

Operační systémy a sítě

Petr Štěpán, K13133

KN-E-229

stepan@labe.felk.cvut.cz

Téma 11.

ICMP, IPv6

Transportní vrstva

IPv6

- IPv6 má adresu danou 128 bity (IPv4 pouze 32 bit)
- IPv6 vylepšuje některé vlastnosti IPv4, ale stará se pouze o vrstvu síťování
- Proč 6? Internet Stream Protocol z roku 1979 používá IP hlavičku s číslem verze 5. Další volné číslo je 6.
- Vylepšení
 - Velký prostor adres – přibližně 3.4×10^{38} různých adres
 - Multicasting – vyslání jednoho paketu na více různých počítačů
 - SLAAC – Stateless address autoconfiguration – automatická konfigurace za použití algoritmu prohledávání okolí Neighbor Discovery (lze použít i DHCPv6 nebo statické nastavení adresy)
 - Použití síťové bezpečnosti (šifrování a ověřování) je povinné v IPv6
 - Mobilita – zabraňuje různému směrování paketů při cestě ze zařízení a zpět (avoid triangular routing)
 - Jumbogram – datagram s velikostí až $2^{32} - 1$ (v IPv4 max 65535)

Adresy IPv6

Zápis 8 skupin s 16 bity – každá skupina má 4 hexadecimální čísla

- 2001:0db8:85a3:0000:0000:8a2e:0370:7334
- 0 nejsou důležité, pokud nemají žádný význam
- 2001:db8:85a3:::8a2e:370:7334

Formát adresy:

- 48 nebo více – routing prefix - směrování
- 16 nebo méně – subnet-id (subnet-id+routing prefix = 64 bitů)
- 64 – identifikace zařízení (může to být MAC-adresa síťové karty, identifikátor od DHCPv6, náhodně vygenerované číslo, ručně zadané číslo)

Multicast – datagram pro více počítačů:

- 8 bitů prefix – začínající FF
- 4 bity příznaků
- 4 bity rozsah – scope 16 předefinovaných rozsahů pro multicast
- 112 bitů číslo skupiny

IPv6 směrování

- Zjednodušené zpracování pro směrovače (routery)
- Hlavička IPv6 je jednodušší
- IPv6 směrovače neprovádějí fragmentaci
 - Minimální MTU (Maximal transmission unit) je 1280
 - Směrovače umožňují detekci MTU na specifikované cestě
- IPv6 nemá kontrolní součet (to zajišťuje transportní a linková vrstva)
- TTL – (Time To Live) je nahrazen Hop Limit – maximální počet směrovačů na cestě, není třeba měřit čas strávený v bufferech
- Směrovací prefix obsahuje veškeré informace potřebné k směrování
 - RFC 3177 (2001) navrhuje směrování všech počítačů na základě /48 lokace
 - RFC 6177 (2011) zredukovalo lokaci na /56

Od IPv4 k IPv6

- IPv4 s CIDR směrováním a použití NAT zpomalily nutnost přechodu na IPv6
- Jak přejít od IPv4 k IPv6
 - Dual-stack – asi nejčastější současné řešení, směrovač podporuje současně obě verze IP protokolu
 - Tunneling – zapouzdření telegramů IPv6 do IPv4
 - Použití proxy a překlad – pro počítače s IPv6, který chce využít služeb IPv4 serveru je nutné zajistit převedení a překlad IPv6 na IPv4
- Zkontrolujte si své připojení na:
 - www.test-ipv6.cz
 - www.test-ipv6.com
 - ipv6test.google.com

Transportní vrstva

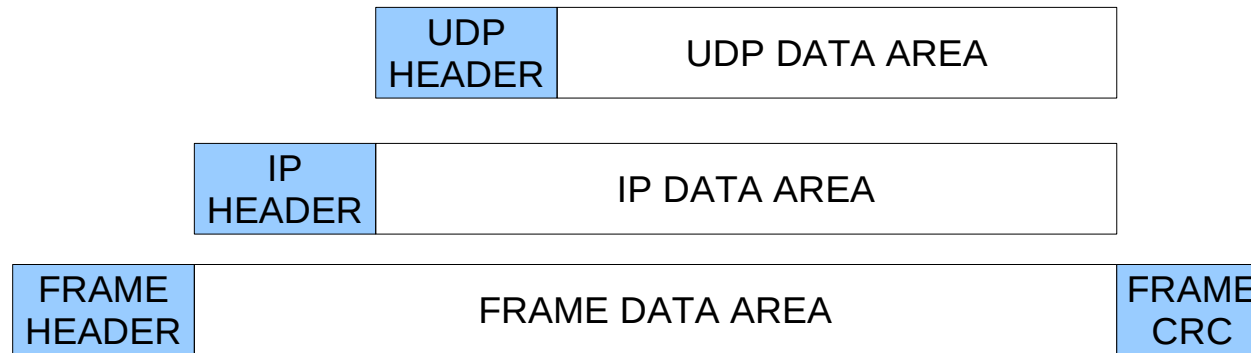
- Cíle transportní vrstvy
 - Zajistit komunikaci mezi procesy
 - Rozlišit různé adresáty na jednom počítači
 - Zajistit spojovaný přenos dat
 - Zvýšit spolehlivost
 - Zvýšit kvalitu služby (QoS Quality of Service)
 - Kontrolovat přenos dat
- Rozlišují se 3 typy sítí
 - Kategorie A – sítě bez ztrát paketů a bez chyb spojení – lokální sítě
 - Kategorie B – sítě bez ztrát paketů s možností chyb spojení – privátní sítě
 - Kategorie C – sítě s možností ztrát paketů i chyb spojení - internet

