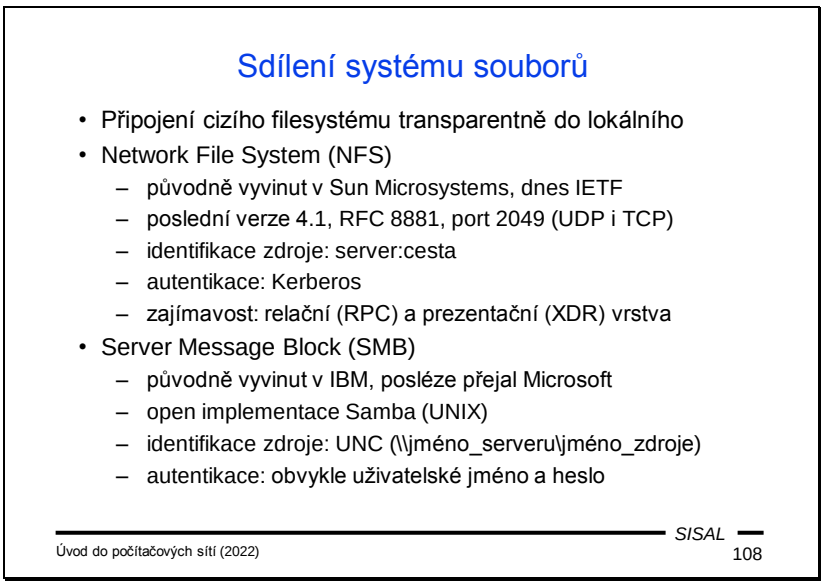


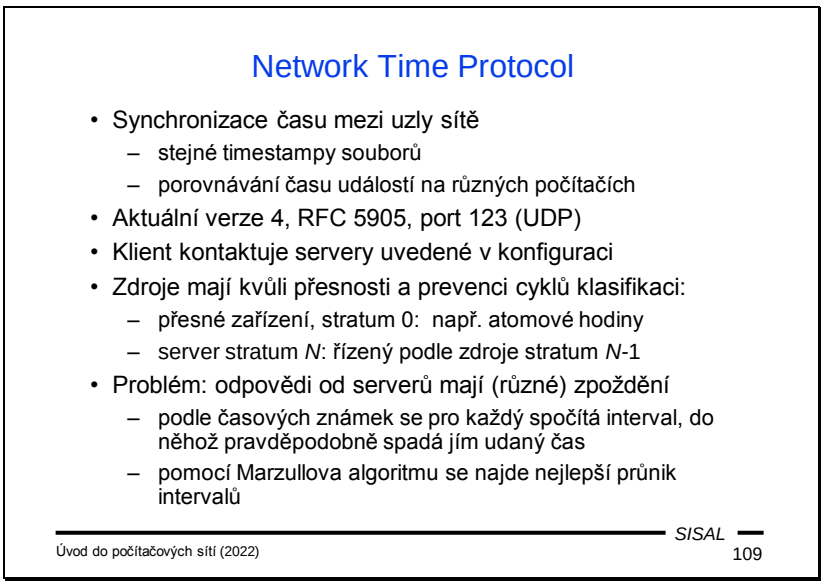
Poslední skupinou aplikačních protokolů, o nichž si budeme povídat, jsou protokoly, které většinou zůstávají pro uživatele poněkud skryté.

Velmi důležitým komunikačním nástrojem je možnost připojit ke svému počítači disk z jiného počítače a pracovat s ním, jako by to byl lokální disk. Mezi nejpoužívanější technologie patří NFS a SMB.

Network Fíle System pochází od firmy Sun Microsystems, nicméně v průběhu času se stal otevřeným protokolem popsaným jako RFC. Patří dodnes mezi nejpoužívanější systémy svého druhu v UNIXovém světě, pokud místní prostředí nevyžaduje propracovanější systém přístupových práv, než má klasický UNIX (v tom případě se nahrazuje např. systémem AFS, Andrew File System). Na připojený disk se systém odkazuje jako na dvojici *server*:*cesta*, pro uživatele je toto propojení transparentní a oni připojený disk vidí jako součást místního stromu adresářů. Jako autentikační mechanizmus se obvykle používá protokol Kerberos. Protokol NFS pracuje standardně nad UDP, i když je možné ho provozovat i nad TCP. Zajímavostí je to, že vnitřní struktura protokolu kopíruje detailněji OSI model a můžeme u něj rozlišit vrstvu relační (Remote Procedure Call, RPC), prezentační (Exchange Data Representation, XDR). RPC je přitom obecný mechanizmus, který využívají i jiné služby – klient pošle na server požadavek na zavolání funkce včetně seznamu argumentů a server operaci provede a pošle klientovi odpověď.

Server Message Block rovněž pochází z komerční sféry, konkrétně od IBM. Jeho vývoj potom převzal Microsoft, takže nedošlo k jeho zveřejnění formou RFC. Nicméně snaha propojit svět UNIXu a MS Windows vedla k tomu, že protokol byl podroben reverznímu engeneeringu v rámci projektu Samba a tato volná implementace skutečně ono propojení umožnila (klienti z obou rodin systémů si

mohou připojit disk ze serverů běžících na kterémkoliv systému). Připojený disk se zde označuje jako *server**cesta*, autentikaci si provádí systém sám pomocí uživatelského jména a hesla.



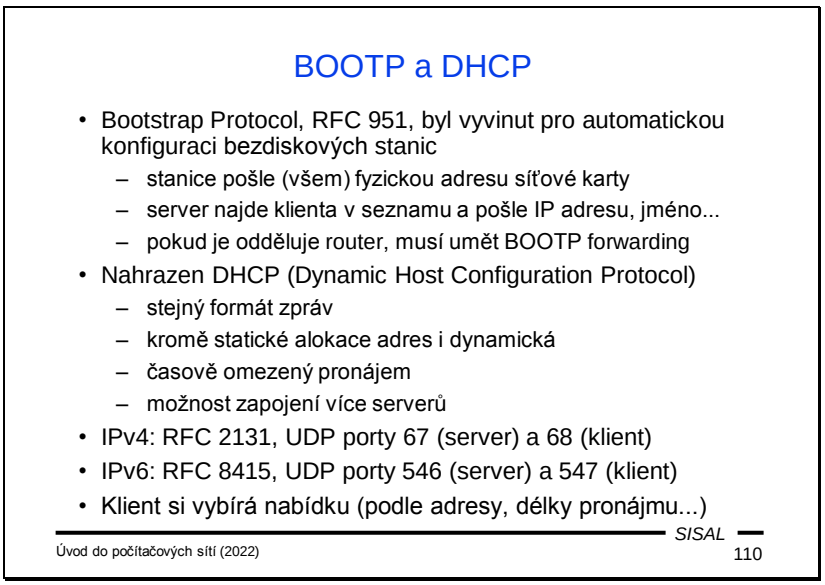
Velmi důležitou vlastností lokální sítě je synchronizace času na jednotlivých uzlech. Není nutná pro vlastní fungování sítě – v opačném případě by desynchronizace, k níž velmi snadno dojde, vedla k rozpadu sítě. Zato je podstatná z uživatelského hlediska, zejména ve dvou situacích:

- Pokud uživatelé přenášejí soubory mezi uzly sítě (a zvláště při sdílení disků) je podstatné, aby oba počítače měly shodný čas. Jinak dojde k tomu, že soubor uložený na jednom počítači bude na druhém považovaný za neaktuální. Takový rozpor by mohl způsobit např. opakované pokusy o aktualizaci nebo opakované zcela zbytečné překlady již přeložených modulů apod.
- Síťoví administrátoři velmi často řeší nějaký problém tak, že porovnávají záznamy z různých počítačů. Pokud jsou tyto záznamy pořízeny na strojích s nesynchronními hodinami, je noční můrou, když u každé řádky musí administrátor přemýšlet, o kolik minut musí který časový údaj upravit, aby tvořily správnou posloupnost.

Jedním z protokolů, které synchronizaci realizují, je Network Time Protocol. Základem protokolu je skutečnost, že někde existují zdroje s absolutně přesným časem (např. atomové hodiny). Ty se nazývají zdrojem *stratum 0*. Od nich čas postupně přebírají další servery (zdroj *stratum 1*), které opět slouží jako zdroj pro další úroveň. Idea označování zdroje podle „logické vzdálenosti“ od absolutně přesného zdroje slouží jednak pro odhad míry přesnosti konkrétního zdroje a jednak jako obrana před zacyklením (zdroj stratum N bude ignorovat časový údaj od zdroje stratum $N + 1$). Obvyklá konfigurace pro LAN pak vypadá tak, že v ní běží jeden nebo více NTP serverů, které se synchronizují proti NTP serveru, který poskytuje ISP, a proti nimž se synchronizují všichni klienti v lokální síti.

Při realizaci vlastního algoritmu se musí počítat s latencí sítě. Klient nemůže jednoduše vzít čas, který dostane v odpovědi, protože mezi odesláním a přijetím uběhla určitá doba. Výpočet proto používá časové známky v odpovědi, které určují interval, do něhož s jistou pravděpodobností spadá skutečný čas, a pomocí speciálního Marzullova algoritmu se z těchto intervalů určuje co možná nejpřesnější čas s co nejvyšší pravděpodobností. Díky tomu, že klient vlastně nedokáže stanovit zcela přesný čas, dochází mezi jednotlivými úrovněmi NTP serverů ke zvyšování nepřesnosti. Z praktického hlediska jsou ale tyto nepřesnosti hluboko pod rozlišovací schopností uživatelů.

Snímek 110



Posledními technickými aplikačními protokoly, kterými se budeme zabývat, jsou BOOTP a DHCP. Používají se k tomu, aby klient, který se připojuje do sítě, mohl získat IP adresu, kterou smí používat, a další informace o lokální síti.

