#4 TCP/IP – transportná vrstva

Úvod – transportná vrstva

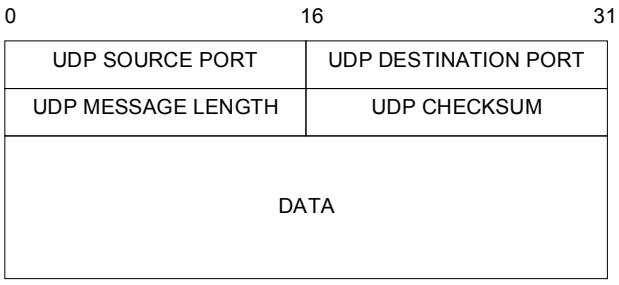
- využíva služby sieťovej vrstvy (napr. IP)
- poskytuje vyšším vrstvám, resp. priamo aplikáciám (user procesom) transportnú službu
- môže byť CONL a connection oriented, spoľahlivá/nespoľahlivá – v každom prípade však poskytuje vyšším vrstvám služby nezávislé od konkrétnej siete pod ním
- funkcie, ktoré realizuje možno zhruba prirovnať k funkciám DataLink vrstvy (bit retransmission, error control, flow control) – avšak ďaleko zložitejšie (oveľa zložitejšia štruktúra siete oproti „jednému kábliku“ pri napr. HDLC)
- tieto služby poskytuje prostredníctvom interfejsu – výmenou servisných primitív a cez service access pointy – čiže znovu je potrebné zadefinovať adresáciu

	Transport
	Network
	Dl
	Phy

UDP – User Datagram Protocol

- je protokol z rodiny TCP/IP
- je to transportný protokol – veľmi jednoduchý, poskytuje základný mechanizmus, akým môžu aplikácie prenášať datagramy
- využíva IP protokol, poskytuje nespoľahlivý, connectionless prenos datagramov
- je teda definovaný nad IP, ktorý používa na adresáciu IP adresy – tieto jednoznačne určujú hosta, resp. jeho interface v IP internete (virtuálnej siete zloženej z rôznych druhov sietí spojených pomocou IP protokolu na tretej vrstve)
- UDP protokol pridáva adresovanie na transportnej vrstve (T-SAPs) – a definuje tak prístupové miesta v rámci jedného hosta (fyzického interfejsu hosta) a viacerými aplikáciami
- tieto SAPs nazývame *protocol ports*
- aplikácie teda komunikujú pomocou User datagramov, ktoré sú prístupné cez tieto porty
- UDP teda k funkcionalite IP pridáva len možnosť rozlíšiť viacero destinácií na jednom hostovi, so samotným zabezpečením prenosu (duplikáty paketov, straty, poruchy, poprehadzovanie poradia, oneskorenie, ...) sa musí vysporiadať vrstva nad UDP (obyčajne priamo aplikácia)

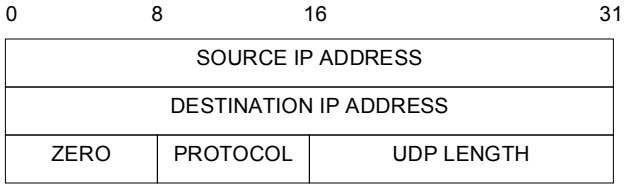
formát UDP správy



- porty slúžia na multiplexovanie datagramov medzi procesmi
- source port je optional (ak nepoužitý, tak vyplnený 0-lami)
- Length – dĺžka UDP datagramu, vrátane Headra v oktetoch (minimum je 8 – akurát header)
- checksum
  - je optional – ak nie je použitý, tak samé 0-ly
  - (ak je výsledok nula, tak samé jednotky – lebo je použitý doplnkový kód, ten má dve reprezentácie pre nulu)
  - ráta sa z celého datagramu, pričom sa pred neho ešte predradí Pseudoheader

- toto je samozrejme silné porušenie vrstvového modelu, lebo vyššia vrstva si musí pýtať info od spodnej – musí s ňou úzko spolupracovať
- je to ale ústupok oproti „peknému“ modelu, za výhodu, ktorou je jednoznačná identifikovateľnosť správneho doručenia datagramu

Pseudo-Header



- pole Protocol obsahuje IP kód pre UDP protokol, t.j. hodnotu 17_{DEC} (11_{HEX})
- takýto psedoheader sa pred samotným počítaním checksumu predradí pred samotný datagram,
- ak je datagram nezaokrúhlený na 16-bitové slová, tak sa pridá jeden oktet s nulami
- spočíta sa checksum, vloží sa do datagramu
- !!! pseudoheader, ani prípadný oktet s nulami na konci sa neposielajú!!! – slúžia iba na výpočet kontrolnej sumy (tá je aj tak optional, takže to ani netreba:)

Porty

- každá aplikácia musí od operačného systému dostať port
- tento je ďalej uvádzaný v odchádzajúcich datagramoch ako source port (je optional, je to len info, aby príjemca vedel odpovedať – ale UDP prenos je len jednosmerný prenos jedného datagramu)
- pri príjme, UDP prijíma datagramy a buffruje ich na jednotlivé porty (do queues)
- pokiaľ dostane datagram s cieľovým portom, ktorý je nie je používaný (nemá ho kam zaradiť), tak ho zahodí a pošle ICMP správu – port unreachable. Keď je ale port plný (teda fronta daného portu), tak datagram zahodí tiež.

- ako získať čísla portov?
  - source
    - sú tzv. well-known – pridelené (napr. HTTP má 80)
    - pri vysielaní prideľuje systém
  - destination
    - well-known porty (zase viem, že HTTP má 80 – tak idem rovno naň)
    - spýtam sa démona – ten je well-known – a ten mi pošle žiadaný cieľový port

TCP – Transmission control protocol

všeobecne - spoľahlivá Stream Transportná služba

- (UDP bola nespoľahlivá packet-delivery služba bez spojovej orientácie)
- 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)

- pre mnohé aplikácie je potrebné zabezpečiť spoľahlivý prenos údajov –ako tok dát z bodu A do bodu B
- používať priamo IP, resp. UDP by bolo zložité (vyrovnávať sa so stratami, duplikáciami... paketov)
- preto je potrebné, aby bola pre aplikácie poskytovaná spoľahlivá prenosová (transportná) služba, ktorá má nasledovné vlastnosti:
  - je stream orientovaná – tok dát (obyčajne v Bajtoch) medzi dvomi entitami
  - virtual circuit orientation – analógia telefónneho spojenia (samozrejme „iba“ virtuálna)
  - vyrovnáva sa s nárazmi dát – buffruje - a sekanie na pakety robí neviditeľné pre komunikujúce entity. Delenie dát je úplne na transportnej vrstve, kvôli efektivite môže však dlho čakať na naplnenie svojho buffra – v nežiadúcich prípadoch (napr. pri telnete) poskytuje PUSH mechanizmus
  - nepozná štruktúru toku – správa sa k toku dát ako k neznámemu toku Bajtov
  - zabezpečuje full-duplex (efektívne potvrdzovanie cez piggybacking), samozrejme aj half-duplex.

