MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Komunikační vrstva aktivního směrovače

DIPLOMOVÁ PRÁCE

Tomáš Rebok

Brno, 2004

Prohlášení

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

Vedoucí práce: RNDr. Eva Hladká

Poděkování

Mnohokrát děkuji své vedoucí RNDr. Evě Hladké, Ph.D. za rady a připomínky a všechen čas, který mi věnovala. Dále bych chtěl poděkovat Mgr. Zdeňku Salvetovi za jeho cenné připomínky k návrhu protokolu.

Můj dík za veškerou pomoc patří také mým kolegům z Laboratoře pokročilých síťových technologií Fakulty informatiky – především Mgr. Jiřímu Denemarkovi, Mgr. Davidovi Antošovi a Mgr. Petrovi Holubovi.

Shrnutí

Počítačové sítě budoucnosti musí být rychlejší a flexibilnější než ty dnešní. Jednou z možností, jak v sítích flexibilitu zajistit, jsou aktivní (neboli programovatelné) sítě. Tento nový přístup dělá z pasivního transportního média, jakým jsou stávající sítě, programovatelný distribuovaný výpočetní prostředek s řadou nových možností, ale i nových problémů. Bylo navrženo více architektur, jak aktivní sítě implementovat. Jednou z nich je architektura založená na aktivním uzlu a aktivních paketech, která se stala základem návrhu modelu aktivní sítě pro prostředí PC směrovačů s operačním systémem NetBSD. Hlavním úkolem těchto směrovačů je, stejně jako ve všech stávajících sítích, přenos dat od vysílajícího uzlu směrem k příjemci. A právě pro tuto komunikaci jsou zapotřebí protokoly, které tento přenos s garancí požadované kvality služby zajistí.

Hlavním cílem práce je navrhnout, specifikovat a implementovat komunikační (transportní) protokol použitelný pro komunikační vrstvu aktivního směrovače vyvíjeného na Masarykově univerzitě v Brně. Přestože transportních protokolů existuje velké množství, málokterý vyhovuje speciálním požadavkům vyvíjeného směrovače (zejména požadavku spolehlivého doručování dat bez nutnosti zachování jejich pořadí). Vyvinutý protokol, ačkoli je primárně určen pro potřeby aktivního směrovače, je navržen jako dostatečně obecný, aby jej bylo možno použít i mimo prostředí aktivních sítí.

Dále práce stručně popisuje historii a vývoj počítačových sítí, detailněji se také věnuje architektuře aktivních sítí; především se zaměřuje na aktivní sítě se zaváděnými funkcemi a prostředí programovatelných směrovačů.

Klíčová slova

počítačové sítě, aktivní sítě, aktivní směrovač, sítě se zaváděnými funkcemi, transportní protokol, spolehlivý přenos dat bez zachování pořadí.

Místo této strany bude vložena kopie zadání.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Již od prvopočátku mělo lidstvo potřebu komunikovat. Zpočátku se uplatňovala pouze přímá komunikace pomocí mluveného slova. Mnohem později, když bylo potřeba komunikovat na větší vzdálenosti, se informace začaly psát na papír, který tento problém díky možnosti jeho zasílání částečně vyřešil. Vyskytl se však problém nový, a tím byla rychlost této komunikace. Začalo se uvažovat o efektivnějším způsobu – byl vynalezen telefon, který komunikaci na větší vzdálenosti výrazně urychlil. S příchodem počítačů se začalo uvažovat o jejich využití pro efektivní a rychlou komunikaci mezi lidmi, čímž by byl problém se vzdáleností téměř odbourán. Začaly vznikat první počítačové sítě, které byly zpočátku určeny pouze pro přenos souborů s daty. Až později byly navrženy mechanismy, jak sítě využít i pro komunikaci.

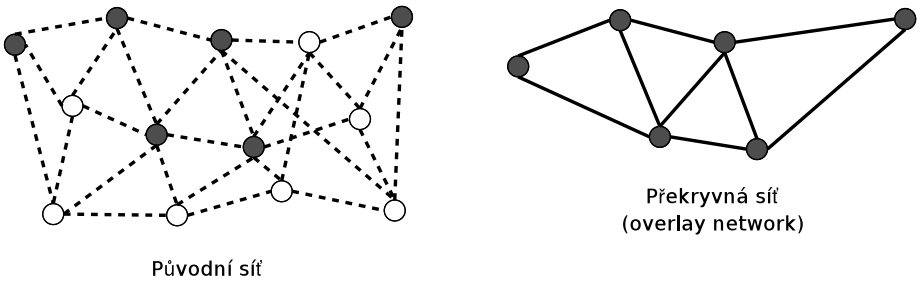
Hlavním úkolem dnešních paketových sítí je přeposílání datových paketů směrem k místu jejich určení. Všechny výpočty uvnitř sítě se však omezují pouze na úkony spojené s přeposíláním procházejících paketů podle zadaných pravidel, nikoli na případné zpracování jejich obsahu. Tyto pasivní sítě jsou charakterizovány zásadou *end-to-end argument*, který zdůrazňuje, že funkcionalitu požadovanou aplikací po síti je možno zajistit pouze se znalostí a prostřednictvím samotné aplikace (více viz [3]). To znamená, že po síti nemůžeme chtít nic víc, něž jen vlastní přenos dat – všechny možné výpočty si musí aplikace dělat samy. Existují aplikace, kterým tento přístup vůbec nevyhovuje a které určitou aktivitu v síti vyžadují. Takovou aplikací jsou například videokonference – uzly ideální sítě jsou dodržením definovaných vlastností spojení (maximálního zpoždění, jeho rozptylu či potřebné šířky pásma) schopny potřebnou kvalitu služby zajistit. V případě velkého zatížení sítě však lze tyto požadavky dodržet pouze stěží. Proto by bylo vhodné, aby uzly, přes které data procházejí, změnily například kódování přenášeného videa a tím i objem přenášených dat. Podobných aplikací lze najít mnohem více.

Jako možné řešení zmíněných požadavků vznikla myšlenka programovatelných sítí, které by aplikacím umožnily řídit si na síťových uzlech zpra-

cování svých dat. Tyto sítě se souhrnně nazývají aktivní sítě. Protipólem aktivních sítí jsou dnešní široce používané sítě – sítě pasivní, jejichž uzly provádějí pouze velmi omezené množství operací souvisejících s přeposíláním paketů.

Může se zdát, že aktivní sítě jsou protikladem sítí postavených se zásadou *end-to-end argument*, tj. že díky možnosti vykonávat libovolné výpočty uvnitř sítě jej zneplatňují. To ovšem není pravda, poněvadž existují aplikace, které právě *end-to-end* přístup vyžadují. Typickým příkladem je zabezpečená komunikace, u které musí být na koncových uzlech uloženo sdílené tajemství. Těžko si však lze představit, že by toto tajemství bylo rozprostřeno po celé síti, aby mohly aktivní prvky nad přenášeným tokem dat provádět nějaké výpočty.

Architektura aktivních sítí však s sebou přináší také určité nevýhody, zejména v pomalejším zpracování paketů a větších požadavcích na zabezpečení. Proto vznikla myšlenka spojit výhody obou typů sítí vybudováním aktivní sítě nad stávající pasivní architekturou tak, že některé uzly budou aktivní a ostatní zůstanou pasivní. Této architektuře se říká překryvná síť (anglicky *overlay network*) a její detailní popis lze nalézt v [18]. Jedná se v podstatě o virtuální síť vytvořenou nad původní sítí, která poskytuje specifickou funkcionalitu. Původní síť se využívá pouze pro přenos paketů – jako spojení mezi uzly překryvné sítě. Informace specifické pro překryvnou síť jsou obsaženy v datové části paketů původní sítě, takže je tato síť zpracovává, aniž by o překryvné síti věděla.



Obrázek 1.1: Příklad překryvné sítě

Model překryvných sítí předpokládá i architektura aktivního směrovače vyvíjeného na Masarykově univerzitě, jehož součástí má být transportní protokol popisovaný v této práci.

1.1 Cíl práce

Cílem této práce je navrhnout a implementovat komunikační (transportní) protokol pro aktivní směrovač vyvíjený na Masarykově univerzitě (dle [10]). Navržený protokol by měl splňovat všechny požadavky na něj kladené, ale měl by být navržen dostatečně univerzálně tak, aby byl schopen pracovat i s jinými transportními protokoly, nad něž může být posazen (např. UDP). Zároveň by měl umožňovat jednoduchou rozšiřitelnost použitím více adresních protokolů (IPv4, IPv6, aj.).

1.2 Rozvržení kapitol

První kapitola této práce je věnována stručnému popisu principů počítačových sítí a jejich historii spolu s motivací pro zavedení nové architektury aktivních sítí. Druhá kapitola se věnuje popisu principů aktivních sítí, jejich využitelnosti a jejich základním architekturám. Třetí kapitola je věnována stručnému popisu architektury aktivního směrovače, jenž je vyvíjen na Masarykově univerzitě, s cílem nastínit použití navrhovaného transportního protokolu. Ve čtvrté kapitole jsou popsány některé již existující transportní protokoly podobné navrhovanému protokolu, jejich výhody a nevýhody pro použití v architektuře aktivního směrovače. Pátá kapitola se věnuje specifikaci navrhovaného protokolu, v předposlední, šesté, kapitole je popsána jeho vzorová implementace spolu s testy měřícími její efektivitu. Závěrečná kapitola pak shrnuje výsledky dosažené touto prací.

