narastať oneskorenie (delay), príp. dôjde až k zahadzovaniu paketov. Koncové zariadenia samozrejme o zahltení routra nevedia, pre nich je to len **nárast oneskorenia**, resp. strata paketov a pri TCP **dôvod na retransmisie**. Takto môže ale dôjsť k **úplnému kolapsu** siete. Zabrániť sa tomu dá tak, že odosielateľ spomalí vysielanie, keď mu začne narastať delay, teda čas čakania na potvrdenie. Pokiaľ dôjde

k retransmisii, zmenší sa vysielacie okno na polovicu a zdvojnásobia sa časovače pre segmenty čakajúce na potvrdenie. Takto sa spomaľuje rýchlosť vysielania až kým nenastane úspešný prenos (príde potvrdenie). Túto techniku nazývame **Multiplicative Decrease** Congestion Avoidance. Aby nedošlo k nestabilite, štartovanie nemôže byť také rýchle ako pokles. Preto vysielač začína zvyšovať počet vysielaných segmentov (zväčšuje veľkosť okna) po jednom segmente. Nazývame to **Slow-Start** (additive).

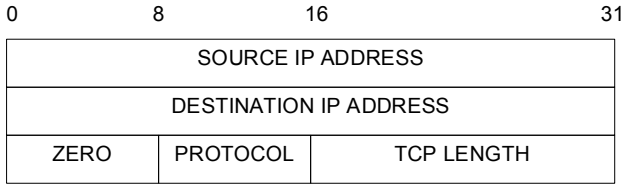
formát TCP segmentu



- každý segment má dve časti – hlavička (TCP header) a samotné dáta
- *sequence number* – pozícia posielaných dát v toku Bajtov u odosielateľa
- *ACK number* – číslo Bajtu, ktoré prijímač očakáva (potvrdzuje teda prijatie o 1 Bajt nižší)
- *Data offset* – (Header Length) – dĺžka TCP hlavičky v 32-bitových násobkoch (pretože Options môžu byť rôzne dlhé, aby bolo možné určiť začiatok dát)
- Code bits:
  - URG – *Urgent pointer field*
    - označuje urgentné data pre prijímaciu aplikáciu
    - prijímač musí byť informovaný o týchto datach okamžite (napr. interruptom), bez ohľadu na pozíciu v streame
  - ACK – *Acknowledgement field*
  - PSH – tento TCP segment vyžaduje tzv. PUSH
  - RST – Reset connection
  - SYN – Synchronize sequence numbers
  - FIN – odosielateľ dosiahol koniec posielaných dát
- *Window* – prijímač oznamuje, koľko dát je schopný prijať (veľkosť prijímacieho buffra)
- *URGENT POINTER* – označuje pozíciu v segmente, kde *urgent data* končia

- *Options* – maximum segment size option (MSS)
  - aby si pomalšie stanice mohli dohodnúť maximálnu veľkosť segmentu
  - na prispôsobenie sa LAN sieťam, aby veľkosť TCP segmentu neprekročila maximálnu veľkosť, ktorú je schopná daná LAN sieť preniesť v jednom rámci
  - celkovo stanovenie MSS v sieti je problematické – malé segmenty využívajú sieť neefektívne, veľké nútia IP k fragmentácii. Navyše stačí stratiť jeden fragment IP datagramu, a vyslanie celého segmentu sa musí retransmitovať

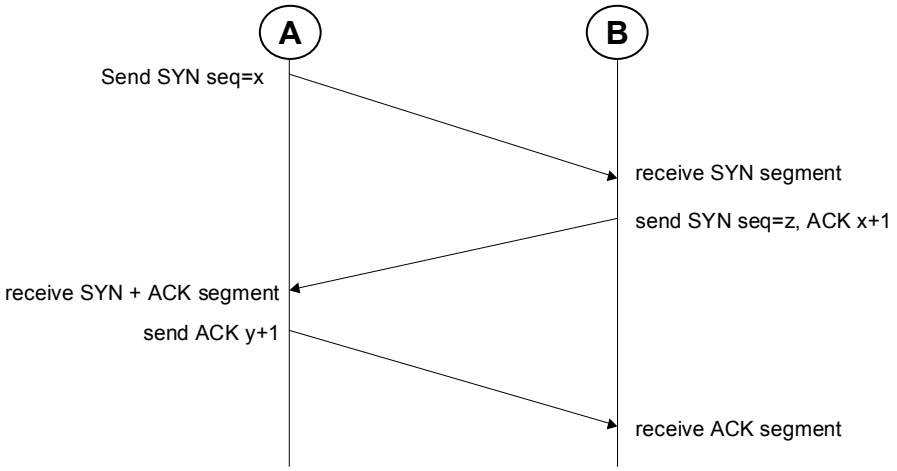
CHECKSUM
- výpočet je obdobný ako pri UDP, používa pseudoheader