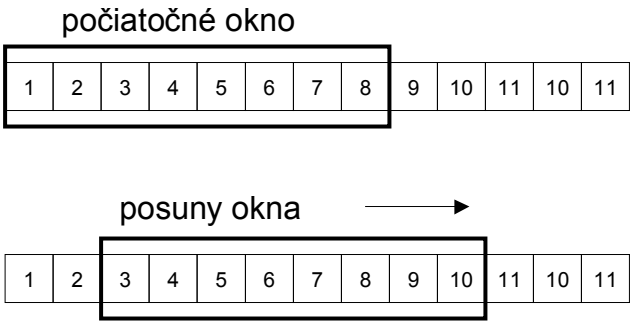
spoľahlivosť (error-control)

- spoľahlivý stream delivery zabezpečuje pomocou positive acknowledgement with retransmission
- (t.j. každý vyslaný paket musí byť potvrdený ACK, ak nie, tak vyprší timeout a nastane retransmisia
- samozrejme, každý paket a následne jeho ACK sú osobitne číslované kvôli identifikácii
-

Sliding Window

- sieť (IP sieť) môže mať aj veľké delay – preto S&W by bolo neefektívne
- preto sa používa tzv. sliding window
  - window – t.j. sender môže vyslať toľko paketov, koľko má veľkosť okna bez toho, aby čakal na potvrdenie
  - sliding – okno sa posúva (kĺže sa) vždy po potvrdení prvého paketu čakajúceho na ACK
- sender si teda v buffri drží tri druhy paketov – už vyslané a potvrdené (naľavo od okna), ešte nevyslané (napravo od okna) a vyslané ale čakajúce na potvrdenie (v okne).

obrázok – sliding window (okno) o veľkosti 8



TCP – Transmission control protocol

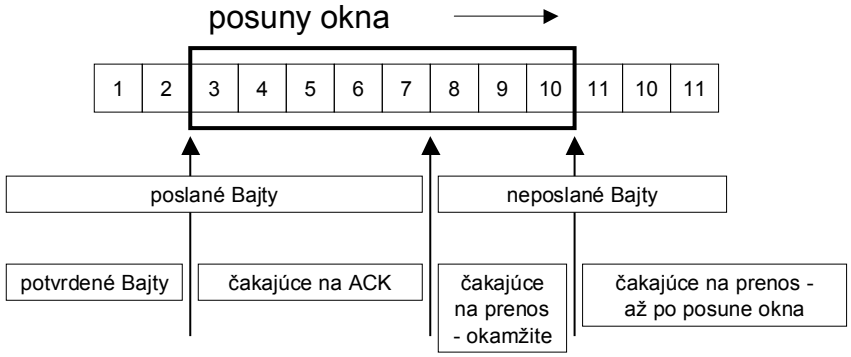
- definovaný v RFC793 z roku 1981
- relatívne rozsiahly
- definuje spoľahlivú transportnú službu na prenos toku dát – 4. vrstva v RM OSI
- definuje vytváranie, rušenie a managovanie spojení, flow control, error-control
- špecifikuje protokol, nie služby – t.j. intrerface je ponechaný na konkrétnu implementáciu (iba dáva odporúčania, čo by mal interfejs poskytovať)
- podobne ako UDP sprístupňuje svoje služby cez tzv. porty (protocol ports)
- konkrétne spojenie (virtual connection) je jednoznačne identifikované dvomi koncami, preto môže byť jeden port na jednom hostovi zdieľaný pre viaceré spojenia (napr. web server – jeden host, 80-ka port a má viacero nezávislých spojení – tie sú jednoznačne identifikované druhou stranou)

príklad – tabuľka predstavuje tri **úplne iné spojenia** (aj keď napr. zroj je rovnaký pre viacero)

spojenie	**zdroj**		**cieľ**	
prvé	170.20.20.20	1220	10.10.10.10	1320
druhé	170.20.20.21	1250	10.10.10.10	1320
tretie	170.20.20.20	1220	10.10.10.10	1330

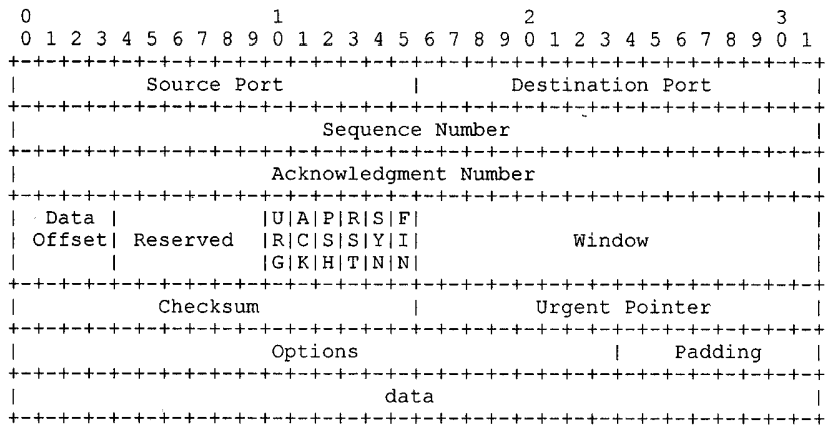
- na rozdiel od UDP je potrebné pred samotným prenosom naviazať spojenie, a preto musí byť jedna stanica v stave *passive open* (t.j. čakajúca na spojenie) a jedna v stave *active open* (inicializátor spojenia)

- TCP pracuje s datami ako s tokom Bajtov
- keďže ich má preniesť cez IP, tak ich musí rozsekať - na tzv. **segmenty** (obyčajne posiela jeden segment v jednom datagrame)
- používa sliding window – jednak na efektívne využitie prenosového pásma (oproti S&W) a ešte na end-to-end flow-control
- sliding window pracuje na úrovni Bajtov (t.j. aj veľkosť okna je v Bajtoch, nie v segmentoch, paketoch alebo podobne)
- sender si teda drží tri pointre - podľa obrázka



