MASARYKOVA UNIVERZITA

FAKULTA INFORMATIKY

Modelování síťových topologií

DIPLOMOVÁ PRÁCE

Bc. Jiří Szabó

Brno, jaro 2014

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

.....

Vedoucí práce: RNDr. Pavel Troubil

Poděkování

Touto cestou bych rád poděkoval RNDr. Pavlu Troubilovi za vedení této práce a za ochotu a rady při jejím vytváření.

Shrnutí

Diplomová práce „Modelování síťových topologií“ se věnuje historii modelování počítačových sítí a tvorbě programu `GenerateTopology`, jenž při své činnosti využívá nejnovějších poznatků tohoto oboru. Výstupem programu je pak krom samotné topologie o požadované velikosti také zadaný počet strojů umístěných v této síti, které bude možno použít při modelování náročných datových přenosů.

Klíčová slova

topologie, Internet, CoUniverse, Java, GenerateTopology

Obsah

1. Úvod ... 4
1.1. Uvedení do problematiky ... 4
1.2. Obsah jednotlivých kapitol ... 5
1.3. Přílohy ... 5
2. Historie modelování ... 6
2.1. Náhodnostní modely ... 6
2.2. Strukturované modely ... 6
2.3. Mocninné zákony ... 7
2.4. dK-distribuce ... 8
2.5. Vztahy mezi AS ... 8
3. Program GenerateTopology ... 10
3.1. Úvod, motivace ... 10
3.2. Návrh implementace ... 10
3.2.1. Základní návrh ... 10
3.2.2. Škálování grafu ... 11
3.2.3. Určování typů hran ... 12
3.2.4. Generování vlastností a strojů ... 12
3.2.4.1. Řešení rychlostí ... 12
3.2.4.2. Řešení zpoždění a ztrátovosti ... 15
3.2.4.3. Generování strojů ... 16
4. Implementace ... 19
4.1. Technické aspekty ... 19
4.2. Základní informace ... 20
4.3. Zpracování iniciálního grafu ... 22
4.4. Třída RescaleGraph ... 22
4.4.1. Vytvoření 2K-distribuce iniciálního grafu ... 22
4.4.2. Samotné škálování ... 23
4.4.3. Vygenerování finálního grafu ... 27
4.5. Třída ASRelations ... 30
4.5.1. Vygenerování cest ... 30
4.5.2. Hrany typu „s2s“ ... 31

4.5.3. Hrany typu „p2c“ ... 31
4.5.4. Hrany typu „p2p“ ... 33
4.6. Třída FindAttributes ... 35
4.6.1. Vlastnosti hran ... 35
4.6.2. Generování strojů ... 37
5. Vyhodnocení ... 40
5.1. Velikost vygenerovaného grafu ... 40
5.2. Porovnání 2K-distribucí ... 41
5.3. Další vlastnosti grafu ... 42
5.4. Poměry mezi rychlostmi ... 43
6. Prostor pro zlepšení ... 44
6.1. Generování BGP cest ... 44
6.2. Škálování směrem dolů ... 44
6.3. Zpoždění a ztrátovost paketů ... 45
7. Závěr ... 46
Seznam literatury ... 47
Příloha A – rozdělení linek sítě ENREN ... 50
Příloha B – Porovnání velikostí (upravený alg.) ... 51
Příloha C – Porovnání velikostí (původní alg.) ... 52
Příloha D – Porovnávání 2K-distribucí ... 53
Příloha E – Porovnávání 2K-distribucí (detail) ... 54
Příloha F – Škálování směrem dolů ... 55

1. Úvod

1.1. Uvedení do problematiky

Výkon a chování nových síťových aplikací či protokolů[1] musí být důkladně otestováno. Ideálně ještě před uvedením do skutečného provozu, neboť v opačném případě by se autoři při případných nedostatcích vystavovali problémům jako například odliv zákazníků, ztráta důvěry v aplikaci/protokol a podobně.

Možnosti simulace na vlastních počítačových sítích jsou však velice omezené. Především u protokolů je tato situace nejsnáze viditelná. Nelze si totiž představit uměle vytvořenou síť, která by se alespoň částečně blížila realitě. Její vytvoření a následná údržba by byla zkrátka příliš finančně nákladná, nemluvě o potřebě ji neustále rozšiřovat. Ale i kdyby to bylo možné, pořád by tu byl problém určit topologii dané sítě, aby odpovídala reálnému prostředí.

Ke slovu tak nutně musí přijít snaha pokusit se takovouto počítačovou síť vytvořit coby abstraktní model a běh a chování daného produktu simulovat na něm. Jedná se o výrazně levnější a snadněji uskutečnitelné řešení než to popsané v předchozím odstavci, problém však může nastat při vyhodnocování testování. Vypovídající hodnota výsledků se totiž odvíjí od faktu, s jakou přesností daný model odráží realitu. Od konce osmdesátých let vzniklo již několik generací generátorů topologií, které se zvyšující se přesností reflektují reálné počítačové sítě, zejména Internet.

Cílem diplomové práce "Modelování síťových topologií" je nejprve zmapovat vývoj používaných modelů a na základě získaných znalostí navrhnout a implementovat vlastní generátor síťové topologie. Výstupem vzniklé aplikace je graf, jehož podoba je v souladu se současným pohledem na tuto problematiku. Ohodnocení hran a vrcholů se pak odvíjí od důležitých vlastností pro plánování náročných datových přenosů.

[1] Protokol je sada pravidel, podle níž probíhá komunikace a výměna dat mezi dvěma uzly v počítačové síti.

1.2. Obsah jednotlivých kapitol

Druhá kapitola rozebírá historii používaných modelů od kompletně náhodných přístupů z konce osmdesátých let až po současnost. Každá podkapitola je věnována jedné generaci generátorů a zběžně popisuje jejich principy a zmiňuje nejpoužívanější řešení.

Další a obsahově nejrozsáhlejší pasáž textu je věnována vzniku hlavního výstupu diplomové práce – programu `GenerateTopology`. Každá kapitola reflektuje jednotlivé vývojové fáze aplikace, od prvotního návrhu přes implementaci až po závěrečné zhodnocení a identifikace nejzásadnějších nedostatků.

Závěr práce se pak snaží o nastínění možností dalšího vývoje aplikace. Diskutuje také její přínos do problematiky modelování síťových topologií a náročných datových přenosů.

1.3. Přílohy

K textu je přiloženo také několik příloh, které nabízí především doplňující grafy a tabulky prezentující výsledky z kapitoly 5.

2. Historie modelování

Jak bylo řečeno v úvodu, modelování síťových topologií hraje v informačních technologiích zásadní roli. Lze pomocí něj simulovat chování a výkonnost nejenom nových protokolů či aplikací, ale také šíření virů. Tato kapitola rekapituluje historii této problematiky s důrazem na používané techniky a modely.

2.1. Náhodnostní modely

První generace modelů využívala náhodné grafy. Nejslavnějším a nejpoužívanějším z nich se stal Waxmanův model (1988) [1], což byla upravená verze staršího náhodného grafu autorské dvojice Erdös-Rényi ze začátku šedesátých let. Každý vrchol měl prostřednictvím souřadnic určenu svoji pozici ve dvojrozměrném prostoru. Hrana mezi dvěma vrcholy pak vznikala s určitou pravděpodobností P, která závisela na Euklidovské vzdálenosti d mezi nimi, maximální vzdálenosti dvou uzlů L a konstantách $\alpha < 0$ a $\beta \leq 1$. Ty ovlivňovaly další parametry modelu jako celkový počet hran v grafu, respektive poměr mezi krátkými a dlouhými linkami [1]. Výpočet P pro dvojici uzlů u a v byl tedy určen následujícím výrazem:

$$P(u, v) = \alpha e^{-d/(\beta L)}$$

Během následujících let se objevilo několik návrhů na jeho úpravu [2]. Zmiňme například model Doara a Leslieho [3], kteří se pokusili pravděpodobnost vzniku hrany škálovat za pomocí požadovaného průměrného stupně ve výsledném grafu. Tím mohli dosáhnout daleko větší kontroly nad počtem vygenerovaných hran. V konečném důsledku se však tento model od Waxmana příliš nelišil, neboť při vhodné volbě konstanty α šlo dosáhnout stejného výsledku [2].

2.2. Strukturované modely

V roce 1996 vzniká zcela opačný způsob generování topologií – strukturovaný. Autoři argumentovali především faktem, že jimi zkoumané počítačové sítě zcela jasně nevznikají náhodně, ale vykazují určitou hierarchickou strukturu [4]. Krom toho bylo poukazováno i na jednu z nevýhod Waxmanova modelu, který nedokázal při každém spuštění vytvořit jedinou souvislou komponentu a proces generování tedy musel být několikrát opakován.

Z jejich pohledu tedy měly generátory pracovat spíše na principu postupného budování sítě zezdola. Jako první by byly vytvořeny malé počítačové sítě, ty pak mezi sebou propojovat do větších celků a tak dál a dál, dokud nevznikne jeden velký souvislý graf [4]. Modely založené na těchto principech byly později pojmenovány jako Georgia Tech Internetwork Topology Models (GT-ITM).

2.3. Mocninné zákony

Generátory strukturované topologie převládaly až do roku 1999, kdy Michalis Faloutsos společně se svým týmem zjistil, že propojenost uzlů ve velkých počítačových sítích reflektuje mocninné zákony [5]. Na základě pozorování růstu Internetu v rozmezí listopadu 1997 až prosince 1998 předložili tři mocninné zákony [5]:

- pro každý uzel v je velikost jeho výstupního stupně d_v přímo úměrná jeho pozici r_v na logaritmické stupnici mocněné konstantou R; tedy $d_v \propto r_v^R$
- frekvence výskytu tohoto stupně f_d je mu při mocnění konstantou O přímo úměrná; $f_d \propto d^O$
- vlastní čísla λ_i grafu jsou přímo úměrná jejich pořadí i při mocnění konstantou ε; $\lambda_i \propto i^\varepsilon$

Krom učení hodnot všech tří konstant také uvedli několik praktických aplikací, jak mohou být při modelování síťových topologií využity.

Modely ze skupiny GT-ITM mocninné zákony nesplňovaly [6] a tudíž byly velice rychle nahrazovány těmi, které to dokážou. Lze je rozdělit do dvou základních kategorií [7]:

- **causality-aware** (v překladu „řešící souvislosti“) – typickými zástupci jsou model přiřazování na základě preferencí Barabásiho a Alberta [8] a Heuristically Optimal Topology (dále HOT) [9]. Topologie jsou tvořeny nejenom na základě mocninných zákonů, ale také pomocí formalizovaných pravidel pro evoluci počítačové sítě.
- **causality-oblivious** („neřešící souvislosti“) – zaměřují se pouze na reprodukci mocninných zákonů a nesnaží se reflektovat jiné vlivy růstu sítě. Patří sem náhodné grafy reprodukující očekávanou či přesně

danou posloupnost stupňů, Markovův model pro přepojování grafů, generátor topologií Inet a také dK-distribuce.

2.4. dK-distribuce

Principy dK-distribucí byly popsány v polovině roku 2006. Čtyřčlenný autorský tým je ve své publikaci označil za novou systematickou techniku pro analyzování síťových topologií. Krom jejich definice předložil také několik metod tvorby náhodných grafů, které dokáží při správné volbě parametru d reprodukovat všechny důležité vlastnosti počítačových sítí [10].

Při použití dK-distribucí je graf rozčleněn na menší podsítě o velikosti d a v rámci nich se pak specifikují vztahy mezi stupni vrcholů [10]. Parametr d může nabývat celočíselných hodnot od nuly až po n, kde n = počet vrcholů grafu. Jeho volbou pak určujeme přesnost, s jakou jsou zaznamenány vlastnosti daného grafu. Zatímco při $d = 0$ je zapamatován pouze průměrný stupeň, při d = n už popisujeme celý graf.

Autoři na základě výzkumu označili $d = 2$ za dostatečnou přesnost schopnou reprodukovat většinu důležitých vlastností. Při volbě $d = 3$ jsou pak věrně splněny všechny z nich [10].

Na svoji práci o rok později navázali vytvořením algoritmu, který dokáže 1K- a 2K-distribuce škálovat [11]. Díky tomu bylo možné model topologie zvětšit nebo zmenšit na jakoukoliv velikost, přičemž se zachovají jeho požadované vlastnosti.

2.5. Vztahy mezi AS

Jednou z možností, jak na velkou počítačovou síť pohlížet, je jako na vzájemně propojené samostatné sítě, takzvané Autonomní systémy (dále AS). Hrany mezi nimi pak mohou reprezentovat jeden ze tří následujících vztahů [12]:

- **Sibling-to-sibling** („s2s“) – AS spadají do vlastnictví stejné organizace a proudící datové toky si tedy mezi sebou volně vyměňují.
- **Customer-to-provider** („c2p“ nebo „p2c“) – AS si rozdělí role poskytovatele a zákazníka. Zákazník pak platí za všechna data, která se po této lince posílají.
- **Peer-to-peer** („p2p“) – obě AS si mezi sebou a svými zákazníky volně vyměňují data, ovšem ne mezi jejich poskytovateli a ostatními peery.

Jako první se modelováním těchto vztahů zabývala Lixin Gao (2001) [13]. Určila, že každá BGP[2] cesta musí mít následující strukturu: žádná nebo několik linek typu „c2p“ nebo „s2s“, maximálně jedna „p2p“ linka a pak opět žádná nebo více „p2c“ či „s2s“ linek. Cesty splňující tuto strukturu označila za platné. Následně pak definovala heuristiku identifikující poskytovatele a peery na základě stupně daného AS a počtu platných cest.

Na její práci o rok později navázal Subramanian se svým týmem [14]. Problém identifikace typů hran popsali jako matematický problém a jeho řešení převedli na kombinatorický optimalizační problém pojmenovaný Type of Relationship (ToR): každé hraně grafu přiřaď typ tak, aby byl maximalizován počet platných cest. Spekulovali, že se jedná o NP-úplný problém a nabídli své vlastní heuristické řešení.

Domněnka o NP-úplnosti byla později potvrzena ze dvou na sobě nezávislých zdrojů [15][16]. Oba také nabídli svá vlastní aproximativní řešení tohoto problému a dokázali, že kvůli formulaci ToRu není možné správně identifikovat „p2p“ hrany. Proto se ve svých řešeních zaměřili pouze na „c2p“ linky a zbylé dva typy ignorovali.

