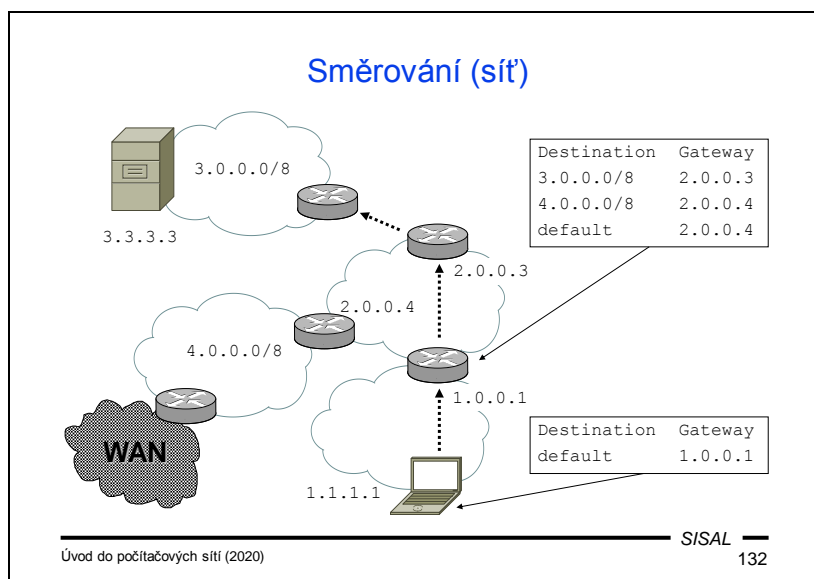


Problematiku směrování můžeme přirovnat k rozhodování řidiče auta na křižovatce. Řidič má určitý cíl a používá směrové tabule, aby si vybral výjezd z křižovatky. Situace na obrázku schematicky znázorňuje, jak by mohla vypadat křižovatka v České Lípě při příjezdu ze severu.

Řekněme, že řidičův cíl jsou České Budějovice. Ty ovšem na cedulích nejsou. Řidič proto musí použít geografickou znalost a usoudit, že je-li téměř přesně na sever od Prahy a jeho cíl je téměř přesně na jih od Prahy, lze říct, že České Budějovice jsou „v okolí Prahy“ a směr Praha je pro něj dobrá volba. Kdyby toto rozhodování nedělal v České Lípě, ale někde mezi Prahou a Budějovicemi, např. v Táboře, pravděpodobně by volba směru na Prahu nebyla správná. Pokud by si řidič nedokázal z cedulí vybrat, musel by použít směr „Ostatní tranzit“.

Pokud řidičův cíl budou Zákupy, je určitě pravda, že Zákupy jsou v okolí Prahy, ale protože má k dispozici daleko přesnější směrovou tabuli, bude se řídit jí, a nikoliv směrem na Prahu nebo pro Ostatní tranzit.



Analogický postup používá síťový software, když hledá pro cestu paketu další směrovač, tzv. *next-hop router*. Používá přitom tzv. *směrovací tabulku*, jejíž obsah má pod kontrolou operační systém. Záznamy v ní se trochu podobají směrovým cedulím na křižovatce. Také na nich je **směr**, což v počítačovém světě znamená next-hop router (*gateway*), a je na nich **cíl** (*destination*). Výhodou směrovací tabulky je, že cíl obsahuje vždy adresu sítě včetně jejího rozsahu zadaného **síťovou maskou** (*netmask*), takže zde nejsme závislí na žádné geografické znalosti a víme přesně, jaké „okolí“ cíle je pro nás relevantní.

Na obrázku máme část sítě, do níž je zapojen náš notebook s adresou 1.1.1.1, který chce poslat paket na adresu 3.3.3.3.

Typická konfigurace koncové stanice odpovídá situaci řidiče na parkovišti supermarketu. Tam nejspíše bude jen jediná směrová cedule „Výjezd“. Stejně tak na našem notebooku bude ve směrovací tabulce pouze jediný záznam, tzv. *defaultní záznam*, který bude ukazovat na směrovač (1.0.0.1), přes nějž je naše síť připojena.

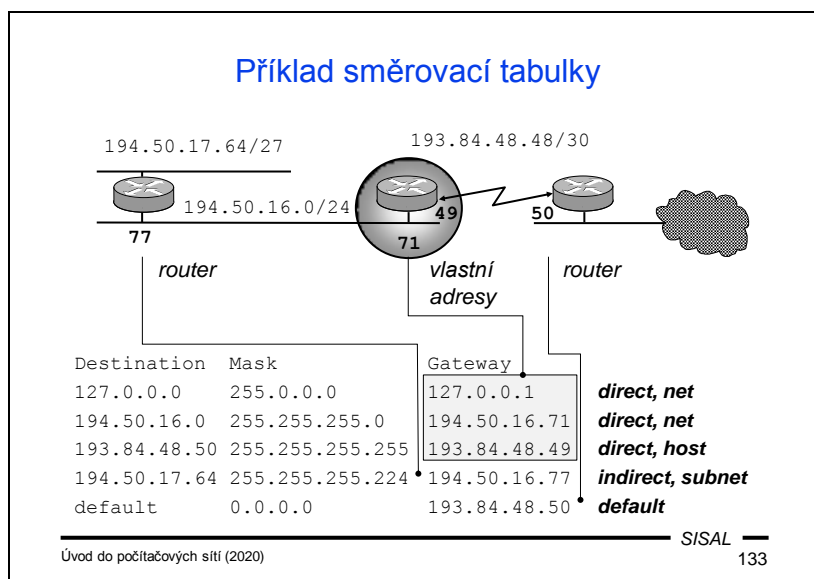
Jakmile náš paket dorazí na router 1.0.0.1, proběhne na něm prohledávání jeho směrovací tabulky. Tady už bude tabulka trochu komplikovanější. Vidíme v ní tři záznamy:

- Do sítě 3.0.0.0 s maskou /8 (čili vlastně sítě třídy A) se dostaneme přes router 2.0.0.3.
- Do sítě 4.0.0.0 s maskou /8 se dostaneme přes router 2.0.0.4.
- Do ostatních sítí se dostaneme také přes router 2.0.0.4.

Které z těchto záznamů vyhovují našemu cíli? Stačí se u každého záznamu podívat na masku a porovnat odpovídající počet prvních bitů adresy záznamu v tabulce s naší cílovou adresou:

- První záznam má masku 8 bitů, porovnáme tedy první bajty obou adres, oba mají hodnotu 3, takže záznam vyhovuje.
- U druhého záznamu porovnááme rovněž první bajt, ale ten se neshoduje (4 není rovno 3).
- Třetí záznam lze považovat za záznam s maskou **0 bitů**, porovnáme tedy 0 bitů obou adres a opět dojdeme ke shodě. Třetí záznam tedy rovněž vyhovuje.

Máme nyní dva vyhovující záznamy a musíme si z nich vybrat. A to je stejná situace, jako když si řidič jedoucí do Zákup vybírá mezi cedulí Zákupy a Praha – vybere si ten směr, kde mu stačí pro shodu **menší** okolí (okolí Prahy, které pokryje Zákupy, by muselo být velké aspoň 100 km). I zde si počítač vybere ten záznam, který má menší okolí, a tedy **širší masku**. V našem případě tedy padne volba na záznam 3.0.0.0/8 a ne default (což je vlastně 0.0.0.0/0).



Podívejme se teď na příklad směrovací tabulky podrobněji. Představme si situaci z obrázku, kde máme router propojující LAN na levé straně dvoubodovým propojením na router ISP napravo.