BOOTP, Bootstrap Protocol byl vyvinut velmi dávno, v době, kdy bylo ekonomicky výhodné pořizovat *bezdiskové stanice*, tj. počítače, které neměly žádné zařízení pro trvalé uchování dat (disky byly tehdy relativně drahé). Na takové stanici nebylo možné nikam uložit konfiguraci, včetně IP adresy. BOOTP protokol umožňoval stanici vyslat žádost o přidělení adresy, kterou vyhrazený BOOTP server posoudil a případně poslal odpověď. Server posuzoval žádost ověřením MAC adresy proti seznamu povolených adres – MAC adresa bylo to jediné, co klient znal a čím se mohl identifikovat, protože ta byla vypálená v síťové kartě. Pokud byl klient na seznamu, server zjistil IP adresu, kterou mu má poslat (přiřazení tehdy bylo fixní) a poslal odpověď. Jakmile klient odpověď obdržel, mohl adresu začít používat.

Z technického hlediska je důležité zmínit to, že klient musí svou žádost poslat neadresně, protože nemá žádnou informaci o adresách v síti, kde se nachází. Použije tedy jako cílovou IP adresu tzv. *limited broadcast*, což znamená, že žádost dorazí všem uzlům v síti (kteří ji ale ignorují). Aby se takové žádosti nešířily nesmyslně po celém internetu, směrovače je nepropouštějí mimo síť, kde vznikly. To představuje mírnou komplikaci, pokud máme v LAN složitější strukturu podsítí oddělených routery. Pak buďto musíme mít BOOTP server v každé podsíti, anebo na směrovačích spustit tzv. *BOOTP forwarding*, kdy směrovač BOOTP dotazy přeposílá ze sítě, kde vznikly, určitému BOOTP serveru a jeho odpověď zase zpátky klientovi.

Postupem času se ukázalo, že IP adresa nestačí. Sami jsme už viděli několik příkladů dodatečných informací užitečných pro klienty: adresy směrovačů,

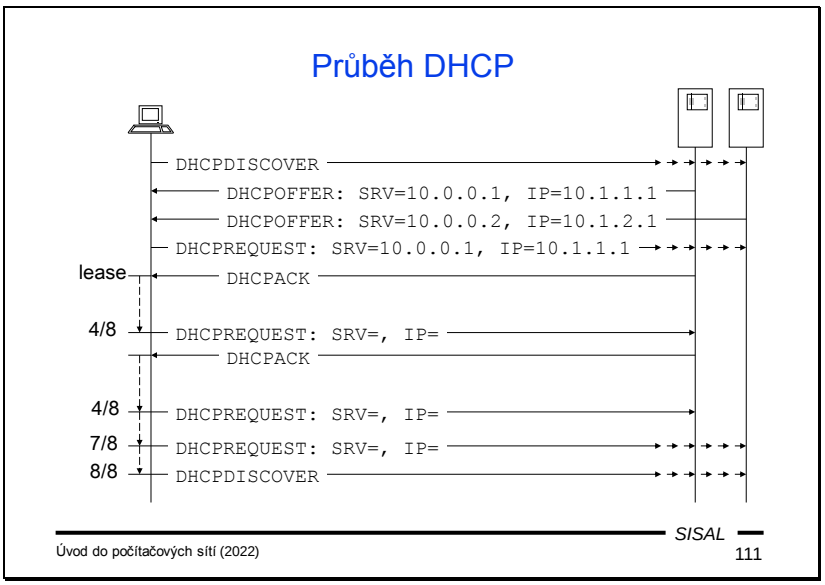
nameserverů, NTP serverů, mail forwarderů atd. Protokol se tedy postupně několikrát rozšiřoval, až došlo k zásadní změně na DHCP, Dynamic Host Configuration Protocol. Mezi hlavní rozšíření DHCP patří:

- Dynamická alokace adres. Pevné přidělování adres ztratilo dnes díky přepisovatelným MAC adresám bezpečnostní význam. Navíc často správa sítě ani nezná všechny klienty. A v neposlední řadě síť může nabízet výrazně méně adres než je počet potenciálních klientů.
- Časově omezený pronájem. Díky dynamické alokaci je nutné klientům přístup k adresám nějak časově omezit. Klient tedy dostává adresu jen na určitou dobu, tzv. *dobu pronájmu* neboli *lease-time*, a po jejím uplynutí musí adresu přestat používat.
- Kooperace více serverů. V síti může pracovat více serverů, mohou se lišit svým účelem i rozsahem poskytovaných adres, a klient si z jejich nabídek vybírá.

DHCP je zpětně kompatibilní s BOOTP, což umožňuje BOOTP klientům komunikovat s DHCP servery a dostávat od nich potřebné informace.

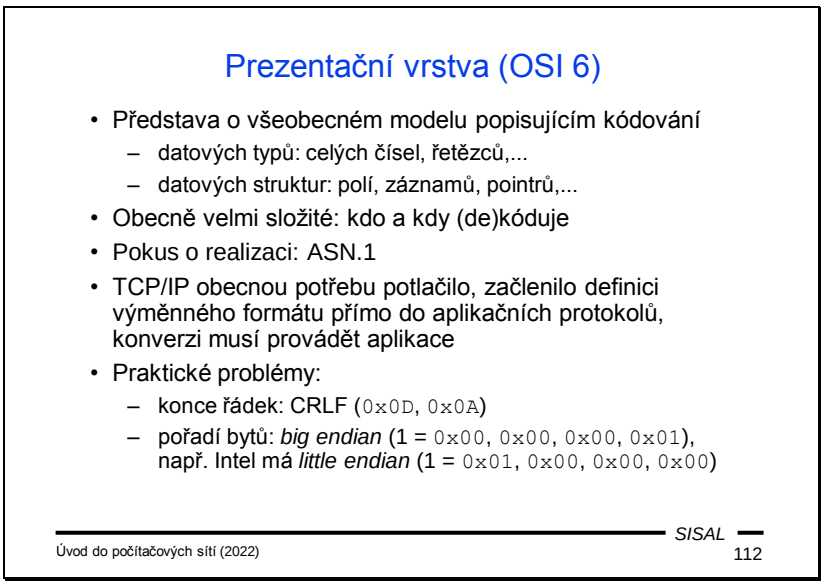
Dnes je DHCP nejběžnější způsob, jakým se klienti zapojují do sítě. Např. na systémech MS Windows tuto volbu najdeme pod poněkud zatemňujícím názvem „získat IP adresu automaticky“.

Snímek 111



Ukázka průběhu komunikace v DHCP:

- Klient posílá broadcastový požadavek DHCPDISCOVER.
- Jednotlivé DHCP servery v síti posílají své nabídky (DHCPOFFER).
- Klient má nakonfigurovaný určitý timeout, po který čeká, sbírá a posuzuje odpovědi.
- Z odpovědí si vybere tu nejlepší. Konkrétní algoritmus se může na různých klientech lišit, ale obvykle klient primárně preferuje, pokud mu některý server nabídne adresu, kterou už používá (nebo naposledy používal). Pokud takovou nabídku nedostane, vybírá obvykle podle délky nabízené doby pronájmu.
- Klient pošle zprávu DHCPREQUEST, která obsahuje adresu, kterou si zvolil. Tato zpráva jde ještě broadcastem, protože ji musí dostat všechny servery. Ty totiž po odeslání své nabídky nabízenou adresu na nějakou dobu zablokují, aby ji nenabídli dvěma klientům. Pokud si jejich nabídku klient nezvolil, musí adresu opět odblokovat.
- Server, jehož nabídku si klient vybral, následně potvrdí zprávou DHCPACK, že adresa je opravdu stále volná.
- Od tohoto okamžiku začíná běžet doba pronájmu.
- Nicméně v polovině této doby by klient měl poslat zprávu DHCPREQUEST, už pouze svému zvolenému serveru, aby se ujistil, že adresu má stále k dispozici.
- Pokud odpověď dostane, startuje se mu nový interval doby pronájmu.
- Pokud odpověď nedostane, v sedmi osminách doby pronájmu posílá nový DHCPREQUEST, tentokrát ale už broadcastem.
- Pokud ani tentokrát adresu nedostane, musí po uplynutí doby pronájmu zahájit proceduru zase od začátku.



Od aplikační vrstvy se přesuneme k vrstvě OSI modelu číslo 6, vrstvě **prezentační**.

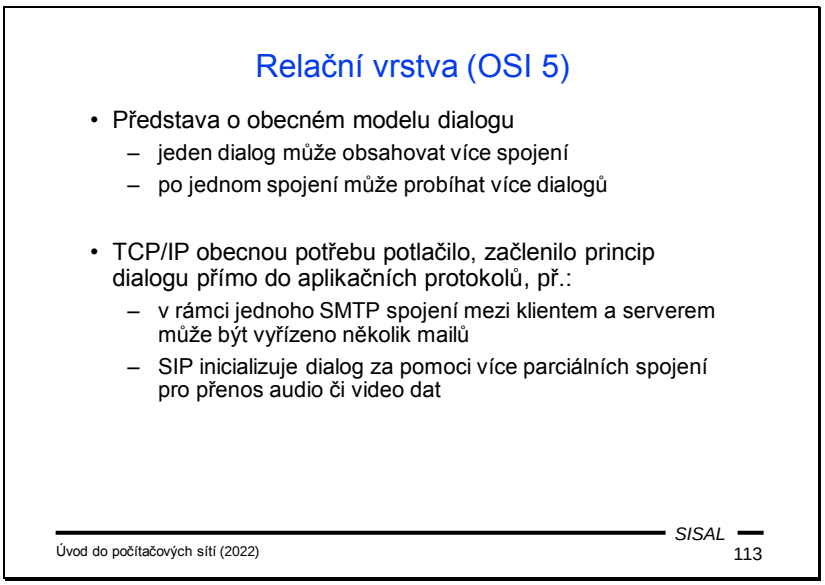
Záměrem návrhu bylo vytvořit obecný mechanizmus, jak odstínit konkrétní architekturu uzlu sítě od formátu používaného nižšími vrstvami. Vytvoření takového mechanizmu pro zcela obecné datové komunikace, tj. kódování různých datových typů nebo struktur je ale poměrně složité. Jedním z pokusů o řešení byla ASN.1, o které jsme si povídali v souvislosti s H.323. Jak jsme tehdy zmínili, z hlediska popisu to byl pokus dobrý, ale realizace jako univerzální nástroj byla díky obrovské složitosti daleko méně úspěšná.