Dimitropoulos a spol. pak v roce 2006 popsali svoji vlastní heuristiku [12], která kombinuje všechna předchozí řešení a dokáže s velkou přesností správně identifikovat jednotlivé typy hran. Závěrem pak zmiňme necelých pět let starou práci stejného autora [7], která vztahy mezi AS kombinuje s myšlenkami dK-distribucí. Výsledkem je algoritmus, jenž společně s velikostí grafu škáluje i typy hran.

[2] Zkratka pro Border Gateway Protocol.

3. Program GenerateTopology

3.1. Úvod, motivace

Systém CoUniverse, jenž vznikl v Laboratoři pokročilých síťových technologií Masarykovy univerzity, je aplikační middleware dovolující ovládat rozsáhlé kolaborativní prostředí skládající se z velkého množství komponent [17]. Byl navržen jako samoorganizující se systém schopný automaticky řídit libovolné aplikace (tedy i ty, které mu nejsou uzpůsobeny) a adaptovat se na změny a výpadky používané infrastruktury.

Jedním z cílů systému je podpora pro náročné datové přenosy, tedy souvislé proudy dat, jejichž objem se blíží přenosové kapacitě linek [17]. Z tohoto důvodu je součástí CoUniverse i plánovač, který je schopen tyto přenosy nad jednotlivými linkami dynamicky plánovat. Měření efektivity takového plánovače však není možno provádět během skutečného provozu, ale na vhodném modelu rozsáhlé počítačové sítě.

Pro účely testování tedy bylo potřeba vytvořit aplikaci, která by dokázala vygenerovat nejenom samotný model topologie, ale i stroje nutné pro simulaci datových přenosů.

3.2. Návrh implementace

3.2.1. Základní návrh

Vstupem celého algoritmu je vstupní graf, konstanty ovlivňující generování zpoždění a ztrátovosti paketů, velikost výsledného grafu a počet požadovaných producentů. Všechny zadávané hodnoty musí být kladná čísla, v případě posledních dvou také celá. Výstupem je pak neorientovaný graf o zadaném počtu vrcholů, v němž má každá hrana určený svůj typ (viz podkapitola 2.5.), přenosovou rychlost, zpoždění a ztrátovost paketů a stroje pro datové přenosy.

Díky poznatkům nabytým v kapitole 2 bylo možné celý problém generování rozdělit do tří samostatných celků:

- škálování iniciálního grafu,
- identifikace typů hran,
- vygenerování komponent, jejich vlastností a vlastností hran.

Každý tento problém je možno řešit samostatně a výstup jednoho použít coby vstup dalšího.

Uvedený způsob řešení však není jediný použitelný. Alternativně lze využít algoritmus popsaný v materiálu [7]. Obě metody využívají podobné prostředky (dK-distribuce a určování typů hran), liší se však v pořadí jejich provádění.

3.2.2. Škálování grafu

První podproblém definovaný v návrhu aplikace má za úkol z iniciálního grafu vytvořit jeho dK-distribuci, tu naškálovat na požadovanou velikost zadanou uživatelem a pomocí ní vytvořit graf nový, jenž bude tuto distribuci splňovat.

Výsledek škálování je tedy závislý na vstupním grafu, z něhož je iniciální dK-distribuce vytvořena. Pokud `GenerateTopology` na vstupu žádný graf neobdrží, využije jeden z pětice grafů obsažených v aplikaci Orbis[3], což je nástroj napsaný v jazyce C pro generování a škálování topologií. Konkrétně byl vybrán ten nejmenší z nich, HOT graf o velikosti 939 vrcholů.

Pro účely škálování byla vybrána 2K-distribuce[4]. Ta udává pravděpodobnost, že mezi dvojicí uzlů konkrétních stupňů vznikne hrana. Byla zvolena pro svoji snadnou implementovatelnost a dobré vlastnosti při reprodukování většiny důležitých vlastností daného grafu [10]. Krom toho bylo možné využít již existující algoritmus pro škálování definovaný v [11], jehož implementuje dříve zmiňovaný nástroj Orbis.

Logicky se však nabízí otázka, proč nejít v tomto směru ještě dál a pro výpočet nevyužít 3K-distribuci, která dokáže reprodukovat všechny důležité vlastnosti pro topologie sítí [10]. V jejím případě se určuje pravděpodobnost pro každou trojici vrcholů a komponentu, kterou mezi sebou vytváří (buď úplný graf nebo cestu). Avšak zatímco postupy pro vytváření náhodných grafů splňující určitou 3K-distribuci existují, zatím nebyl objeven algoritmus, jenž by byl schopen provádět v rámci této distribuce škálování na libovolnou velikost.

[3] http://www.sysnet.ucsd.edu/~pmahadevan/topo_research/topo.html
[4] Parametr *d* je tedy roven dvěma.

3.2.3. Určování typů hran

Po vygenerování základního grafu jsou identifikovány typy jednotlivých hran. Vytvoření vztahů mezi uzly grafu (viz podkapitola 2.5.) je důležité například pro vývoj pokročilejších síťových protokolů, které tato data používají pro zvýšení své efektivnosti [18]. Výstup této části je navíc následně použit při generování přenosových rychlostí. `GenerateTopology` při řešení tohoto podproblému implementuje algoritmus navržený autory [12].

3.2.4. Generování vlastností a strojů

Posledním podproblémem je samotné generování vlastností pro všechny hrany a vytvoření strojů pro modelování náročných datových přenosů. Celý proces se dá shrnout do několika bodů:

- identifikace podsítí podle „s2s“ hran,
- generování přenosových rychlostí pro tyto podsítě a hrany „p2c“ a „p2p“,
- určení zpoždění a ztrátovost paketů,
- vygenerování strojů a jejich parametrů.

3.2.4.1. Řešení rychlostí

Nejprve je nutno identifikovat všechny podsítě vzniklé díky přítomnosti hran typu „s2s“ a určit jejich „vlastníky“. Těmi se stávají uzly s největším stupněm a do jejich podsítě jsou začleněny všechny vrcholy, které jsou z vlastníků dosažitelné pomocí těchto hran. Celou situaci si lze představit například tak, že se jedná o páteřní síť nějaké společnosti, která propojuje jednotlivé pobočky (= vrcholy dosažitelné po „s2s“ hranách) ve městě.

Po určení všech podsítí je možno přejít k určování rychlostí. Prvním možným řešením by bylo použití náhodného generování založené na stupních incidentních vrcholů každé hrany.

Toto na první pohled jednoduché řešení však v sobě skrývá několik problémů. Především je to fakt, že se ve své podstatě jedná pouze o přímočarý postup závislý na velikosti daných uzlů, což neodpovídá reálnému prostředí. V něm jsou totiž nejdůležitější požadavky zákazníků na provozovatele sítí [19]. Bylo tedy potřeba přijít s propracovanějším řešením, nejlépe s takovým, které ke generování využívá poznatky z reálných dat.

Coby předloha ze skutečného světa byla zvolena síť European National Research and Education Network[5] (dále ENREN). Stalo se tak nejenom kvůli relativně dobře zpracovaným datům (byť pochází z roku 2011), ale také kvůli tomu, že počtem vrcholů (1157) přibližně odpovídá implicitnímu vstupnímu grafu (939, viz podkapitola 3.2.2.). Samotná data pak pochází z internetového portálu Topology Zoo[6], kde jsou k dispozici ve třech formátech pro zápis grafů – GML, GraphML, DOT – a v podobě klasické matice sousednosti.

```
node [
  id 829
  label "Brno"
  GeoLocation "Czech Republic"
  Network "SANET"
  Country "Czech Republic"
  NetworkDate "2008"
  Longitude 16.60796
  Internal 1
  Latitude 49.19522
  type "PoP"
  asn "2607"
]
```

```
edge [
  source 297
  target 308
  LinkSpeed "20"
  LinkLabel "20Mb/s"
  LinkSpeedUnits "M"
  LinkSpeedRaw 20000000.0
]
```

Obr. 1: Příklad zápisu vrcholů a hran ve formátu GML

Ne všechny linky byly ohodnoceny tak podrobným způsobem jako ukazuje Obr. 1. Některým chyběly pro generování kritické atributy o přenosové rychlosti. Nemuselo to však pokaždé znamenat problémy. Například u linek sítě NORDUnet[7] šlo informace pomocí oficiálních stránek poměrně snadno zpětně dohledat a správně určit jejich rychlost. Bohužel, část hran měla vyplněné údaje pouze o incidentních vrcholech. U těchto případů nezbylo nic jiného než je pro následné určování poměrů mezi přenosovými rychlostmi ignorovat (viz tabulka 1 a 2).

Výběr hranic mezi nimi probíhal stejným způsobem jako výpočet konstant k_{low} a k_{high} při škálování 1K-distribuce (viz podkapitola 4.4.2.). Od tohoto dělení se odvíjela práce s linkami.

Každá byla podle svých incidentních vrcholů a přenosové rychlosti zařazena do jedné z možných kategorií definovaných ve tvaru „kategorie stupně, kategorie stupně, rychlost“ (tedy například „nízký, nízký, 1 až 9 Gb/s“, viz příloha A). Toto rozdělení pak bylo využito při vytváření následující tabulky, jenž udá-

[5] Její součástí je i český CESNET.
[6] http://www.topology-zoo.org/eu_nren.html
[7] Zkratka pro Nordic Infrastructure for Research and Education.

vá počet incidentních hran u každého typu vrcholů a jejich přenosovou rychlost. Pokud linka spojuje uzly ze stejné kategorie, je započítána dvakrát. Otazník označuje hrany, u nichž se nepodařilo identifikovat jejich rychlost.

	nízký	**střední**	**vysoký**
1 až 99 Mb/s	316	148	70
100 až 999 Mb/s	231	98	25
1 až 9 Gb/s	487	191	6
10 a více Gb/s	245	185	0
?	591	331	4
celkem	1 870	953	105

Tab. 1: Základní rozřazení linek ENREN

	nízký	**střední**	**vysoký**
1 až 99 Mb/s	24,7%	23,8%	69,3%
100 až 999 Mb/s	18,1%	15,6%	24,8%
1 až 9 Gb/s	38,1%	30,7%	5,9%
10 a více Gb/s	19,1%	29,7%	0%

Tab. 2: Pravděpodobnosti použité pro generování rychlostí

Na základně získaných dat je pak možno přejít k určování přenosových rychlostí jednotlivých hran. Pro každý typ je však nutno užít trochu jiného přístupu.

V případě „s2s“ nepoužíváme incidentní vrcholy dané hrany, ale její příslušnost do podsítě konkrétního uzlu. Princip vychází z předpokladu, že je daná podsíť spravována vlastníkem, jenž linky sám buduje. Ten je pak pod tlakem ze strany svých zákazníků, kteří od něj vyžadují určitou kvalitu poskytovaných služeb [19]. Tím pádem je pro něj z ekonomického hlediska výhodnější vybudovat v rámci své páteřní podsítě spojení určité rychlosti i do uzlů nižšího stupně než čelit riziku, že za nějaký čas (například při rozšiřování daných vrcholů) bude muset investovat další peníze do přestavby tohoto úseku.

Uvažme tedy situaci, kdy je „správcem“ některé z podsítí uzel střední velikosti. V takovém případě se všechny linky spadající do takovéto podsítě ohodnotí na základě pravděpodobnosti, jakou přenosovou rychlost má linka v síti ENREN (viz tabulka 2).

Obdobným způsobem jsou řešeny i hrany typu „p2c“. Místo příslušnosti do určité podsítě se generátor zaměří na stupeň zákazníka. Opět se tedy pomocí tabulky 2 vybere příslušná rychlostní kategorie a pak se podle velikosti poskytovatele dogeneruje konkrétní hodnota. Algoritmus se tímto způsobem snaží simulovat chování skutečného zákazníka. Dává totiž smysl nechat si od budoucího poskytovatele vybudovat linku přiměřenou své velikosti a potřebám [19].

Jako poslední jsou určeny rychlosti linek „p2p“. Oba vrcholy jsou si v tomto případě rovnoceny, tudíž nelze použít metody nastíněné v předchozích odstavcích. Místo toho využijeme jemnější dělení linek sítě ENREN podle dvojic stupňů (viz příloha A). Každou hranu pak na základě incidentních uzlů zařadíme do jedné z těchto kategorií, pomocí čehož vygenerujeme její přenosovou rychlost.

3.2.4.2. Řešení zpoždění a ztrátovosti

Stejným způsobem lze přistoupit také ke generování zpoždění a ztrátovosti paketů. Vzhledem k tomu, že v tomto případě není potřeba rozlišovat mezi jednotlivými typy hran, je generování o něco jednodušší. Bohužel se však nepodařilo najít stejně podrobná reálná data jako v případě přenosových rychlostí, tudíž je `GenerateTopology` odkázáno na normální rozdělení okolo pevně zvolených konstant a standardních odchylek.

Pokud uživatel nevyužije možnosti je při spuštění generátoru změnit, jsou využity následující předdefinované hodnoty:

	konstanta	**standardní odchylka**
zpoždění	19ms	5
ztrátovost paketů	0,5%	0,15

Tab. 3: Předdefinované konstanty pro zpoždění a ztrátovost

Obě konstanty byly voleny na základě reálného prostředí. U zpoždění je to průměr evropské části počítačové sítě společnosti AT&T za posledních dvanáct

měsíců [20]. Ztrátovost paketů naopak využívá monitorovacího nástroje PingER, který ve svých úvodních materiálech označuje ztrátovost v rozmezí 0-1% za dobrou [21]. Standardní odchylky pak byly v obou případech voleny takovým způsobem, aby byl rozsah možných hodnot co největší, ale aby zároveň nedocházelo ke generování záporných čísel.

S malou pravděpodobností (přibližně pod deset procent) také může dojít k vygenerování vyšších hodnot. Konkrétně jde o normální rozdělení okolo trojnásobku, respektive desetinásobku konstant uvedených v tabulce 3 (případně hodnot od uživatele). Standardní odchylky zůstávají stejné. Díky tomu je možné do sítě zanést i určité anomálie jako pomalejší linky, vysoce ztrátové linky a podobně, což lze využít při simulaci datových přenosů (nutnost se s těmito problémy vypořádat a podobně).

