ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Analýza a testování protokolů pro spolehlivý multicast
Student:	Uladzislau Maher
Vedoucí:	Ing. Alexandru Moucha, Ph.D.
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

Prozkoumejte existující metody a protokoly pro spolehlivý přenos dat pomocí multicastu v sítích typu Ethernet, postavené na Cisco technologii, které se dají simulovat pomocí infrastruktury v Cisco laboratoře FIT.
Na základě výsledku zkoumání a po dohodě s vedoucím práce zvolte nejvhodnější protokoly pro spolehlivý přenos dat.
Navrhněte experiment, který umožní porovnat spolehlivost a efektivitu vybraných protokolů.
Otestujte efektivitu vybraných protokolů pro různé případy použití a možné výpadky, jako například různé datové toky velkých nebo malých souborů.
Diskutujte jednotlivé faktory, které ovlivňují spolehlivost přenosu a efektivitu protokolů.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 25. února 2017

České vysoké učení technické v Praze

Fakulta informačních technologií

Katedra počítačových systémů

Bakalářská práce

Analýza a testování protokolů pro spolehlivý multicast

Uladzislau Maher

Vedoucí práce: Ing. Alexandru Moucha, Ph.D.

10. května 2017

Poděkování

Nejprve bych velice rád chtěl poděkovat panu Ing. Alexandru Mouchovi, Ph.D. za vedení mé bakalářské práce. Dále moc děkuji svým rodičům, kteří mi umožnili studovat na této fakultě a také bych rád poděkoval svým kamarádům a spolužákům, kteří mě po dobu mého studia morálně podporovali.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. května 2017 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

MAHER, Uladzislau. *Analýza a testování protokolů pro spolehlivý multicast.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato bakalářská práce se zabývá analýzou protokolů sloužícím ke spolehlivému přenosu dat od odesílatele k více příjemcům, neboli spolehlivý multicast. Cílem práce je prozkoumat a porovnat existující protokoly řešící tento problém, najít z nich vhodné implementace nebo další programy pro spolehlivý přenos souborů pomocí multicastu. Dalším cílem je navrhnout a realizovat testovací prostředí vhodné pro vyzkoušení a následující srovnání těchto programů. Výsledkem analýzy protokolů je rozdělení probraných protokolů do skupin dle metody zajištění spolehlivosti přenosu. Závěrem testování programů je srovnání programů podle efektivity při využití v sítích se shodnými parametry s testovacím prostředím.

Klíčová slova spolehlivý multicast, analýza, testování, efektivita, srovnání, porovnání, síťové protokoly, přenos dat, redundance, síť, Cisco laboratoř, UFTP, UDPcast, MAD-FLUTE

Abstract

This bachelor's thesis is concerned with the analysis of protocols for reliable transfer of data from one sender to multiple receivers, known as reliable multicast. The goal of this thesis is to examine and compare existing protocols relevant to this problem, find their suitable implementations or other programs for reliable multicast transfer. Another goal is to design and implement a testing environment suitable for testing and following comparison of these programs. The result of the analysis of protocols divides them into groups according to the reliability of transfer they provide. The final part of programs testing is the comparison of them by their efficiency using networks with similar settings with the testing environment.

Keywords reliable multicast, analysis, testing, efficiency, srovnání, comparison, network protocols, data transfer, redundancy, network, Cisco lab, UFTP, UDPcast, MAD-FLUTE

Obsah

Seznam tabulek

Seznam obrázků

Úvod

V současné době se počet dat přenášejících v počítačových sítích zvyšuje každým dnem. Stejný obsah může být a je často vyžadován několika příjemci najednou (např. multimediální obsah, aktualizace operačních systémů, nové verze různých aplikací). Běžně používaným typem přenosu dat je unicast.

Při použití unicastu pro přenos stejného obsahu zasílá vysílač novou kopii dat pro každého z N příjemců, což je neefektivní a zvyšuje to N-krát datový tok a to má za následek přetížení sítě přenosem duplicitních dat. Ideálním řešením by bylo zasílat stejný paket dat všem příjemcům, kteří požadují stejný obsah. Takovým řešením je multicast.

Multicast umožňuje zasílaní dat z jednoho nebo více vysílačů mnoha příjemcům, ale neřeší problém zajištění doručení, tak je vhodný především pro vysílaní dat, kde občasná ztráta několika paketů nemá velký vliv na kvalitu obsahu, např. video, audio, VoIP kde *„významný vliv na kvalitu mají ztráty mezi 5 a 10% z celkového přenosu“*[14]. Ve většině případu se však posílají data, kde jediný ztracený paket může zapříčinit nečitelnost celého souboru nebo jeho části. V takových případech potřebujeme zajistit spolehlivost přenosu, aby každý příjemce dostal přesnou kopii původních dat.

První kapitola této bakalářské práce má za cíl uvést čtenáře do principu fungování multicastu v současných sítích a seznámit s problematikou spolehlivosti multicastu. Cílem druhé kapitoly je analýza protokolů pro spolehlivý multicast, v rámci níž bude prozkoumán princip fungování a způsob zajištění spolehlivosti přenosu každého z vybraných protokolů. Cílem třetí kapitoly je nalezení programů, které slouží ke spolehlivému vysílání souboru pomocí multicastu. Cílem poslední kapitoly je návrh

a realizace testovacího prostředí v rámci Cisco laboratoře, vyzkoušení programu pro následující psaní skriptů pro testování. Podle výsledků testování bude provedena analýza programu a jejich následující srovnání.

Kapitola 1

Úvod do problematiky

1.1 Úvod do multicastu

Tři hlavní typy vysíláni v počítačových sítích jsou:

- Unicast
- Broadcast
- Multicast

1.1.1 Unicast

Unicast popisuje komunikaci, kde paket je zaslán jedním zdrojem k jednomu příjemci. *„V současné době však unicast není vhodný pro všechny typy komunikace. Příkladem může být vysílání internetového rádia nebo televize, kdy je vyžadován typ komunikace ‚jeden k mnoho' (paket je zaslán jedním zdrojem k více příjemcům)*"[8]. Nicméně unicast je stále převládající formou přenosu v LAN sítích a na Internetu.[4]

1.1.2 Broadcast

Broadcast popisuje komunikaci, kde paket zaslaný jedním zdrojem přijímají všechna zařízeni v lokálním segmentu síti. Je důležité, aby směrovače omezovali šíření broadcast paketů, protože v jiném případě by se pakety ze segmentů šířily do všech ostatních segmentů síti, což by vedlo k zahlcení celé sítě. Z toho důvodu broadcast nelze použit pro přenos mezi segmenty síti.

1.1.3 Multicast

Dle [8], multicast popisuje komunikaci, kde paket je pomocí multicastové skupinové adresy současně zasílán definované skupině přijímačů. *„Vysílač (zdroj dat) odesílá pakety na multicastovou adresu (která neslouží k identifikaci příjemce, ale skupiny), zdrojová adresa je jeho normální unicastová adresa. Na routerech se pak paket odesílá do všech směrů, kde je nějaký příjemce (provádí se duplikace paketu)*“[1].

Multicast umožňuje vysíláni nejen od jednoho zdroje několika příjemcům, ale i od několika zdrojů několika příjemcům (např. dva vysílače mohou posílat různé části stejného souboru do jedné multicastové skupiny). Tato vlastnost je použita některými z protokolů pro spolehlivý multicast.

1.2 Multicast v TCP/IP sítích

Současné lokální sítě používají rodinu protokolů TCP/IP. Každé síťové rozhraní v počítačových sítích má svoji unikátní MAC a IP adresu. MAC je adresa linkové vrstvy (L2 — Layer 2) a IP je adresa síťové vrstvy (L3 — Layer 3) modelu ISO/OSI.

Unikátnost adres zajišťuje možnost unicast komunikaci. Pro účely skupinového zasíláni paketů existují speciální broadcastové a multicastové adresy. Broadcast MAC adresa je FF:FF:FF:FF:FF:FF. IP broadcastové adresy se spočítají na základě masky síti (v bitmapě adresy, v části označující adresu rozhraní v podsíti, budou nastavené všechny bity na jedničku).

1.2.1 Multicastové skupiny

Multicastové skupiny jsou skupiny přijímačů s následujícími vlastnostmi:

- Skupiny jsou „otevřené“ — může do nich vysílat i stanice, která není členem skupiny.
- Do skupiny se může přihlásit kterákoliv stanice.
- Stanice může být členem i více skupin.

Pro identifikaci skupin v IPv4, dle RFC 5771 [3], jsou vyhrazeny adresy z rozsahu 224.0.0.0 – 239.255.255.255 (224.0.0/4). Tento rozsah je rozdělen na další částí:

- Rezervovaný rozsah 224.0.0.0/24 je vyhrazen pouze pro lokální podsítě (pakety mají TTL = 1). Několik příkladů takových adres je uvedeno v tabulce 1.1.
- Administrativní rozsah 239.0.0.0/8 je vyhrazen pro použití uvnitř organizace (LAN).
- Globální multicastové adresy (Globally scoped adresses) jsou všechny zbývající. Používají se mezi organizacemi a přes internet.

Tabulka 1.1: Příklady rezervovaných lokálních multicastových adres

Address	Description
224.0.0.1	All Systems on this Subnet
224.0.0.2	All Routers on this Subnet
224.0.0.9	RIP2 Routers
224.0.0.5	OSPF All Routers
224.0.0.6	OSPF Designated Routers
224.0.0.10	EIGRP Routers
224.0.0.12	DHCP Server / Relay Agent
224.0.0.13	All PIM Routers
224.0.0.22	IGMP

IPv6 má definovány své rozsahy pro každý typ vysílání, nicméně pro účely této bakalářské práci není potřeba se zabývat přenosem v IPv6.

1.2.2 L2 multicast

V rámcích jednoho lokálního segmentu sítě (subnetu) se komunikace provádí na 2. vrstvě (L2) modelu ISO/OSI a používají se speciální multicastové MAC adresy, které se vytváří z multicastových IP adres. O tom podrobně ve článku [15] . Tyto MAC adresy mohou být použity v přepínačích pro účely zasílaní rámců do správných portů, na kterých se nacházejí příjemci z mapované IP multicastové skupiny. V realitě většina přepínačů nepodporuje tuto možnost, a proto jsou tyto rámce zaslány na všechny porty přepínače a každý přijímač na L3 vrstvě už kontroluje, jestli paket patři jemu nebo ne.

Z toho vyplývá, že v rámci běžného přepínače se multicast chová jako broadcast, tedy jsou jím zaplaveny všechny porty. Aby mohl přepínač provádět lepší rozhodnutí, tak musí analyzovat veškerý multicastový provoz na L3 úrovni a na základě analýzy paketů sestavovat tabulku kam multicast směrovat. Dnes se na to používá hlavně IGMP Snooping, který bude popsán v další sekci.

1.2.3 Multicast mezi subnety

Důležitou částí multicastu je komunikace mezi subnety, tj. mezi routery. Směrovací tabulka pro běžné unicast adresy je statická nebo se nemění tak často (záleží na protokolu pro dynamické směrovaní). Vysílači a příjemci se v multicastu dynamicky připojují a odpojují z různých segmentů sítě, čímž vzniká potřeba současně a správně řešit směrovaní mezi nimi.

V současných sítích se pro řešení teto problematiky využívají protokoly IMGP a PIM.

1.2.4 Internet group management Protocol (IGMP)

IGMP je protokol pracující na třetí vrstvě ISO/OSI modelu. Využívá se pro dynamické přihlášení a odhlášení klientů z multicastové skupiny u směrovačů v lokálním segmentu sítě. Aktuální verze protokolu je IGMPv3, ale v současné době je nejčastěji využíván IGMPv2. Hlavní rozdíl třetí verze protokolu od druhé je v možnosti filtrace paketů od skupiny přijímačů na základě IP adresy zdroje.

IGMP funguje následujícím způsobem:

- Před tím, než klient začne poslouchat v nějaké skupině (např. 224.2.2.2), pošle paket **IGMP Membership Report (IGMP Report)** na adresu teto skupiny — reportuje že chce přijímat pakety teto skupiny.
- Router dostává **IGMP Report** a od této doby si pamatuje, na jakém rozhraní jsou přijímače z této skupiny a směruje pakety pro tuto skupinu na dané rozhraní.
- Router kontroluje, jestli příjemci stále patří ke skupinám, a proto periodicky (ve výchozím nastavení každých 60 sekund) odesílá do připojených sítí dotaz **IGMP Membership Query (IGMP Query)** na adresu 224.0.0.1 s TTL=1. Ve chvíli, kdy přijímač v segmentu dostává tento dotaz, tak čeká náhodnou dobu od 0 do **Max Response Time** (uveden v doručeném **IGMP Query**) a posílá **IGMP Report**. Pokud jiný přijímač ze stejné skupiny poslal **IGMP Report** dřív, tak ostatní příjemci neposílají svůj **IGMP Report**.

- V případě, že klient chce opustit skupinu, pošle **IGMP Leave Group (IGMP Leave)** a přestane odpovídat na **IGMP Query** dotazy. Router pro jistotu pošle dvakrát **IGMP Query** dotaz na adresu skupiny a jestli po době čekaní nedostane žádný **IGMP Report** od skupiny na rozhraní, tak smaže informaci o skupině a přestane vysílat pakety pro tuto skupinu na tomto rozhraní.
- Periodicky pokračuje v posílání **IGMP Query** dotazů i v případě, že žádný klient nepatří do žádné skupiny.

1.2.5 IGMP Snooping

Současné pokročilejší přepínače přeposílají pakety do správných portů nejen na základě L2 hlavičky, ale umožňují se dívat i uvnitř rámců do vyšších vrstev. Tato vlastnost může být použita na odposlech multicastového provozu a je využita pro IGMP Snooping.

IGMP Snooping je definován v RFC 4541. Funguje na principu odposlouchaní IGMP zpráv na přepínači. Přepínač nahlíží dovnitř rámce a jestli je paket typu **IGMP Report** nebo **IGMP Leave**, tak získává informaci na jakém portu se nachází příslušná multicastová skupina. Tato metoda zatěžuje přepínač, a proto má smysl jí používat při podpoře hardwarové akcelerace.

Jak víme, v případě že jeden přijímač ze skupiny odesílá **IGMP Report**, tak ostatní přijímače ze skupiny neodesílají svůj **IGMP Report**, protože vidí jiný **IGMP Report** ve skupině. Tady nastává problém pro IGMP Snooping, protože v takovém případě přepínač neví, kde jsou ostatní členové skupiny.

IGMP Snooping řeší tento problém a existují dva způsoby:

- Pasivní IGMP Snooping (IGMP querier) — přepínač poslouchá provoz a omezuje přenos **IGMP Report** paketů mezí přijímači. V takovém případě budou k směrovači zaslaný **IGMP Reporty** ze všech přijímačů, čím řešíme tento problém, ale zatěžujeme linku mezi směrovačem a přepínačem.
- Aktivní IGMP Snooping (Proxy reporting) — přepínač navíc filtruje IGMP pakety za účelem snížení počtu dotazů na směrovač: *„Vytvoří tak pro směrovač zdání, že je směrem k němu pouze jeden příjemce, přestože jich může být i více. Pokud jsou za přepínačem dva přijímače a jeden se odhlásí, pak se směrovači neodesílá žádná informace. V následujícím dotazu směrovače na členství ve skupině přepínač propustí informaci od posledního zbylého přijímače ve skupině, že je členem dané skupiny, aby provoz nevypnul“*[9].