Proto se návrh TCP/IP touto cestou nevydal a potřebné kódování zakotvil přímo do aplikačních protokolů, takže přenesl starost o případné kódování a dekódování na implementaci aplikace. Naštěstí pro programátora existují jednoduché nástroje, jak to udělat. Nejmarkantnějšími rozdíly mezi platformami spočívají ve dvou aspektech:

- Různé operační systémy používají různé znaky nebo kombinace znaků pro označení **konce řádek**. Microsoft DOS vyšel z principu, který používaly dálnopisy, mechanické psací stroje a staré tiskárny: pro založení nové řádky bylo nutné posunout tzv. *vozík* (carriage) na začátek řádky a poté otočit válcem o jednu řádku. Důsledkem bylo používání dvojice znaků CR (carriage return, hexa kód 0x0D) a LF (line feed, hexa kód 0x0A) na konci řádek. MS Windows v této tradici pokračují. Autoři UNIXu vyhodnotili nepraktičnost této kombinace a zavedli používání jediného znaku LF. Pro zajímavost můžeme zmínit starý systém Apple, který naopak používal pouze znak CR. Protokoly TCP/IP bohužel začaly používat model CR+LF. To má některé nepříjemné důsledky – např. jak se má uzel zachovat, když protistrana pošle znak CR? Má čekat, zda přijde ještě LF? Jak dlouho?
- Různé hardwarové architektury používají odlišné pořadí zápisu bajtů u vícebajtových hodnot do paměti. Kromě okrajových zvláštností existují dva

základní principy: buďto se bajty zapisuji tak, že nejprve se zapíše nejméně významný bajt („little end“ jako první) a po něm postupně další, anebo naopak („big end“ jako první). V TCP/IP protokolech se používá **big-endian** systém, neboli např. první bajt IP adresy jde po síti jako první. Pro programátora jsou připravené knihovny, které provádějí patřičnou konverzi, je-li překlad proveden na little-endian systému.

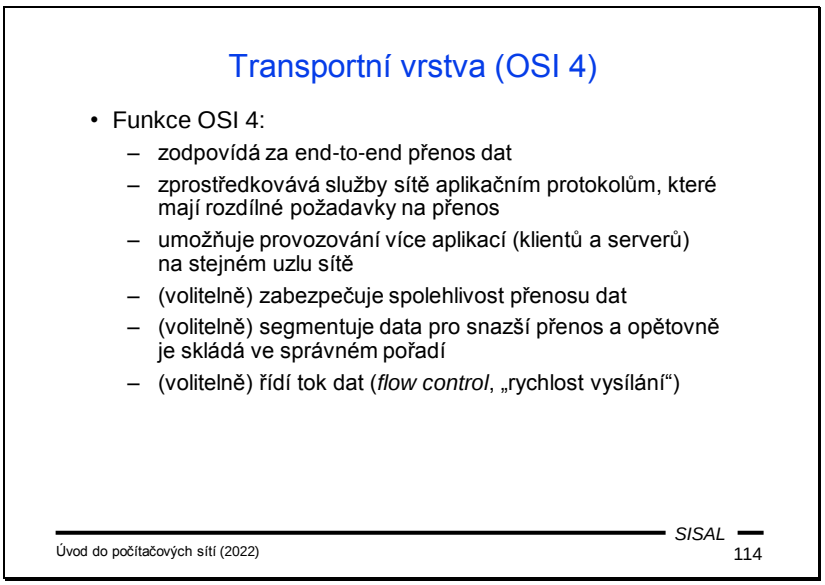
Snímek 113



Podobný osud jako prezentační vrstvu potkal i **vrstvu relační**. V některých protokolech můžeme tuto funkčnost najít (jako RPC v NFS), ale představa obecného nástroje, který pro všechny typy aplikačních protokolů dokáže řídit dialog komunikujících stran, se neukázala jako prakticky proveditelná.

V TCP/IP modelu se proto funkce OSI 5, stejně jako OSI 6 včleňují přímo do aplikačního protokolu. Příklady situací, kdy logický dialog komunikujících stran neodpovídá jednomu spojení, můžeme najít třeba v SMTP (klient předává postupně serveru několik samostatných zpráv v rámci stejného TCP spojení) nebo v SIP (jeden videohovor může být realizován jedním řídícím spojením a dvěma datovými kanály pro audio a video). Vlastní řízení průběhu dialogů pak popisují všechny aplikační protokoly.

Snímek 114



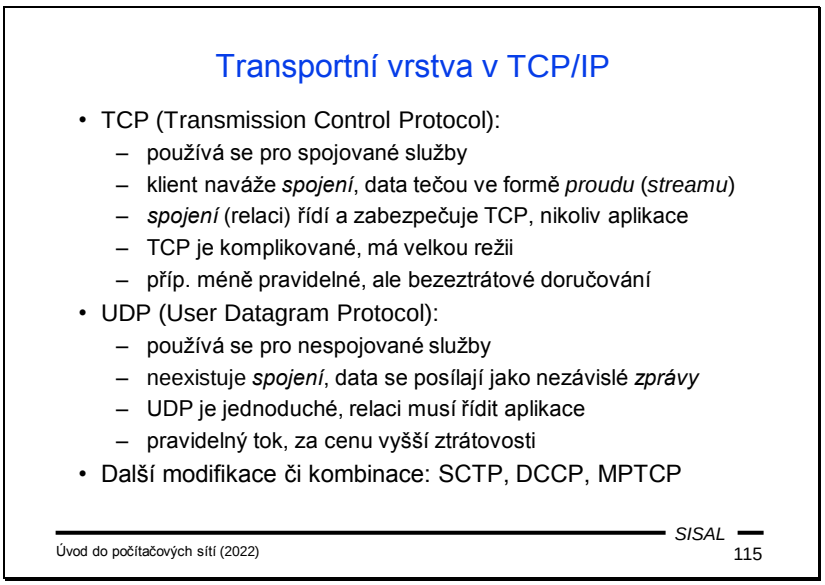
Úkolem **transportní vrstvy** je poskytnout aplikacím komunikační kanál pro přenos datových celků mezi aplikacemi běžícími na koncových zařízeních. Může být realizována pomocí různých protokolů, které se liší rozsahem poskytovaných služeb.

Rozhraní mezi transportní a aplikační vrstvou obecně poskytuje aplikaci možnost odeslat blok dat protistraně, konkrétní parametry a podmínky jsou závislé na potřebách aplikace. Zásadní rozdíl mezi různými protokoly transportní vrstvy spočívá v garanci doručení. Některé protokoly (jako TCP) poskytují tzv. *spolehlivou* (*reliable*) službu, což znamená, že funkce rozhraní vrátí návratovou hodnotu, která říká, zda se data podařilo doručit nebo ne. Některé protokoly (jako UDP) poskytují službu *nespolehlivou* (*unreliable*), takže funkce vrací pouze výsledek **odeslání**, nikoliv **doručení** dat. Detekci doručení a případnou reakci na nedoručení musí řešit aplikace sama.

Samozřejmou funkci vrstvy je *multiplexing*, v tomto případě přístup k síti pro různá spojení mezi lokálními á vzdálenými sockety (klienty i servery). Rozlišení jednotlivých komunikačních kanálů se děje prostřednictvím **portů**.

Některé protokoly (např. opět TCP) provádějí *segmentaci* dat. Jsou ochotny přijmout i bloky přesahující velikost jednotek dat, která lze poslat po připojené síti, data rozdělí do menších bloků (segmentů), ty posílá samostatně a přijímající strana je opět skládá do původní podoby. Přitom současně komunikuje s odesílající stranou, zda data dorazila, takže je schopná vyřešit případný výpadek nějakého segmentu.

Dodatečnou funkcí protokolu může ještě být *řízení toku*, neboli mechanizmus, jak ovlivňovat rychlost odesílání dat tak, aby se komunikační kanál využíval efektivně, ale přitom nedošlo k zahlcení.



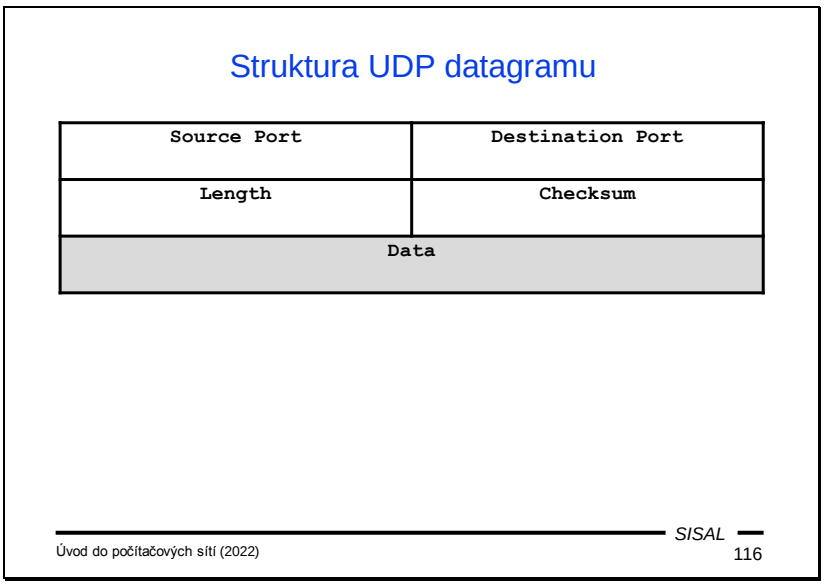
Majoritními protokoly používanými na transportní vrstvě TCP/IP jsou TCP a UDP.