Transportní vrstva

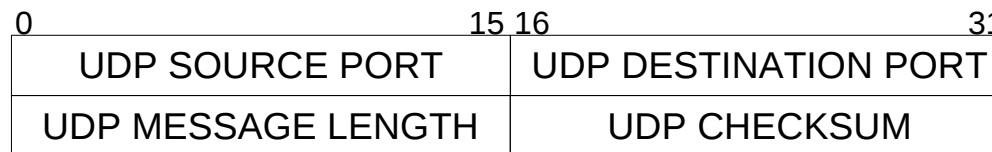
- 5 tříd transportní vrstvy
 - TP0 – jednoduchá vrstva pro sítě kategorie A
 - TP1 – vrstva řešící rozpojení pro sítě kategorie B
 - TP2 – vrstva pro sítě kategorie A s použitím portů
 - TP3 – vrstva řešící rozpojení pro sítě kategorie B s použitím portů
 - TP4 – transportní vrstva pro sítě kategorie C s použitím portů a spolehlivého doručení dat s potvrzováním
- Příklady
 - UDP – transportní vrstva třídy TP2
 - TCP – transportní vrstva třídy TP4

Protokol UDP

- UDP (= *User Datagram Protocol*)
 - jeden z nejjednodušších transportních protokolů.
 - Poskytuje tzv. **nespojovanou** a **nezabezpečenou** službu doručování uživatelských datagramů
 - Oproti ryzím IP datagramům má schopnost rozlišit mezi různými cílovými procesy na adresovaném počítači pomocí položky **port**



Zapouzdření dat v transportní, síťové a spojové vrstvě



Hlavička UDP telegramu

- Položky PORT se používají k rozlišení výpočetních procesů čekajících na cílovém stroji na UDP datagramy.
 - Položka SOURCE PORT je nepovinná; není-li použita, musí být 0.
 - Jinak označuje číslo portu, na něž má být zaslána případná odpověď.

Protokol UDP

- Aby se zajistilo, že různé stroje na Internetu si budou rozumět, IANA vydává závazný seznam tzv. **obecně známých čísel portů**
- Některá vybraná čísla UDP/TCP portů:
<http://www.iana.org/assignments/port-numbers>

Port	Keyword	Použití
0		Rezervován
7	echo	Vrátí zaslaný datagram
13	daytime	Vrátí datum a čas jako text
22	ssh	Bezpečný shell
53	dns	Domain Name Server
67	bootps	Bootstrap Server Protocol
69	tftp	Trivial FTP
123	ntp	Synchronizace hodin počítačů
137	netbios-ns	NetBIOS Name Service

Protokol UDP

- UDP protokol nezabezpečuje, že:
 - datagram se během přenosu neztratí
 - datagram nebude doručen vícekrát
- Potřebné zabezpečení musí řešit aplikace, které UDP používají
- Příklad použití UDP
 - DNS (překlad mezi symbolickými jmény strojů a jejich IP adresami)
 - realizuje komponenta lokálního OS zvaná *resolver*
 - Utilita **nslookup** slouží k explicitnímu použití (a testování) DNS

```
zubrina > nslookup proxy.felk.cvut.cz
```

```
IP(udp) 147.32.85.46:41245 > 147.32.80.9:53 11033- A? proxy.felk.cvut.cz. (36)
```

```
IP(udp) 147.32.80.9:53 > 147.32.85.46:41245 11033* 1/2/3 A 147.32.80.13 (146)
```

```
zubrina> nslookup 147.32.80.13
```

```
IP(udp) 147.32.85.46:38523 > 147.32.80.9:53 2- PTR? 13.80.32.147.in-addr.arpa. (43)
```

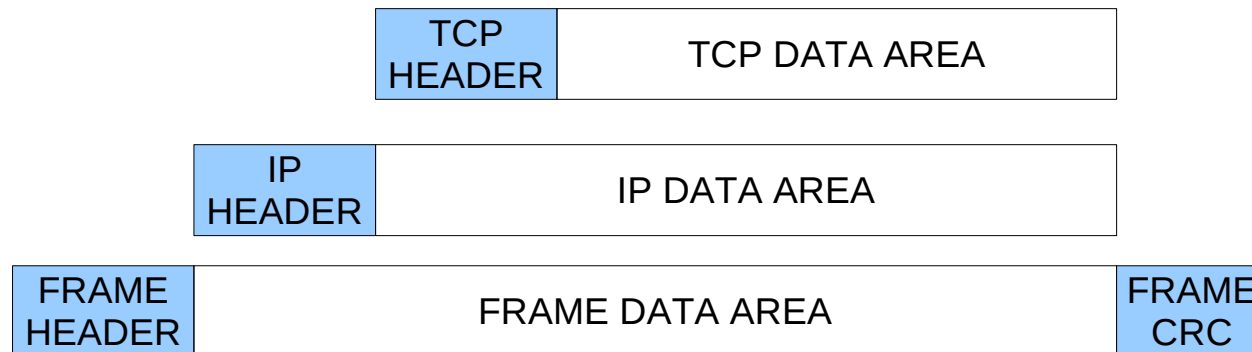
```
IP(udp) 147.32.80.9:53 > 147.32.85.46:38523 2* 1/2/4 PTR proxy.felk.cvut.cz. (197)
```

Protokol zabezpečeného datového toku TCP

- TCP je nejdůležitější obecná **zabezpečená** služba realizující přímé spojení mezi dvěma počítači
 - TCP/IP je Internetová implementace této služby
- **Vlastnosti TCP**
 - **Datový tok**
 - Aplikace komunikující po TCP/IP spoji považují komunikační kanál za tok bytů (oktetů) podobně jako soubor
 - **Virtuální spoj**
 - Před začátkem přenosu dat se komunikující aplikace musí dohodnout na spojení prostřednictvím síťových komponent svých operačních systémů
 - Protokolový software v operačních systémech obou počítačů se dohodne zasíláním zpráv po síti a ověří, že spojení lze spolehlivě navázat a že oba koncové systémy jsou připraveny ke komunikaci
 - Poté jsou aplikace informovány o ustaveném spojení a datová komunikace může být zahájena
 - Přerušil-li se spojení během komunikace, obě strany jsou o tom informovány
 - Termín **virtuální spoj** je používán k vyjádření iluze, že aplikace jsou propojeny vyhrazeným spojem. Spolehlivosti je dosaženo **plně vázanou komunikací** po spoji (úplný "handshake")

