Západočeská univerzita v Plzni

Programové techniky

KIV/PT

Dokumentace semestrální práce

Jan Čarnogurský
Jakub Vaněk

3. prosince 2017

Obsah

1 Zadání

1.1 Motivace

Firma KIV-Internet vlastní rozsáhlou počítačovou síť (v řádech tisíců uzlů). Počítačová síť slouží k přenosu dat mezi jednotlivými stroji a jejich přístupu k internetu. Síť využívá různé technologie od wifi po optické spoje. Kvalita a rychlost jednotlivých uzlů je dána typem spoje. V současné době se provoz po síti řídí protokolem OSPF, který ale není úplně optimálním řešením. Cílem Vaší práce je proto navrhnout protokol pro optimalizaci datových toků sítí.

1.2 Zadání implementace

Síť lze reprezentovat jako mesh topologii pomocí grafu. Získáme tím graf s ohodnocením jednotlivých hran (dvojice čísel), které nám určuje maximální propustnost a stabilitu daného spoje. Maximální propustnost udává, kolik můžeme spojem poslat Mbit/s (1s = nejmenší krok simulace propustnosti sítě). Tato hodnota bude v rozsahu 1 – 10000 Mbit/s. Druhá hodnota vyjadřuje spolehlivost a bude v rozsahu 0 - 1. Hodnota 1 udává maximální spolehlivost, kdy nedochází k žádné ztrátě packetů, jakákoliv hodnota menší než 1 udává, kdy začne docházet ke ztrátovosti 50% při zatížení. Například 0.9 znamená, že při vytížení linky nad 90% může dojít ke ztrátě 50% všech dat, které v dané vteřině přenášíte, tj. budou se muset odeslat znovu. Pravděpodobnost ztráty bude také 50% (realizujte generátorem náhodných čísel).

Na uzlech sítě jsou „chytré“ routery se síťovým smart-stackem (implementujete Vy), který dokáže uchovávat až 100Mb dat pro případ, že do uzlu přiteče více dat, než můžete aktuálně odeslat. Router má inteligentní routovací systémem, který se může měnit v čase (implementujete Vy). Routery mohou ovlivnit jak cestu, tak počet posílaných dat sousednímu uzlu. V případě přetečení stacku v cílovém uzlu dojde ke ztrátě celého balíku dat a musí se poslat znovu.

2 Analýza

2.1 Struktura

Klíčová myšlenka pro pochopení problému je uvědomění si, jakým způsobem je síť reprezentována. Jak již bylo řečeno v zadání, topologii sítě lze reprezentovat pomocí **grafu**. Graf je matematicky definován jako dvojice **vrcholů** a **hran**. V našem případě budeme chápat routery v topoligii jako vrcholy a jejich propojení jako hrany.

Grafy mohou být dvojího typu - **orientované** a **neorientované**. I když by se mohlo zdát, že náš graf bude orientovaný, kvůli určování směru dat, mezi všemi routery funguje obousměrná komunikace, a tudíž bude graf neorientovaný.

Důležitou vlastností grafů je, zda je graf **ohodnocený** nebo **neohodnocený**. Ohodnocení grafu znamená přiřazení váhy (ceny) hranám. V našem případě budeme ohodnocovat hrany pomocí dvou vlastností a těmi je **maximální propustnost** a **maximální spolehlivost**. Význam těchto dvou vlastností je popsán v zadání. Pro naše účely jsme se rozhodli toto ohodnocení řešit součinem těchto dvou hodnot, protože čím větší spolehlivost a zároveň čím větší objem dat můžeme linkou poslat, tím je pro nás linka lepší.

2.2 Algoritmy

Tento graf bude fungovat k přenášení packetů. Abychom věděli, kudy packet v síti (grafu) poslat, budeme potřebovat algoritmus, který mu najde cestu. Algoritmů pro hledání cest v grafu existuje spoustu. Začneme s **Dijkstrovým** algoritmem, poté si proberem algoritmus Floyd-Warshall.

Dijkstrův Algoritmus je algoritmus hojně používaný k nalezení nejkratší cesty. Algoritmus bohužel nenachází cestu, nýbrž délku nejkratší cesty. Tento problém se dá ale odstranit, pokud naimplementujeme zpětné vyhledání cesty poté co algoritmus proběhne a nalezne délku. Algoritmus prochází graf od zdrojového uzlu. Na začátku ohodnotí všechny uzly nekonečnem a uzly, ke kterým existují hrany, ohodnocením hran, které tyto uzly se zdrojovým uzlem spojují. Následně vybere uzel s nejmenším dosavadním ohodnocením a ohodnotí všechny připojené uzly tak, že spočítá součet výchozího uzlu a přičte k němu ohodnocení hrany, kterou je další uzel připojen. Pokud je na uzlu větší hodnota, než hodnota takto získaná, přepíše uzlu hodnotu na tu menší. Tímto způsobem algoritmus probíhá až do té doby, než nalezne cílový uzel. Tímto způsobem algoritmus najde celkové ohodnocení nejkratší cesty k cílovému uzlu. Alogoritmus má také příjemnou asymptotickou časovou složitost.