TCP se používá pro **spojované aplikace**. Pro lepší představu lze spojovanou aplikaci přirovnat k telefonnímu hovoru. Volající zvolí číslo, jeho telefonní operátor vytvoří spojení k cílovému zařízení, a jakmile se hovor spojen, uživatel zahájí „aplikaci“, tj. začne hovořit a poslouchat. Aplikace má práci velmi jednoduchou, nemusí řešit, zda jednotlivé věty dorazily a zda dorazily ve správném pořadí. Podobně se chovají aplikace pracující nad TCP – klient naváže spojení a vznikne tzv. *stream*, kanál, kterým tečou data oběma směry. Protokol TCP je *spolehlivý*, neboli pokud aplikace nedostane zprávu o chybě přenosu dat, má jistotu, že všechna data dorazila v pořádku. Režie za tuto garanci je plně na straně TCP, a proto je poměrně komplikované a robustní; aplikace naopak mohou být relativně jednoduché. Pochopitelně, že režie TCP (potvrzování doručení, čekání na potvrzení, přeposílání ztracených dat) snižuje logickou přenosovou kapacitu a může způsobovat kolísání pravidelnosti doručování. Nevýhodou TCP je i to, že aplikace má velmi malé možnosti, jak reagovat na stav sítě – jakmile zavolá funkci pro přenos, nezbývá jí, než čekat, až se data přenesou nebo vyprší patřičné timeouty a funkce oznámí neúspěšné doručení.

UDP se používá pro **nespojované aplikace**. Jako analogii můžeme uvést komunikaci pomocí klasické pošty. Pokud vhodíme dopis do poštovní schránky, odpovídá to zavolání funkce pro odeslání a jejím výsledkem je pouze „povedlo se odeslat“ nebo „nepovedlo se odeslat“. Pokud odešleme víc dopisů, nemáme záruku, že dojdou všechny a že dojdou ve správném pořadí (a v počítačovém světě se dokonce může stát i to, že některé zprávy budou doručeny dvakrát). Pokud chceme vybudovat touto cestou spolehlivý kanál, musíme my („aplikace“) převzít na sebe režii. V UDP se posílají samostatné zprávy, UDP samo řeší jen přenos, takže má

velmi malou režii. Celá režie se přenáší na aplikaci, což ale má i výhodu v tom, že aplikace může pružněji reagovat na aktuální stav sítě. Na rozdíl od TCP mohou data v UDP téct pravidelněji, případnou ztrátu pak musí řešit aplikace – buďto sama zařídí přeposlání, anebo je schopna se bez chybějící části dat obejít.

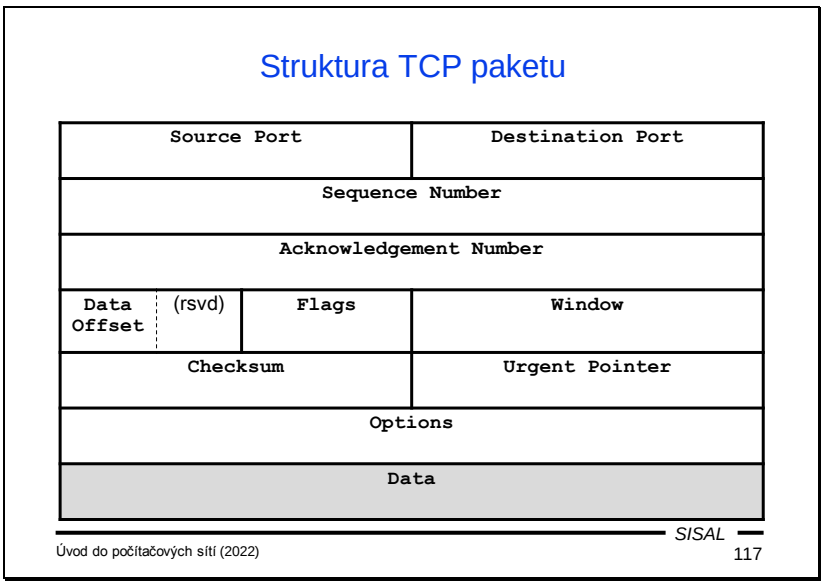
Snímek 116



Z předchozího popisu je zřejmé, že UDP nepotřebuje pro svou práci doplňovat k datové jednotce aplikační vrstvy mnoho informací a zapouzdření aplikačních dat do PDU transportní vrstvy je velmi jednoduché.

V UDP hlavičce se přenášejí pouze informace o multiplexingu, tj. zdrojový a cílový port, a řídící informace (délka a kontrolní součet).

Snímek 117



Hlavička TCP obsahuje oproti UDP řadu dalších informací.

Aby TCP mohlo garantovat kompletnost přenosu, musí k jednotlivým segmentům připojovat jejich identifikaci. Zjednodušeně můžeme říci, že každý segment obsahuje relativní **posun** (offset) vůči počátku streamu (v hlavičce vidíme tzv. *Sequence number*). Aby zároveň bylo možné potvrzovat doručení, potřebujeme analogické pole pro opačný směr (*Acknowledgement number*).

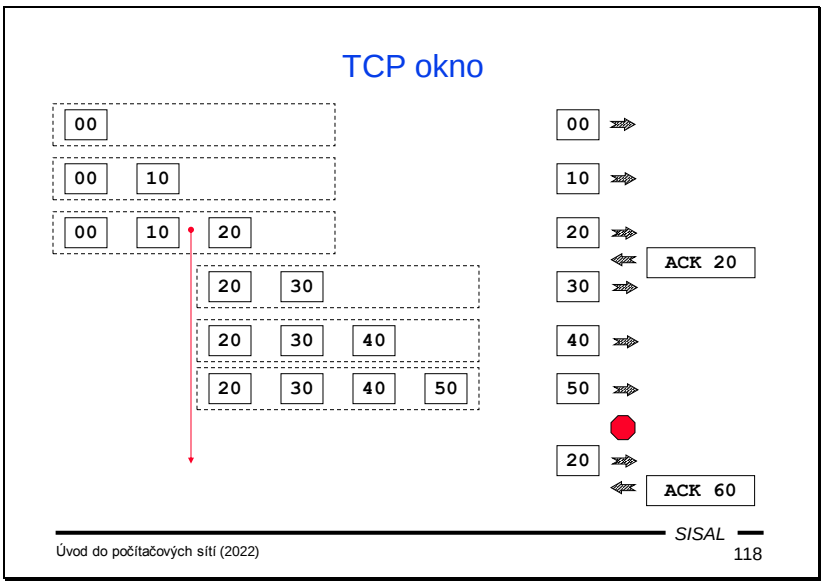
Pole *Flags* obsahuje příznaky, většinu z nich probereme vzápětí.

Pole *Urgent pointer* je určeno pro funkci *out-of-band* přenosu. Aplikace má právo označit určitá data za **urgentní** příznakem URG. Např. příkaz, kterým FTP klient chce přerušit datový přenos, je posílán jako urgentní. Taková data jsou pak vložena do normální komunikace a jejich relativní adresa v rámci datového bloku je zapsána v tomto poli.

K ostatním důležitým polím hlavičky se dostaneme podrobněji později.

Poznámka: Strukturu TCP paketu zde uvádím pro demonstraci jeho funkcí, zkoušet polohu jednotlivých polí se určitě nechystám.

Snímek 118



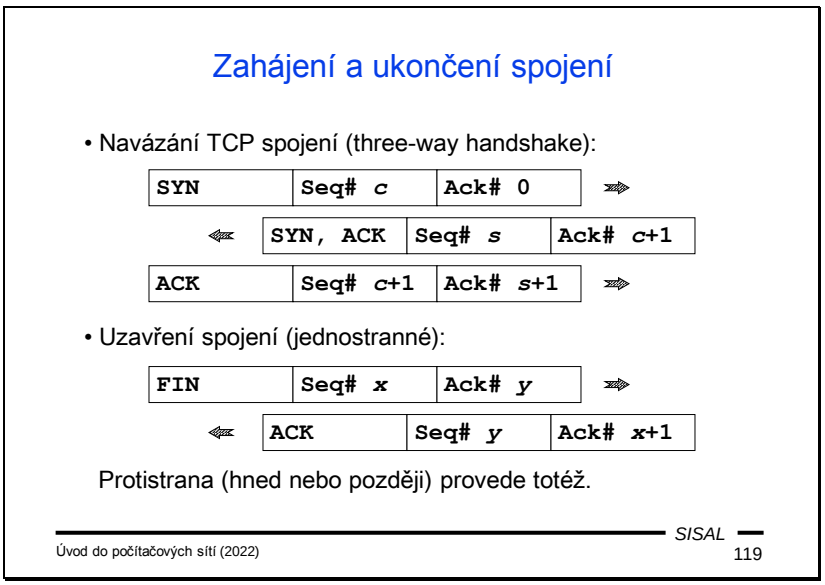
Podívejme se nyní blíže na to, jak se v TCP řeší potvrzování doručení a řízení toku dat.

Jak už bylo řečeno, TCP potvrzuje doručení dat. Příjemce potvrdí doručení bloku tak, že pošle odesilateli paket s nastaveným příznakem ACK a hodnotou Acknowledgement number nastavenou na offset konce dat, která byla doručena. Pokud by se ale tato potvrzení posílala za každý paket v samostatném paketu, představovalo by to zbytečnou zátěž. Proto je možné potvrzeni (příznak + offset) „přibalit“ k nějakým jiným datům, která příjemce potřebuje poslat zpátky. Obvyklá implementace je, že počítač čeká s odesláním ACK určitý čas, jestli se neobjeví data, která bude třeba poslat na druhou stranu spojení, aby k nim ACK připojil. Pokud se tak do daného limitu nestane, pošle ACK samostatně.