- takéto okno si udržuje (skladá si ho) aj prijímateľ, pokiaľ ide o full-duplex, tak sú takéto okná vytvorené v oboch smeroch (dokopy teda 4)
- TCP používa premenlivú veľkosť okna – na flow-control
- preto každé ACK, ktoré potvrdí počet prijatých Bajtov zároveň obsahuje aj info o prijímacom buffri, t.j. koľko dokáže prijímacia stanica prijať (tzv. window advertisement)
- na základe toho môže vysielač zväčšit, resp. zmenšiť svoje okno (zmenšuje ho samozrejme pri pohybe, nie kým už čaká na potvrdenie vyslaných)
- extrém – receiver pošle window advertisement = 0, t.j. pozastaví vysielanie, a obnoví ho poslaním nenulového okna
  - sender to môže ignorovať, pokiaľ potrebuje poslať URGENT data
  - pre prípad, že by sa stratil obnovovací, nenulový paket od prijímateľa (nastal by tzv deadlock), vysielač v pravidelných intervaloch posiela „overovací paket“, či je ešte stále receiver neschopný prijímania
- sliding window teda rieši zahltenie konca, ale dobre implementované sa dokáže vysporiadať aj so zahltením vo vnútri siete (napr. na routroch) – keď mi príde veľa zahodených, tak zmenším okno

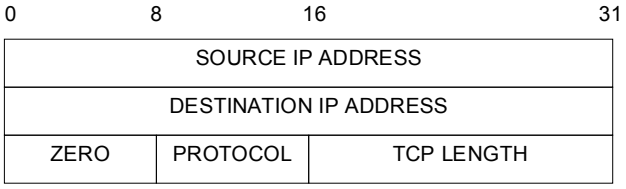
formát TCP segmentu



- každý segment má dve časti – header (TCP header) a samotné data
- sequence number – pozícia posielaných dát v senderovom toku Bajtov
- ACK number – číslo Bajtu, ktoré prijímač očakáva (potvrdzuje teda prijatie o 1 Bajt nižší)
- Data offset – Header Length – dĺžka headra v 32-bitových násobkoch (pretože Options môžu byť rôzne dlhé, aby bolo možné určiť začiatok dát)
- Code bits:
  - URG – Urgent pointer field is valid
    - 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 is valid
  - PSH – This segment requires a push
  - RST – Reset connection
  - SYN – Synchronize sequence numbers
  - FIN – sender has reached end of its byte stream
- Window – prijímač oznamuje, koľko dát je schopný prijať (veľkosť prijímacieho buffra)
- URGENT POINTER – označuje pozíciu v segmente, kde urgent data končia

- OPTION – maximum segmetn size option (MSS)
  - aby aj slabšie stanice mohli dojednať maximálnu veľkosť segmentu
  - na prispôsobenie sa LANom – aby sadli do MTU
  - celkovo stanovenie MSS v sieti je problematické – malé segmenty využívajú sieť neefektívne, veľké nútia IP k fragmentácii – a stačí stratiť jeden fragment IP datagramu, a celý segment sa musí retransmitovať

CHECKSUM
- výpočet je rovnako ako pri UDP –sa používa pseudoheader



- hodnota, ktorou spodnejší protokol (tu IP) označuje TCP – teda = 6
- TCP length - dĺžka segmentu vrátane headra

Potvrdzovanie a retransmisie

- TCP posiela data v segmentoch, ktoré majú premenlivú dĺžku
- jednotka potvrdzovania je Bajt, retransmitovaný segment môže obsahovať viac dát ako originál
- preto sa potvrdzuje na pozíciu v roku Bajtov – v buffri (poskadanom v prijímači)
- pakety, a teda segmenty môžu byť stratené – preto môžu vznikať rôzne diery
- prijímač potvrduje ale iba ucelenú, kontinuálnu časť od začiatku prijímania toku

- oznamuje teda, koľko súvisle pozbieral – posiela číslo očakávaného Bajtu
- neoznamuje ale, že hneď za ním má prijatých niekoľko ďalších segmentov – a to môže byť neefektívne, lebo kvôli malej diere sa stane, že vysielač retransmitne aj to, čo nemusel, resp. keby zakaždým retransmitol len kúsok a čakal na potvrdenie, tak by to bolo znovu neefektívne
- prijímač teda posiela málo informácií :(
-

Timeouty a retransmisie

- vždy keď pošle segment, tak si nastaví timer pre ACK
- keďže TCP je navrhnuté na dynamicky sa meniace parametre siete (medziľahlých sietí), musia sa aj hodnoty timeoutov prispôsobovať – nemožno ich nastaviť na pevno
- preto používa *addaptive retransmission algorithm,* t.j. sleduje každú konekciu a prispôsobuje timeouty
- meria tzv. Round Trip Sample – čas od vyslania segmentu do prijatia jeho ACK
- z tohto vypočítava priemerný Round Trip Time **RTT** (váhovaním s pôvodným RTT) nasledovne:

$$\mathbf{RTT = (\alpha * Old\_RTT) + ((1- \alpha) * New\_Round\_Trip\_Sample)}$$

- pričom hodnota α určuje, ako rýchlo sa bude priemerná hodnota prispôsobovať zmenám

- na základe takéhoto zmeraného času je teda možné nastavovať hodnotu timeoutov, spravidla na niekoľkonásobok tohto času

$$\mathbf{Timeout = \beta * RTT}$$

- Pôvodné špecifikácie odporúčali β=2, sú však lepšie techniky

Meranie Round Trip Sample

- teoreticky jednoduché – rozdiel dvoch nameraných časov
- problém nastane pri retransmisiách – kedy nie je možné určiť, či ACK prišlo k originálu, resp. k retransmitnutému segmentu – prolém!
- keď si poviem, že ACK vždy patrí originálu, tak mi meraný delay bude neustále narastať
- keď si poviem, že ACK patrí k posledne vyslanému, tak si nepravdivo skrátim hodnotu timeoutu, a samozrejme bude dochádzať k retransmisiám - a ustabilizuje sa to v takom stave, že TCP bude posielať každý segment 2x

riešením je Karnov Algoritmus

- základná idea - nenastavovať RTT podľa časov meraných z retransmitovaných segmentov
- samozrejme, v prípadoch, kedy sa veľmi zhorší delay, a teda začalo by dochádzať k retransmisiám by sa Timeout nikdy neprispôsobil
- preto sa to rieši tzv. *timer backoff* stratégiou, kedy sa normálen RTT ráta z dobre prenesených segmentov a následne sa upravuje Timeout ako je uvedené hore, a pri retransmisii sa Timeout upraví tak, že sa natvrdo zvýši (napr. zakaždým na 2-násobok pôvodného – menej ako 2 vedie k nestabilite) – a takto ho zvyšuje, až kým nedosiahne korektný prenos bez retransmisie, resp. nastavený strop (teoretický maximálny delay v celej sieti)

reakcie na veľké variácie v Delay-och

- podľa teórie queuingu je variácia v delay-i (σ) závislá od záťaže siete (L) podľa vzťahu (1/(1-L))
- Karnov algoritmus opísaný hore (β=2) funguje dobre asi do 30% zaťaženosti siete
- preto je potrebné hodnotu β určovať zložitejšie, a to:

DIFF = Round_Trip_sample – Old_RTT	rozdiel novej vzorky od priemeru
Smoothed_RTT = Old_RTT + δ * DIFF	upravené RTT – iba o zlomok zmeny
DEV = Old_DEV + ρ (|DIFF| - Old_DEV)	
Timeout = Smoothed_RTT + η * DEV	

$\delta = 1/2^3$
$\rho = 1/2^2$, resp. $1/2^3$
$\eta = 2$, resp. 4

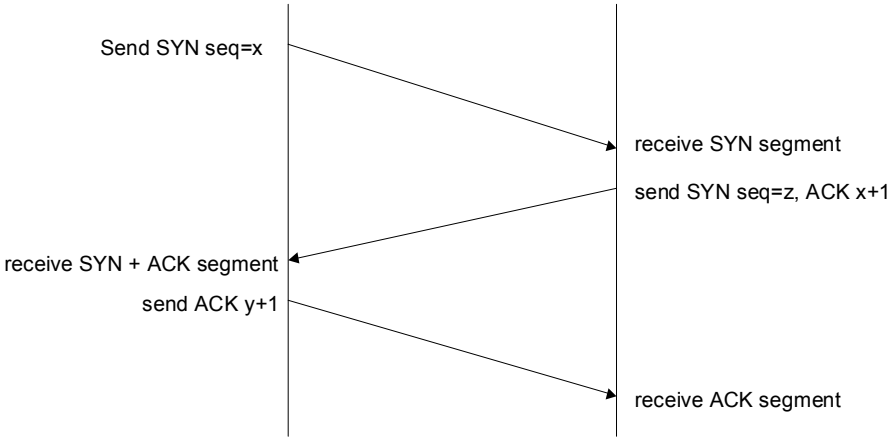
Reakcia na zahltenie

- TCP sa musí vedieť vysporiadať aj so zahltením v sieti (napr. na routroch)
- pri zahltení na routri začne narastať delay, príp. dôjde až k zahadzovaniu paketov
- koncové zariadenia samozrejme o zahltení routra nevedia, pre nich je to len nárast delay-u, resp. strata paketov a pri TCP dôvod na retransmisie – t.j. spôsobenie úplného kolapsu siete
- ako tomu predísť?
  - routre posielajú ICMP source quench (stlmenie, zhasínanie)
  - TCP sender spomalí vysielanie, keď mu začne narastať delay – 2 techniky:
- slow start a multiplicative decrease
- TCP sender si udržiava dve okná – jedno podľa advertisementov od prijímača a jedno svoje, ktorým modeluje stav siete – *congestion window*
- na vysielanie potom používa to menšie z nich
- **Multiplicative Decrease** Congestion Avoidance
  - kedykoľvek dôjde k retransmisii, zmenší sa okno na polovicu a zdvojnásob Timer-y pre segmenty čakajúce na ACK
  - toto rob až do úspešného prenosu, resp. dosiahnutie spodku okna
  - toto je teda exponenciálna redukcia trafiky
- **Slow-Start** (additive) recovery
  - aby nedošlo k nestabilite, štartovanie nemôže byť také rýchle ako pokles
  - preto s začína s jedným segmentom, a zvyšuje sa po jednom segmente – podľa toho, ako prichádzajú ACK
  - takýto nárast ale vôbec nie je „slow“, lebo keď začne jedným, a ten príde potvrdený, tak pošle 2, a tie prídu potvrdené – tak o každý z nich zväčší okno a teda pošle 4 ....
  - preto zavádza ďalšiu reštrikciu
    - akonáhle dosiahne congestion window polovicu svojej hodnoty pred zahltením, prepne do **congestion avoidance** módu a zväčší okno o 1 segment iba vtedy, keď sú všetky segmenty v okne potvrdené

Tieto techniky dokopy už dokážu veľmi efektívne reagovať na stav zahltenia.

Vytvorenie TCP spojenia

- používa sa tzv. three-way handshake
- rieši sa problém dvoch armád – t.j. nakoľkokrát treba potvrdiť cez nespoľahlivý kanál, aby sa dalo na to spoľahnúť (zrejme donekonečna) – 3x je kompromis
- v TCP to vyzerá nasledovne:



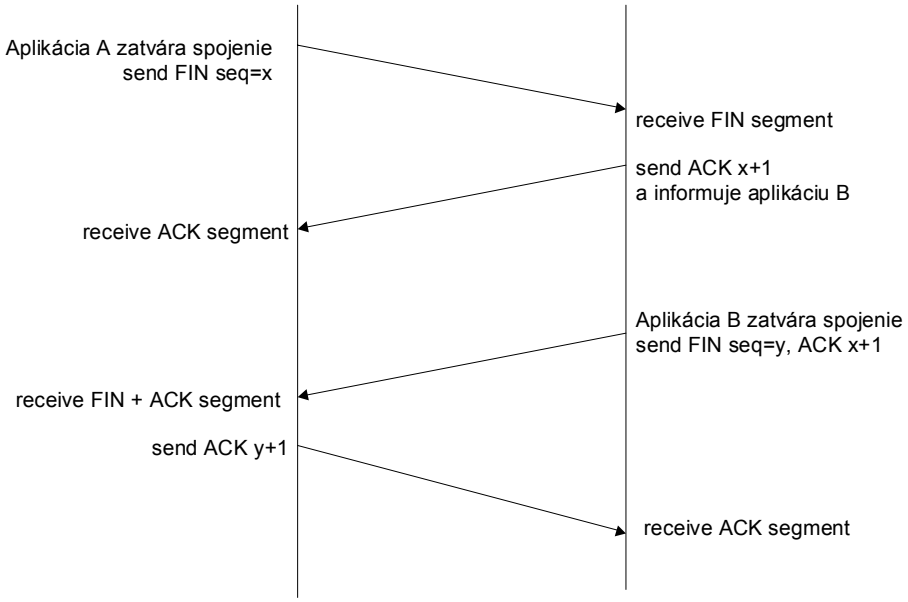
- normálne stanica B (v pravo) čaká na inicializáciu spojenia stanicou A (vľavo)
- mechanizmus je však navrhnutý aj na simultánne zahájenie spojenia
- nie je tu definovaný ani mastar, ani slave
- protokol má timeout mechanizmy – lebo ráta so stratou segmentov (využíva sa pri útokoch – half open sessions)

Sequence numbers

- trojcestné nadväzovanie spojenia zabezpečí, že obe strany sú pripravené na komunikáciu a obe strany si dohodnú inicializačné hodnoty pre číslovanie
- každá si volí svoje „náhodne“, nemôže sa začínať od začiatku – napr. pri opakovaných, resp. resetnutých, príp. nových spojeniach by mohlo dochádzať k nejednoznačnostiam
- počíta sa podľa času, pravdepodobnosť zopakovania čísla je až po nejakých 4 hodinách

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 zhodení. Spojenie teda padá okamžite o oboch smeroch, všetky buffre sú vyprázdnené.