1.2.6 Protokol Independent Multicast (PIM)

PIM je protokol pracující na třetí vrstvě ISO/OSI modelu. Používá se pro směrovaní multicastového vysílání v IP sítích, tedy pro směrování od vysílače či vysílačů k řadě přijímačů z různých segmentech síti. Hlavním účelem je sestrojit **strom nejkratších cest**. Na to existují 4 režimy PIM:

- Hustý režim (Dense Mode) — RFC 3973.
- Řídký režim (Sparse Mode) — RFC 4601.
- Obousměrný řežím (Bidirectional) — RFC 5015.
- Režim specifický na zdroji (Source-specific multicast) — RFC 3569.

Nejčastěji se ale využívá **Dense Mode** nebo **Sparse Mode**, které jsou popsané dál.

1.2.6.1 Dense Mode

Dense Mode (PIM-DM) je vhodný pro následující případy:

- Příjemci i vysílače jsou blízko.
- Vysílačů je málo a příjemců mnoho.
- Probíhá intenzivní datový tok.

Dle [1], Dense Mode vychází z představy, že téměř všichni chtějí provoz přijmout, takže jej odesílá do všech směrů (na všechny routery mimo toho, od kterého přišel). Pokud některý sousední router provoz nepotřebuje, tak to musí oznámit tzv. **Prune** zprávou. Každé 3 minuty router znovu začne vysílaní do všech směru a ořeze větve, kde nejsou příjemci. Tato metoda se nazývá **flood and prune**.

1.2.6.2 Sparse Mode

Sparse Mode (PIM-SM) se využívá pro sítě:

- S malým počtem přijímačů.
- S větším rozptylem přijímačů.

„Vychází z představy, že klienti, kteří chtějí přijímat multicast, se v síti nachází velmi řídce. Takže Sparse mode posílá provoz pouze routerům, kteří si o něj požádají“ [1]. Routery žádají o připojení pomoci **Join** zprávy a pro odpojení pomoci **Prune** zprávy.

PIM-SM sestavují jednosměrné sdílené stromy s kořenem v RP (Rendezvous Point). RP je router, na který vysílají zdroje a ze kterého se sestavují cesty do každého přijímačů, kam dál RP posílá pakety.

1.3 Spolehlivé vysílaní

V této bakalářské práci pod pojmem „spolehlivý přenos dat“ rozumíme takový přenos, kde pří ztrátě častí dat bude tato část poslána znovu. Na konci přenosu by všichni příjemci měli mít stejnou kopií původních dat.

Pro zajištění doručení dat v libovolných typech vysílaní potřebujeme detekovat a opravovat chyby. Oprava chyb při komunikaci může být realizována dvěma různými způsoby:

- *Opravou chyb bez zpětné vazby (Forward Error Correction, FEC):* vysílají se speciální datové pakety, které obsahují redundantní informaci, na základě které se dají opravit datové pakety. Pakety pro FEC se spočítají na základě Reed-Solomonových samoopravných kódu, jak je popsáno v [16, 20].

- *Zpětnou vazbou s automatickým opakováním (Automatic Repeat Request, ARQ):* opravné pakety se posílají jen při ztratě. V tomto případě musí příjemce komunikovat s vysílačem pro opravu paketů. V multicastu navíc existuje možnost využiti paritních paketů pro ARQ metodu. Tím se umožňuje opravit pakety u několika příjemců, na základě jejich ostatních dat, pomocí menšího počtu opravných paketů.

1.4 Problematika spolehlivého multicastu

V minulých sekcích jsme popsali, jak se řeší zasíláni a směrovaní multicastového provozu. Tyto operace se provádějí na druhé a třetí vrstvě modelu ISO/OSI, a sestavují cesty mezí přijímači a vysílači v multicastové skupině.

Pro multicast na čtvrté vrstvě (L4) ve skupině protokolu TCP/IP máme možnost využívat jenom protokol UDP, který nezajišťuje ani spolehlivost přenosu, ani doručení paketů ve správném pořadí. Proto je potřeba řešit problém spolehlivého přenosu buď jiným L4 protokolem nebo protokolem ve vyšších vrstvách nad UDP. Dále si popíšeme, s jakými problémy se, v porovnaní s unicast komunikací, kde se používá protokol TCP, ve spolehlivém multicastu setkáváme.

1.4.1 ACK-implosion

TCP využívá ACK potvrzováni, kde přijímač pro každý paket dat od serveru pošle paket s potvrzováním. Naivní implementace spolehlivého multicastu s pozitivním potvrzováním (ACK) vytváří několik problémů.

Jeden z problémů je zahlcení kanálu od příjemců k serveru, protože pro každý vyslaný paket dat musí všichni příjemci poslat unicast ACK zprávu serveru, což s růstem počtu příjemců zvýší i počet těchto zpráv, přičemž pro velký počet příjemců muže být datový tok od klientů i mnohokrát větší, než od serveru ke klientům. Tento problém se nazývá **ACK-implosion**.

1.4.2 Neznámý počet příjemců

Další problém spočívá v neznámém počtu příjemců, protože server neví, kolik ACK zpráv musí doručit. I v případě, že je počet známý, nastává problém s odpojením klientů, kde všichni příjemci musejí čekat, než dojde k detekci odpojeni klienta. Na druhou stranu, dynamické připojení klientů po začátku vysílaní není problém spolehlivého multicastu, ale protokolů vyšších vrstev (obvykle to řeší tzv. Session Manager).

1.4.3 NACK-impolsion

Jedno z možných řešení spočívá v používaní negativního potvrzovaní (**NACK**). Na rozdíl od ACK, příjemci posílají NACK jen při ztrátě nebo poškozeni paketu. Při ztrátě velkého počtu paketů ale stejné nastává problém se zahlcením kanálu (**NACK-impolsion**). Většina protokolů s NACK potvrzováním řeší tento problém.

Kapitola 2

Analýza existujících protokolů

Na začátku zkoumaní bylo pro účely této bakalářské práce řešeno zvolit protokoly, které jsou popsáné v literatuře a/nebo mají přesnou specifikaci v RFC. Existuje spousta protokolů pro spolehlivý multicast, např. v přednášce [10] jsou uvedeny:

- SRM — Scalable Reliable Multicast (1996)
- RMTP — The Reliable Multicast Transport Protocol (1996)
- RLM — Receiver driven Layer Multicast (1996)
- RMDP — Reliable Multicast data Distribution Protocol (1997)
- PGM — Pragmatic General Multicast (2003)
- FLUTE — File Delivery over Unidirectional Transport (2002)
- NORM — NACK Oriented Reliable Multicast (1999)
- MDP — Multicast Delivery Protocol
- XCAST
- IRMA — (1999)
- TCP-XM

Ne všechny protokoly ze seznamu splňují tuto podmínku, a proto pro podrobné zkoumání v této kapitole byly vybrány následující protokoly: RMTP, PGM, SRM, FLUTE.

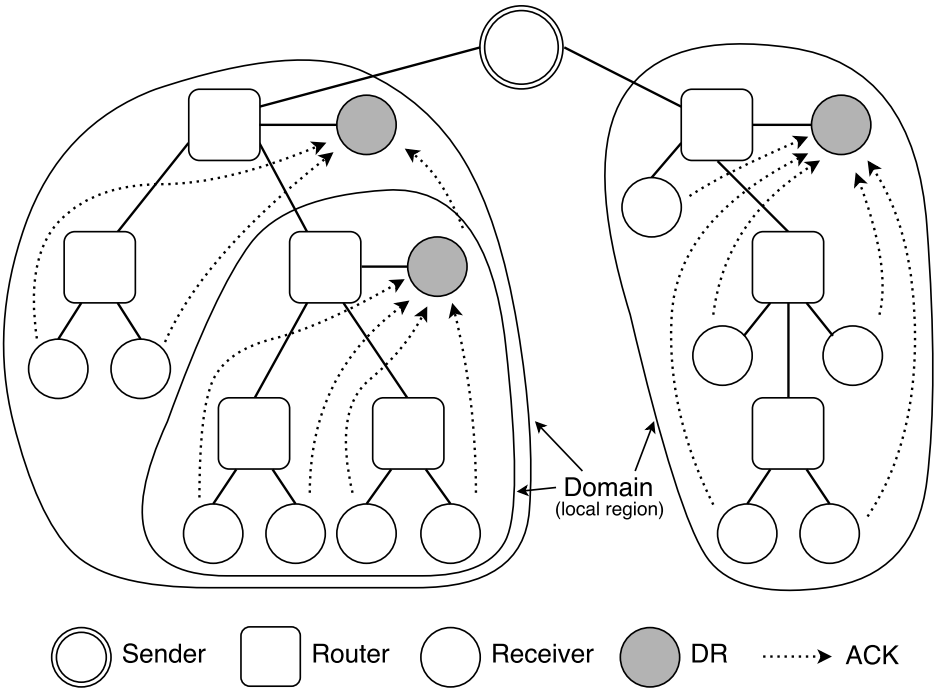
2.1 RMTP — The Reliable Multicast Transport Protocol

Dle [19], RMTP — je transport-layer (L4) protokol umožnující spolehlivé vysílaní z jednoho zdroje mnoha příjemcům. Protokol zajišťuje spolehlivost přenosu metodou ARQ s pozitivním potvrzováním (ACK). Je navržen se zaměřením na vysílaní velkých souboru (dokumenty, software, atd.) pro velké skupiny příjemců široko rozšířených po Internetu.

Dále se podíváme na architekturu a charakteristické rysy protokolu RMTP.

2.1.1 Architektura

Architektura RMTP má za účel snížit/odstranit **zahlcení síti ACK zprávami (ACK-implosion)** a snížit **latenci mezi přijímači a zdrojem dat (end-to-end latency)**.



Obrázek 2.1: Architektura RMTP

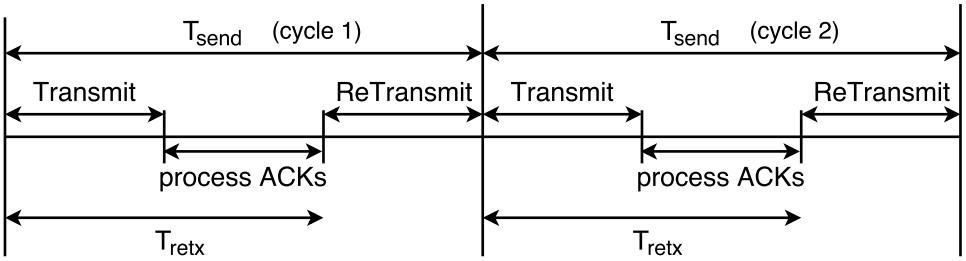
Na obrázku 2.1 je vidět, že skupina přijímačů je rozdělena na lokální regiony (local regions) nebo jinak řečeno domény (domains). V každé doméně je speciální přijímač, který se nazývá Designated Receiver (DR). DR cachuje přijímané pakety, zpracovává ACK odpovědi od přijímačů v jeho podstromě a odesílá ACK odpovědi dál.

Pro každý blok dat odešlou všechny přijímače unicast ACK zprávu pro příslušný DR v podstromu. DR zpracuje ACK odpovědi a pro každý blok dat pošle pravě jednu ACK zprávu dalšímu DR směrem k serveru nebo, v případě, že žádný další DR není na cestě k serveru, přímo serveru. Tím se snižuje nejen počet ACK zpráv, které server musí zpracovat, ale se i zvyšuje efektivita využití prostředku síti.

Kromě toho, jak bylo uvedeno, DR navíc cachuje správně doručené pakety a, v případě, že došlo k ztratě paketu u přijímače v lokálním regionu, pošle příslušný opravný paket multicastové skupině a směrovač omezí distribuci paketu mimo lokální region. Takto opravíme paket bez vlivu na server a přijímače mimo naší doménu a navíc snížíme end-to-end latency.

2.1.2 Implementační rysy

Server rozděluje data na pakety stejného rozměru typu *DATA*, s výjimkou pro poslední paket, který muže mít jinou velikost a je typu *DATA_EOF*. Navíc každý datový paket má svoje sekvenční číslo (první paket posloupnosti má nulu).



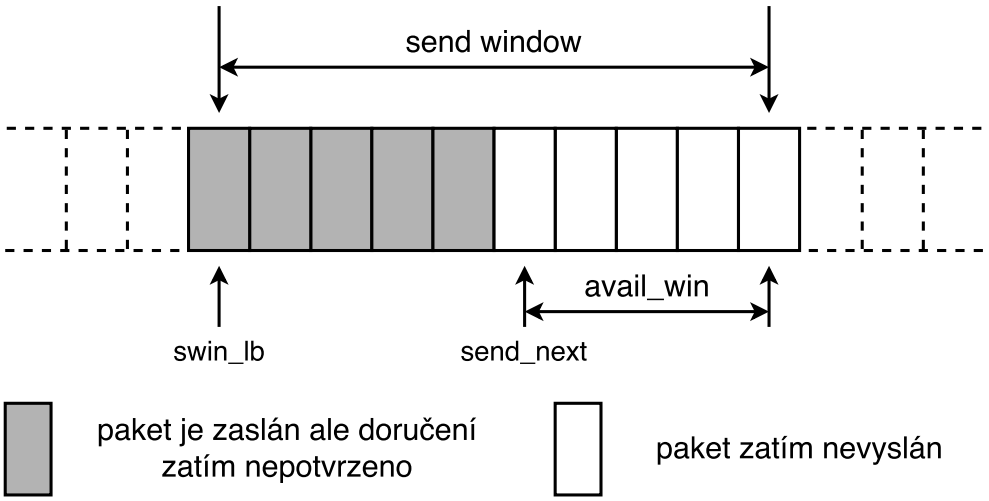
Obrázek 2.2: RMTP operace vysílače

RMTP vysílač (sender) pracuje v cyklech, jak je ukázáno na obrázku 2.2. Každý krok cyklu trvá T_{send} času, za který jsou prováděny následující operace:

1. *Vysílaní (transmit):* Na obrázku 2.3 je vidět, že vysílač postupně vysílá pakety z okénka (*send windows*). Počet paketů v okénku je dán parametrem vysílaní W_s. Když *avail_win* pakety z okénka dojdou, začne následující operace.

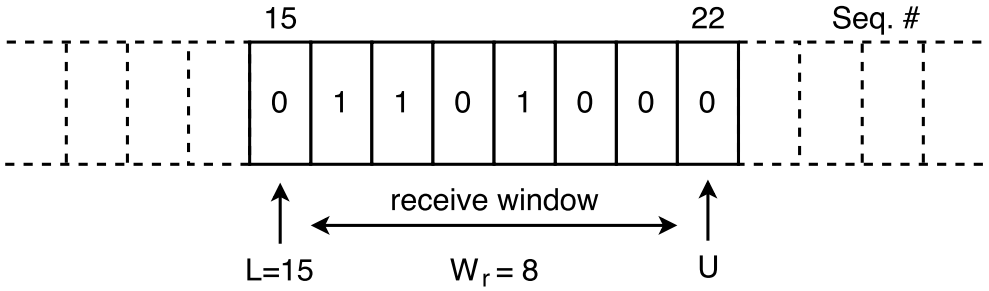
2. *Zpracování ACK zpráv (process ACK):* Vysílač zpracovává ACK odpovědi do skončení doby zpracovaní. Na základě ACK zpráv doplňuje speciální frontu, která se nazývá *retransmission queue* a obsahuje sekvenční čísla paketů pro opravu spolu s adresami přijímačů, které potřebují tyto pakety.