V tabulce budou tři tzv. **přímé** záznamy, neboli záznamy, které popisují přímo připojenou síť. Takový záznam vznikne v tabulce automaticky, jakmile nakonfigurujeme patřičné síťové rozhraní.

- První záznam popisuje formální síťové rozhraní s loopback adresou. Ve sloupečku *gateway* není v případě přímých záznamů žádný next-hop router, ale **naše vlastní adresa**, kterou jsme připojeni do sítě (zde 127.0.0.1). To má dobrý smysl – když software vybere tento záznam, ví, že už nemusí hledat žádný router, ale paket prostě jen odeslat pomocí síťového rozhraní s danou adresou.
- Druhý záznam popisuje hlavní síť naší LAN s adresou třídy C 194.50.16.0. Ve sloupečku *gateway* máme opět vlastní adresu 194.50.16.71.
- Třetím přímým záznamem je záznam pro point-to-point síť, kterou jsme připojeni k ISP. Tady si můžeme všimnout toho, že cíl má dokonce masku /32, a tedy jeden jediný stroj, protože na konci point-to-point spojení je jen jediný stroj, router ISP s adresou 193.84.48.50.

Dalším záznamem je **nepřímý** záznam ukazující na jednu z podsítí v naší LAN, a to konkrétně 194.50.17.64/27. Jako *gateway* zde bude uveden vnitřní router v naší síti. Jen je třeba si uvědomit, která z adres tohoto routeru v naší tabulce bude. Router má samozřejmě adresy v obou sítích 194.50.17.64/27 i 194.50.16.0/8. V naší tabulce musí pochopitelně být adresa ze sítě, kterou známe (naší), protože jinak by nám tento záznam při směrování moc nepomohl.

Posledním záznamem je **default**, který veškerý ostatní provoz směruje na router ISP.

Principy směrování

- Směrování by měla umět každá stanice v TCP/IP síti
- Záznam ve směrovací tabulce obsahuje „sloupce“:
cíl, maska, gateway
- Maska vyjadřuje „uvažovanou část“ adresy cíle
- Dřívější členění cílů: host (/32), net, default (/0)
- Typy záznamů:
 - *direct* (přímo připojená síť, „gateway“ je vlastní adresa)
 - *indirect, default*
- Vznik záznamu:
 - *implicitní* (automaticky po přiřazení adresy rozhraní)
 - *explicitní* („ručně“ zadán příkazem)
 - *dynamický* (v průběhu práce od partnerů v síti)

Pojďme si nyní shrnout poznatky o směrování.

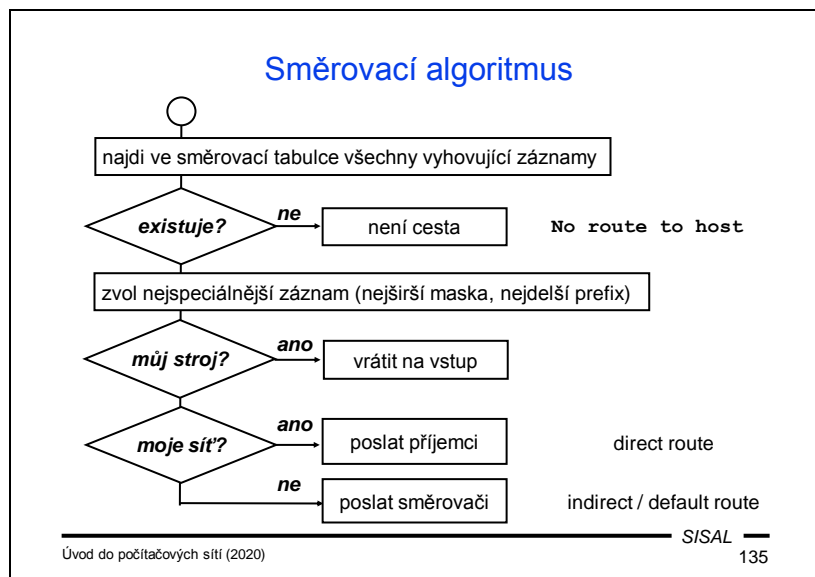
Směřovat by měla být schopna každá stanice pracující v síti TCP/IP. Porovnává přitom cílovou IP adresu v paketu se záznamy ve směrovací tabulce. Tyto záznamy obsahují tři základní údaje:

- adresu sítě, pro niž záznam platí (*destination*)
- maska (rozsah) této sítě (*netmask*)
- *gateway*, což je buďto
 - next-hop router, kterému je třeba paket předat, pokud se jedná o záznam nepřímý (vede do cizí sítě)
 - adresa některého vlastního síťového rozhraní, pokud se jedná o záznam přímý (vede do přímo připojené sítě).

Z hlediska vzniku dělíme záznamy na

- *implicitní* – vznikne automaticky nakonfigurováním síťového rozhraní
- *explicitní* – do tabulky se přidá příkazem, který buďto zadá administrátor ručně, nebo ho zavolá operační systém při startu počítače
- *dynamický* – záznam se vytvoří až v průběhu práce za pomoci informací obdržených od dalších uzlů v síti.

Dříve se záznamy dělily ještě podle masky, záznamům s maskou /32 se říkalo *host route*, záznamům s užší maskou buďto *net* nebo *subnet route*.



Vlastní směrovací algoritmus pak můžeme popsat následujícím způsobem:

- Ve směrovací tabulce se vyhledají všechny záznamy, které se shodují s cílem paketu.
- Pokud nebyl žádný záznam nalezen, paket nelze doručit. Tento případ ale může nastat pouze tehdy, pokud daná směrovací tabulka neobsahovala **defaultní záznam** (tedy „Ostatní tranzit“ z analogie s křižovatkou). Pokud by ho obsahovala, tak alespoň tento záznam vždy bude vyhovovat.
- Z nalezených záznamů se vybere ten nejspeciálnější, ten s nejširší maskou. Případný defaultní záznam se tedy použije pouze v případě, že tabulka neobsahuje jiný záznam, který by vyhovoval cíli.
- Pokud záznam odkazuje na náš vlastní počítač (loopback), paket se zařadí na vstup, jako by právě přišel ze sítě.
- Pokud je nalezený záznam přímý, paket se přímo předá linkové vrstvě k odeslání příjemci.
- Pokud je nalezený záznam nepřímý, paket se předá linkové vrstvě s příkazem poslat ho next-hop routeru.

Konfigurace sítě

UNIX

- IP adresa: `ifconfig interface IP_adr[netmask maska]`
- defaultní router: `route add default router`
- DHCP: `dhclient interface`
- často uložené v konfiguračním souboru, liší se podle typu OS

Windows

Control Panel ⇒ Network and Internet
⇒ Network Connections
⇒ Local Area Connection ⇒ Properties
⇒ TCP/IPv4 ⇒ Properties
⇒ General

Úvod do počítačových sítí (2020) SISAL 136

Nyní již známe nejdůležitější parametry, které je třeba zadat, když chceme nakonfigurovat na nějakém počítači síť TCP/IP. Pojďme se tedy podívat, jak se to dělá.

Na UNIXových systémech se nastavení ovládá následujícími příkazy:

- adresa síťového rozhraní se nastavuje příkazem **ifconfig**
- záznamy v routovací tabulce se mění příkazem **route**
- pokud chceme nechat konfiguraci na DHCP, je třeba spustit příkaz **dhclient**.

Příkazy jsou prakticky stejné na všech UNIXových systémech, na každém ale existují administrátorské nástroje, které konfiguraci umožňují snáze zadat a uložit, a tyto nástroje už se výrazně odlišují.

