Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Dynamické směrování v overlay síti

Plzeň 2016 Lukáš Kvídera

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 11. května 2016

Lukáš Kvídera

Abstract

The main goal of this master thesis is implementation of an overlay network. Theoretical part discusses existing routing algorithms and overlay networks. Practical part si focused on measurement of prediction's impact to Quality of Service.

Abstrakt

Hlavním cílem této diplomové práce je návrh a implementace překryvné sítě využívající predikci stavu linek. V teoretické části jsou podrobněji popsány stávající směrovací algoritmy a implementace překryvných sítí. Praktická část je zaměřena na měření vlivu predikce na kvalitu spojení.

Obsah

1 Úvod

V současné době všudypřítomného internetového připojení vznikají služby, které byly dříve nemyslitelné. Internet poskytuje globální přístup k videím, hudbě, umožňuje uskutečnění mezikontinentálních hovorů i videohovorů. Stále častěji se ovšem naráží na fyzikální problémy, které znesnadňují či znepříjemňují požitek z těchto služeb.

Například se pomalu stává standardem video v rozlišení 4K, jehož datové nároky jsou znatelně vyšší než u předchůdce 1080p. Důsledkem jsou mnohdy zahlcené linky v lepším případě, nebo nespokojený zákazník sledující trhaný obraz v tom horším. Směrovače sice mohou do jisté míry uplatňovat klasifikaci provozu a dle požadavků zajišťovat kvalitu služeb, ale jen v omezené míře, kvůli rozlehlosti internetu.

Dalším příkladem může být telefonní hovor, vedený sice dle směrovače po nejkratší cestě, ale velmi ztrátové z důvodu vadného optického kabelu. Spojení je z pohledu směrovače funkční a tak ho využívá, dokud může, což nemusí nutně znamenat optimální kvalitu pro koncového uživatele.

K řešení těchto a jiných se běžně využívají překryvné sítě. Některé mohou využívat rozsáhlosti internetu, jiné naopak mohou tvořit abstraktní struktury, které jeho rozlehlost naopak redukují.

V překryvných sítích zaměřujících se na kvalitu služeb je obvykle prováděno směrování, které může být dvojího druhu: proaktivní a reaktivní. Reaktivní směrování rozhoduje o směru přeposlání na základě nedávného naměřeného stavu. Oproti tomu proaktivní odhaduje, jaký stav bude v následujících časových okamžicích a v ideální případě poskytuje lepší cesty.

Teoretická část diplomové práce je věnována směrovacím algoritmům a jejich metrikám, které jsou podstatné pro určení nejvýhodnější cesty. Dále budou popsány způsoby měření metrik a jejich uchování v podobě časových řad pro pozdější využití predikčními algoritmy. Následně budou rozebrány odlišnosti směrování v překryvné síti oproti klasické síti a komplikace s tím spojené.

V praktické části je uvedena analýza architektury aplikace, komunikační protokoly a implementované algoritmy směrování. V závěru budou dosažené výsledky porovnány na jedné překryvné síti s reaktivním i proaktivním směrováním.

2 Překryvná síť

Překryvnou síť si lze představit jako síť v síti. Na rozdíl od klasického chápání sítě, kdy přepínání rámců v působnosti linkové vrstvy broadcastové domény obstarávají přepínače a směrování mezi jednotlivými sítěmi směrovače na síťové vrstvě, se jedná o směrování na vyšších vrstvách, například na aplikační vrstvě.

Překryvné sítě se budují vždy s nějakým jasně definovaným cílem, jenž většinou odstraňuje nebo potlačuje nedostatky na nižších vrstvách, či přináší naprosto novou funkcionalitu v podobě poskytování služeb. Službou může být míněna anonymizace uživatelů nebo i serverů. Překryvných sítí zabývajících se anonymizací uživatelů existuje několik. Některé jsou již na ústupu – *TOR*, kvůli známým zranitelnostem [4], jiné naopak mohou být na vzestupu – *I2P*

Matematickým pohledem lze překryvnou síť vnímat jako podgraf grafu podkladové sítě, kde uzly překryvné sítě jsou reprezentovány vrcholy a spoje mezi nimi hranami. Díky této vlastnosti lze aplikovat grafové algoritmy na hledání nejkratší vzdálenosti, viz sekce 3.2.

Obrázek 2.1 zobrazuje překryvnou síť v modrém obláčku, využívající fyzickou topologii pod obláčkem.

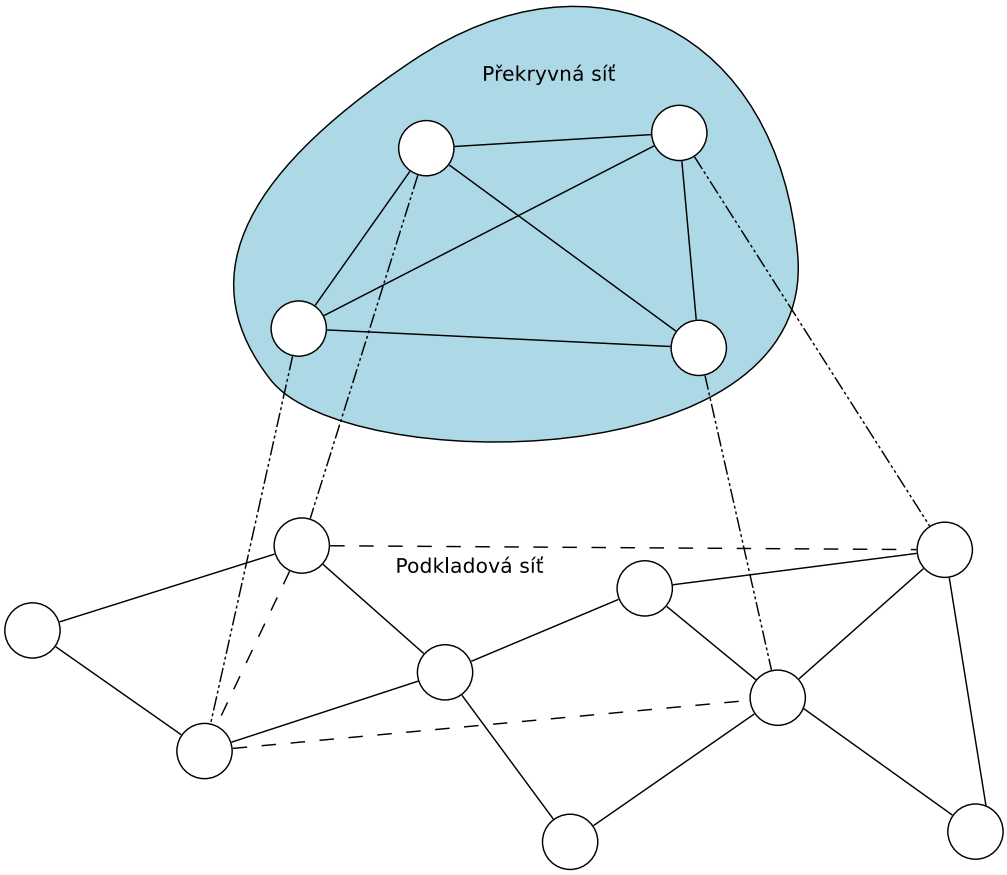
2.1 Architektura překryvných sítí

Překryvné sítě sestávají z množiny uzlů komunikujících všechny mezi sebou navzájem, nebo dle pravidel stanovených implementátorem překryvné sítě.

Překryvné sítě určené ke sdílení souborů lze rozdělit na strukturované a nestrukturované. Strukturované sítě využívají definice své struktury a požadavky směrují dle svých strukturních možností. Většinou jde o rozdělení prostoru hashů klientů a hashů souborů na menší části. Prostor může nabývat tvaru čtverce, krychle nebo jiných N-rozměrných útvarů. Určení místa, kde je soubor uložen, je pak dáno nejmenší Hammingovo vzdáleností hashe klientské adresy a nejbližšího hashe v prostoru.

Strukturovanost sítí navíc shora omezuje vyhledávací složitost do polynomiálních mezí. Nemůže se tedy stát, že bude vyhledávání trvat exponencielně dlouhou dobu vůči počtu klientů do sítě připojených.

Nestrukturované sítě využívají dotazovacích metod a záplavového šíření dotazu. Na dotaz je pak odpovězeno zpět po cestě, kudy byl dotaz šířen. [31]



Obrázek 2.1: Překryvná síť nad fyzickou sítí

2.2 Zástupci překryvných sítí

Účely kvůli nimž vznikají překryvné sítě se mohou velmi různit. Některé sítě si kladou za cíl zvýšení anonymity uživatelů náhodným směrováním přes několik uzlů. Jiné jsou tvořeny za účelem zvýšení dostupnosti sdílených souborů a další slouží ke zvýšení kvality služeb hledáním optimálních cest v Internetu.

V následujících odstavcích budou zmíněny některé význačné překryvné sítě, které mají své účely existence významně odlišné.

2.2.1 Tor

Nejznámějším představitelem překryvné sítě by mohl být *TOR*. Ten funguje na sedmé vrstvě a jeho hlavním cílem je pokud možno zachování anonymity uživatele. Tohoto cíle je dosaženo tím, že uživatel chtějící komunikovat s nějakým cílovým serverem s ním nekomunikuje přímo, ale je nějakým způsobem připojen do *TOR* sítě.

V této síti si zvolí náhodně několik uzlů a zprávu, jež má být poslána vybranému serveru zašifruje klíči uzlů, kterými zpráva bude procházet. Uzel, jímž daná zpráva prochází, zprávu rozšifruje a buď ji dle pokynu pošle dalšímu uzlu, či přímo serveru. Tomuto způsobu směrování se říká cibulové, právě kvůli tomu, že je zpráva zaobalena do několika šifrovaných vrstev jako cibule. Způsob zaobalování zpráv dal i název celé síti – The Onion Routing.

Hlavní podstatou tohoto přístupu je, že transportní uzly nemohou zjistit, kdo je iniciátorem komunikace, ani co je obsahem posílaných zpráv[1]. [19]

Jednotlivé uzly *TOR* sítě mohou sloužit buďto jako koncový uzel – výstupní uzel, nebo jako transportní uzel, jenž pouze přeposílá pakety dalšímu uzlu. [22]

V síti *TOR* je predikování síťového provozu naopak nežádoucí kvůli tomu, že by mohla být zpětně prolomena anonymita koncového uživatele.

2.2.2 Bittorrent

Bittorrent patří mezi nestrukturované překryvné sítě orientované na dostupnost sdílených souborů. Základem je centrální prvek – tracker, který má přehled o tom, na kterých uzlech leží jaké soubory. Trackerů může být více, pro zajištění vyšší spolehlivosti, nicméně kvůli snížení zátěže těchto serverů a zvýšení decentralizovanosti se používají další technologie, mezi něž patří například *DHT*. *DHT* se sdílí již přímo mezi klienty, kteří kladou distribuovaně dotazy na požadované soubory. [20]

Zde není příliš prostoru pro využití predikčních algoritmů, což ovšem neplatí pro překryvné sítě zaměřené na spolehlivost přenosu. Této síti je věnována následující sekce.

[1] Neplatí pro koncové uzly, kde mohou být nainstalovány sniffovací nástroje pro zachytávání případné nešifrované komunikace mezi koncovým uzlem a cílovým serverem.

2.2.3 Resilient overlay network

Resilient overlay network – *RON*, si klade za cíl důslednější prosazování QoS (kapitola 3.3) napříč internetem. V dnešní době se stává stále obvyklejším jevem cílené a úmyslné zpomalování, či degradace linek [27], za účelem získání více peněz za nakoupenou konektivitu. [8]

Prostředkem pro zlepšení kvality přenosu je množina uzlů, které si navzájem udržují sousedské vztahy a provádějí aktivní či pasivní měření kvality linek mezi sebou. Na základě aktuálních naměřených hodnot a krátké historie uložené v Round Robin databázi vyhodnocují nejlepší cestu pro paket, který je pak směrován uvnitř překryvné sítě. [1]

Tato práce se zaměřuje na zjištění vlivu využití predikčních algoritmů na kvalitu takto vybrané cesty a jak výběr takové cesty bude ovlivněn. Způsoby měření a uchování naměřených dat jsou detailně popsány v kapitolách 4.2 a 4.3.

3 Směrování

Internet jako takový je rozlehlý a musí být nějakým způsobem definováno, jak doručit data z bodu A do bodu B. V některých případech může být i řečeno, jakou kvalitu služby přenášená data vyžadují. Směrovací algoritmy pak dle svých možností mohou vybírat různé cesty pro real-time data a jiné pro přenos vyžadující velkou šířku pásma.

Historicky se směrovací algoritmy vyvíjely od jednoduchých, které nepodporovaly sítě s různou délkou masky a nekonvergovaly příliš rychle, až po složitější využívající grafové algoritmy a aktivní oznámení o výpadku linky. Hlavní myšlenky směrovacích algoritmů budou uvedeny v následujících odstavcích, ze kterých čerpá i praktická implementace této práce.

3.1 Algoritmy směrování

K sestavení směrovací tabulky je nezbytné mít sadu pravidel a protokolů, jež zajistí výběr nejlepší cesty, případně vyvažování zátěže mezi cestami rovnocennými. Výběr nejlepší cesty je možné provést různými algoritmy, z nichž některé kombinují více naměřených souborů dat s určitými vahami.

Směrovací algoritmy se mohou členit na dva základní typy: statické a dynamické. Statické směrování spočívá v konfiguraci cest administrátorem, čímž není zaručena spolehlivost v případě výpadku cesty, proto se spíše používají složitější dynamické směrovací algoritmy. [26]

Dynamické směrovací algoritmy se dokáží rychleji zotavit z výpadků linek a zároveň mohou využívat stavových parametrů linek (dostupnost, spolehlivost, latence, propustnost).

3.1.1 Vektor vzdálenosti

Směrovací algoritmy typu vektor vzdálenosti zajišťují výběr nejvýhodnější cesty na základě směru a vzdálenosti. Vzdáleností je chápána metrika, což může být počet skoků [26]. Jedním z prvních dynamických směrovacích algoritmů je *RIP* – Routing Information Protocol, který ve své první verzi využíval třídní dělení IP adres, a proto zprávy posílané mezi routery neobsahovaly síťové masky[3].

S přibývající potřebou dělit sítě na menší byla vyvinuta druhá verze protokolu *RIP*, která již podporovala proměnné masky sítí. Nevýhodou tohoto

protokolu je pomalá konvergence kvůli periodickým zprávám o změně stavu přilehlých sítí. [11] [21]

Ačkoliv jsou zprávy zasílány periodicky, může nastat taková situace, kdy vypadne linka těsně po odeslání poslední aktualizace. Po celou dobu periody mají pak sousední směrovače neplatnou informaci a to vede k zahazování paketů a zhoršení kvality spojení.

Částečným řešením je vynucená aktualizace mimo danou periodu. Tím ale vzniká prostor pro tvorbu směrovacích smyček, kdy jeden směrovač je přesvědčen, že cesta vede přes směrovač druhý a druhý naopak tvrdí, že cesta vede přes první. Kvůli tomuto vzniklo vylepšení „rozdělený horizont“.

V rozděleném horizontu si každý směrovač pamatuje, od koho dostal danou směrovací informaci. Tu pak propaguje na všechna rozhraní kromě toho, ze kterého tato informace pochází.

3.1.2 Stav linky

Algoritmy stavu linky nezohledňují jen počet skoků, ale započítávají i aktuální stav linky[1]. Takto například funguje protokol *Open Shortest Path First – OSPF*. Cena rozhraní u protokolu *OSPF* je pak spočítána dle vzorce $cost = \frac{referencebandwidth}{interfacebandwidth}$ a použita ve výpočtu nejkratší cesty v grafu (algoritmy výpočtu jsou uvedeny v sekci 3.2). [9] [2] [24]

Nicméně nemusí se jednat jen o nastavenou rychlost rozhraní, ale i další metriky. Například zpoždění na lince mezi směrovači, aktuální vytížení linky, či ztrátovost paketů.

OSPF směrovaná doména sestává z backbone sítě označené jako *Area 0*. K této síti mohou být navázány další podsítě s různým číslem oblasti. Směrovače ležící ve více oblastech najednou jsou názývány „hraniční“ směrovače, jelikož leží na hranicích oblastí.

Každý směrovač využívající protokol *OSPF* neudržuje topologii celé sítě, nýbrž si udržuje kompletní topologii své oblasti. Hlavním důvodem je redukce výpočetní složitosti nad rozsáhlými sítěmi členěním do oblastí. Změny stavu linek jsou pak aktivně propagovány v rámci dané oblasti.

[1] Rychlost jež je nakonfigurována v nastavení portu

3.1.3 Hybridní algoritmy

Proprietární protokol *Extended Interior Gateway Routing Protocol* (*EIGRP*) je hybridním směrovacím protokolem, který využívá principů vektoru vzdálenosti a zároveň výhod rychlé konvergence algoritmů využívajících stavu linky. Metrika *EIGRP* je počítána dle složitého vzroce 3.1.

$$metric = 256 \cdot (K_1 \cdot BW + \frac{K_2 \cdot BW}{256 - LOAD} + K_3 \cdot DELAY \cdot \frac{K_5}{REL + K_4}) \quad (3.1)$$

Proměnná $DELAY$ je součtem latencí všech odchozích rozhraní. Zatížení ($LOAD$) a spolehlivost (REL) jsou procenta vyjádřená hodnotou od 1 do 255. BW je proměnná obsahující propustnost rozhraní.

Hodnoty ale nejsou měřeny pravidelně, jen při opětovném zapnutí vypnuté linky. Koeficienty K_i jsou určeny konfigurací administrátorem dle svých pořeb. Nastavením libovolného koeficientu na nulu je možno vypustit konkrétní parametr metriky. [5]

Směrovače s *EIGRP* používají periodických zpráv pro udržení sousedství. Není-li obdržena zpráva do vypršení časového limitu, je linka označena jako neaktivní, ale tato informace není dále propagována. Až po uplynutí druhého časového limitu a neaktivitě linky je rozšířena sousedním směrovačům informace o změně stavu linky.

3.2 Hledání nejkratší cesty v grafu

Hledání nejkratší cesty v grafu je často využívaný grafový algoritmus. Jeho cílem je zjistit, přes jaké vrcholy se lze dostat z počátečního bodu do bodu koncového co nejrychleji. Podstatná je volba reprezentace grafu v programu a tím i implicitně definovaná efektivita výpočtu na CPU.

V případě hustých grafů sítě je výhodnější využít matice sousednosti, které budou obsahovat nekonečný prvek v distanční matici jen zřídkakdy. Nekonečno v distanční matici znamená, že do vrcholu j nevede hrana z vrcholu i a je tak nedostupný.

Řídké grafy by naopak obsahovaly mnoho nevyužitelných prvků v matici sousednosti a výpočet by pak počítal vzdálenosti vrcholů přes neexistující hrany zbytečně, proto je pro řídké grafy vhodnější spíše spojový seznam či stromová datová struktura, která efektivněji využívá paměť a výkon počítače. [35]

3.2.1 Dijkstrův algoritmus

Dijkstrův algoritmus funguje na principu prohledávání do šířky s prioritní frontou. Prioritní fronta obsahuje vrcholy seřazené vzestupně podle vzdálenosti od výchozího bodu. Tím je zajištěno, že každý vrchol vyjmutý z fronty je nejblíže ze všech ke zvolenému počátku prohledávání.

Při vyjmutí je vrchol označen jako již zpracovaný a dále se při svém znovuobjevení do fronty nevkládá. Zároveň je aktualizována vzdálenost mezi tímto vyjmutým vrcholem a jeho přímými sousedy dle vztahu 3.2, kde x je počáteční vrchol, z vrchol vyjmutý z prioritní fronty a y vrchol sousední s vyjmutým vrcholem. Vzdálenost $d(x, y)$ může být nějaké číslo, pokud již byla objevena jiná cesta vedoucí do daného vrcholu, ale vrchol samotný ještě nebyl zpracován. V tomto případě je vzdálenost aktualizována na nižší hodnotu z $d(x, y)$ a $d(x, z)+d(z, y)$ a fronta nezpracovaných vrcholů je znovu seřazena. [16]

Hledáme-li nejkratší cestu mezi dvěma vrcholy, pak prohledávání může skončit ihned, jakmile je z prioritní fronty vyjmut cílový vrchol. Průběh hledání cesty je ilustrován na následujících obrázcích. Obrázek 3.1 zachycuje počáteční stav grafu s ohodnocenými hranami.

$$d(x, y) = min(d(x, y), d(x, z) + d(z, y)) \tag{3.2}$$

Na obrázku 3.2 byl vyjmut počáteční vrchol (označen červeně) z fronty a aktualizovány vzdálenosti k sousedům (světle modře).