Kapitola 2

Vývoj počítačových sítí

Vývoj počítačových sítí započal v 60. letech. V roce 1962 byla Paulem Baranem poprvé představena idea paketových přepínaných sítí, na jejichž základě byla v roce 1969 vybudována první paketová přepínaná počítačová síť s názvem ARPANET. Její první uzel byl instalován na University of California v Los Angeles. Na konci roku 1969 byly do sítě ARPANET připojeny další tři uzly – Stanford Research Institute ve Stanfordu, University of California v Santa Barbaře a University of Utah. Síť byla tvořena 50 Kbps linkou a byla využívána především pro výměnu dat mezi připojenými uzly. V letech 1971 měla síť ARPANET již patnáct uzlů, v roce 1973 třicet sedm. V těchto letech začala být síť využívána také pro zasílání emailových zpráv jako prostředek pro vzdálenou komunikaci spolupracovníků v různých částech světa. Jakmile bylo zřejmé, že bude do sítě potřeba zapojit mnohem více uzlů, začalo se uvažovat o změně stávajícího end-to-end komunikačního protokolu nazývaného NCP (*Network Control Protocol*) na novější, tvořený dvojicí protokolů nazývaných TCP (*Transmission Control Protocol*) a IP (*Internet Protocol*). Oba tyto protokoly byly vytvořeny skupinou vedenou Vintonem Cerfem ze Stanfordu a Bobem Kahnem ze sdružení DARPA v roce 1973. Protokol IP měl být zodpovědný za přenos a směrování paketů mezi více sítěmi a poskytovat služby protokolu TCP, jehož hlavním úkolem měl být převod přenášených zpráv na proud paketů na straně odesílatele a jejich zpětné složení na straně příjemce. Vzhledem k velké chybovosti tehdejších sítí bylo dalším úkolem TCP zajistit spolehlivé doručení všech paketů (včetně zachování jejich pořadí) od vysílajícího uzlu až k přijímacímu. Tyto dva protokoly zajistily spolehlivou komunikaci na tehdejších sítích a vzhledem k jejich velkému úspěchu jsou, byt's drobnými změnami, na sítích používány dodnes.

Do počátku 70. let se struktura sítě opírala o centralizovanou architekturu. Síť vždy obsahovala jeden výkonný počítač (jednalo se výhradně o sálový počítač), který vykonával veškerou výpočetní práci a několik terminálů, které sloužily pro vstup a výstup dat. Tato síť byla relativně jednoduchá. Všechny obtížné problémy, jako například synchronizace procesů,

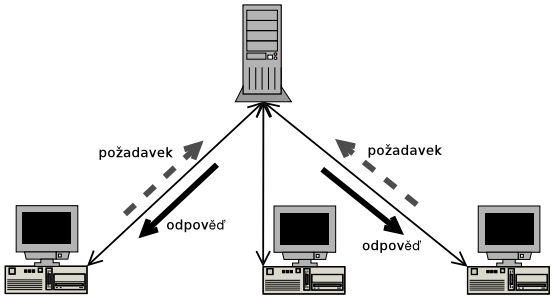
řízení přístupu k datům a ostatním prostředkům, ochrana integrity, se řešily v jediném uzlu pomocí technik operačních systémů. Počátkem 70. let začaly být na počítačové sítě kladeny náročnější požadavky – měly zajistit vzájemnou komunikaci libovolného uživatele s programem na jiném počítači, dvou programů mezi sebou nebo dvou libovolných uživatelů mezi sebou (a to vše při vysoké rychlosti a spolehlivosti komunikace).

V roce 1976 navrhl Dr. Robert M. Metcalfe ethernet – síťovou architekturu, která pomocí koaxiálního kabelu umožňovala přenášet data po síti na tehdejší dobu extrémní rychlostí (při jeho prvním použití dosahovaly linky rychlosti až 1,5 Mbps). Tato architektura se stala rozhodující součástí pro návrh sítí LAN (*Local Area Network*). Vznikají první komplexní výpočetní systémy, které zajišťují různé služby (počítače, které byly součástí počítačové sítě, mohly sdílet data, zprávy, grafiku, tiskárny a další hardwarové zdroje). V těchto letech se sítě začaly velmi rychle rozšiřovat. V roce 1987 počet připojených účastníků pokořil hranici 10000, v roce 1989 bylo do sítě připojeno již přes 100 tisíc účastníků a už v roce 1992 byla překonána tehdejší magická hranice 1 milión připojených uzlů.

Sítě LAN byly navrženy pro dvě základní, dodnes používané, architektury:

klient-server

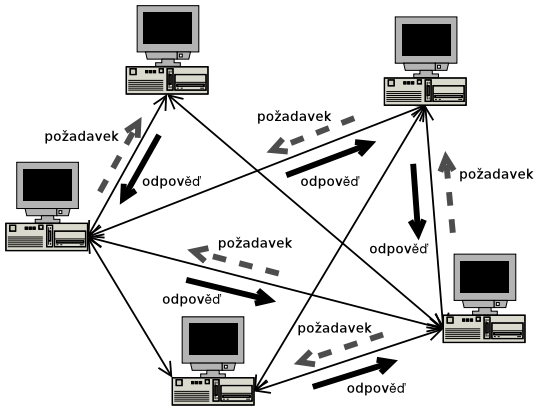
V této architektuře slouží jeden počítač jako server, který na požádání poskytuje služby ostatním v síti zapojeným počítačům. Schéma komunikace v této architektuře je naznačeno na obrázku 2.1.



Obrázek 2.1: Model klient-server

peer-to-peer

Peer-to-peer architektura je decentralizovaný systém, ve kterém mají všechny počítače rovnoprávné postavení – všechny mohou nabízet své služby ostatním. Tato architektura je znázorněna na obrázku 2.2.

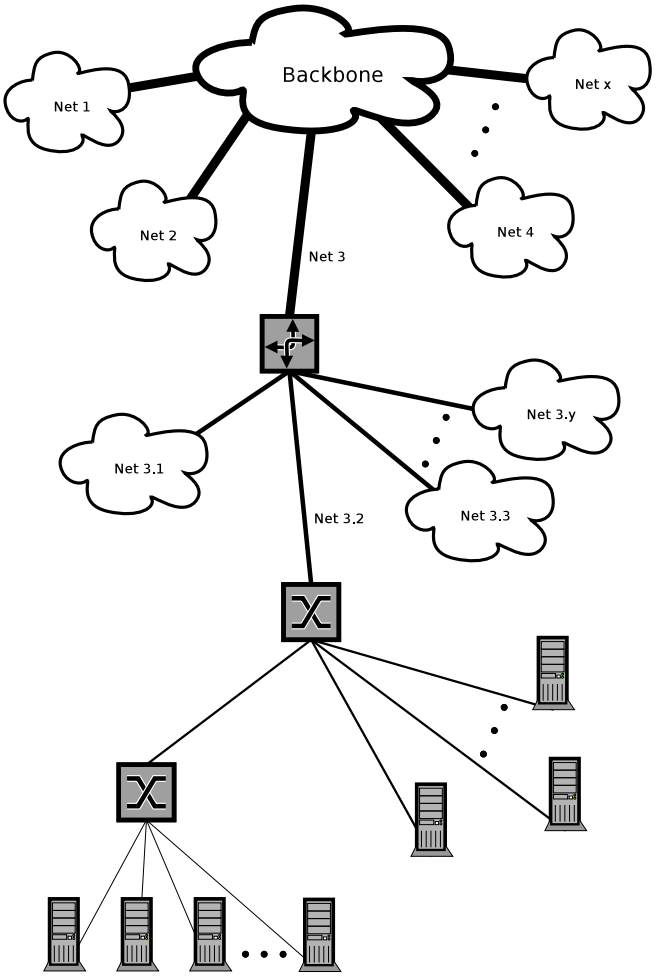


Obrázek 2.2: Model peer-to-peer

Po krátké době přestaly lokální sítě vyhovovat velkým společnostem, které měly svá sídla v různých státech. Proto vznikly sítě MAN (*Metropolitan Area Network*, sítě v rozsahu jednoho města) a sítě WAN (*Wide Area Network*, sítě pokrývající celou planetu) – tyto sítě v podstatě sdružují sítě LAN do větších celků pomocí speciálních propojovacích prvků (více počítačových sítí navzájem propojených a schopných komunikace se anglicky označuje pojmem *internetwork*). Tato infrastruktura, ve které jsou menší (například firemní) sítě spojovány do větších celků (např. městské sítě), jež jsou součástí celků dalších až k páteřní síti (*backbone*), se používá z důvodu řady výhod, které poskytuje – zatížení lokální sítě nezatěžuje páteř ani ostatní větve, výpadek lokální sítě neohrozí zbytek sítě, apod. Její nástin je zobrazen na obrázku 2.3 (obrázek je pouze ilustrativní, skutečná architektura současných mezinárodních sítí je mnohem složitější).

Diskuze uvnitř výzkumného sdružení DARPA, konané v letech 1994 a 1995 na téma budoucího rozvoje síťových systémů, odhalily několik problémů dosavadních sítí – například obtížnost integrace nových technologií a standardů do sdílené síťové infrastruktury, slabý výkon sítí díky nadbytečným operacím na více protokolových vrstvách či obtížnost vyhovět novým službám v existující architektuře. Z nich poprvé vyplynul pojem aktivní sítě jako jednotný název pro více strategií určených k vyřešení odhalených

problémů. Byla navržena idea zpráv přenášejících data i procedury, které mají být na data aplikovány. Toto nové prostředí, ve kterém lze na síťových uzlech nad procházejícími daty provádět požadované operace, umožňuje vznik speciálních síťových systémů, které jsou utvořeny z mnoha menších komponent se specifickými vlastnostmi. Služby mohou být distribuovány a konfigurovány na základě požadavků aplikací, díky čemuž může být tato síť rychle adaptována na změny požadavků.