Stejně tak neefektivně by byl komunikační kanál využíván, pokud by před odesláním každého nového paketu musel odesilatel čekat na potvrzení doručení toho předchozího. TCP proto umožňuje, aby se obě strany domluvily na rozsahu dat, která smí odesilatel odeslat, aniž by čekal na potvrzení. Tomuto rozsahu se říká **velikost okna** a navrhovanou hodnotu oznamuje uzel v poli *Window* v TCP hlavičce.

Uvažujme situaci, kdy jedna strana spojení začne posílat bloky o velikost dat 10 B (reálně se používají samozřejmě bloky větší) při nastavené velikosti okna 40 B. Odesilatel začne odesílat bloky, aniž čeká na potvrzení, ale hlídá si maximální kapacitu okna. Pokud mezitím přijde nějaké potvrzení doručení, posune „okno“ na pozici, kterou potvrzení obsahuje. Např. byl-li to offset 20, může nyní posílat data až po offset 60. Pokud do té doby žádné další potvrzení nedorazí, musí přerušit odesílání a zahájit čekání na potvrzení. Pokud potvrzení nedorazí, musí zopakovat poslání prvního nepotvrzeného bloku dat.

Snímek 119



Pojem „TCP spojení“ označuje stav, kdy klient a server potvrdili ochotu spolu komunikovat a dohodli se na sekvenčních číslech, která budou používat. Ve skutečnosti totiž tento „offset“ datové komunikace nezačíná z bezpečnostních důvodů od nuly, ale od **náhodného čísla**, které si volí odesílající strana. Proto je nutné iniciální hodnotu protistraně poslat. Tato dohoda se odehrává na začátku spojení pomocí tří speciálních paketů, které mají prázdnou datovou část a nesou informace pouze v hlavičce, a nazývá se *three-way handshake*.

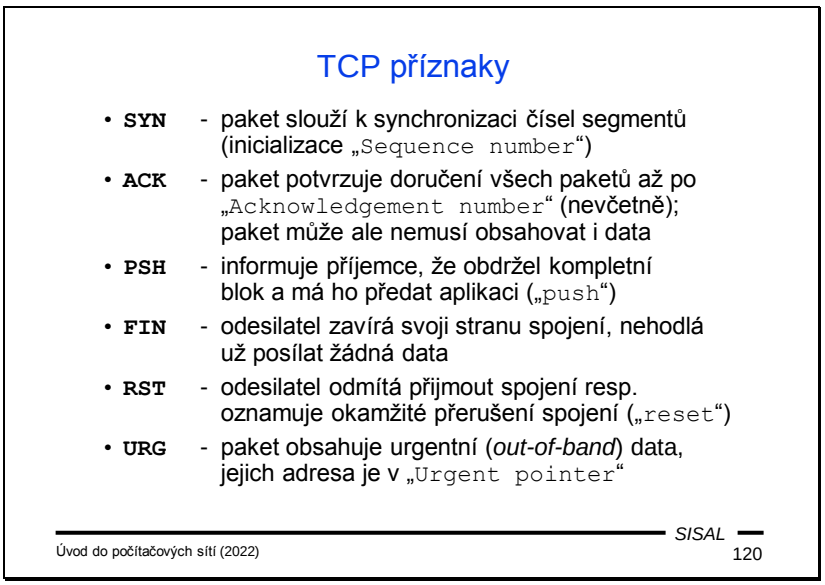
První z těchto paketů má nastavený příznak SYN (synchronizační paket) a jako Sequence number má nastavenu iniciální hodnotu zvolenou klientem, označme si ji *c*. Server potvrdí příjem tím, že pošle paket s příznakem ACK a hodnotou Acknowledge number nastavenou na *c* + 1. Současně ale vygeneruje svoji iniciální hodnotu sekvenčního čísla (*s*) a doplní rovněž příznak SYN. Klient poté dokončí proceduru posláním paketu, kde potvrdí příjem odesláním ACK s hodnotou *s* + 1.

Od této chvíle mohou obě strany posílat data a postupně o jejich délku navyšovat hodnoty sekvenčních čísel.

Pokud chce libovolná strana spojení ukončit, pošle paket obsahující příznak FIN. Tím sděluje protistraně, že už nehodlá posílat **žádná data**. Obvykle protistrana vzápětí rovněž pošle FIN paket, ale pokud spojení nechce ukončit, může svá data dál posílat. V tom případě bude muset i strana, co FIN už poslala, posílat dál své ACK **pakety**, aby se spojení nerozpadlo. Tuto situaci si můžeme snadno představit třeba v HTTP. Poté, co klient pošle svůj požadavek, může spojení ukončit a pouze čekat na odpověď serveru. Tomuto stavu se říká *jednostranně uzavřené* (*half-closed*) spojení a je legitimní, ačkoliv ne úplně doporučované, neboť různé kontrolní mechanizmy po cestě mohou spojení vyhodnotit jako mrtvé a ukončit ho. Proto by např. zmiňovaný

HTTP klient měl spojení nechat ze své strany otevřené a zavřít ho až v okamžiku, kdy už přijal celou odpověď.

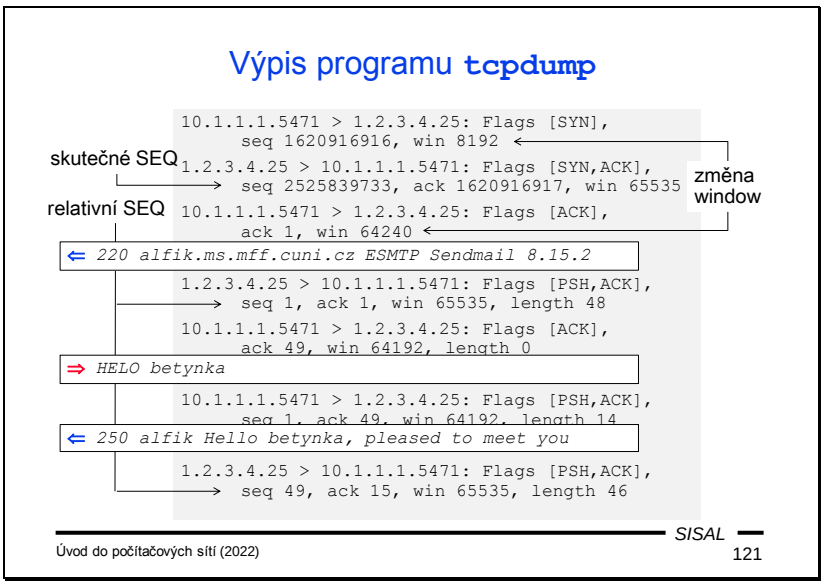
Snímek 120



Kromě již zmíněných příznaků URG, SYN, ACK a FIN jsou důležité ještě:

- PSH (push) – Tímto příznakem odesílající strana oznamuje protistraně poslední segment bloku dat. Pro příjemce to znamená, že pokud už obdržel všechny ostatní segmenty bloku, může ho předat k dalšímu zpracování aplikaci.
- RST (reset) – Tímto příznakem oznamuje odesílatel protistraně, že nehodlá v komunikaci dál pokračovat. Příznak může použít jednak server hned během three-way handshake jako odpověď v případě, že nechce spojení přijmout, anebo kterákoliv strana v průběhu komunikace. Na rozdíl od ukončení pomocí FIN znamená RST okamžitý konec bez čekání na potvrzení resp. FIN od protistrany.

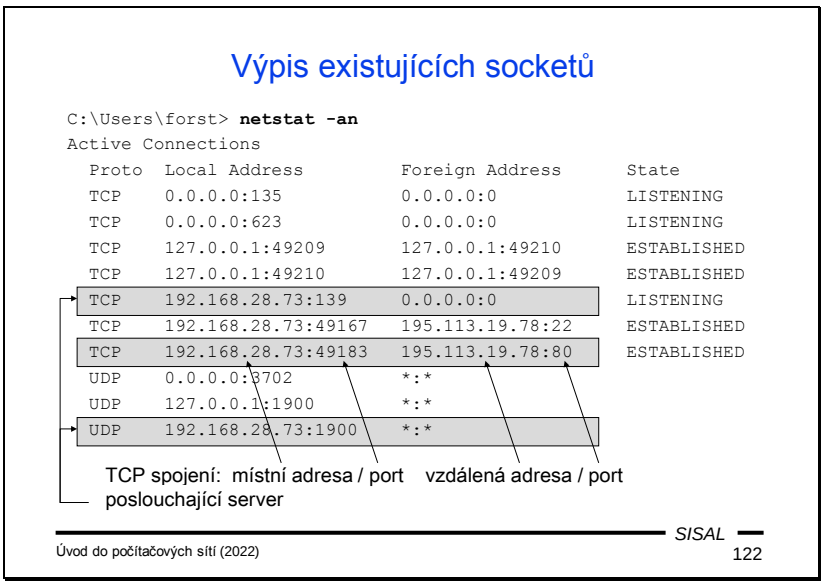
Snímek 121



Pokud si chceme podrobněji prohlédnout průběh TCP/IP komunikace, můžeme použít nějaký program, který ji umí zachytit a zobrazit v čitelné podobě (např. tcpdump, Wireshark, WinPcap). Podívejme se na ukázku výpisu prvního ze jmenovaných…

- První paket je první paket three-way handshake, což poznáme podle toho, že nese pouze příznak SYN. Na řádce vidíme nejprve zdrojovou IP adresu a tečkou oddělené číslo portu a za šipkou cílovou IP adresu a port. Jedná se o port 25, takže zjevně půjde o začátek SMTP spojení. Vidíme rovněž iniciální sekvenční číslo (**seq**) a velikost okna (**win**).
- Druhý paket je odpověď serveru (nese příznaky SYN a ACK), vidíme potvrzené klientovo sekvenční číslo (**ack**) a také návrh většího okna.
- V posledním paketu three-way handshake už vidíme hodnotu Acknowledgement number **relativně**. Nástroje jako tcpdump totiž pro lepší čitelnost zobrazují sekvenční čísla jako relativní posun vůči začátku komunikace (odečítají iniciální hodnotu).
- Další paket je úvodní zpráva serveru. Vidíme poprvé uvedenou délku dat (**length**, 48 znaků) a příznak PSH (push), protože řádka je kompletní.
- Klient zjevně trochu otálel s posláním svého prvního příkazu, takže TCP software poslal v pátém paketu samostatné potvrzení doručení pro úvodní řádku od serveru. Jako hodnotu Acknowledge number použil iniciální hodnotu zvětšenou o délku předchozích dat (v relativních číslech 1+48=49).
- V šestém paketu vidíme první příkaz klienta.
- Sedmý paket obsahuje odpověď serveru. Kromě samotných dat ale nese rovněž příznak ACK a hodnotu, která TCP vrstvě klienta potvrzuje přijetí předchozích 14 bajtů (v relativních číslech 1+14=15). Server zde tedy spojil odeslání potvrzení s vlastními daty.