V dalším kroku algoritmu je z prioritní fronty vyňat vrchol ve vzdálenosti jedna od počátku a zároveň je označen jako zpracovaný 3.3. Do vrcholu ve vzdálenosti sedm byla nově objevena kratší cesta přes aktuální vrchol.

Obrázek 3.4 zachycuje zpracování v pořadí třetího vrcholu. Zde nebyl nalezen nový vrchol s celkovou vzdáleností pět od počátku a vrchol ve vzdálenosti dva, který je ale dále přes aktuální vrchol a tedy není jeho vzdálenost upravena. Aktuální vrchol je samozřejmě označen jako zpracovaný a není již dále zařazován do fronty.

Následující krok zajistí objevení nového vrcholu v pravém horním rohu obrázku a aktualizaci pravého dolního vrcholu, jelikož existuje přes aktuálně zpracovávaný vrchol kratší cesta (obr. 3.5). Pokud by byl následující vrchol vyjmutý z fronty, mohl by být algoritmus ukončen a vypsána nejkratší cesta z počátku do hledaného vrcholu. Pro účely směrování je ale žádoucí pokračovat ve výpočtu, dokud nebudou všechny vrcholy označeny jako zpracované. Žádoucí je to kvůli tomu, že směrovač může z takto předpočítaného grafu sestavit rovnou optimální směrovací tabulku, aniž by opakoval hledání jednotlivých cest mezi všemi směrovači.

Dijkstrův algoritmus může fungovat s časovou složitostí $O(|E|log(|V|)$, kde E je množina hran a V množina vrcholů. Záleží ovšem na implementaci prioritní fronty. [18]

3.2.2 Floyd-Warshallův algoritmus

Floyd-Warshallův algoritmus je odlišný od Dijkstrova tím, že dokáže najít nejkratší cestu mezi libovolnými dvěma vrcholy. Algoritmus funguje tak, že porovnává přímou cestu mezi vrcholy nejprve s vrcholem ležícím na cestě délky jedna. V dalším kroku jsou zahrnuty transportní vrcholy na cestě délky dva a tak dále, dokud se neprojde celý graf. [23]

Algoritmus má časovou složitost $O(|V|^3)$, jelikož pro každý uzel je otestováno, zda do všech ostatních uzlů nevede kratší cesta právě přes tento uzel. Výpočet lze předčasně ukončit v případě, kdy se matice přestane měnit.

Implementace je velmi jednoduchá viz listing 3.1. [16]

Tabulka 3.1: Počáteční stav matice vzdáleností

	1	2	3	4	5	6	7
1	0	7	1	2	∞	∞	∞
2	7	0	5	∞	1	∞	∞
3	1	5	0	1	∞	1	∞
4	2	∞	1	0	∞	1	3
5	∞	1	∞	∞	0	1	∞
6	∞	∞	1	1	1	0	1
7	∞	∞	∞	3	∞	1	0

Tabulka 3.2: Aktualizovaná vzdálenost mezi vrcholy 2 a 4 přes vrchol 1

	1	2	3	4	5	6	7
1	0	7	1	2	∞	∞	∞
2	7	0	5	**9**	1	∞	∞
3	1	5	0	1	∞	1	∞
4	2	∞	1	0	∞	1	3
5	∞	1	∞	∞	0	1	∞
6	∞	∞	1	1	1	0	1
7	∞	∞	∞	3	∞	1	0

Tabulka 3.3: Výsledná matice vzdáleností

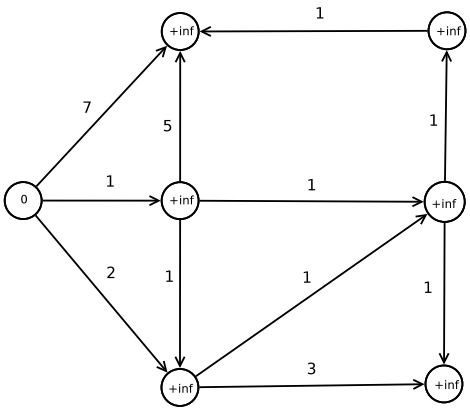
	1	2	3	4	5	6	7
1	0	4	1	2	3	2	3
2	4	0	5	3	1	2	3
3	1	5	0	1	2	1	2
4	2	3	1	0	2	1	2
5	3	1	2	2	0	1	2
6	2	2	1	1	1	0	1
7	3	3	2	2	2	1	0

Výpis 3.1: Pseudokód Floyd-Warshallova algoritmu

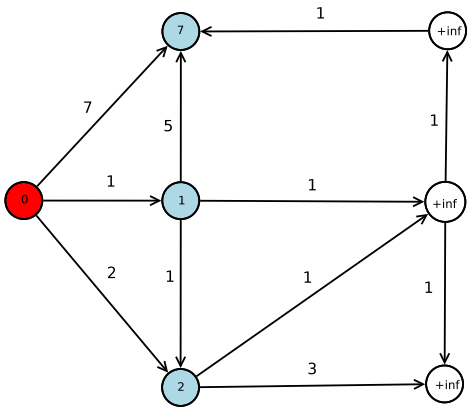
```
DistanceMatrix D[|V|][|V|];

for (int k = 0; k < |V|; ++k) {
  for (int i = 0; i < |V|; ++i) {
    for (int j = 0; j < |V|; ++j) {
      D[i][j] = min(D[i][j], D[i][k] + D[k][j]);
    }
  }
}
```

Výpis 3.2: Pseudokód dijkstrova algoritmu

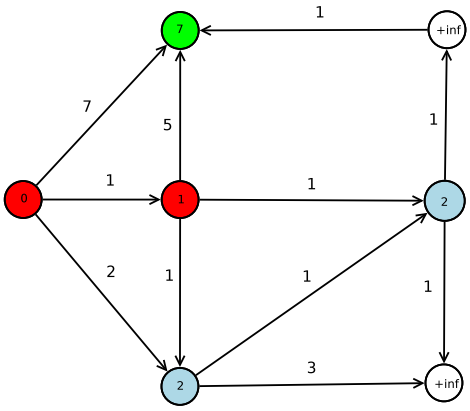
```
  priorityQueue = []
  queue add sourceNode

  while (priorityQueue not empty) {
    node = queue.getNearestNode()
    node.processed = trued

    for (Edge e in node.edges) {
      if (e.neighbor.processed {
        continue
      }

      newDistance = min(
        e.neighbor.distance,
        node.distance + e.cost
        )
      e.neighbor.distance = newDistance
      queue.add(neighbor)
    }
  }
```



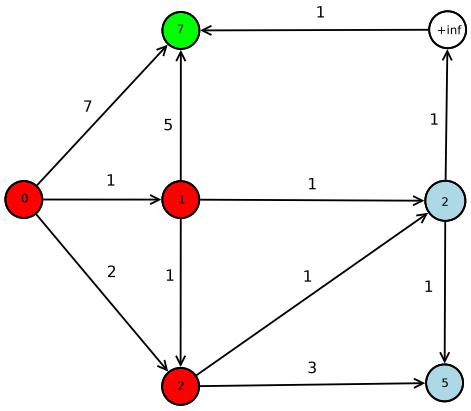
Obrázek 3.1: Počáteční hodnoty vzdáleností



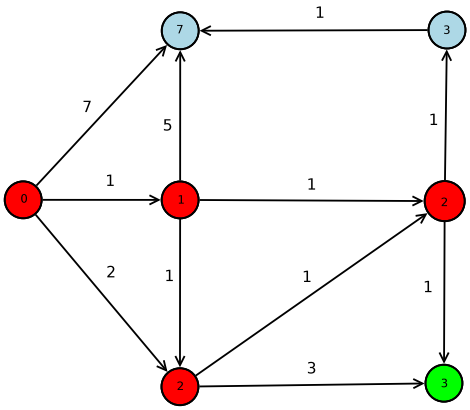
Obrázek 3.2: Vyjmutí počátečního vrcholu z fronty a aktualizace vzdáleností sousedů



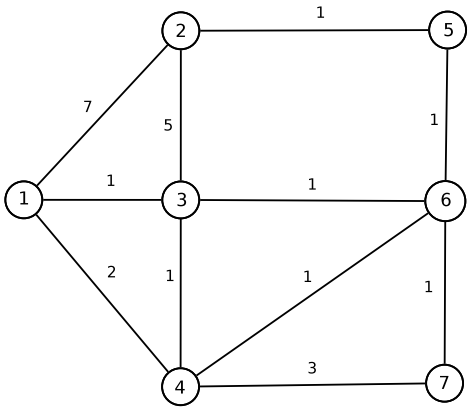
Obrázek 3.3: Vyjmutí nejbližšího vrcholu k počátku a zpracování jeho sousedů.



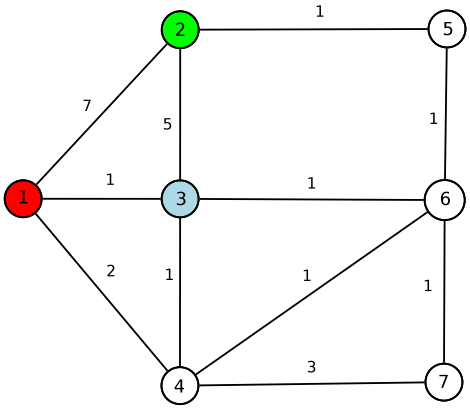
Obrázek 3.4: Vyjmutí dalšího nejbližšího vrcholu a aktualizace vzdáleností sousedů



Obrázek 3.5: Vyjmutí vrcholu 2 (vpravo) a aktualizace vzdáleností sousedů



Obrázek 3.6: Graf s očíslovanými vrcholy, jenž je zachycen maticí vzdáleností 3.1



Obrázek 3.7: Průchod první iterací barevně znázorněn v tabulce 3.2

3.3 Požadavky na kvalitu přenosu

Dnešní aplikace kladou různé nároky na vlastnosti přenosové soustavy a lze je zařadit minimálně do dvou kategorií. Například aplikace sloužící pro online komunikaci v podobě interaktivní textové komunikace či telefonních hovorů, by měly být spojovány přes spoje, na kterých je velmi nízká latence a ztrátovost paketů. Nebylo-li by tomu tak, komunikující uživatelé by museli čekat dlouho na protistranu a nejspíše by si velmi často skákali do řeči nebo by komunikace vypadávala zcela. Maximální přenosová rychlost hovorů není až tak podstatná, jelikož jsou hovory většinou komprimovány.

Ve druhé kategorii jsou aplikace, které stahují velké soubory. U nich je spíše preferována celková rychlost přenosu a na samotné latenci by nemělo až tak záležet. Data mohou být vysílána dávkově vysokou rychlostí, i když je mezi zdrojovým a cílovým bodem vysoká latence. Dopředné potvrzování paketů a dostatečná velikost přenosových bufferů zajistí plynulý přenos dat. [25] [32]

3.3.1 Způsoby vylepšení spolehlivosti přenosu

Klientská aplikace se může připojovat přímo na server, ale pak je odkázána na dané klasickými směrovacími algoritmy (především BGP), které mohou konvergovat velmi pomalu při změnách v topologii sítě.

Proto vznikají překryvné sítě, technologie a algoritmy, jejichž úkolem je především detekovat výpadek linky a transparentně změnit nekvalitní linku za nějakou kvalitnější. Toho je docíleno interními směrovacími tabulkami směrovacích uzlů překryvné sítě, které si udržují statistiky kvality linek mezi všemi sousedními uzly. Existuje tedy N^2 logických spojů, což může být hodně, ale vezme-li se v potaz počet směrovačů v překryvné síti, který bude řádově menší než počet směrovačů na třetí vrstvě *ISO/OSI*, není toto číslo tak závratné[2].

[2]Vztaženo k datovým strukturám jednoho směrovače.

3.3.2 Druhy uplatňování QoS

Kvalitu služeb lze zajišťovat mnoha způsoby. Záleží na okolnostech a účelu využití. Následující odstavce popíší dva významné způsoby zajištění kvality služeb.

Integrované služby

Pro zajištění kvality služby pro integrované služby je nutné, aby každý směrovací prvek podporoval rezervační protokol, například *RSVP*. Klientská aplikace si zažádá o tvorbu cesty s určením jakou přenosovou kapacitu vyžaduje.

Přenosová kapacita je přiřazována z pásem *CBR* (Constant bit rate), *VBR* (Variable bit rate), *ABR* (Available bit rate) a *UBR* (Unspecified bit rate). Klientská aplikace si zažádá o přidělení šířky pásma z vybraných podpásem a ta je mu rezervována skrze celou síť.

Rezervační protokol tedy musí umět každý směrovací prvek v síti. Není-li tomu tak, pak je možné zajistit kvalitu služeb na základě „nejlepší snahy“ rozlišováním služeb, které je popsáno v následujícím odstavci.

Diferencované služby

Alternativou nebo doplněním integrovaných služeb je zajišťování *QoS* rozlišováním jednotlivých služeb. K rozlišování služeb je nejprve nutné definovat, jakým způsobem se dané služby budou klasifikovat.

Dnes je hojně využíván protokol *TCP* a jeho rozlišení služeb spočívá v rozdílném čísle portu. Proto je možné jednotlivé dobře známé služby využívající dobře známé porty [28] klasifikovat do kategorií dle požadavků na kvalitu přenosu. Dle této vazby lze určit, jaká aplikace využívá dané spojení a podle toho volit vhodnou třídu kvality služeb.

V případě využití políčka *TOS* (Type of Service) v záhlaví *IP* je možné pakety značkovat a přímo je klasifikovat již na klientské stanici či kdekoliv v síti.

Výhodou tohoto přístupu je, že není nutné, aby každý směrovač v síti podporoval rezervační protokol. Další výhodou je, že klasifikace paketů dle čísla portu nebo *TOS* není stavová. Směrovač si nemusí uchovávat informace o rezervovaných linkách a jejich parametrech.

4 Časové řady a predikce

Pro směrování v překryvné síti s predikcí stavu linek je nezbytně nutné mít reprezentativní data, ze kterých je možné predikci tvořit. Data jsou získávána měřením, popsaným v sekci 4.2, ve formě časových řad.

Časová řada obsahuje tedy periodicky vzorkované veličiny týkající se stavu linky. Měřenými veličinami může být doba odezvy, četnost výpadků v daném časovém úseku nebo maximální propustnost linky.

Tyto časové řady je možné transformovat do různých podob, jež mohou zvýraznit či odhalit složku, která ovlivňuje její chování. Získané časové řady jsou dále předány predikčním algoritmům, které jsou více popsány v sekci 4.4. Výsledná předpověď časové řady pak může být využita směrovacím algortimem jako náhrada posledního známého stavu.

4.1 Časové řady

V běžných sítích se dají toky dat či latence považovat za spojitou veličinu. Z tohoto důvodu ovšem není možné uchovávat nekonečně detailní záznam v každém časovém okamžiku, ale je nutné provádět vzorkování. Obzvláště měření propustnosti spoje by zároveň spoj samotný zatěžovalo měřícími daty.

Časovou řadu lze matematicky definovat vzorcem 4.1

$$x[t] = x(0), x(\Delta t), x(2\Delta t), x(3\Delta t), ... \tag{4.1}$$

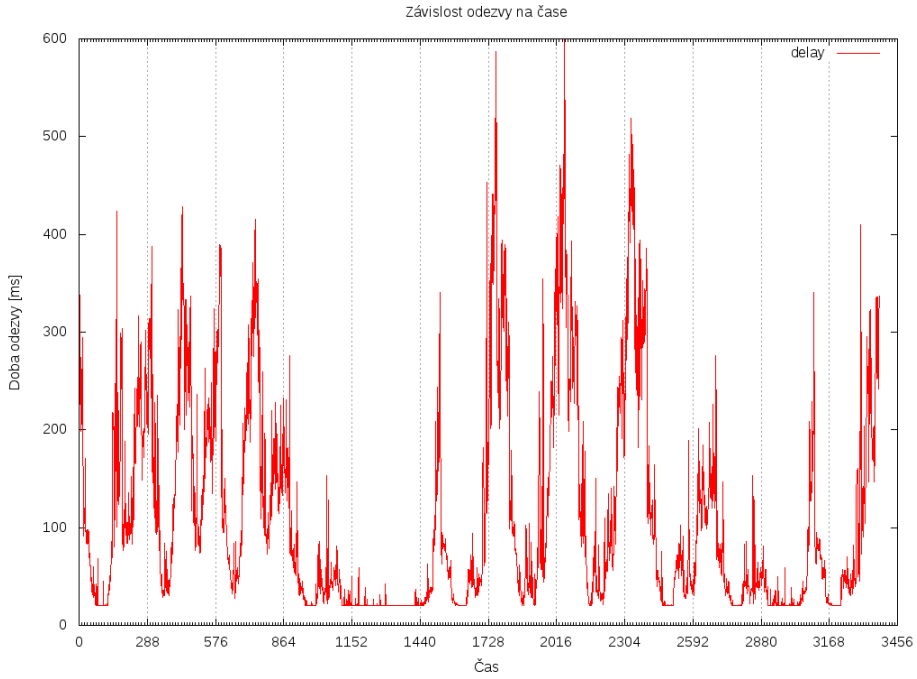
Dá se očekávat, že síťové toky se budou měnit převážně v závislosti na denní době podle toho, jak lidé vstávají a přesouvají se do práce v ranních hodinách a ve večerních opět zpět domů. Pokud bychom se podívali na graf 4.1 obsahující denní statistiky datových přenosů[1], je možné si všimnout opakujících se vzorců.

Tyto vzorce mají vliv na kvalitu a výpočetní náročnost predikce [6]. Dle určitých pravidel je možné je rozdělit do několika kategorií, které jsou popsány v následujících odstavcích.

4.1.1 Složky časových řad

Časové řady lze v mnoha případech rozdělit na podsložky, jejichž součet dává výslednou křivku, jež odpovídá naměřeným hodnotám. [14]

[1] Hodnoty jsou vykresleny z pětiminutových průměrů.



Obrázek 4.1: Doba odezvy v závislosti na denní době

Sezónní složka

Data, u nichž lze vypozorovat periodu opakování, lze označit jako sezónní. Z pohledu překryvných sítí je zajímavá denní sezónnost datových přenosů v závislosti na denní době.

Časová řada nemusí mít jen jednu sezónní složku, může jich být klidně více. Například kromě denní složky je možné nalézt i roční sezónní složku, kdy lidé odjíždějí na dovolenou a nesledují online televize.

Pro predikci je nutné zvolit pro nás nejvýznamnější sezónní složku a té přizpůsobit model výpočtu a ostatní méně významné sezónní složky ignorovat.

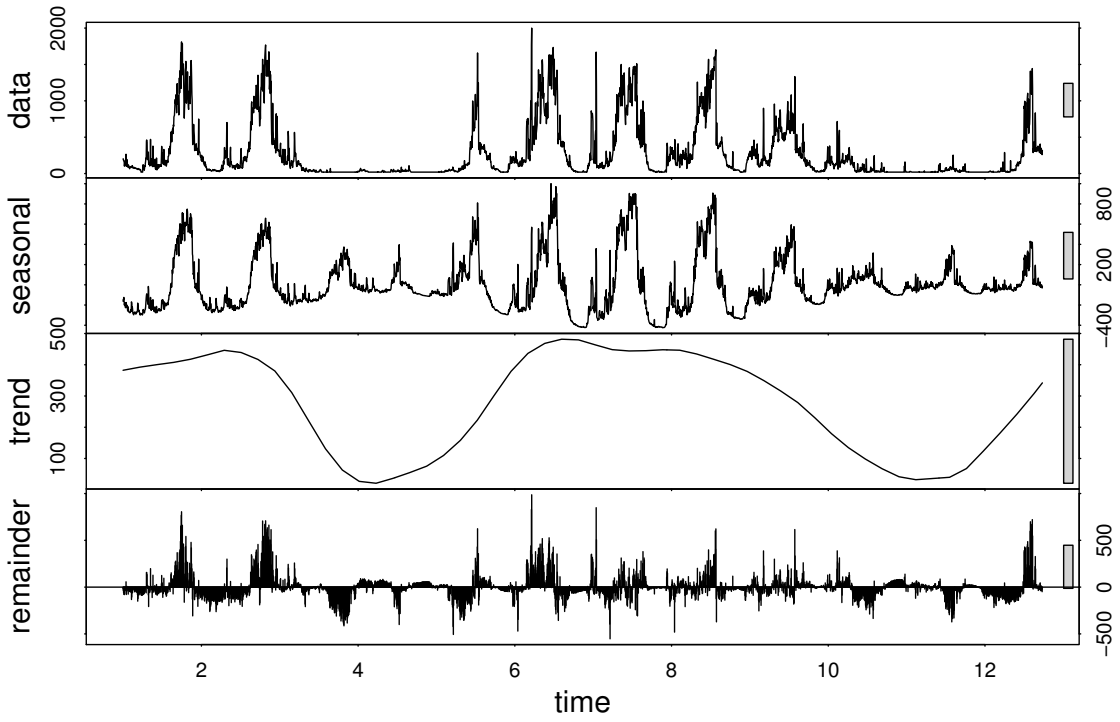
Roční sezónní složka pro směrování není tolik důležitá, jako znalost denní sezónní složky, která zachycuje podstatnější změny pro směrování v rámci dne.