Na systémech Microsoft Windows se parametry sítě zadávají podobně jako jiná nastavení – ve starších systémech pomocí Ovládacích panelů, v novějších pomocí nástroje Nastavení.

Internet Control Message Protocol

- ICMP slouží pro posílání řídicích informací pro IP, např.:
 - Echo, Echo Reply** ... testování dosažitelnosti počítače (**ping**)
 - Destination Unreachable** ... nedostupný stroj, služba, síť, zakázaná fragmentace
 - Time Exceeded** ... vypršel Time-to-live (chyba v routování)
 - Source Quench** ... žádost o snížení rychlosti toku datagramů
 - Router Solicitation, Router Advertisement** ... vyhledávání routerů
 - Redirect** ... výzva ke změně záznamu v routovací tabulce
 - Parameter Problem** ... chyba v záhlaví datagramu
- Používá IP datagramy, ale není to transportní protokol
- ICMPv6 podstatně doplněn a rozšířen (např. o zprávy pseudoprotokolu Neighbor Discovery Protocol)

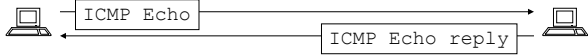
Před dalším výkladem se musíme seznámit s jedním pomocným protokolem, který IP síť používá pro rozesílání informací o různých situacích a které pak lze využít pro lepší řízení sítě. Jedná se o protokol ICMP. Jeho zprávy se posílají jako IP datagramy, takže ho lze zařadit nad třetí vrstvu OSI, zároveň ho ale nelze zařadit na čtvrtou vrstvu jako transportní protokol. Ve verzi 6 došlo k podstatnému rozšíření jeho funkcí, např. se pomocí něj posílají zprávy protokolu NDP (Neighbor Discovery Protocol), kterým stanice dokáže sama zjistit stav okolní sítě.

Nejdůležitějším typům zpráv se budeme dále věnovat, proto v této chvíli zmíním jen **Destination Unreachable**. Touto zprávou stanice nebo router oznamuje, že nemá jak doručit paket. V případě routeru je to obvykle způsobeno neúspěchem při hledání v routovacích tabulkách. V případě stanice tato zpráva nejčastěji znamená, že došlo k pokusu doručit data nějaké **UDP** službě, která není **dostupná** (v případě TCP se to řeší pomocí odpovědi s příznakem RST na úrovni transportního protokolu, ale v UDP taková možnost není, takže to řeší ICMP).

Ping

- Základní prostředek pro diagnostiku sítě

betynka:~> ping alfik



```

PING alfik.ms.mff.cuni.cz (195.113.19.71): 56 data bytes
64 bytes from 195.113.19.71: icmp_seq=0 ttl=64 time=0.214 ms
64 bytes from 195.113.19.71: icmp_seq=1 ttl=64 time=0.323 ms
64 bytes from 195.113.19.71: icmp_seq=2 ttl=64 time=0.334 ms
^C
--- alfik.ms.mff.cuni.cz ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.214/0.290/0.334/0.054 ms

```

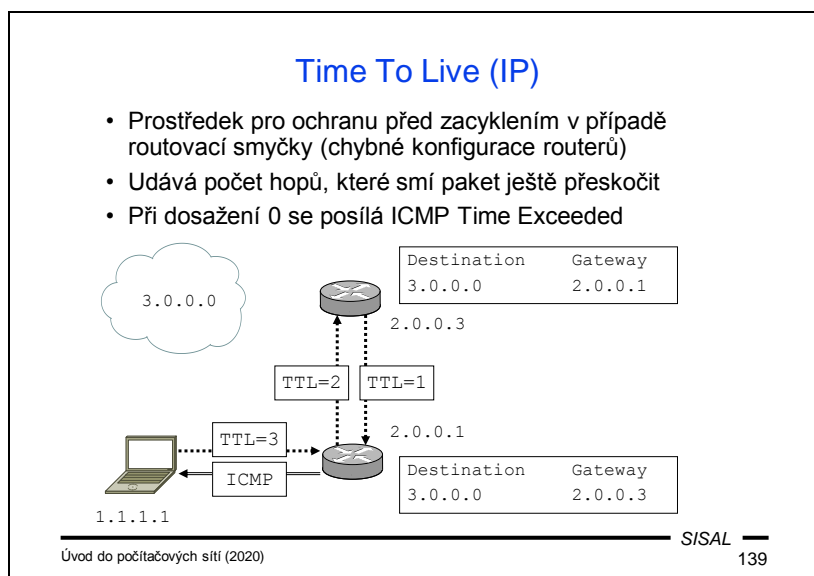
- na cílovém uzlu nemusí běžet žádný speciální program
- nezaručuje dostupnost služeb (pouze síťové vrstvy)

Úvod do počítačových sítí (2020) S/SAL 138

Zprávy **ICMP Echo** a **ICMP Echo reply** používá program **ping**, základní nástroj pro diagnostiku sítě. Testujeme s ním dostupnost vzdáleného uzlu sítě.

Když zavoláme program ping, začne periodicky vysílat zprávy ICMP Echo. Pokud zpráva dorazí na cílový stroj, jeho síťový software odpoví zprávou ICMP Echo reply. Pokud odpověď dorazí zpátky, ping vypíše řádek s informací o době, jak dlouho trvalo, než dorazila. Program vysílá zprávy s periodou 1 vteřina, dokud ho nepřeručíme nebo dokud neodešle předem stanovený počet zpráv. Poté vypíše statistiku – tj. počet a procento došlých odpovědí a minimum, maximum, průměr a směrodatnou odchylku hodnoty nazývané *round-trip time* (obousměrné zpoždění).

Důležitým faktem je, že na cílovém stroji nemusí běžet žádný speciální server, na ICMP odpovídá síťový software sám. Programem ping ovšem ověříme pouze to, zda se naše pakety dokážou dostat k cíli a zpět, nikoliv to, zda nám daný cíl bude ochoten odpovídat i na nějaké aplikační služby. Ve skutečnosti může dokonce nastat opačný případ, a to, že s nějakým počítačem dokážeme vytvořit HTTP spojení, ačkoliv ping neprošel – jednak může mít server odesílání odpovědi ICMP Echo reply konfiguračně zakázané a jednak mohou být ICMP Echo/Echo reply pakety odstraněny některým routerem po cestě.



Další důležitou zprávou ICMP je **Time Exceeded**. Souvisí s polem **TTL** (Time-to-live) v IP záhlaví a společně mají za úkol zbránit tomu, aby kvůli chybě v routovacích tabulkách nějaké pakety běžely v nekonečné smyčce.

V situaci na obrázku došlo k tomu, že mezi routery 2.0.0.1 a 2.0.0.3 vznikla routovací smyčka na cestě k síti 3.0.0.0. Oba routery mají ve svých tabulkách záznam, že pakety pro dotyčnou síť se mají posílat druhému routeru. Může k tomu dojít buďto chybou administrátora nebo chybou software. TTL zabrání tomu, aby tato smyčka měla fatální důsledky. Vyjadřuje totiž **počet routerů**, které smějí paket forwardovat (počet „hopů“).

Řekněme, že náš notebook se chystá odeslat paket na adresu 3.3.3.3 a jeho software nastaví paketu TTL na hodnotu 3 (toto nastavení lze ovlivňovat speciálním způsobem, obvyklý default je dnes 64). Paket dorazí na router 2.0.0.1, který zjistí, že by paket měl forwardovat na router 2.0.0.3, sníží tedy TTL na hodnotu 2, a protože tato hodnota není nula, paket opravdu odešle. Na routeru 2.0.0.3 proběhne podobná procedura a paket se vrátí na router 2.0.0.1 s hodnotou TTL 1. Jenže k novému forwardování paketu už nedojde, router paket **zahodí**, a naopak vyšle zprávu ICMP Time Exceeded původnímu odesílateli.