Protokol zabezpečeného datového toku TCP

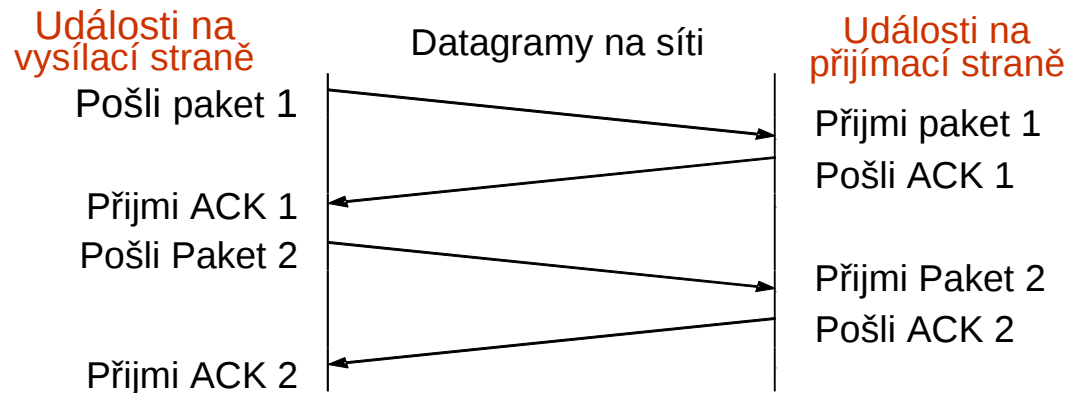
- Přenos s vyrovnávací pamětí
 - Pro zlepšení efektivity přenosu skládá protokolový modul v OS data tak, aby se po síti posílaly **pakety** rozumné velikosti. Pokud to není žádoucí (např. TELNET), je TCP/IP vybaveno mechanismem, který vynutí přednostní přenos i velmi krátkého datagramu „mimo pořadí“
- Plně duplexní spojení
 - Aplikační procesy vidí TCP/IP spojení jako **dva nezávislé datové toky** běžící v opačných směrech bez zjevné interakce. Protokolový software **potvrzuje** (ACK) data běžící v jednom směru v paketech posílaných spolu s daty ve směru opačném



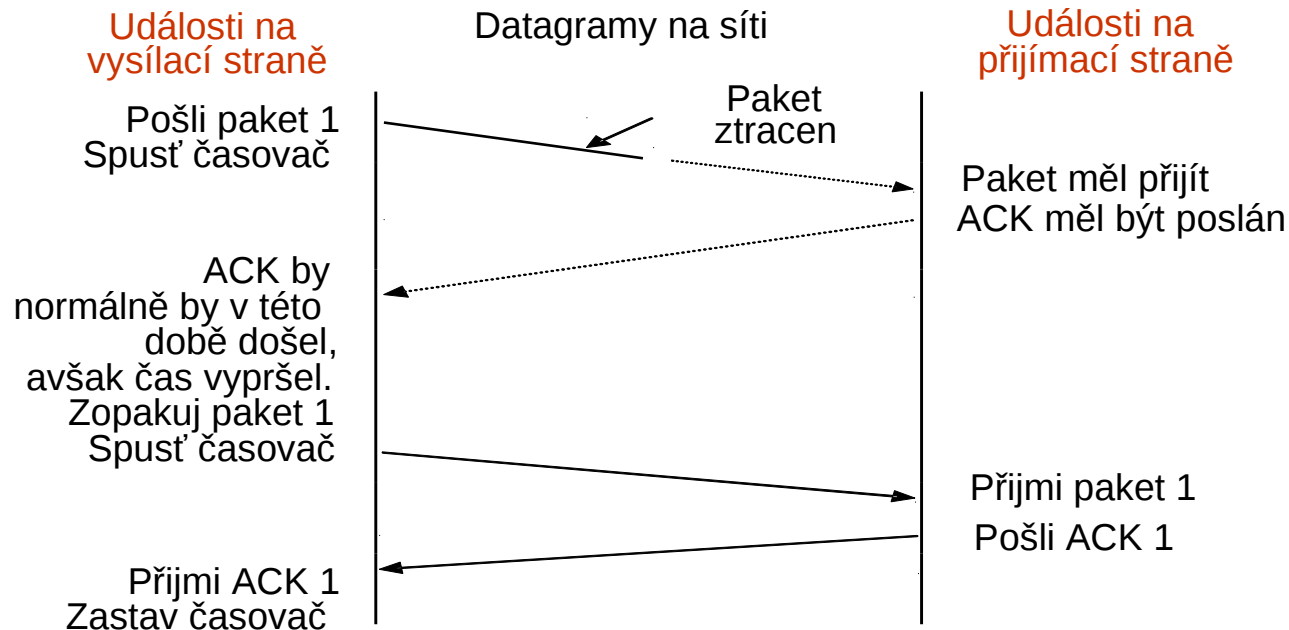
Zapouzdření dat v transportní, síťové a spojové vrstvě

Řešení spolehlivosti TCP

- Zajištění spolehlivého přenosu
 - pozitivní potvrzování došlých dat spolu s opakováním přenosu