3.2.4.3. Generování strojů

Druhá polovina závěrečného problému se věnuje tvorbě strojů, které jsou využity při simulaci náročných datových přenosů. Každý z nich má přiřazenu jednu ze tří možných rolí [22]:

- **producent** – vytváří jednu kopii dat,
- **distributor** – směruje příchozí data, v případě potřeby vytváří jejich kopie,
- **konzument** – přijímá jednu kopii dat.

Smyslem této části je krom vytvoření zmiňovaných strojů také vygenerovat jejich vlastnosti jako výkonnost procesoru a grafické karty a dostupnou paměť pro obě komponenty.

`GenerateTopology` pracuje se situací, kdy jeden datový přenos tvoří jeden producent, jeden distributor a $(n-1)$ konzumentů. Na základě vstupu od uživatele, počtu producentů (označme jej n), je náhodně vybráno n vrcholů grafu. Celkově je tedy pro ně potřeba vytvořit n producentů, n distributorů a $n \times (n-1)$ konzumentů. Každému z nich je přidělen producent a distributor stejného přenosu a konzumenti všech ostatních. U jednoho konkrétního vrcholu je tak na konci jeden producent, jeden distributor a $(n-1)$ konzumentů.

Mimo ně se k náhodným vrcholům grafu dogeneruje i několik distributorů navíc. Ty mohou být využity například pro simulaci výpadku distributora a jeho nahrazení během přenosu.

Intervaly hodnot pro jednotlivé parametry vychází podobně jako v případě přenosových rychlostí linek z dat od reálných zdrojů. Tentokráte je čerpáno z internetového portálu CPU Benchmarks, respektive jeho verze pro grafické karty[8]. Stránky tyto komponenty hodnotí na základě vlastního testovacího programu, jehož výsledky zasílají uživatelé z celého světa. Hodnocení jsou pak vynesena do několika typů tabulek, jako například „nejlepší komponenty v poměru cena/výkon".

`GenerateTopology` pro účely generování vlastností obou komponent využívá tyto tři typy tabulek: „high-end" (v překladu výkonné), „common" (běžně dostupné) a „low-end" (s nízkým výkonem). Z každého z nich byla vybrána patnáctka reprezentantů. V případě výkonných komponent to byla patnáctka nejlepších, u dalších už bylo nutno tuto hranici posunout směrem dolů.

Je to z toho důvodu, že se některé komponenty nacházejí ve více tabulkách zároveň (například procesor Intel Core i7-4930K je jak mezi „výkonnými", tak díky své ceně i mezi „běžně dostupnými"). Během výběru tak bylo nutno zajistit, aby se reprezentanti jednotlivých kategorií nepřekrývali.

Údaje o výkonu pochází u procesorů ze dvou zdrojů. Jedná se o improvizované databáze[9], které tvoři sami uživatelé pomocí výsledků z benchmarku IntelBurnTest v2.5 (aplikace funguje i na procesorech společnosti AMD). Princip je tedy podobný jako u portálu CPU Benchmarks. V případě grafických karet jsou využity informace pocházející z internetové databáze na portálu TechPowerUp[10].

Hranice intervalů hodnot pro generování (viz tabulka 4) byly určeny na základě nejslabšího, respektive nejsilnějšího výkonu reprezentantů v dané kategorii.

	CPU	**GPU**
výkonné	100 – 120 Gflops	3 200 – 5 000 Gflops
běžné	34 – 75 Gflops	1 881 – 3 000 Gflops
nízký výkon	3 – 20 Gflops	705 – 1 425 Gflops

Tab. 4: Intervaly Gflops použité pro generování

[8] http://www.cpubenchmark.net/, respektive http://www.videocardbenchmark.net/
[9] http://www.techpowerup.com/forums/threads/processor-gflops-compilation.94721/ a https://spreadsheets.google.com/pub?key=tONE28lwT8clO8MrXe4ILpg&w=100&h=500
[10] www.techpowerup.com/forums/threads/tpus-gpu-database-updates.171000/

Generované velikosti pamětí pro CPU jsou založené na doporučeních dostupných na internetové stránce BuildComputers[11]. U grafických karet pak hodnoty pochází přímo od vybraných reprezentantů.

	CPU	**GPU**
výkonné	8 – 16 GB	2 nebo 3 GB
běžné	4 – 8 GB	2 GB
nízký výkon	2 – 4 GB	0,5 nebo 1 GB

Tab. 5: Intervaly a hodnoty použité pro generování pamětí

[11] http://www.buildcomputers.net/how-much-ram.html

4. Implementace

Tato kapitola se věnuje samotné implementaci `GenerateTopology`. Přibližuje její technické detaily a především detailně popisuje řešení každého problému definovaného během návrhu.

4.1. Technické aspekty

Program `GenerateTopology` je napsán v aktuální verzi objektově orientovaného jazyka Java. Tento jazyk byl zvolen především kvůli vzájemné kompatibilitě s prostředím CoUniverse, což byla jedna z podmínek uvedená v zadání diplomové práce. Během programování jsem využil vývojové prostředí NetBeans (verze 7.4) pro platformu MS Windows. Při psaní kódu byly dodržovány následující konvence:

- anglické názvy proměnných,
- jména všech funkcí a proměnných jsou psány malými písmeny s výjimkou víceslovných názvů, kdy je každé další slovo uvozeno velkým znakem,
- komentáře a výstupy programu jsou taktéž v anglickém jazyce,
- maximální šířka jednoho řádku kódu je sto dvacet znaků.

Pro spuštění `GenerateTopology` jsou krom samotného jazyka Java zapotřebí také tři externí knihovny: `JGraphT`[12], `Sat4j`[13] a `Commons CLI`[14]. Všechny jsou k dispozici zdarma. Jejich konkrétní využití v programu je specifikováno v následujících kapitolách.

Knihovna `JGraphT` vznikla v roce 2003 a zahrnuje v sobě řadu algoritmů a nástrojů pro realizaci objektů z teorie grafů. Základní stavební jednotkou je graf, ať už orientovaný či neorientovaný, případně multigraf nebo pseudograf. Hlavním důvodem pro použití této knihovny je snadná manipulace s vytvořenými objekty a také možnost použití vlastních definovaných tříd coby hran, respektive vrcholů. Krom toho ji používá i samotný CoUniverse.

`Sat4j` je oproti tomu nástroj pro řešení splnitelnosti booleovské formule a optimalizačních problémů. Jeho autory jsou Niklas Eén a Niklas Sörensson,

[12] http://www.jgrapht.org
[13] http://www.sat4j.org
[14] http://commons.apache.org/proper/commons-cli/

kteří navázali na svoji dřívější implementaci v jazyce C++. `GenerateTopology` ji využívá během identifikace hran typu „p2c“ (viz podkapitola 4.5.3.).

Poslední jmenovaná knihovna `Commons CLI` má na starosti zpracování vstupních argumentů od uživatele. Aktuální stabilní verze 1.2 pochází z konce letošního února.

4.2. Základní informace

Program `GenerateTopology` je možno spouštět přímo z příkazové řádky a ovlivňovat jeho běh za pomocí několika předdefinovaných vstupních parametrů, takzvaných přepínačů:

- **`-s, --size <hodnota>`**
  Uživatel pomocí něj volí velikost vygenerovaného grafu. Pokud ji nezadá, je použita předdefinovaná hodnota 1878, což je dvojnásobná velikost iniciálního grafu.
- **`-p, --prod <hodnota>`**
  Určuje počet producentů k vygenerování. Standardní hodnota je stanovena na 3.
- **`-g, --graph <cesta>`**
  Dovoluje zvolit jiný iniciální graf.
- **`-lm, --latMean <hodnota>`**
  `-ld, --latDevi <hodnota>`
  Ovlivňuje konstantu, respektive standardní odchylku normálního rozdělení pro generování zpoždění.
- **`-plm, --pacLossMean <hodnota>`**
  `-pld, --pacLossDevi <hodnota>`
  Ovlivňuje konstantu, respektive standardní odchylku normálního rozdělení pro generování ztrátovosti paketů.
- **`-h, --help`**
  Vypíše nápovědu na standardní výstup. Ta nabízí seznam všech dostupných přepínačů včetně jejich použití.

Krom tohoto způsobu lze celou aplikaci používat také coby externí knihovnu v rámci jiného projektu, jako je například CoUniverse. Každá třída má pro tyto účely definované konkrétní procedury. Mimo těch vracející důležité atributy

jako naškálovaná 2K-distribuce, vygenerovaný graf a podobně jsou k dispozici i ty vypisující informace na standardní výstup. Například třída `FindAttributes` umožňuje vypsat stroje nejenom na základě jejich role, ale i podle příslušnosti do datového přenosu.

Každý z podproblémů výpočtu rozebíraný v kapitole návrhu je v programu řešen coby jedna samostatná třída. Důvod k tomuto kroku spočívá nejenom ve volbě objektově orientovaného programovacího jazyka (a využití jeho vlastností), ale také kvůli snadné budoucí editaci. Představme si například situaci, kdy výzkum v oblasti identifikace typů hran dojde opět o něco dále a bude popsán nový a lépe fungující algoritmus, založený třeba na jiných principech než dosud. V takovou chvíli stačí příslušnou třídu nahradit novou a `GenerateTopology` může být dál používán.

Důležitou roli hraje třída `MainClass`, která obsahuje hlavní proceduru programu, `main()`, a vhodným způsobem volá ostatní třídy. Celý její průběh lze popsat následujícím pseudokódem:

```
main() {
    zpracuj vstup od uživatele

    vytvoř iniciální graf G_init
    pomocí třídy RescaleGraph jej naškáluj
        na požadovanou velikost, vznikne G_new

    předej G_new třídě ASRelations a urči typy
        jednotlivých hran

    předej G_new třídě FindAttributes a vytvoř vlastnosti
        pro všechny hrany a vygeneruj požadovaný
        počet strojů a jejich vlastnosti
}
```

Výstupem `GenerateToplogy` je pak vygenerovaný graf, jehož velikost si určil uživatel pomocí daného parametru. Jeho 2K-distribuce se shoduje s naškálovanou 2K-distribucí iniciálního grafu, každá hrana má definovaný svůj typ, rychlost, zpoždění a ztrátovost paketů. Krom toho je k náhodným vrcholům vytvořen zadaný počet producentů, distributorů a odpovídající počet konzumentů. Každý stroj má vygenerovanou výkonnost (v Gflops) a dostupnou paměť (v MB) pro svůj procesor a grafickou kartu.

4.3. Zpracování iniciálního grafu

Iniciální graf je uložen mimo výkonný kód `GenerateTopology` v samostatném textovém souboru pojmenovaném `initGraph`. Pokud chce uživatel využít svůj vlastní iniciální graf, může použít příslušného přepínače (viz předchozí podkapitola). Musí však být následujícího formátu.

Jedná se o dvojice čísel, jenž reprezentují indexy vrcholů, jeden řádek pak představuje hranu mezi nimi. Během načítání se tak nejprve přečte celý soubor, hrany a vrcholy se načtou do dvou seznamů a na jejich základě se graf zrekonstruuje. Pro jeho reprezentaci je využita třída `SimpleGraph` externí knihovny `JGraphT`. Jedná se o neorientovaný graf, v němž nejsou dovoleny multihrany a cykly délky jedna (takzvané „smyčky“).

4.4. Třída RescaleGraph

Účelem třídy `RescaleGraph` je pomocí předloženého grafu a zadané nové velikosti vygenerovat graf nový, jenž bude mít požadovaný počet vrcholů a obě 2K-distribuce se budou shodovat. Tento proces lze rozdělit do několika samostatných celků: vytvoření základní 2K-distribuce, její naškálování a vygenerování náhodného grafu splňující tuto distribuci.

4.4.1. Vytvoření 2K-distribuce iniciálního grafu

Hned po předání iniciálního grafu konstruktoru této třídy je nutno vygenerovat jeho 2K-distribuci.

Hrany grafu jsou ze všeho nejdříve na základě svých určitých vlastností rozřazeny do kategorií. V případě 2K-distribuce je touto vlastností dvojice stupňů incidentních vrcholů. Jedna kategorie tak například obsahuje všechny hrany spojující vrcholy stupně jedna a dva, další jedna a tři, a tak podobně. Po dokončení tohoto procesu je pro každou existující dvojici stupňů k_1, k_2 určena hodnota 2K-distribuce $P(k_1, k_2)$, jejíž výpočet probíhá následovně:

$$P(k_1, k_2) = \begin{cases} \dfrac{m(k_1, k_2)}{m}, & pokud\ k_1 = k_2 \\ \dfrac{m(k_1, k_2)}{2 \times m}, & jinak \end{cases}$$

kde $m(k_1, k_2)$ udává počet hran mezi vrcholy stupně k_1 a k_2 a m je počet hran celého grafu. Výstupem této části je pak uspořádaná množina dvojic (k_1, k_2) a k ní odpovídající reálné číslo $P(k_1, k_2)$.

4.4.2. Samotné škálování

Poté přichází na řadu přeškálování 2K-distribuce na požadovanou velikost. `GenerateTopology` využívá algoritmus popsaný v [11] a implementovaný v aplikaci Orbis, z níž také pochází předdefinovaný iniciální graf (viz podkapitola 3.2.2.).

Na začátku je však nutno upravit velikost, na jakou bude graf škálován. Algoritmus v [11] se totiž se zvětšujícím se poměrem mezi novou a původní velikostí čím dál více vzdaluje od požadavku uživatele. `GenerateTopology` pro převod používá následující výraz:

$$n_{nová} = n_{uživ} - \left(\frac{n_{uživ}}{n_{inic}} \times n_{uživ} \times 0{,}035\right),$$

kde $n_{nová}$ znamená velikost, která se použije pro škálování, $n_{uživ}$ je požadavek uživatele a n_{inic} je velikost iniciálního grafu. Hodnota v závorce roste přibližně stejným způsobem, jako se algoritmus v [11] vzdaluje od požadované velikosti. Tento převod je tedy proveden ve snaze nedostatek [11] napravit. Konkrétní výsledky a grafy jsou k nahlédnutí v kapitole 5.