Poznámka 1: Všimněte si, že ani v „Time Exceeded“ ani v „Time-to-live“ se ve skutečnosti **nejedná o čas**, ale o počet hopů! Název je daný historicky. Naproti tomu u TTL v záznamech DNS protokolu se opravdu o vteřiny jedná.

Poznámka 2: Každý router musí snížit hodnotu TTL a tím **změnit** obsah IP hlavičky, což ho donutí přepočítat **kontrolní součet**. V IPv6 byl proto kontrolní součet hlavičky vypuštěn.

Diagnostika směrování

- Výpis směrovací tabulky: `netstat -r[n]`
příp.: `route print`

Destination	Gateway	Flags	Ipkts	...	Colls	Interface
194.50.16.0	this	U	15943	...	0	tu0
127.0.0.1	loopback	UH		...		lo0
default	gw	UG		...		tu0
193.84.57.0	gate	UGD		...		tu0

- Kontrola cesty: `tracert`, `tracert`

```

1 gw.thisdomain (194.50.16.222)  2 ms  1 ms  1 ms
2 gw.otherdomain (193.84.48.49) 12 ms 15 ms 15 ms
3 * * *
```

Pokud potřebujeme diagnostikovat problémy ve směrování, začínáme obvykle výpisem routovací tabulky. Ten se dá vyvolat příkazem **netstat -r** nebo **route print**. Kromě sloupečků, které už jsme viděli na předchozích slajdech, na výpisu najdeme ještě příznaky (H jako *host route*, G jako *gateway*, čili nepřímý záznam, D jako *dynamický* záznam), jméno síťového rozhraní a různé statistiky (podle nichž mimochodem nese příkaz `netstat` jméno).

Poznámka: Přepínač **-n** brání programu, aby se pokoušel IP adresy překládat na jména. To má dva důvody: jednak nás číselné hodnoty obvykle zajímají víc a jednak se nevystavujeme riziku, že díky nefungující síti se překlad nepovede a my se nedozvíme nic.

Dalším krokem bude nejspíše **ping**, který ale většinou nepomůže. Pokud se totiž odpověď nevrátí, důvody mohou být různé a my je nerozlišíme. Jednak naše pakety nemusely dorazit k cíli. Druhou možností je, že dorazily, ale cíl nefunguje. Rovněž je ale možné, že se ztratily pakety po cestě zpátky. Směrovací tabulky totiž fungují **jednosměrně!** Je to opět stejné jako v našem příkladu s autem – fakt, že podle směrových tabulí dorazíme z Č. Lípy do Č. Budějovic, nic neříká o tom, zda se nám povede dojet také zpátky.

Lepším nástrojem je příkaz **tracert**, který využívá vlastností TTL. Vyšle k cíli paket s nastaveným TTL 1. Důsledkem je, že pokud cíl leží alespoň za jedním směrovačem, tak k němu nedorazí, protože směrovač ho odmítne forwardovat a pošle zpět zprávu ICMP Time Exceeded. Tu `tracert` zachytí a vypíše adresu (a příp. i jméno) směrovače, který ji poslal, včetně round-trip-time. Tento pokus provede program třikrát a poté začne posílat úplně stejné pakety, ale zvýší TTL na 2. Tyto pakety přes první směrovač projdou, ale zastaví je směrovač následující. Tímto

způsobem program postupně odhalí strukturu cesty k cíli. Proces skončí buďto úspěchem, anebo se od určité hodnoty TTL odpovědi přestanou vracet. Poslední směrovač, který nám ještě odpověděl, je potom výchozím bodem pátrání – buďto je chyba přímo na něm, nebo na směrovači následujícím, anebo na přecházejícím (to v případě, že poslední směrovač naše pakety vůbec neměl dostat).

Statické řízení směrovacích tabulek

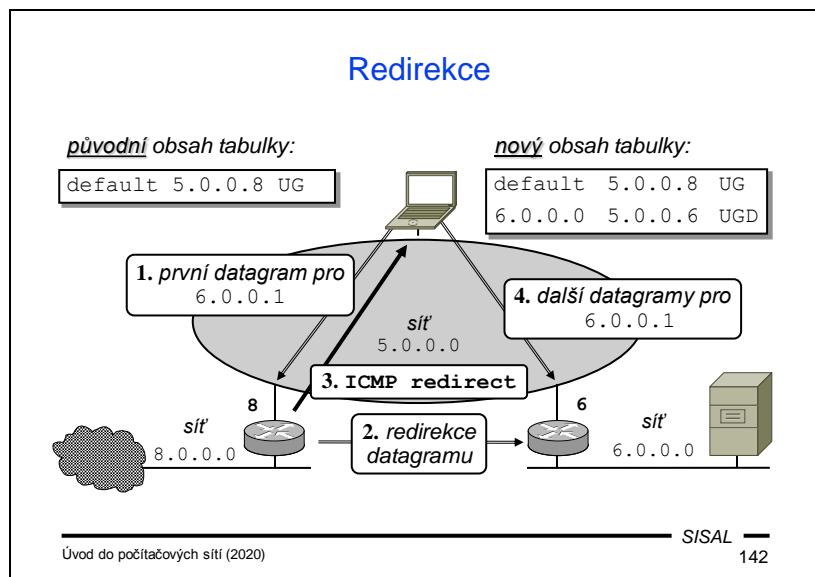
Cesty se nastavují při startu podle konfigurace

- nepružné při změnách
- problémy se subnettingem
- nesnadné zálohování spojení
- + méně citlivé na problémy v síti
- + dostupné i ve zcela heterogenním prostředí
- ⇒ vhodné pro jednodušší, stabilní sítě

```
route {
  add
  delete
  flush | -f
} {
  [[-]host] host
  [[-]net] net [[-netmask] mask]
  default | 0
}
[ gw ] {
  router
  interface [-interface]
} [metric]
```

Nyní už víme, jak se počítač směrovací tabulkou řídí, ale ještě nevíme, jak se správný obsah tabulky obhospodařuje.

Nejjednodušší je **statická metoda** řízení tabulky, kdy počítač má někde uložené všechny záznamy, a ty postupně do tabulky po bootu přidá. Výhodou této metody je její stabilita – pokud se síť nemění a není příliš složitá, zaručuje tato metoda správné fungování za všech okolností a s nejrůznějšími typy uzlů. Je to mimochodem přesně případ, kdy se se svým počítačem připojíme do sítě a on přes DHCP dostane adresu **defaultního routeru** pro lokální síť. Nic víc pro směrování nepotřebuje. Pokud je ale síť příliš rozsáhlá nebo se dynamicky se mění, není toto řešení vhodné.



Jednou z možností, jak i se statickým řízením routovacích tabulek pokrýt složitější síť, je použití dalšího typu ICMP zprávy, **ICMP Redirect**.

V LAN na obrázku vidíme síť 5.0.0.0 s defaultním routerem 5.0.0.8. Tento údaj se také distribuuje stanicím a vidíme ho tedy v routovací tabulce našeho notebooku. Do sítě je ale zapojena ještě další síť 6.0.0.0, o které náš notebook neví. Co se stane, když se pokusí do této sítě poslat paket?