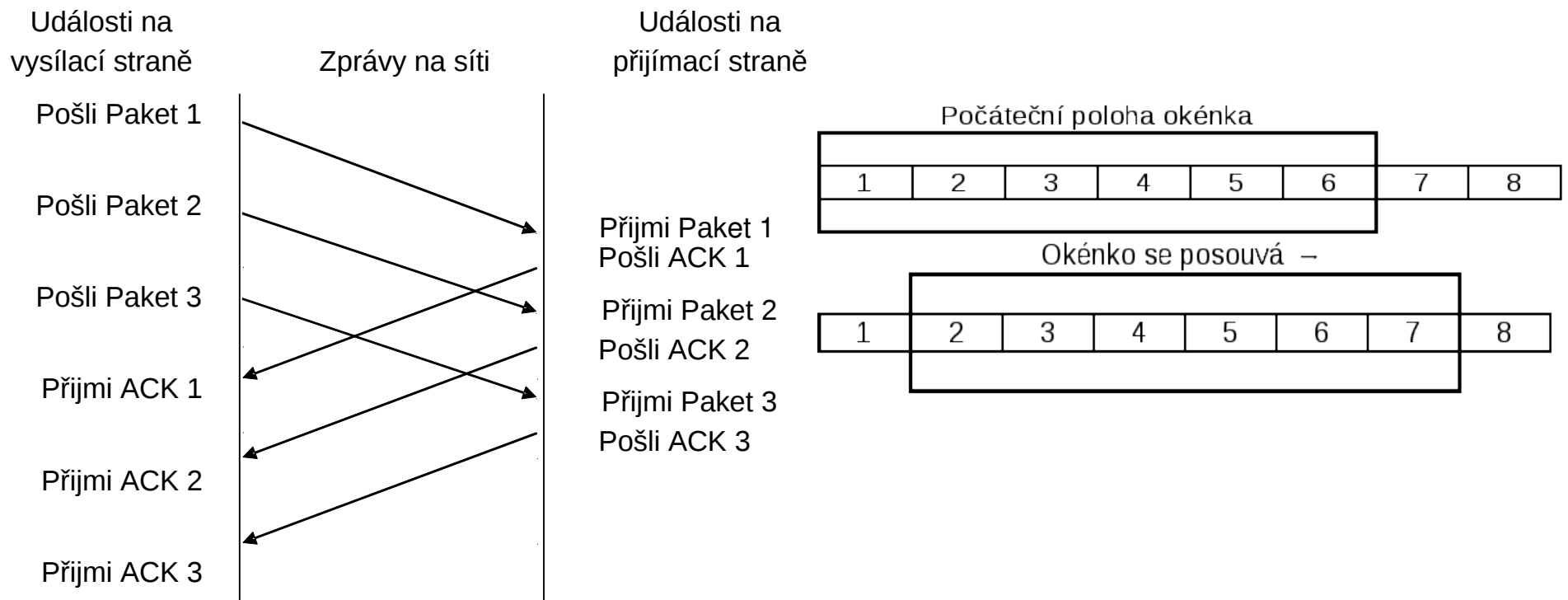


- Ztracené pakety budou zopakovány na základě vhodných časových prodlev



Řešení spolehlivosti TCP

- Datagramy se mohou po cestě i duplikovat (data i ACK).
 - Tento problém se řeší pomocí **sekvenčního číslování datagramů**
- Problém efektivity pozitivního potvrzování
 - Čekání na potvrzení každého paketu je časově nákladné
 - **Metoda** posouvajícího se **okna**