Snímek 122



Pokud chceme získat představu o tom, jaká spojení jsou momentálně na našem počítači otevřená, můžeme použít příkaz **netstat**. S parametrem **–a** vypíše seznam všech TCP i UDP serverů a otevřených TCP spojení.

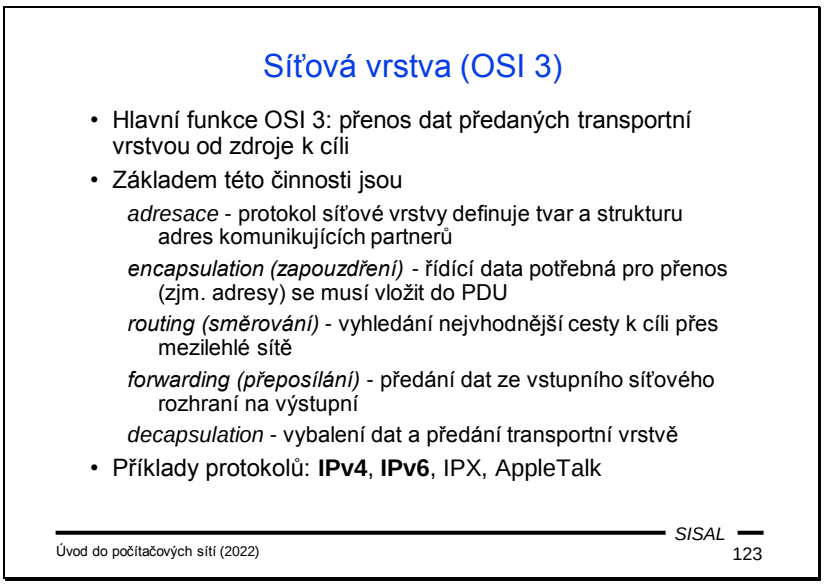
U TCP jako stavového protokolu obsahuje výpis v posledním sloupečku i stav spojení, na ukázce vidíme LISTENING (poslouchající server) a ESTABLISHED (otevřené spojení), ale jinak existuje stavů celá řada.

Ve druhém sloupečku máme adresu lokálního socketu, tam vidíme buďto skutečnou adresu některého síťového rozhraní, anebo adresu 0.0.0.0, což znamená, že daný server poslouchá na všech rozhraních, co na počítači jsou.

Ve třetím sloupečku je socketová adresa protistrany, zde je buďto skutečná adresa a port vzdáleného socketu, anebo v případě serveru hodnota 0.0.0.0 nebo *, což znamená, že připojit se může libovolný klient.

V případě UDP máme ve výpisu pouze běžící **servery**, protože UDP žádná **spojení nemá**, takže o výměně zpráv mezi klienty a servery netstat nemá informace.

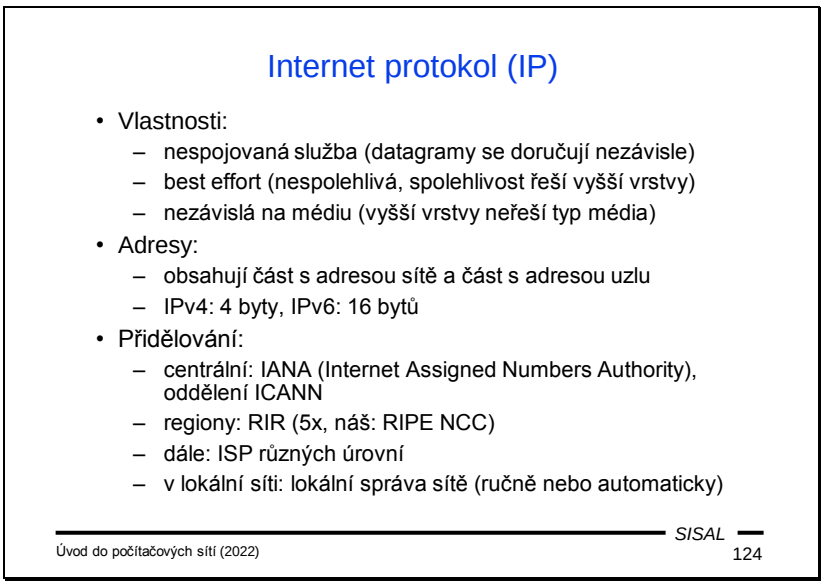
Snímek 123



Úkolem síťové vrstvy je vyhledání cesty k cílovému počítači tak, aby bylo možné doručit data předaná transportní vrstvou bez ohledu na technologie použité v nižších vrstvách.

Vrstva stojí na dvou základních pilířích:
- Adresní schéma – způsob, jak identifikovat jednotlivé uzly v síti tak, aby bylo možné rozpoznat, do jaké sítě patří.
- Směrování – způsob, jak na základě adresy najít správnou cestu ze zdrojové do cílové sítě.

Mezi další funkce vrstvy patří zapouzdření, které už jsme poznali dříve, a **forwarding**, neboli princip, kdy směrovač přijme paket, ačkoliv není konečným adresátem, a pokusí se ho doručit o další krok směrem k cíli stejně, jako by se doručoval paket vzniklý přímo na směrovači.

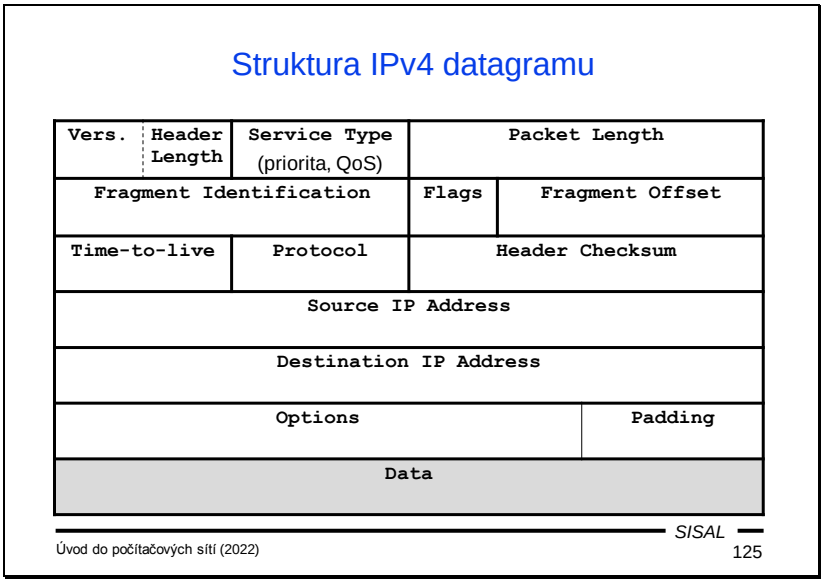


V TCP/IP se na síťové vrstvě používá Internet protokol (IP). Jedná se o službu
- nespojovanou – jednotlivé datagramy se doručují nezávisle, nevytvářejí se žádná formální spojeni
- nespolehlivou – síťová vrstva negarantuje doručení (stejně, jako jsme to poznali u UDP, které přebírá tuto vlastnost od IP)
- nezávislou na médiu – transportní vrstva obecně nemusí řešit detaily použité technologie (až na MTU, o němž si povíme později).

V současnosti existují dvě verze protokolu, verze 4 má adresu délky 4 bajty, u verze 6 je to 16 bajtů. U obou verzí existuje metoda, jak stanovit část s **adresou sítě** (která je zodpovědná za směrování) a část s **adresou uzlu** (která je zodpovědná za doručení cílovému uzlu).

IP adresu musí mít každý uzel, který chce v TCP/IP síti komunikovat. O způsobu přidělování adres připojeným koncovým stanicím rozhoduje správa sítě. Každá síť používá pro své uzly určité rozsahy adres – buďto privátní (ty si volí správa sítě sama), anebo veřejné (ty dostává přidělené od ISP, který ji připojuje do internetu). ISP dostávají bloky adres od nadřazených ISP, na vrcholu hierarchie stojí IANA (Internet Assigned Numbers Authority, součást ICANN), která má pod sebou pět regionálních registrátorů:
- RIPE NCC spravuje Evropu, Rusko a západní Asii
- APNIC spravuje zbytek Asie, Austrálii a Oceánii
- ARIN spravuje USA, Kanadu, Antarktidu a část Karibiku
- LACNIC spravuje Latinskou Ameriku a zbytek Karibiku
- AFRINIC spravuje Afriku.