Ke správnému zachycení sezónních složek je nutné vzorkovat minimálně dvojnásobnou frekvencí, než má sledovaná sezónní složka dle Nyquistova teorému. [33]

Druhý graf na obrázku 4.2 zachycuje sezónní složku časové řady.

Trend

Trendová složka existuje, pokud naměřené hodnoty dlouhodobě rostou nebo klesají. Na grafu 4.2 přenesených dat je vidět, že i trendová složka může být periodická (sezónní). Zachycený trend má týdenní periodu, kde o víkendu není přenášeno velké množství dat a přes týden se přenosy znásobují. [14]



Obrázek 4.2: STL dekompozice časové řady

Náhodná složka

Náhodná složka může být chápána jako posloupnost odchylek součtu sezónní a trendové složky od naměřených dat. Většinou se jedná o bílý šum. [14]

4.2 Sběr statistických dat

Jelikož tato práce má za cíl implementovat predikci v překryvné síti. Je tedy žádoucí na jednotlivých směrovačích měřit určité charakteristiky linek, aby na základě naměřených dat a jejich analýzy mohla být spočítána predikce pro následující časový interval.

U síťových linek jsou pro směrování důležité měřitelné charakteristiky linek. Mezi měřitelné charakteristiky lze zařadit latenci mezi sousedními uzly, která je důležitá pro určení nejvýhodnější trasy pro služby vyžadující nízkou latenci se zanedbatelnými nároky na datovou šířku pásma. Telefonní hovory jsou komprimované a při nedostatečné šířce pásma lze do určité míri snížit hlasovou kvalitu a tím i datový tok. Další měřitelnou veličinou je chybovost linky. Tu lze měřit buď zvlášť zasíláním paketů přes nespojované protokoly *UDP* nebo *ICMP* nebo transparentním vkládáním měřících dat do existujících spojení.

Výpadek linky je pak detekován nedoručením odpovědi na požadavek v daném časovém limitu. Spolehlivost linky je možné do jisté míry sloučit s měřením doby odezvy, která bude při nedoručení odpovědi nahrazena hodnotou ∞ a tím bude ztrátovost přetransoformována na dobu odezvy.

Měření lze provádět dvěma způsoby a ty jsou detailněji rozebrány v následujících dvou sekcích. Vhodné způsoby uložení dat jsou uvedeny v sekci 4.3.

4.2.1 Aktivní měření

Aktivní měření znamená vysílání měřících paketů ať už mezi směrovači, tak mezi směrovači a externími servery, což by mohlo způsobit zahlcení zúčastněných stran a popřít tak celou ideu překryvné sítě – zlepšení provozních parametrů linek.

Z těchto důvodů je vhodné aktivní měření neprovádět kontinuálně, ale spíše si vybírat náhodné cíle a ty testovat v náhodném intervalu.

Aktivně lze měřit latenci mezi jednotlivými směrovači odesíláním ping požadavků s časovým razítkem. Router, který obdrží požadavek na ping v zápětí odešle na zdrojovou adresu datagramu odpověď se stejným časovým razítkem. Iniciátor měření tak může spárovat požadavky s odpověďmi a naměřit Roud Trip Time (RTT) – čas cesty datagramu od iniciátora do cíle a zpět.

Výhodou aktivního měření je rychlost získání požadovaných dat, za cenu zvýšení zatížení sítě o měřící pakety. Měření propustnosti nelze opakovat tak často, a proto je lepší v tomto případě využít pasivního měření.

4.2.2 Pasivní měření

Pasivní měření lze využít k získání přehledu o nevyužité kapacitě linky z dat protékajících směrovačem. Není tedy nutné iniciovat měření v periodických intervalech a zatěžovat tak zbytečně spoje mezi směrovači. Naměřená data ovšem nelze brát jako dogma, které je pravdivé ve všech případech. Směrovačem může protékat málo dat a vypočtená volná kapacita by tedy byla velká, ale musí se brát v potaz, že nízký průtok dat může být způsoben vysokou ztrátovostí linky.

Pasivně naměřená data by z tohoto důvodu měla být vážena koeficientem ztrátovosti, který lze získat z aktivního měření.

4.3 Uchování naměřených dat

Naměřená data je vhodné uchovávat pro využití v predikčním algoritmu. Existuje více možností, které se liší především paměťovou náročností a náročností na zpracování uchovaných dat. Pro krátkodobou predikci je možné využít krátkou časovou historii, jelikož dlouhé historické časové úseky nemají příliš velký vliv na kvalitu predikce predikčními metodami uvedenými v sekci 4.4.

K uchování je možné zvolit úspornou Round robin databázi fungující i nad operační pamětí, nebo odlehčenou variantu relační databáze.

V následujících odstavcích budou prodiskutovány možnosti uchování naměřených dat a jejich výhody či nevýhody.

4.3.1 Round robin databáze

Round robin databáze dokáže uchovávat N posledních položek za konkrétní časové intervaly. Nová naměřená data pak přepisují ta stará, čímž se zajišťuje aktuálnost statistických dat – průměry, mediány atd.

Další výhodou je malá paměťová náročnost takovéto databáze, proto je možné ji provozovat celou v RAM. Výsledkem je pak i vysoká rychlost výpočtů nad daty v ní.

Nevýhodou ovšem je ztráta historických dat, která v sobě zahrnují sezónnost změn přenášených objemů dat a dlouhodobé trendy provozu. Zároveň se tím ztrácí možnost optimalizace parametrů předpovědních modelů, kvůli absenci dlouhodobě uchovávaných statistik. Tuto mezeru pak vyplňují klasické databáze.

4.3.2 Relační databáze

Relační databáze jsou většinou pomalejší než Round robin databáze, protože obsahují množství naměřených dat za celou dobu měření. Efektivita vyhledávání může být zvýšena přidáním indexů nad sloupečky s časovým razítkem měření.

Nad daty v relační databázi lze provádět seskupování a transformace za účelem získání jiných dat a výsledků, které nemusí být na první pohled patrné. Z časových řad lze transformačními metodami získat sezónní složku, dlouhodobý trend a chybovou složku naměřených charakteristik linky. Tato naměřená data pak lze předložit predikčním algoritmům uvedeným v sekci predikce 4.4.

Relační databáze je výhodná především díky historizaci naměřených údajů, které lze přes pohledy reprezentovat, jako by naměřená data byla uchovávána v Round Robin databázi. Výkonnostní handicap může být eliminován využitím aplikačních cache, které omezí počet kladených dotazů do databáze.

4.4 Algoritmy predikce

Naměřené časové řady lze různými způsoby prodloužit a získat tak více či méně přesnou předpověď. Nic ovšem nelze předpovídat zcela přesně a předpověď tak nemusí odpovídat realitě. Čím více se prodlužuje předpovídaný interval, tím více je předpověď nepřesná.

V překryvné síti není nutné předpovídat na celé dny dopředu, protože většina přenosů se vejde řádově do několika hodin nebo méně. [15]

V následujících odstavcích budou popsány základní principy a možnosti predikce. Prakticky naměřené hodnoty jsou konzultovány v kapitole měření a simulace 9.

4.4.1 Předpoklady pro predikci v překryvné síti

Nejprve je nutné definovat předpoklady pro predikci v překryvné síti. Ze znalosti předpokladů pak lze navrhnout optimální způsob měření a algortimus predikce.

Prvním důležitým faktorem, který je nutné stanovit, je očekávaná délka predikovaného intervalu. Ta nepřímo ovlivňuje frekvenci snímkování, které by dostatečně pokrylo vlastnosti konkrétní linky, a také výpočetní složitost předpovědi. Dle statistik [15] je většina přenosů uskutečněna v krátkém časovém horizontu. Pro agilní směrování v překryvné síti tedy není nutné mít předpověď na několik hodin dopředu, která navíc nemusí být ani přesná kvůli tomu, že se nepředvídatelně změnila trasa na podkladové síti.

Druhým faktorem je, že již byly ověřeny nedostatky metody $ARIMA$ používané pro dlouhodobou predikci síťových přenosů. [10]

Z těchto důvodů bude praktická práce zaměřena na krátkodobou predikci stavů linek.

4.4.2 Metoda nejmenších čtverců

Metodou nejmenších čtverců lze získat koeficienty polynomu, kterým se dá proložit naměřená časová řada. Prodloužením rozsahu mimo rozsah naměřených dat je provedena extrapolace, která ale nemusí vůbec odpovídat realitě, protože jsou zanedbány trendy časové řady.

Metoda nejmenších čtverců je využívána jen jako prostředek pro vyhlazení signálu, na který je pak aplikován *FIR* filtr popsaný v odstavci 4.4.3. [17]

4.4.3 Filtrace signálu

Naměřená data většinou nemají hladké průběhy a před jejich použitím je nutné je předzpracovat. K předzpracování signálu jsou využívány nejčastěji dva druhy filtrů – *FIR* a *IIR*. Tyto číslicové filtry jsou využívány nejen k vyčištění digitalizovaného signálu, ale také jejich teoretické základy slouží jako stavební metody predikčních metod včetně zde zmiňované metody $ARIMA$.

FIR filtr

Finite impulse response je implementací konvoluce vstupního signálu s vektorem koeficientů β_i, jejichž počet je $q+1$; viz rovnice 4.2. Název je odvozen od pozorování skutečnosti. Když je vstup $u[t]$ impulzní funkcí, pak výstup x trvá tak dlouho, jako $q+1$, které musí být konečné. [33] [17]

Prakticky je naměřená časová řada přenásobena váhovými koeficienty, které byly zjištěny experimentálními pokusy. Produkty násobení jsou pak sečteny a výsledkem je očekávaná hodnota v budoucnosti. Zahrnutím očekávané hodnoty do výpočtu je pak možné vypočítat další očekávanou hodnotu a takto lze pokračovat, dokud není vypočten požadovaný počet predikovaných hodnot.

$$x[t] = \sum_{i=0}^{q} \beta_i u[t-i] \tag{4.2}$$

IIR filtr

IIR filtr je využíván Kalmanovým filtrem. Oproti *FIR* filtru může být *IIR* filtr nižšího řádu a tím pádem lze dosáhnout i nižší výpočetní složitosti. Zároveň díky nižšímu řádu dosahuje kratšího zpoždění mezi vstupem a výstupem. Nevýhodou je ovšem nestabilita, která se projevuje rozkmitáním signálu, a složitější implementace.

Infinite impulse response filter je charakterizován p koeficienty dle vztahu 4.3, kde t je bod na časové ose, i je pořadí časového bodu v historii a $x[t]$ odhadovaná veličina v čase t. Vstup $u[t]$ se zároveň přímo podílí na tvorbě výsledného signálu a zároveň výsledný signál je roven váženému součtu předchozích naměřených výstupních hodnot $x[t-i]$.

$$x[t] = \sum_{i=1}^{p} \alpha_i x[t-i] + u[t] \tag{4.3}$$

4.4.4 ARIMA

Autoregresivní integrační metoda s plovoucími průměry je sofistikovaná metoda, jež ze znalosti historických dat dokáže získat předpověď na určitý časový úsek. ARIMA model je kombinací AR(p) modelu, který je definován vztahem 4.4, a MA(q) modelu klouzavého průměru chyb daného vztahem 4.7.

Parametrem $AR(p)$ modelu je řád autoregresivní části. Řádem je míněno, kolik hodnot v minulosti bude započítáno do regresivní části.

Pokud je průměrná odchylka $\epsilon[t]$ relativně zanedbatelná vůči $x[t]$, pak lze $x[t]$ odhadnout dle vzorce 4.6, kde je zavedena proměnná w_i 4.5, která je odhadem jednotlivých α_i. Je možné zpozorovat souvislost s FIR filtrem ze sekce 4.4.3.

$$x[t] = \sum_{i=1}^{p} \alpha_i x[t-i] + \epsilon[t] \tag{4.4}$$

$$\hat{x}[t] = x[t] - \epsilon[t] \tag{4.5}$$

$$x[t] = \sum_{i=1}^{p} w_i x[t] \tag{4.6}$$

Druhou složkou $ARIMA(p, q, d)$ modelu je model plovoucích průměrů, jehož základem je FIR filtr aplikovaný na změřený neznámý integrační signál. Model lze matematicky zapsat vztahem 4.7, kde $q + 1$ je řádem filtru, $x[t]$ predikovaná hodnota v čase t, vektor β_i váhových koeficientů a vektor chyb ϵ.

$$x[t] = \sum_{i=0}^{q} \beta_i \epsilon[t-i] \tag{4.7}$$

Třetí složkou jest $I(d)$, což je součet rozdílů d naměřených hodnot vůči poslední naměřené hodnotě. Matematicky lze tuto metodu zapsat výrazem v „back shift“ notaci 4.8. [13]

$$(1-B)^d y_t \tag{4.8}$$

Výsledný výpočet $ARIMA(p, d, q)$ je dán využitím všech tří výše zmíněných vzorců tak, jak je naznačeno v zápisu 4.9. [14]

$$(1 - \alpha_1 B - ... - \alpha_p B^p)(1-B)^d y_t = c + (1 + \beta_1 B + ... + \beta_q B^q)\epsilon_t \tag{4.9}$$

Dle pramenů [10] a [12] není $ARIMA$ příliš vhodná pro dlouhodobou předpověď síťových přenosů. Z hlediska směrování za účelem zvýšení kvality přenosu je ale dlouhodobá předpověď příliš velkým luxusem, který není nezbytně nutný. Krátkodobá předpověď v řádu několika minut uspokojivě pokryje větší část běžných spojení.

4.4.5 Neuronové sítě

Moderní informatika se nemusí spoléhat jen na nové vynálezy, které vznikají na zelené louce. Ve velké míře lze přejímat vzory a postupy již vymyšlené samou přírodou. Neuronové sítě jsou implementací mozku, který je sám o sobě velmi výkonným výpočetním systémem.

Neuronové sítě se používají v mnoha odvětvích kybernetikou počínaje a zpracováním analytických dat konče. Základem je několik vrstev neuronů, jejichž počet se odvíjí od požadavků na granularitu očekávaných výsledků. Nutnou podmínkou fungování neuronové sítě je vstupní vrstva neuronů, jež snímají vstupy a přes váhově ohodnocené spoje s neurony další vrstvy vysílají rozpoznané signály. Další vrstvou může být rovnou výstupní vrstva, nebo může být tok informací směrován do skrytých vnitřních vrstev.

Jednotlivé neurony jsou mezi sebou napříč vrstvami propojeny ohodnocenými hranami. Ohodnocovací funkce určuje, zda bude neuron vstupním signálem aktivován nebo zůstane v neaktivním stavu.

Neuronové sítě lze natrénovat jednou staticky a posléze využívat natrénovaných koeficientů k predikci dalších hodnot. Nicméně staticky natrénovaná síť nemusí reflektovat měnící se charakteristiky linky v dlouhodobém časovém horizontu.

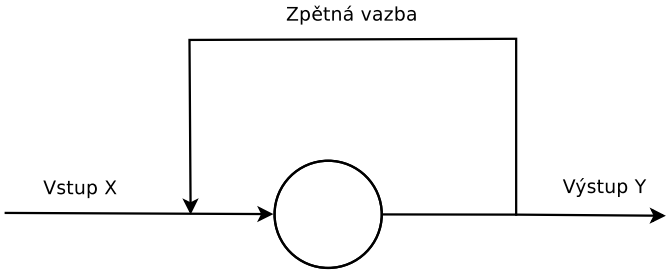
Alternativou může být ale i dynamické učení, které dokáže naučené koeficienty upravovat za běhu a přizpůsobovat se tak novým dříve nenaučeným vlastnostem a charakteristikám měřené veličiny. [31] [34]

Neuronové sítě mohou mít mnoho různých podob, které se liší množstvím použitých vrstev i způsobem jejich propojení. V následujících odstavcích jsou zmíněny druhy, jež jsou využitelné pro predikci časových řad.

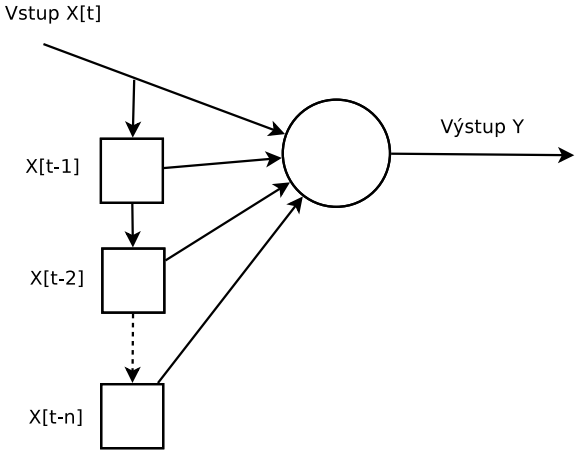
Sítě se zpětnou vazbou

Reprezentantem neuronových sítí se zpětnou vazbou je Hopfieldova síť. Základem je řada neuronů, kterým je vstupem měřená veličina spolu s předchozím výstupem.

Na obrázku 4.3 je zobrazena struktura neuronu se zpětnou vazbou.



Obrázek 4.3: Zpětnovazební neuronová síť



Obrázek 4.4: Neuronová síť s pamětí

Sítě s pamětí

Sítě s pamětí využívají paměťových článků ke zpoždění vstupního signálu. Zpoždění vstupního signálu je využíváno ve filtrech uvedených v sekci 4.4.3. Není tedy náhodou, že neuronová síť může posloužit jako implementace těchto filtrů.

Obrázek 4.4 zobrazuje neuron, jehož vstupem je aktuální hodnota zároveň s hodnotami získanými z paměťových členů.

Sítě se zpětnou propagací chyby

Výše zmíněné sítě mohou být vylepšeny o zpětnou propagaci chyby, která zajišťuje přizpůsobování koeficientů vah lineárními gradientními metodami pro minimalizaci cenové funkce. Cenovou funkcí je míněna odchylka predikovaných hodnot od skutečnosti.

5 Problematika směrování v překryvné síti

V kapitole 3 byly popsány algoritmy směrování v klasických *TCP/IP* sítích. Směrování v překryvné síti má ale svá úskalí, která směrování mohou komplikovat. Problémům, které mohou nastat v překryvné síti, budou věnovány následující odstavce.

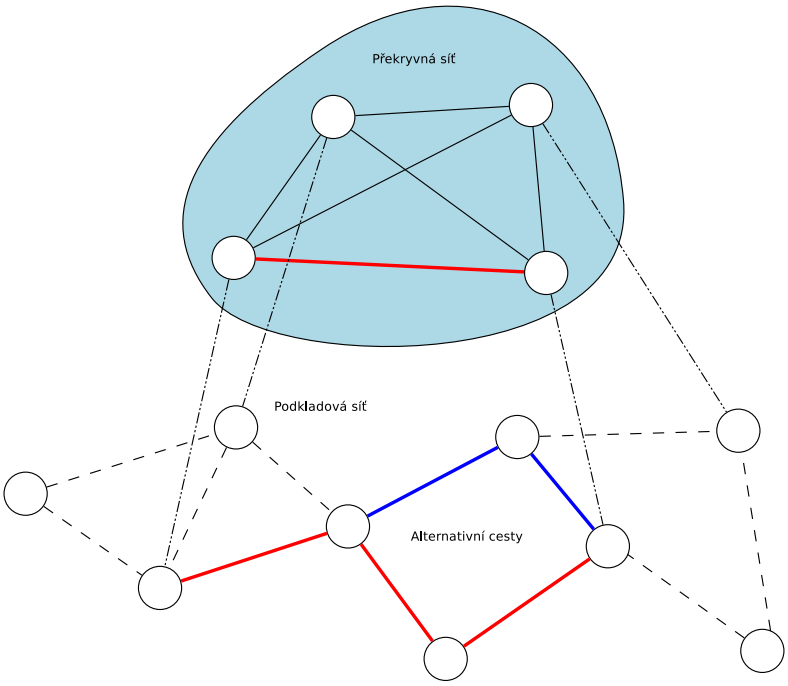
5.1 Detekce výpadku a propagace této informace

Směrování v internetu funguje převážně na reaktivní bázi, to znamená mimo jiné, že výpadky jsou detekovány až když se stanou. Problém je, že vzdálené uzly sítě mohou dostat informaci o výpadku cesty s velkým několikaminutovým zpožděním. Naproti tomu překryvná síť dokáže zkonvergovat mnohem rychleji díky tomu, že je mnohem méně rozsáhlá co se týče počtu směrovacích uzlů.