Floyd–Warshallův algoritmus porovnává všechny možné cesty v grafu mezi všemi dvojicemi vrcholů. Pracuje tak, že postupně vylepšuje odhad na nejkratší cestu do té doby, než je zřejmé, že odhad je optimální.

Mějme graf G s vrcholy V očíslovanými 1 až N. Dále mějme funkci *nejkratsiCesta(i,j,k)*, která vrací nejkratší možnou cestu z i do j s použitím pouze vrcholů 1 až k jako mezivrcholů. Pomocí této funkce chceme najít nejkratší cestu mezi všemi dvojicemi i a j s použitím mezivrcholů 1 až *k+1*.

Na nejkratší cestu máme dva kandidáty: buď je nejkratší cesta v množině vrcholů *(1...k)*, nebo existuje cesta jdoucí z i do *k+1*, a poté z *k+1* do j, která je lepší (kratší) než ta stávající. Nejlepší cesta z i do j používající pouze vrcholy 1 až k je definována funkcí *nejkratsiCesta(i,j,k)*. Délka nejlepší cesty z i *k+1* a poté do j je pak zřejmě součet délek nejkratší cesty z i do *k+1* a nejkratší cesty z *k+1* do j.

Tento algoritmus má poměrně jednoduchou implementaci. Zavrhli jsme jej však z důvodu jeho časové složitosti. Jeho časová složitost N^3 by způsobovala, že hledání cest mezi tisíci routery by se mohlo protáhnout i na pár hodin. Z tohoto důvodu jsme si vybrali pro hledání cest v grafu **Dijkstrův** algoritmus.

3 Implementace

3.1 Reprezentace grafu

Graf se dá programově reprezentovat dvěma způsoby. Maticí sousednosti a seznamem uzlů se sousedy.

Matice by se mohla jevit jako lepší řešení, ale zamysleme se nad její náročností na paměť. Tato matice by byla dvourozměrná, kde v řádku i sloupci by byly všechny routery a na pozici *(i,j)* ohodnocení hrany mezi nimi. Předpokládejme, že by na všech pozicích byla čísla typu integer. Velikost integeru je 4 byty. Pokud bychom měli například 10 000 routerů, dostali bychom v poli 100 000 000 integerů. V takovém případě by bylo potřeba zhruba 400mb paměti. V případě, že by však routerů bylo například 60000, potřebovali bychom už přibližně 14,5gb paměti, což už je s našimi možnostmi nepřijatelné. Pro reprezentaci grafu jsme tedy zvolili seznam routerů, kde každý router má svůj vlastní seznam sousedních routerů.

Graf tedy reprezentujeme seznamem routerů a a seznamem linků. Routery představují uzly grafu a linky jeho hrany. Tyto linky a routery jsou načteny z přiloženého souboru. Čtení ze souboru vždy načte řádek, který je popsaný jako hrana mezi dvěma routery s propustností a svou spolehlivostí. Pro každý řádek zkontroluje, zda router již existuje. Pokud ano, přidá druhý router do jeho seznamu sousedů a druhému routeru přidá do seznamu sousedů router první. Pokud ne, vytvoří router a přidá sousedy stejným způsobem. Zároveň vytvoří hranu, mezi těmito dvěma routery.