Snímek 125

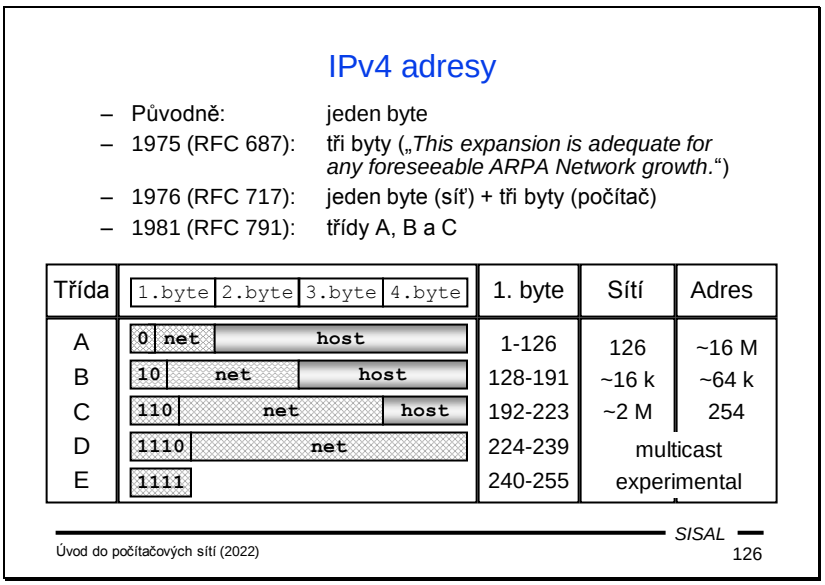


Zapouzdření na síťové vrstvě přidává IP hlavičku, která obsahuje informace důležité pro doručování. V případě protokolu verze 4 hlavička obsahuje následující informace:

- Verze. Na verzi je rezervována polovina bajtu, takže pro IP verze 16 už bude třeba jiný formát.
- Délka hlavičky. Ta zabírá druhou polovinu prvního bajtu. Pohledem na diagram je asi jasné, že hodnota 15 by na celou délku nestačila. Je tomu opravdu tak, délka se totiž udává ve **32bitových slovech** a ne v bajtech. To znamená, že maximální délka hlavičky je 60 bajtů a pokud její obsah není násobkem 4B, musí se doplnit „vatou“.
- Druhý bajt obsahuje QoS a zbytek prvního slova vyplňuje délka celého datagramu.
- Druhé slovo slouží pro **fragmentaci**. Tak nazýváme postup, kdy síťová vrstva dostane k doručení paket dlouhý tak, že při jeho zapouzdření by vznikl rámec delší než maximální povolená délka pro danou linkovou vrstvu (MTU, Maximum Transmission Unit). V takovém případě musí síťová vrstva paket fragmentovat na více datagramů a poslat je postupně. Fragmentace představuje zbytečnou komplikaci, pokud se jí lze vyhnout. To je případ TCP, které samo data dělí (segmentuje), takže by stačilo zjistit správnou velikost segmentu, kterou má použít. Proto se používá metoda hledání **Path MTU** – pakety se posílají s příznakem *Do not fragment*, díky němuž se odesilatel dozví o případných problémech s velikostí MTU po cestě, může určit správně velikost segmentu a předat ho transportní vrstvě.
- Třetí slovo obsahuje hodnotu Time-to-live, k níž se ještě vrátíme, číslo protokolu, který je v datagramu zapouzdřen, a kontrolní součet hlavičky.
- Poté následují IP adresy odesilatele a příjemce.
- Dále hlavička může obsahovat volitelné položky.

Poznámka: Strukturu IP datagramu zde uvádím pro demonstraci jeho funkcí, zkoušet polohu jednotlivých polí se určitě nechystám.

Snímek 126



Situaci, ve které se začaly budovat počítačové sítě, hezky ilustruje fakt, že původně byl pro adresu počítače vyhrazen **jediný bajt**! Ani autoři návrhu si zjevně nedokázali představit, jak dalece zasáhne náš každodenní život to, co právě začínají budovat.

V roce 1975 se rozhodlo, že adresu je třeba prodloužit. Nová délka byla stanovena na tři bajty a v RFC 687 je z dnešního pohledu úsměvná poznámka, že toto rozšíření je **dostačující pro jakýkoliv myslitelný růst** sítě!

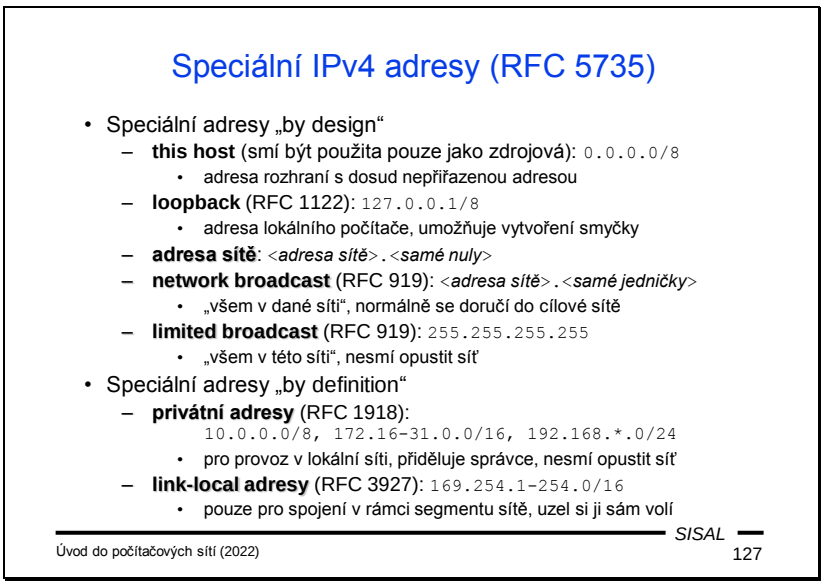
Hned o rok později se adresa znovu rozšiřovala. Ne snad, že by tak rychle adresy došly, ale adresa byla doplněna o část pro adresu sítě (1B) a původní 3B zůstaly pro adresu počítače.

O dalších pět let později se ukázalo, že počet sítí roste daleko rychleji, než se čekalo, původní model byl zjemněn a vznikly tři třídy adres podle velikosti sítě. Třída A pokrývá dolní polovinu adresního prostoru a zůstalo jí původní dělení 1:3. Třída B zabírá třetí čtvrtinu prostoru a adresa se zde dělí půl na půl. Třída C zabírá předposlední osminu a dělí se 3:1, což znamená, že nabízí zhruba 2 miliony sítí o rozsahu max. 254 počítačů.

Poznámka: Možná byste čekali namísto hodnoty 254 pro počet adres v C síti spíše 256, ale první a poslední adresa v každé síti má vyhrazený význam.

Poslední osmina byla vyhrazena pro další rozvoj a v 1986 se z ní ještě oddělila třída D pro tzv. *multicastové* adresy. U nich zcela chybí část pro adresu počítače, protože se používají pro oslovení skupiny příjemců – pro speciální služby (např. NTP používá adresu 224.0.1.1), videokonference apod.

Snímek 127



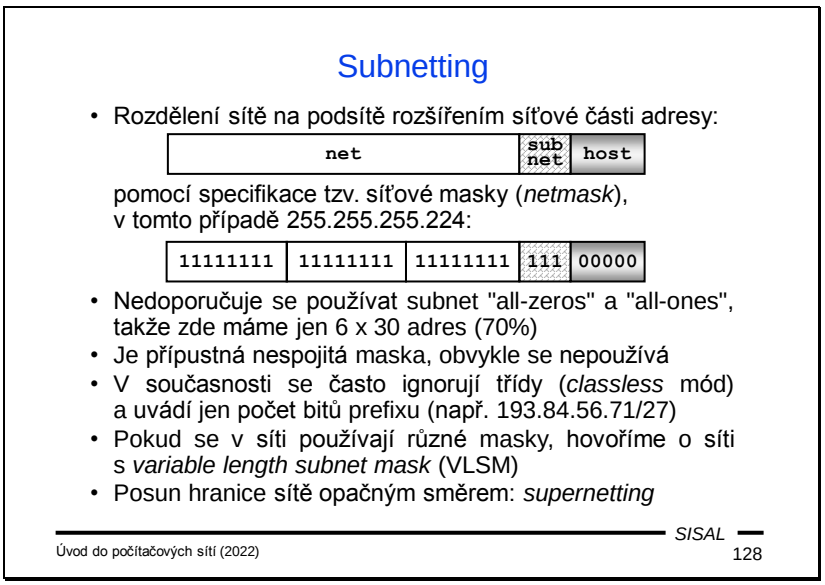
Některé IP adresy mají z definice protokolu speciální význam:

- Adresa se samými nulami se používá jako zdrojová v situaci, kdy potřebujeme komunikovat, ale ještě neznáme svoji adresu.
- Adresa 127.0.0.1/8 je vyhrazena pro speciální rozhraní existující na každém uzlu sítě a představující „**tento počítač**“, tzv. *loopback adresa*. Když potřebujeme navázat komunikaci mezi klientem a serverem, kdy oba běží na našem počítači, mohou oba používat tuto adresu.
- IP adresa s nulovou částí pro adresu počítače je *adresa sítě*.
- Poslední IP adresa v bloku adres nějaké (pod)sítě představuje *síťový broadcast*, neboli adresu, kterou můžeme použít, chceme-li oslovit všechny počítače v dané síti.
- Vedle síťového broadcastu, který se normálně doručuje mezí sítěmi, existuje ještě *omezený broadcast* (255.255.255.255), který nesmí opustit síť, kde vznikl.

Kromě těchto speciálních adres existují ještě dvě kategorie adres, jimž byl jejich speciální význam stanoven organizačním rozhodnutím, nikoliv architekturou IP:

- První skupinou jsou *privátní* adresy. K dispozici je jedna síť třídy A, 16 sítí třídy B a 256 sítí třídy C. Jak už víme, používají se v lokálních sítích a lokální sít mohou pakety s touto adresou opustit pouze ve speciálních případech. Jinak se musí používat překlad adres (NAT)
- Druhou skupinou jsou *link-local* adresy. Tyto adresy jsou volně k dispozici, každý počítač si může libovolnou tuto adresu vzít a zahájit komunikaci na segmentu sítě, kde je připojen. Na rozdíl od privátních adres se pomocí nich ale vůbec nedá komunikovat mimo vlastní síť.