5.2 Měnící se směrování na síťové vrstvě

Překryvná síť operuje nad jinou sítí, která neví, co v ní protéká, a proto směrovací algoritmy nižších vrstev mohou dynamicky měnit cesty bez informování překryvné sítě. Ta tuto změnu může detekovat aktivním měřením parametrů linek a následně reagovat změnou svých cest. Stávala-li by se tato změna periodicky, pak by nepředstavovala takový problém, jako když se cesty mění náhodně. Predikce stavu linky pak musí najít nové parametry, které by odpovídaly novému stavu. Během této doby mohou být předpovědi stavu linky spíš kontraproduktivní.

Na obrázku 5.1 je v obláčku červeně zobrazena virtuální cesta mezi uzly překryvné sítě a v dolní části táž cesta namapovaná na fyzické spoje mezi směrovači. Modře je zobrazena alternativní cesta, mezi kterou mohou směrovače přepínat a komplikovat tak detekci parametrů virtuální linky.



Obrázek 5.1: Alternativní cesta v podkladové síti mezi dvěma uzly překryvné sítě.

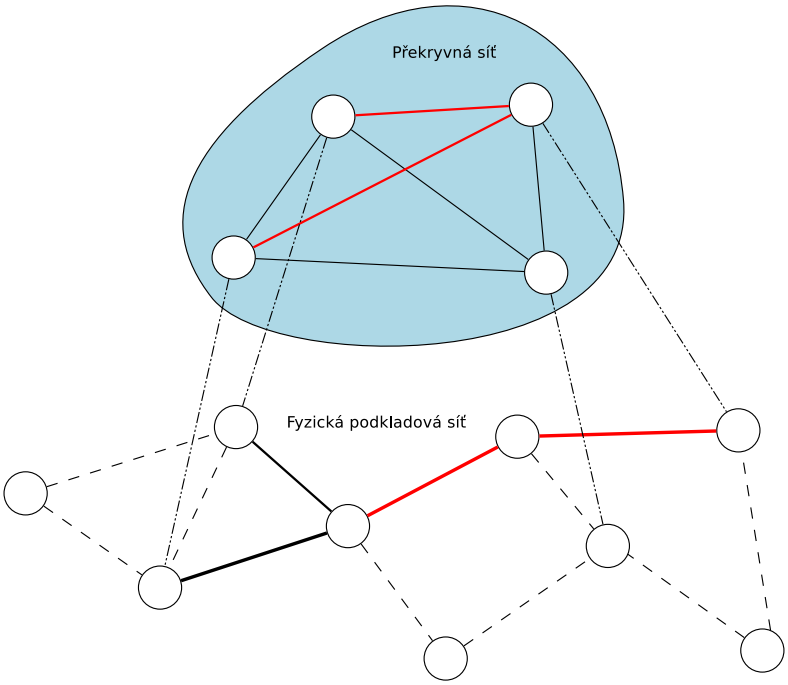
5.3 Stejná fyzická cesta

Z hlediska vyvažování zátěže nebo volby kvalitnějších cest je další komplikací směrování různých virtuálních linek na stejné fyzické linky. Jakmile nastane závada nebo výpadek na fyzické lince, pak i všechny virtuální budou postiženy stejným problémem. Záložní cesta by tedy měla být volena s ohledem na podkladovou topologii sítě.

Obrázek 5.2 zachycuje tuto problémovou situaci. V obláčku jsou červeně zvýrazněny virtuální linky a ve spodní části společná fyzická cesta, jejíž výpadek by obě virtuální cesty ovlivnil.

5.4 Samoovlivnění kvality linek směrováním

Překryvná síť s vlastním predikčním směrováním může ovlivnit sama sebe skrz nevhodnou předpověď tím způsobem, že jedné lince bude předpovězena vyšší kvalita než ostatním linkám, ale tím, že se veškerý provoz bude přeposílat přes takto optimisticky předpovězenou reprezentativní cestu, může nastat paradoxně zahlcení uzlu nebo linky a celkové snížení kvality přenosu.



Obrázek 5.2: Společná fyzická linka pro dvě různé virtuální linky.

6 Predikce reálných dat

V této kapitole budou zmíněny problémy, které byly objeveny při zkoušení parametrů předpovědi na reálných datech.

Knihovny jazyka R poskytují komplexní nástroje pro různorodé výpočty. V diplomové práci byla využita knihovna *forecast*, poskytující implementaci predikčních algoritmů.

Mezi implementované predikční algoritmy patří metoda $ARIMA$ s uživatelsky zadanými řády, `auto.arima` dynamicky odhadující řád modelu z dodaných dat a $NNETAR$ implementující autoregresivní neuronovou síť.

6.1 ARIMA

Autoregresní metoda s pevně danými parametry by mohla být rychlejší nežli `auto.arima`. Ovšem konstantnost daných parametrů nemusí vhodně modelovat konkrétní linku mezi uzly překryvné sítě. V některých případech může selhat hledání řešení kořenů rovnic a predikce není uskutečněna vůbec nebo s chybami, viz 6.1.

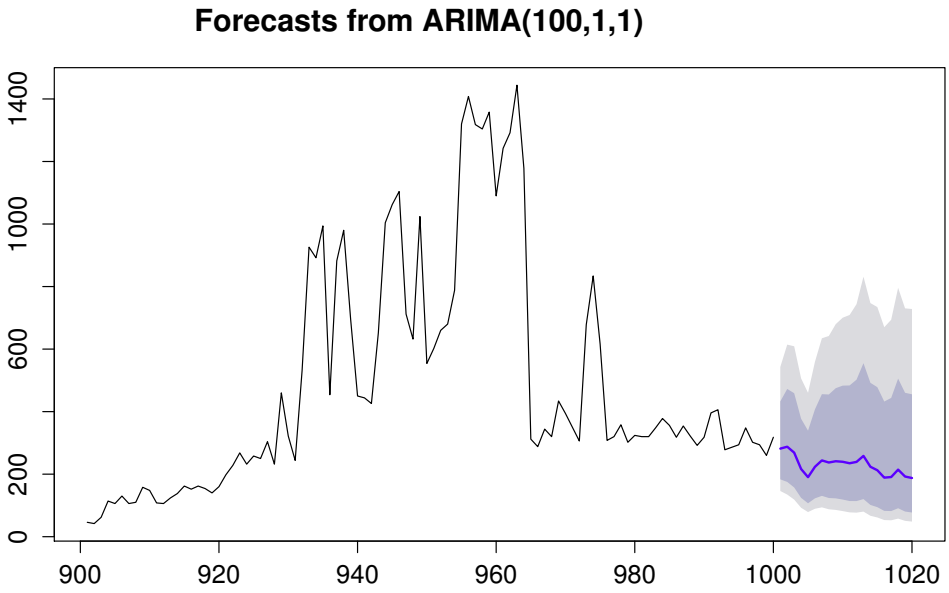
Výpis 6.1: Chybové hlášení při nevhodných parametrech modelu

```
ar <- Arima(ts, order = c(10,1,10)
Warning message:
In sqrt(diag(x\$var.coef)) : NaNs produced
```

Z výkonového hlediska také není vhodné používat příliš velkou historii, protože doba výpočtu je pak stejná nebo delší než predikovaný interval, který je v době získání predikce již minulostí. Stejným způsobem se projevují i vyšší řády p, q, d a násobně prodlužují výpočet.

Na obrázku 6.1 je použita predikce na základě modelu $ARIMA(100, 1, 1)$, jejíž predikce je sice relativně přesná, ale výpočet samotný trval více než patnáct minut. Tudíž tento model nelze využít pro směrování s predikcí na desetiminutové úseky.

Parametry byly zvoleny tak, aby autoregresivní část zasahovala 100 záznamů do historie. $ARIMA$ model pak vytvoří 100 výhových koeficientů tak, aby vznikla co nejmenší chyba v předpovědi.



Obrázek 6.1: Předpověď modelu ARIMA(100,1,1)

6.2 Auto ARIMA

Funkce `auto.arima` z knihovny *forecast* poskytuje mechanismus pro automatické hledání parametrů modelu. Tato metoda je vhodná pro data s měnící se charakteristikou, jelikož dokáže ve většině případů najít vhodné parametry modelu reflektující aktuální dění na síti.

V některých případech ovšem automatický proces není dokonalý a nemusí poskytnout relevantní výsledky. Na výpisech 6.2 a 6.3 jsou zachyceny dva chybové stavy, z nichž první skončil chybou a druhý nepřesným výpočtem.

Výpis 6.2: Chybová hláška automatického hledání parametrů modelu

```
Error in myarima(x, order = c(p, d, q),
                    seasonal = c(P + 1, D, Q + 1),  :
  root finding code failed
  Calls: auto.arima -> myarima
  Execution halted
```

Výpis 6.3: Chybová hláška automatického hledání parametrů modelu

```
ar <- auto.arima(ts)
Warning message:
In auto.arima(ts) :
  Unable to fit final model using maximum likelihood.
  AIC value approximated
```

6.3 Neuronová síť NNETAR

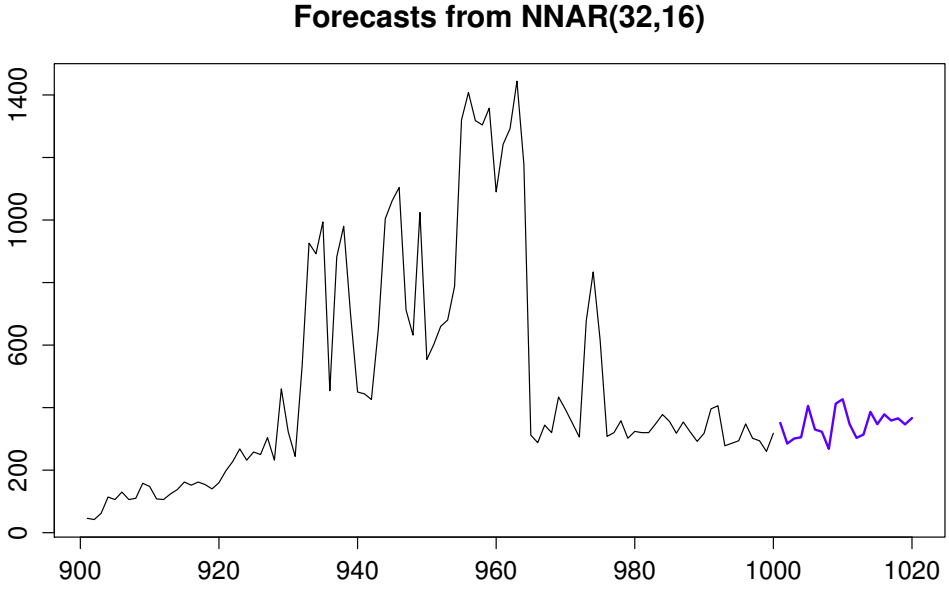
Praktickým ověřováním bylo zjištěno, že neuronová síť $NNETAR$ dosahuje podobných výsledků jako $ARIMA$ s tím rozdílem, že si nedovede dobře poradit s malými zdrojovými daty. Pokud se neuronové síti zadá příliš malá učící množina, odmítne pak předpovídat hodnoty.

Další nevýhodou je vyšší výpočetní náročnost, kdy výpočet nad reprezentativní množinou dat trvá řádově minuty.

6.4 Výběr predikční metody pro implementaci

Ačkoli neuronová síť poskytuje mocnější aparát pro predikci, její výpočetní i paměťová složitost je výrazně větší než u metody $ARIMA$ při učení koeficientů. Znovupoužitím modelu překryvné sítě by mohla ale výpočení náročnost poklesnout, nicméně testovací prostředí nebylo kompatibilní s nejnovější verzí knihovny forecast, kde je implementováno znovupoužití předpočítaného modelu.

Kvůli těmto nevýhodám byla zvolena úspornější implementace základní metody $ARIMA$. Na základě výsledků simulací byly navrženy další variace $ARIMA$, které mají za cíl snížit výpočetní náročnost nebo zvýšit přesnost predikce. Viz následující sekce.



Obrázek 6.2: Předpověď modelu NNETAR(32,16)

6.5 Modifikace ARIMA

Metoda *ARIMA* nepotřebuje velké množství dat ke krátkodobé předpovědi. Nicméně dle výsledků experimentů s predikcí neposkytuje optimální výsledky pro směrování. Proto byly implementovány a otestovány vlastní modifikace, které vylepšují kvalitu predikce.

6.5.1 ARIMA s posuvem

Hlavní myšlenkou, která vedla k tomuto řešení, je pozorování skutečných časových řad datových přenosů odhalující periodičnost. Každý den vyobrazený v grafu 9.1 má přibližně stejný průběh jako den následující i předcházející[1].

ARIMA s posuvem tedy nepředpovídá dle posledních N naměřených hodnot, ale dle hodnot naměřených předcházející den – tedy $N - 24h$.

Nevýhodou tohoto algoritmu je, že nereflektuje aktuální stav sítě a může tím pádem poskytovat horší výsledky.

[1] vyjma víkendů

6.5.2 ARIMA kombinující dvě časové řady

Jelikož výpočetní výkon je omezující záležitostí, vznikl nápad na redukci množství dat, předávaných predikčnímu algoritmu. Omezení bylo docíleno vedením dvou časových řad.

Jedna řada je získávána periodickým frekventovaným měřením řádově v minutách. Díky této řadě lze získat trend posledních několika desítek minut a pružně reflektovat aktuální změny na síti.

Druhá řada vzniká hodinovou agregací řady první. Díky agregaci je možné ze stejného množství agregovaných vzorků získat dlouhodobější trend a tím zpřesňovat predikci z první časové řady.

Zpřesnění je provedeno váženým průměrem výsledků predikcí obou časových řad.

6.5.3 ARIMA kombinující dvě řady s posuvem

Kombinací dvou předchozích alternativ byla vytvořena tato metoda, která využívá aktuální časové řady získávané periodickým měřením každou minutu a agregovanou hodinovou řadu. Hodinová řada není ovšem aktuálně měřená, ale pochází z měření předchozího dne.

Obě řady jsou opět váženým průměrem složeny do výsledné predikce. Predikce tohoto algoritmu tedy reflektuje dění na síti v posledních minutách a zároveň je předpoveď každou hodinu zpřesněna o dění ve stejnou hodinu minulý den.

7 Analýza architektury aplikace

V této kapitole budou rozebrány možnosti implementace překryvné sítě a jejich teoretické výhody či nevýhody. V sekci 7.1 jsou uvedeny možnosti členění aplikace. Sekce 7.2 se zaobírá vhodností využití různých programovacích jazyků a důvody konkrétní volby u daných modulů.

7.1 Architektura aplikace

Dnešní aplikace jsou většinou členěny do několika vrstev, které mají jasně definovanou zodpovědnost nad činností, kterou dělají. Složitější aplikace využívají alespoň modelu *MVC*, kde každé písmeno symbolizuje jednu vrstvu aplikace.

- **M**odel: Tato vrstva obsahuje především přepravky – data transfer objekty (*DTO*); a rozhraní pro přístup k datům – data access objekty (*DAO*) , které z libovolného zdroje dat poskytují datové objekty. *DAO* vrstva má přesně dané rozhraní a implementace tohoto rozhraní může poskytovat data například z relační databáze, *No-SQL* databáze, či prostého textového souboru. Podstatné je, že je splněn kontrakt rozhraní a vrácena požadovaná data.

- **V**iew: Vrstva určená k prezentaci dat uživateli či jinému programu přes definované *API*. Vstupem bývají data transfer objekty, které jsou zpracovány a převedeny na datové struktury vhodné k prezentaci vstupních dat. Výstupem může být buďto *HTML* stránka, *JSON* dokument, či jiný formát vhodný k zpracování uživatelem nebo externím programem.

- **C**ontroller: Základem aplikace je právě vrstva řídící, ve které se nachází aplikační logika. U rozsáhlých webových aplikací je tato vrstva většinou členěna na další podvrstvy, které oddělují logické celky dle specifického zaměření – validace dat, transformace dat modelu na data view, kontrola oprávnění, operace nad daty.

Členění aplikace na vrstvy nicméně neříká nic o tom, jak bude celý program členěn fyzicky. Existují aplikace, které jsou sice interně rozvrstveny,

ale výsledkem překladu je monolitický binární spustitelný soubor. Takováto aplikace je spustitelná bez závislostí na systémové knihovny. Výhodou je jednoduchost použití a spouštění aplikace, nicméně nevýhodnou se stává aktualizace aplikace nebo nějakého jejího modulu. Při každé změně je nutné distribuovat opravený binární soubor jako celek. Před samotnou aktualizací je nutné nejprve vypnout běžící aplikaci a až poté nasadit novou verzi. To má za následek výpadek služby na určitý čas.

Výpadky a aktualizace služby lze redukovat využitím modulární architektury, kde jednotlivé moduly aplikace běží jako samostatné procesy, které využívají prostředky meziprocesové komunikace. Aktualizace jednotlivých modulů je pak podmíněna jen kompatibilitou rozhraní a aktualizace části aplikace se pak nemusí projevit výpadkem služby[1]. Z těchto výhod čerpá vlastní implementace překryvné sítě v rámci této diplomové práce. Podstatné prvky implementace a jejich závislosti jsou zobrazeny na obrázku 7.1.

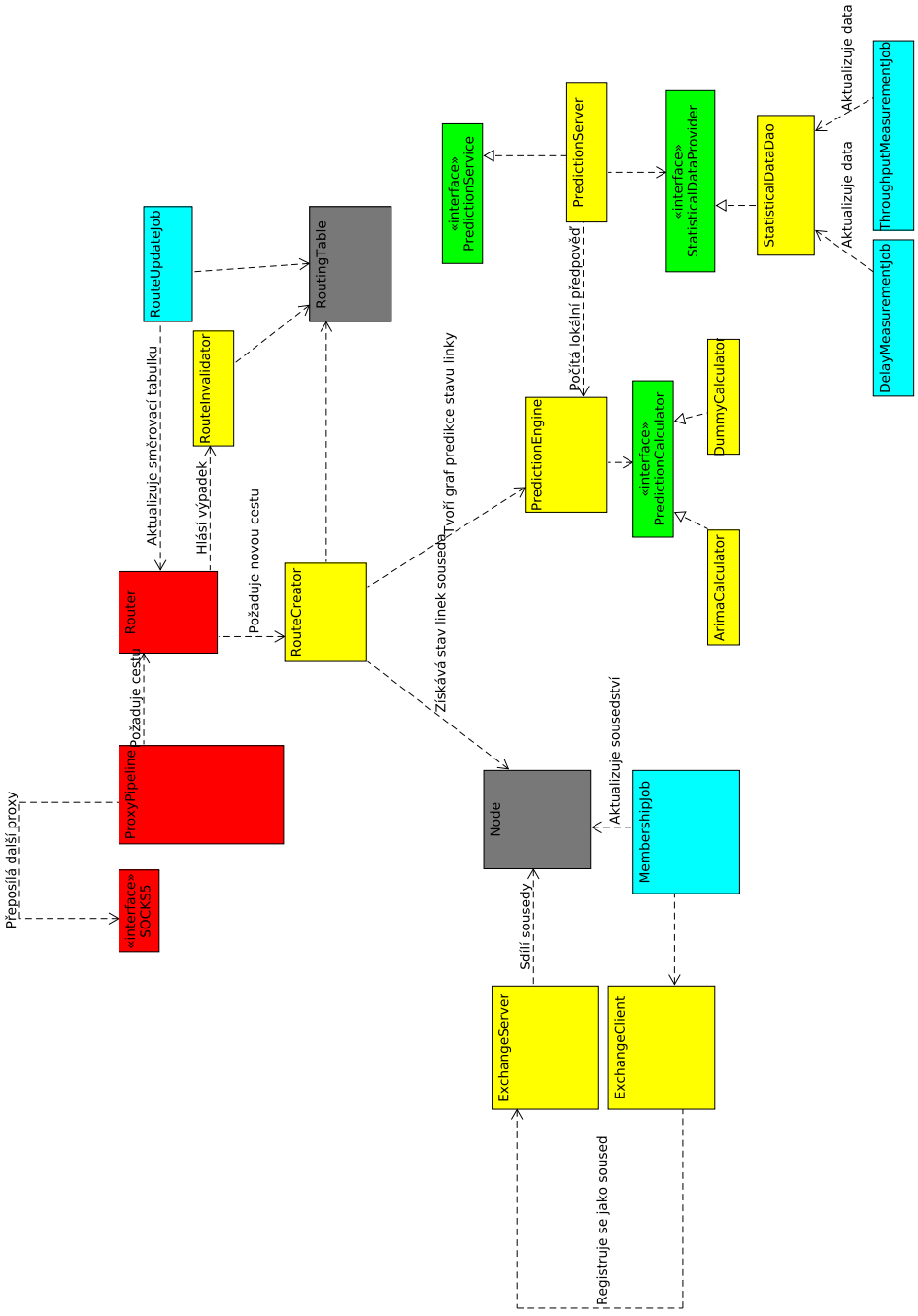
Červenou barvou je zvýrazněna implementace směrování v *C++*. Modře jsou podbarveny periodické úlohy obstarávající měření a aktualizaci směrovací tabulky. Zeleně rozhraní pro získání dat a predikcí. Ostatní funkční třídy jsou obarveny žlutě.