Delenia toku dát

- TCP má voľné ruky v tom, ako delí (a následne skladá) tok dát na segmenty.
- dbá pri tom na efektívnosť prenosu (nebude posielať každé 2 Bajty, ale počká, kým ich bude do segmentu „tak akurát“)
- pre niektoré prípady (napr. pre telnet) je však potrebné vyprázdniť buffer hneď
- na to používa *push* mechanizmus
  - jednak prinúti okamžité odostanie segmentu
  - nastaví PSH bit – a tak umožní, aby prijímacia strana okamžite podala data prijímacej aplikácii

Rezervované TCP port numbers

- podobne ako pri UDP
- 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 aj na UDP aj na TCP používajú rovnaké číslo portu
-

Výkonnosť TCP

- aj keď sa zdá zložitý, dá sa dosiahnuť aj 8Mbps na 10Mbps Ethernete

Silly window Syndrome

- zdroj posiela rýchlo data
- prijímaču sa zaplní buffer a tak oznámi window až na 0
- keď uvoľní buffer o 1Byte, tak oznámi vysielaču
- ten teda pošle segment s 1-ným Bajtom:(
- a takto sa to aj môže ustáliť – sender bude posielať segmenty s 1 Bajtom
  - Dosť neefektívne!! (obronský overhead, prepočítavanie checksumov, routing ...)
- rovnako neefektívny môže byť aj sám zdroj, keď od aplikácie dostáva data po malých kúskoch, nepočká si a hneď ich balí do segmentov a posiela, alebo keď mu posiela akurát také kusy dát, ktoré sú len o málo väčšie ako maximum segment size (najhoršie o 1 väčšie:)

toto nazývame silly window sydróm (SWS)– a prvé implementácie TCP tým aj trpeli

Predchádzanie SWS

- zavádza sa určitá heuristika ako na vysielači, tak na prijímači

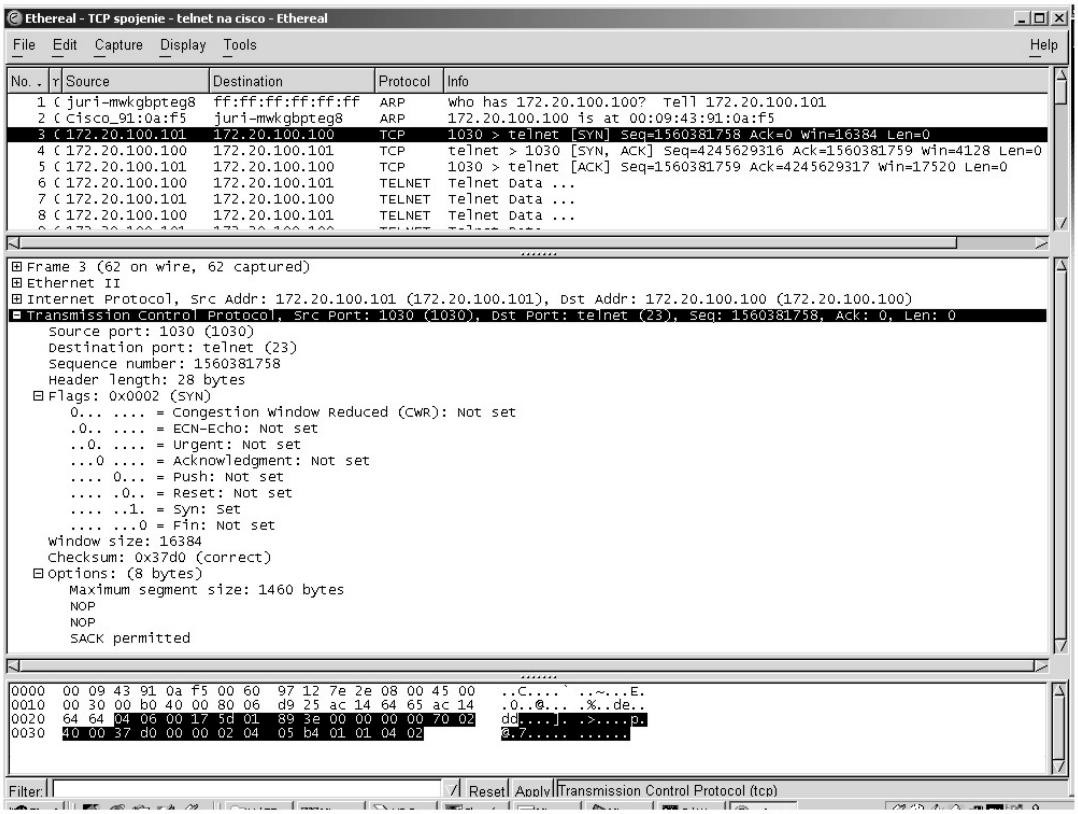
Receive-Side

- prijímač pošle nový window advertisement (po predošlom ronom nule) si počká, kým sa buffer neuvoľní aspon na 50%
- pritom sú dve možnosti
  - buď posielať ACK všetkým došlým segmentom, ale window advertisement držať na nule
  - neposielať ACKs, a potvrdiť príjem až keď bude možné poslať rozumný window advertisement
  - odporúča sa táto druhá možnosť
    - samozrejme nie je rozumné držať ACK príliš dlho, preto ich je možné pozdržať najviac o 500ms
    - okrem toho by sa mal pozdržať maximálne každý druhý ACK (aby bolo možné relatívne dobre počíta RTT)