3. *Obnova paketů (retransmit):* Po skončení doby T_{retx} začíná vysílač krok za krokem vyzvedávat elementy z fronty a na základě počtu zájemců o paket,



Obrázek 2.3: RMTP okénko vysílaní

posílá paket buď pomocí unicastu nebo multicastu (v případě překročení hranice danou parametrem *mcast_thresh*).



Obrázek 2.4: RMTP ACK zpráva

RMTP přijímač provádí následující operace:

1. *Posílaní ACK zpráv:* Přijímač periodicky posílá ACK zprávy příslušnému DR. ACK zpráva, viz. obrázek č. 2.4, zahrnuje sekvenční číslo L a bitmapu V:

   a) *L: Lower end of receiver's window.* Číslo L ukazuje, že přijímač úspěšně dostal pakety se sekvenčním číslem nižším než L.

   b) *V: Bit-vector mirroring the status of receiver's buffer.* Bitmapa indikuje úspěšnost přijetí paketu se sekvenčním číslem následujícím za číslem L. Jednička v bitmapě na pozicí i označuje, že paket se sekvenčním číslem $L+i$ byl úspěšné přijat. V případě nuly přijat nebyl.

2. *Spočítání RTT:* Každý přijímač potřebuje spočítat obousměrné zpoždění (round-trip time) do svého DR. Přijímač využívá RTT pro výpočet periody vyslaní ACK zpráv, aby od zaslaní ACK zprávy stihla odpověď z DR přijít před začátkem zasílání následující ACK zprávy.

Designated receiver (DR) je kombinací přijímače a vysílače a také provádí stejné operace. Navíc ale periodicky posílá zprávy o svoji existenci do podskupiny níže ve stromu, podle kterých příjemci obnovují adresu nejbližšího DR.

2.1.3 Závěr

RMTP je příkladem Tree-Based protokolu s jedním vysílačem. Při použití DR snižujeme jak datový tok od klientů k serveru, tak end-to-end latency. Využívá pozitivní potvrzovaní (ACK), ale ne pro každý paket a blok dat. Zajišťuje správné pořadí paketů (in-order delivery) a spolehlivost přenosu. Je vhodný pro sítě různé velikosti, ale pro efektivitu ve velkých sítích je potřeba používat speciální přijímače — DR.

2.2 PGM — Pragmatic General Multicast

PGM je postaven nad protokoly IP a UDP. Patří k Tree-Based protokolům. Pro spolehlivý přenos dat využívá metody ARQ a FEC. V ARQ, na rozdíl od RTMP, používá negativní potvrzovaní (NACK/NAK).

2.2.1 Architektura

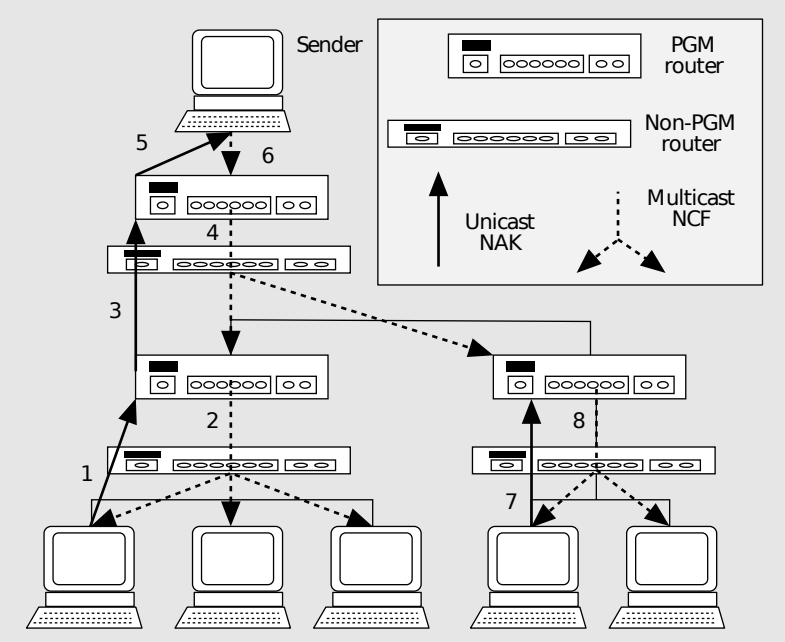
PGM strom se strojí přes routery, které podporují PGM, tzv. Network Elements (NEs). NE se chovají podobně jako DR u RMTP. Řídí provoz ve svém podstromu, ale, na rozdíl od DR, necachuje datové pakety.

Na obrázku 2.5 převzatého z článků [5] je vidět příklad zasíláni NAK/NCF, ale můžeme ho použít i pro popis fungovaní protokolu:

- Vysílač (server) sekvenčně vysílá **datové pakety (ODATA)**.
- Příjemce v případě ztráty paketu čeká náhodnou dobu. Zatím, v případě že nepřijal příslušné **potvrzovaní doručení (NCF — NAK confirmation)**, unicastem odesílá **NAK zprávu** kořenu stromu (vysílači nebo NE) (1).
- Při doručení **NAK zprávy** multicastem NE odesílá **NCF** (2,4,8) do podstromu. V případě, že nepřijal příslušnou **NCF zprávu** (7), tak přeposílá **NAK** ke kořeni podstromu (vysílači nebo NE) (3,5).

- Vysílač přijímá **NAK zprávu** a odesílá **NCF** (6). Mezitím generuje **opravný paket (RDATA)** a odesílá do multicastové skupiny.

- NE, na základě informací, odkud přišel, **NAK** přeposílá **RDATA** do správných podstromů.



Obrázek 2.5: A NAK/NCF scenario.

(1) Receiver NAKs for a lost packet. (2) Its PGM parent router multicasts an NCF (3). The router unicasts a NAK to its parent, who (4) multicasts an NCF. (5) A unicast NAK is sent to the sender, who responds with a multicast NCF. (6) At a later time, another receiver detects the loss of the same packet and unicasts a NAK (7) to its PGM parent, who multicasts an NCF (8). The parent does not send an upstream NAK, since it has already noted the reception of the corresponding NCF. [5]

NEs řeší problém NAK-impolsion, ale, v případě, že přijímač ztratil paket, vysílač pošle opravu paketu pro všechny příjemce, dokonce i pro ty, kteří tento paket nepotřebují. **Desinated Local Repairer (DLR)** řeší tento problém. DLR je nepovinný síťový uzel, který přijímá a cachuje datové pakety a v případě potřeby posílá opravy. V případě, že je připojen k podstromu, tak NE bude přeposílat **NAK** jemu. DLR, v případě, jestli má opravu pro tento paket, pošle **NCF** a **RDATA** do podstromu. V případě že opravu nemá, tak pošle NAK dalším DLRs, dokud NAK nedojde k vysílači, který má určitě opravu.

2.2.2 Implementační rysy

PGM podporuje možnost opravy pomoci obyčejných RDATA paketů nebo efektivně pomoci paritních paketů.

V PGM, dle [21] jsou dva způsoby doručení paritních paketů:

Proactive (FEC): paritní pakety se zasílají v dobu vysílaní ODATA paketů a příjemce má možnost bez NAK zpráv opravit ztracené pakety sám, ale jen v případě malé ztráty paketů.

On-demand (ARQ): paritní pakety se zasílají jen v případě doručení speciální Parity NAK zprávy, kde jsou ukázány pakety pro opravu. Tím umožňuje pomocí jednoho paritního paketu opravit pakety u několika příjemců na základě jejich ostatních dat.

Vysílač a NEs jsou kořeny podstromu a aby všechny PGM zařízení níže v podstromu věděli jejích adresu, tak vysílač periodicky posílá Source Path Messages (SPMs). SPM má v sobě adresu zdroje a každý NE ji mění na svou adresu a posílá dal do svého podstromu. To umožňuje všem dostat adresu rodiče v podstromu a, v případě odpojení NE, obnovit strom.

NE si musí pamatovat, odkud přišel NAK a příslušné číslo paketu pro opravu. Z toho důvodu může NE, při velkém počtu příjemců na jeden NE, přijít o paměť. V takovém případě se začne chovat jak obyčejný směrovač.

2.2.3 Závěr

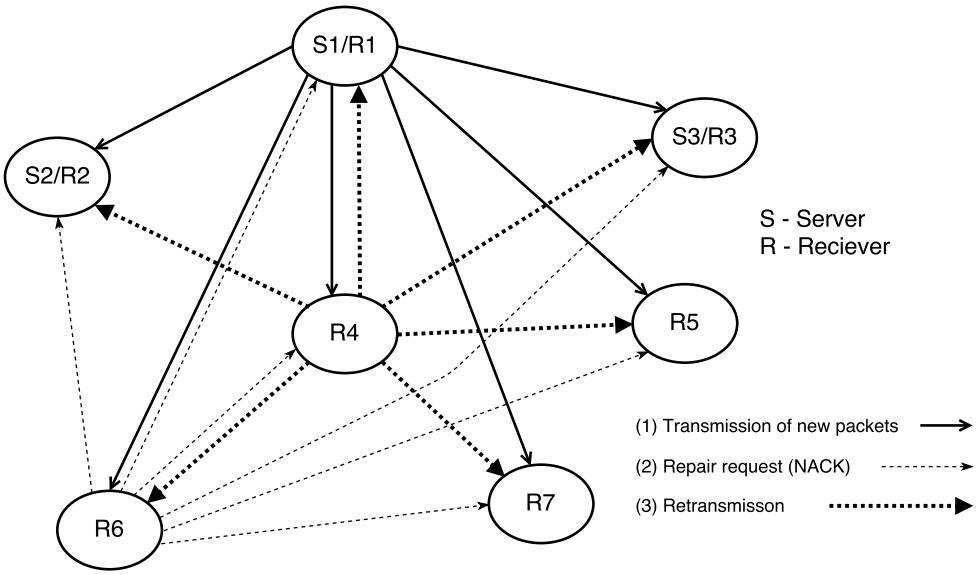
Architektura PGM je Tree-Based a se strojí na základě routeru z podporou PGM (PGM NEs). Pro opravu paketu využívá negativní potvrzovaní (NAK) a speciální zařízeni pro lokální opravu (DLRs). Navíc podporuje metodu FEC. Při využiti směrovačů s podporou PGM muže byt efektivně použit pro sítě libovolné velikosti.

2.3 SRM — Scalable Reliable Multicast

SRM — je transport-layer (L4) framework umožnující vysílaní z více zdrojů mnoha příjemcům. Využívá protokoly Application-Level Framing (ALF)[2] a Light-Weight Sessions (LWS)[7].

2.3.1 Architektura

Hlavní myšlenka SRM se spočívá v tom, že obnovu ztraceného paketu neposílá originální zdroj, ale nejbližší příjemce který tento paket má.



Obrázek 2.6: Příklad komunikaci v protokolu SRM

Na obrázku 2.6 je vidět příklad komunikaci v protokolu SRM:

- Na začátku server S1 vysila novy paket dat do multicastové skupiny (1).
- Příjemci S2/R2, S3/R3 a R4 dostávají paket, ale R5, R6 a R7 nedostávají.
- Myšlenka SRM spočívá v tom, že jenom jeden příjemce v případě ztráty paketu posila NAK, a jenom jeden člen skupiny (ideálně nejbližší) posila opravný paket dat. Proto R5, R6 a R7 čekají náhodnou dobu, a po skončení své doby čekaní, vysílají NAK do multicastové skupiny. V našem případě bude prvním R6(2). R5 a R7 přijmou tento NAK, a proto nebudou odesílat svůj.
- Poslat opravný paket může každý člen skupiny (včetně příjemců). Ale aby jen jeden z nich poslal paket, je potřeba využit metodu s náhodným čekáním, jako v minulém kroku. V našem případě R4 první dostane NAK a pošle opravný paket (3).

2.3.2 Implementační rysy

Pro efektivní fungovaní metody s čekáním je potřeba správně zvolit dobu čekaní, přičemž tato hodnota by měla být částečně náhodná. V SRM máme dva případy, kdy potřebujeme čekat:

- *Před vysíláním NAK zprávy:* doba čekaní se vybírá náhodně z intervalu $[a_1 * d_{S_x,R_x}, (a_1 + a_2) * d_{S_x,R_x}]$, kde a_1 a a_2 jsou konstanty a d_{S_x,R_x} je odhad jednostranného zpoždění mezi příjemcem R_x a vysílačem dat S_x.
- *Před vysíláním opravy:* doba čekaní se vybírá náhodně z intervalu $[b_1 * d_{R_x,R_y}, (b_1 + b_2) * d_{R_x,R_y}]$, kde b_1 a b_2 jsou konstanty a d_{R_x,R_y} je odhad jednostranného zpoždění mezi příjemcem R_x, který paket pro opravu má, a příjemcem R_y.

2.3.3 Závěr

Protokol SRM, na rozdíl od ostatních, není tree-based. Všechny pakety (včetně NAK) se zasílají na skupinovou adresu a členové skupiny využívají metodu s náhodným čekáním pro sníženi zátěží sítě. Je vhodný pro případy, kde mnoho vysílačů posílá různá data (např. editor s možností spolupráce), ale není efektivní při přenosu velkých souboru od jednoho vysílači.

2.4 FLUTE — File Delivery over Unidirectional Transport

FLUTE je jedním z protokolů nabídnutým IETF (Internet Engineering Task Force) RMT WG (Reliable Multicast Transport Working Group). V současné době má dvě verze. První verze z roku 2004 je experimentální a popsaná v RFC 3926 [17]. Druhá verze z roku 2012 už není experimentální, je popsaná v RFC 6726 [18], využívá novější specifikaci protokolů, ale základní princip zůstal stejným.

FLUTE strojí na základě Asynchronous Layered Coding (ALC) protokolu, který je sám o sobě protokolem pro spolehlivé vysílaní. Jediné, co má FLUTE navíc, je File Delivery Table (FDT).

2.4.1 ALC — Asynchronous Layered Coding

ALC je popsan v RFC 5775 [13]. Je navržen pro jednostranné vysílaní. Dle [6] se skládá z následujících stavebních kamenů (building blocks): Layered Coding

Transport (LCT) — RFC 3451 [12], Congestion Control (CC) a Forward Error Correction (FEC) — RFC 3452 [11].

LCT: protokol pro zasílaní dat nad UDP. Obsahuje informace o probíhajícím přenosu. Je kompatibilní z FEC a CC.

CC: dává možnost příjemci zvolit jiný kanál přenosu s menší (větší) přenosovou rychlostí.

FEC: využívá se pro dosažení spolehlivosti.

2.4.2 FDT — File Delivery Table

ALC jen přenáší data, ale příjemce musí vědět, jaký soubor byl poslán (jméno, id, typ, velikost). Pro tyto účely se využívá FDT.