7.2 Volba programovacích jazyků a modulů

Implementace překryvné sítě je specifická tím, že přijímá požadavky od klientských aplikací, či prvků překryvné sítě samotných a dle *IP* adresy cíle je vyhledán ve směrovací tabulce nejvýhodnější další skok, nebo přímé spojení s cílovou adresou. Tvorba směrovací tabulky a udržování grafu sítě je na směrování nezávislá, a proto se skýtá možnost oddělit tyto dvě části na komponenty (každá může být složena z více modulů, restartovat nebo aktualizovat se dá jen komponenta jako celek).

Na každou komponentu/modul jsou kladeny jiné nároky, a proto bylo zvoleno více různých programovacích jazyků k implementaci. *C++* se stará o efektivní směrování a přeposílání dat mezi klienty a servery nebo dalšími uzly překryvné sítě. *Java* disponuje prostředky pro rychlý vývoj a testování algoritmů. V *Javě* je také k dispozici velké množství knihoven, které usnadňují implementaci složitějších programů. Mezi využité knihovny patří knihovna *Quartz2* implementující plánovač a knihovna *REngine* zpřístupňující rozhraní knihoven jazyka *R*. Jazyk *R* byl zvolen díky existující implementaci predikčních metod rozebraných v kapitole 4.4 a volné licenci. Alternativní implementace predikčních algoritmů napsané přímo v *Javě* jsou

[1] Není-li aktualizováno samotné jádro aplikace.



Obrázek 7.1: Struktura aplikace

zpoplatněny nemalou částkou.

7.2.1 Směrovací modul

První modul by měl poskytovat vysoký výkon při přeposílání dat a vyhledávání směrovacích informací ve směrovacích tabulkách. Z těchto požadavků vyplývá, že by interpretované jazyky nemusely podávat optimální výkon z důvodu garbage kolekce a automatické správy paměti. Vhodným jazykem pro tuto výkonnou část je tedy buď *C*, nebo *C++*. Byla-li by směrovací část implementována v C, musel by být brán zřetel na správnou synchronizaci sdílených datových struktur a především na správnou správu paměti. Pracnost takovéhoto řešení by převyšovala pracnost implementace v C++, které má již hotové knihovny pro paralelní zpracování dat a zároveň poskytuje šablony automatické správy paměti. Zároveň je možné využít principu *RAII* a napsat si obalovací třídy – wrappery, které spravují například sokety; ty jsou automaticky uzavřeny, dojde-li k ukončení spojení.

7.2.2 Modul tvořící směrovací tabulku

Druhý modul by měla obstarávat komunikaci mezi jednotlivými uzly ve smyslu registrace do sítě, správu známých uzlů a údržbu aktuální směrovací tabulky na základě požadavků ze směrovacího modulu. Tato část není výkonově kritická a může být implementována ve vyšších interpretovaných programovacích jazycích. Jelikož by práce měla zjistit vliv predikce na směrování v překryvné síti, byla jako programovací jazyk tohoto modulu zvolena *Java*, ke které existují knihovny propojující *Javu* a statistický programovací jazyk *R*, který má knihovny pro předpověď z časových řad.

Směrovací tabulka by měla být periodicky pročišťována, aby bylo zamezeno využívání směrovacích cest, které již neodpovídají předpovězenému stavu sítě. Směrovací si automaticky vyžádá tvorbu cesty, pokud neexistuje v jeho interní směrovací tabulce.

Další možností by byla automatická aktualizace zastarávajících cest ještě než jejich platnost vyprchá úplně. Snížila by se tím prvotní latence při navazování spojení na server, pro nějž neexistuje záznam ve směrovací tabulce. Narostl by tím ale externí měřící provoz, který zatěžuje servery nesouvisející s překryvnou sítí.

7.2.3 Modul měření

Třetí modul by měl zajišťovat měření stavů linek a naměřená data uchovávat ve sdíleném úložišti, ze kterého druhý modul bude čerpat data pro předpověď

následujícího stavu linky.

Měření by mělo zahrnovat dobu odezvy a rychlost přenosu mezi jednotlivými uzly překryvné sítě. Měření odezvy síť příliš nezatěžuje a je možné ho provádět častěji. Častější vzorkování je vhodné i pro adaptabilnější předpověď odezvy v následujícím časovém úseku.

Měření propustnosti naopak síť může výrazně zatížit a nemělo by být prováděno příliš často. Časové řady budou tedy zaznamenávány s větším intervalem vzorkování a predikce bude hrubšího charakteru než predikce zpoždění.

K ukládání časových řad by bylo možné využít klasický databázový systém, který by běžel jako proces, nicméně existuje kompaktnější pasivní alternativa v podobě *SQLite3*, které vytváří databázi jako soubor a operace nad tímto souborem zajišťují vlákna programu, který k databázi přistupuje. Databáze by měla sloužit jen k lokálnímu uchování časových řad. Centralizace by pro předpověď z lokálního pohledu uzlu neměla žádný smysl[2]. Respektive by smysl měla v případě, že by každý jeden uzel měl přepočítávat předpověď celé sítě. Implementace byla zvolena jako distribuovaná s ohledem na výpočetní výkon jednoho uzlu tak, aby se výpočetní zátěž rozložila do celé překryvné sítě rovnoměrně.

K implementaci byla opět zvolena *Java*, kde již existuje knihovna *Quartz2* implementující periodický plánovač akcí.

7.2.4 Komunikace mezi moduly

Vzhledem k tomu, že každý modul může být spuštěn na odlišném fyzickém stroji, je využito klasických soketů pro přenos řídících informací a datových struktur mezi jednotlivými moduly. Výpadek jednoho modulu by tak neměl ovlivnit běh ostatních modulů, které by měly být připraveny na nedostupnost služby směrovací tabulky či měření parametrů linek.

7.3 Klientské rozhraní

Klientské rozhraní slouží k přijímání příchozích spojení od aplikací nebo jiných uzlů překryvné sítě.

Jelikož požadavkem byla aplikační nezávislost, byl v první řadě vyloučen vlastní proprietární protokol zpřístupnění sdílenou knihovnou. Z hlediska existujících řešení a jednoduchosti integrace se nabízel protokol *SOCKS5*.

[2]Za předpokladu, že moduly nepoběží na různých fyzických strojích bez sdílení souborového systému.

Webové prohlížeče tento protokol podporují v základu a aplikace, které jeho podporou nedisponují, lze upravit přednačtením knihovny `libtsocks.so`, která přemapuje síťová systémová volání na sebe a zajistí svůj vlastní handshake se *SOCKS5* proxy serverem.

7.4 Obsluha příchozích požadavků

Obsluhu příchozích požadavků je možné řešit vlastními vlákny. Pro každý požadavek by bylo vytvořeno nové vlákno, které by se staralo o vyřízení požadavku a přeposílání dat. Tento přístup ovšem není příliš efektivní a způsoboval by tvorbu příliš mnoha krátkožijících vláken.

Druhou možností by byl thread pool, který by měl předem připravený počet vláken. Tím by byla odstraněna nevýhoda velké spotřeby systémových volání na klonování procesů, nicméně na odbavení velkého počtu krátkých požadavků a zároveň dlouhých přenosů ani toto není optimální řešení.

Pro část starající se o směrování byla zvolena knihovna *Intel TBB*, která poskytuje funkcionalitu pipeline, jež se hodí přímo pro tyto případy a Intel sám poskytuje ukázku, jak směrovač implementovat [29]. Výhodou využití `tbb::parallel_pipeline` je zabudovaný task stealing plánovač, který zajišťuje efektivní plánování jednotlivých fází pipeline.

7.5 Směrování a navazování spojení

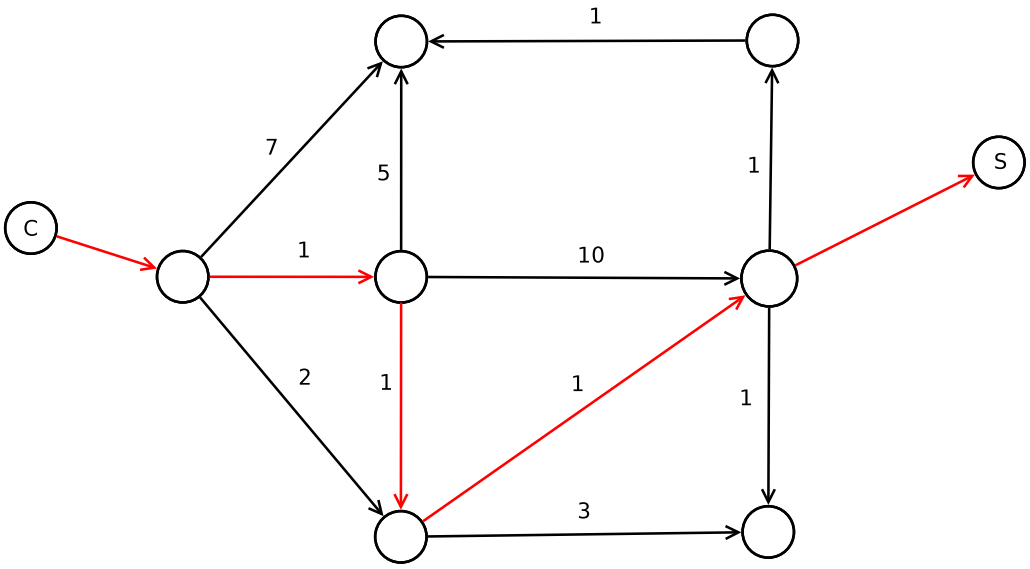
V případě navazování spojení lze jmenovat dva stěžejní přístupy, z nichž jeden je jednoduchý na implementaci a druhý poskytuje lepší možnosti škálování, za cenu vyšší komplikovanosti.

7.5.1 Statická rezervace cest

První způsob čerpá inspiraci z *ATM* prostředí, kde dochází k rezervaci virtuálních okruhů a cest. Klient musí nejprve vyslat požadavek s informací, s kým chce komunikovat. Směrovače sítě *ATM* pak postupně spolu komunikují a vyjednávají virtuální okruhy. Po úspěšném vytvoření virtuálního okruhu je možné zahájit komunikaci. Výhodou tohoto přístupu je relativní jednoduchost navazování spojení a zároveň alokace zdrojů, jež souvisí s Quality of Service a rezervací přenosového pásma. Zároveň není nutné tvořit nový komunikační protokol, který by nesl dodatečné informace o směrování a identifikaci toků, která by byla nutná pro správnou serializaci dat na koncovém uzlu. [30]

Nevýhodou je ovšem nutnost udržovat si stavové informace o navázaných virtuálních okruzích a jejich sjednaných parametrech. Tento problém v překryvné síti, implementované v rámci této diplomové práce, je potlačen tím, že se pouze naváží spoje na základě dobré víry v predikci stavu linky vygenerovanou některým predikčním algoritmem z kapitoly 4.4. Celý přenos pak využívá jednu konkrétní cestu a není možné změnit tok dat uvnitř překryvné sítě, což může mít za následek zhoršení kvality přenosu, pokud predikce předpoví nesprávně parametry linky v následujícím časovém intervalu či intervalech, ve kterých je přenos uskutečňován. Na obrázku 7.2 je zobrazena sjednaná cesta od klientské aplikace C po cílový server S. [7]

Limitací překryvné sítě je v tomto případě navazování *TCP* spojení, které používá stavové informace, především sekvenční číslo paketu. Kvůli tomuto omezení nelze dynamicky a efektivně měnit vytvořenou cestu. Zhorší-li se parametry některé z linek na sjednané cestě, jsou tím ovlivněny i všechny probíhající komunikace na těchto spojích.



Obrázek 7.2: Navázaná neměnná cesta mezi klientem C a serverem S

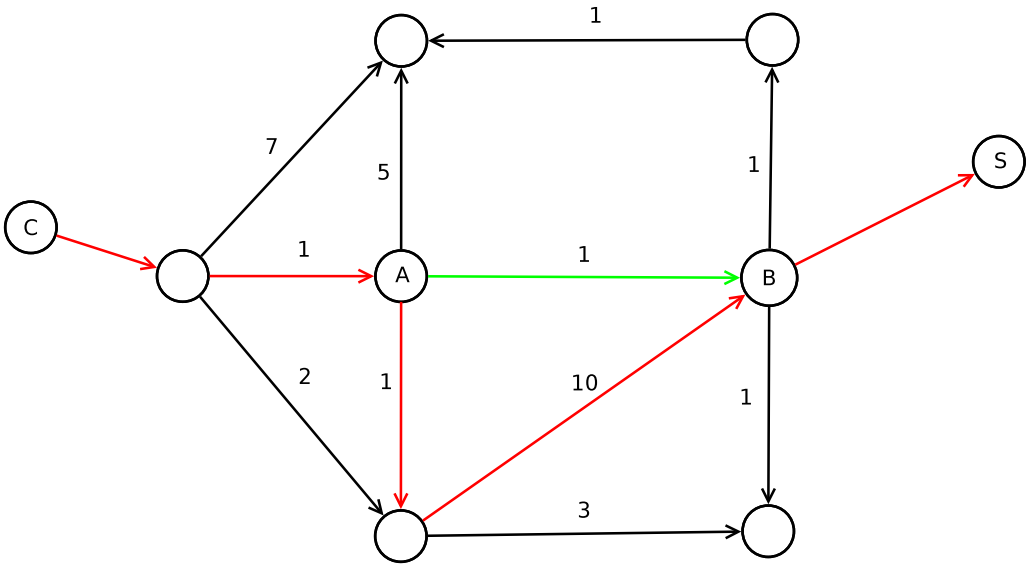
Výraznějším limitem je ale počet navazovaných spojení, které moderní prohlížeče udržují paralelně aktivní. Uzly uvnitř sítě tak v tomto případě musí zvládat stovky či tisíce konexí při větším počtu uživatelů.

Pro implementaci byl zvolen tento přístup navzdory nevýhodám kvůli jednodušší implementaci a cíli ověřit, zda predikce stavu linek bude mít pozitivní vliv na kvalitu přenosu. Škálování pro více uživatelů zde nebylo potřeba.

7.5.2 Dynamická změna cest

Alternativou ke statické rezervaci cest je dynamické směrování s využitím vlastního formátu přenášených dat, který by data uživatele obaloval do vlastních paketů, nesoucích informaci o požadavcích na kvalitu služeb, cílové adrese a identifikátoru toku a sekvenčním čísle. Ve své podstatě by se jednalo o obdobu *TCP* na aplikační vrstvě.

Na obrázku 7.3 je zachycena situace, kdy se změnily parametry linek tak, že místo aktuálně sjednané červené cesty se stala výhodnější zelená linka z uzlu A do uzlu B. Podmínkou je, že koncový uzel lepší cesty zůstane ekvivalentní s koncovým uzlem cesty předchozí. Uzel A by nejprve odeslal oznámení uzlu B, že má být připraven na příjem dat, náležících nějakému konkrétnímu spojení označeného identifikátorem. Uzel B by byl připraven na změnu zdroje dat a ukončení spojení po předchozí trase, pokud by nebyla využívána[3].



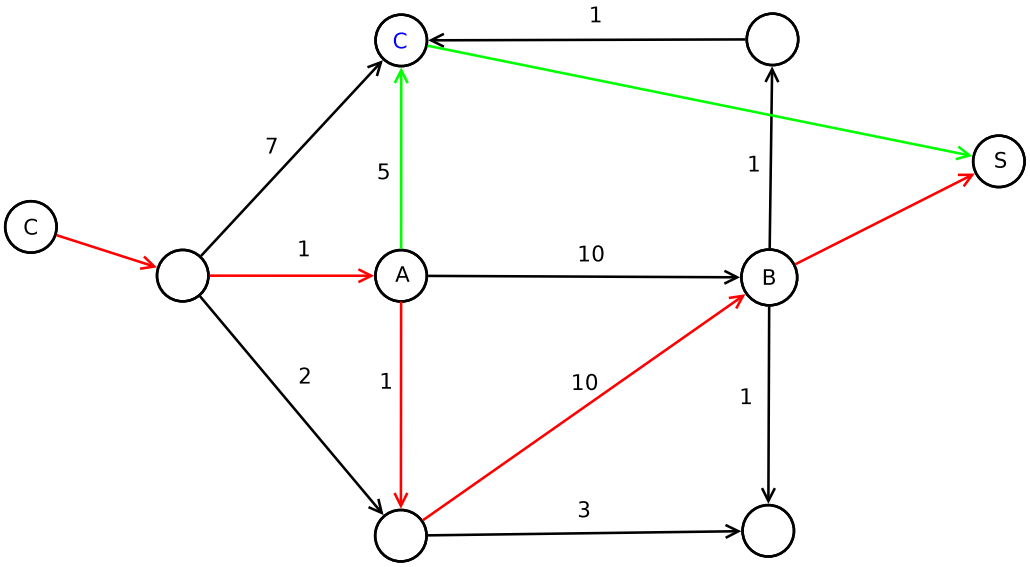
Obrázek 7.3: Možná změna navázené cesty z červené na zelenou bez výpadku

Při změně optimální cesty může ovšem nastat i situace, kdy nová lepší cesta nepovede přes stejný koncový uzel. Tato situace je prakticky neřešitelná bez ukončení a navázání spojení z nového koncového uzlu na server, což může způsobit měřitelné zvýšení odezvy, či zpomalení přenosu. Tato situace je zachycena na obrázku 7.4.

Další možnou komplikací u globálních služeb může být změna lokality uživatele z pohledu serveru. Cílová služba pak může generovat varovné

[3]Také by byla možná varianta, kdy se udržuje spojení mezi uzly konstantně, za cenu vyššího počtu paralelně aktivních spojení

zprávy uživateli o podezřelých přihlášeních na účet z jiných zemí, i když se stále jedná o jednoho a toho samého uživatele.



Obrázek 7.4: Nemožná změna cesty z červené na zelenou, kvůli nutnosti přerušit spojení

7.6 Přeposílání dat

Přeposílání dat je hlavním úkolem směrovacího modulu a mělo by být napsáno s ohledem na efektivitu. Hledání vhodné cesty ve směrovací tabulce je činnost, která trvá velmi krátkou dobu, kdežto přeposílání dat je dlouhodobého charakteru. V tomto případě by task stealing plánovač vykazoval horší výsledky, jelikož není určen ke správě blokujících spojení. Aktivní spojení čekající na data by blokovalo ostatní, dokud by samo nebylo ukončeno. Z tohoto důvodu je vhodnější použít threadpool, který má předpřipraven určitý počet vláken. Každé vlákno čeká nad frontou spojení a je-li nějaké spojení vloženo do fronty, čekající vlákno si ho převezme ke zpracování. Nedostačoval-li by počet aktivních vláken, je možné za běhu vytvářet nová, která budou zrušena po dokončení své práce, aby se šetřilo systémovými prostředky.

8 Implementace překryvné sítě

Tato kapitola bude věnována praktické implementaci překryvné sítě a detailnímu popisu komunikačních protokolů.