- Podle své routovací tabulky paket pošle routeru 5.0.0.8.
- Ten paket přijme a chystá se ho forwardovat dál. Jenže zjistí, že paket musí poslat zpátky **do stejné sítě**, ze které přišel, jen jinému routeru 5.0.0.6. To ale signalizuje chybu na straně odesílatele, protože ten mohl paket routeru 5.0.0.6 poslat sám. Router 5.0.0.8 tedy paket sice pošle správnému routeru, ale...
- ...odešle původnímu odesílateli zprávu **ICMP Redirect**, ve které mu sděluje, aby si přidal do routovací tabulky nový záznam pro síť 6.0.0.0.
- Pokud náš notebook změnu provede, bude další pakety pro síť už posílat správnému routeru. V routovací tabulce můžeme potom vidět **nový záznam** pro tuto síť, a to s příznakem D (dynamicky vzniklý záznam).

Redirekce sice řeší podobné situace, ale zároveň v sobě skrývá nebezpečí. Pokud bude v lokální síti někdo (záměrně nebo chybou software) šířit vadné ICMP Redirect pakety, může způsobit nefunkčnost sítě. Většinou je proto lepší distribuovat klientům kompletní strukturu sítě a příjem ICMP Redirect zakázat.

Dynamické řízení směrovacích tabulek

Routery si navzájem vyměňují informace o síti pomocí *routovacího protokolu*, stanice se jím mohou řídit také, ale v režimu read-only

- + jednoduché změny konfigurace
 - + síť se dokáže sama „opravovat“
 - + směrovací tabulky se udržují automaticky
 - citlivější na problémy příp. útoky
- na počítači musí běžet program obsluhující protokol
 - př. routed, gated, BIRD (vyvinutý na MFF),...
 - pro lokální síť (*interní routery*) se používají nejčastěji protokoly RIP a OSPF

Řešením pro složitější síť je **dynamický** způsob správy tabulek. Ten je založen na informacích, které si vyměňují sousední routery mezi sebou a na základě nichž si upravují svoje tabulky. Komunikují spolu přitom pomocí některého z **routovacích protokolů**. Obyčejné stanice mohou tyto informace sledovat rovněž, ale pouze v pasivním módu.

Takto řízená síť se dokáže adaptovat sama podle aktuálních podmínek a změny konfigurace lze provádět centrálně na jednom místě. Drobnou daní za to je určité zatížení sítě zprávami routovacího protokolu a větší náchylnost sítě k chybám způsobeným útokem nebo chybami software.

Pokud se chce uzel (router nebo stanice) podílet na komunikaci v routovacím protokolu, musí na něm běžet speciální software. Jeden velmi úspěšných routovacích programů, **BIRD**, který se dnes používá na řadě klíčových routerů ve světě, byl původně vyvíjen jako **studentský projekt na MFF**.

Routovací protokoly pro lokální síť se dělí na distance-vector protokoly (např. RIP) a link-state protokoly (např. OSPF).

Distance vector protokoly

- Základní myšlenka:
 - uzel má u záznamů ve směrovací tabulce i „vzdálenosti“
 - svou tabulku periodicky posílá sousedům, ti si upraví svoji tabulku a v dalším taktu ji posílají dál
- Výhody:
 - jednoduché, snadno implementovatelné
- Nevýhody:
 - pomalá reakce na chyby
 - metrika špatně zohledňuje vlastnosti linek (rychlost, spolehlivost, cenu...)
 - omezený rozsah sítě
 - chyba ve výpočtu jednoho routeru ovlivňuje celou síť (možnost vzniku routovacích smyček)

Základní myšlenkou **distance-vector** protokolů je opět něco, co si snadno můžeme představit na našem příkladu s křižovatkou: na směrových cedulích jsou uvedeny kilometry, takže pokud budeme pravidelně posílat fotografii našeho rozcestníku na sousední křižovatky, mohou si tamější správci snadno ověřit, zda náhodou pro nějaký cíl neexistuje v dané chvíli přes naši křižovátku lepší cesta, než ta, co mají na cedulích nyní, pokud ano, prostě danou ceduli otočí správným směrem a napíší na ni novou vzdálenost.

A úplně stejně to funguje s routery – periodicky posílají svoji tabulku sousedům a ti si překontrolují, zda na jejím základě nemají změnit nějaký záznam ve své routovací tabulce.

Výhodou těchto protokolů je relativní jednoduchost a snadná implementace.

Mezi nevýhody ale patří pomalá reakce na chyby, nedostatečně jemné hodnocení jednotlivých cest (o daném cíli víme pouze počet kilometrů, ale už ne třeba kvalitu cesty) a omezený rozsah sítě. Naprosto zásadním problémem ale je to, že pokud nějaký router udělá **chybu** ve výpočtu, **začne** ji **šířit** a může způsobit nefunkčnost celé sítě.

Routing Information Protocol

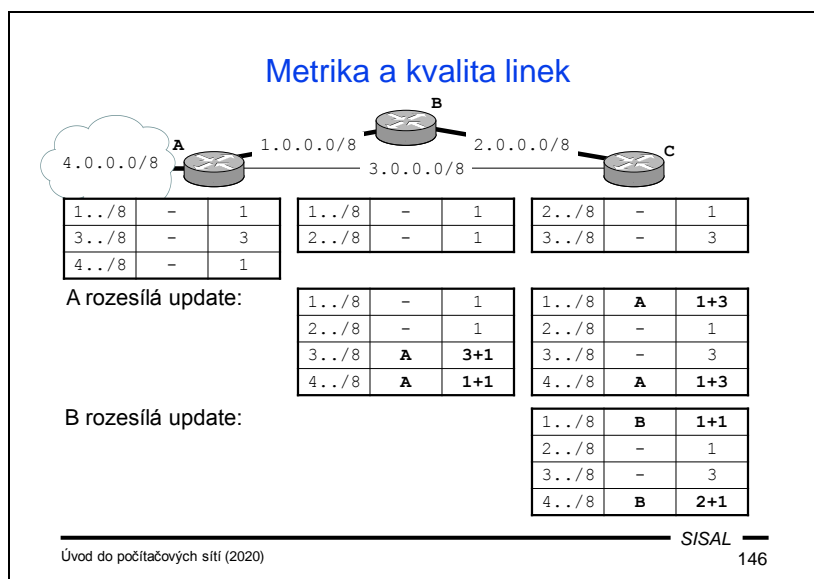
- Nejstarší směrovací protokol, RFC 1058
- Vlastnosti:
 - metrikou je počet routerů v cestě (*hop count*)
 - rozsah sítě je omezen na 15 hopů, 16 je „nekonečno“
 - pro výpočet nejkratších cest používá Bellman-Fordův algoritmus
- Aktuálně verze 2, RFC 2453
 - používá UDP port 520, multicast adresu 224.0.0.9
 - umí subnetting vč. VLSM
 - obsahuje mechanismy na urychlení detekce chyb (triggered updates, split horizon, poison reverse)
- Dostupný na nejrůznějších systémech
- Nepoužitelný pro velké, složité nebo dynamické sítě

Nejznámějším zástupcem distance-vector protokolů je RIP (Routing Information Protocol). Patří mezi velmi staré protokoly, díky čemuž je jeho implementace velmi rozšířená, což je jeho hlavní výhoda.

Zároveň ale doba jeho vzniku ovlivnila některá rozhodnutí v návrhu. Roli **metriky** („vzálenosti“) zde hraje počet routerů (hopů) na cestě k cíli. Tento počet je ale v návrhu omezen pouze na 15, což má nepříjemné důsledky, o nichž ještě budeme hovořit. Tyto důsledky částečně odstraňují rozšíření doplněná ve verzi 2. I přes to není RIP vhodný pro příliš rozsáhlé sítě nebo sítě, jejichž struktura se často mění.