Pro každý soubor ve vysílaní musí existovat FDT soubor. Tento soubor muže byt vygenerován při vysílaní programem nebo může byt napsán ručně ve formátu XML. V tomto souboru musí být uvedeny povinné udaje (id souboru, FEC metoda, velikost, přenosové rychlostí) a nepovinné (URI, typ souboru a další). Více informací v 3. kapitole RFC 6726 [18].

2.4.3 Závěr

Z pohledu typu dosažení spolehlivosti je FLUTE (ALC) dost jednoduchý protokol. Je protokolem pro jednosměrné přenosy — nevyžaduje zpětnou vazbu mezi vysílačem a příjemci. Je masivně škálovatelným — může být použit v sítích libovolné velikosti a typu (např. satelitní linky, radiové spoje). Pro zabezpečení doručení využívá FEC.

Nevýhodou ale je, že v případě, kdy informace z paritních paketů nestačí pro obnovu paketů, musí příjemce čekat na nové vysílání stejného souboru, aby dostat chybějící data, což může dlouho trvat.

2.5 Výsledek analýzy protokolů

Každý z probraných protokolů řeší odlišnými způsobnými základní problémy spolehlivého vysílaní v multicastu. Hlavním problémem je zátěž kanálu od příjemců k vysílači, a proto cílem všech protokolů je především sníženi této zátěže. Výsledkem této analýzy je rozdělení probraných protokolů do několika skupin na základě metod, kterými je dosazen tento cíl.

2.5.1 Tree-based protokoly

Hlavní myšlenka spočívá v rozdělení skupiny příjemců na lokální regiony (domény), které mohou se dělit na subdomény. Takovým způsobem se tvoří stromová hierarchie, kde vysílač je kořen stromu a jeho potomci jsou domény. Každá doména muže mít subdoménu, která má svoje speciální zařízeni (DR v RMTP, DLR v PGM), které je kořenem subdomény a všichni příjemci v lokálním regionu s ní komunikují. Z probraných protokolů k tree-based patří RMTP a PGM. RMTP využívá pozitivní potvrzováni a PGM — negativní.

Hlavní výhodou tree-based protokolu je efektivní řešení problémů zahlcení zpětného kanálu ACK či NACK zprávami. Na druhou stranu je potřeba využívat speciální zařízení na cestě od vysílači k příjemcům, což se stává problémem na Internetu.

2.5.2 Skupinové komunikující protokoly

K skupinově komunikujícím protokolům patří protokol SRM. Všechny pakety (včetně NAK a opravných) se zasílají celé skupině a každý příjemce muže poslat opravu dat. Toto schéma je vhodné pro účely, kde mnoho členů skupiny vysílá různá data, např. spolupráce v programu nebo video konference. Pro efektivní přenos není potřeba mít speciální zařízeni na cestách, ale nevýhodou je mála škálovatelnost.

2.5.3 Protokoly bez zpětné vazby

K protokolům bez zpětné vazby především patři ALC, na kterém už je postaven FLUTE. Hlavní výhodou takových protokolů je to, že příjemce netvoří žádný provoz zpět k vysílači, a proto je masivně škálovatelný. V případě využiti metody FEC může byt odolný proti malým ztrátám paketů.

Problém ale je v dlouhé době doručení celého souboru, protože v případě ztráty paketu, kdy ho FEC nemůže opravit, je potřeba počkat, než začne znovu vysílání souboru a s tímto vysíláním už přijde potřebný paket.

Kapitola 3

Existující implementace protokolů

Po provedeni analýzy protokolů začala fáze hledaní implementaci těchto protokolů. Překvapením ale bylo, že většina z protokolů nemá implementaci. Některé ze starších protokolů měly realizaci, ale v době psaní bakalářské práce ještě existující odkazy nebyly funkční. Jiná možnost je, že univerzity mají implementaci, ale nezveřejnili ji na Internetu.

Podle mého názoru je malý počet implementací ovlivněn tím, že současné programy pro zasílaní dat jsou navržené především pro komunikací v Internetu. Spousta protokolů pro spolehlivý multicast ale potřebují směrovače s podporou specifického protokolu nebo, pro efektivní využiti kanálu od příjemců k vysílači, potřebují speciální zařízení v síti. V Internetu nemusí takováto zařízení existovat a je je obtížné je tam přidat.

V běžné lokální sítí se využívají jednoduché zařízeni a požadavky na efektivitu využiti zpětného kanálu je menší. Z těchto důvodů jsou veškeré implementace založené na jednodušších protokolech, na protokolech bez nucené zpětné vazby k vysílači nebo na protokolech, které podporují větší množství hardwaru.

Konkrétně byly nalezeny následující programy:

- MAD-FCL — je realizací protokolu FLUTE.
- UDPcast — využívá svůj protokol, podrobná specifikace nebyla nalezena.
- UFTP — využívá svůj protokol UFTP, který je popsán na stránkách programu.
- OpenPGM — je knihovna realizující protokol PGM.

3.1 MAD-FCL

MAD-FCL (MAD FileCasting Library) je open-source (pod licenci GNU) realizací první verze protokolu FLUTE. Protokol byl již probrán, proto stručně uvedeme informaci o realizací:

- Hlavní stránka projektu je `http://mad.cs.tut.fi/`.
- Projekt je napsán v jazyce C.
- Poslední verze distribuce je 1.7 z 5.března 2007.
- Distribuce má v sobě knihovny MAD-ALCLIB, MAD-SDPLIB a MAD-FLUTELIB a aplikaci MAD-FLUTE.
- Knihovna MAD-ALCLIB je implementaci ALC/LCT protokolu dle RFC 3450 (ALC), RFC 3451 (LCT), RFC 3452 (FEC), RFC 3695 (Compact FEC Schemes).
- Podporuje Simple XOR, Reed-Solomon a Parity Check Matrix-based FEC schematy.

Nás hlavně zajímá aplikace MAD-FLUTE, která je hotovým řešením pro vysílání souboru. V následující kapitole vyzkoušíme a otestujeme tuto aplikaci.

3.2 UDPcast

UDPcast je aplikace pro přenos dat určité skupině příjemců, která se určuje před začátkem vysílaní.

- Hlavní stránka projektu je `https://www.udpcast.linux.lu/`.
- Poslední verze distribuci je 20120424 z 24.dubna 2012.
- Distribuci má v sobě 2 programy: `udp-sender` a `udp-reciever`.
- Čte data ze standardního vstupu na serveru, vypisuje na standardní výstup u klientů.
- Podporuje asynchronní mode a FEC.

Využívá svůj protokol a nemá přesnou specifikaci, a proto je potřeba uvést princip fungováni:

- Na začátku se spouštějí klienti a čekají na CMD_HELLO zprávu.

- Při spouštění serveru vysila server CMD_HELLO zprávu do síti broadcastem. V případě, že multicastová adresa je určená parametrem `--mcast-data-address address`, vysílá CMD_HELLO už multicastem na tuto adresu a tím umožňuje komunikaci mezi subnety.
- Příjemci unicastem se registruji ke skupině příjemců na serveru.
- Mezitím server nebo klienti spouštějí vysílaní.
- Server rozděluje data na části (slices). Po odeslaní každé části si server nechá u klientů potvrdit doručení paketů. Klienti unicastem posílají ACK (potvrzeni že doručil všechno) nebo NACK zprávu a server případně odesílá opravné pakety.
- V případě, že všechní přijali aktuální čast (slice), server odesílá další.

Protože ACK a NACK zprávy se posílají jenom na konci vysílací části, tak je tím vyřešen problém NACK-impolsion. Problém s neočekávaným odpojením klientů se jednoduše vyřešen timeoutem.

3.3 UFTP

UFTP je program pro přenos souboru určité skupině příjemců.

- Hlavní stránka projektu je `http://uftp-multicast.sourceforge.net/`.
- Poslední verze distribuce je 4.9.3 z 21.ledna 2017.
- Umožňuje vysílaní jednoho nebo více souborů.
- Podporuje TLS šifrování.
- Na stránkách programu má specifikaci: `http://uftp-multicast.sourceforge.net/protocol3.txt`.
- Wireshark od verze 2.2 umí sledovat provoz UFTP.
- V současné době se využívá časopisem The Wall Street Journal pro zasílaní zdrojů pro tisk pomoci satelitního vysílání.

Vysílaní souboru pomoci UFTP se provádí v 3 fázích:

1. *Registrační fáze (Announce/Register phase):* server vysílá prohlášení (announce) na multicastovou adresu (public multicast address), na které poslouchají

klienti. Všechny následující zprávy se vysílají na speciální multicastovou adresu (private multicast address), která je zaslána v prohlášení. Při doručení prohlášení, klienty unicastem posílají požadavek na registraci.

2. *Přenos souboru (File Transfer phase):* tato fáze je rozdělená na další dvě fáze:

   a) *Posílaní informaci o souboru (File Info phase):* server posílá zprávu popisující soubor. Tato zpráva obsahuje název a velikost souboru a popis, jak je soubor rozdělen na části. Soubor je rozdělen na bloky, které jsou dále sdružené do několika sekcí. Jeden blok je přenášen jedním paketem, sekce je skupina bloků. Celkový počet bloků a sekcí se přenáší v této zpravě. Klienti při doručení této zprávy posílají unicastem potvrzení doručení.

   b) *Posílaní dat (Data Transfer phase):* server vysílá všechny sekce souboru. Klienti při detekci konce každé sekce vysílají v případě ztráty bloků NAK zprávu obsahující číslo sekce a pole bitů označující ztracené bloky v sekci. Server zachovává informaci o ztracených blocích, ale v této fázi neodesílá opravné pakety. Dále, když jsou všechny pakety poslány, server posílá zprávu, která to oznamuje.

3. *Potvrzení doručení (Completion/Confirmation phase):* klienti, při přijetí *end of file* zprávy od serveru, odesílají *complete* nebo NAK zprávu. Od teto doby, server posílá opravné pakety a analyzuje další NAK zprávy. To pokračuje do té doby, než server dostane *complete* zprávy od všech klientů.

3.4 OpenPGM

OpenPGM je open-source knihovna realizující protokol PGM.

- Hlavní stránka projektu je `https://github.com/steve-o/openpgm`.
- Poslední verze distribuce je 5.2.122 z 4.prosince 2012.
- Má licenci GNU Lesser GPL.
- Podporuje FEC.

Z důvodu, že OpenPGM nemá v distribuce program pro přenos souboru, nebude využit v testováni.

Kapitola 4

Testování

Hlavním účelem testovaní je především vyzkoušet programy z minulé kapitoly v praxi. Pro tyto účely je potřeba navrhnout schéma sítě, realizovat ji v Cisco laboratoři a správně tuto síť nakonfigurovat.

Další cíl spočívá v porovnaní efektivity těchto implementaci, které mají ve své distribucí programy pro multicastové přenosy souborů v sítě (nejen v lokálním segmentu síti, ale také mezí subnety). Je potřeba navrhnout experiment, který umožní porovnat efektivitu implementací, proto musí být vymyšlené schéma sítě vhodné i pro tento cíl.

V následující sekci podrobně popisuji návrh a realizaci testovacího prostředí, průběh testovaní a porovnaní výsledků.

4.1 Testovací prostředí

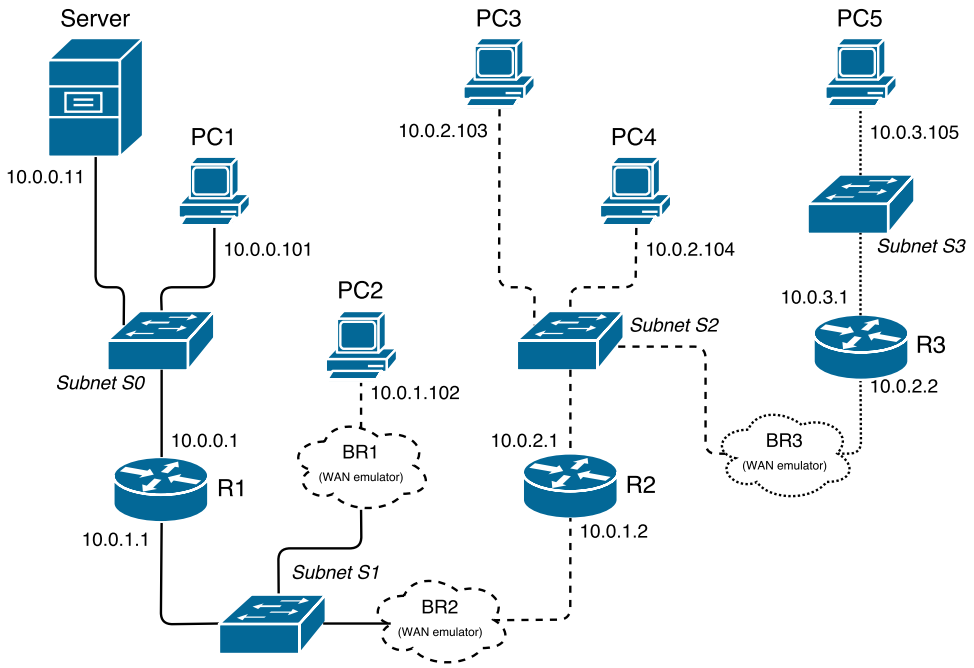
V Cisco laboratoři jsou k dispozici následující prostředky:

- 24 počítačů s třemi síťovými rozhraními (1x GigabitEthernet, 2x FastEthernet).
- 24 routerů Cisco 2901 s dvěma síťovými rozhraními (2x GigabitEthernet).
- 24 přepínačů Cisco Catalyst 2960 s 24 FastEthernet porty.

U počítačů existuje možnost vybrat a nainstalovat obrazy různých operačních systémů (Linux, Unix, Windows), které jsou již připravené k použiti. Pro účely teto bakalářské práce byl použit obraz „Netacad NG v18“ s Ubuntu 14.04. Tento obraz má nainstalovaný program Wireshark a od startu systému běžící SSH server.

4.1.1 Schéma síti

Na obrázku 4.1 je schéma konfigurace testovacího prostředí. Podrobnou verzi můžeme vidět na obrázku 4.2.



Obrázek 4.1: Schéma síti

Schéma má následující prvky:

Server: počítač, který je vysílačem souborů.

PC: počítač, který může být přijímačem.

R: router s podporou PIM a IGMP.

BR: bridge (most) je speciálně konfigurovaný počítač, který přeposílá pakety z jednoho rozhraní na druhé (bez změny MAC a IP adres). Umožňuje emulovat WAN spoje (např. zpoždění, míchaní a ztrátu paketů).

Síť je rozdělena do čtyř subnetů s maskou /24 (255.255.255.0). V nulovém subnetu máme jeden server (vysílač dat) a počítač PC1, který je přijímačem v tomto lokálním segmentu sítě. Z tohoto důvodu není závislý na konfiguraci směrovače a při testovaní dojdou pakety zaručené k němu.

V subnetu S1 slouží počítač PC2 k testování správnosti konfigurace routeru R1. V případě, že je konfigurace správná a BR1 nefiltruje pakety, bude PC2 dostávat všechny pakety od serveru. Bridge (most) při testování slouží k emulovaní ztráty paketů. Po nastavení BR2 nebude příjemce vpravo od BR2, na rozdíl od PC1, dostávat některé pakety. BR1 se zapnutým filtrováním bude ztrácet jiné pakety než BR2, ale některé z nich mohou byt stejné. Tím emulujeme dvě odlišné WAN linky.