8.1 Architektura

Implementace překryvné sítě je rozdělena do tří modulů. Modul rychlého směrování je napsán v jazyce *C++14* a využívá jak rychlé pipeline pro zpracování příchozích spojení, tak poolu vláken pro přeposílání dat mezi klientem a serverem nebo dalším uzlem překryvné sítě.

Modul tvorby směrovací tabulky společně s měřícím modulem je napsán v jazyce Java a to především kvůli knihovnám, zpřístupňujícím rozhraní jazyka R a jeho funkcionalit knihovnou *REngine*, a také kvůli snažšímu ladění programu. K zajištění periodického spouštění měřících částí kódu s možností konfigurace byla zvolena knihovna *Quartz 2*, která poskytuje prostředky pro snadné spouštění jednotlivých úloh se specifickými požadavky na interval spouštění.

Část měřící latenci musela být napsána opět v *C++*, jelikož *Java* neumožňuje přístup na linkovou vrstvu, který je nutný pro tvorbu paketů protokolu *ICMP*, kterým je měřena latence.

Komunikace mezi těmito částmi je blíže popsána v sekci 8.2

8.2 Komunikace komponent

Jednotlivé komponenty spolu komunikují přes *TCP* sokety a *UDP* sokety na různých portech. *TCP* je využíváno pro přenos rozměrnějších datových struktur, protože délka zprávy posílané přes *UDP* je omezena na 65 KiB. Krátké zprávy jsou posílány rovnou protokolem *UDP*, aby byla omezena režie navazování spojení.

Vlastní nově vymyšlený komunikační protokol zahrnuje z důvodu minimalizace přenášených dat i číslo portu tak, aby typ zprávy byl dán přímo portem a nemusel tak být přenášen spolu se zprávou.

Význam portů je zdokumentován v tabulce 8.1. Formáty zpráv jsou popsány v sekcích 8.2.1 a 8.2.2.

Tabulka 8.1: Porty využívané překryvnou sítí

protokol	port	využití
TCP	1080	příchozí SOCK5 spojení
TCP	1082	aktualizace směrovací tabulky
TCP	1083	požadavky na tvorbu nové cesty
TCP	1084	oznámení o selhání spojení na dané cestě
UDP	1082	požadavek na měření latence

8.2.1 Formát zprávy aktualizace směrovací tabulky

Typ zprávy obsahující aktualizace tabulky je členěn na dvě části. První část o velikosti jeden bajt obsahuje identifikátor QoS politiky – tabulky, která je aktualizována. Po přijetí identifikátoru je očekáván seznam dvojic typu zdrojová adresa, cílová adresa. Zdrojová adresa může být typu *IPv4* nebo *IPv6*. Typ je rozlišován položkou `ss_family`, jejíž hodnoty jsou definovány v hlavičkovém souboru `socket.h`.

Graficky zobrazený formát je na výpisu 8.1.

Výpis 8.1: Formát zpráv aktualizace směrové tabulky

```
0     1           3                   7
+-----+-----+-----+----+----+----+----+----+----+----+----+--
| QoS |    AF     |  IPv4 SRC         |  IPv4 DST         | ..
+-----+-----+-----+----+----+----+----+----+----+----+----+--
```

8.2.2 Formát zprávy požadavku na měření externí latence

Požadavek měření latence obsahuje dvoubajtový identifikátor typu adresy následovaný adresou samotnou. Odpovědí je číslo typu `long` obsahující počet milisekund, který uplynul mezi odesláním zprávy *ICMP ECHO* a přijetím zprávy *ICMP REPLY*.

Struktura zpráv je zachycena na výpisu 8.2.

Výpis 8.2: Formát požadavku měření

```
0    2                6
+----+----+----+-----+
| AF | IPv4           |
+----+----+----+-----+

0    2                                                              18
+----+----+----+-----+----+----+----+----+----+----+----+----+----+---+
| AF | IPv6                                                          |
+----+----+----+-----+----+----+----+----+----+----+----+----+----+---+

0                              8
+----+----+----+----+----+----+
|    Timestamp                |
+----+----+----+----+----+----+
```

8.3 Získání adresy dalšího skoku

Modul zpracovávající příchozí spojení zjistí z přijaté hlavičky protokolu *SOCKS5*, jakou adresu si klientská aplikace žádá. Adresa je předána směrovací třídě, která má přístup do směrovacích tabulek. Dle cílového portu je zvolena politika *QoS* a vybrána tabulka, dle níž se bude směrovat. Není-li port spárován s konkrétní třídou kvality služeb, je použita výchozí hodnota – doba odezvy, jelikož měření odezvy je prováděno častěji než měření propustnosti a obsahuje tak aktuálnější data. Konfigurace klasifikace kvality služeb je do programu zakódována přímo pro testovací účely. Možným vylepšením by mohlo být čtení konfigurace ze souboru.

Je-li cesta nalezena, pak je navrácena zpět do předchozího modulu. V opačném případě je zaslán požadavek na vytvoření cesty do modulu spravujícímu směrovací tabulky. Ten využije buďto predikci stavů linek z lokální cache, nebo si ji vyžádá od okolních uzlů překryvné sítě. Dle predikce spočte nejvýhodnější trasu a zanese záznam dalšího skoku do směrovací tabulky.

Tato tabulka je následně aktualizována ve směrovací třídě, která pak může využít nově získanou cestu.

Stane-li se, že modul spravující směrovací tabulku je nedostupný, či jiný druh výpadku sítě, je jako další skok vrácena adresa serveru, o který si zažádal klient. Tento případ je chápán jako přímé spojení bez využití *SOCKS5* hlaviček.

8.4 Přeposílání dat

Přeposílací modul obsahuje vstupní frontu, která je plněna modulem přijímajícím a navazujícím spojení mezi uzly překryvné sítě a cílovými servery. Každá položka ve frontě představuje úlohu přeposílání dat v jednom směru. Každý komunikační kanál tedy obsluhují dvě vlákna, aby byla zajištěna propustnost dat v obou směrech bez nutnosti využívat neblokující sokety.

Aby toto fungovalo, je zdrojový a cílový soket zduplikován tak, aby bylo možné vytvořit dvě úlohy pro zařazení do fronty. Jedna úloha je dána dvojicí [zdrojový soket, cílový soket] a druhá úloha má sokety v prohozeném pořadí: [cílový soket, zdrojový soket].

Vlákna jsou připravena pro přeposílání dat z jednoho soketu do druhého. Jejich počet je zdola omezen počtem jader procesoru a další jsou tvořena na požádání.

8.5 Tvorba a údržba směrovací tabulky

Směrovací tabulka je výhradně tvořena na vyžádání modulu směrování. Při příjmu požadavku vytvoření cesty je zkontrolována cache předpovědi stavu linek. Je-li v cache aktuální předpověď, pak se využijí její data k tvorbě grafu sítě. Jinak je vzat seznam sousedních uzlů a přes *RMI* odeslán požadavek na spočtení jejich lokálních předpovědí. Vzdálené uzly zkontrolují svou cache, aby šetřily výpočetním výkonem. Zpět odešlou buďto předpočítanou predikci z cache, nebo vytvoří novou.

Dotazující uzel sestrojí graf sítě s ohodnocenými hranami a nad ním spustí Dijkstrův algoritmus. Nad přepočítaným grafem je následně spuštěno vyhledávání předchůdců od cílového serveru až k aktuálnímu uzlu. Předposlední uzel, který je takto nalezen je použit jako další skok, jenž je pak zanesen do směrovací tabulky.

Směrovací tabulky by také měla reflektovat změny stavu sítě v čase a proto ji spravuje periodická úloha, která porovnává čas vložení cesty s aktuálním časem. Je-li cesta zastaralá, pak ji úloha z tabulky vymaže. Směrovací modul si o ni v případě potřeby znovu zažádá.

8.6 Směrování

Směrování je prováděno na základě dvou směrovacích tabulek, kde každá náleží jinému požadavku na kvalitu přenosu. Směrovací tabulka je vybrána dle cílového portu. Dle cílové adresy je daná tabulka prohledávána na existenci

dalšího skoku. V případě nálezu je vrácena adresa dalšího skoku. Jinak je vrácena původní cílová adresa.

První směrovací tabulka je tvořena kombinovaně na základě doby odezvy a spolehlivosti linky. Předpokladem je, že nespolehlivá linka bude mít automaticky vyšší dobu odezvy a není tedy nutné vést spolehlivost v jiné tabulce.

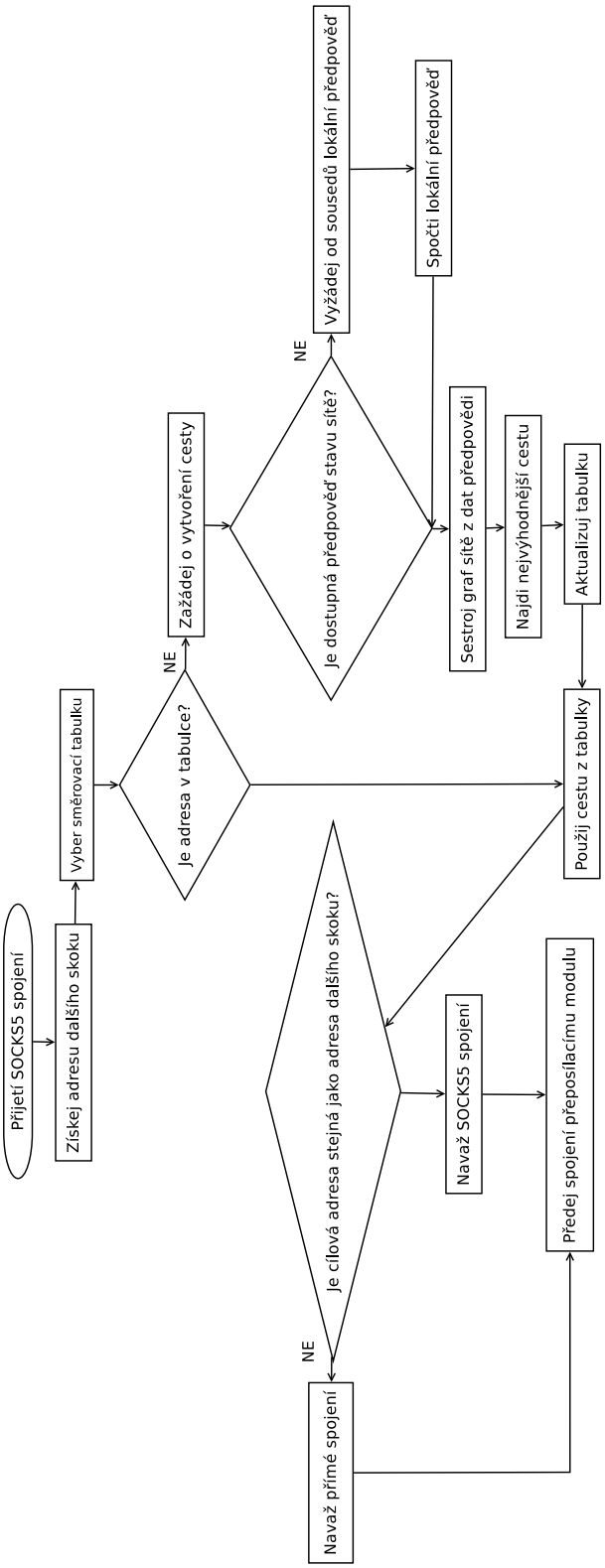
Druhá směrovací tabulka využívá metriku naměřených propustností stejným vzorcem, jenž je využit u protokolu *OSPF* 3.1.2.

8.7 Měření doby odezvy a dostupnosti

Měření doby odezvy je prováděno periodickou úlohou, která je spouštěna každou minutu. Úloha získá seznam sousedních uzlů a postupně odesílá adresu každého uzlu do modulu spravujícího *ICMP* soket. Ten pošle krátkou zprávu *ICMP ECHO* s generovaným id. Id zařadí do seznamu požadavků spolu se zdrojovou adresou žadatele a časem přijetí požadavku. Vlákno dále přijímá nové požadavky na měření. Jiné vlákno čeká na zprávy typu *ICMP REPLY* a dle id vyhledává spojení na žadatele měření. Je-li žadatel nalezen, pak je požadavek odstraněn z fronty a odeslán naměřený čas zpět žadateli.

Další vlákno maže požadavky, na něž nepřišla odpověď.

Předpokládá se, že doba odezvy bude přibližně polovinou RTT, které je součtem délky trvání cesty od zdroje k cíli a zpět. Reálně ale může nastat situace, kdy toto neplatí.



Obrázek 8.1: Tok obsluhy spojení

9 Simulace

K naměření vlivu predikce na kvalitu přenosu byla naprogramována simulace s využitím open source nástrojů obsažených v běžné Linuxové distribuci. Specializované síťové simulátory se mi nepodařilo přizpůsobit potřebám překryvné sítě, který má své výhody a nevýhody.

Původně měla být překryvná síť otestována některým z dostupných síťových simulátoru *Cloonix*, *CORE*, *IMUNES* a další[1]. Nicméně tyto simulátory jsou spíše určeny pro testování síťové vrstvy a směrování na ní. Překryvnou síť se mi v nich nepodařilo zprovoznit, a proto jsem napsal vlastní simulační „framework“ využívající bash, python a systémové prostředky GNU/Linux.

Simulace musí být především opakovatelná. Opakovatelnost z hlediska spouštění zajišťuje inicializační skript `start-simulation.sh`, který spustí *Docker* kontejnery překryvné sítě skriptem `start-overlay-nodes.sh`. Za nimi následují kontejnery měření spouštěné skriptem `start-measurement.sh` a jako poslední je spuštěn skript `simulation.py`, jenž načítá časové řady a postupně aplikuje parametry linek na virtuální rozhraní kontejnerů.

Výhodou je absolutní moc nad virtuálními síťovými kartami a vysoká flexibilita. Nevýhodnou ovšem je složitější příprava dat pro konfiguraci časových řad jednotlivých linek.

9.1 Cíle měření

Před zahájením simulace je vhodné definovat cíle měření a očekávané výsledky. Vzhledem k tomu, že předpověď pracuje nad časovými řadami, které nenesou informaci o kvalitě služby, bude prováděno jen měření vlivu predikce na latenci. Latence je přesněji měřitelná a naměřená data jsou dostatečně reprezentativní.

Měření bude provedeno teoretickým výpočtem přímo z časových řad i praktickým měřením v překryvné síti, kde bude zahrnut i vliv zpoždění tvorby cest a navazování nového spojení.

Výsledky pak lze aplikovat i na časové řady jiných kvalitativních metrik.

[1] Další síťove simulátory k nalezení na `http://goo.gl/JKyUh3`

9.2 Programové prostředky

Ke spuštění simulace je nutné mít na počítači nainstalován operační systém Linux s interpretrem skriptů bash; Python 3, ve kterém jsou napsány transformační skripty reálných dat na simulační časové řady a simulace samotná. V neposlední řadě je také zapotřebí mít jádro 3.19 a Docker ve verzi 1.11. Starší verze Dockeru přidělují *IP* adresy kontejnerům jiným způsobem, se kterým simulace není kompatibilní.

Tabulka 9.1: Software použitý pro simulaci

sw	použití
Python3	transformace dat, simulace
bash	řízení transformace dat, spouštění simulace
Docker 1.11	virtualizace uzlů sítě
gnuplot	vizualizace dat
Linux 3.19	jádro operačního systému
tc	řízení virtuálních rozhraní
R 3.0.2	vypočetní prostředí
R forecast 6.2	předpověď časových řad

9.2.1 Docker

Docker slouží jako virtualizační nástroj na aplikační úrovni. Každá aplikace má svůj virtuální pracovní prostor a přístup k virtuálním síťovým rozhraním.

Virtuální síťová rozhraní lze ovládat z hostovaného systému tak, jako by se jednalo o skutečná fyzická rozhraní. Této vlastnosti je využíváno k úpravě vlastností jednotlivých linek dle naměřených reálních charakteristik.

9.2.2 tc

Program tc byl využit k nastavení parametrů rozhraní dle časové řady čtené skriptem v pythonu. Při nastavování parametrů rozhraní je využito emulační vrstvy *netem* a možnosti ji nastavit jako frontu pro příchozí i odchozí rámce.

9.2.3 Python a bash

Pro přípravu dat a spuštění simulace jsou využity skriptovací jazyky *python* a *bash*, které se vzájemně doplňují funkcionalitou.

Python poskytuje snadnou práci s kolekcemi a čtením souborů, kdežto bash je využíván pro bezestavové transformace dat a koordinaci volání jiných skriptů.

9.2.4 gnuplot

Gnuplot byl využit v kombinaci s bashem pro generování grafů z datových souborů.

9.3 Hardwarové prostředky

Tabulka 9.2: Hardware použitý pro simulaci

komponenta	název
CPU	Intel i5 4x3350@3.1GHz
RAM	Corsair 16 GiB@1600MHz
HDD	SSD Corsair 120 GB

Hardware, na kterém byla spouštěna simulace, se skládal z konfigurace uvedené v tabulce 9.2.

9.4 Zdrojová data

Zdrojová data sloužící pro generování simulačních časových řad byla získána ze síťových prvků zatížených reálným provozem[2].

Zdrojová data jsou uložena v proprietárním formátu, ze kterého jsou generovány statistické grafy (`https://goo.gl/cyVtvn`). Reverzním inženýrstvím bylo zjištěno, že data v jednom konkrétním souboru jsou složena ze čtyř časových řad tří granularit. Každá řada obsahuje průměrné hodnoty za jinou délku časového úseku – 5 minutové průměry, 30 minutové průměry a 2 hodinové průměry. V každé časové řadě byly minimálně tři periody, což u denních grafů znamená, že jeden datový soubor obsahuje tři dny záznamu. Jelikož byly datové soubory aktualizovány každý den o půlnoci, obsahuje každý soubor duplicitní data s jinými třemi soubory.

Získaná data musela být předzpracována v několika fázích skriptem `merge-real-data.sh`, který vybere jednu časovou řadu a tu ze zdrojových souborů sloučí do vyčištěných souboru v adresáři `/tmp/merged-data`.

[2]Poděkování patří VOŠ a SPŠE Plzeň za poskytnutí logů z přepínačů.

1. Rozdělení časových řad do vlastních souborů

2. Odstranění duplicitních záznamů

Po provedení čištění dat bylo možné provést vizuální výběr zajímavých časových průběhů zachyceného provozu. K vytvoření grafů slouží skript `graph-real-data.sh`, který předpokládá existenci vyčištěných zdrojových dat v adresáři `/tmp/merged-data`. Časové průběhy zpoždění vybraných linek jsou zobrazeny na grafu 9.1.

9.4.1 Formát transformovaných dat

Transformovaná data jsou uložena ve složce `/tmp` s názvy souborů rozlišujícími ke kterému zařízení daná časová řada patří a granularitu naměřených dat. Například název „192.168.0.1_5min“ obsahuje časovou řadu náležící zařízení `192.168.0.1` a naměřená data jsou z pětiminutových intervalů.