Škálování probíhá ve dvou hlavních krocích, vytvoření a naškálování 1K-distribuce a na jejím základě přetvoření 2K-distribuce. Ze všecho nejdříve se tedy vytvoří 1K-distribuce, což může být buď přímo z iniciálního grafu nebo přepočtem skrze 2K-distribuci. Program používá první možnost, tedy pro každý stupeň k vypočítá:

$$P(k) = \frac{k_{avg}}{k} \times \sum_{k_1} P(k, k_1)$$

Kde k_{avg} značí průměrný stupeň a k_1 všechny stupně, pro něž platí $P(k, k_1) \neq 0$. Proces škálování obou distribucí probíhá stejným způsobem jako jej popisují autoři [11].

Jako první je nutno určit stupně k_{low} a k_{high}. Jedná se o hranice určující stupně, které se škálují jiným způsobem než ostatní. Musí pro ně platit následující podmínky [11]:

- k_{low} musí být zvoleno tak, aby pro stupně $k > k_{low}$ platilo $\sum_k P(k) < 0{,}1$;
- k_{high} je naopak nejmenší stupeň takový, že hodnoty $P(k)$ jsou pro $k > k_{high}$ přibližně stejné.

Škálování stupňů nacházející se pod, respektive nad těmito hranicemi je realizováno následovně:

```
// P' - naškálovaná 1K-distribuce
// n_old - původní velikost grafu
// n_new - nová velikost grafu
// round(k) - standardní zaokrouhlení

for(k : stupně grafu){
    if (k <= k_low)
        P'(k) = P(k);

    if (k >= k_high)
        P'(round(k * n_new / n_old)) = n_old / n_new * P(k);
}
```

Pro škálování ostatních stupňů je nutno nejprve vytvořit mapování mezi starými a novými stupni $Map_{old \to new}$. Tato proměnná musí při dotazu na konkrétní starý stupeň vracet seznam, neboť může být škálován na jeden nebo více nových stupňů. Je k tomu potřeba následující:

- otevřené intervaly $M = (k_{low}, k_{high})$ pro původní 1K-distribuci a $M' = (k_{low}, k_{high} \times n_{nová}/n_{init})$ pro škálovanou,
- $u = n_{init} \times \sum_{k \in M} P(k)$ udávající počet vrcholů mezi k_{low} a k_{high},
- seznam u_{sez} obsahující nerostoucí posloupnost těchto stupňů; tedy musí platit $|u_{sez}| = u$
- $u' = n_{nová} \times (1 - \sum_{k' \notin M'} P'(k'))$ udávající počet uzlů v naškálované 1K-distribuci, které budou mít stupně z M'

```
// k_new - nový stupeň v 1K-distribuci

for (j z <1; u'>){
    část1 = (((1 - (n_new / n_old)) * (j - 1)) / (u' - 1))
        + (n_new / n_old);
    část2 = u_sez[round((((u - 1) * (j - 1)) / (u' - 1))
        + 1)];

    k_new = round(část4 * část2);
    if (k_new je z M'){
            P'(k_new) = -1.0; // zatím není definováno
            Map_old->new k part2 přidej k_new;
    }
}
```

Předcházející pseudokód ukazuje, jakým způsobem k mapování starých na nové stupně dochází. Výpočet kopíruje algoritmus z [11]. Po skončení cyklu `for` je pak možno přistoupit k naškálování hodnot 1K-distribuce.

```
a1 = ((nold / nnew) * P(round(khigh * nnew / nold))) - P(klow);
a2 = (khigh * nnew/ nold) - klow;
a = a1 / a2;

b = P(klow) - a * klow;

for (knew z M')
    P'(knew) = a * knew + b;
```

Nakonec už zbývá pouze hodnoty $P'(k)$, kde $k \in M'$, normalizovat tak, aby platilo $\sum_k P'(k) = 1$, kde k jsou všechny stupně v naškálované 1K-distribuci.

Nyní je možno přistoupit k naškálování 2K-distribuce na základě výsledků škálování 1K-distribuce. Jsou k tomu za potřebí interval M a mapování $Map_{old \to new}$ mezi starými a novými stupni.

```
X = suppPinM.size();
X2 = 0;
for (k : stupně v P')
    if (k > klow && k < round(khigh * nnew / nold)
            && P'(k) > 0)
        X2++;
```

Hodnoty X a X2 vyjadřují počet nenulových $P(k)$ a $P'(k')$, kde $k \in M$ a $k' \in M'$. Algoritmus může v tuto chvíli postupovat dvěma způsoby, záleží na tom, zda iniciání graf škálujeme směrem nahoru (tedy na větší velikost) nebo dolů (na menší velikost).

```
// P - značí původní 2K-distribuci
// P' - značí naškálovanou 2K-distribuci

X_X2 = X / X2;
for (pár v P){
    for (k1 v Map_old->new(pár.levý)){
        for (k2 v Map_old->new(pár.pravý)){
            if (pár.levý v M && pár.pravý v M)
                P'(k1, k2) = X_X2 * X_X2 * P(pár);
            else
                if (pár.levý není v M
                    && pár.pravý není v M)
                    P'(k1, k2) = P(pár);
                else
                    P'(k1, k2) = X_X2 * P(pár);
        }
    }
}
```

Při škálování směrem vzhůru je nutno pro každou dvojici původní 2K-distribuce nejprve určit všechny možné dvojice nových stupňů, které díky mapování $Map_{old \to new}$ mohou vzniknout a pak pro ně vypočítat hodnotu naškálované 2K-distribuce. Její výpočet se liší podle toho, zda se původní stupně nachází v intervalu M nebo ne.

Škálování směrem dolů naopak využívá mapování nových stupňů na staré, tedy $Map_{new \to old}$. Následující pseudokód nejprve ukazuje, jak převod mezi oběma mapováními probíhá a poté 2K-distribuci škáluje směrem dolů.

```
map_new->old = new TreeMap<>();
for (k_old : map_old->new.keySet()){
    for (k_new : map_old->new.get(k_old)){
        map_new->old ke k_new přidej k_old;
    }
}
```

```
for (k1_new v map_new->old()){
    for (k2_new v map_new->old() od hodnoty k1_new){
        hodnota = 0.0;

        for (k1_old : map_new->old.get(k1_new)){
            for (k2_old : map_new->old.get(k2_new)){
                pár = new Pair(k1_old, k2_old);
                if (P(pár) != 0)
                    hodnota += P(pár);
            }
        }

        if (hodnota > 0)
            P'(k1_new, k2_new) = hodnota;
    }
}
```

Pro každou dvojici nových stupňů $(k1_{new}, k2_{new})$ jsou pomocí mapování $Map_{new \to old}$ nalezeny všechny možné kombinace starých stupňů $(k1_{old}, k2_{old})$ a sečteny hodnoty jejich $P(k1_{old}, k2_{old})$. Pokud výsledek není nulový, uloží se do naškálované $P'(k1_{new}, k2_{new})$.

4.4.3. Vygenerování finálního grafu

Cílem této části je vytvoření náhodného grafu, jenž má odpovídající počet vrcholů a jehož 2K-distribuce odpovídá distribuci naškálované v předchozí podkapitole.

```
// d(k) - stupeň uzlu k v grafu

generategrafFrom2k(n_new){
    početHran = round((novýPrůmStupeň * n_new) / 2.0);
    graf = new Simplegraf<>(Edge.class);

    hranyStupně = vygeneruj hrany podle 2K
        a rozřaď je podle incidentních stupňů;
    chybnéVrcholy = new ArrayList<>();

    for (k : možné stupně grafu){
        while (hranyStupně.get(k) není prázdné){
            novýUzel = new Vertex();

            for (i = 1; i <= k; i++){
                hrana = vyber náhodnou hranu
                    z hranyStupně.get(k);
                hranyStupně.get(k).remove(hrana);
```

```
                    přiřaď novýUzel k hraně;

                    if (hranyStupně.get(k) je prázdné
                        a d(novýUzel) != k){
                        vlož novýUzel do chybnéVrcholy;
                        break;
                    }
                }
            }
        }

        for (hrana : hranaVrcholy)
            if (hrana nemá přiřazený vrchol)
                vyber vrchol z chybnéVrcholy
                    a přiřaď ho;

        graf = vlož vrcholy;
        for (hrany : vytvořené hrany)
        {
            if (hrana lze přidat)
                graf = vlož hranu;
            else
                do chybnéVrcholy vlož incidentní
                    vrcholy hrany;
        }

        for (vrchol : chybnéVrcholy)
             pro každý stupeň vypočti kolik
                chybí do splnění stupně;

        while (v chybnéVrcholy nezbývá jeden vrchol)
        {
            náhodně vyber dva vrcholy a propoj je;
            if (splníš stupeň některého z nich)
                odstraň jej z chybnéVrcholy;
        }

        za použití JGraphT vyber největší komponentu grafu;
        return největší komponenta grafu;
    }
```

Základní myšlenka algoritmu je následující: vygeneruje se vyžadovaný počet hran pro každou nenulovou dvojici 2K-distribuce a jejich konce se na základě stupně, na nějž musí být napojeny, rozřadí do skupin. Každá hrana se tedy poté ocitne ve dvou skupinách, případně dvakrát v jedné, pokud spojuje dva vrcholy stejného stupně. Následně jsou vygenerovány uzly, které se napojují na náhodně vybrané konce z konkrétní skupiny. Je jasné, že je k jednomu vrcholu

připojeno vždy pouze tolik konců, kolik má být jeho stupeň. Tento proces se opakuje dokud nejsou všechny konce hran připojeny k nějakému vrcholu.

Postup nastolený v předchozím odstavci však v sobě skrývá několik úskalí. Během náhodného generování totiž může dojít k několika nežádoucím výsledkům. Konkrétně se může pro oba konce jedné hrany vybrat stejný vrchol (vznikla by tedy „smyčka“), případně se může pro dvě různé hrany vybrat stejná dvojice uzlů. Obě tyto sitace v reálné topologii nemohou nastat, je tedy nutno se s nimi vypořádat.

Program `GenerateTopology` tak činí během přidávání hran do výsledného grafu. Pokud narazí na smyčku nebo hranu, kterou už do grafu přidal dříve, přeskočí ji a oba uzly (respektive vrchol) vloží na seznam chybných přiřazení. Tento seznam se skládá z dvojic celočíselných hodnot, kde první složka udává interní index vrcholu, ta druhá pak předpokládaný stupeň ve výsledném grafu.

Na chybový seznam jsou také hned při generování přidávány uzly, které nedosáhnou požadovaného stupně. K této situaci může dojít velice snadno, například když se pro vrcholy stupně dva vygeneruje lichý počet hran. Poslední z nich tedy nutně bude mít přiřazenu pouze jednu hranu a musí být proto vložen do chybového seznamu. Tato nepřesnost vzniká během převádění $P(k_1, k_2)$ na počet hran, kdy je potřeba vypočtenou hodnotu zaokrouhlit.

Po přidání všech použitelných hran do grafu přichází na řadu zpracování chybového seznamu. Nejprve je u každého vrcholu vypočten rozdíl mezi aktuálním a požadovaným stupněm, označme jej x. Hodnota x udává počet hran, které je potřeba ještě k danému vrcholu připojit. Následně se začnou náhodně vybírat dvojice uzlů a pokud je to možné, přidá se mezi nimi v grafu hrana a x se u obou sníží o jedničku. Pokud klesne na nulu, je vrchol ze seznamu odebrán. Výpočet končí, jakmile v seznamu zbývá pouze jediný vrchol.

Závěrem je nutno upozornit, že takto vzniklý graf rozhodně nemusí být souvislý. Je tedy potřeba z něj vyextrahovat největší komponentu, přesněji řečeno komponentu s největším počtem vrcholů. Program `GenerateToplogy` k tomuto účelu využívá třídu `ConnectivityInspector` knihovny `JGraphT`. Po nalezení maximální komponenty jsou ty ostatní odstraněny, vrcholy a hrany se očíslují a graf je připraven k dalšímu použití.

4.5. Třída ASRelations

Každé hraně vygenerovaného grafu je v této části přiřazen jeden ze tří možných typů (viz předcházející kapitola).

4.5.1. Vygenerování cest

Ještě před určováním samotných typů je však nutno pro potřeby výpočtu zvolit množinu BGP cest. Algoritmus navržený v [12] pro tento účel využívá skutečných směrovacích cest používaných tímto protokolem. `GenerateTopology` pochopitelně nic takového k dispozici nemá a proto je nutné tyto cesty vytvořit uměle.

```
generateBGP(){
    seznamVrcholů = seznam vrcholů grafu;
    BGPcesty = seznam vygenerovaných cest;

    while (v seznamVrcholů je více než jeden vrchol){
        vygeneruj intervaly pro vrcholy z seznamVrcholů;
        vyber dva vrcholy k1 a k2;

        pomocí JGraphT nalezni nejkratší cestu;
        přidej ji do BGPcesty;
        odstraň ze seznamVrcholů k1 a k2;
    }

    return BGPcesty;
}
```

Nejprve je všem vrcholům grafu na základě jejich stupně přiřazena určitá část intervalu celých čísel začínajících od nuly (maximální hodnotou je součet stupňů všech vrcholů). Vrcholu stupně například dva je tedy přiřazen interval nula až dva, dalšímu v pořadí třeba tři až sedm, a tak dále. Je totiž pravděpodobnější, že BGP cesta začíná nebo končí u uzlu s vyšším počtem hran v grafu [12]. Jedná se tedy o výběr ruletovým kolem, který je používaný například u genetických algoritmů.

Mezi nimi je pak nalezena cesta za pomocí Dijkstrova algoritmu, kdy všechny hrany mají identickou váhu. Program pro tento účel využívá třídu `DijkstraShortestPath` knihovny `JGraphT`. Všechny vrcholy na nalezené cestě jsou pak ze seznamu pro „další kolo“ generování intervalů odstraněny. Hledání cest končí, jakmile zbývá pouze jediný uzel.

Na první pohled se může zdát, že opakované přepočítávání vygenerovaných intervalů je zbytečné. Stačilo by přece indexy již vybraných vrcholů nějakým způsobem zneplatnit a případně výběr opakovat, dokud nebude vybrán použitelný uzel. Tento přístup by byl samozřejmě také možný, jeho hlavním nedostatkem je však narůstající počet „špatných výběrů“ se zmenšující se množinou použitelných vrcholů.

4.5.2. Hrany typu „s2s“

Hledání hran typu „s2s“ je velice jednoduché. Algortmus za ně považuje všechny takové, které nejsou součástí žádné vygenerované BGP cesty.