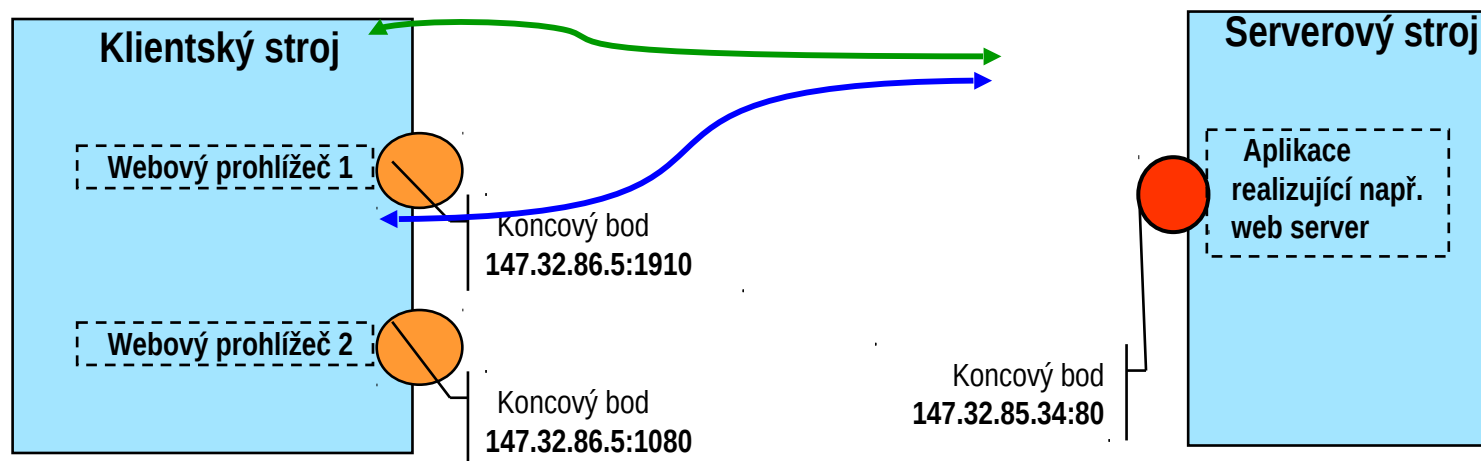


Porty, spojení a koncové body TCP

- TCP rovněž používá **porty** k rozlišení cílové aplikace na spojených počítačích
 - Čísla portů pro TCP mohou být stejná jako pro UDP, neboť protokoly jsou rozlišeny na obou koncích spoje automaticky
 - Stroj přijímající datagram se napřed "podívá" na pole `PROTOCOL` v hlavičce datagramu a podle něj předá zpracování buď UDP nebo TCP "větví" v síťové komponentě OS
 - Aby bylo možno využívat též služby počítače (serveru) větším počtem jiných počítačů (klientů), TCP/IP zavádí tzv. **virtuální spojení (virtuální kanály)**.
 - Tyto virtuální kanály jsou vlastně spojení mezi tzv. **koncovými body**, což jsou IP adresy s připojeným číslem portu, např. 147.32.85.34:80.
 - **TCP/IP virtuální spojení** je pak identifikováno **dvěma koncovými body** tohoto spojení
 - v IP v4 je to vlastně 12 bytů – tj. 2 x (4 byty adresy + 2 byty port)
 - Pár koncových bodů "celosvětově" odlišuje existující TCP/IP spoj
 - Různé spoje mohou mít na jednom konci též "koncový bod", avšak na druhém konci musí být různé koncové body.

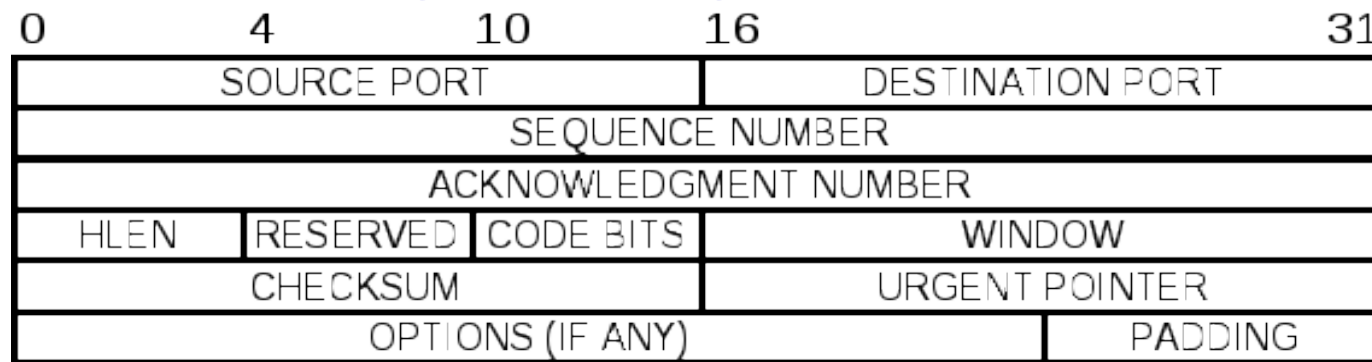
Tvorba TCP spojení

- Pasivní a aktivní otevření
 - TCP vyžaduje, aby se systémy, mezi nimiž se spojení navazuje, předem dohodly o vzniku spojení
 - Aplikace na jednom konci musí požádat svůj lokální OS a uskutečnit tzv. **pasivní otevření** na daném portu indikující ochotu aplikace přijímat příchozí žádosti o spojení. Tento konec kanálu je obvykle označován jako **server**.
 - Server "poslouchá" na daném portu
 - Chce-li klientská aplikace se serverem navázat spojení, **požádá svůj OS o aktivní otevření**, kdy zadá IP adresu serveru a příslušný port. Lokální klientský OS přiřadí navazovanému spojení vhodný **volný lokální port** (obvykle 1024 – 2047). Oba stroje pak naváží spojení (→) a mohou spolu komunikovat.



TCP segmenty a jejich formát

- Datový tok TCP se dělí na segmenty
 - Segmenty putují po síti jako IP datagramy
 - Každý byte v datovém toku má své 32-bitové sekvenční číslo v rámci spojení
- Hlavička TCP datagramu (segmentu)



- Význam položek

- SOURCE PORT, DESTINATION PORT: Identifikace aplikací na obou koncích spojení
- SEQUENCE NUMBER: Sekvenční číslo bytu v datovém toku
- ACKNOWLEDGMENT NUMBER: sekvenční číslo bytu v protisměrném toku, který odesílatel očekává v odpovědi od příjemce

Poznámka: SEQUENCE NUMBER se vztahuje ke směru přenosu, v němž se posílá segment, zatímco ACKNOWLEDGMENT NUMBER se vztahuje ke směru opačnému

TCP segmenty a jejich formát

- Význam položek TCP hlavičky (pokračování)
 - HLEN: Délka hlavičky ve 32-bitových slovech
 - CODE: Pole obsahující 1-bitové příznaky:
 - URG: Pole URGENT POINTER je platné →
 - ACK: Datagram nese potvrzení protisměrného datového segmentu
 - PSH: Tento segment požaduje "push", tj. okamžité doručení aplikaci bez použití vyrovnávací paměti na přijímající straně
 - RST: Reset spojení
 - SYN: Aktivní žádost o zřízení spojení (synchronizace sekvenčních čísel)
 - FIN: Ukončení spojení (odesílatel detekoval konec datového toku)
 - WINDOW: Určuje kolik dat je odesílatel ochoten přijmout od příjemce v rámci datového toku běžícího v opačném směru
 - URGENT POINTER: Toto pole je ukazatel na urgentní datový element uvnitř datového úseku segmentu (např. ^C v TELNET-ovém spojení) – platí jen ve spojení s příznakem URG
 - OPTIONS: Volitelné položky používané při vytváření spojení (např. max. velikost segmentu)