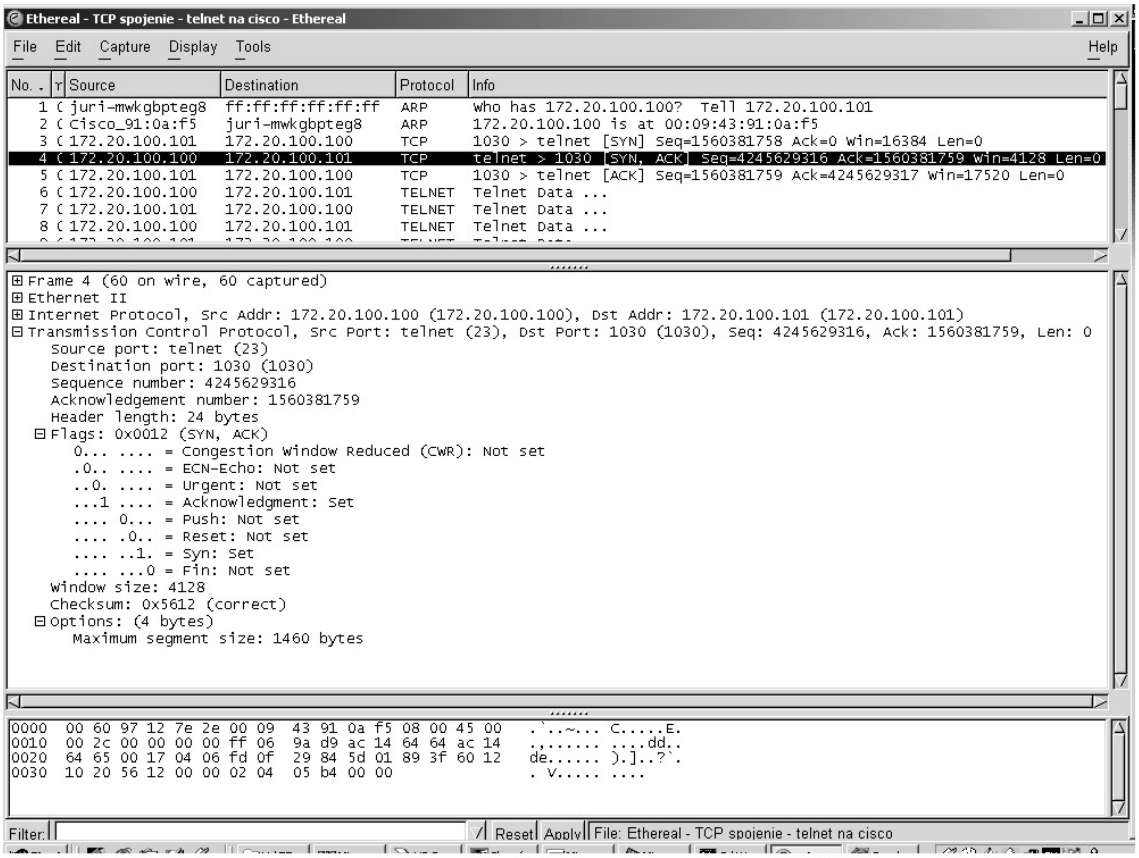
Send-Side

- technika sa volá *clumpin* (zhluknutie)
- otázka je, ako dlho môže vysielač pozdržať data v buffri, aby ich nazbieral dosť?
- pevne nastavené oneskorovacie časy nie sú vhodné – používajú sa adaptívne
- princíp je prekvapivo jednoduchý a efektívny:
  - pre dané spojenie sa neposielajú data z buffra dovtedy, pokiaľ nedosiahnu Maximums-sized segment. Ak stále ešte túto veľkosť nedosiahli, ale prišlo ACK na predošlý segment, tak sú všetky data z buffra odoslané (pošle sa menší segment)
  - pri ftp bude buffer skoro vždy plný – lebo aplikácia to nasype rýchlo – a posielajú sa segmenty s maximálnou rýchlosťou
  - t.j. pri písaní cez telnet sa ďalší znak (znaky) pošle už skôr ako buffer dosiahne veľkosť maximálneho segmentu – keď príde ACK na predošlý segment (znak)

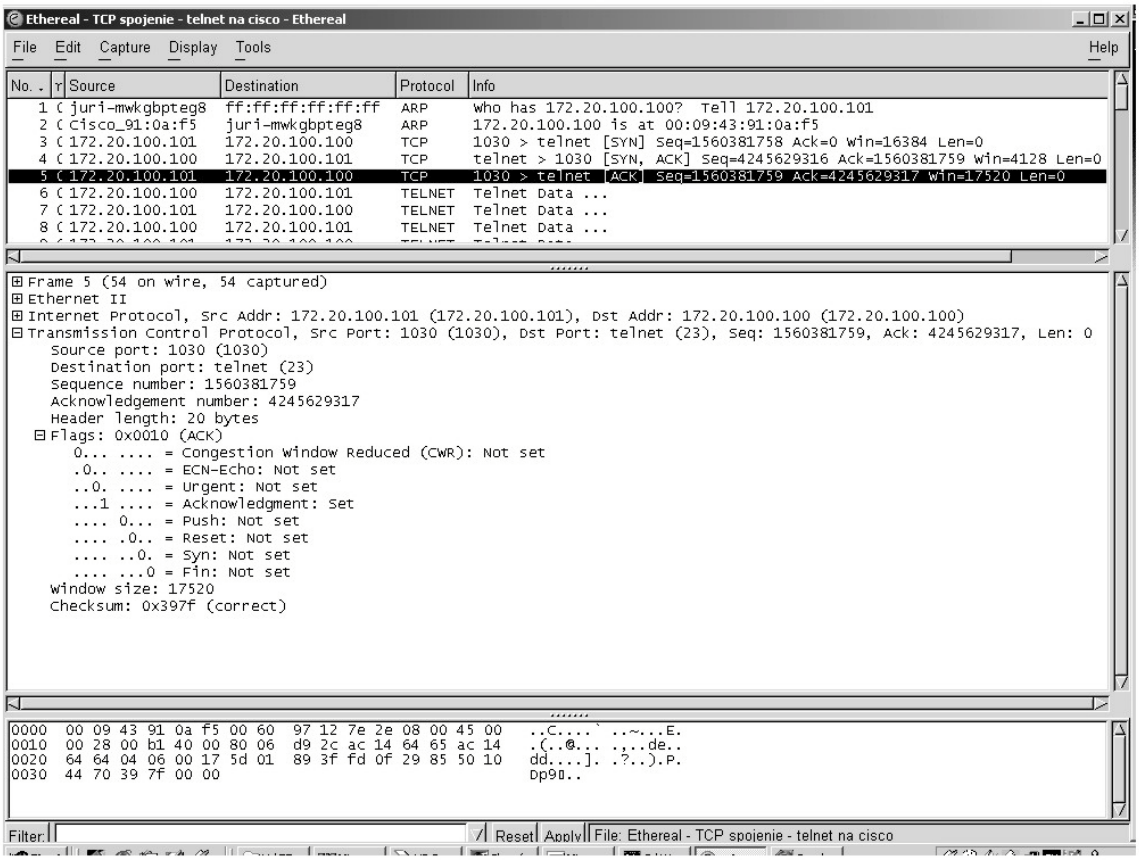
nadviazanie TCP spojenia – 1.krok (z 3-och)