4.5.3. Hrany typu „p2c“

Určování typu „p2c“ tkví v přeformulování celého problému na vážený problém splnitelnosti. Úkolem algoritmu je tedy nejprve z hran obsažených v BGP cestách připravit formuli v Konjuktivní normální formě, vygenerovat váhy pro jednotlivé klauzule a k řešení využít externí knihovnu Sat4j. Získané řešení pak slouží pro určení poskytovatelů, respektive zákazníků.

```
findC2Pedges(){
    for (hrana : hrany grafu)
        vypočítej f_hrana;

    for (cesta : BGP cesty)
        vygeneruj klauzule prvního typu a jejich váhu;
        vygeneruj klauzule druhého typu a jejich váhu;

    vyřeš MAX2SAT problém pomocí Sat4j;

    for (hrana : hrany grafu)
        podle x_hrana přiřaď role producenta, konzumenta;

    return true;
}
```

Jak už bylo ukázáno v [12], vytvářený vážený problém splnitelnosti je potřeba složit ze dvou typů klauzulí. První se snaží maximalizovat počet validních BGP cest, přesně v duchu staršího heuristického řešení problému Type of Relationship (ToR) od Subramaniana a jeho kolektivu [14]. Druhý typ, s nímž přišli právě autoři [12], se snaží o maximalizaci vlastní váhové funkce *f*.

Ta jako svůj vstup bere dvojici stupňů $(k_1,\ k_2)$, kde $k_2 \geq k_1$, a vrací kladné reálné číslo. Je definována následujícím předpisem [12]:

$$f(k_1, k_2) = \frac{k_2 - k_1}{k_2 + k_1} \times log(k_2 + k_1)$$

Každá hrana je ve výsledné formuli reprezentována pomocí booleovské proměnné x_i, kde i odpovídá indexu hrany v interním seznamu a může nabývat hodnot od jedničky až po počet hran v grafu.

Klauzule prvního typu jsou generovány přímo z BGP cest. Konkrétně se pro každou dvojici po sobě následujících hran h_i, h_j v rámci jedné cesty vytvoří výraz $(x_i \vee x_j)$, označme jej X_{ij}. Jeho váha je pak určena výrazem $w(X_{ij}) = 1/m \times \alpha$, kde m značí celkový počet vytvořených klauzulí typu jedna a $\alpha = 0{,}01$. Tato konstanta byla autory [12] zvolena kvůli nejrealističtější reprodukci skutečných „p2c“ hran.

Druhý typ je pak generován pro každou hranu h_i zvlášť. Pokud v rámci BGP cesty směřuje od uzlu s menším stupněm k většímu, přiřadíme ji klauzuli $(x_i \vee x_i)$, v opačném případě $(\neg x_i \vee \neg x_i)$. Váha každé z nich je pak určena výrazem $w(X_{ii}) = c \times (1 - \alpha) \times f(k_1, k_2)$, kde k_1 a k_2 označují stupně incidentních vrcholů konkrétní hrany a c je normalizační koeficient. Ten je volen tak, aby platila podmínka $\sum_i w(X_{ii}) = 1 - \alpha$.

Ještě před vyřešením optimalizačního problemu je však nutno provést určité úpravy. Externí knihovna `Sat4j` totiž podporuje pouze celočíselné váhy klauzulí. Každá z nich tedy musí být vynásobena miliardou a standardním způsobem zaokrouhlena. Stejně tak je potřeba se zbavit nulových vah. Je to z toho důvodu, že `Sat4j` využívá kladná celá čísla coby hodnotu „true“, záporná pak jako „false“ (v případně klauzulí znamená záporné číslo negaci) a samotné číslo nula není definováno.

Nulové váhy u klauzulí typu dvě tedy musí být upraveny na jedničkové. Tato drobná úprava výsledek nijak neovlivní a to díky převodu do celých čísel, kdy je mezi „skutečnými“ a „upravenými“ hodnotami dostatečně velký rozdíl.

Po vyřešení váženého MAX2SAT problému je nutno na základě získaných přiřazení hodnot true/false určit poskytovatele a zákazníky. Pokud je proměnná x_i, která reprezentuje hranu h_i, nastavena na hodnotu true, hrana směřuje od poskytovatele k zákazníkovi. Při false je to přesně opačně.

`GenerateTopology` tyto údaje zapisuje do parametru „type“ každé hrany a to ve tvaru „p2c:“, za čímž následuje jméno poskytovatele.

4.5.4. Hrany typu „p2p“

Jako poslední jsou identifikovány hrany typu „p2p“. Ta může být na každé BGP cestě maximálně jedna a musí být incidentní s uzlem maximálního stupně [12]. Hned na začátku je tedy vytvořen seznam všech potenciálních kandidátů na „p2p“ hrany, pro něž platí právě tato podmínka.

```
findP2Pedges(){
    kandidáti = všechny kandidátské hrany na p2p typ;
    max_f = maximální hodnota funkce f;
    for (hrana : kandidáti){
        vypočítej hrana_g;
        if (hrana_g < g(3, 545))
            odstraň hranu z kandidátů;
    }

    vyřeš cesty s jediným kandidátem;
    vyřeš ostatní cesty;

    zkontroluj, že každá cesta má max. jednu p2p hranu;
}
```

Každé hraně na seznamu je pak vypočtena její váha a to pomocí funkce $g(d_1, d_2)$, která je definovaná následujícím výrazem:

$$g(d_1, d_2) = 1 - \frac{f(d_1, d_2)}{f_{max}}$$

kde $d_2 \geq d_1$ a f_{max} je rovna maximu ze všech vypočtených hodnot funkce $f(d_1, d_2)$ v předešlé podkapitole. Následně jsou ze seznamu potenciálních hran vyřazeny všechny, které mají menší váhu než $g(3, 545)$. Tuto hodnotu podobně jako v případě předešlé konstanty α určili autoři [12].

Pro každou cestu je pak určen počet možných kandidátů na „p2p“ hranu a provede se následující tělo cyklu:

```
while (některá z cest má jen jednoho kandidáta){
    hrana = kandidátská hrana s nejvyšší váhou;
    hrana.setType("p2p");

    kandidátskéHrany.remove(hrana);

    // znovu přepočítej počet kandidátů každé cesty
    početKandidNaCestu.clear();
    for (cesta : BGP cesty){
        pocKandid = počet kandidátů na cestě;
        if (pocKandid > 0)
            početKandidNaCestu.put(cesta, pocKandid);
    }
}
```

Stále však ještě mohou existovat cesty, u kterých nebyla určena žádná „p2p“ hrana, například z toho důvodu, že i po skončení předcházejícího cyklu zbývají dva možní kandidáti. Je tedy nutno provést druhý pokus o určení typu „p2p“.

```
while (existují kandidáti na p2p hranu){
    for (hrana : kandidátské hrany){
        hrana_výskyt = v kolika BGP cestách hrana figuruje;
    }

    hrana = hrana s max(hrana_výskyt * hrana_váha);
    hrana.setType("p2p");

    // hrany cest obsahující hranu vymaž z kandidátů
    for (cesta : BGP cesty){
        if (cesta obsahuje hranu)
            odstraň všechny hrany cesty z kandidátů;
    }
}
```

Závěrem výpočtu se pak ještě zkontroluje, zda každá BGP cesta obsahuje maximálně jednu hranu typu „p2p“. Pokud ne, identifikace končí neúspěchem, stejně tak pokud nebyl určen typ některé z hran. V takové situaci `GenerateTopology` chybu uživateli oznámí a skončí.

4.6. Třída FindAttributes

Po určení typů hran zbývá poslední úkol – vygenerovat pro každou z nich její rychlost, zpoždění a ztrátovost paketů a nakonec vyřešit generování konzumentů, distributorů a producentů.

4.6.1. Vlastnosti hran

Úkolem této části je na základě typů hran, které byly určeny v rámci třídy ASRelations, vygenerovat rychlosti jednotlivých linek.

```
generates2s(){
    sítě = new HashMap<>();
    for (hrana : hrany v grafu){
        if (hrana není typu "s2s")
            continue;

        tmpSíť = new HashSet<>();
        tmpSíť.addAll(incidentní uzly s edge);

        while (velikost tmpSíť se mění){
            for (uzel : tmpSíť){
                for (hrana2 : hrany incidentní s uzlem){
                    if (hrana2 je typu "s2s")
                        tmpSíť.addAll(incidentní
                            uzly s hrana2);
                }
            }
        }

        vlastník = uzel nejvyššího stupně v tmpSíť;
        sítě.put(vlastník, tmpSíť);
    }
    return sítě;
}
```

Jako první jsou určeny podsítě definované hranami „s2s“. Procedura prochází jednotlivé hrany a pro každou z nich vypočte maximální množinu vrcholů dosažitelnou pouze po „s2s“ hranách. Poté z této množiny určí uzel s nejvyšším stupněm, tedy vlastníka, a podsíť si uloží.

```
// next(k) - vrací následující stupeň
// prev(k) - vrací předcházející stupeň
// kmin, kmax - minimální a maximální stupeň

generateBorders(){
    rozřaď vrcholy podle stupňů;

    dolníHranice = kmin;

    while (vrcholy pod dolníHranice
        netvoří 90% všech vrcholů){
        dolníHranice = next(dolníHranice);
    }

    horníHranice = kmax;
    while (počet vrcholů stupně horníHranice == 1)
        horníHranice = prev(horníHranice);
}
```

Jak už bylo zmiňováno v návrhu (viz podkapitola 3.2.4.1.), hledání hranic pro tři kategorie stupňů probíhá úplně stejně jako při hledání k_{low} a k_{high} během škálování 1K-distribuce. Horní hranice „nízké“ kategorie je určena pokrytím 90 % uzlů grafu. Skupina nad horní hranicí v sobě naopak zahrnuje všechny stupně, které mají pouze jediný vrchol.

Samotné generování pak využívá statistiky získané z reálných dat sítě ENREN (viz podkapitola 3.2.4.1.). V případě hran „s2s“ a „p2c“ postupuje algoritmus přesně tak, jak už bylo vysvětleno v návrhu.

Pokud tedy narazí na hranu „s2s“, zjistí, do jaké podsítě patří a pomocí vlastníka dané podsítě a poměrů definovaných v tabulce 2 určí její rychlost. V případě „p2c“ pak GenerateTopology postupuje následujícím způsobem:

```
generatep2cSpeed(){
    for (hrana : hrany grafu){
        if (typ hrany je "p2c"){
            standRychl = podle kategorie konzumenta
                urči interval základní rychlosti;

            rychlost = standRychl * kat. producenta;

            edge.setSpeed(rychlost);
        }
    }
}
```

Základní rychlostí (proměnná `defaultSpeed`) je v předcházejícím pseudokódu myšlen interval hodnot pro generování (například 1 až 99 Mb/s). Při identifikaci poksytovatele je tento interval rozdělen na tři části a pak dojde ke generování finální hodnoty. Pokud je tedy poskytovatelem vrchol z kategorie „vyšší“, bude se výsledná rychlost nacházet v intervalu 70 až 99 Mb/s, atd.

```
generatep2pSpeed(){
    for (hrana : hrany grafu){
        if (typ hrany je "p2p"){
            switch (kategorie dvojice incid. vrcholů){
            // L - nízký, M - střední, H - vysoký
                rychlost = urči rychlost linky;
            }
            edge.setSpeed(rychlost);
        }
    }
}
```

U „p2p“ hran je nejprve na základě incidentních stupňů určena jedna ze šesti kategorií (viz podkapitola 3.2.4.1.) a na základě toho pomocí náhodnostního rozdělení určena příslušná rychlost.

Jak už bylo řečeno v návrhu této části, generování zpoždění a ztrátovosti paketů je výrazně jednodušší. Jak už bylo řečeno v návrhu, je zde použito normálního rozdělení. U obou případů se s takřka devadesáti pěti procentní pravděpodobností zvolí „obvyklé“ hodnoty těchto atributů (viz tabulka 3). Zbylá procenta jsou využita pro vytvoření určitých „anomálií“, které se mohou objevit i ve skutečné síti – tedy vyšší zpoždění či ztrátovost.

4.6.2. Generování strojů

Tato část generuje data potřebná pro modelování náročných datových přenosů. Konkrétně je vytvořen požadovaný počet producentů a od nich se odvíjející počet distributorů a konzumentů. Tyto stroje jsou pak přiřazeny k náhodným vrcholům grafu.

Ještě před tím je však nutno zkontrolovat, zda počet generovaných producentů nepřekročí počet vrcholů grafu. Pokud uživatel zadá takovouto hodnotu, `GenerateTopology` jej na to upozorní a zvolí předdefinovaný počet producentů (3). Poté je určen odpovídající počet distributorů a konzumentů.

Počet dodatečných distributorů je určen pomocí náhodné hodnoty r normálního rozdělení a to tímto způsobem:

- producenti / 2; pokud $r \in$ <-0,2; 1,2>
- producenti / 3; pokud $r \in$ <-1,2; -0,2)
- 1 + náhodné celé číslo z <0; počet prod. / 3); jinak

```
// prod - počet producentů;

chooseVertices(){
    náhodně vyber prod vrcholů;

    for (vrchol : vybrané vrcholy){
        vygeneruj k vrcholu producenta a distributora;

        for (vrchol2 : vybrané vrcholy bez vrchol)
            dogeneruj k vrchol2 konzumenty pro přenos;
    }

    Vertex chosenVertex = null;
    for (počet extra distributorů){
        k náhodnému vrcholu z vybrané vrcholy
            dogeneruj extra distributora;
    }
}
```

Výběr vrcholů, k nimž budou později přiřazeny jednotlivé stroje, probíhá za pomocí náhodnostní veličiny. Ke každému vrcholu je pak vygenerován producent a distributor ze stejného datového toku a ke všem ostatním uzlům jejich konzumenti. Distributoři navíc se náhodně rozmístí na jakékoliv vrcholy.