3.2 Simulace

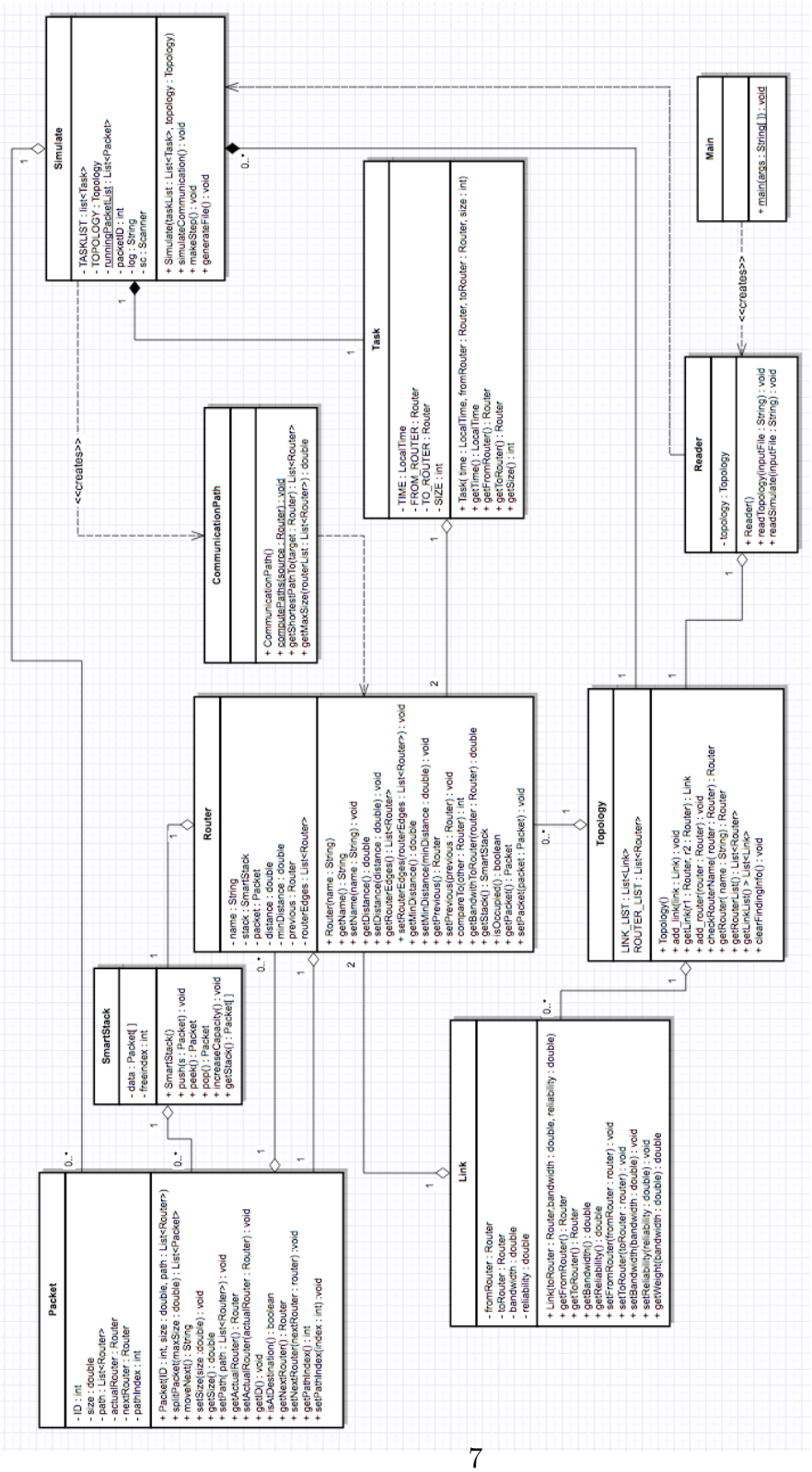
Po vytvoření grafu začne probíhat simulace. Simulace se také načítá ze souboru, kde je na každém řádku popsán packet, který má zadaný výchozí uzel, cílový uzel a svou velikost.

Program načte řádek, vytvoří packet s popsanou velikostí a Dijkstrovým algoritmem nalezne cestu, po které se má packet dostat do cílového uzlu. Tuto cestu má každý packet uloženou formou seznamu routerů, přes které se má do cílového uzlu dostat. Na začátku se každý packet nachází u tzv. sourceUsera. SourceUser je vlastně počítač, ve kterém se nacházejí posílaná data. Před posláním packetu prvnímu routeru, se packet rozdělí na n packetů. Jejich velikost je získána z cesty, kde zjišťujeme na základě propustnosti a spolehlivosti linků mezi routery maximální velikost packetu takovou, aby nedocházelo k jejich ztrátě . Všechny tyto packety jsou na začátku přiřazené sourceUseru. Všechny tyto packety se uloží do seznamu aktuálně posílaných packetů.

Postupně se seznam prochází a simulace zkouší pohnout každým packetem o jeden router dále. Packet zjistí, zda je router, na který se chce posunout volný. Pokud volný je, přesune se na něj. Pokud volný není, zůstane na svém místě. Takováto funkce zajišťuje, že pokud je původní packet rozdělen na více packetů, neposouvají se najednou. V první kroku je posunut první packet o jeden router dále a ostatní packety čekají v sourceUseru. V dalším kroku se první packet posune na druhý router, druhý packet na první router a ostatní dále čekají. Tímto způsobem jsou přenášeny všechny packety až k cílovému uživateli (targetUser).