V subnetu S2 jsou dva počítače (PC3, PC4), které mohou sloužit k testovaní IGMP Snoopingu. V případě, že jen jeden počítač patří k multicastové skupině, by druhý počítač neměl dostávat multicastové pakety na svoje rozhraní. Bridge BR3 slouží k emulovaní ztráty paketů mezí subnety S2 a S3.

4.1.2 Konfigurace síti

Počítače mají tři rozhraní: eth0, eth1 a eth2. Všechny počítače, včetně serveru a mostů, jsou připojeny na rozhraní eth0 k síti 192.168.0.0/24 a navíc je připojen počítač (notebook ve schématu), který řídí ostatní počítače pomocí SSH. V této síti je potřeba ručně nastavit IP adresy dle schématu na obrázku 4.2. Další nastavení počítačů muže byt provedeno pomoci skriptů.

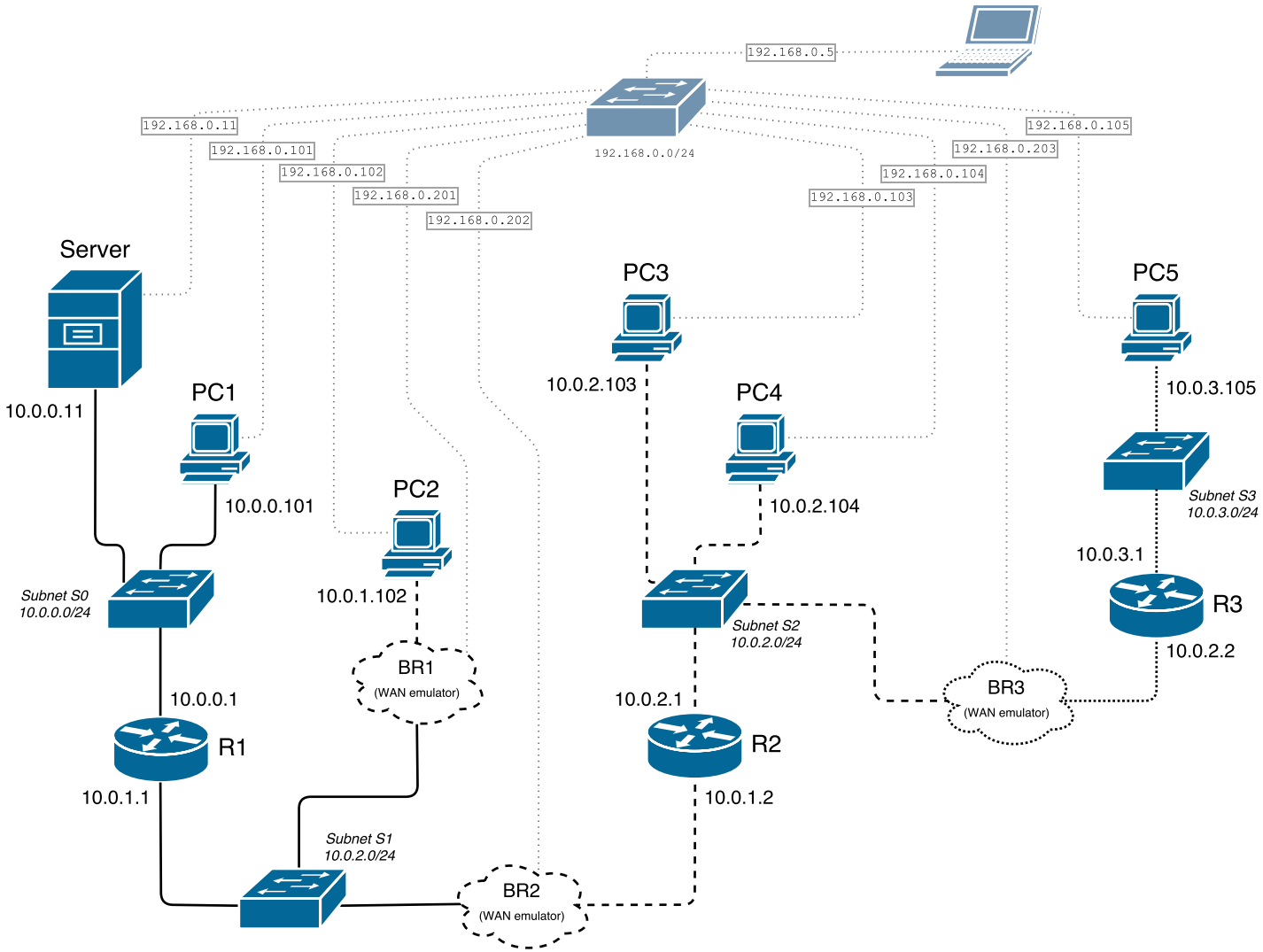
Server a počítače PCx jsou připojeny k různým podsítím s maskou /24 z rozsahu 10.0.0.0/8 pomocí rozhraní eth1 a konfigurují se pomocí skriptů. U každého počítače je nastavena IP adresa a implicitní brána (default gateway) dle schématu 4.2.

Routery mají rozhraní Gi0/0 a Gi0/1. Gi0/0 je připojen k subnetu s menším číslem a Gi0/1 s větším (např. S0 k Gi0/0 a S1 k Gi0/1). Na každém routeru je zapnuté směrovaní pro multicast a aktivován protokol PIM v Sparse-Dense modu. Příkazy pro konfiguraci routeru naleznete v dodatku B.1.

Program pro emulaci WAN umožňuje filtrovat jen výstupní pakety. Pokud je nastavena ztrátovost paketů na rozhraní eth2, pakety přicházející z rozhraní eth1 budou filtrované. Proto vstupním rozhraním mostů BRx je eth1 a výstupním je eth2.

4.1.3 Příprava počítačů

Na začátku je na každý počítač (Server, PC, BR) nahrán obraz „Netacad NG v18“ s Ubuntu 14.04. Po spuštění Ubuntu je potřeba ručně na všech počítačích nastavit adresu ze sítě 192.168.0.0/24 na rozhraní eth0 dle horní častí obrázku 4.2. To se provádí následujícím příkazem, kde X je poslední oktet adresy dle schématu 4.2 a root heslem pro obraz „Netacad NG v18“ je „`net123`“:



Obrázek 4.2: Podrobná schéma síti

```
sudo ifconfig eth0 192.168.0.X 255.255.255.0
```

Po konfiguraci těchto IP adres můžeme pomoci `ssh` vzdáleně ovládat počítače, a proto můžeme adresy dále konfigurovat pomoci skriptů na jiném počítači (notebook ve schématu).

Pro konfiguraci počítačů PC a Serveru spouštíme skript `serv_pcs_preparation.sh` (B.3.2), který pro každý počítač spouští skript `init_pc.sh` (B.3.1) a provádí následující operace:

- Konfiguruje rozhraní eth1 dle schématu 4.2.
- Vytváří nový oddíl na disku o velikosti 20 GB a připojuje v systému do adresáře `/mnt`.
- Vytváří další oddíl na disku o velikosti 20 GB a připojuje do adresáře `/tmp`.

Dále potřebujeme zkopírovat testované programy na všechny počítače. To provádíme skriptem `copy_programs.sh`, který, s využitím skriptu `copy_files.sh` (B.2.2), pošle soubory z adresářů `./Programs` na počítače. Program MAD-FLUTE potřebuje některé knihovny, které nejsou nainstalované v použitém operačním systému, a proto je potřeba skriptem `install_deps.sh` nakopírovat a nainstalovat balíky obsahující tyto knihovny. V posledním kroku potřebujeme skriptem `copy_files.sh` (B.2.2) zkopírovat na server soubory pro provedení následujícího testování. Programy `copy_programs.sh` a `install_deps.sh` jsou na příslušném disku u bakalářské práce v adresáři `Scripts`.

4.1.4 Příprava mostů

Příprava mostů se provádí pomocí skriptu `init_br.sh`. Kód skriptu naleznete v dodatku B.4.1. Skript vytvoří nové virtuální rozhraní br0, které je L2 mostem mezi eth1 a eth2, a vypne STP (Spanning Tree Protocol).

4.1.5 Emulace WAN

Pro emulaci WANu je použit program *NetEm*, který je součástí jádra Linuxu od verze 2.6. Příkazem pro ovládaní je `tc`. Příkaz `tc` umožňuje nastavit zpožděni, míchání a ztrátu paketů. Pro účely vyzkoušení programů stačí nastavit jen pravděpodobnost ztráty paketů, ale můžeme nastavit zpožděni pro emulaci internetového spoje.

Pro nastavení na rozhraní eth2 pravděpodobnosti ztráty paketů na 10% a zpoždění na 15ms používáme následující příkaz:

```
tc qdisc add dev eth2 root netem loss 10% delay 15ms
```

Pro změnu pravděpodobnosti ztráty například na hodnotu 20% a zpoždění na 10ms měním už existující pravidlo:

```
tc qdisc change dev eth2 root netem loss 20% delay 15ms
```

4.2 Hledání vhodné konfigurace programů

Cílem této sekce je prozkoumat princip fungování programů, seznámit se s jejich parametry a napsat skripty, které umožní vzdáleně spouštět servery a klientské programy a sbírat dump přenosu paketů na rozhraní v době vysílání dat serverem.

4.2.1 UFTP

UFTP má dva programy: `uftpd` a `uftp`. Na začátku spouštíme klienty. Pak spouštíme server. Vysílání souboru probíhá podle kroků popsaných v minulé kapitole. Klienti při ztrátě paketu odesílají NAK zprávu a v logu serveru (implicitně stderr) se objevuje počet ztracených paketů. Klient (`uftpd`) se při přijetí všech souborů odpojuje (IGMP Leave) od „privátní“ multicast skupiny, ale pokračuje v poslouchání v public multicast skupině na nové prohlášení od serveru. Server (`uftp`) při ukončení vysílání všech souborů a oprav vypíše v logu čas doručení souboru pro každého příjemce a ukončí se.

4.2.1.1 Server, program `uftp`

Program `uftp` funguje v popředí. Implicitně má public adresu nastavenou na 230.4.4.1 a vysílá s rychlostí 1000 Kbps. Parametry konfigurujeme podle našich potřeb:

- Při spouštění s implicitním nastavením nejdou pakety mezi segmenty, protože *ttl* je nastaven na 1. Z tohoto důvodu nastavíme *ttl* přepínačem `"-t"` na hodnotu `"100"`.
- Vysíláme na rozhraní eth1, a proto použijeme přepínač `"-I"` s hodnotou `"eth1"`.
- Dále nastavíme rychlost na hodnotu 10 Mbps přepínačem `"-R"` s hodnotou `"10000"`.

Pro program `uftpd` je výsledný příkaz:

```
./uftp -I eth1 -t 100 -R 10000 /mnt/TestFiles/bash.txt
```

4.2.1.2 Klient, program `uftpd`

Funguje implicitně na pozadí, ale dá se spustit na popředí pomoci přepínače `-d`. Pro klienta potřebujeme jen nastavit rozhraní a adresář pro uloženi souboru. Výsledný příkaz je:

```
./uftpd -I eth1 -D /tmp
```

4.2.2 UDPcast

UDPcast má dva programy: `udp-sender` a `udp-receiver`. Program `uftpd` je klient. Implicitně se vypisuje na standardní výstup a běží v popředí. Program `uftp` je server, který funguje v popředí.

4.2.2.1 Parametrizace programu `udp-sender`

Implicitní chování programu neumožňuje ani vysílání mezi segmenty, a proto potřebujeme nastavit několik parametrů.

- Na začátku nastavíme rozhraní na eth1: přepínač `"--interface"` s hodnotou `"eth1"`.
- Implicitně `udp-sender` vysílá prohlášení na broadcastovu adresu, a proto nastavíme multicast adresu pro vysílání prohlášení a jiných, než datových paketů: přepínač `"--mcast-rdv-address"` s hodnotou `"224.2.2.4"`.
- Pakety mají implicitně $ttl = 1$, a proto nastavíme na jinou hodnotu: přepínač `"--ttl"` s hodnotou `"100"`.
- Po doručení prohlášení čekají klienti a server, než někdo z nich zmáčkne Enter. Protože takové chování ve skriptu není užitečné, tak to přepínačem `"--nokbd"` vypneme.
- Dále nastavíme autostart vysílání: přepínač `"--autostart"` s hodnotou `"20"`.
- Nastavíme navíc rychlost přenosu, aby se mohly srovnávat jiné programy s UDPcastem: přepínač `"--max-bitrate"` s hodnotou `"30m"` (30 Mbps).
- Vysíláme soubory, a proto použijeme přepínač `"--file"`, kde hodnota je cesta k souboru pro přenos.

Výsledkem je následující příkaz, kde FILE je cesta k souboru:

```
./udp_send --interface eth1 --mcast-rdv-address 224.2.2.4  \
--ttl 100 --nokbd --autostart 20 --max-bitrate 30m --file FILE
```

4.2.2.2 Parametrizace programu `udp-receiver`

Program `udp-receiver` má podobné parametry. Rozdíl spočívá v tom, že přepínač `"--file"` má za hodnotu cestu k souboru pro zápis. Výsledkem je následující příkaz:

```
./udp_receiver --interface eth1 --mcast-rdv-address 224.2.2.4 \
--ttl 100 --nokbd --file SAVE_FILE
```

4.2.3 MAD-FLUTE

U MAD-FLUTE (FLUTE) je pro klienta a server stejný program `flute`. Implicitně se chová jako klient, pro spouštění v režimu serveru se využívá přepínač `"-S"`.

4.2.3.1 Parametrizace serveru

- Pustíme program `flute` s přepínačem `"-S"`.
- Nastavíme adresu pro vysílání přepínačem `"-m:239.1.1.1"`.
- Port nastavíme přepínačem `"-p:4000"`.
- Implicitně se ttl rovná 1. Proto nastavíme přepínačem `"-T:100"` jeho hodnotu na 100.
- Přenosovou rychlost nastavíme na 30 Mbps přepínačem `"-r:30000"`. Tuto hodnotu je nutné zapsat v Kbps.
- Nastavíme Reed-Solomon FEC kódování přepínačem `"-x:2"`.
- Poměr redundantních informací nastavíme na hodnotu 50% a to přepínačem `"-X:50"`.
- Přepínač `"-B:DIR"` nastavuje kořenový adresář pro vysílání souboru, kde DIR je cesta k adresáři.
- Soubor pro vysílání zvolíme přepínačem `"-F:file///FILE"`, kde FILE je název souboru.
- Implicitně server vysílá soubor jen jednou a protože může nastat ztráta, kde redundantní data nepomohou, musíme soubor vysílat cyklicky. Takové chování se dělá přepínačem `"-C"`.

4.2.3.2 Parametrizace klienta

- Nastavíme přijímání rozhraní na eth1 pomocí přepínače `"-I:eth1"`.
- Adresa a port je podle nastavení serveru, takže přepínači `"-m:239.1.1.1"` a `"-p:4000"`.
- Přepínačem `"-B:/tmp"` nastavíme adresář pro uložení souboru, v našem případě je to /tmp.
- Musíme upřesnit, jaký soubor budeme doručovat, protože v jiném případě se program nepřeruší po doručení souboru, ale bude čekat na další. To se dělá přepínačem `"-F:file///FILE"`, kde FILE je název zasílaného souboru.