V každém článku o RIPu najdete poznámku, že na výpočet nejkratších cest používá **Bellman-Fordův** algoritmus, ale téměř nikde není zdůvodnění, proč tomu tak je a proč se nepoužívá obecně rychlejší algoritmus Dijkstrův. Tím důvodem je to, že princip Bellman-Fordova algoritmu vlastně „kopíruje“ postupné opravování spočítaných vzdáleností na základě nových informací ze sousedních uzlů. Zjednodušeně tak můžeme říci, že příchod nové zprávy se směrovací tabulkou našeho souseda vyvolá výpočet **jednoho** kroku Bellman-Fordova algoritmu, zatímco u Dijkstrova algoritmu bychom museli opakovat výpočet **celý**.

Zajímavostí je, že verze 1 nepodporovala sítě s proměnlivou maskou (VLSM), protože ve formátu paketů nebyla maska zahrnuta. Verze 2 přitom masku ve formátu již obsahuje, a to přes to, že se pakety nijak neprodloužily a jsou zpětně kompatibilní – prostě se jen využilo místo, které bylo ve formátu verze 1 vynechané.



Počet hopů, bohužel, nereflektuje dobře vlastnosti linek. Je to opět podobné situaci na silnici – máme-li do nějakého města 80 km po silnici a 100 km po dálnici, rychlejší bude asi delší cesta, ale pokud se řidič rozhoduje jen podle počtu kilometrů, pojedje pomalejší cestou. V situaci na obrázku propojuje routery A a C přímá linka (počet hopů je tedy 0), která je ale „pomalá“, anebo „rychlá“ linka, která ale vede přes router B, a tedy má o jeden hop více. RIP by proto preferoval pomalejší linku s metrikou 0. Tomu se dá zabránit tím, že pomalou linku uměle „zatížíme“ větší metrikou, než by odpovídalo počtu hopů. V našem příkladu jsme na linku „přidali dva routery“.

Pojďme se nyní podívat, jak se bude situace v síti měnit, jak si routery budou vyměňovat informace (posílat **update** svých tabulek).

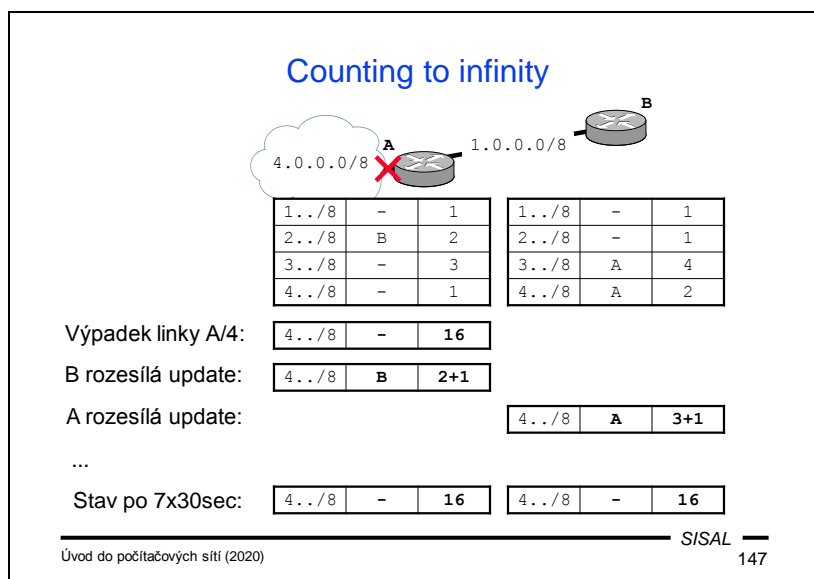
- Na začátku se všechny routery nainicializují hodnotami pro přímo připojené sítě. Ty všechny budou mít metriku 1, s výjimkou linky mezi A a C, která bude mít metriku uměle zvětšenou na hodnotu 3.
- Řekněme, že první pošle svůj update router A. Tabulka dorazí na oba routery B a C a ty si do svých tabulek přidají cesty k novým sítím, které byly zahrnuty v RIP zprávě. Při výpočtu metriky pro nové sítě vezmou metriku uvedenou ve zprávě a připočtou k ní vlastní metriku do sítě, ze které paket přišel. Router B tedy bude k hodnotám přidávat jedničku, takže si do své tabulky zařadí síť 3.0.0.0 s metrikou 3+1 a síť 4.0.0.0 s metrikou 1+1. Router C má metriku do sítě 3.0.0.0, ze které zpráva přišla, nastavenou na 3 a bude tedy přidávat obě nové sítě 1.0.0.0 i 4.0.0.0 s metrikou 1+3. Vlastní přímo připojené sítě obou routerů mají metriku lepší než tu, co přišla, takže ty se měnit nebudou.
- Jako další pošle update tabulek router B. Na router C dorazí jeho paket po síti, která má metriku 1, takže k hodnotám ve zprávě bude přidávat jedničku. Síť 1.0.0.0 měla hlášenou metriku 1, po přičtení jedničky dostaneme hodnotu 2, což je lepší než stávající hodnota 4. Router C si tedy opraví záznam v tabulce, změní

směr na router B a metriku na 2. Podobně dopadne kontrola sítě 4.0.0.0, nová metrika 2+1 je opět lepší než stávající 4 a záznam se opraví. Přímo připojené sítě nadále zůstávají nezměněny. Tato podoba tabulky na routeru C už je konečná, pokud nedojde v síti k nějaké změně.

- Celý proces zakončí update poslaný routerem C, díky němuž si své tabulky upraví do konečné podoby i zbývající routery.

Vzhledem k tomu, že routery posílají první zprávy s obsahem tabulek hned po bootu, ustálí se tato síť velmi rychle.

Problém různé kvality linek jsme zde vyřešili elegantně, ale pouze proto, že tato síť je velmi malá a linky mají pouze dvě různé „rychlosti“ (šířky pásma). Jakmile bychom měli síť složitější, velmi rychle narazíme na strop maximální metriky 15. Logickým řešením by bylo zásadně zvednout tuto hodnotu.



Proč je „nekonečno“ u RIPu tak malé? Důvodem je snaha minimalizovat dopad problému nazývaného **counting to infinity** (počítání do nekonečna).

Představme si, že v naší síti dojde k výpadku připojení do sítě 4.0.0.0 na routeru A. Router na to zareguje nastavením metriky této sítě na nekonečno (16). Než mu ale vyprší perioda, kdy bude rozesílat svoje tabulky a distribuovat tak informaci o nedostupnosti sítě 4.0.0.0, dorazí mu update paket od routeru B, který ve svých tabulkách síť stále ještě má s metrikou 2. Router A provede výpočet a zjistí, že „nová“ cesta do sítě 4.0.0.0 přes router B má metriku $2+1 < 16$, a tak si ji v tabulce opraví. Tím zmizí informace o nedostupnosti sítě.

Jakmile perioda pro update routeru A vyprší a on pošle svoji tabulku všem, k routeru B se dostane informace o tom, že síť 4.0.0.0 je nyní dostupná přes router A, ale s metrikou 3. Jelikož stávající záznam na routeru B pochází také od routeru A, musí router B respektovat změnu metriky na routeru A a záznam o síti 4.0.0.0 si musí opravit na novou metriku $3+1$.