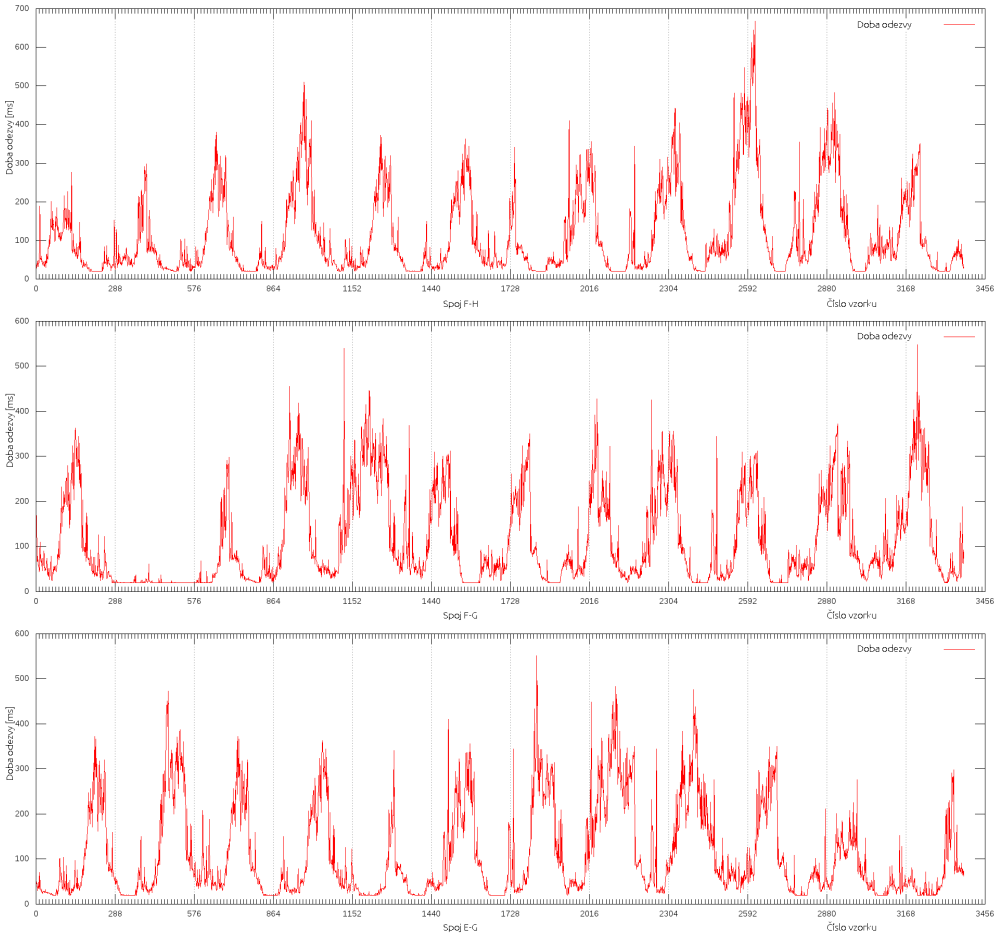
9.4.2 Tvorba dat pro simulaci

Každá linka by měla mít svou vlastní charakteristiku. Jelikož byla k dispozici reálná data, byla využita jedna časová řada z nejzatíženějšího zařízení. Tato řada byla využita jako základ pro generování parametrů jednotlivých linek v obou směrech skriptem `generate-link-states.py`, kterému je předložena libovolná časová řada z výstupu předchozí fáze transformace dat. Konvence tvorby názvu souborů je dána tak, aby bylo možné snadno pohledem spárovat soubor s konkrétním kontejnerem a také aby byly časové řady programově zpracovatelné. Název souboru je tedy složen jako dvojice *IP* adres s příponou `.ts`[3], např.: `172.17.0.3-172.17.0.7.ts`.

Pro každý směr byla řada rotována o náhodný počet minut tak, aby se simuloval časový posuv v různých částech planety. Každá linka má jednu časovou řadu pro každý směr. Výsledná charakteristika linky je dána součtem obou směrů a právě tato charakteristika je přímo měřitelná z pohledu překryvné sítě. Nad jednotlivými řadami byla spočtena predikce skriptem `predict-all-links.sh` využívající skriptu `time-serie.R` obsahujícího implementaci predikčního algoritmu. Zdrojovými soubory jsou soubory s koncovkou `.ts` a výstupní soubory predikce mají navíc koncovku `.prediction`.

Po spočtení predikce byl stejným způsobem jako v předchozím odstavci proveden součet určený k simulaci jedné linky obsouměrně. Tím byly získány predikované časové řady jednotlivých linek, kterými by se měla řídit

[3]Dle účelu využití časové řady může být typ upřesněn před koncovkou `.ts` – 172.17.0.3-172.17.0.7.delay.ts

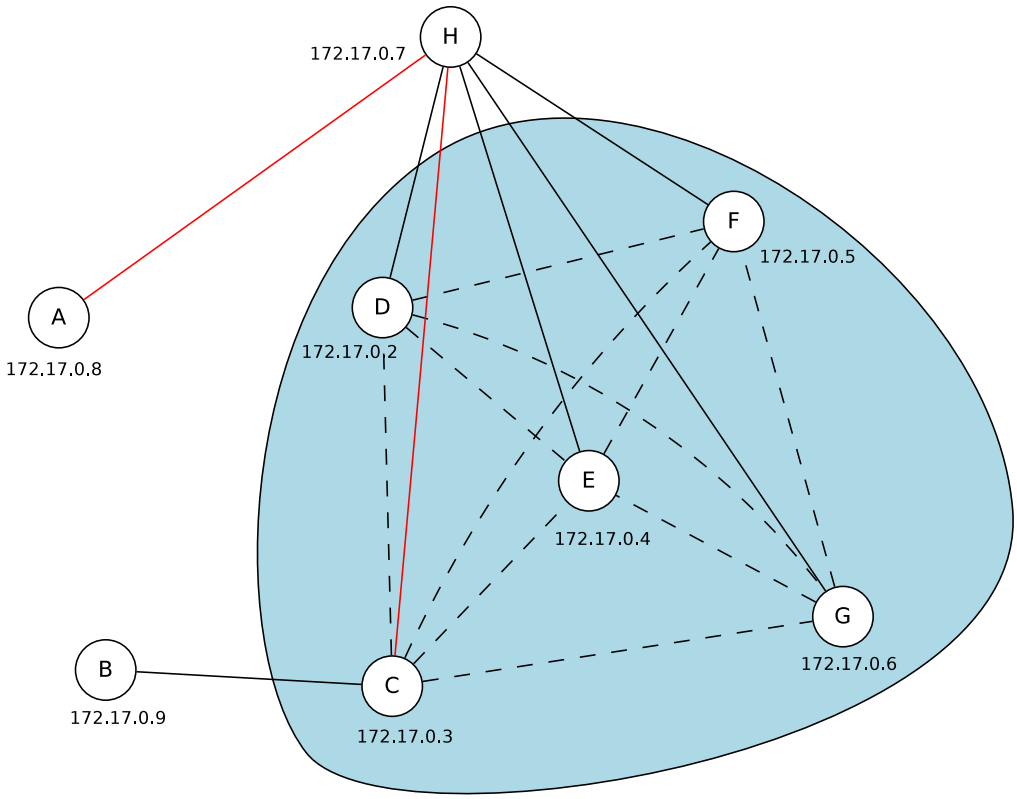


Obrázek 9.1: Zpoždění na linkách

překryvná síť při tvorbě směrovacích tabulek. Soubory součtů jsou rozpoznatelné koncovkou `.sum`.

9.5 Topologie simulace

V grafu 9.2 je zobrazena topologie sítě při simulaci. Červeně označené linky mají stejné parametry a to kvůli koncepci kontejnerů v Dockeru. Koncové uzly **A** a **B** jsou virtuálně rozdílné, ale reálně jde o jeden a týž uzel, který má jednu linku do serverového uzlu **H**, jenž slouží jako cílový server. Díky této koncepci může být porovnána efektivita směrování v překryvné síti proti přímému spojení. Vybrané cesty jsou zobrazeny na grafu 9.5.



Obrázek 9.2: Topologie simulace

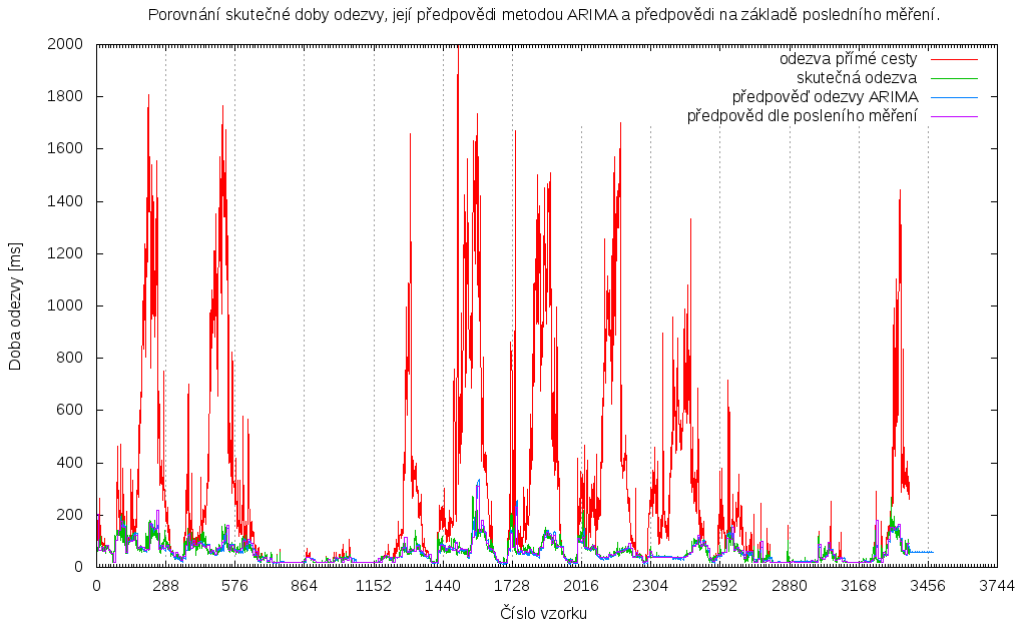
9.6 Výběr cest

Teoretickým výpočtem bylo ověřeno, že předpovězený stav linek a směrování na základě této předpovědi i směrování na základě posledního známého stavu poskytuje výrazné snížení doby odezvy. Na obrázku 9.4 je vidět, že přímý spoj (zobrazen zeleně) dosahuje ve špičkách latence až 1600 milisekund. Spoje v překryvné síti tuto maximální hodnotu snižují na necelých 400 milisekund. V průměru je doba odezvy přibližně 60 milisekund v překryvné síti a 270 ms bez využití překryvné sítě.

9.7 Porovnání volby cest

K porovnání volby cest byl implementován upravený Floyd-Warshallův algoritmus. Úprava spočívala v přidání třetí (časové), dimenze. Pseudokód tohoto algoritmu je na výpisu 9.1.

Implementace tohoto algoritmu je využita ve skriptu `sumarize-routes.py`, který do souboru zapíše časovou řadu obsahující nejkratší časovou odezvu mezi dvěma uzly. Vstupem jsou součty časových řad jednotlivých linek



Obrázek 9.3: Porovnání přímého spoje s nejvýhodnější cestou v překryvné síti a její předpovědí

s příponami `.ts.sum` a `.ts.prediction.sum`. Výstupní soubory mají prefix `route-` následovaným čísly[4] uzlů `3-7`,typem časové řady a příponou `.ts`. Například soubor `route-3-7.delay.prediction.ts` obsahuje nejkratší cestu v čase dle doby odezvy založené na předpovědi.

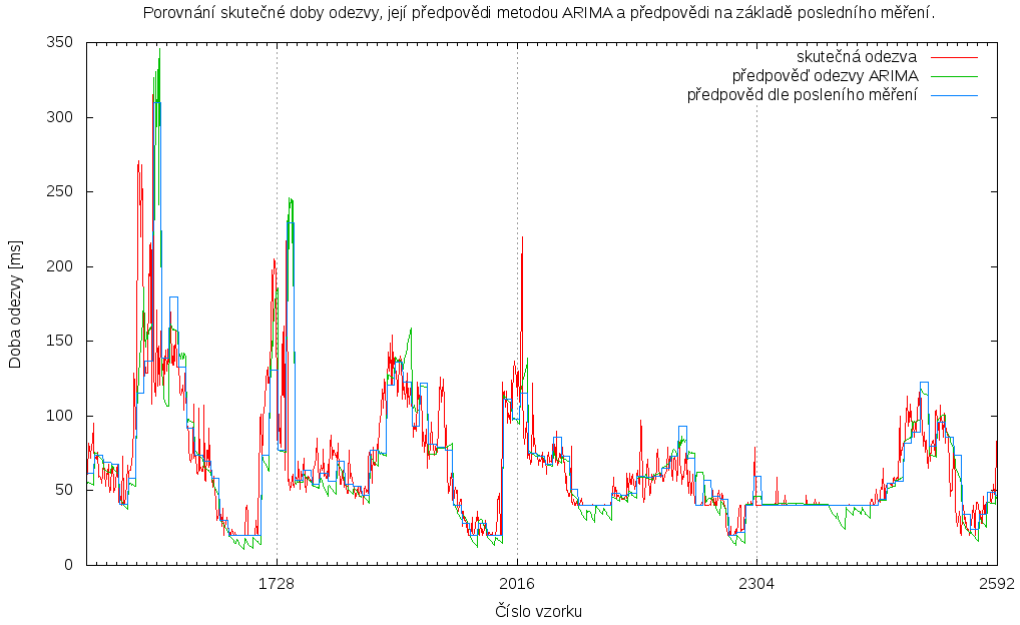
Výpis 9.1: Třírozměrný Floyd-Warshallův algoritmus

```
DistanceMatrix D [|V|][|V|][|T|];
for (int t = 0; t < |T|; ++t) {
  for (int k = 0; k < |V|; ++k) {
    for (int i = 0; i < |V|; ++i) {
      for (int j = 0; j < |V|; ++j) {
        D [i][j][t] = min(D[i][j][t] , D[i][k][t] + D [k][j][t]);
      }
    }
  }
}
```

Na obrázku 9.4 jsou zobrazeny průběhy odezvy tak, jak by je počítal implementovaný směrovací algoritmus. Zelený graf představuje předpověď následujícího stavu metodou *ARIMA*, modrý graf využívá směrování na základě poslední naměřené hodnoty a červený graf je skutečná odezva na trase od klienta k serveru.

[4]Číslem uzlu je poslední číslice *IP* adresy v kanonickém tvaru.

Je možné si všimnout určitých výkyvů zeleného grafu mezi vzorky 2304 a 2592, kdy predikční algoritmus odhaduje, že by mohlo nastat uvolnění linky, i když se tomu ve skutečnosti nestalo. Algoritmus směrování na základě posledního stavu odhadoval stav následující přesněji.

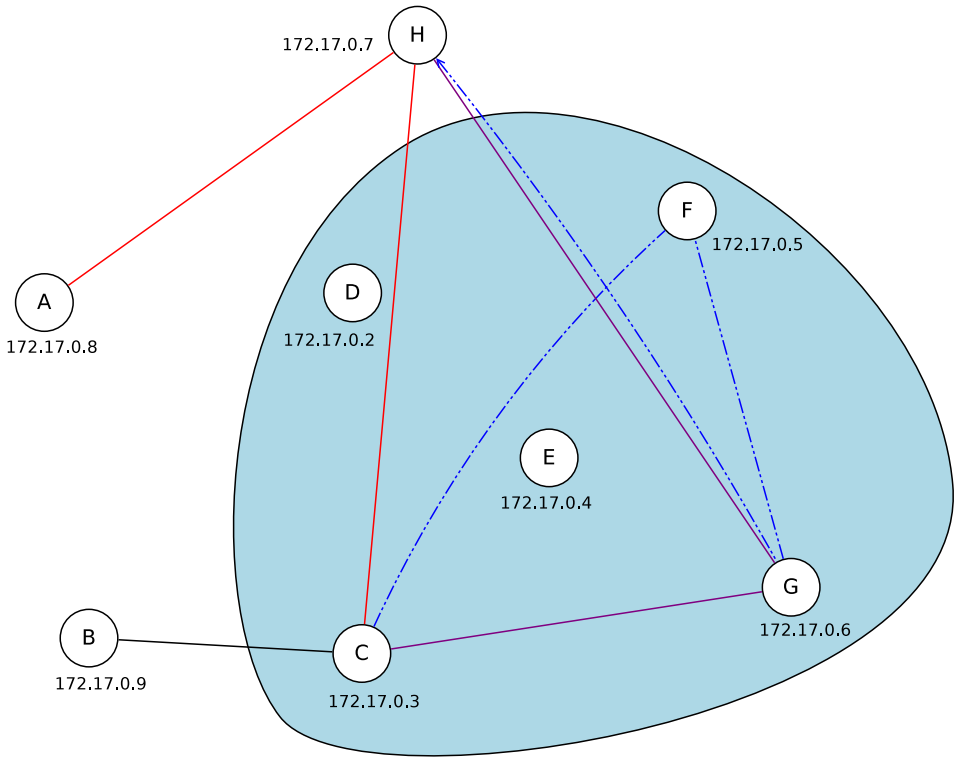


Obrázek 9.4: Porovnání standardního směrování se směrováním s predikcí a skutečnou odezvou

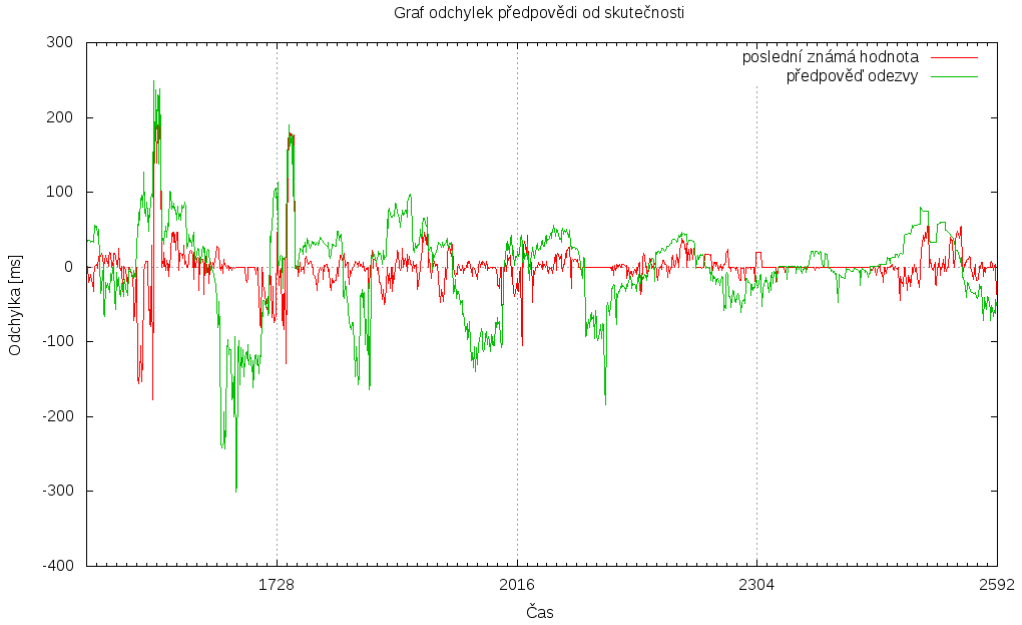
V grafu 9.6 je zobrazena odchylka směrovacího algoritmu *ARIMA* (zeleně) a standardního směrování (červeně). Na první pohled je zcela zřejmé, že predikce odhaduje stav linky s větším rozptylem a tím pádem méně přesně, než směrování na základě posledního naměřeného stavu.

Nicméně toto ovšem nemusí nutně znamenat, že by směrování využívající predikci muselo nutně podávat horší výsledky. Naměřená data lze interpretovat tak, že sice odhady nejsou přesné, ale pro směrování není podstatná přesnost odhadu, ale zachycení trendu v následujících minutách. Předpověď je při směrování využita jako abstraktní číslo, které je porovnáváno relativně mezi jednotlivými linkami. Dá se očekávat, že předpověď trendu odpovídá skutečné tendenci, a tak výsledné směrování bude výhodnější než směrování na základě posledního stavu.

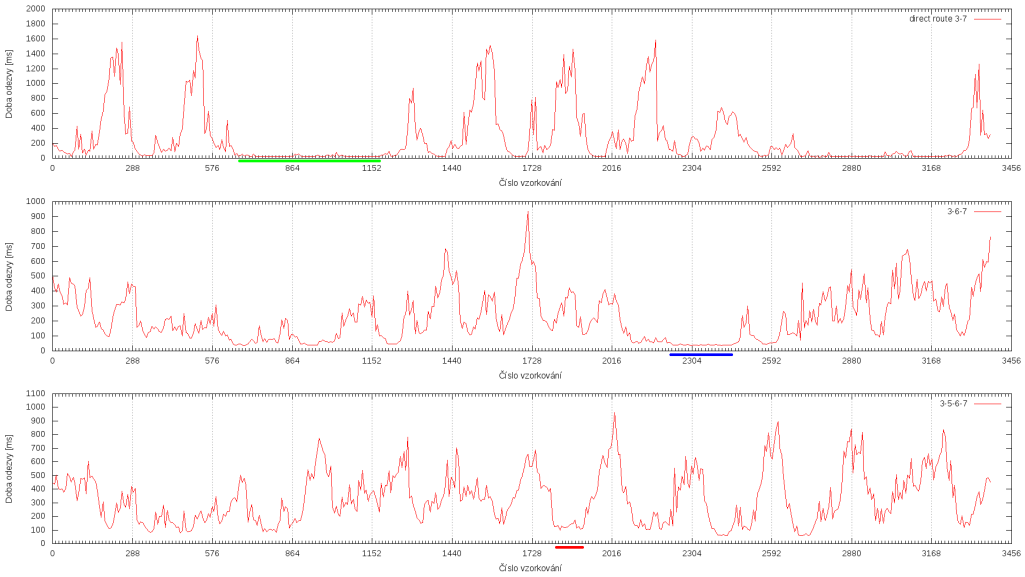
Na obrázku 9.7 je možné zpozorovat, že přímá cesta (horní graf) nemusí být vždy nutně výhodnější, než cesta přes několik uzlů překryvné sítě. Viditelné intervaly, kdy je využita daná cesta, jsou podtrženy barevným pruhem.