4.2.4 Výsledné skripty

Skripty `test_uftp.sh` (B.5.1), `test_udpcast.sh` (B.5.2) a `test_flute.sh` (B.5.3) mají následující chování:

- Na začátku spouštění skriptu se na všech testovacích počítačích spustí program `dumpcap`, který zapisuje provoz z rozhraní eth1 do souboru v adresáři `/tmp`.
- Dále se na všech počítačích sloužících k přijímání souboru od serveru spustí klientský program příslušné aplikace.
- Po spuštění klientů se spouští server.
- Na konci, kdy je soubor odeslán všem klientům, se server vypíná. Ale, v případě programu `flute`, se vypínají klienti a po jejich skončení skript vypíná server.
- Dále se na všech počítačích vypíná dumpcap.
- Posledním krokem je odesílání dumpu provozu na řídící počítač (notebook na schématu 4.2)

Vzhledem k tomu, že se při testování mohou měnit různé parametry v příkazu, které spouští server, nemá skript vstupní parametry z příkazové řádky od začátku, a je možnost tyto parametry buď ručně měnit ve skriptu nebo je nahradit proměnnými s argumenty. Proto se skript parametrizuje dle potřeb před začátkem testování (spouštění skriptu).

4.3 Návrh a provedení experimentu

Jedna z možností srovnáni efektivity protokolu je srovnání počtu odeslaných dat od serveru a příjemců. Další možnost je srovnání protokolů na základě počtu odeslaných dat a času potřebného pro tento přenos. V tomto testování nás hlavně zajímá druhá možnost, protože programy, které vyhovují pro testovaní (FLUTE, UFTP, UDPcast), buď nemají zpětnou vazbu (FLUTE) nebo, při zvolení pro testováni i 20 počítačů, bude počet zpětných paketů menší, než jedno procento.

4.3.1 Návrh

Pro testováni protokolů použijeme skripty z minulé sekce. Jeden z parametrů, který budeme měnit, je různá přenosová rychlost. Hodnoty pro testování jsou: 30, 20, 10 a 5 Mbps. Přenosová rychlost by neměla mít vliv na čas doručení, ale při prvním vyzkoušení programu vliv byl. Proto zkontrolujeme podrobně, jestli má přenosová rychlost vliv v prázdném 100 Mbps kanálu.

Další parametr je přenos souborů různé velikosti. Pro přenos máme 4 různé soubory:

`bash.txt`: je desetkrát za sebou zapakovaný manuál interpretu `bash`. Jeho velikost je 3405 KB.

`bash4x.txt`: je čtyrykrát zapakovaný soubor `bash.txt`. Jeho velikost je 13621 KB.

`bbb180p.mp4`: je video „Big Buck Bunny“ o velikosti 64657 KB. Má licenci Creative Commons a často se používá v různých pracích. Stránka projektu je `https://peach.blender.org/`.

`bbb1080p.mp4`: je full-hd verze videa „Big Buck Bunny“ o velikosti 355857 KB.

Nastavíme filtrování na mostech BRx s následujícími parametry:

- BR1 má ztrátovost (loss) 15% a zpoždění (delay) 30ms.
- BR2 má ztrátovost (loss) 10% a zpoždění (delay) 65ms.
- BR3 má ztrátovost (loss) 10% a zpoždění (delay) 5ms.

Celková ztrátovost v segmentu S3 se bude rovnat $1 - (1 - 0.1) * (1 - 0.1) = 1 - 0.81 = 0.19 = 19\%$ a celkové zpožděni bude 70ms. Počítač PC2 nebude dostávat 15% provozu. Protože ztrátovost v segmentu S3 je 19% a u počítače PC2 je 15% a oni

jsou nezávislé, tak výsledná hodnota ztrátovosti paketů v celé síti bude:

$$0.15 + 0.19 - 0.15 * 0.19 = 0.3115$$

Dále spočítáme teoretickou hodnotu, která je odhadem počtu přenášených dat při zadané ztrátovosti u NAK-oriented protokolů, např. u UFTP. Při přenosu 1000 paketů ztratíme v první iteraci UFTP 190 z nich. Dále ze 190 ztratíme 36.1 paketů, atd. Je to klesající geometrická posloupnost a potřebujeme najít sumu teto posloupnosti, na to použijeme vzorec $s = a_1/(1 - q)$, kde $a_1 = 1000$ a $q = 0.3115$. Výsledkem je

$$1000/(1 - 0.3115) = 1000/0.6885 \approx 1452$$

paketů. Z toho plyne, že počet přenášených dat bude přibližně 1.45krát vetší (145%), než velikost souboru. Při analýze výsledku testováni otestujeme tuto hypotézu.

4.3.2 Provedení

Při provádění experimentu vysílá server data počítačům PC101, PC102, PC103 a PC105. Počítač PC104 slouží ke kontrole fungování metody IGMP Snooping a neměl by dostávat pakety z multicastové skupiny ostatních počítačů. Testování pro soubory `bash.txt`, `bash4x.txt` a `bbb180p.mp4` bylo provedeno pro rychlost 30, 20, 10 a 5 Mbps. Soubor `bbb1080p.mp4` byl testován jen pro rychlosti 30, 20 a 10 Mbps.

4.3.3 Výsledky

Výsledkem provedení experimentu a analýzy dump souborů jsou následující tabulky:

- Tabulky s časem ztraceným na přenos souboru všem příjemcům. Hodnoty jsou pro soubory různé velikosti a s různou přenosovou rychlostí. Jsou to tabulky 4.1, 4.4 a 4.7. Tabulky jsou vytvořeny pro každý ze třech programů.
- Tabulky s celkovým počtem kilobajtů odeslaných do multicastové skupiny pro různé soubory a přenosovou rychlost. Jsou to tabulky 4.2, 4.5 a 4.8.
- Tabulky s celkovým počtem kilobajtů přenášených a přijatých na rozhraní eth1 každého počítače při komunikaci v multicastové skupině v době vysíláni souboru `bash4x.txt` pro různé hodnoty přenosové rychlosti. Jsou to tabulky 4.3, 4.6 a 4.9.

4.4 Analýza výsledků

Vzhledem k malé velikosti souboru `bash.txt` je doba přenášeni několik vteřin, a proto je chybovost měření poměrně velká, protože odeslání odhlášení trvá nějaký čas. Pro ostatní soubory chybovost měření už tak kritická není. Čím větší soubor a menší přenosová rychlost, tím je chyba v porovnání programů menší. Výsledky se mohou neshodovat kvůli chybě při měřeni, ale jsou blízko k očekávaným. Například pro soubor `bbb180p.mp4` při přenosu protokolem UFTP se rozdíl mezi dobami vysílání při rychlostech 30 Mbps a 5 Mbps rovná $146.917/27.624 = 5.319$, což je blízko k hodnotě $30/5 = 6$.

4.4.1 UFTP

V tabulce 4.1 je vidět přímá závislost přenosové rychlosti a času.

V tabulce 4.2 je počet přenášených dat téměř stejný pro všechny přenosové rychlosti. Takové chovaní je očekávané ode všech programů, ale jenom UFTP prokazuje tuto vlastnost.

V tabulce 4.3 vidíme následující vlastnosti:

- Počítač PC104 nedostává žádné pakety. To dokazuje, že je IGMP Snooping zapnut a funguje správně.
- Velikost dat přijatých počítačem PC105 je nejmenší. Je to způsobeno tím, že linka mezi serverem a počítačem PC105 má největší ztrátovost.
- Protože se až do konce vysílání počítač PC101 neodpojuje od multicast skupiny, tak přijímá všechny pakety, i když vždy přijme celý soubor jako první.

Na příkladu UFTP potvrdíme hypotézu pro předpokládanou velikost přenosu. Vezmeme hodnotu 19 715 z tabulky 4.3 a vydělíme velikostí souboru `bash4x.txt`:

$$19715/13621 \approx 1.447 \approx 1.45$$

Podle předpokladu by hodnota měla být 1.45, proto UFTP funguje dle předpokladu.

Speed	bash.txt	bash4x.txt	bbb180p.mp4	bbb1080p.mp4
30 Mbps	3.794	6.932	27.624	138.899
20 Mbps	4.654	11.435	38.793	205.522
10 Mbps	5.561	17.919	74.988	407.106
5 Mbps	9.002	32.124	146.917	-

Tabulka 4.1: Čas ve vteřinách potřebný pro přenos souborů programem UFTP.

Speed	bash.txt	bash4x.txt	bbb180p.mp4	bbb1080p.mp4
File size	3405	13 621	64 657	355 857
30 Mbps	4950	19 715	93 681	521 320
20 Mbps	5025	19 994	93 791	520 182
10 Mbps	4908	19 725	93 799	520 182
5 Mbps	4968	19 723	93 640	-

Tabulka 4.2: Velikost odeslaných dat programem UFTP v kilobajtech.

Speed	Server	PC 101	PC 102	PC 103	PC 104	PC 105
30 Mbps	19 715	19 715	16 836	17 761	0	15 943
20 Mbps	19 994	19 994	16 902	17 939	0	16 081
10 Mbps	19 725	19 725	16 759	17 856	0	16 018
5 Mbps	19 723	19 723	16 757	17 816	0	15 983

Tabulka 4.3: Velikost odeslaných či přijatých kilobajtů při testováni programu UFTP pro soubor `bash4x.txt` (13 621 KB).

4.4.2 UDPcast

UDPcast funguje podobným způsobem, jako UFTP. Hlavní rozdíl je v tom, že oprava sekcí souboru se provádí po přenášení této sekce. Z tohoto důvodu očekáváme podobné UFTP výsledky měření.

V tabulce 4.4 je vidět, že se čas pro přenos s menší rychlostí přenosu vzrůstá, ale, na rozdíl od UFTP, není závislost natolik přímá. Pro přenášení všech souborů mimo `bash.txt` máme, na rozdíl od UFTP, čas mnohem větší, ale, pro přenosovou rychlost rovnající se 5 Mbps, máme shodné výsledky. Pro přenášení malých souborů to může být chybou měření, ale pro soubory `bbb180p.mp4` a `bbb1080p.mp4` není tato možná chyba výrazná, přitom máme výsledky 3-4x větší, než u UFTP. Tato vlastnost může být způsobena zapnutou kontrolou rychlosti (Congestion Control), která se nedá vypnout, nebo zpožděním v síti, které může způsobit, že různé NAK zprávy se stejnou opravou přijdou v čase, kdy část opravných paketů je zaslána a server si myslí, že znova došlo ke ztrátě těchto paketů.

Speed	bash.txt	bash4x.txt	bbb180p.mp4	bbb1080p.mp4
30 Mbps	4.5152	21.6951	81.9354	458.7367
20 Mbps	6.4324	20.6665	97.3336	539.7606
10 Mbps	6.8103	27.2665	123.9786	688.6954
5 Mbps	9.9335	39.7648	185.2781	-

Tabulka 4.4: Čas ve vteřinách potřebný pro přenos souborů programem UDPcast.

Speed	bash.txt	bash4x.txt	bbb180p.mp4	bbb1080p.mp4
File size	3405	13 621	64 657	355 857
30 Mbps	8150	36 325	171 201	942 060
20 Mbps	7008	25 560	120 025	656 622
10 Mbps	5320	23 291	100 881	556 508
5 Mbps	5196	21 004	97 692	-

Tabulka 4.5: Velikost odeslaných dat programem UDPcast v kilobajtech.

V tabulce 4.5 při vysílání souboru `bbb180p.mp4` s rychlostí 5 Mbps je počet bajtů $98MB/65MB \approx 1.5$krát větší, než velikost původního souboru, což se shoduje s předpokladem ze sekce 4.3.1. Pro 30 Mbps vysílá server téměř 3x větší počet bajtů, což není podle očekáváni, a je znova vidět problém s velkými přenosovými rychlostmi.

V tabulce 4.6 vidíme následující vlastnosti:

- Počítač PC104 nedostává žádné pakety. Proto je IGMP Snooping zapnut a funguje správně.
- Počet přijatých bajtů počítačem PC101 se rovná poctu odeslaných serverem. To se shoduje s principem fungováni programu.

Závěrem bych chtěl napsat, že chování programu na větších rychlostech je v rozporu s očekáváním, které jsem měl po analýze stránek programu. Z důvodu, že oficiální specifikace protokolu používaného programem UDPcast neexistuje, není možné s jistotou odpovědět na otázku, čím je ovlivněno toto chování. V této práci ale budeme vycházet z existujících výsledků testováni a na základě nich dělat závěr.

4.4.3 FLUTE

FLUTE má, na rozdíl od ostatních programů, odlišné chováni a výsledky testování by nám měli ukazovat vlastnosti chováni protokolu FLUTE.

Speed	Server	PC 101	PC 102	PC 103	PC 104	PC 105
30 Mbps	36 325	36 325	30 801	32 741	0	29 498
20 Mbps	25 560	25 560	21 537	22 972	0	20 625
10 Mbps	21 004	21 004	17 770	18 895	0	17 044
5 Mbps	20 474	20 474	17 415	18 410	0	16 487

Tabulka 4.6: Velikost odeslaných či přijatých kilobajtů při testováni programu UDPcast pro soubor `bash4x.txt` (13 621 KB).

Při analýze tabulky 4.7 se ukázalo zvláštní chování programu pro přenosové rychlosti 30 a 20 Mbps, kde doba přenosu mezi nimi je skoro stejná pro soubor `bbb180p.mp4` a nebo, v případě souboru `bash.txt`, je pro 20 Mbps menší, než pro 30 Mbps. Při srovnání velikosti přenesených dat z tabulky 4.8 pro tyto rychlosti pro soubor `bbb180p.mp4` se ukázalo, že i velikosti přenášených dat jsou podobné, což pro stejný čas při různé rychlosti přenosu není možné.

Tím, že se vydělí počet přenesených dat časem trvání přenosu, dostaneme hodnotu přenosové rychlosti. Spočítáním pro soubor `bbb180p.mp4` a očekávanou rychlostí 20 Mbps z tabulek 4.7 a 4.8, dostáváme následující výsledky:

$$(116\,919 * 8)/9.7782 = 95656.87Kbps = 95.7Mbps$$

Je vidět, že reálná přenosová rychlost je blízká k 100 Mbps. Po dosažení tohoto výsledku, byly zkontrolovány parametry, se kterými se provádělo testování, a žádná chyba v nich nebyla nalezena. Pro stejné výpočty dle předpokládané přenosové rychlosti 10 a 5 Mbps máme následující výsledky:

$$(103\,282 * 8)/78.2030 = 10565.53Kbps = 10.6Mbps$$

$$(104\,801 * 8)/160.0819 = 5237.36Kbps = 5.2Mbps$$

Tyto výsledky ale mají hodnoty podle předpokladu. Z toho plyne, že je problém v programu. Experimentálně byla nalezena mez, od které tento parametr nefunguje, je to hodnota mezi $11.75Mbps$ a $12Mbps$.

Z výše uvedených důvodů se v tabulkách opravil sloupec s přenosovou rychlostí dle reálných hodnot. Další analýza je prováděna na základě tohoto faktu.