Obrázek 2.3: Propojování sítí do větších celků

Kapitola 3

Aktivní (programovatelné) sítě

3.1 Princip

Aktivní sítě (jinak známé také pod pojmem programovatelné sítě) jsou novým přístupem k síťové architektuře, v níž aktivní prvky sítě vykonávají požadované operace na zprávách, které přes ně procházejí. Tato snaha je motivována především dvěma hlavními aplikacemi této architektury – možností provádět uživatelské operace nad daty procházejícími přes síťové uzly a prudkým rozvojem mobilních technologií, které vyžadují dynamické změny síťových služeb.

Jak již bylo uvedeno, aktivní sítě jsou aktivní v tom smyslu, že jejich uzly mohou provádět různé modifikace a operace nad obsahem procházejících paketů (například měnit šifrování jejich datové části). Vzhledem k tomu, že si tyto operace volí sama aplikace (resp. uživatel), která data vygenerovala, lze toto zpracování provádět na uživatelském základě. Každý aplikací vygenerovaný tok dat může mít definovány jiné operace, které se na něm mají v průběhu cesty k příjemci vykonat. Naproti tomu, v tradičních (pasivních) sítích jsou možnosti zpracování průchozích dat velmi omezené. Ačkoliv směrovače mohou modifikovat hlavičky paketů, uživatelská data ponechávají beze změn či jiných prohlídek obsahu[1]. Tyto modifikace hlaviček a další akce spojené s daným směrovačem jsou navíc prováděny nezávisle na uživatelském procesu či aplikaci, která data vygenerovala.

3.2 Základní architektury aktivních sítí

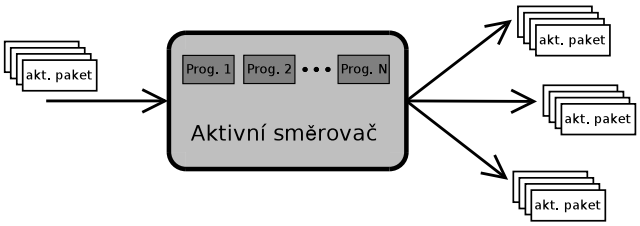
První navrženou architekturou aktivní sítě byla možnost výběru operací, které se měly nad přenášenými daty provést, ze seznamu všech operací, které daný směrovač podporoval. Tento přístup však brzy přerostl v myšlenku možnosti vkládání uživatelských programů do směrovače. A právě

[1] Některé specializované uzly, jako například firewally, jsou sice schopny provádět i operace nad uživatelskými daty, ovšem toto zpracování je pevně dané a uživatel ani aplikace jej nemají šanci ovlivnit.

způsobem vkládání programů do směrovače se základní architektury aktivních sítí odlišují.

Aktivní sítě s vestavěnými funkcemi

V této architektuře obsahují přenášená data identifikátor funkce (případně včetně jejích argumentů), která se má na směrovači nad přenášenými daty vyvolat. Zavedení funkcí do směrovače je však výhradní záležitostí jeho správce. Možná aktivita uživatele je tak omezena pouze na výběr vhodné funkce, kterou chce na přenášená data použít, případně pak nastavení hodnot jejích parametrů. Výhodou tohoto přístupu je, že se přenášený paket nemusí prodlužovat a ustavení spojení netrvá příliš dlouho. Podstatnou výhodou je také to, že vestavěné funkce lze snadno kontrolovat, čímž se zjednodušuje problém s bezpečností. Nevýhodou je malá flexibilita a to, že uživatel vlastně neprogramuje směrovač, ale pouze využívá jeho rozšířené funkcionality.

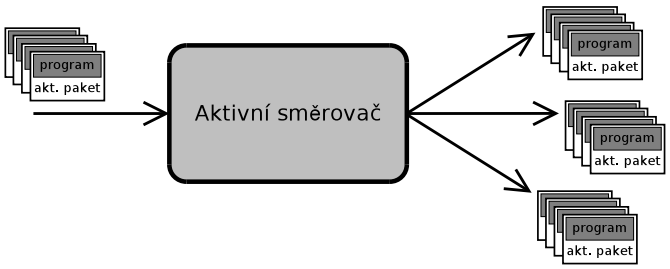


Obrázek 3.1: Aktivní sítě s vestavěnými funkcemi

Aktivní sítě se zapouzdřenými programy

Kód aktivního programu je umístěn přímo v datagramu, takže každý přenášený datagram je program. Implementací této ideje vznikl jazyk Netscript, jehož kód je vkládán přímo do datagramů. Na síť je pak nahlíženo jako na soustavu virtuálních síťových strojů spojených virtuálními spoji, což spolu s jazykem Netscript tvoří virtuální výpočetní prostředek, jenž je tímto jazykem programován. Do úvahy je však nutno vzít to, že program je součástí každého datového paketu a tím je tedy omezena jeho velikost.

Pro tuto architekturu se také vžil název aktivní sítě založené na aktivních paketech.

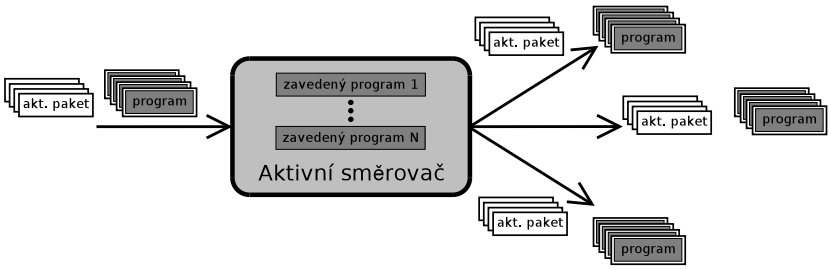


Obrázek 3.2: Aktivní sítě se zapouzdřenými programy

Aktivní sítě se zaváděnými funkcemi

Tato architektura umožňuje uživateli umístit do směrovače vlastní programy a spouštět je nad směrovanými daty. Uživatelské programy jsou před vlastním průchodem dat umístěny na směrovač a poté nad daty vykonávány. Tento přístup je velmi obecný. Jeho výhodou je, že se neprodlužuje délka paketů. Na druhé straně ovšem roste režie spojená s navázáním spojení a rozesláním programů do směrovačů před vlastním přenosem, a zvyšují se nároky na zajištění bezpečnosti takovýchto systémů.

Tuto architekturu lze kombinovat s architekturou založenou na aktivních paketech – po umístění programů na směrovači mohou přenášená data navíc obsahovat parametry pro své zpracování.



Obrázek 3.3: Aktivní sítě se zaváděnými funkcemi

Aktivní sítě se zapouzdřenými programy jsou jakousi obdobou nespojovaného přenosu datagramů v tradičních paketových sítích. Veškeré informace pro zpracování každého paketu jsou totiž přímo jeho součástí. Naproti tomu u obou zbývajících modelů (souhrnně se nazývají modely aktivních uzlů) lze s výhodou použít spojení (anglicky *session*) – obdobu spojení ve

spojovaných sítích či protokolu TCP. Během zahajování spojení je možné do aktivního uzlu nahrát potřebné programy. Poté se již posílají jen samotná data a podle spojení, ke kterému náleží, se na ně aplikují patřičné zpracovávací procedury.

3.3 Výhody a použití

Stejně jako v mnoha jiných oblastech je i výzkum aktivních sítí motivován především požadavky jejich uživatelů. Takovýmito „uživateli" mohou být firewally, webové proxy servery, multicastové směrovače, mobilní proxy, video brány, atd. – všechna tato zařízení mohou provádět uživatelem řízené operace na uzlech uvnitř sítě.

Hlavní oblasti, ve kterých mohou být aktivní sítě s výhodou použity, jsou následující:

Management sítě
: Aktivní uzly mohou uvnitř sledované sítě vytvořit podsíť, detekovat její anomálie a řídit její provoz. Zajímavé použití se otevírá i v oblasti sledování stavu sítě – síťové komponenty, jako například směrovače, na sebe mohou vzít částečnou zodpovědnost za monitorování sebe sama, například nahráním patřičných monitorovacích a diagnostických programů svým nejbližším sousedům.

Ochrana sítě na uživatelské úrovni
: Zabezpečení informací znamená, že se správná informace dostane ke správným lidem na správné místo, a navíc ve správném čase. Ačkoliv již bylo navrženo mnoho autentizačních a zabezpečujících mechanismů, aktivní sítě mohou umožnit návrh jednotného mechanismu, který pokrývá všechny síťové zdroje a informace přes ně procházející. Tímto můžeme odstranit nutnost více autentizačně-zabezpečujících systémů, které se aplikují nezávisle na sobě na různých síťových vrstvách. Tento přístup umožní naprogramovat speciální zabezpečující síťové mechanismy pro každého uživatele či každé použití.

Experimentální technologie
: Poněvadž aktivní sítě představují ideální prostředí pro budování překryvných sítí, lze pomocí aktivních programů snadno implementovat nové protokoly a služby a ověřovat jejich vlastnosti. Tempo zavádění nových síťových technologií je totiž stále velmi pomalé a jejich zavádění je pořád těžší a těžší. Aktivní uzly mohou naproti tomu provádět mnoho různých programů – místo požadavku, že každý směrovač

musí na každém paketu provést stejnou operaci, aktivní sítě umožní provozovat na každém uzlu stejný výpočetní model, například virtuální instrukční sadu. Tím zvyšují úroveň abstrakce, na které jsou operace realizovány, a aplikacím umožňují zvolit si zpracování přeposílaných zpráv tak, jak si samy přejí.

Kvalita služeb

Aktivními programy, které mohou ovlivnit průchod dat směrovači, lze dosáhnout silnějších požadavků na kvalitu síťových služeb – například je možno požadovat, aby data prošly sítí s jistou kvalitou služby, je-li to možné. A pokud není, pak s nejvyšší takovou, jakou síť dovoluje. Tento přístup je vhodný zejména pro přenášení videosignálu – směrovač může selektivně posílat pouze obrazy celé scény a vynechávat změny, čímž lze dosáhnout podstatně lepších výsledků, než kdyby pakety zahazoval zcela náhodně.