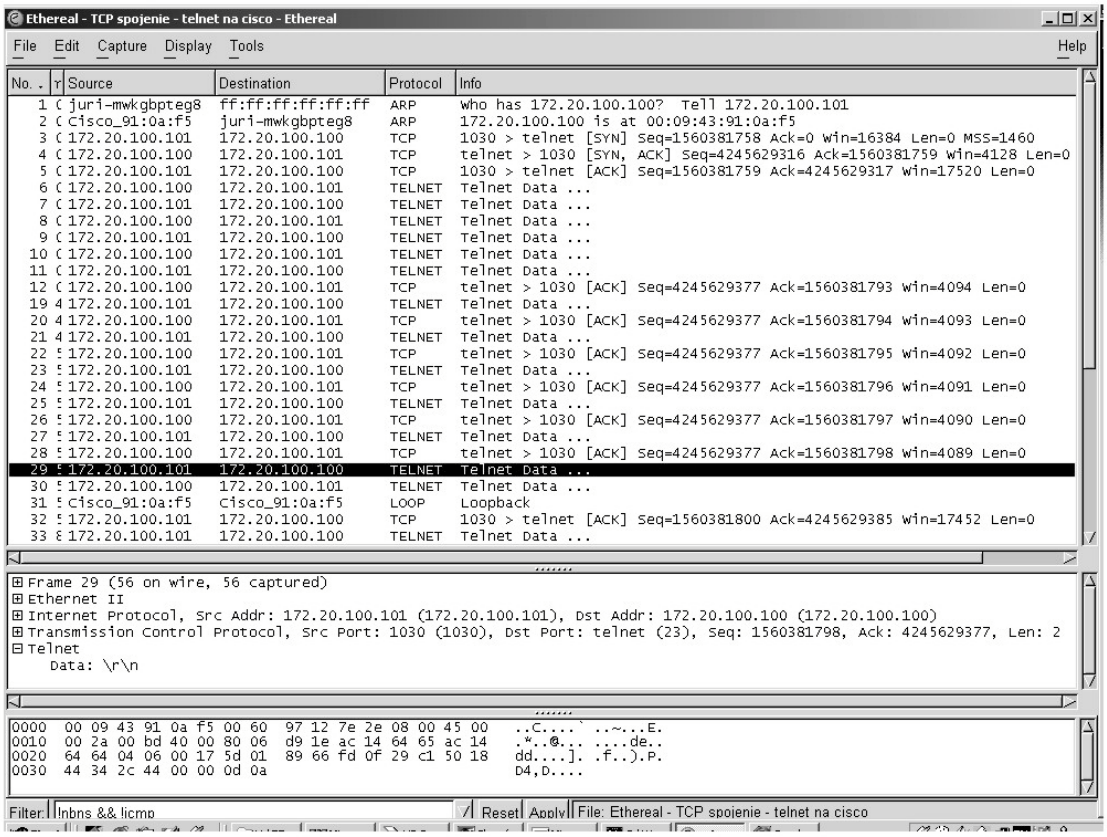
nadviazanie TCP spojenia – 2.krok (z 3-och)



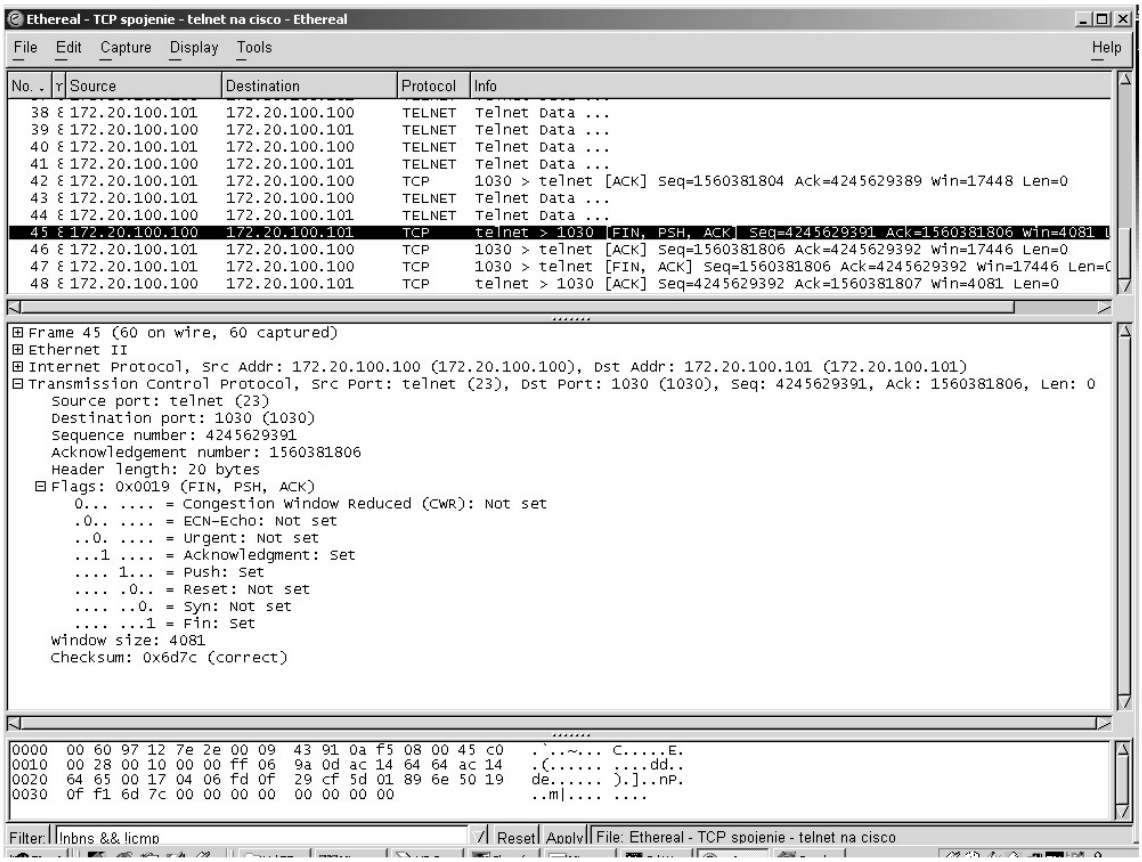
nadviazanie TCP spojenia – 3.krok (z 3-och)