S další výměnou update paketů se postupně metrika na obou routerech zvyšuje, až se konečně vrátí na správnou hodnotu „nekonečno“. Vzhledem k délce periody (30 sec) se přes 3 minuty síť 4.0.0.0 tváří jako živá, ačkoliv reálně dostupná není.

Kdyby „nekonečno“ bylo řádově delší, byl by řádově delší i výpadek.

Tento problém zároveň ilustruje jeden důležitý princip při návrhu algoritmů: vždy je třeba vyšetřit a předejít možným **race conditions**, neboli situacím, kdy dva nezávislé jevy mohou nastat v nevhodném pořadí.

Negativní dopady počítání do nekonečna minimalizují rozšíření RIPv2:

- *Triggered updates* je mechanismus, kdy router při detekci problému odešle svůj **update okamžitě** a nečeká na periodu. Tím se sníží riziko race condition, že update od partnera přijde dříve. Ale pozor, toto riziko se mechanismem jen **sníží**, nikoliv zcela **odstraní**!
- Mechanismus *split horizon* spočívá v tom, že router neposílá partnerovi informace o sítích, o nichž se dozvěděl **právě od něj** (to nedává smysl – partner má o daných sítích lepší informace než my). Tento mechanismus race condition popsané výše efektivně **brání**. Drobnou daní za to je, že router nemůže stejnou update zprávu posílat všem sousedům, ale musí každou připravit zvlášť.
- *Poison reverse* je mírné vylepšení metody split horizon – router sousedovi data o „jeho“ sítích posílá, ale s metrikou 16, záznam tím „otráví“.

Link state protokoly

- Základní myšlenka:
 - každý router zná „mapu“ celé sítě
 - routery si navzájem sdělují stav svých linek a podle toho si každý modifikuje svoji mapu sítě
- Nevýhody:
 - výpočet mapy je náročnější na výkon CPU i na paměť
 - při startu a na nestabilních sítích může výměna dat znamenat významnou zátěž sítě
- Výhody:
 - pružná reakce na změny topologie
 - každý si počítá sám za sebe, chyba neovlivní ostatní
 - síť je možné rozdělit na menší podsítě (rychlost výpočtu!)
 - výměna dat probíhá pouze při změnách

Druhá skupina routovacích protokolů se nazývá **link-state protokoly**. Název vychází ze základního principu protokolů, že místo tabulky se vzdálenostmi se posílají pouze informace o stavech linek. Každý router si drží sám celou **mapu sítě** a podle hlášení o stavu linek si sám přepočítává optimální cesty. Pokud bychom použili znova analogii s řidičem, tak v tomto případě má řidič k dispozici nikoliv směrové cedule, ale rovnou autoatlas. Pokud se z rádia dozví o problému, dokáže si sám v atlasu najít nejlepší alternativu.

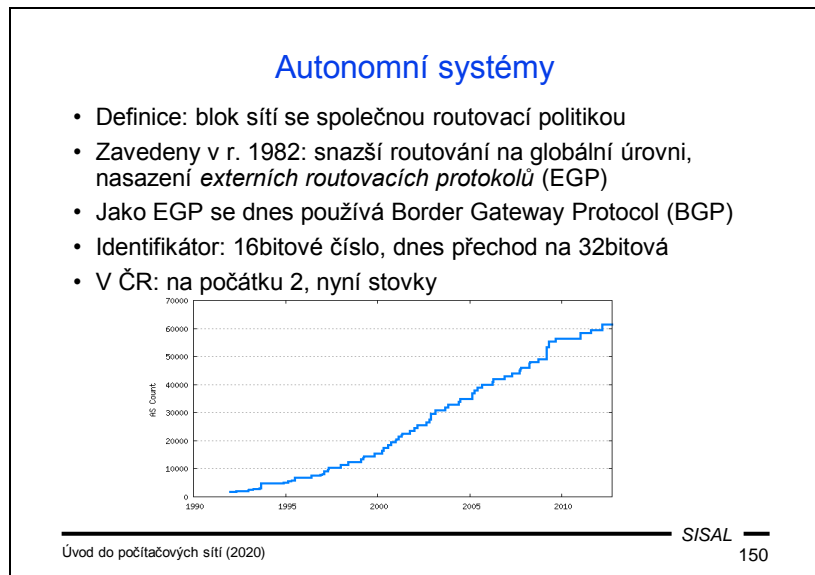
Tento přístup je sice výpočetně náročnější a v případě rychle se měnících podmínek znamenají okamžité výměny dat určitou síťovou zátěž, ale výhody převažují. Klíčové jsou dvě: síť reaguje daleko **pružněji** na změny nebo výpadky a každý router si mapu počítá sám, takže případné chyby **neovlivní** nikoho ze sousedů. Pro stabilní síť naopak představuje výhodu to, že si routery nemusejí pravidelně vyměňovat dvě databáze.

Open Shortest Path First

- Nejrozšířenější link-state interní routovací protokol
- Vlastnosti:
 - používá Dijkstrův algoritmus nalezení nejkratší cesty
 - používá hierarchický model sítě:
 - oblast (area) 0 tvoří páteř
 - ostatní oblasti se připojují pouze na páteř
 - každý router zná mapu své oblasti a cestu k páteři
 - metriku je možné konfigurovat, implicitně je to *path cost*, součet „cen“ na cestě, kde cena je dána šířkou pásma
- Používá samostatný protokol transportní vrstvy 89 a multicast adresy 224.0.0.5 a 224.0.0.6
- Aktuální je verze 2 pro IPv4 (RFC 2328) a revize pro IPv6 označovaná jako verze 3 (RFC 5340)

Nejznámějším představitelem link-state protokolů je OSPF (Open Shortest Path First). Tady se už na rozdíl od RIPu opravdu používá pro výpočet mapy sítě **Dijkstrův** algoritmus. Nicméně u velkých sítí by i ten měl s rostoucí velikostí sítě problémy, a proto je možné OSPF síť rozdělit na **oblasti**, takže výpočet probíhá vždy jen v rámci oblasti, a tedy na podstatně menší množině uzlů. Síť je uspořádána dvoustupňově: oblast s číslem 0 se nazývá **páteř** (backbone) a všechny další oblasti jsou připojené přímo na páteř. Jakákoliv cesta v síti se tedy skládá nejvýše ze tří částí: ze zdrojové oblasti na páteř, po páteři k přípojnému bodu cílové oblasti a po cílové oblasti.

Metrika se v OSPF nazývá *path cost* (tedy „cena cesty“) a určuje se pomocí složitého vzorce, který si konfiguračně definuje administrátor sítě a který tak dokáže ve správné míře zahrnout nejen „vzdálenost“, ale také šířku pásma, latenci, propustnost, ale třeba také skutečnou „cenu“ provozu po nějaké lince.



Až doposud jsme si povídali o protokolech určených pro směrování v „relativně blízkých“ sítích. Jak se ale situace změní, když se podíváme na internetovou páteř?

Jednotlivé bloky sítí tvoří tzv. **autonomní systémy** (AS). Přesná definice není příliš nosná, v podstatě se v ní hovoří jen o tom, že sítě musejí mít společnou routovací politiku, takže není až tak složité o přiřazení AS požádat a tomu zase odpovídá růst počtu AS. Při vstupu České republiky do internetu zde byly dva autonomní systémy a nyní se počítají na stovky. Vlastní AS mají typicky ISP nebo velké společnosti. V roce 2009 bylo nutné přejít z původně 16bitových čísel AS na 32bitová.