Konkrétními aplikacemi aktivních sítí mohou být například tyto:

1. **Firewally**
   Firewally implementují filtry zjišťující, které příchozí pakety budou dále odeslány, a které budou blokovány. Jsou podobné směrovačům, oproti klasickému směrování paketů však navíc implementují některé aplikačně a uživatelsky specifické funkce. Velmi obtížná situace nastává v případě nutnosti firewall aktualizovat. V aktivních sítích může být tento proces aktualizace automatizován, jestliže aplikacím umožníme nahrávat si ze schválených zdrojů potřebné moduly. Díky tomu se zvyšuje i možnost obrany proti specifickým útokům, například útoku DoS (Denial of Service). V případě tohoto útoku posílá útočník velké množství dat na vybraný stroj za účelem jeho zahlcení, takže stroj přestane zpracovávat regulérní požadavky. V pasivních sítích neexistuje žádná rychlá cesta, jak tento útok zastavit. Oproti tomu v aktivních sítích lze tento útok zastavit velmi snadno – po odhalení útoku lze zpětně po cestě, po které přicházejí útočící pakety, firewallům vyslat požadavek k blokování daného útočníka. Jakmile se tento požadavek dostane až na první směrovač v útočníkově cestě, je tento útok zcela zastaven a útočník je „odstřižen“ od sítě.

2. **Webové proxy servery**
   Tyto servery poskytují uživatelsky transparentní službu umožňující cacheování a poskytování webových stránek. Aby však mohly efektivně plnit svůj účel, musí být cacheovací uzly umístěny co nejblíže

ke koncovým uživatelským stanicím. V aktivních sítích lze tento systém nastavit tak, aby se tyto uzly zcela automaticky umisťovaly na strategických místech uvnitř sítě (a tím byly koncovým uživatelům co nejblíže).

3. **Nomadic router**
   L. Kleinrock navrhl princip „kočovných směrovačů“, které jsou vloženy mezi koncovou stanici a síť. Tento směrovač sleduje stav připojení koncové stanice a přizpůsobuje své chování měnícím se vlastnostem sítě (například připojení pomocí modemové linky v hotelu či připojení pomocí LAN v kanceláři). Tento směrovač pak může rozhodnout o větším cacheování souborů či kompresi odesílaného toku dat v případě pomalého připojení, či o použití doplňkového zabezpečení (například šifrování odesílaného toku dat) v případě připojení mimo kancelář.

4. **Transportní brány**
   Transportní brány jsou speciální síťové uzly, které jsou umístěny na strategických místech sítě, sloužící ke spojení sítí s velmi rozdílnou propustností či spolehlivostí (například na spojení drátových a bezdrátových sítí).

5. **Aplikační služby**
   Zde se jedná o aplikačně specifické brány podporující služby typu překódování obrazů mezi videokonferencujícími uživateli s rozdílnou kvalitou připojení a jiné.

Jak je vidět, užitečných aplikací aktivních sítí lze najít nepřeberné množství. V mnoha případech jsou tyto služby implementovány na uzlech, které přebírají vlastnosti směrovačů (jako například firewally), avšak navíc provádějí aplikačně-specifické operace. Cílem aktivních sítí je zaměnit aktuálně četné ad hoc přístupy k síťově závislým operacím za obecně použitelnou schopnost, která uživatelům umožní sítě programovat.

Kapitola 4

Vyvíjený aktivní směrovač

Architektura aktivního směrovače vyvíjeného na Masarykově univerzitě využívá metody aktivních uzlů s možností do směrovače zavádět uživatelské programy (a řadí se tak do kategorie aktivních sítí se zaváděnými funkcemi). Jeho podrobný popis lze nalézt v [10].

Klíčovou roli hraje v tomto modelu struktura daného směrovače. Jedná se o síťový prvek, který je schopen přijímat uživatelské programy, a poté je provádět. Zpracování uživatelských programů probíhá ve dvou fázích. Úkolem první fáze je řídit ustavení spojení; poté je jejím sekundárním úkolem toto ustavené spojení dále spravovat. V aktivním směrovači má tato fáze řídící funkci a zahrnuje všechny uživatelské funkce (iniciální či nahrané) na směrovačích od odesílatele až k příjemci dat. Druhá fáze zpracovávaní uživatelského kódu, vyvolaná fází první, pak tvoří ústřední součást samotného zpracování datových paketů.

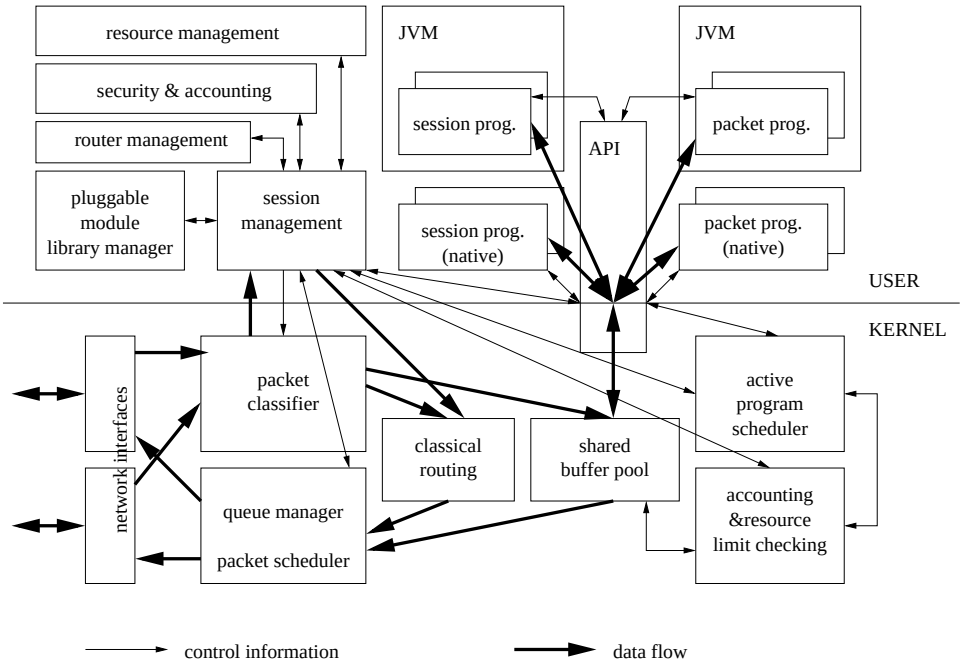
Aktivní spojení (neboli relace, anglicky *session*) se skládá ze stavových informací uložených v každém aktivním směrovači po cestě mezi oběma komunikujícími koncovými uzly. Nejdůležitější částí každé aktivní relace je množina běžících instancí nahraných uživatelských programů. Tyto programy pak spravují stavové informace dané relace, například nastavování vstupních paketových filtrů, konfiguraci vstupních a výstupních front, aj. Samotná relace může být řízena (tj. vytvářena, rozšiřována na nové aktivní uzly, částečně nebo zcela zrušena, atd.) buď v závislosti na požadavcích uživatele, nebo také zevnitř – pomocí programů dané relace. Její správa může probíhat buď vzdáleně s využitím protokolu pro správu relací (*session management protocol*), nebo lokálně na směrovači s použitím metod určených pro správu relací, které software směrovače obsahuje.

Typicky uživatelská aplikace komunikuje (s využitím protokolu pro správu relací) s nejbližším aktivním směrovačem na cestě k cíli, na který mají být data doručena. Tímto je s požadovanými parametry nastartována první relace (a k ní náležející programy). Tato relace pak kromě jiného určí následující aktivní uzel, jenž je potřebný pro výslednou komunikaci mezi oběma koncovými uzly, a nastartuje novou relaci (včetně všech nutných

programů k ní náležejících). Tento postup se opakuje do té doby, dokud není ustavena celá cesta až k cíli. Jakmile je celé spojení ustaveno, lze začít vysílat data, která budou na jednotlivých síťových uzlech patřičně zpracována. V případě, že v průběhu sestavování spojení dojde k nějaké chybě, lze použít zotavovací techniky pro nalezení jiné cesty v síti.

Model aktivního směrovače rozšiřuje stávající model „hloupých“ směrovačů o nové protokoly, které lze rozdělit do dvou kategorií podle fází směrovače, ve kterých se uplatňují. Jedná se o protokoly zajišťující síťové spojení a řídící nahrávání potřebných programů, a protokoly řídící zpracování takto nahraných programů. Důležitou roli hraje také bezpečnost, která je založena na protokolech autentizačních a autorizačních.

Jedním z hlavních cílů návrhu tohoto směrovače je vytvořit obecný síťový protokol, který by na aplikační a prezentační vrstvě umožnil použití existujících síťových protokolů (či jejich drobných modifikací). A právě takový protokol se snaží navrhnout a implementovat tato práce.



Obrázek 4.1: Architektura vyvíjeného aktivního směrovače (převzato z [10])

Kapitola 5

Transportní vrstva, transportní protokoly

Mezinárodní organizací ISO byl v roce 1979 představen jednotný standard pro vzájemné propojování systémů různých typů a koncepcí nazývaný ISO/OSI model. Navržený model má sedm vrstev, které tvoří hierarchii začínající aplikacemi na vrcholu a končící fyzickými spojeními vespod. Přestože se tento model v praxi neuplatnil, nadále slouží jako metoda popisu komunikačních systémů. Znalost této architektury je tak základním předpokladem pro pochopení funkce počítačových sítí, přenosu dat a návazných technologií. Další podrobnosti například v [1].