V tabulce 4.7 je přímá závislost času a přenosové rychlosti. Výsledky pro `bash.txt` jsou ovlivněny chybou měření a faktem, že více ztracených paketů se opravilo pomocí redundantních dat.

Speed	bash.txt	bash4x.txt	bbb180p.mp4	bbb1080p.mp4
100 ~~30~~ Mbps	3.5327	3.2015	9.7782	47.1855
100 ~~20~~ Mbps	1.7552	2.8788	9.5334	76.5530
10 Mbps	5.2441	17.4173	78.2030	728.7907
5 Mbps	9.2470	35.5865	160.0819	-

Tabulka 4.7: Čas ve vteřinách potřebný pro přenos souborů programem FLUTE s redundancí 50%.

Speed	bash.txt	bash4x.txt	bbb180p.mp4	bbb1080p.mp4
File size	3405	13 621	64 657	355 857
100 ~~30~~ Mbps	43 496	39 331	120 225	580 498
100 ~~20~~ Mbps	21 630	34 891	116 919	941 765
10 Mbps	6862	22 940	103 282	961 479
5 Mbps	6072	22 160	104 801	-

Tabulka 4.8: Velikost odeslaných dat v kilobajtech programem FLUTE s redundancí 50%.

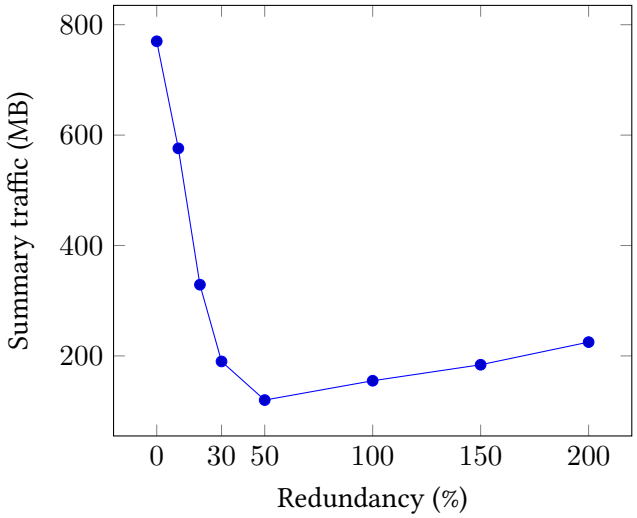
Speed	Server	PC 101	PC 102	PC 103	PC 104	PC 105
100 ~~30~~ Mbps	39 332	39 332	33 384	35 212	0	31 665
100 ~~20~~ Mbps	34 891	34 891	29 582	31 363	0	28 227
10 Mbps	22 940	22 940	19 538	20 693	0	18 702
5 Mbps	22 160	22 160	18 731	20 024	0	17 966

Tabulka 4.9: Velikost odeslaných či přijatých kilobajtů při testováni programu FLUTE s redundancí 50% pro soubor `bash4x.txt` (13 621 KB).

V tabulce 4.8 je vidět, že pro vysílání souboru se střední hodnota podílu přenesených dat a velikosti souboru přibližně rovná 170%. Podobná hodnota by mohla by mohla být velmi užitečná pro ostatní protokoly, ale u FLUTE se může stát situace, že pro opravu je potřeba se dočkat dalšího vysílání souboru. Toto můžeme vidět například ve sloupci `bbb1080p.mp4`, kde rozdíl mezi první a druhou hodnotou je hodnota blízká velikosti souboru:

$$961\,479 - 580\,498 = 380\,981 \approx 355\,857$$

V poslední tabulce 4.9 znova vidíme, že IGMP funguje, větší ztrátovost je na PC 105. Vidíme, že pro větší přenosovou rychlost máme větší přenos, ale to je ovlivněno tím, že se pro 100 Mbps může zvětšit ztrátovost z důvodu úplného využití kanálu, a protože přenos je rychlý, tak máme poměrně velký vliv kvůli zpoždění vypnutí serveru.



Obrázek 4.3: Graf počtu přenášených dat k hodnotě redundance

Pro efektivní využití programu FLUTE z pohledu velikosti přenosu je důležité zvolit správnou hodnotu redundance. Pro zjištění, jestli implicitně nastavená hodnota 50% vyhovuje našemu prostředí, byly vyzkoušeny různé hodnoty. Výsledkem tohoto zkoušení je graf 4.3, na kterém je vidět, že zvolená hodnota je nejefektivnější pro tuto síť.

4.5 Srovnání programů dle výsledků

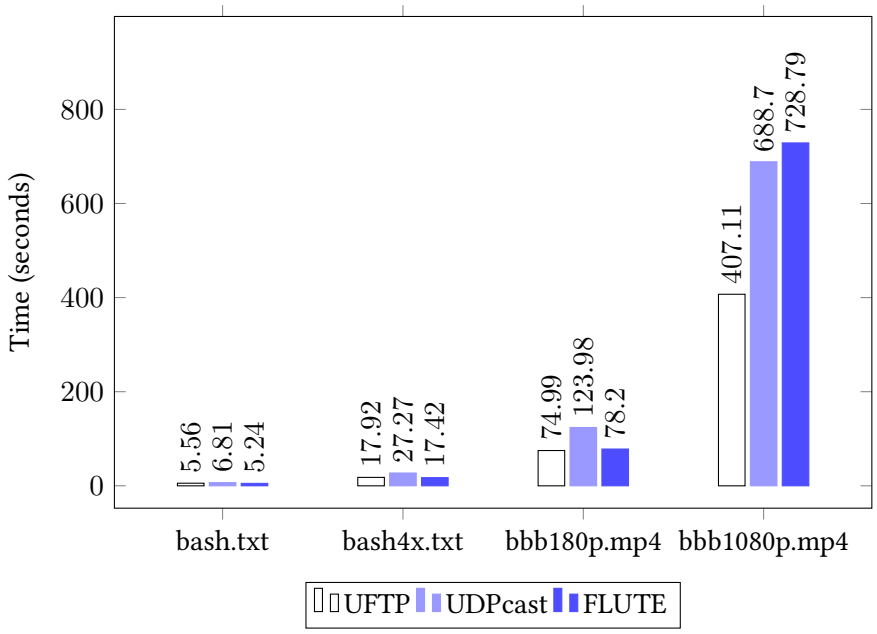
Vzhledem k tomu, že se u FLUTE nedal vyzkoušet přenos při rychlosti 30 a 20 Mbps a protokol UDPcast na stejných rychlostech má své problémy, bylo řešeno zvolit k srovnání hodnoty pro rychlost 10 Mbps. Pro toto srovnání byly vytvořeny diagramy 4.4 a 4.5, na základě kterých porovnáme testované programy.

4.5.1 UFTP

UFTP má nejmenší čas přenosu a počet přenášených dat. Funguje úplně dle předpokladů a je, pro naše testovací prostředí a účely, nejefektivnější protokol.

4.5.2 UDPcast

UDPcast má přibližně o 10% větší provoz za stejných podmínek. Jeho čas je ale o 50% větší. Je to ovlivněno zpožděním linky, protože než program začne vysílat nový segment dat, tak posílá dotazy na NAK nebo potvrzovací zprávy a klienti mu odpovídají. Tato komunikace ale pro linky s velkým zpožděním trvá

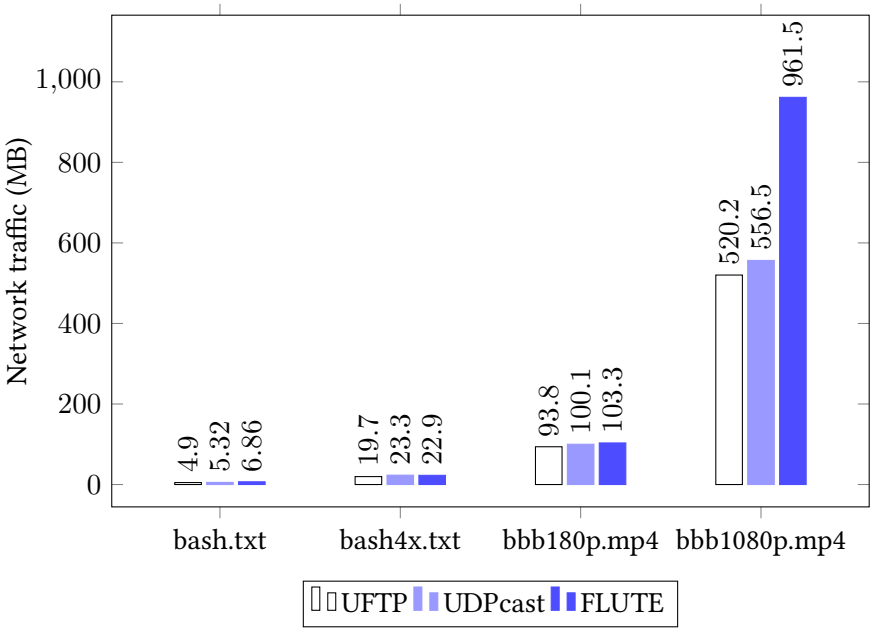


Obrázek 4.4: Diagram srovnání programů na základě času přenosu dat pro různé soubory

dlouho a zdržuje vysílání dalších segmentů. Na druhou stranu nám to zajišťuje in-order doručení segmentu, což může být v některých případech vhodné. Například v multimédiích, kde je možnost využít část souboru bez celého obsahu.

4.5.3 FLUTE

U FLUTE je vidět, že čas a počet přenášených dat pro všechny soubory `bash.txt`, `bash4x.txt` a `bbb180p.mp4` přibližně stejný. Je to ovlivněno tím, že pro doručení stačila jedna iterace vysílání souboru, a protože hodnota redundance je 50%, takže přenášená data by měla být 1.5krát větší, než původní soubor. Pro tyto soubory to tak je, ale pro soubor `bbb1080p.mp4` byla potřeba ještě jedna skoro celá iterace, a proto je velikost přenosu skoro dvakrát větší. Samozřejmě z pohledu naší sítě s velmi malým počtem příjemců může být takové chování neefektivní z toho důvodu, že i malý počet ztracených dat klientem může způsobit výrazně větší počet dat odeslaných serverem. FLUTE kvůli absenci zpětné vazby je velmi škálovatelný a může být použitý úplně libovolně na velký počet příjemců (stovky až miliony).



Obrázek 4.5: Diagram srovnání programů na základě počtu přenášených dat pro různé soubory

Závěr

V práci byl nejdříve prozkoumán princip vysílání pomocí multicastu a to jak pro běžné sítě, tak i pro sítě postavené na Cisco technologiích, to jest s podporou protokolu IGMP a PIM. Dále byly nalezené a prozkoumané protokoly pro spolehlivý multicast a některé z nich byly v této práci analyzované. Na základě analýzy byly protokoly rozděleny do třech skupin dle základního principu: tree-based protokoly, skupinové komunikující protokoly a protokoly bez zpětné vazby.

Dále začala fáze hledání implementace těchto protokolů. Ale ukázalo se, že většina protokolů z průzkumu nemá implementaci a v praxi se využívají programy postavené na svých protokolech, které nemají specifikaci v oficiálních zdrojích, například v RFC. Konkrétně byly nalezeny implementace protokolu FLUTE a PGM, který ale nemá nástroj pro přenos souboru, a programy UFTP a UDPcast, které mají své protokoly.

V Cisco laboratoři FIT bylo realizováno testovací prostředí vhodné k testování pro ty implementace, které mají nástroj pro vysílání souboru. Návrh a hlavně realizace testovacího prostředí, které zabralo hodně času, je popsáno v rámci této bakalářské práce a může být užitečná pro ty, kteří budou provádět podobné zkoumání. Testovací prostředí umožňuje emulovat WAN linky nastavováním zpoždění a ztrátovostí paketů, což je využité při testování spolehlivosti přenosu.

Pro testované programy UFTP, UDPcast a FLUTE byly nalezeny vhodné konfigurace, podle kterých byly napsané skripty pro automatické testování umožňující sběr provozu včetně statistiky. Tyto skripty byly použity při provedení experimentu navrženého v této práci, cílem kterého bylo vyzkoušet tyto testované programy a srovnat jejich efektivitu při přenosu dat.

Hlavním faktorem ovlivňujícím spolehlivost přenosu je ztrátovost paketů, která může být způsobena zachycením v síti, omezením přenosové rychlosti na kanálu nebo špatnou konfigurací zařízení v síti. Dle srovnání výsledků experimentu s nastaveným poměrně velkým zpožděním a ztrátovostí paketů můžeme posoudit testované programy.

UFTP ukázal v našem testovacím prostředí maximální efektivitu podle času i podle malé velikosti přenášených dat. Funguje dle dokumentace, jak má a je stále ve vývoji. Je v současné době nejlepším kandidátem pro posílání souborů v lokálních sítích.

UDPcast je podobný UFTP. Pro malé přenosové rychlosti má podobnou efektivitu, ale pro velké přenosové rychlosti se v rámci experimentu ukázalo, že efektivita výrazně klesá.

FLUTE má hlavní výhodu v neomezené škálovatelnosti z důvodu absence zpětné vazby. Může být použit pro libovolné sítě, ale v rámci lokálních sítí, jak se ukázalo při testování, není tak efektivní.

Zdroje

1. BOUŠKA, Petr. *TCP/IP - skupinové vysílání IP Multicast a Cisco* [online]. 2009 [cit. 2017-02-22]. Dostupné z: http://www.samuraj-cz.com/clanek/tcpip-skupinove-vysilani-ip-multicast-a-cisco/.
2. CLARK, D. D.; TENNENHOUSE, D. L. Architectural Considerations for a New Generation of Protocols. *SIGCOMM Comput. Commun. Rev.* 1990, roč. 20, č. 4, s. 200–208. ISSN 0146-4833. Dostupné z DOI: 10.1145/99517.99553.
3. COTTON, M.; VEGODA, L.; MEYER, D. *IANA Guidelines for IPv4 Multicast Address Assignments*. 2010. Dostupné z DOI: 10.17487/RFC5771. RFC.
4. FAIRHURST, Prof. Godred. *Unicast, Broadcast, and Multicast* [online]. 2009 [cit. 2017-02-20]. Dostupné z: http://www.erg.abdn.ac.uk/users/gorry/course/intro-pages/uni-b-mcast.html.
5. GEMMELL, J.; MONTGOMERY, T.; SPEAKMAN, T.; CROWCROFT, J. The PGM Reliable Multicast Protocol. *Netwrk. Mag. of Global Internetwkg.* 2003, roč. 17, č. 1, s. 16–22. ISSN 0890-8044. Dostupné z DOI: 10.1109/MNET.2003.1174173.
6. GRÖNDAHL, Jan. Reliable Multicast Transport - The new modular and highly scalable protocols. 2005. Dostupné také z: http://www.tml.tkk.fi/Publications/C/18/grondahl.pdf.
7. JACOBSON, Van; MCCANNE, Steve; FLOYD, Sally. *Lightweight Sessions - A new architecture for realtime applications and protocols*. 3rd Annual Principal Investigators Meeting, ARPA, Santa Rosa, CA, 1993. Dostupné také z: ftp://ee.lbl.gov/talks/vj-nws93-2.ps.Z.
8. KOMOSNÝ, Dan. *Nové směry vývoje protokolu RTP/RTCP pro rozsáhlé konference v Internetu* [online]. 2004 [cit. 2017-02-20]. Dostupné z: http://www.elektrorevue.cz/clanky/04052/index.html.