Po přiřazení všech distributorů, konzumentů a producentů k uzlům grafu je třeba vygenerovat výkonnost a dostupnou paměť pro jejich procesory a grafické karty. V obou případech je k tomuto účelu opět využito náhodnostní rozdělení, které s více než šedesátiprocentní pravděpodobností vybere na dnešní dobu průměrně výkonnou komponentu, s třicetiprocentní šancí velmi výkonnou komponentu a jinak podprůměrnou.

```
switch (kvalita procesoru){
    case kvalitní:
        se 60% šancí vyber velkou paměť;
        s 30% šancí vyber střední paměť;
        jinak malou paměť;

    case průměrný:
        se 60% šancí vyber středně velkou paměť;
        s 30% šancí vyber velkou paměť;
        jinak malou paměť;

    case podprůměrný:
        se 60% šancí vyber malou paměť;
        s 30% šancí vyber středí paměť;
        jinak velkou paměť;
}
```

Dogenerování pamětí se však pro CPU a GPU liší. V případě procesoru se její velikost odvíjí od jeho kvality. Jinak řečeno, pokud algoritmus nějakému stroji přiřadí výkonné CPU, je daleko pravděpodobnější, že bude mít i větší operační paměť než obvyklou nebo dokonce nízkou. Díky opětovnému užití náhodnostního rozdělení však může dojít i k této situaci.

V případě grafických karet vybírá `GenerateTopology` pouze ze dvou možností, u průměrných je dokonce jediná (viz tabulka 5).

```
Network Site 1 ---------------------------------------------------------------------------
Machine{name=1: Distributor, CPU=(36 Gflops, 8 GB RAM), GPU=(944 Gflops, 1 GB RAM)}
Machine{name=1: Producent, CPU=(116 Gflops, 4 GB RAM), GPU=(3807 Gflops, 2 GB RAM)}
Machine{name=1: Consument 1, CPU=(114 Gflops, 10 GB RAM), GPU=(1108 Gflops, 2 GB RAM)}
Machine{name=1: Consument 1, CPU=(103 Gflops, 5 GB RAM), GPU=(1213 Gflops, 1 GB RAM)}
Network Site 2 ---------------------------------------------------------------------------
Machine{name=2: Distributor, CPU=(101 Gflops, 8 GB RAM), GPU=(1251 Gflops, 2 GB RAM)}
Machine{name=2: Producent, CPU=(119 Gflops, 11 GB RAM), GPU=(696 Gflops, 2 GB RAM)}
Machine{name=2: Consument 2, CPU=(46 Gflops, 6 GB RAM), GPU=(3928 Gflops, 3 GB RAM)}
Machine{name=2: Consument 2, CPU=(57 Gflops, 9 GB RAM), GPU=(852 Gflops, 1 GB RAM)}
Network Site 3 ---------------------------------------------------------------------------
Machine{name=3: Distributor, CPU=(112 Gflops, 9 GB RAM), GPU=(3356 Gflops, 3 GB RAM)}
Machine{name=3: Producent, CPU=(3 Gflops, 5 GB RAM), GPU=(1026 Gflops, 2 GB RAM)}
Machine{name=3: Consument 3, CPU=(45 Gflops, 16 GB RAM), GPU=(1311 Gflops, 2 GB RAM)}
Machine{name=3: Consument 3, CPU=(49 Gflops, 9 GB RAM), GPU=(995 Gflops, 2 GB RAM)}
```

Obr. 1: Ukázka vygenerovaných strojů

5. Vyhodnocení

Následující podkapitola se snaží o zhodnocení výsledků aplikace `GenerateTopology` z několika úhlů pohledu.

5.1. Velikost vygenerovaného grafu

Jako první je potřeba porovnat velikosti vygenerovaného grafu s hodnotou vstupního parametru od uživatele. Smysl tohoto kroku je očividný. Pokud by každá dvojice nebyla alespoň přibližně stejná, muselo by být generování náhodného grafu přepracováno nebo upraveno zpracovávání uživatelova vstupu[15].

Hodnoty uváděné ve druhém a třetím sloupci jsou výsledkem zprůměrování pěti nezávislých spuštění `GenerateTopology`, přičemž výstupy jednotlivých běhů jsou uvedeny v závorkách. Grafické znázornění následující tabulky 6 je k nahlédnutí v příloze B.

Požadovaná velikost	Vygenerovaný graf	Maximální komponenta
200	74 (74, 74, 74, 74, 74)	44 (44, 44, 44, 44, 44)
500	429 (429, 429, 429, 429, 429)	340,4 (325, 343, 349, 348, 337)
1 000	987 (987, 987, 987, 987, 987)	798,8 (788, 832, 814, 777, 783)
3 000	3 353,4 (3 350, 3 353, 3 353, 3 353, 3 358)	3 038,2 (3 109, 3 055, 3 027, 3 013, 2 987)
5 000	5 565 (5 565, 5 566, 5 565, 5 564, 5 565)	5 142,4 (5 115, 5 111, 5 227, 5 122, 5 137)
7 000	7 790,6 (7 786, 7 794, 7 791, 7 794, 7 788)	7 195 (7 209, 7 334, 7 209, 7 229, 7 174)
9 000	9 692,6 (9 687, 9 691, 9 708, 9 690, 9 687)	8 952 (8 946, 8 940, 8 956, 8 923, 8 995)

Tab. 6: Porovnání velikostí (upravený algoritmus)

Stojí za to také zmínit, jak by výsledky tohoto porovnávání dopadly, kdyby nebyl upraven vstup algoritmu škálování iniciální 2K-distribuce (viz podkapitola 4.4.2.). Podobně jako i v předchozím případě je součástí přílohy diplomové práce vynesení této tabulky do grafu, jedná se o přílohu C.

[15] Kdyby například pokaždé vznikl graf dvakrát větší, mělo by smysl vstup od uživatele podělit dvěma.

Požadovaná velikost	Vygenerovaný graf	Maximální komponenta
200	74 (74, 74, 74, 74, 74)	44 (44, 44, 44, 44, 44)
500	440,6 (443, 440, 440, 440, 440)	274,4 (323, 358, 338, 353)
1 000	1 029,6 (1 031, 1027, 1 030, 1 030, 1 030)	856 (855, 831, 869, 881, 844)
3 000	3 855,6 (3 855, 3 855, 3 854, 3 858, 3 856)	3 549,6 (3 549, 3 565, 3 581, 3 530, 3 523)
5 000	7 471,4 (7 471, 7 471, 7 472, 7 471, 7 472)	6 915,6 (6 858, 6 871, 6 909, 7 013, 6 927)
7 000	11 950,4 (11 961, 11 972, 11 941, 11 939, 11 939)	11 048,4 (11 035, 11 088, 10 992, 11 102, 11 025)
9 000	17 526,6 (17 525, 17 526, 17 528, 17 526, 17 528)	16 246,2 (16 199, 16 148, 16 368, 16 368, 16 148)

Tab. 7: Porovnání velikostí (původní algoritmus)

Z prezentovaných výsledků je tedy patrné, že úprava algoritmu pro škálování byla úspěšná. Pro prvních několik velikostí jsou výsledky obou metod velice podobné, od hranice tří tisíc vrcholů se však původní algoritmus [11] od požadované velikosti vzdaluje, zatímco upravená metoda používaná `GenerateTopology` se stále blíží požadavku uživatele.

5.2. Porovnání 2K-distribucí

Jako další je nutno porovnat naškálovanou 2K-distribuci inicálního grafu s 2K-distribucí vygenerovaného grafu. Důležitost tohoto kroku je stejně jako v předchozím případě očividná. Kdyby si nebyly navzájem podobné, byl by graf vygenerován špatně a nebyl by vhodným kandidátem na reprezentanta modelu topologie počítačové sítě.

Během získávání výsledků pro porovnání obou 2K-distribucí byla využita speciální procedura, jenž je dokáže pospojovat pomocí dvojic stupňů. Výstup jednoho řádku byl tohoto formátu:

k1_k2_2Kgrafu(k1,dk)_2Kškálovaná(k1,k2).

Podtržítka slouží coby snadný oddělovač sloupců při kopírování výsledků do kteréhokoliv tabulkového procesoru (například OpenOffice Calc nebo MS Excel). Zde byly výsledky seřazeny podle dvojic stupňů a vyneseny na graf. Vzhledem k rozsahu získaných dat je v tabulce 8 odprezentována pouze část porovnávané 2K-distribuce při škálování na velikost 2000 vrcholů.

(k_1, k_2)	Škálovaná 2K-dist.	2K-dist. grafu	(k_1, k_2)	Škálovaná 2K-dist.	2K-dist. grafu
(1, 39)	0,005876	0,005635	(5, 19)	0,000619	0,00065
(1, 43)	0,006495	0,006068	(5, 26)	0,000309	0,000217
(1, 45)	0,022267	0,018422	(5, 29)	0	0,000217
(1, 47)	0,01164	0,009753	(5, 31)	0,000309	0,000217
(1, 63)	0,015688	0,013221	(5, 32)	0,000309	0
(1, 95)	0,023785	0,020156	(6, 6)	0,013158	0,01257
(1, 179)	0,045547	0,038362	(6, 7)	0,003093	0,002818
(4, 5)	0,005061	0,004118	(6, 8)	0,003093	0,002818

Tab. 8: Porovnání obou 2K-distribucí

Na základě prezentovaných hodnot nebo grafů (příloha D, detail první čtvrtiny v příloze E) lze pozorovat, že průběh obou 2K-distribucí je velice podobný. Liší se pouze v některých hodnotách. Tyto anomálie jsou důsledkem náhodného přiřazování hran k vrcholům během generování a díky tomu vznikajícím chybám, jak bylo komentováno v podkapitole 4.4.3.

5.3. Další vlastnosti grafu

Krom velikostí grafů a 2K-distribucí lze výstup `GenerateTopology` porovnávat také prostřednictvím metrik používaných při hodnocení modelů počítačových sítí. Čtveřice vygenerovaných grafů (čísla v hlavičce tabulky udávají jejich velikost) byla porovnávána následujícími metrikami [11]:

- **avg(k)** – průměrný stupeň grafu,
- **r** – anglicky assortativity coefficient; jde o souhrnou statistiku propojenosti uzlů, nižší hodnoty indikují, že převládají hrany spojující uzly s rozdílnými stupni,
- **avg(c)** – průměrný koeficient shlukování (anglicky clustering),
- **avg(d)** – průměrná vzdálenost mezi uzly grafu.

Jako vzor pro porovnávání slouží předdefinovaný iniciální graf (viz podkapitola 3.2.2.).

	Iniciální HOT graf	500	1000	3000	7000
avg(k)	2,10	2,20	2,31	2,50	2,57
r	-0,22	-0,27	-0,22	-0,17	-0,15
avg(c)	0	0,003	0,007	0,003	0,002
avg(d)	6,81	5,43	6,34	5,54	5,29

Tab. 9: Výsledky porovnávání dalšími metrikami

Správnost výpočtu každé metriky lze ověřit porovnáním hodnot pro inicální HOT graf vůči tabulce 7 v [11]. Při porovnávání výsledků generování s původním grafem nebo výsledky v [11] můžeme pozorovat, že `GenerateTopology` tyto metriky reprodukuje poměrně přesně. Jedinou výjimkou je průměrná vzdálenost mezi dvěma vrcholy grafu, která mírně klesá, zatímco v prezentovaných výsledcích [11] mírně roste.

5.4. Poměry mezi rychlostmi

Jako poslední je nutno porovnat vygenerované přenosové rychlosti se skutečnými daty. Zde je však potřeba upozornit, že síť ENREN, jenž byla využita jako podklad pro generování, je mezinárodní sítí pro výzkum a vzdělávání. Všechny uzly jsou si tedy rovnocené a linky typu „s2s“ a „p2c“ by v ní nedávaly smysl. Poměry zastupující jednotlivé kategorie rychlostí se tedy mohou mezi vygenerovaným grafem a ENREN lišit.

Přesto můžeme učinit vhodné porovnání a to právě pomocí „p2p“ linek. Stačí vypnout generování na základě typů hran a dostáváme následující výsledky (poměry přenosových rychlostí jsou rozmístěny stejně jako v případě přílohy A, s níž lze data porovnat):

	nízké		**středí**		**vysoké**	
nízké	13,8%	16,6%	30,9%	19,6%	71,8%	24,5%
	47,8%	21,7%	27,9%	21,4%	3,7%	0%
střední	30,9%	19,6%	3,6%	4,9%	0%	30%
	27,9%	21,4%	30,8%	62,1%	40%	30%
vysoké	71,8%	24,5%	0%	30%	0%	0%
	3,7%	0%	40%	30%	0%	0%

Tab. 10: Poměry rychlostí podle generování „p2p“ linek (2 000 vrcholů)

6. Prostor pro zlepšení

Jako každý program není ani `GenerateTopology` dokonalý a tato podkapitola se snaží identifikovat alespoň pár jeho nejzásadnějších nedostatků.

6.1. Generování BGP cest

První prostor pro zlepšení nabízí generování BGP cest pro určování typů hran. Jak již bylo vysvětleno v podkapitole 4.5.1., jejich tvorba probíhá na základě proporciálně závislého náhodného výběru z množiny vrcholů a nalezení cesty mezi nimi za pomocí Dijkstrova algoritmu.

S jinými způsoby generování BGP cest bylo experimentováno již během vývoje `GenerateTopology`. Jedním z možných řešení bylo využití třídy `KShortestPaths` knihovny `JGraphT`, jenž umí nalézt až k nejkratších cest mezi dvěma uzly grafu, které jsou odstupňovány na základě své váhy (v případě BGP cest by to znamenalo délku každé z nich). Po nalezení dostatečně velké množiny použitelných cest by se pak náhodně vybrala jedna z nich.

Toto řešení, byť na první pohled vypadá rozumně a použitelně, má však jeden zásadní nedostatek. Generování cest pomocí `KShortestPaths` je totiž extrémně zdlouhavé. Už pro $k = 2$ a dvojnásobnou velikost iniciálního grafu trvalo vyhledávání takovýchto cest neúnosně dlouhou dobu (kolem čtyřicetipěti minut), tudíž byl tento způsob řešení zamítnut. Je to z toho důvodu, že třída pro výpočet používá variaci Bellman-Fordova algoritmu.

6.2. Škálování směrem dolů

Zatímco škálování na větší grafy se povedlo zpracovat velmi dobře a velikost vygenerovaného grafu se výrazně blíží hodnotě od uživatele, pro škálování dolů už to příliš neplatí. Od hodnoty 500 náhle nastane prudký pokles (viz příloha F). Velikosti pod 200 uzlů pak už nejsou příliš doporučovány, neboť při generování vzniká velice řídký graf a počet vrcholů největší komponenty ani zdaleka nedosahuje požadavku uživatele. Na této části by tedy bylo potřeba zapracovat.