Obrázek 9.5: Vybrané cesty vyobrazené v grafu 9.7



Obrázek 9.6: Porovnání odchylky standardního směrování bez předpovědi s odchylkou směrování metodou $ARIMA$



Obrázek 9.7: Porovnání doby odezvy tří cest (každá je alespoň jednou za simulaci zvolena)

10 Výsledky měření

V předchozí kapitole byly spočteny teoretické výsledky. V této kapitole budou rozebrány výsledky měření reálné simulace. Graf souvisejících naměřených latencí je zobrazen na obrázku 10.1.

10.1 Standardní ARIMA

Při měření standardní metody $ARIMA$ bylo zjištěno, že nepodává očekávané výsledky. Graf doby odezvy sice dosahuje nižší amplitudy, ale velikost latence je stále vyšší než při směrování dle posledního stavu. Toto chování si lze vysvětlit jedině tím, že teoretická simulace využívá „hladkých“ dat, kdežto v praktickém měření je aplikována náhodná odchylka k nastavenému zpoždění linky. Naměřená data jsou pak rozkmitaná a to může výrazným způsobem ovlivnit predikci.

10.2 ARIMA kombinující dvě řady

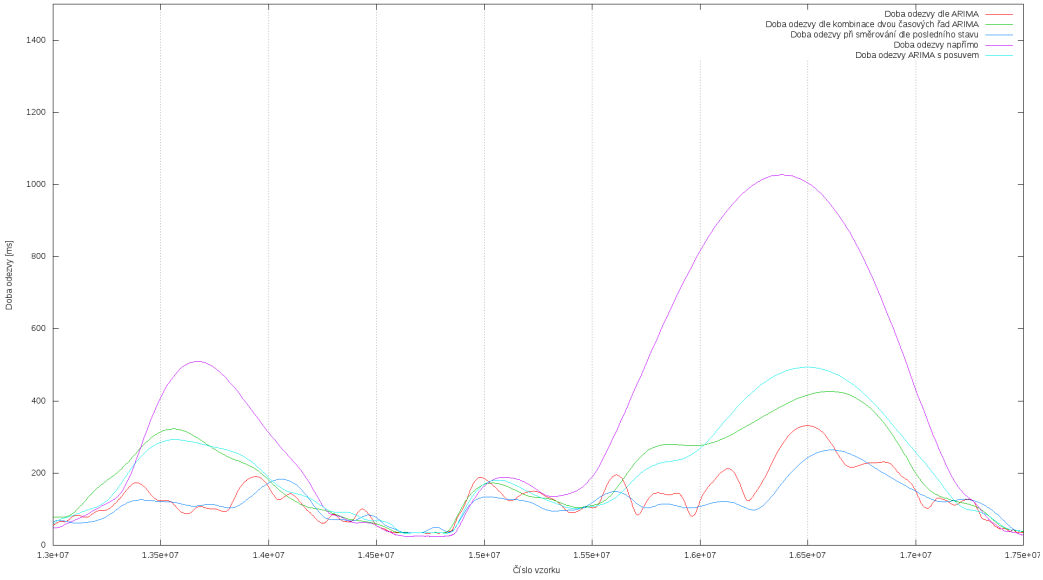
Tato metoda podává stejně špatné výsledky jako standardní $ARIMA$. Vliv kombinované metriky neměl kýžený efekt a graf zpoždění změnil svůj tvar a to směrem k vyšším latencím.

10.3 ARIMA kombinující dvě řady s posuvem

Dvě řady kombinované váženým průměrem s posuvem jsou kvalitativně prakticky stejné jako kombinace dvou řad bez posuvu. Střední doba odezvy obou metod v simulovaném případě je 71.4820 ms, což je více než 64.1173 ms u směrování dle předchozího stavu.

10.4 Statistické srovnání predikčních metod

Z grafů je patrné, že predikce metodou $ARIMA$ nepodává očekávané výsledky. Vyjádřeno čísly je využitím predikce zvýšen rozptyl latencí i její velikost. Jedině u metody bez predikce byla latence snížena za cenu zvýšení rozptylu, způsobeného navazováním spojení přes několuk uzlů překryvné sítě.



Obrázek 10.1: Porovnání vlivu predikce na latenci v reálné simulaci

Predikční metody by dle naměřených dat mohly být využívány pro směrování v překryvných sítí s cílem snížení kvality služeb. Data naměřená v praktické simulaci ale mohla být ovlivněna časovou kontrakcí simulace, kdy několik dní bylo nasimulováno během cca 10 hodin. Aplikace byla sice upravena pro simulované časové měřítko, ale vlivem nezanedbatelnosti doby výpočtu predikce na velmi krátký časový interval mohlo dojít ke znepřesnění výsledků.

10.4.1 Sloupce v tabulce

V tabulkách 10.1 a 10.2 jsou použity zkrácené názvy sloupců a jejich význam bude popsán v tomto odstavci.

První sloupec se zkratkou *AVG* obsahuje průměrnou dobu odezvy v milisekundách. Následují sloupce s maximální a minimální dobou odezvy. Poslední sloupec obsahuje směrodatnou odchylku odezvy, která by pro ideálně kvalitní linku měla být nulová.

10.4.2 Přímé srování naměřených hodnot

V tabulce 10.1 jsou uvedeny statisticky spočtené hodnoty měření.

Je zajímavé si všimnout, že ačkoli z grafu 10.1 je zřetelně vidět, že odezva na přímé cestě dosahuje nejvyšších hodnot, tak průměrná hodnota odezvy je nižší než u predikčních metod. Toto lze vysvětlit tím, že vzorkovací program

Tabulka 10.1: Naměřená doby odezvy v milisekundách

Měřená metoda	AVG	MAX	MIN	STDEV
Přímo	67.71	1034	26	158.23
Bez predikce	64.12	250	34	201.95
ARIMA	71.00	384	35	206.46
ARIMA2+posuv	71.48	476	36	243.42
ARIMA2	71.64	452	35	244.22

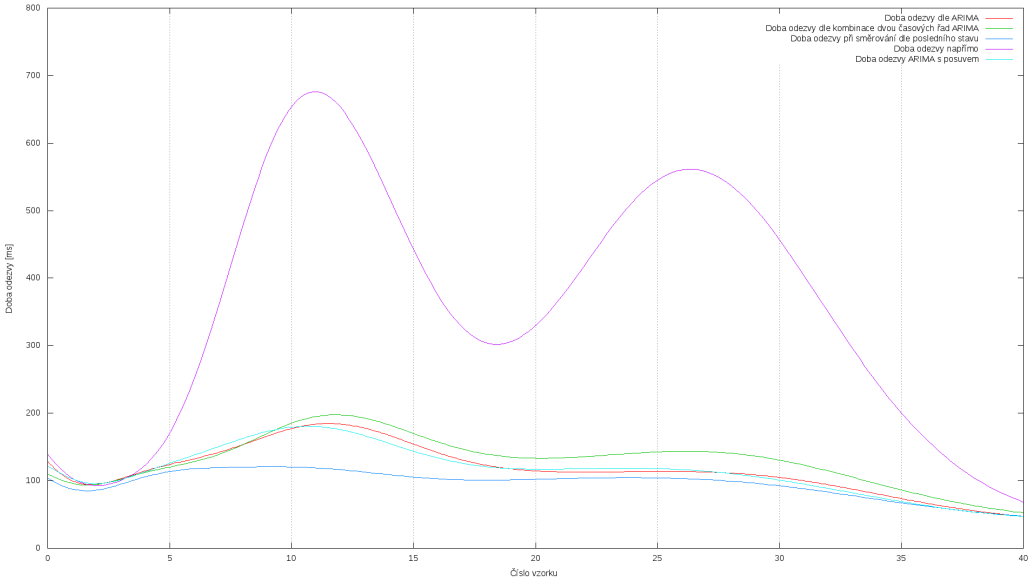
tvořil v časových úsecích s krátkou dobou odezvy více vzorků, které svým počtem převážily výrazně nad vzorky z úseků s velkou dobou odezvy.

Kdežto v překryvné síti je průměrná doba odezvy nižší, ale navazování spojení trvá delší dobu a kvůli tomu jsou horší statistické údaje než u přímého spojení.

10.4.3 Srovnání dat po normalizaci

Data bylo možné normalizovat tak, aby byla potlačena nestejnoměrnost frekvence vzorkování na různě rychlých linkách.

Skriptem `agregate-simulation-results.py` jsou zprůměrovány naměřené hodnoty odezvy za pevný časový úsek.



Obrázek 10.2: Normalizované porovnání vlivu predikce na latenci v reálné simulaci

Na grafu 10.2 je zobrazen normalizovaný průběh odezvy. Z normalizovaných dat pak vzešla opravená tabulka 10.2. Zde je již podle očekávání přímá

cesta horší, než cesty v překryvné síti, nicméně predikované cesty jsou stále méně výhodné než cesty tvořené bez predikce.

Tabulka 10.2: Normalizované doby odezvy v milisekundách

Měřená metoda	AVG	MAX	MIN	STDEV
Přímo	365.38	1849	26.35	474.43
Bez predikce	97.50	178	34.23	39.18
ARIMA	118.38	331	36.18	70.30
ARIMA2+posuv	117.09	431	35.29	70.03
ARIMA2	131.90	529	35.13	94.99

10.4.4 Význam měření pro telefonní hovor

Převedeme-li naměřená data do reálné situace telefonního hovoru, pak hovor vedený přes síť se směrováním dle predikce stavu linek bude mít delší prodlevu při navazování spojení a jeho průběh bude vykazovat větší zpoždění než kdyby překryvná síť směrovala jen dle posledního stavu linek. Oproti přímému spojení si ale telefonista polepší snížením průměrné doby odezvy.

11 Závěr

V rámci diplomové práce jsem se seznámil se stávajícími směrovacími algoritmy, existujícími překryvnými sítěmi, časovými řadami a predikcí z nich.

Na základě těchto znalostí jsem implementoval překryvnou síť s tvorbou spojení dle aktuálních stavu i dle predikce následujícího stavu linek.

Pro ověření přínosu predikce jsem také napsal vlastní simulaci sestávající ze skriptů využívajících prostředků operačního systému a emulace síťových parametrů.

V návaznosti na vyvinutý simulační aparát jsem navrhl experiment, který jsem následně i provedl. Výsledky získané teoretickým výpočtem a praktickou simulací jsem porovnal s verdiktem, že predikce stavu linek nezlepšuje kvalitu služeb lepší volbou cest.

Předpokládaným výsledkem teoretického i praktického experimentu bylo celkové zlepšení doby odezvy oproti směrování bez predikce. Tento předpoklad se ovšem nenaplnil a měřením bylo dokázáno, že predikce stavu sítě základní metodou $ARIMA$ není příliš uspokojivá. Hlavními důvody jsou výpočetní náročnost a nepřesnost predikce.

Mnou optimalizované metody s posuvem jsou dle grafů a statistických dat méně efektivní než směrování na základě posledního stavu a navíc je jejich nevýhodou vyšší výpočetní náročnost.

Výsledkem mého měření tedy je, že cena v podobě výpočetního výkonu využitého na predikci nepřináší příliš velký přínos.

Další navazující práce se mohou zaměřit na zefektivnění redikčních algoritmů a navržení vhodných koeficientů pro vážený průměr více časových řad. Dalším možným vylepšením je využití multiplexování a persistentních spojení uvnitř překryvné sítě.

Zadání bylo splněno v plném rozsahu.

Seznam obrázků

Seznam tabulek

Seznam zkratek

1080p 1080 pixelů na výšku neprokládaně

4K čtyři tisíce, označení pro rozlišení cca 4000 px na šířku

ABR Available bit rate

API Application programming interface

ATM Asynchronous transfer mode

CBR Constant bit rate

CPU Central processing unit

DAO Data access object

DHT Distribuovaná hash tabulka

DTO Data transfer object

EIGRP Extended Interior Gateway Protocol

FIR Finite impulse response

HTML Hypertext Markup Language

ICMP Internet Control Message Protocol

IIR Infinite impulse response

Intel TBB Intel Thread Building Blocks

JSON Javascript Object Notation

MVC Model-View-Controller

No-SQL Nerelační databáze

OSPF Open Shortest Path Firts

RAII Resource Acquisiton Is Initialization

RMI Remote method invocation

RON Resilient overlay network

RSVP Resource reservation protocol

RTT Round Trip Time

TCP Transmission control protocol

TOS Type of service

UBR Unspecified bit rate

UDP User Datagram Protocol

VBR Variable bit rate

Seznam výpisů

Literatura

[1] Andersen, D. G. Resilient Overlay Networks. Master's thesis, Massachusets Institute of Technology, 2001.

[2] Apostolopoulos, G. et al. *QoS Routing Mechanisms and OSPF Extensions* [online]. august 1999. [cit. 26.1.2016]. Dostupné z: `https://tools.ietf.org/html/rfc2676`.

[3] Balchunas, A. *Complete CCNA Study Guide* [online]. 2014. [cit. 24.1.2016]. Dostupné z: `http://www.routeralley.com/guides.html`.

[4] Bauer, K. – Grunwald, D. – Sicker, D. Predicting Tor path compromise by exit port. In *2009 IEEE 28th International Performance Computing and Communications Conference*, s. 384–387, Dec 2009. doi: 10.1109/PCCC.2009.5403852.

[5] Brownlee, N. *OSPF protocol analysis* [online]. August 2015. [cit. 8.5.2016]. Dostupné z: `https://datatracker.ietf.org/doc/draft-savage-eigrp`.

[6] Bříza, M. *Anylýza výkonnosti firmy Mida, a.s. pomocí časových řad* [online]. 2010. [cit. 6.4.2016]. Dostupné z: `https://goo.gl/Q3yQbW`.

[7] Durresi, A. – Jain, R. *Handbook for Computer Networks.* Wiley, 12.2007. Dostupné z: `http://www.cse.wustl.edu/~jain/books/ftp/atm_chp.pdf`. ISBN 0471784613.

[8] Feamster, N. *Why Your Netflix Traffic is Slow, and Why the Open Internet Order Won't (Necessarily) Make It Faster* [online]. Freedom to tinker, 2015. [cit. 17.4.2016]. Dostupné z: `https://goo.gl/GP7TdH`.

[9] Giacalone, S. et al. *OSPF Traffic Engineering (TE) Metric Extensions* [online]. March 2015. [cit. 26.1.2016]. Dostupné z: `https://tools.ietf.org/html/rfc7471`.

[10] Hag, H. M. A. E. – Sharif, S. M. An adjusted ARIMA model for internet traffic. In *AFRICON 2007*, s. 1–6, Sept 2007. doi: 10.1109/AFRCON.2007.4401554.

[11] Hedrick, C. *Routing Information Protocol* [online]. 1988. [cit. 24.1.2016]. Dostupné z: `https://tools.ietf.org/html/rfc1058`.

[12] QIONG, H. – HAO, T. Short-term Traffic Flow Forecasting Based on ARIMA-ANN. In *2007 IEEE International Conference on Control and Automation*, s. 2370–2373, May 2007. doi: 10.1109/ICCA.2007.4376785.

[13] HUFFER, F. W. *Backshift notation* [online]. [cit. 20.4.2016]. Dostupné z: `http://goo.gl/EZljTx`.

[14] HYNDMAN, R. J. – ATHANASOPOULOS, G. *Forecasting: principles and practice* [online]. 2012. [cit. 8.2.2016]. Dostupné z: `https://www.otexts.org/fpp`.

[15] IDOL, R. D. *Large-Scale TCP Packet Flow Analysis for Common Protocols Using Apache Hadoop* [online]. 2004. [cit. 6.4.2016]. Dostupné z: `http://goo.gl/p5mcAl`.

[16] KHAN, P. – KONAR, G. – CHAKRABORTY, N. Modification of Floyd-Warshall's algorithm for Shortest Path routing in wireless sensor networks. In *2014 Annual IEEE India Conference (INDICON)*, s. 1–6, Dec 2014. doi: 10.1109/INDICON.2014.7030504.

[17] KIM, P. S. – KWON, W. H. Forgetting least squares estimation FIR filters without noise covariance information. In *SICE 2000. Proceedings of the 39th SICE Annual Conference. International Session Papers*, s. 1–6, 2000. doi: 10.1109/SICE.2000.889643.

[18] KOLÁŘ, J. *Teoretická informatika.* Praha : Česká informatická společnost, 2. edition, 2004. Dostupné z: `http://www.exfort.org/2005l/4/x36tin/pdf/ti.pdf`. ISBN 80–900853–8–5.

[19] LIU, X. – WANG, N. An Improved Tor Circuit-Building Protocol. In *Artificial Intelligence, 2009. JCAI '09. International Joint Conference on*, s. 671–675, April 2009. doi: 10.1109/JCAI.2009.27.

[20] LUAN, E. K. et al. *A Survey and Comparison of Peer-To-Peer Overlay Network Schemes* [online]. IEEE COMMUNICATIONS The Electronic Magazine of Original Peer-Reviewed Survey Articles, 2005. [cit. 16.4.2016]. Dostupné z: `https://www.cl.cam.ac.uk/research/dtg/www/files/publications/public/mp431/ieee-survey.pdf`.

[21] MALKIN, G. *RIP Version 2* [online]. 1998. [cit. 24.1.2016]. Dostupné z: `https://tools.ietf.org/html/rfc2453`.

[22] MANILS, P. et al. *Compromising Tor Anonymity Exploiting P2P Information Leakage* [online]. HAL archives-ouvertes, 2010. [cit. 17.4.2016]. Dostupné z: `https://hal.inria.fr/file/index/docid/471556/filename/TorBT.pdf`.

[23] Mount, D. *Algorithms* [online]. 1998. [cit. 4.2.2016]. Dostupné z: http://goo.gl/EuuBLI.

[24] Moy, J. *OSPF protocol analysis* [online]. July 1991. [cit. 26.1.2016]. Dostupné z: https://tools.ietf.org/html/rfc1245.

[25] Puype, B. et al. *Multilayer Traffic Engineering Performance in Overlay Networks.* Ghent University, 2. edition, 2004.

[26] Páv, M. *Překlad CCNA Exploration - Routing Protocols and Concepts.* VOŠ a SPŠE Plzeň, 2010.

[27] Rayburn, D. *Cogent Now Admits They Slowed Down Netflix's Traffic, Creating A Fast Lane and Slow Lane* [online]. 2014. [cit. 17.4.2016]. Dostupné z: http://goo.gl/t7ZeLT.

[28] Reynolds, J. – Postel, J. *ASSIGNED NUMBERS* [online]. 1988. [cit. 24.1.2016]. Dostupné z: https://www.ietf.org/rfc/rfc1700.

[29] Rogozhin, K. *Using Intel® TBB in network applications: Network Router emulator* [online]. Intel, 2011. [cit. 17.4.2016]. Dostupné z: https://goo.gl/vEjAOB.

[30] Singh, S. – Townsley, M. – Pignataro, C. *Asynchronous Transfer Mode (ATM) over Layer 2 Tunneling Protocol Version 3 (L2TPv3)* [online]. 1988. [cit. 24.1.2016]. Dostupné z: https://tools.ietf.org/html/rfc4454.

[31] Skupa, J. Dynamické směrování v překryvných sítích s využitím predikce. 2016.

[32] Song, M. – Mathieu, B. QSON: QoS-aware Service Overlay Network. In *Communications and Networking in China, 2007. CHINACOM '07. Second International Conference on*, s. 739–746, Aug 2007. doi: 10.1109/CHINACOM.2007.4469494.

[33] Touretzky, D. – Laskowski, K. *Neural Networks for Time Series Prediction* [online]. 2011. [cit. 20.4.2016]. Dostupné z: https://goo.gl/xWKWzP.

[34] Yung-Chin – Yung-Chien – Su, K. L. Evolutionary Neural Networks for Time Series Prediction. In *Genetic and Evolutionary Computing (ICGEC), 2010 Fourth International Conference on*, s. 219–223, Dec 2010. doi: 10.1109/ICGEC.2010.61.

[35] Čada, R. – Kaiser, T. – Ryjáček, Z. *Základní grafové algoritmy.* 2004. Dostupné z: http://goo.gl/cSe3yQ.