9. LÁZNIČKA, Jakub. *Nasazení IPTV do počítačové sítě*. Praha, 2014. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií.

10. LEDVINA, Jiří. *Protokoly pro spolehlivý multicast* [online]. 2006 [cit. 2017-01-08]. Dostupné z: `http://docplayer.cz/4739699-Protokoly-pro-spolehlivy-multicast.html`.

11. LUBY, M.; WATSON, M.; VICISANO, L. *Forward Error Correction (FEC) Building Block*. 2007. Dostupné také z: `http://tools.ietf.org/html/rfc3452`. RFC.

12. LUBY, M.; WATSON, M.; VICISANO, L. *Layered Coding Transport (LCT) Building Block*. 2009. Dostupné také z: `http://tools.ietf.org/html/rfc3451`. RFC.

13. LUBY, M.; WATSON, M.; VICISANO, L. *Asynchronous Layered Coding (ALC) Protocol Instantiation*. 2010. Dostupné také z: `http://tools.ietf.org/html/rfc5775`. RFC.

14. MANSFIELD, K. C.; ANTONAKOS, J. L. *Computer Networking from LANs to WANs: Hardware, Software, and Security*. Boston: Course Technology, Cengage Learning, 2009. ISBN 978-1423903161.

15. MOLENAAR, René. *Multicast IP Address to MAC address mapping* [online] [cit. 2017-03-22]. Dostupné z: `https://networklessons.com/multicast/multicast-ip-address-to-mac-address-mapping/`.

16. NONNENMACHER, Jörg; BIERSACK, Ernst W.; TOWSLEY, Don. Parity-based Loss Recovery for Reliable Multicast Transmission. *IEEE/ACM Trans. Netw.* 1998, roč. 6, č. 4, s. 349–361. ISSN 1063-6692. Dostupné z DOI: `10.1109/90.720869`.

17. PAILA, T.; LUBY, M.; LEHTONEN, R.; ROCA, V.; WALSH, R. *FLUTE - File Delivery over Unidirectional Transport*. 2004. Dostupné také z: `http://tools.ietf.org/html/rfc3926`. RFC.

18. PAILA, T.; WALSH, R.; LUBY, M.; ROCA, V.; LEHTONEN, R. *FLUTE - File Delivery over Unidirectional Transport*. 2012. Dostupné také z: `http://tools.ietf.org/html/rfc6726`. RFC.

19. PAUL, Sanjoy. *Multicasting on the Internet and Its Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1998. ISBN 0792382005.

20. RIZZO, Luigi. Effective Erasure Codes for Reliable Computer Communication Protocols. *SIGCOMM Comput. Commun. Rev.* 1997, roč. 27, č. 2, s. 24–36. ISSN 0146-4833. Dostupné z DOI: `10.1145/263876.263881`.

21. SPEAKMAN, T.; CROWCROFT, J.; GEMMELL, J.; FARINACCI, D.; LIN, S.; LESHCHINER, D.; LUBY, M.; MONTGOMERY, T.; RIZZO, L. *PGM Reliable Transport Protocol Specification*. 2001. Dostupné také z: `http://tools.ietf.org/html/rfc3208`. RFC.

Dodatek A

Seznam použitých zkratek

LAN	Local Area Network
IP	Internet Protocol
MAC	Media Access Control
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
TTL	Time To Live
IGMP	Internet Group Management Protocol
PIM	Protokol Independent Multicast
FEC	Forward Error Correction
ARQ	Automatic Repeat Request
RFC	Request for Comments
RMTP	The Reliable Multicast Transport Protocol
SRM	Scalable Reliable Multicast
PGM	Pragmatic General Multicast
FLUTE	File Delivery over Unidirectional Transport
TLS	Transport Layer Security
SSH	Secure Shell
STP	Spanning Tree Protocol
GNU	GNU's Not UNIX
GPL	GNU General Public License

Dodatek B

Skripty

B.1 Konfigurace routerů

B.1.1 Router R1

```
en
conf t
hostname R1
ip multicast-routing
int Gi0/0
ip address 10.0.0.1 255.255.255.0
ip pim sparse-dense-mode
no sh
int Gi0/1
ip address 10.0.1.1 255.255.255.0
ip pim sparse-dense-mode
no sh
exit
ip route 10.0.2.0 255.255.255.0 10.0.1.2
ip route 10.0.3.0 255.255.255.0 10.0.1.2
```

B.1.2 Router R2

```
en
conf t
hostname R2
```

```
ip multicast-routing
int Gi0/0
ip address 10.0.1.2 255.255.255.0
ip pim sparse-dense-mode
no sh
int Gi0/1
ip address 10.0.2.1 255.255.255.0
ip pim sparse-dense-mode
no sh
exit
ip route 10.0.0.0 255.255.255.0 10.0.1.1
ip route 10.0.3.0 255.255.255.0 10.0.2.2
```

B.1.3 Router R3

```
en
conf t
hostname R3
ip multicast-routing
int Gi0/0
ip address 10.0.2.2 255.255.255.0
ip pim sparse-dense-mode
no sh
int Gi0/1
ip address 10.0.3.1 255.255.255.0
ip pim sparse-dense-mode
no sh
exit
ip route 10.0.0.0 255.255.255.0 10.0.2.1
ip route 10.0.1.0 255.255.255.0 10.0.2.1
```

B.2 Komunikace s počítači

B.2.1 Skript *send_cmd.sh*

```
#!/bin/bash
[[ ! "$1" ]] && { echo "PC last oktet missing" ; exit ; }
PC_NUM="$1"
```

```
shift
echo -e "$PC_NUM\tCMD: ${@}"
sshpass -p"net123" ssh -o StrictHostKeyChecking=no \
student@192.168.0.$PC_NUM \
echo "net123" \| sudo -p "'' -S bash -c '${@}'"
```

B.2.2 Skript *copy_files.sh*

```
#!/bin/bash
[[ ! "$1" ]] && { echo "PC last oktet missing" ; exit ; }
PC_NUM="$1"
shift
echo "${@}"
sshpass -p"net123" scp -o StrictHostKeyChecking=no \
-r "${@}" student@192.168.0.${PC_NUM}:"/mnt/"
```

B.3 Konfigurace serveru a počítačů

B.3.1 Skript *init_pc.sh*

```
[[ ! "$1" ]] && { echo "PC subnet missing" ; exit ; }
[[ ! "$2" ]] && { echo "PC last oktet missing" ; exit ; }
SUBNET="$1"
PC_NUM="$2"
PC_ADDR="${SUBNET}.$PC_NUM"
MASK="24"
ARR+=("ifconfig eth1 $PC_ADDR/$MASK up")
ARR+=("route del default gw 0.0.0.0")
ARR+=("route add default gw $SUBNET.1")
# Make partition for testing files
ARR+=("parted /dev/sda mkpart primary 10GiB 30GiB")
ARR+=("mkfs.ext4 /dev/sda3")
ARR+=("mount /dev/sda3 /mnt")
ARR+=("chmod 0777 /mnt")
# Increase /tmp size for Wireshark
ARR+=("parted /dev/sda mkpart primary 30GiB 50GiB")
ARR+=("mkfs.ext4 /dev/sda4")
ARR+=("mount /dev/sda4 /tmp")
```

```
ARR+=("chmod 0777 /tmp")
for cmd in "${ARR[@]}" ; do
./send_cmd.sh $PC_NUM $cmd
done
```

B.3.2 Skript *serv_pcs_preparation.sh*

```
#!/bin/bash
for PC in 11 101 ; do
./init_pc.sh 10.0.0 $PC
done
for PC in 102 ; do
./init_pc.sh 10.0.1 $PC
done
for PC in 103 104 ; do
./init_pc.sh 10.0.2 $PC
done
for PC in 105 ; do
./init_pc.sh 10.0.3 $PC
done
```

B.4 Konfigurace mostu

B.4.1 Skript *init_br.sh*

```
#!/bin/bash
[[ ! "$1" ]] && { echo "PC last oktet missing" ; exit ; }
ARR+=("ifconfig eth1 0.0.0.0 up")
ARR+=("ifconfig eth2 0.0.0.0 up")
ARR+=("brctl addbr br0")
ARR+=("brctl addif br0 eth1")
ARR+=("brctl addif br0 eth2")
ARR+=("brctl stp br0 off")
ARR+=("ifconfig br0 up")
ARR+=("echo 0 \
>/sys/device/virtual/net/br0/bridge/multicast_snooping")
for cmd in "${ARR[@]}" ; do
./send_cmd $1 $cmd
```

```
done < "bridge_init_cmds.txt"
```

B.5 Testovací skripty

B.5.1 Skript *test_uftp.sh*

```
#/bin/bash
SERV_ADDR=11
C_ARR=( 101 102 103 105 )
ALL_ARR=( $SERV_ADDR 101 102 103 104 105 )
SDATE="$(date +%F_%T)"
UFTP="/mnt/Programs/uftp-4.9.3/uftp"
UFTPD="/mnt/Programs/uftp-4.9.3/uftpd"
F_PATH="/mnt/TestFiles/"
POSTFIX="UFTP_${F_NAME}_${SDATE}.pcapng"
F_NAME="bash4x.txt"
CMD="$UFTP -I eth1 -t 100 -R 20000 $F_PATH/$F_NAME"

SAVE_DIR="/tmp/UFTP_${F_NAME}_${SDATE}"
mkdir $SAVE_DIR
echo "$CMD" >> "${SAVE_DIR}/cmd.txt"

for i in "${ALL_ARR[@]}" ; do
./send_cmd.sh $i dumpcap -g -i eth1 -w \
"/tmp/PC_${i}_$POSTFIX" 2>/dev/null &
done

for i in "${C_ARR[@]}" ; do
./send_cmd.sh $i killall uftpd 2>/dev/null 1>2
./send_cmd.sh $i $UFTPD -I eth1
done

./send_cmd.sh $SERV_ADDR "$CMD" 2>"$SAVE_DIR/serv_log.txt"

for i in "${ALL_ARR[@]}" ; do
./send_cmd.sh $i 'killall dumpcap'
./send_cmd.sh $i "chmod 777 /tmp/PC_${i}_$POSTFIX"
sshpass -p"net123" scp -o StrictHostKeyChecking=no \
```

```
 student@192.168.0.${i}:"/tmp/*_${POSTFIX}" "$SAVE_DIR"
done
```

B.5.2 Skript *test_udpcast.sh*

```
#/bin/bash
SERV_ADDR=11
C_ARR=(101 102 103 105)
ALL_ARR=( $SERV_ADDR 101 102 103 104 105 )
SDATE="$(date +%F_%T)"
POSTFIX="UDPCAST_${F_NAME}_${SDATE}.pcapng"
SAVE_DIR="/tmp/UDPCAST_${F_NAME}_${SDATE}"
UDP_SEND="/mnt/Programs/udpcast-20120424/udp-sender"
UDP_RECV="/mnt/Programs/udpcast-20120424/udp-receiver"
F_PATH="/mnt/TestFiles/"
F_NAME="bash4x.txt"

SCMD="$UDP_SEND --interface eth1 --mcast-rdv-address 224.2.2.4 "
SCMD+="--ttl 100 --nokbd --max-bitrate ${1}k "
SCMD+="--autostart 20 --file $F_PATH$F_NAME"
RCMD="$UDP_RECV --interface eth1 --mcast-rdv-address 224.2.2.4 "
RCMD+="--ttl 100 --nokbd --file /tmp/$F_NAME"

mkdir $SAVE_DIR
echo "$SCMD" >> "${SAVE_DIR}/cmd.txt"

for i in "${ALL_ARR[@]}" ; do
./send_cmd.sh $i dumpcap -g -i eth1 -w \
"/tmp/PC_${i}_$POSTFIX" 2>/dev/null & #Dumpcap start
done

for i in "${C_ARR[@]}" ; do
./send_cmd.sh $i killall udp-receiver 2>/dev/null
./send_cmd.sh $i "$RCMD" 2>/dev/null & #Klient start
done

./send_cmd.sh 11 killall udp-sender
# Server start
```

```
./send_cmd.sh 11 "$SCMD" 2>"$SAVE_DIR/serv_log.txt"

for i in "${ALL_ARR[@]}" ; do # Save dumps
./send_cmd.sh $i 'killall dumpcap'
./send_cmd.sh $i "chmod 777 /tmp/PC_${i}_$POSTFIX"
sshpass -p"net123" scp -o StrictHostKeyChecking=no \
 student@192.168.0.${i}:"/tmp/*_${POSTFIX}" "$SAVE_DIR"
done
```

B.5.3 Skript *test_flute.sh*

```
#/bin/bash
SERV_ADDR=11
C_ARR=( 101 102 103 105 )
ALL_ARR=( $SERV_ADDR 101 102 103 104 105 )
RECV_PIDS=()
SDATE="$(date +%F_%T)"
POSTFIX="FLUTE_${F_NAME}_${SDATE}.pcapng"
SAVE_DIR="/tmp/FLUTE_${F_NAME}_${SDATE}"
FLUTE="/mnt/Programs/mad_fcl_v1.7_linux_bin/flute"
F_PATH="/mnt/TestFiles/"
F_NAME="bash4x.txt"
SCMD="$FLUTE -S -m:239.1.1.1 -p:4000 -T:100 -r:30000"
SCMD+=" -x:2 -X:50 -B:$F_PATH -F:$F_NAME -C"
RCMD="$FLUTE -I:eth1 -m:239.1.1.1 -p:4000"
RCMD=" -B:/tmp -F:file:///$F_NAME"

mkdir $SAVE_DIR
echo "$SCMD" >> "${SAVE_DIR}/cmd.txt"

for i in "${ALL_ARR[@]}" ; do
./send_cmd.sh $i dumpcap -g -i eth1 -w \
"/tmp/PC_${i}_$POSTFIX" 2>/dev/null &
done

for i in "${C_ARR[@]}" ; do
./send_cmd.sh $i killall flute 2>/dev/null
./send_cmd.sh $i "$RCMD" 2>/dev/null &
```

```
RECV_PIDS+=("$!")
done

./send_cmd.sh $SERV_ADDR killall flute 2>/dev/null
./send_cmd.sh $SERV_ADDR "$SCMD" 1>>"$SAVE_DIR/serv_log.txt" &

wait ${RECV_PIDS[@]}
./send_cmd.sh $SERV_ADDR 'killall flute' 2>/dev/null

for i in "${ALL_ARR[@]}" ; do
./send_cmd.sh $i 'killall dumpcap' 2>/dev/null
./send_cmd.sh $i "chmod 777 /tmp/PC_${i}_$POSTFIX"
sshpass -p"net123" scp -o StrictHostKeyChecking=no \
 student@192.168.0.${i}:"/tmp/*_${POSTFIX}" "$SAVE_DIR"
done
```

Dodatek C

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu XeLaTeX
  - `Skripts` ..... skripty pro testování
    - `Pkgs` ..... knihovny pro program flute
    - `Programs` ..... programy k testování
    - `TestFiles` ..... testovací data
- `BP_Maher_Uladzislau_2017.pdf` ..... text práce ve formátu PDF

Adresářová struktura C.1: Obsah přiloženého média