Časové prodlevy pro opakování přenosů

- Konstantní hodnota časového zpoždění pro opakované vyslání paketu je nevhodná
 - Internet je příliš různorodý a je složen z mnoha různých LAN a „point-to-point“ spojů založených na různých HW technologiích
- TCP/IP přizpůsobuje časové parametry virtuálního spoje
 - Používá adaptivní algoritmus pro zopakování posílaného paketu.
 - Algoritmus je založen na průběžném sledování tzv. „*round trip time*“ (RTT)
 - Doba mezi odesláním paketu a přijetím jeho potvrzení.
 - Skutečná prodleva pro opakování paketu je určována jako vážený průměr z RTT naměřených v nedávné historii.
 - Strategie se rychle přizpůsobuje okamžité zátěži mezilehlých sítí a směrovačů

Navázání TCP spojení

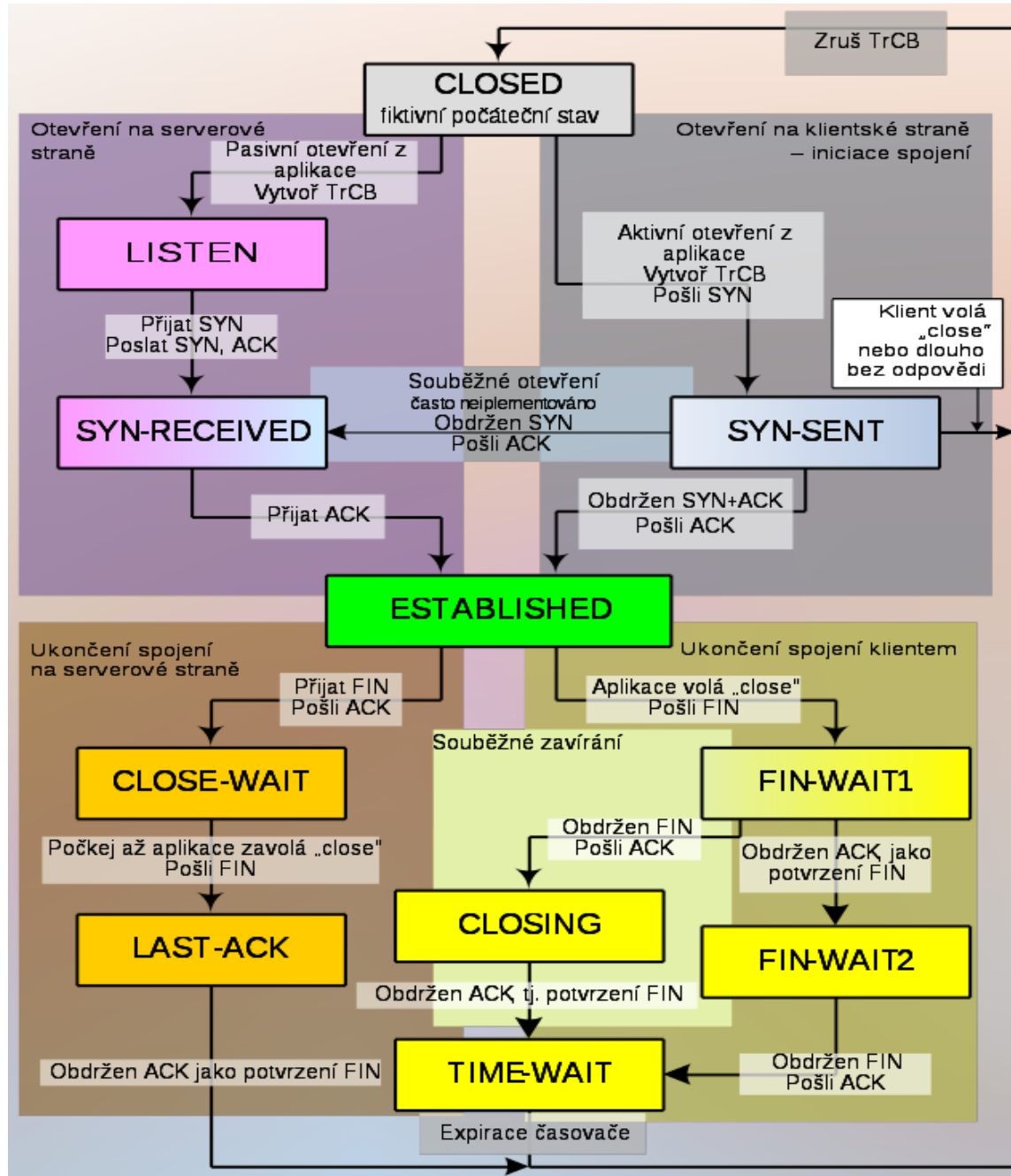
- TCP používá **třístupňový postup** navazování spojení (předpokládáme, že server „naslouchá“):
 1. V prvním kroku iniciátor spojení (**klient**) pošle adresátovi (**serveru**) segment s nastaveným **SYN** bitem, náhodně vygenerovaným SEQUENCE NUMBER = x a prázdnou datovou sekci
 2. Když **server** obdrží tento segment, odpoví na něj segmentem s nastaveným **SYN** a ACK, náhodným SEQUENCE NUMBER = y a ACKNOWLEDGMENT NUMBER = $x+1$.
 3. Když **klient** dostane tento segment, potvrdí jeho přijetí zasláním segmentu s nastaveným ACK, nulovým SYN bitem a ACKNOWLEDGMENT NUMBER = $y+1$.
 - Tak jsou ustaveny počáteční hodnoty SEQUENCE NUMBER a ACKNOWLEDGMENT NUMBER pro dané spojení
 - Sekvenční čísla jsou **náhodná**
 - Možnost detekovat havárii či restartování strojů na koncích spoje v rámci časové prodlevy