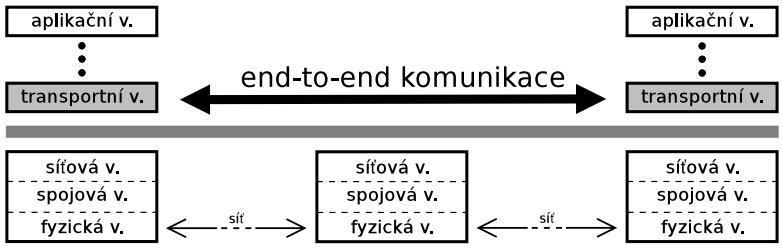
Obrázek 5.1: ISO/OSI model

Každá vrstva tohoto modelu má přesně definované funkce, které musí splňovat. Z pohledu této práce je nejzajímavější transportní vrstva, do které se navrhovaný protokol řadí.

Úkolem transportní vrstvy je zajistit spolehlivost a kvalitu přenosu v rámci end-to-end komunikace tak, jak ji požadují vyšší vrstvy. Poskytuje například funkce pro vytvoření příslušných spojení, řídí tok dat nebo rozkládá data na menší části, tzv. pakety, do kterých rozděluje přenášená data. Při příjmu pak data z paketu zase vyjímá a skládá je do původního tvaru. Do-

káže tak zajistit přenos libovolně velkých zpráv, přestože jednotlivé pakety mají omezenou velikost. V pasivních sítích není přítomna v přepojovacích uzlech (např. směrovačích), ale je implementována až v koncových stanicích (viz obrázek 5.2).

Principiálně nabízí tato vrstva dva typy služeb: spojově orientované a nespojové. Spojově orientované služby zajišťují spolehlivý přenos navázáním virtuálního spojení, výměnou informací o průběhu přenosu (potvrzováním příjmu paketů) a ukončením spojení. Na základě potvrzování je vysílající uzel schopen ztracené nebo opožděné pakety zopakovat. Konkrétním představitelem tohoto typu protokolů je TCP (*Transmission Control Protocol*). Nespojové služby slouží k jednoduchému odeslání dat. Neexistuje zde mechanismus kontroly spolehlivosti, kterou je potřeba zajistit na úrovni vyšších vrstev. Typickým představitelem tohoto typu protokolů je UDP (*User Datagram Protocol*).



Obrázek 5.2: Schéma end-to-end komunikace na pasivních sítích

5.1 Protokol UDP

Jak již bylo uvedeno, protokol UDP (*User Datagram Protocol*) je typickým představitelem nespojových transportních protokolů. Nepřináší žádné vlastnosti spolehlivosti přenosu, řízení toku nebo funkce opravy chyb; slouží především jako jednoduché rozhraní mezi protokoly vyšší vrstvy a IP protokolem (například pro protokol RTP (*Real-time Transport Protocol*)).

Jeho hlavní přednost spočívá v jeho jednoduchosti. Hlavička protokolu UDP obsahuje menší množství informací než hlavička TCP, takže má tento protokol menší režii. Díky tomu, že nedefinuje žádné mechanismy pro dorozumívání obou komunikujících stran, je přenos také rychlejší než v případě TCP.

Jako transportní protokol vyvíjeného aktivního směrovače jej však nelze použít především díky nespolehlivému přenosu dat, který je jedním z hlav-

ních požadavků kladených na transportní vrstvu tohoto směrovače (viz následující kapitola).

Podrobnosti k protokolu UDP lze nalézt v [16].

5.2 Protokol RUDP

Protokol RUDP (*Reliable User Datagram Protocol*) je jednoduchý transportní protokol navržený pro spolehlivý přenos telefonních signálů v IP sítích. Jedná se o sadu rozšiřujících mechanismů původního UDP protokolu pro zajištění kvality přenosu (například řízení toku dat, retransmise a podobně). Pro každé virtuální spojení poskytuje spolehlivý přenos dat se zachováním pořadí (až do maximálního počtu retransmisí), což tento protokol činí použitelným pro aplikace vyžadující po síti určitou kvalitu služby. Další informace viz [4].

Tento protokol je svou architekturou nejvíce podobný potřebnému transportnímu protokolu, avšak stále nedostačuje zcela. Protokol RUDP například poskytuje spolehlivý přenos dat se zachováním pořadí; transportní protokol vyvíjeného aktivního směrovače by měl pouze zajistit spolehlivý přenos dat – aplikaci by měl zajistit jen předání informací o uspořádání datového toku (to znamená, kterou část dat právě předává).

5.3 Protokol TCP

Protokol TCP (*Transmission Control Protocol*) poskytuje spolehlivý přenos dat včetně zachování pořadí odeslaných datagramů na straně příjemce. Podobně jako RUDP také podporuje mechanismus virtuálních spojení. Spolu se síťovým protokolem IP tvoří v současné době základní mechanismus pro spolehlivou komunikaci na síti. Podrobnější informace viz [12].

Stejně jako v případě RUDP není ani TCP vhodný pro použití jako komunikační protokol ve vyvíjeném směrovači především díky odlišnému přístupu k zachování pořadí přijatých datagramů (více viz RUDP).

Kapitola 6

Protokol ARTP

6.1 Terminologie

V průběhu této kapitoly, zejména pak při popisu protokolu, se vyskytují některé pojmy s následujícím významem:

klient – strana zahajující (inicializující) spojení. Ve zjednodušeném případě jednosměrné komunikace se jedná o stranu vysílající data.

server – strana přijímající spojení, při jednosměrné komunikaci vystupuje v roli příjemce dat. Každý server může být zároveň klientem jinému serveru (jedná se o mezilehlý uzel, nikoli koncový).

relace (*session*) – spojení mezi klientem a serverem umožňující přenos dat. Relace může být:

- **neustavená:** slouží pro přenos požadavků o ustavení nových spojení a reakcí na ně.
- **ustavená:** slouží pro přenos vlastních dat. Obě komunikující strany mají vytvořeny potřebné struktury a uloženy všechny informace nutné k zajištění kvality přenosu.

datagram – aplikací předávaná jednotka dat určená k přenesení protokolem. Velikost datagramu není protokolem omezena (je omezena konfigurací komunikujících počítačů).

paket – jednotka dat přenášená po síti. Má přesně danou strukturu jednotlivých polí (viz dále) a jeho velikost je omezena architekturou sítě. V protokolu ARTP se vyskytují tyto typy paketů:

- **datové:** přenášejí aplikační data nebo jejich části (fragmenty).
- **řídící:** jsou určeny k přenosu řídících informací, které slouží pro domluvu a správu přenosu mezi klientskou a serverovou aplikací.

- **potvrzovací pakety:** přenášejí identifikační čísla paketů, jejichž přijetí je potřeba odesílateli potvrdit.

fragment – část datagramu přenášená v jednom paketu. Každý příliš velký datagram je na straně odesílatele rozdělen (fragmentován) do fragmentů, aby mohl být v paketech přenesen k příjemci.

timestamp – časová značka uložená v hlavičce paketu. Vyjadřuje čas odeslání paketu podle jeho odesílatele (časy odesílatele a příjemce se v obecném případě mohou lišit – viz dále).

buffer – zásobník sloužící k ukládání dat na komunikujících uzlech. Buffery mohou být:

- **odesílací:** jsou určeny pro ukládání odesílaných dat.
- **přijímací:** jsou určeny pro ukládání přijatých dat.
- **potvrzovací:** jsou určeny pro ukládání identifikačních čísel příchozích paketů, které je nutno potvrdit.

6.2 Požadavky na navrhovaný protokol

Společným úkolem všech transportních protokolů, ke kterým se navrhovaný protokol ARTP (*Active Router Transport Protocol*) řadí, je přenášet data od odesílatele k jejich příjemci. Vlastnosti tohoto přenosu jsou však u každého protokolu jiné – záleží především na prostředí, v jakém má být daný protokol použit.

Protokol ARTP je primárně určen pro použití ve vyvíjeném aktivním směrovači, a proto se jeho vlastnosti odvíjí od jeho potřeb. Všechny zásadní požadavky lze shrnout do následujících bodů:

1. **Základní přenos dat**
   Navržený komunikační protokol musí být mezi dvěma komunikujícími uzly schopen v libovolném směru přenášet souvislé bloky dat. Přenášená data mohou být dvou druhů – řídící data, jejichž úkolem je přenos spravovat, a aplikační data, což jsou vlastní přenášená data. U aplikačních dat by měl protokol pamatovat na možnost jejich šifrování a podepisování.

2. **Spolehlivost přenosu**
   Protokol musí zaručit spolehlivý přenos dat (všechna odesílacím uzlem vyslaná data musí být přijímajícím uzlem řádně přijata). Musí se

vypořádat se ztrátami, zpožděními či duplicitami dat a všechny tyto defekty sítě řádně opravit. Přijaté bloky dat nemusí být přijímající aplikaci předány v pořadí, v jakém byly odeslány (tj. není zapotřebí zachovávat pořadí přijatých datagramů).

3. **Řízení toku dat**
   Vysílající strana protokolu je povinna sledovat a řídit množství do sítě odesílaných dat (*congestion control*). V případě, že komunikace probíhá bez problémů, může toto množství zvyšovat. Jakmile se začne přijímající strana nebo síť odesílanými daty zahlcovat, musí protokol podniknout nutné kroky k odstranění tohoto stavu.

4. **Multiplexing**
   Protokol by měl umožnit vytvoření více spojení mezi dvěma komunikujícími stranami určenými svými IP adresami a porty. Tato spojení musí být vzájemně rozlišitelná a jednoznačně identifikovatelná.

6.3 Filozofie protokolu

Protokol ARTP je spojově orientovaný transportní protokol umožňující spolehlivý přenos dat bez nutnosti zachovávat jejich pořadí. Klientská aplikace, využívající protokol ARTP pro přenos dat na serverovou aplikaci, je před vlastním přenosem nucena s přijímající stranou vytvořit spojení (v protokolu je označováno jako relace), které je po ukončení přenosu potřeba uzavřít.