- pole *protocol* obsahuje číslo/kód ktorý používa spodnejší protokol (tu IP) na označenie protokolu TCP (6)
- *TCP length* - dĺžka segmentu vrátane hlavičky

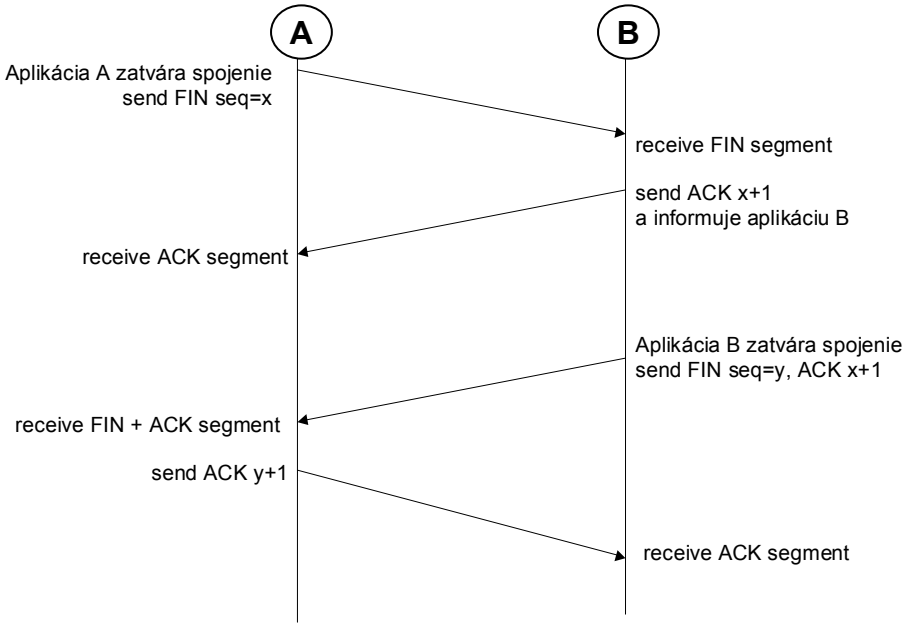
Vytvorenie TCP spojenia

- používa sa tzv. *three-way handshake*, t.j. nadviazanie spojenie na 3 kroky.



Ukončenie TCP spojenia

- používa sa tzv. modifikovaný three-way handshake mechanizmus
- modifikovaný preto, lebo v podstate ide o dve separátne spojenia (pri full-duplexe)



Reset TCP spojenia

Za osobitných okolností, kedy nie je možné (alebo vhodné) ukončiť spojenie štandardne, môže jedna strana poslať segment s nastaveným RST bitom. Vtedy druhá strana okamžite zruší spojenie a informuje aplikáciu o zrušení spojenia. Spojenie teda padá okamžite o oboch smeroch, všetky buffre sú vyprázdnené.

Rezervované TCP port numbers

- podobne ako pri UDP aj tu sú definované tzv. well-known ports
- pôvodne bolo rezervovaných 256, teraz 1024
- ostaté sú ponechané na používanie (operačným systémom, aplikáciou ...)
- aj keď to nie je nutné, ale je dobrým zvykom, že aplikácie, ktoré sú implementované aj na UDP aj na TCP používajú rovnaké číslo portu

Výkonnosť TCP

Aj keď sa protokol TCP zdá zložitý, je relatívne efektívny. Napríklad na 10Mbps Ethernete je možné dosiahnuť 8Mbps prenosovú rýchlosť.