Ukončení TCP spojení, TCP porty

- Ukončení spojení nastává obvykle na žádost aplikace, která spojení ustavila
 - Aplikace sdělí TCP, že už nemá další data
 - TCP software uzavře spojení v jednom směru, což se děje zasláním segmentu s nastaveným FIN bitem
 - K úplnému ukončení je třeba spojení zavřít i v opačném směru podobným způsobem (tj. FIN bitem)
 - TCP spojení lze i okamžitě násilně přerušit užitím bitu RST
- Příklady obecně známých čísel TCP portů

Port	Keyword	Použití
20	ftp-data	File Transfer Protocol - data
21	ftp	File Transfer Protocol - ovládání
22	ssh	Bezpečný shell
23	telnet	Terminal console
25	smtp	Simple Mail Transport Protocol
80	http	World-wide web
110	pop3	Post Office Protocol v.3
143	imap	Internet Message Access Protocol
443	https	Bezpečné www

Konečný automat TCP



Uveden je zjednodušený konečný automat

- předepíše chování síťové vrstvy OS pro TCP spojení
- jde o „popis implementace“
- každé samostatné spojení může být v daném okamžiku v jiném vývojovém stavu

Každé TCP spojení má svůj řídicí blok TrCB

(Transmission Control Block)

- je propojen s příslušným socketem
- registruje průběh navazování spojení a jeho "stav"
- odkazuje na vyrovnávací paměti pro přenos dat

API pro síťové služby

- Základním prostředkem pro síťové komunikace je tzv. **socket**
 - obecný objekt pro meziprocesní komunikaci (IPC)
 - nejčastěji však pro IPC prostřednictvím počítačových sítí
 - tzv. rodina **POSIX socketů** – zprostředkují IP komunikaci
 - POSIX sockety se vyvinuly z původních BSD socketů
 - z pohledu API se socket jeví jako POSIX „soubor“
- Vytvoření socketu (získání manipulačního čísla „souboru“)
`int sock_fd = socket(int domain, int type, int protocol)`
 - `domain` – specifikuje rodinu socketu (pro IP domain = `AF_INET`)
 - `type` – určuje způsob komunikace zprostředkované socketem
 - `SOCK_STREAM` = socket zprostředkuje datový tok (nejčastěji TCP)
 - `SOCK_DGRAM` = socket zprostředkuje předávání datagramů (např. UDP)
 - `SOCK_RAW` = socket umožňuje přímý přístup k síťovým službám (užívá se např. pro přístup aplikací k ICMP)
 - `protocol` – konkrétní protokol (TCP, UDP, ...)
 - `IPPROTO_IP` = protokol je automaticky zvolen podle parametru `type`
 - `IPPROTO_ICMP` = socket zprostředkuje ICMP protokol
 - `IPPROTO_UDP` = UDP přenos datagramů
 - `IPPROTO_TCP` = TCP datový tok
 - ... další protokoly viz RFC 1700

Operace se sockety

- Navázání socketu na lokální adresu (pasivní otevření na serverové straně)
`bind(int sock_fd, const struct sockaddr *my_addr, socklen_t addr_len)`
 - `sock_fd` – socket
 - `my_addr` – lokální adresa, pro `AF_INET` struktura včetně portu
 - `addr_len` – délka adresy v bytech
 - Čekání socketu na žádost o příchozí spojení (na serverové straně)
`listen(int sock_fd, int backlog)`
 - `sock_fd` – socket
 - `backlog` – maximální počet čekajících spojení
 - Přijetí žádosti klienta o spojení se serverem (včetně identifikace klienta)
`new_fd = accept(int sock_fd, struct sockaddr *client_addr, socklen_t *client_addr_len)`
 - `sock_fd` – socket
 - `client_addr` – adresa klienta, pro `AF_INET` struktura včetně portu
 - `addr_len` – délka adresy v bytech
 - `new_fd` je „souborový deskriptor“, jehož prostřednictvím bude probíhat obousměrná komunikace mezi klientem a serverem
- Původní socket zůstává ve stavu „listen“ a je chopen přijímat další příchozí spojení a řadit je do fronty. Existují-li takové žádosti o spojení, další volání `accept` vrátí ihned další klientské spojení; v opačném případě `accept` způsobí zablokování volajícího procesu

Operace se sockety

– Připojení na vzdálenou adresu (aktivní otevření klientem)

`connect(int sock_fd, const struct sockaddr *serv_addr, socklen_t addr_len)`

- `sock_fd` – socket
- `serv_addr` – adresa serveru, k němuž se klient připojuje, pro `AF_INET` struktura včetně portu
- `addr_len` – délka adresy v bytech

– K přenosům dat mezi klientem a serverem poté, kdy příchozí žádost byla akceptována (spojení bylo úspěšně navázáno)

`send()`, `recv()`

`sendto()`, `recvfrom()`

`write()`, `read()`

`sendmsg()`, `recvmsg()`

- Prvním parametrem všech těchto funkcí je `sock_fd`
- Detaily viz specifikace POSIX

– Ukončení spojení

`close(int sock_fd)`

Použití API pro síťové služby

• Server

1. Vytvoř socket voláním služby `socket()`
2. Navaž socket na lokální adresu a port voláním `bind()`
3. Připrav socket na příchod žádostí o spojení voláním `listen()`, vznikne „naslouchající socket“
4. Voláním služby `accept()` se server zablokuje, dokud nepřijde žádost o spojení. Návratovou hodnotou `accept()` je nový souborový deskriptor (`fd`), otevřený pro komunikaci. Původní socket stále naslouchá a lze znovu volat `accept()`.
5. Komunikace pomocí `send()` a `recv()` nebo `write()` a `read()`
6. Případné volání `close()` končícím serverem (passive close)

• Klient

1. Vytvoř socket voláním služby `socket()`
2. Volání služby `connect()` naváže spojení se serverem a vrátí souborový deskriptor pro další komunikaci
3. Komunikace se serverem pomocí `send()` a `recv()` nebo `write()` a `read()`
4. Volání `close()` k ukončení spojení se serverem (active close)