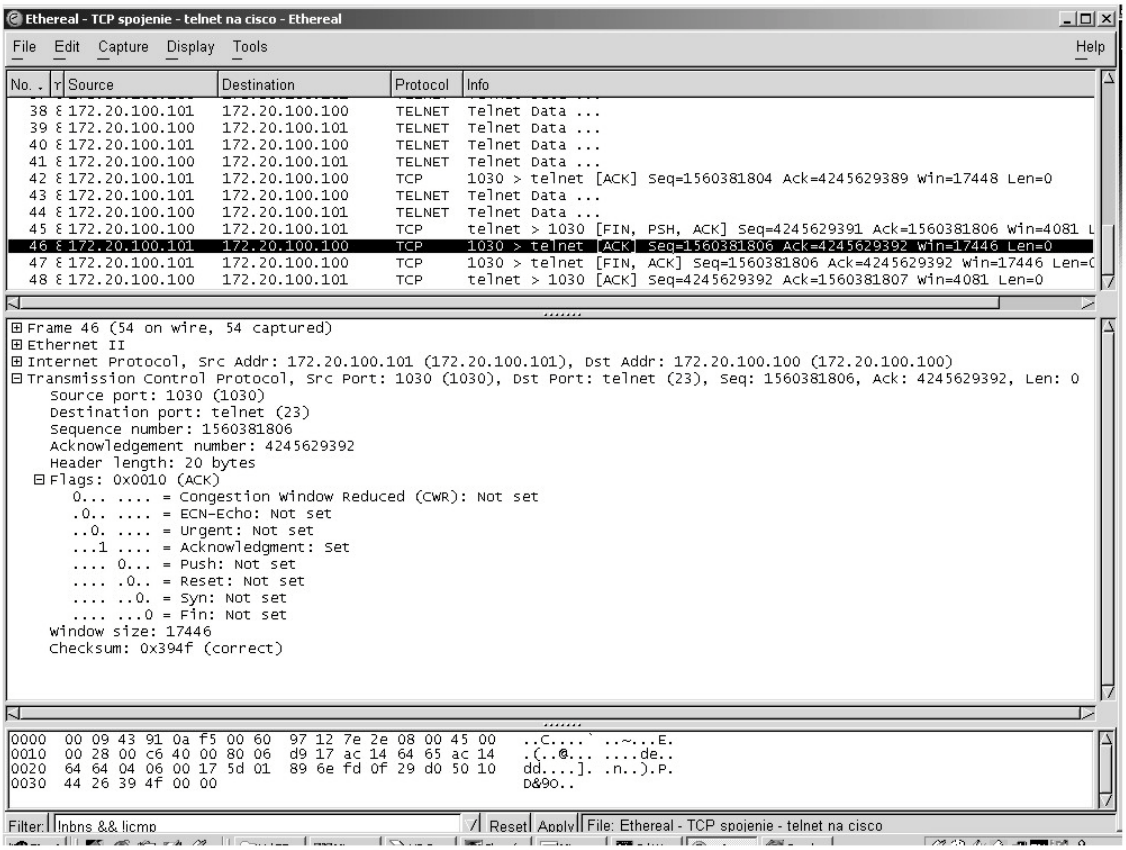
TCP spojenie – prenos dát (telnet)



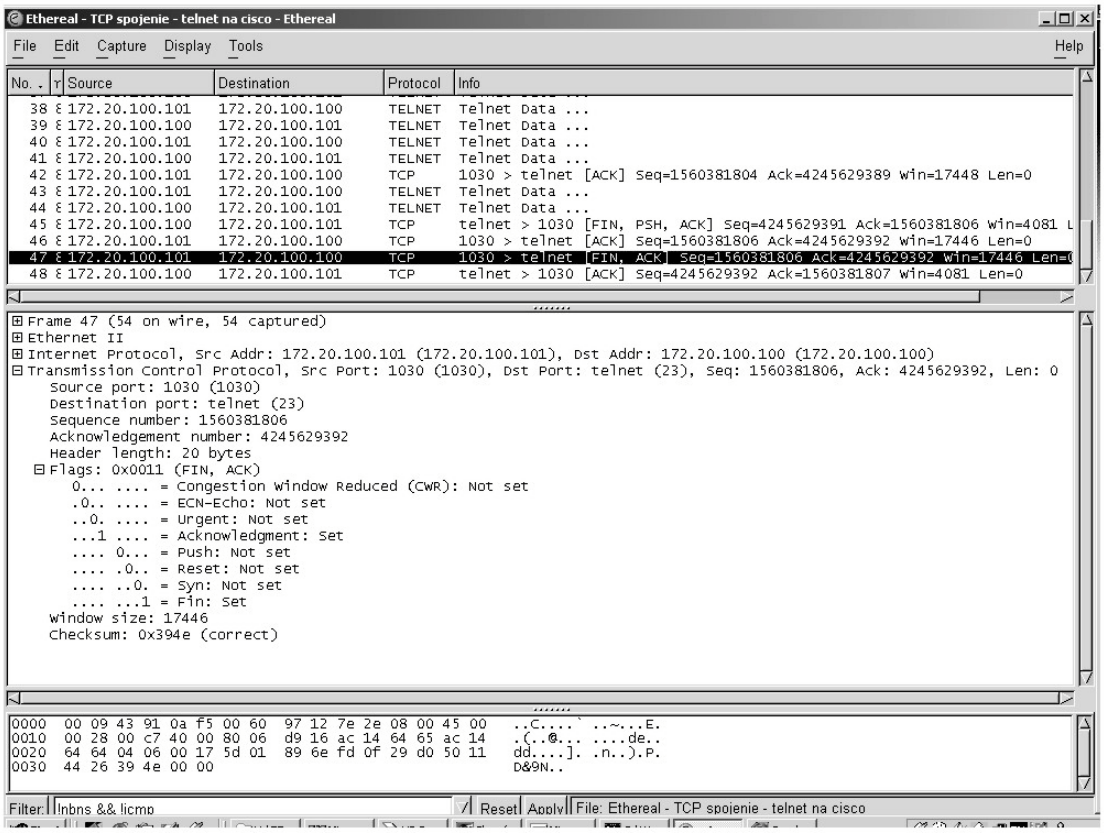
ukončenie TCP spojenia – 1.krok (zo 4-och)



ukončenie TCP spojenia – 2.krok (zo 4-och)



ukončenie TCP spojenia – 3.krok (zo 4-och)



ukončenie TCP spojenia – 4.krok (zo 4-och)