Krom úpravy samotného algoritmu by se jako vhodné řešení nabízelo i užití speciálního iniciálního grafu pro škálování směrem dolů. Kdyby byl dostatečně malý, například by se počet jeho vrcholů pohyboval okolo zmiňované hranice 200, grafy o velikosti od 200 do 939 uzlů by se tvořily pomocí škálování tohoto

grafu směrem nahoru. Při tvoření menšího grafu by se pak použilo stávající škálování dolů.

6.3. Zpoždění a ztrátovost paketů

Zatímco rychlost každé linky je určena na základě skutečných dat, zpoždění a ztrátovost paketů jsou dogenerovány de facto uměle. Úkolem pro budoucí vývoj by tak mohlo být také nalézt vhodná data pro oba tyto atributy, případně je ve spolupráci s ostatními členy větší počítačové sítě (například právě ENREN) vytvořit.

7. Závěr

Možností pro další vývoj se nabízí rovnou několik. Krom odstranění nedostatků zmiňovaných v předchozí kapitole je to například detailnější rozpracování jednotlivých Autonomních systémů (dále AS).

Současná verze `GenerateTopology` totiž používá v tomto směru velice zjednodušený model, neboť za AS považuje každý uzel grafu. Budoucí verze by mohly pracovat i se složitější strukturou, skládající se z několika vrcholů a podobně. Stejně tak by se ve vygenerované topologii mohly nacházet uzly různých typů, jako routery, servery, brány a další síťová zařízení. Každý z nich by mohl mít vytvořeny své důležité vlastnosti, podobně jako tomu je u linek.

Zajímavým směrem vývoje může být také vytvoření grafického prostředí pro úvodní zadávání parametrů nebo vykreslení vygenerovaného grafu. Používaná knihovna `JGraphT` má pro tento účel vlastní rozšíření, tudíž by implementace tohoto rozšíření nemusela být příliš náročná.

`GenerateTopology` je generátor topologií počítačových sítí, který využívá nejnovější poznatky v tomto oboru. Uživatel si pomocí něj může nechat vygenerovat jakkoliv velký model počítačové sítě, který lze využít při testování nových aplikací, protokolů nebo například simulování šíření virů.

Primárním účelem však bylo vytvořit generátor modelů, jenž bude vhodný pro testování efektivnosti plánovačů náročných datových přenosů, což se povedlo. Pevně tedy doufám, že aplikace najde při vývoji systému CoUniverse své uplatnění a bude jeho vývojářům prospěšná.

Seznam literatury

[1] WAXMAN, B. *Routing of Multipoint Connections* in IEEE Journal on Selected Areas in Communications [online]. 1988. [cit. 25. května 2014]. Dostupné na: <http://www.huaxiaspace.net/academic/classes/fa00/cse202/project/08Waxman.pdf>.

[2] ZEGURA, E., CALVERT, K., BHATTACHARJEE, S. *How to Model an Internetwork* [online]. 1996. [cit. 29. dubna 2014]. Dostupné na: <http://uhost.rmutp.ac.th/tanavoot.n/Paper_References/refer_Qos_routing/17_How%20to%20model%20an%20internetwork.pdf >.

[3] DOAR, M. a LESLIE, I. *How bad is naive multicast routing?* In Proceedings of IEEE INFOCOM [online]. 1993. [cit. 20. května 2014]. Dostupné na: <http://www.huaxiaspace.net/academic/classes/fa00/cse202/project/14Boar.pdf>.

[4] DOAR, M. a NEXION, A.. *A Better Model for Generating Test Networks* [online]. 1996. [cit. 17. května 2014]. Dostupné na: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.32.9451&rep=rep1&type=pdf>.

[5] FALOUTSOS, M., FALOUTSOS, P. a FALOUTSOS, CH. *On Power-Law Relationships of the Internet Topology* [online]. 1999. [cit. 5. května 2014]. Dostupné na: <http://repository.cmu.edu/cgi/viewcontent.cgi?article=1584&context=compsci>.

[6] LI, L., ALDERSON, D., WILLINGER, W. a DOYLE, J. *A First-Principles Approach to Understanding the Internet's Router-level Topology* [online]. 2004. [cit. 5. května 2014]. Dostupné na: <http://netlab.caltech.edu/publications/topology-sigcomm04.pdf >.

[7] DIMITROPOULOS, X. KRIOUKOV, D., VAHDAT, A. a RILEY, G. *Graph Annotations in Modeling Complex Network Topologies* [online]. 2009. [cit 7. května 2014]. Dostupné na: <http://www.csg.ethz.ch/people/dimitroc/papers/D1.pdf>.

[8] BARABÁSI, A. a ALBERT, R. *Emergence of Scaling in Random Networks* [online]. 1999. [cit. 5. května 2014]. Dostupné na: <http://www.barabasilab.com/pubs/CCNR-ALB_Publications/199910-15_Science-Emergence/199910-15_Science-Emergence.pdf>.

[9] CARLSON, J. a DOYLE, J. H*ighly optimized tolerance: A mechanism for power-laws* [online]. 1999. [cit. 9. května 2014]. Dostupné na: <http://snap.stanford.edu/class/cs224w-readings/carlson99tolerance.pdf>.

[10] MAHADEVAN, P., KRIOUKOV, D., FALL, K. a VAHDAT, A. *Systematic Topology Analysis and Generation Using Degree Correlations* [online]. 2006. [cit. 6. května 2014]. Dostupné na: <http://sysnet.ucsd.edu/~pmahadevan/publications/degcor-sigcomm06.pdf >.

[11] MAHADEVAN, P., HUBBLE, C., KRIOUKOV, D., HUFFAER, B. a VAHDAT, A. *Orbis: Rescaling Degree Correlations to Generate Annotated Internet Topologies* [online]. 2007. [cit. 10. května 2014]. Dostupné na: <http://sysnet.ucsd.edu/~pmahadevan/publications/sigcomm07.pdf >.

[12] DIMITROPOULOS, X., KRIOUKOV, D., FOMENKOV, M., HUFFAKER, B., HYUN, Y., KC CLAFFY a RILEY, G. *AS relationships: Inference and validation* [online]. 2006. [cit. 25. dubna 2014]. Dostupné na: <http://www.csg.ethz.ch/people/dimitroc/papers/D6.pdf>.

[13] GAO, Lixin. *On Inferring Autonomous System Relationships* [online]. 2001. [cit. 17. května 2014]. Dostupné na: <http://babu.cs.umd.edu/class/spring2007/cmsc711/papers/gao-as-rel.pdf>.

[14] SUBRAMANIAN, L., AGARWAL, S., REXFORD, J. a KATZ, R. *Characterizing the Internet Hierarchy from Multiple Vantage Points* [online]. 2002. [cit. 20. května 2014]. Dostupné na: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2001/CSD-01-1151.pdf>.

[15] BATTISTA, G., PATRIGNANI, M. a PIZZONIA, M. *Computing the Types of the Relationships between AS* [online]. 2003. [cit. 16. května 2014]. Dostupné na: <http://infocom2003.ieee-infocom.org/papers/04_04.PDF>.

[16] ERLEBACH, T., HALL, A., SCHANK, T. *Classifying Customer-Provider Relationships in the Internet* [online]. 2002. [cit. 11. Května 2014]. Dostupné na: <ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report145.pdf>.

[17] Neznámý autor. *CoUniverse.* In Sitola [online]. 2013. [cit. 20. května 2014]. Dostupné na: <https://www.sitola.cz/CoUniverse/index.php/Main_Page>.

[18] SUBRAMANIAN, L., CAESAR M., TIEN EE CH., HANDLEY M., MAO M., SHENKER S. a STOICA I.. *A next generation inter-domain routing protocol*. In ACM SIGCOMM [online]. 2005. [cit. 18. května 2014]. Dostupné na <https://web.eecs.umich.edu/~zmao/Papers/paper-SubCae.pdf>.

[19] ALDERSON, D.,LI, L.,WILINGER, W. a DOYLE, J., *Understanding Internet Topology: Principles, Model,s and Validation* [online]. 2005. [cit. 13. května 2014]. Dostupné na: <http://authors.library.caltech.edu/1825/1/ALDieeeacmtn05.pdf >.

[20] Neznámý autor. *Global Network Latency Averages.* In AT&T Enterprise Business [online]. 2014. [cit. 19. května 2014]. Dostupné na: <http://ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html>.

[21] Neznámý autor. *Pinger.* In ICTP Science Dissemination Unit [online]. 2005. [cit. 20. května 2014]. Dostupné na: <http://sdu.ictp.it/pinger/pinger.html>.

[22] TROUBIL, P. a RUDOVÁ, H. *Media Streams Planning for Larger Groups.* In TERENA Networking Conference 2011. 2011. [cit. 1. května 2014].

Příloha A – rozdělení linek sítě ENREN

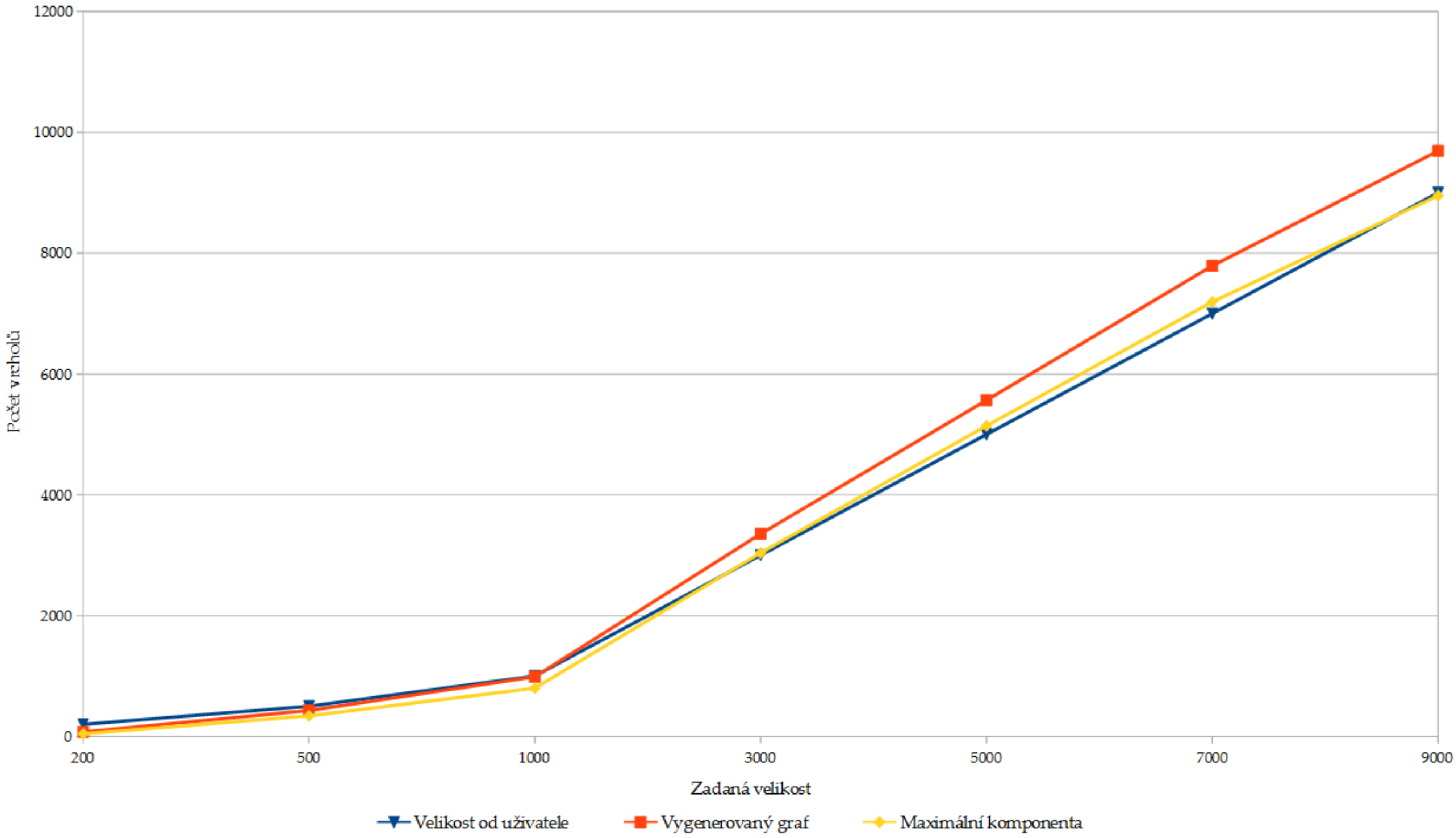
	nízké		středí		vysoké	
nízké	54	60	138	89	70	22
	176	84	133	77	2	0
střední	138	89	5	3	0	3
	133	77	27	54	4	1
vysoké	70	22	0	3	0	0
	2	0	4	1	0	0

	nízké		středí		vysoké	
nízké	14,4%	16%	31,6%	20,3%	74,5%	23,4%
	47,1%	22,5%	30,4%	17,6%	2,1%	0%
střední	31,6%	20,3%	5,6%	3,4%	0%	37,5%
	30,4%	17,6%	30,3%	60,7%	50%	12,5%
vysoké	74,5%	23,4%	0%	37,5%	0%	0%
	2,1%	0%	50%	12,5%	0%	0%