Základní aplikační IP protokoly

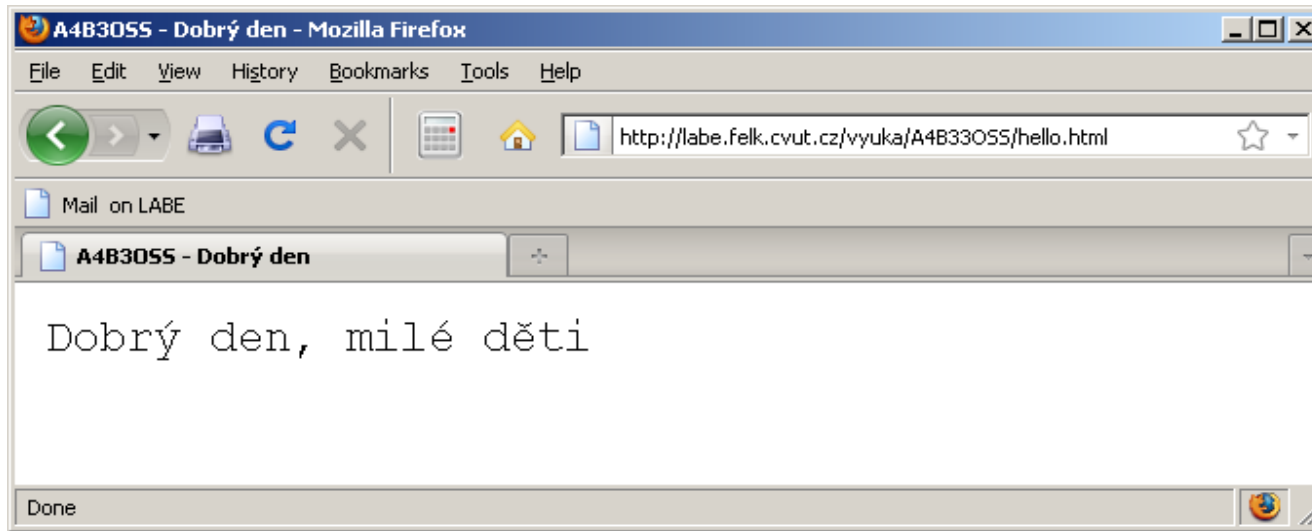
- IP aplikačních protokolů jsou stovky
 - Obvykle se pro uživatelsky orientované protokoly používá zabezpečený transportní protokol TCP/IP
- Většina aplikačních protokolů využívá komunikace „v otevřené řeči“
 - příkazy a reakce na ně jsou „v primitivní angličtině“
 - nebezpečné, proto často existují „zabezpečené“ (zakódované) varianty
- Vyjmenujeme jen pro ukázkou některé základní aplikační protokoly
 - SMTP (*Simple Mail Transfer Protocol*), TCP port 25
 - Protokol je určen pro zasílání e-mailů klientem nebo „serverem“ (který se při tomto přenosu chová jako klient) na cílový server.
 - Základní příkazy zadávané klientem jsou MAIL, RCPT, DATA
 - POP3 (*Post Office Protocol*), TCP port 110
 - Protokol pro stahování e-mailů ze serveru, kam byl e-mail doručen pomocí SMTP, do pracovní stanice (např. do aplikace MS-Outlook)
 - Základní příkazy klienta: USER, PASS, LIST, RETR, DELE
 - FTP (*File Transfer Protocol*), TCP porty 20 a 21
 - Velmi komplexní protokol založený na dvou TCP spojích (řídící a datový); existuje mnoho „klonů“ (např. pasivní FTP) a zabezpečených variant (např. SFTP)
 - Příkazů je asi 50

Protokol HTTP – reálná ukázka

- Web server pro protokol HTTP (*HyperText Transfer Protocol*) poslouchá na TCP portu 80
- Příklad komunikace
 - Na serveru *labe.felk.cvut.cz* je soubor `hello.html` v adresáři `/vyuka/A4B33OSS`
 - serverový proces `httpd` obsluhující TCP port 80 považuje z bezpečnostních důvodů jistý konkrétní adresář na serveru jako kořen adresářového stromu pro „webové soubory“ (dáno konfigurací `httpd`)
 - Obsah souboru `/vyuka/A4B33OSS/hello.html` je např.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1250">
<meta http-equiv="Pragma" content="no-cache">
<title>A4B33OSS - Dobrý den</title>
</head>
<body>
<pre>
Dobrý den, milé děti
</pre>
</body>
</html>
```

Protokol HTTP – reálná ukázka



- Klient Firefox se připojí na TCP port 80 a pošle

GET /vyuka/A4B33055/hello.html HTTP/1.1

Host: labe.felk.cvut.cz

User-Agent: Mozilla/5.0 (Windows NT 5.1; en-US;) Firefox/3.25

Accept: text/html, Accept-Language: cs,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7

Keep-Alive: 15

Connection: keep-alive

Cache-Control: max-age=0

<prázdný řádek>

Povinné komponenty zasláního příkazu GET jsou **tučně**.
Další jsou doplňkové informace pro server

Protokol HTTP – reálná ukázka

- Web server odpoví

```
HTTP/1.1 200 OK
Date: Sat, 18 Dec 2012 19:47:10 GMT
Server: Apache/1.3.37 (Unix)
Last-Modified: Sat, 18 Dec 2010 19:40:14 GMT
ETag: "2da442-ea-4d0d0e1e"
Accept-Ranges: bytes
Content-Length: 234
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

```
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=windows-1250">
<title>A4B3OSS - Dobrý den</title>
</head>
<body>
<pre>Dobrý den, milé děti
</pre>
</body>
</html>
```

A po 15 sekundách server ukončí spojení

Obsah hello.html

To je dnes vše.

Otázky?