Vzhledem k požadavku multiplexity protokolu (možnost více spojení mezi dvěma komunikujícími stranami) poskytuje ARTP identifikační číslo relace – každé vytvořené spojení je tak jednoznačně určeno IP adresou a portem obou komunikujících stran a svým identifikačním číslem. Tento způsob návrhu umožňuje vytvářet více spojení mezi dvěma aplikacemi – tato spojení jsou pak vzájemně odlišitelná přiděleným identifikátorem. Vytvoření relace je klientem iniciováno odesláním požadavku na server, který tento požadavek vyhodnotí a klientovi odešle své rozhodnutí. V případě kladné odpovědi se po oboustranném vytvoření potřebných struktur pro zajištění správného chodu přenosu stává relace ustavenou.

Ustavená relace může být využita pro obousměrný přenos dat, která mohou být v protokolu ARTP dvou druhů – řídící a aplikační. Řídící data jsou určena k řízení chování komunikujících aplikací a jejich dorozumívání se. Každá řídící informace může obsahovat svůj identifikátor, typ (zda se jedná o požadavek nebo o odpověď) a svou hodnotu. Aplikační data jsou

vlastní přenášená data; jsou členěna do datagramů a protokolu jsou předávána ve dvou částech (protokol umožňuje rozdělení přenášených dat na vlastní data a jejich podpisovou informaci). Každý předávaný datový datagram je označen svým identifikátorem (datovým sekvenčním číslem), který si určuje samotná vysílající aplikace. Velikost přenášených datagramů není protokolem omezena.

Fyzický přenos probíhá formou paketů, do kterých jsou přenášená data vkládána. Pakety jsou složeny z hlavičky, která obsahuje nutné informace o přenášeném paketu, a datové části, v níž jsou zakódována přenášená data. Každý paket může navíc obsahovat řídící informace protokolu, které lze využít pro komunikaci vysílající a přijímající části protokolu (například řízení maximální velikosti přenášených paketů, vzájemné prioritizaci ustavených spojení, atd.) a nejsou předávány aplikacím. Konkrétní přenášené informace si může každá implementace protokolu ARTP stanovit zcela libovolně.

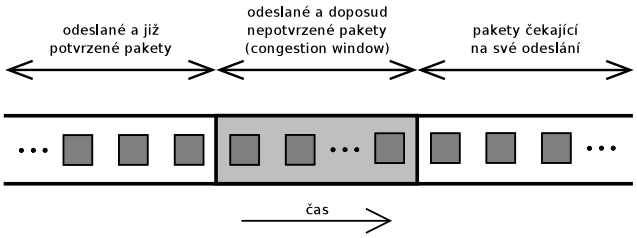
Hlavička ARTP paketu dále obsahuje informaci o času jeho odeslání (*timestamp*) a době, po jejímž uplynutí přestává být paket platný (doba expirace). Tyto informace jsou nezbytné pro detekci zpožděných paketů[1].

Spolehlivost a důvěryhodnost přenosu je zajištěna mechanismem potvrzování přijatých paketů a detekcí jejich duplicit (vícenásobného příjmu stejných paketů). Každý paket má ve své hlavičce uvedeno identifikační (sekvenční) číslo, které jej jednoznačně identifikuje. Přijímající strana protokolu odesílateli periodicky zasílá seznam nově přijatých sekvenčních čísel (těch, která byla přijata od doby odeslání posledního potvrzení), čímž mu potvrzuje jejich příjem. Jakmile je nějaký paket vyhodnocen jako ztracený (po definovanou dobu na něj nepřišlo potvrzení), je příjemci znovu odeslán (tzv. retransmise).

Protokol ARTP řídí množství do sítě vysílaných dat, aby nedocházelo k zahlcení příjemce nebo sítě a tím velké degradaci rychlosti a kvality přenosu. Pro tento účel je zvolen mechanismus okna zahlcení (*congestion window*), které vyjadřuje maximální množství dat, které lze v daném časovém okamžiku do sítě odeslat (mechanismus okna zahlcení je znázorněn na obrázku 6.1). Velikost okna se mění v závislosti na stavu přenosu – pokud probíhá přenos bez problémů (přicházejí potvrzení na všechny odeslané pakety), velikost okna se zvětšuje, poněvadž se předpokládá, že ještě nebyla naplněna dostupná kapacita linky. Zahlcení je detekováno zvýšenými

[1] Identifikační čísla paketů mají sice velký, ale konečný rozsah. V případě, že by pakety neobsahovaly informaci o čase své expirace, mohl by být za jistých okolností příliš zpožděný paket se starým identifikačním číslem pokládán za aktuální (kdyby přišel v okamžiku příjmu paketů s identifikačními čísly „nového kola“, která jsou blízká inkriminovanému, nebyl by „odchycen“ ani detekcí duplicit)

ztrátami paketů, na které vysílající strana reaguje snížením velikosti okna a tím omezením množství vysílaných dat.



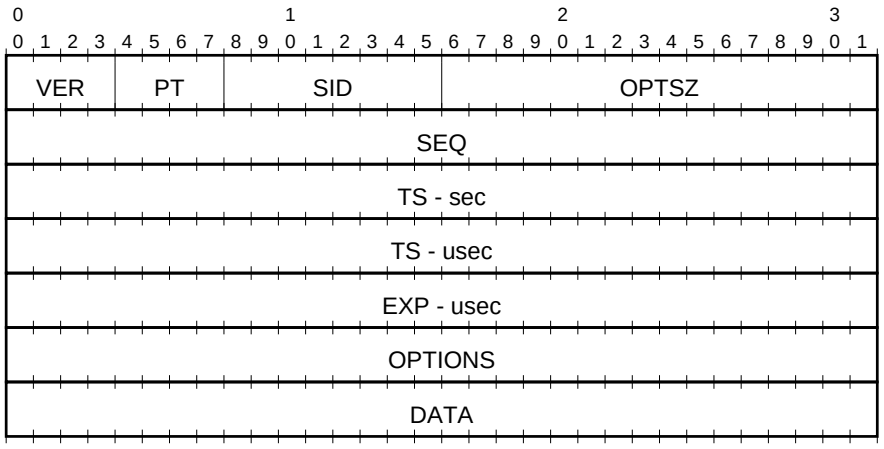
Obrázek 6.1: Mechanismus okna zahlcení (*congestion window*)

6.4 Funkční specifikace

6.4.1 Formát hlavičky paketu

Každý paket ARTP protokolu musí začínat alespoň 20B hlavičkou, po které už mohou následovat přenášená data.

Všechny více-bajtové informace jsou v ARTP paketu uloženy v síťovém formátu (*network byte order*, v kontextu mikroprocesorových architektur též známý jako *big endian*).



Obrázek 6.2: Hlavička ARTP protokolu

Popis jednotlivých polí:

VER (*Version*): 4 bity
Verze ARTP protokolu (aktuálně 1).

PT (*Packet type*): 4 bity
Typ dat uložených v paketu. Možné typy jsou:

- datový paket
- řídící paket
- potvrzující paket

SID (*Session identifier*): 8 bitů
Identifikační číslo relace.

OPTSZ (*Option field size*): 8 bitů
Velikost pole OPTIONS (v bajtech).

SEQ (*Sequence number*): 32 bitů
Sekvenční číslo paketu.

TS – sec (*Timestamp – seconds*): 32 bitů
Čas odeslání paketu – počet sekund od 1.1.1970.

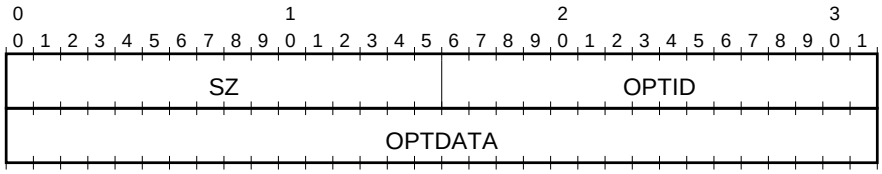
TS – usec (*Timestamp – microseconds*): 32 bitů
Čas odeslání paketu – počet mikrosekund.

EXP – usec (*Expiration time – microseconds*): 32 bitů
Doba platnosti paketu od času jeho odeslání (v mikrosekundách).

OPTIONS: proměnná velikost
Řídící informace protokolu zasílané v paketu (viz dále).

DATA: proměnná velikost
Data zasílaná v paketu (viz dále).

Pole OPTIONS obsahuje 0 nebo více struktur zobrazených na následujícím obrázku.



Obrázek 6.3: Struktura pole OPTIONS

Popis jednotlivých polí:

SZ (*Option total size*): 16 bitů
Celková velikost dané řídící informace.

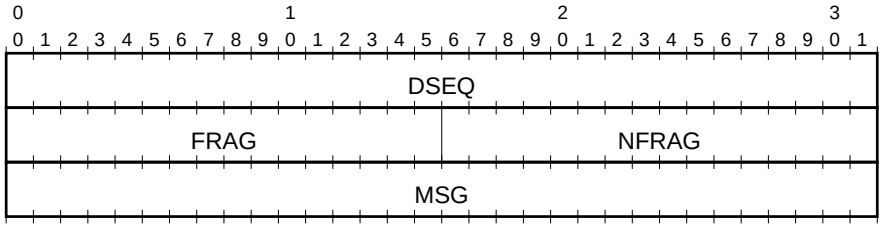
OPTID (*Option identifier*): 16 bitů
Identifikátor zasílané řídící informace.

OPTDATA (*Option data*): proměnná velikost
Data zasílané řídící informace.

Pole DATA slouží k uložení zasílaných dat do paketu. Jeho struktura je závislá na typu paketu (pole PT hlavičky ARTP paketu).