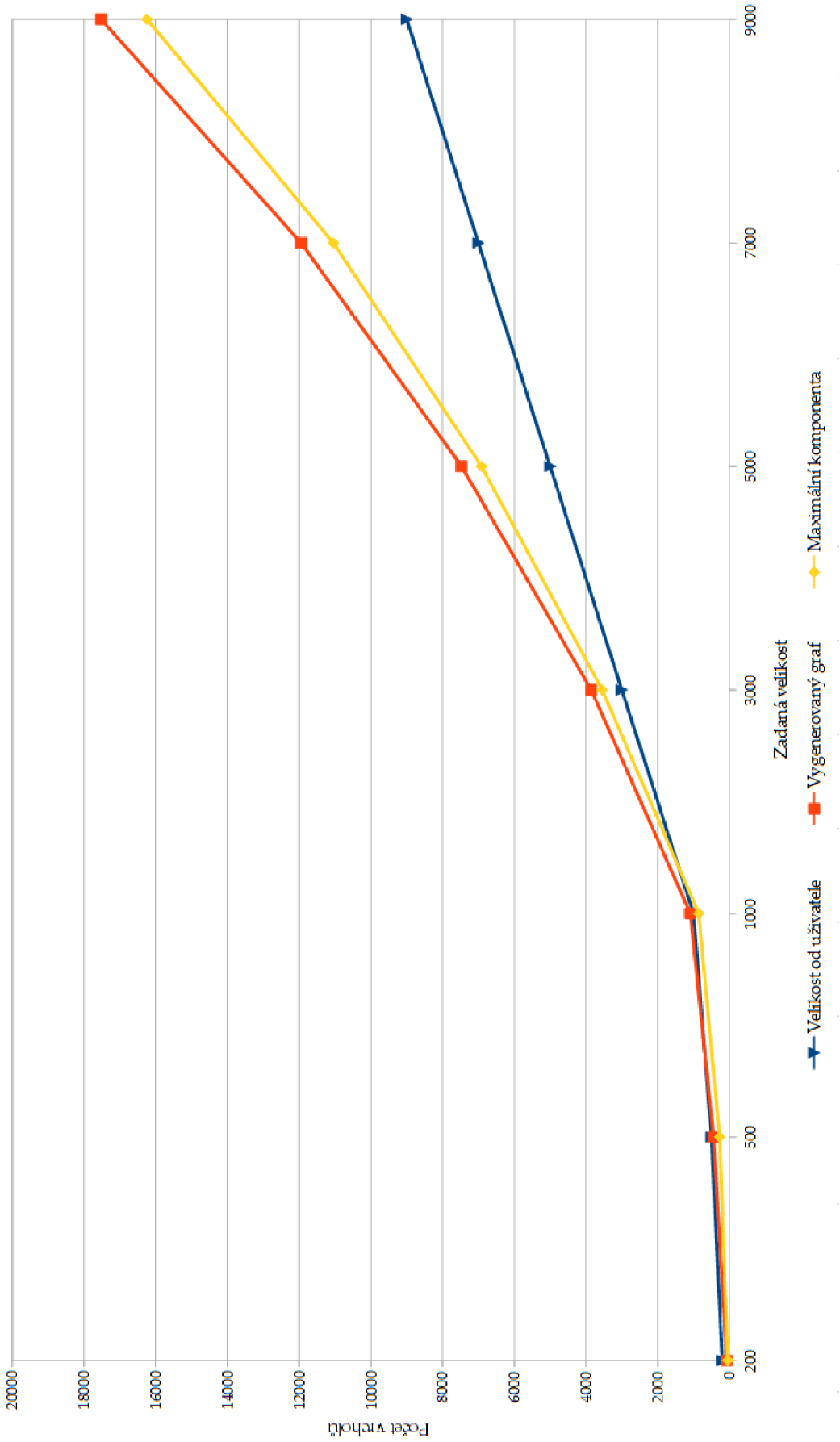
Čtveřice počtu linek a poměrů pro každou dvojici kategorií odpovídá jednotlivým přenosovým rychlostem tímto způsobem:

1 až 99 Mb/s	100 až 999 Mb/s
1 až 9 Gb/s	10 a více Gb/s

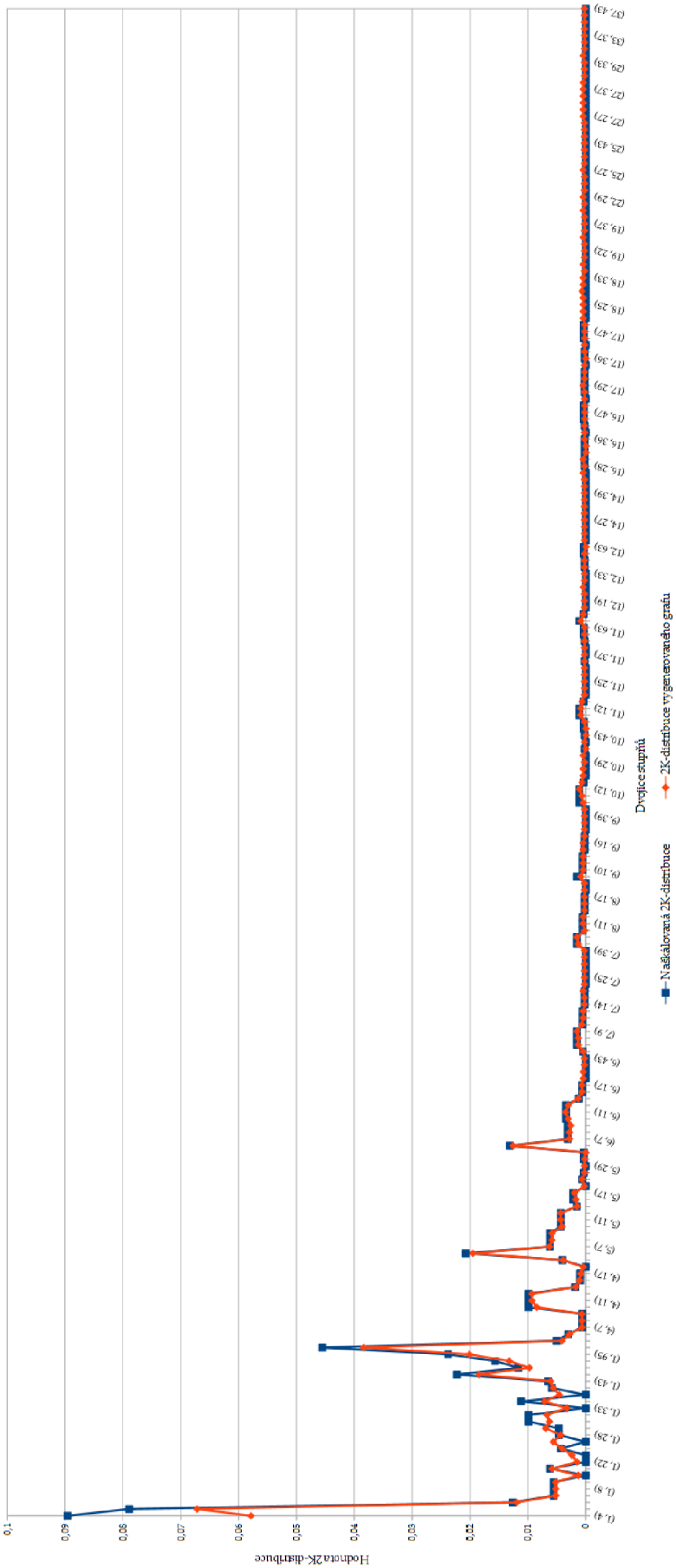
Příloha B – Porovnání velikostí (upravený alg.)



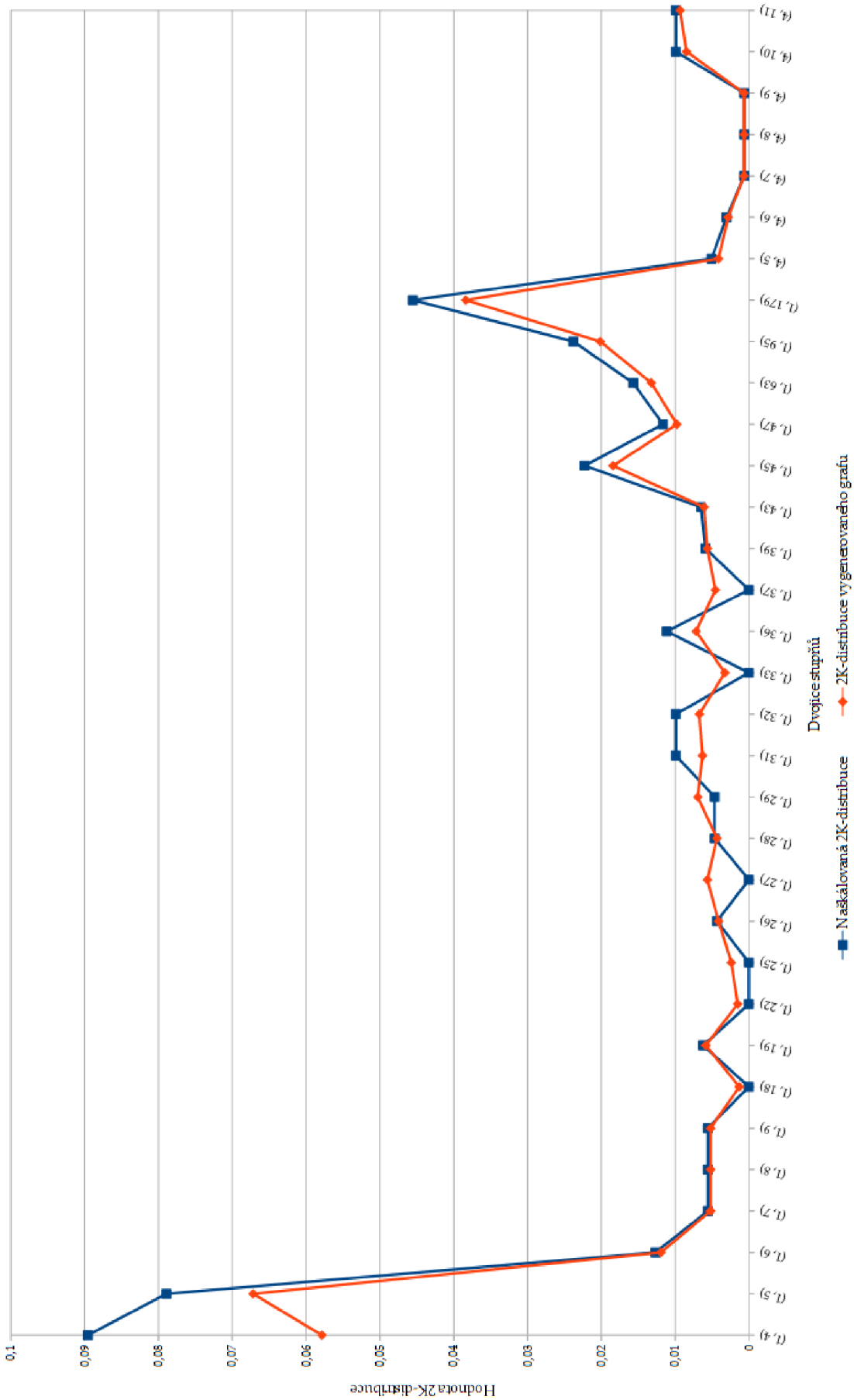
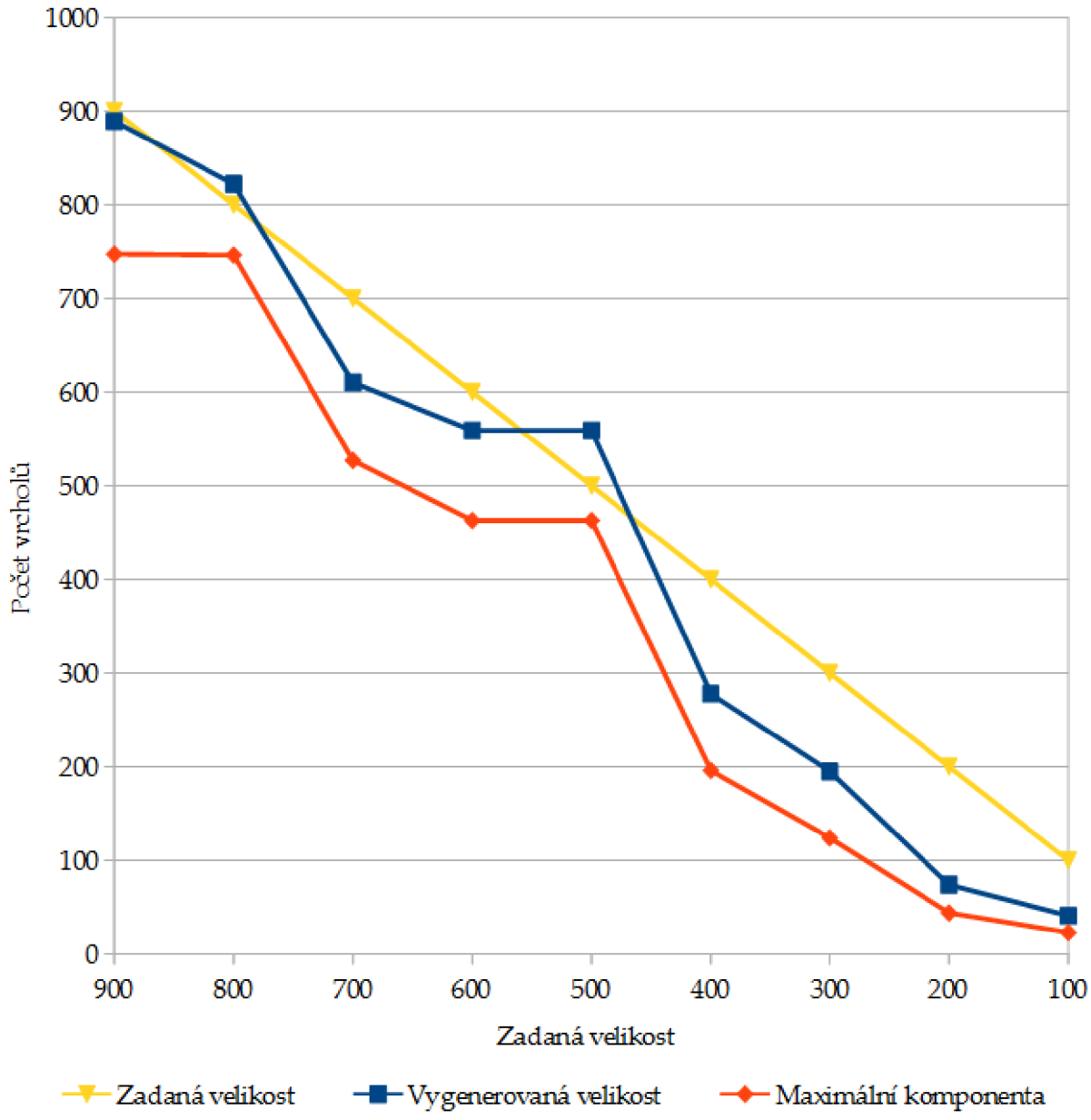
Příloha C – Porovnání velikostí (původní alg.)



Příloha D – Porovnávání 2K-distribucí



Příloha E – Porovnávání 2K-distribucí (detail)



Příloha F – Škálování směrem dolů