Jediný problém, který může nastat, je pokud jdou dva packety proti sobě, nebo jsou tři a více packetů zacyklených. Simulace proto neřeší pouze, zda je router na který chce packet přejít volný, ale i situaci, kdy router volný není, ale zároveň packet, který je na něm uložen chce přejít na aktuální router. V takovém případě se packety prohodí a simulace pokračuje. Pokud se zacyklí tři a více packetů, situaci jsme nezvládli vyřešit.

3.3 UML diagram



4 Uživatelská dokumentace

4.1 Instalace a spuštění

Je nutné, aby v počítači byla nainstalována Java. Program nevyžaduje žádnou instalaci. Ke spuštění postačí pouze soubor **PT-simulation.jar** a dva vstupní soubory **links_.txt** a **simulate_.txt**. Je nutné, aby tyto tři soubory byly ve stejné složce.

Aplikace je pouze konzolová a nelze spustit kliknutím na soubor PT-simulation.jar. Pro spuštění je nutné otevřít konzoli. Otevřete tedy terminál (Windows: příkazem cmd, MacOS: CMD+mezerník - zde napište terminál a potvrďte enterem) ve složce, kde máte všechny tři soubory nutné ke spuštění a příkazem *java -jar PT-simulation.jar* aplikaci spustíte.

4.2 Ovládání

Aplikace je ovládána enterem. Mačkejte enter pro zobrazení jednotlivých kroků aplikace. Pokud chcete aby aplikace proběhla najednou, držte enter, dokud aplikace neskončí.

Aplikace načte topologii a oznámí, že je topologie načtena a oznámí začátek simulace. Z prvního řádku načte packet a zobrazí ho v podobě:
Načítám packet: Packet0 [size=xxxx, path=[sourceUser, A, B, C, D, targetUser], actualRouter=A]. Je vidět označení packetu, následuje jeho velikost, cesta po které bude postupovat a aktuální místo kde se packet nachází. Následuje oznámení, zda byl packet rozdělen a na kolik packetů. Po stisku klávesy enter aplikace vypíše aktuální packety k odeslání, pohyb každého packetu takto: [*packet0:* $A \rightarrow B$], což značí, že packet0 se posunul z routeru A do routeru B. Následuje oddělovací čára a nově načtený packet. Po každém stisknutí klávesy enter proběhne jeden takovýto krok. Po skončení aplikace se vygeneruje soubor *output.txt*, ve kterém je stejným způsobem popsán průběh simulace.

5 Závěr

I přes potíže, které během vytváření aplikace nastaly, se nám podařilo vytvořit aplikaci, která simuluje posílání packetů v síti. Bohužel jsme ale nedokázali naimplementovat několik věcí.

V řešení jsme zcela ignorovali čas poslání různých packetů a posíláme packety postupně tak, jak jsou popsány ve vstupním souboru. Nenaimplementovali jsme ztrácení jednotlivých packetů a tuto implementaci jsme obešli rozkouskováním packetů na packety s takovou velikostí, aby ke ztrátám packetů nedocházelo. V řešení nevyužíváme smartstack, do kterého by se packety ukládali. Kvůli časové náročnosti jsme nenaimplementovali žádné uživatelské prostředí, ani statistiky průběhu simulace apod. O posílání packetů ani o vymýšlení cesty se nestarají routery, ale packety. Packet využívá router jako zastávku na cestě a router je vůbec nekontroluje. Při testování jsme narazili na problém s posíláním packetů proti sobě, který jsme vyřešili nejlépe jak jsme v omezeném čase dokázali. Při zacyklení 3 a více packetů se aplikace zacyklí a nikdy neskončí. Tento problém jsme nedokázali vyřešit. Při načítání větší topologie (řádově tisíce routerů) trvá načtení a vytvoření topologie relativně dlouho (několik sekund). Všechny tyto problémy by se dali pravděpodobně vyřešit lepší analýzou problému před jeho řešením, protože jsme na takové problémy narazili až když nastaly a nemohly jsme měnit celou strukturu aplikace.

Pozitivem aplikace je skutečnost, že pokud nenastane výše popsané zacyklení packetů, měli by všechny packety projít celou cestu z počátečního místa až do cílového. Hledání cesty i mezi tisícovkami routerů trvá max 0,3 vteřiny, tudíž zvolený algoritmus pro hledání cesty je pravděpodobně správný.

S aplikací takového rozsahu jsme se nikdy předtím nesetkali, a zároveň jsme poprvé pracovali ve dvojici, což značně ovlivnilo náš odhad, kolik času bude k vývoji potřeba. Přes všechna negativa jsme se svou prací spokojení a byla pro nás velkým přínosem, jak v oblasti programování, tak v oblasti vývoje v týmu.