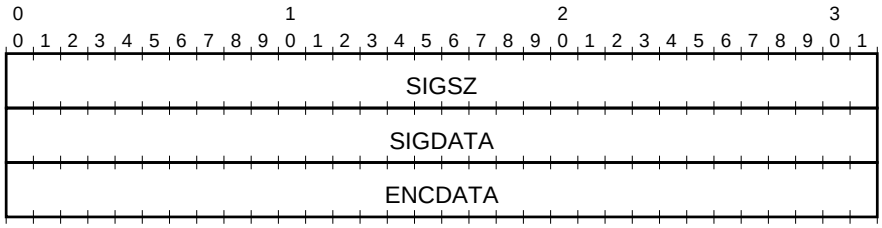
Aplikační data

Pole DATA obsahuje právě jednu strukturu zobrazenou na obrázku 6.4.



Obrázek 6.4: Struktura pole DATA

Pole MSG obsahuje právě jednu strukturu zobrazenou na obrázku 6.5. Právě tato informace je v případě velké velikosti datagramu (tj. pokud je větší než maximální velikost datové části paketu) fragmentována.



Obrázek 6.5: Struktura pole MSG

Popis jednotlivých polí:

DSEQ (*Data sequence number*): 32 bitů
Sekvenční číslo udávající pořadí datového datagramu.

FRAG (*Fragment identifier*): 16 bitů
Číslo zasílaného fragmentu datového datagramu.

NFRAG (*Fragment count*): 16 bitů
Celkový počet fragmentů datového datagramu.

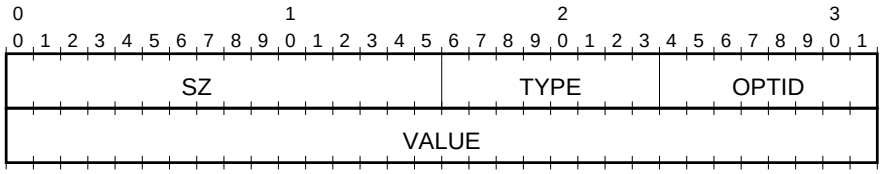
SIGSZ (*Signature size*): 32 bitů
Velikost položky SIGDATA (v bajtech).

SIGDATA (*Data signature*): proměnná velikost
Podpisová informace.

ENCDATA (*Encrypted data*): proměnná velikost
Šifrovaná data.

Řídící informace relace

Pole DATA obsahuje alespoň jednu strukturu zobrazenou na obrázku 6.6.



Obrázek 6.6: Struktura pole CTRL

Popis jednotlivých polí:

SZ (*Control option size*): 16 bitů
Celková velikost dané řídící informace.

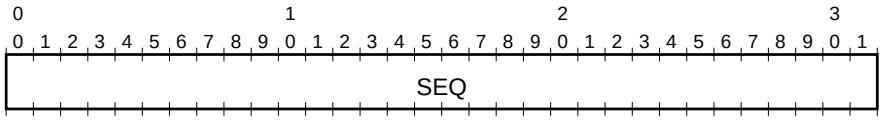
TYPE (*Control option type*): 8 bitů
Typ zasílané řídící informace. Identifikuje, zda se jedná o požadavek nebo odpověď.

OPTID (*Control option identifier*): 8 bitů
Identifikátor vložené řídící informace.

VALUE (*Control option value*): proměnná velikost
Hodnota obsažené řídící informace.

Potvrzovací pakety

Pole DATA obsahuje alespoň jednu strukturu zobrazenou na obrázku 6.7.



Obrázek 6.7: Struktura pole ACK

Popis jednotlivých polí:

SEQ (*Sequence number*): 32 bitů
Sekvenční číslo potvrzovaného paketu.

6.4.2 Časové značky (*timestamps*)

Jak již bylo uvedeno, do hlaviček všech odesílaných paketů jsou ukládány časy jejich odeslání, které spolu s dobou jejich platnosti slouží k detekci zpožděných paketů.

V obecném případě se čas na straně odesílatele může významně lišit od času na straně příjemce, proto nelze tento mechanismus zajistit intuitivní cestou – porovnáním času přijetí paketu (vyjádřeným v čase přijímající strany) s časem odeslání uloženým v paketu (vyjádřeným v čase odesílatele) sečteným s dobou platnosti paketu.

Časové značky protokolu ARTP

V protokolu ARTP si z důvodu možné nesynchronnosti časů obou komunikujících uzlů vysílající strana udržuje a pravidelně aktualizuje předpokládaný rozdíl mezi svým časem a časem druhé strany (v dalším textu bude tento rozdíl označován jako `ts_delta`). Z uvedeného vyplývá, že v případě obousměrné komunikace je tento čas udržován na obou stranách.

Mechanismus vkládání časových značek do hlaviček odesílaných paketů protokolu ARTP je následující:

- pokud není znám rozdíl lokálních časů obou komunikujících stran (tj. `ts_delta`), je do všech odesílaných paketů vkládán aktuální čas odesílatele. Zároveň je doba jejich platnosti (pole EXP v hlavičce ARTP paketu) nastavena na `0`.
- v případě, že je tento rozdíl znám, je do hlaviček paketů vkládána doba platnosti podle požadavků aplikace. Časová značka je pak vkládána podle typu paketu:
  - do datových a řídících paketů je vkládán předpokládaný čas příjemce (tj. aktuální čas odesílatele upravený o hodnotu `ts_delta`).
  - do potvrzovacích paketů je vkládán aktuální čas odesílatele.

Mechanismus zjišťování platnosti paketů na straně příjemce je následující:

- pokud je doba platnosti příchozího paketu nastavena na `0`, je paket vždy prohlášen za platný.
- pokud je doba jeho platnosti nenulová, je další postup závislý na jeho typu:
  - u datových a řídících paketů je součet timestampu uloženého v hlavičce paketu[2] a doby jeho platnosti porovnán s aktuálním časem přijímající strany. Podle výsledku porovnání je určena jeho platnost.
  - přijímané potvrzovací pakety obsahují ve své hlavičce čas svého odeslání vyjádřený časem odesílatele. Tento čas je potřeba po příjmu upravit o aktuální hodnotu `ts_delta`[3], čímž je přepočten

[2] Timestamp je uložen v předpokládaném čase přijímající strany, proto jej není potřeba upravovat podle `ts_delta`.

[3] Tohle neplatí pro první přijatý potvrzovací paket – ten je vždy prohlášen za platný, aby z něj mohly být zjištěny informace o rozdílu času obou komunikujících stran.

na čas přijímající strany. Součet takto vypočteného času a doby platnosti paketu je poté porovnán s aktuálním časem přijímající strany; na základě výsledku tohoto porovnání je určena platnost paketu.

Rozdíl časů mezi odesílací a přijímající stranou protokolu (tj. `ts_delta`) je odhadován s využitím potvrzovacích paketů informujících o doručení nepřeposílaných paketů. Lze jej vyjádřit vzorcem

$$\texttt{ts\_delta} = \mathrm{CT} - \left(\mathrm{ST} + \frac{\mathrm{RTT}}{2}\right)$$

kde

- CT ... čas přijetí daného potvrzovacího paketu,
- ST ... čas odeslání získaný z hlavičky potvrzovacího paketu,
- RTT ... čas uplynulý mezi odesláním potvrzovaného paketu a příchodem potvrzení o jeho přijetí[4].

Z uvedeného algoritmu vyplývá, že na začátku přenosu jsou všechny odesílané pakety nastaveny jako neexpirovatelné (doba jejich platnosti je neomezená). Při příchodu potvrzení o jejich přijetí je z potvrzovacího paketu vypočten předpokládaný rozdíl časů na odesílací a přijímající straně, který je poté použit pro vkládání timestampu do odesílaných datových a řídících paketů (předpokládaný čas přijímající strany). Odhadovaný časový rozdíl je na straně příjemce periodicky počítán s využitím potvrzovacích paketů (jejich čas odeslání je vyjádřen v čase odesílatele).

Uvedený algoritmus je aplikovatelný jak na jednosměrnou, tak na obousměrnou komunikaci.

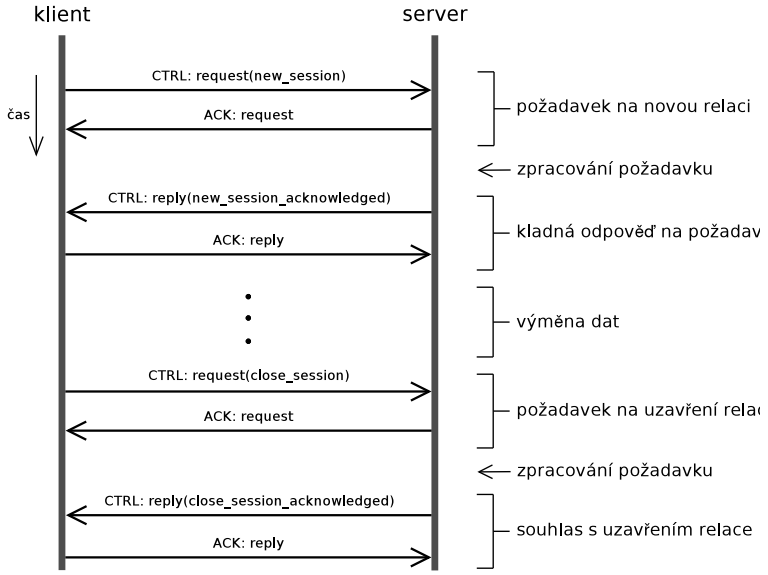
6.4.3 Ustavení spojení (relace)

Ustavení relace probíhá v protokolu ARTP ve dvou fázích. V první fázi je potřeba novou relaci s druhou stranou vyjednat. Na základě výsledku první fáze se uskuteční fáze druhá, která novou relaci fyzicky založí. Po úspěšném ukončení obou těchto fází je relace prohlášena za ustavenou.