Smyslem AS je sjednotit směrování pro celou skupinu sítí na globální úrovni. Routování mezi jednotlivými AS poté probíhá na základě poněkud odlišných podmínek, než to, které probíhá uvnitř AS. Hlavní roli zde nehrají čísla sítí, ale čísla AS. Protokoly, které jsme dosud studovali, se nazývají *interní routovací protokoly* (IGP), pro řízení routování mezi AS se používají tzv. *externí routovací protokoly* (EGP). Nejznámějším zástupcem EGP je **Border Gateway Protocol** (BGP). Kritickými vlastnostmi EGP je zahrnutí dalších faktorů při hodnocení cest (podobně jako jsme hovořili o výpočtu „path cost“ u OSPF) a zabránění vzniku smyček. Protokoly pracují s tzv. **path-vector**, posloupností čísel AS, přes něž cesta vede, a tím vzniku smyček brání.

IP filtrování

- Router na perimetru (intranet/internet) má v konfiguraci uvedeno, jaký provoz je povolen a za jakých podmínek
- Přísná konfigurace: ven vybrané, dovnitř nic
 - dobré pro protokoly s jedním kanálem (HTTP, SMTP)
 - problém u protokolů s více kanály (FTP, SIP)
- Obvyklá konfigurace: ven cokoliv, dovnitř nic
 - naráží např. u FTP s aktivním přenosem
 - nepoužitelné u protokolů s mnoha kanály (SIP)
- Lépe se dá řešit nastavením aplikací a SW na routeru, který musí částečně rozumět aplikační vrstvě
- Problém se službami „uvnitř“ (např. www server, pošta)
 - povolení výjimek je riskantní
 - lepší je oddělený segment, DMZ, demilitarizovaná zóna

Směrovač, který připojuje lokální síť do internetu, je bodem, kde se uplatňuje **bezpečnostní politika** celé sítě. Jejím základem je obvykle **IP filtrování**. Název je poněkud zavádějící, protože se ve skutečnosti nejedná o filtrování na IP vrstvě, ale na vrstvě **transportní**. Stanovuje pravidla, jaký typ provozu (přesněji provoz na „kterých portech“) je dovozen z a do lokální sítě.

Poznámka: Otázkou je, co vlastně znamená „z“ a „do“. Třeba v případě TCP samozřejmě musí pakety chodit oběma směry. Podstatné je ale, kdo spojení **navazoval** (a posílal tedy SYN paket).

Nejstriktnější konfigurace povoluje pouze provoz z vnitřní sítě do internetu, a to pouze na vybrané porty. Tato konfigurace je vyhovující pouze pro jednobanové protokoly jako jsou HTTP nebo SMTP. Jakmile protokol potřebuje pro svůj provoz dodatečné kanály, jejich otevření filtr zabrání. Jedinou možností v tomto případě je použít filtr, který rozumí i patřičnému aplikačnímu protokolu. Např. pokud filtrovací software ve FTP zachytí komunikaci, kterou se server a klient domlouvají na otevření datového kanálu, může na omezenou dobu několika sekund ve filtru „otevřít díru“ pro dvě konkrétní adresy uvnitř a vně sítě, aby se datové spojení mohlo navázat.

Obvyklejší konfigurace není tak omezující na seznam portů, na něž smějí lokální klienti přistupovat. V tom případě už např. s pasivním otevíráním datových kanálů ve FTP nebude problém. Ten zůstane pouze s aktivními kanály anebo tam, kde je kanálů třeba otevírat více (např. u SIPu). Zde už ze bez spolupráce s aplikační vrstvou neobejdeme.

Dalším problémem pro filtrování představují služby, které chce síť poskytovat do veřejného internetu. V minulosti to byl typický problém např. pro provoz www serveru.

Dnes se www servery obvykle řeší pomocí různých **hostingových** služeb, nicméně pokud chceme web nebo jinou službu provozovat ve vlastní síti, budeme muset ve filtru otevřít permanentní díru povolující provozu z vnější sítě přístup na konkrétní server a port. To samozřejmě představuje riziko, protože jakmile se potenciální útočník dostane do vnitřní sítě, může využít jakékoliv chyby v software nebo konfiguraci serveru. Proto bývá zvykem pro podobné služby vyhradit zvláštní segment sítě, tzv. *demilitarizovanou zónu* (DMZ). Na ni pak z hlediska ochrany sítě pohlížíme jako na svět, který není ani zcela bezpečný, ani zcela nebezpečný a filtrovací pravidla do vnitřní sítě i do internetu jsou o něco benevolentnější než ta na rozhraní vnitřní sítě a internetu.

Proxy server

- Transparentní varianta:
 - SW na **routeru** zachytí spojení, uloží požadavek, naváže „svým jménem“ spojení na server a požadavek odešle.
 - Odpověď přijde zpět na router, ten ji uloží (pro další klienty) a zároveň odešle původnímu žadateli.
 - Není třeba konfigurovat na klientovi.
- Netransparentní varianta:
 - Klienty je třeba **nakonfigurovat**, aby se požadavky neposílaly přímo, ale proxy-serveru v lokální síti (lze i automaticky po síti).
 - Proxy server nemusí být nutně router.
 - Je nutná podpora v protokolu.
- Významný bezpečnostní a výkonnostní prvek:
 - umožňuje správě sítě efektivně kontrolovat činnost klientů
 - umožňuje omezit objem provozu na přípojně lince

Software, který kontroluje provoz určitého protokolu (obvykle ne rozhraní lokální sítě a internetu), se nazývá **proxy server**. Provozuje se buďto transparentně nebo netransparentně.

Transparentní varianta pracuje na hraničním routeru sítě. Router zachytí klientův požadavek a předá ho proxy serveru. Ten ho zkontroluje podle pravidel bezpečnostní politiky organizace, a je-li v pořádku, naváže jako klient spojení na skutečný server a požadavek mu odešle. Jakmile dorazí odpověď, je opět zkontrolována (např. antivirovým programem) a odeslána klientovi. Výhodou transparentní proxy je, že na klientech není třeba nijak měnit konfiguraci – požadavek se k proxy dostane, aniž o tom klient musí vědět.

O existenci *netransparentní* proxy se klient musí dozvědět, protože sám musí požadavek poslat proxy serveru a nikoliv cílovému serveru. Nevýhodou tohoto řešení je tedy to, že klienta je třeba patřičně nakonfigurovat (ručně nebo automaticky). Výhodou je to, že proxy server může běžet na vhodnějším hardware resp. operačním systému, než je router. Řešení je ale dostupné pouze u těch protokolů, které pro to mají podporu – klient musí už při odesílání požadavku specifikovat, že se s ním obrací na proxy a ne na server.

Používání proxy serverů má kromě technických důvodů také důvody bezpečnostní (proxy server může filtrovat operace na úrovni aplikačního protokolu) a výkonnostní (stejný požadavek nemusí proxy server opakovaně posílat k vyřešení skutečnému serveru, odpověď může uložit do **cache** a dalším zájemcům ji poslat sám). Poslední bod ovšem třeba u HTTP postupně ztrácí na významu, protože při používání HTTPS kešování využít nelze.

Souhrn 7

- Jak funguje směrovací algoritmus?
- Popište typy sloupců a záznamů směrovací tabulky.
- Čím se liší statické a dynamické řízení správy směrovacích tabulek?
- K čemu slouží routovací protokoly?
- Porovnejte distance-vector a link-state protokoly.
- Co je autonomní systém?
- Vysvětlete význam zpráv ICMP Echo, Time Exceeded, Destination Unreachable.
- K čemu slouží pole TTL v IP záhlaví?
- Co označuje pojem IP filtrování?
- K čemu slouží proxy server?