Snímek 128



Postupem času se ale ukázalo, že i jemnější členění na třídy není dost jemné a zejména v lokálních sítích je třeba posunout hranici mezi oběma částmi adresy ještě více „doprava“, neboli vytvořit z jedné sítě několik **podsítí**. V takovém případě ovšem při konfiguraci síťového rozhraní nestačí zadat adresu, ale je třeba specifikovat rovněž rozsah sítě. Jednou z možností, jak to udělat, je doplnit tzv. **síťovou masku**, číslo, které obsahuje jedničky právě na místech, kde je adresa sítě. Např. chceme-li posunout hranici sítě u adresy třídy C o tři bity, použijeme hodnotu 255.255.255.224. Volba způsobu zadávání je vcelku logická – pokud máme nějakou adresu a chceme vědět, zda patří do naší sítě, stačí vzít síťovou masku a dotyčnou adresu, provést operaci *bitového AND*, a pokud výsledkem bude adresa naší sítě, je daná adresa „naše“. Zápis síťové masky je ale trochu složitý na výpočet, takže se dnes často nahrazuje tzv. *classless* formátem, kdy zapisujeme za lomítko pouze počet bitů, které tvoří síťovou část.

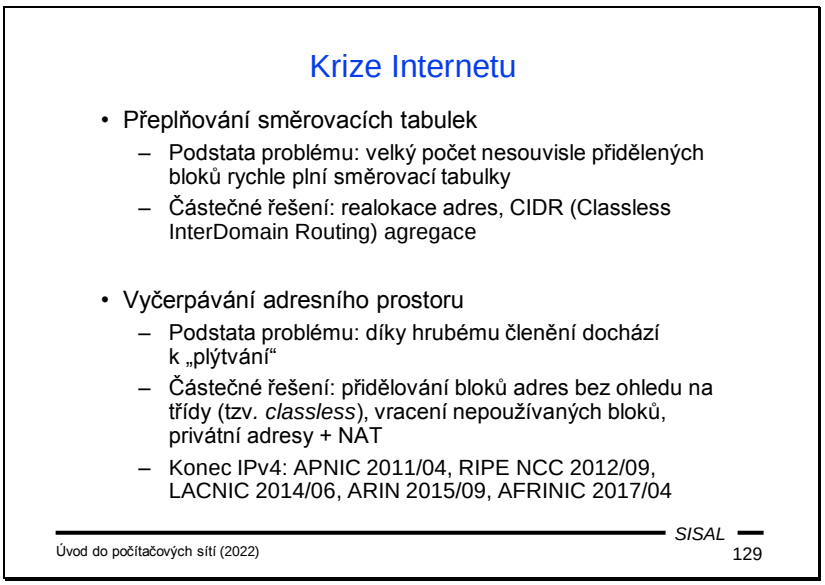
Používání subnettingu umožňuje rozčlenit síť na menší celky, ale redukuje počet použitelných adres. Vzhledem k tomu, že se úplně nedoporučuje použití podsítě se samými nulami a samými jedničkami, subnetting z našeho příkladu omezí síť na 6 podsítí po 30 počítačích.

Toto dělení je navíc opět velmi hrubé, a pokud bychom potřebovali flexibilnější dělení, nevystačíme si s jednou maskou. Např. pokud z našich 6 podsítí má jedna více než 30 počítačů, ale dvě mají nejvýše 14 počítačů, při subnettingu s pevnou maskou nám jedna adresa třídy C nebude stačit. Budeme muset použít tzv. *Variable Length Subnet Mask* (VLSM) a ve dvou malých podsítích použít masku /28, zatímco ve velké síti /26.

Přirozenou otázkou je, zda lze hranici posouvat také opačným směrem, tj. místo rozdělování sítě spojovat. To se hodí např. v situaci, kdy mám v LAN více počítačů než 254 a mám pro ně dvě sítě třídy C. Pokud bych nepoužil *supernetting* a nespojil je do jedné sítě s maskou /23, musel bych řešit zbytečné komplikace při komunikaci dvou počítačů, které ačkoliv jsou v jedné LAN, mají náhodou adresy z různých sítí třídy C.

Naštěstí se dnes masivně používají v LAN privátní adresy, kde je k dispozici dostatečný počet sítí ve všech třídách, takže subnetting není třeba řešit tak často.

Poznámka: Pokud zkoumáme nějakou IP adresu, která nepatří do naší sítě, pochopitelně nemůžeme vědět, jaký subnetting tamější síť používá. To ale nevadí, protože nás to nezajímá. Z pohledu směrování se k cizí adrese chováme, jako by žádný subnetting nepoužívala – naším úkolem je doručit paket do cizí sítě a finální doručování v této síti už je plně v její režii.



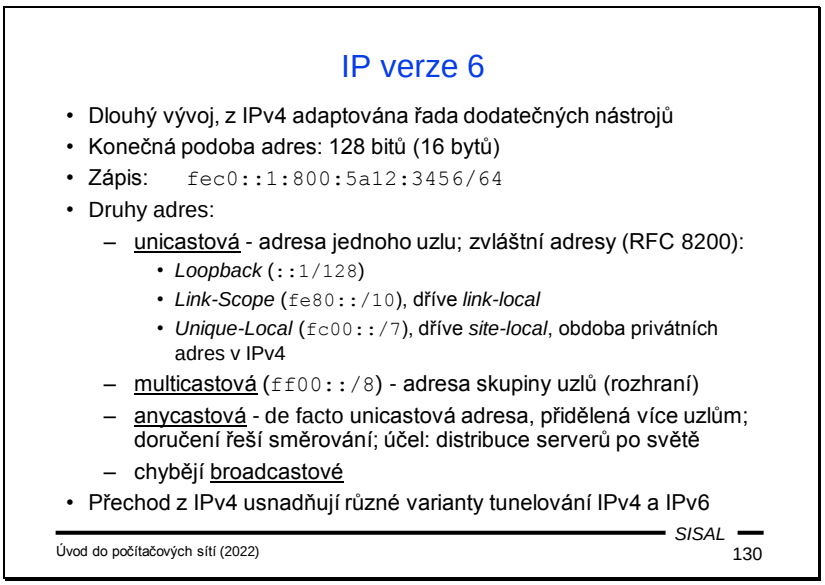
Všeobecný pocit dostatku IP adres v duchu troufalé věty z RFC 687 vedl k tomu, že jejich přidělování se neřídilo žádnými přísnými pravidly. Na přelomu 80. a 90. let se začalo ukazovat, že to začíná přinášet problémy ve dvou směrech.

Jednak se začaly přeplňovat tabulky centrálních směrovačů. Tomu se dalo předcházet tím, že pokud spolu v reálu sousedily sítě, které spolu sousedily i svojí numerickou hodnotou, bylo možné jejich záznamy na většině směrovačů **agregovat**, tj. vlastně udělat nad nimi supernetting a spojit oba záznamy do jednoho s kratší síťovou maskou. Tento princip je tzv. Classless InterDomain Routing (CIDR), ale jeho efektivnímu zavedení muselo předcházet masivní přečíslovávání mnoha sítí v internetu, tak aby je vůbec bylo možné efektivně agregovat.

Současně začínalo být jasné, že přidělování libovolných rozsahů libovolným sítím povede dříve či později k vyčerpání adresního prostoru IPv4. I zde se tady začalo šetřit a přečíslovávat, přidělovat bloky, které neodpovídaly členěním podle tříd apod. Naštěstí se začalo masivně zavádět používání privátních adres a NAT, což dramaticky snížilo potřebu veřejných IP adres (jedna síť s tisícem počítačů potřebuje s NAT jen jednu adresu místo tisíce).

I přes to ale bylo jasné, že tyto metody problém jen oddálí, ale nevyřeší, a začaly intenzivní práce na přípravě tzv. IP next generation (IPng).

Snímek 130



Nová verze vznikala poměrně dlouho, stejně tak jako poté její zavádění do různých operačních systémů a odlaďování problémů. Naštěstí CIDR a NAT oddálily problém, takže než bylo IPv6 začít opravdu nutně nasazovat, byla už připravena poměrně dobře. Navíc bylo počítáno s dlouhým postupným přechodem, který usnadňují různé varianty tunelování IPv6 do IPv4 a obráceně. Z předchozí verze navíc převzali autoři řadu dodatečné doplněných nástrojů a vlastností a po důkladné analýze je upravili a začlenili přímo do návrhu IPv6.

V konečné podobě má adresa 16 bajtů a zapisuje se po 16bitových slovech v hexadecimální podobě s CIDR zápisem masky. Aby současné adresy obsahující dlouhé sekvence nul bylo možné zkracovat, lze (jednu) libovolnou sérii nulových slov vypustit (takže pak jsou v zápisu dvě dvojtečky za sebou).

Z hlediska typů adres najdeme v IPv6 hodně shodných i odlišných rysů. Základní, *unicastové*, adresy obsahují kromě veřejných rozsahů rovněž loopback, link-local adresy a unique-local adresy. Posledně jmenovaná skupina je určitou analogií privátních adres v IPv4. Větší roli než v IPv4 hrají *multicastové* adresy. V IPv6 totiž chybějí **broadcastové** adresy a jsou de facto nahrazeny multicastovou adresou ff02::1 („All IPv6 devices“).

Úplnou novinkou jsou adresy *anycastové*. Používají se v různých situacích, kdy je nějaký zdroj dostupný v geograficky oddělených místech a máme zájem, aby nejvhodnější zdroj „vybrala síť sama“. Jedna anycastová adresa může být nastavena všem dotyčným počítačům a paket poslaný na tuto adresu bude směrován k nejbližšímu cíli, aniž ho klient musí explicitně vybírat.

Souhrn 6

- Proč a jak se synchronizuje čas na počítačích v síti?
- K čemu slouží DHCP?

- Jak se v TCP/IP řeší chybějící prezentační vrstva?

- Vysvětlete rozdíl mezi TCP a UDP.
- Jak souvisejí pojmy potvrzování a TCP okno?
- Co je three-way handshake?

- Jak souvisejí třídy IPv4, subnetting a CIDR?
- Proč je v síti třídy C pouze 254 adres pro počítače?
- Jaké typy broadcastových adres známe?