[4] Místo RTT času je zde možno použít jeho zjemnění na SRTT čas (*Smooth Round Trip Time*), který není tak náchylný k náhlým výkyvům sítě. Může být periodicky počítán například podle vzorce $\mathrm{SRTT} = (\alpha \cdot \mathrm{SRTT}) + ((1 - \alpha) \cdot \mathrm{RTT})$, kde $\alpha = 0{,}875$.

1. **Domluva o ustavení relace**
   Tato fáze není záležitostí samotného protokolu, ale aplikace, která jej využívá (protokol pouze poskytuje nástroje, pomocí kterých lze tuto domluvu uskutečnit). Klientská aplikace požádá protokol o volný identifikátor nově zakládané relace a v případě, že existuje, vyšle serveru řídící paket s požadavkem o ustavení nové relace. Serverová aplikace se po přijetí takového paketu rozhodne, zda tomuto požadavku vyhoví nebo jej zamítne, a toto své rozhodnutí klientské straně sdělí řídícím paketem. V případě kladné odpovědi tuto novou relaci klient i server fyzicky založí, aby mohla probíhat výměna dat.

   Typická komunikace mezi klientem a serverem při domlouvání nové relace je znázorněna na obrázku 6.8.



Obrázek 6.8: Domluva mezi klientem a serverem při zakládání a rušení relace

2. **Fyzické založení relace**
   Fyzickým založením relace se rozumí příprava obou komunikujících stran na přenos dat. Aby byla zaručena spolehlivost tohoto přenosu, zakládají si obě strany spojení (odtud spojově orientovaný protokol) – připravují buffery pro odesílání a příjem dat, inicializují další potřebné struktury a parametry spojení, atd. Právě díky těmto informacím, které

si každá komunikující strana pamatuje, je možno zajistit požadovanou kvalitu služby.

6.4.4 Ukončení spojení

Podobně jako při zakládání spojení probíhá i ukončování spojení ve dvou fázích. V první fázi je potřeba dát druhé straně, se kterou je spojení ustaveno, najevo, že má spojení zrušit[5]. Po této domluvě, která je opět záležitostí samotné aplikace, je možno relaci fyzicky zrušit, čímž dojde k odstranění všech informací, které rušené relaci patřily.

6.4.5 Datová komunikace

Výměna dat

Jakmile je spojení ustaveno, je možno začít přenášet data. Výměna dat mezi aplikací a protokolem ARTP probíhá formou bloků dat (datagramů). Vzhledem k tomu, že tyto datagramy mohou mít libovolnou velikost, nelze je celé přenášet po síti. Odesílací část protokolu je proto fragmentuje do menších částí (paketů), které jsou opatřeny příslušnou hlavičkou a poté po síti přenášeny k příjemci. Pokud příjemce přijme všechny fragmenty jistého datagramu, ve správném pořadí je složí a předá přijímající aplikaci. Pořadí, v jakém jsou kompletně přijaté datagramy předány přijímající aplikaci, protokol ARTP nezaručuje (zaručuje pouze, že všechny datagramy budou z přijatých fragmentů správně složeny).

Po správném přijetí každého platného paketu[6] přijímající strana protokolu ARTP odesílá potvrzující paket se sekvenčním číslem potvrzovaného paketu. Tato sekvenční čísla nemusí být posílána po každém přijatém paketu, ale v zájmu efektivity mohou být sdružována do větších celků. V tomto případě je však nutno nastavit interval odesílání potvrzení tak, aby na odesílací straně nedocházelo ke zbytečným retransmisím (viz dále).

Důležitou činností přijímající strany je také detekce vícenásobného přijetí stejných paketů, která je prováděna uchováváním nutné historie přijatých sekvenčních čísel. Pokud je detekována duplicita, je paket bez ukládání znovu potvrzen (původní potvrzení se mohlo zpozdit nebo dokonce ztratit).

[5] První fázi domluvy je možno vynechat a relaci zrušit okamžitě. V tomto případě by se komunikační partner o rozpadu spojení dozvěděl jinou cestou – více v části Retransmise.

[6] Neplatné (expirované) pakety jsou okamžitě zahazovány.

Retransmise

Aby bylo zaručeno přijetí každého odeslaného paketu, poskytuje protokol ARTP mechanismus retransmise. Každý odeslaný paket je na odesílací straně uchováván do té doby, než na něj přijde potvrzení o doručení zaslané přijímající stranou, po kterém je zahozen. Pokud toto potvrzení nepřijde do jistého časového okamžiku (*retransmission timeout*), je paket znovu odeslán a celý postup se opakuje do té doby, než na něj přijde potvrzení nebo vyprší časový limit na jeho doručení. Po vypršení tohoto limitu je spojení prohlášeno za rozpadlé.

Poznámka: Vzhledem ke kolísavosti rychlosti a kvality nynějších sítí je nutno čas retransmise dynamicky zjišťovat a přepočítávat. Jedním z možných způsobů je jeho počítání s využitím RTT času. Aby byl protokol robustní vůči krátkodobým výkyvům sítě, lze místo RTT času použít jeho zjemnění na SRTT čas (*Smooth Round Trip Time*). Více viz poznámky pod čarou v kapitole 6.4.2 (Časové značky protokolu ARTP) nebo dokumentace ke vzorové implementaci navrženého protokolu.

Zábrana proti zahlcení příjemce

Zábrana proti zahlcení přijímající strany je v protokolu ARTP zajišťována pomocí okna zahlcení. Každá relace si pamatuje maximální velikost dat, která mohou být do sítě odeslána, aniž se musí čekat na jejich potvrzení (okno zahlcení), a velikost aktuálně nepotvrzených dat. Další paket je pak odeslán pouze v případě, kdy je součet jeho velikosti s aktuální velikostí doposud nepotvrzených dat menší nebo roven aktuální velikosti okna.

Velikost okna se zvyšuje po každém přijatém potvrzení o doručení dříve odeslaného paketu. Zahlcení je detekováno zvýšenými ztrátami paketů, proto je při každé retransmisi velikost okna snižována. V případě, že je linka po definovanou dobu nevyužívána, je velikost okna dané relace snížena na iniciální hodnotu, čímž je zaručen pomalejší nárůst při opětovném zahájení přenosu (příjemce nebude najednou zahlcen velkým množstvím dat).

Správa velikosti okna má velký vliv na přenosový výkon protokolu. Pomalý nárůst při bezchybném přenosu či strmý pokles při ztrátách může tento výkon silně degradovat. Ověřené nastavení manipulace s oknem může být například toto:

- iniciální velikost okna je rovna trojnásobku maximální velikosti segmentu (*Maximum Segment Size*, MSS) vysílaného do sítě.

- Při příchodu potvrzení na dříve odeslaný paket je velikost okna zvýšena o $\frac{\mathrm{MSS}^2}{\mathrm{CWND_{old}}}$, kde $\mathrm{CWND_{old}}$ je původní velikost okna.
- Při ztrátě paketu a nutnosti jeho retransmise je nová velikost okna nastavena na

  $$\max(0{,}5 \cdot \mathrm{CWND_{old}}, \mathrm{CWND_{init}})$$

  kde $\mathrm{CWND_{old}}$ je původní velikost okna a $\mathrm{CWND_{init}}$ jeho iniciální velikost.

6.4.6 Rozhraní ARTP protokolu

V této části budou popsány základní uživatelské funkce navrženého protokolu. Každá implementace protokolu ARTP si svou funkcionalitu může specifikovat sama[7], a proto zde budou popsány jen ty základní, které musí každá z nich splňovat.

Zápis následujících funkcí je podobný zápisu funkcí a procedur ve většině programovacích jazyků vyšší úrovně. Bude však uváděn v pseudojazyce, poněvadž má pouze informativní charakter.

1. **Založení relace**
   Formát:

   ```
   CREATE(identifikátor relace, adresa příjemce
          port příjemce[, vlastnosti relace])
   ```

   Úkolem této funkce je fyzicky založit relaci tak, aby mohla probíhat komunikace s uvedeným příjemcem – vytvoří všechny buffery a další pomocné struktury, inicializuje hodnoty všech parametrů, atp.

   Pokud je nastaven parametr `vlastnosti`, může také nastavit požadované vlastnosti a chování přenosu.

2. **Zrušení relace**
   Formát:

   ```
   CLOSE(identifikátor relace, adresa příjemce,
         port příjemce)
   ```

[7] Při popisu bude přihlíženo k vytvořené vzorové implementaci (názvy funkcí se také mohou v jednotlivých implementacích lišit). Detailní popis uvedených funkcí je možno nalézt v dokumentaci k vzorové implementaci.

Tato funkce ruší dříve vytvořenou relaci – uvolňuje všechny informace, které byly pro vytvořenou relaci potřeba. Jelikož není při rušení relace potřeba žádná komunikace s druhou komunikující stranou (není potřeba výměna žádných informací s partnerem, oznámení o rušení relace a ukončení přenosu je ponecháno na aplikaci), je relace zrušena neprodleně.

3. **Odeslání datagramu**
   Formát:

   ```
   SEND(odesílaný datagram, identifikátor relace,
        adresa příjemce, port příjemce)
   ```

   Volání této funkce způsobí odeslání předávaného datagramu v rámci indikované relace. Pro odesílání vlastních dat musí být relace ustavena; odesílání řídících informací (např. požadavků na nová spojení) lze provádět i v rámci neustavených relací. Před vlastním odesláním je zjištěno, zda se spojení nerozpadlo nebo zda už nejsou plné odesílací buffery (v případě, že je jejich velikost omezena). Pokud nastane některá z těchto událostí, končí funkce chybou.

   Datagramy jsou vkládány do paketů (v nutných případech ve formě fragmentů), opatřeny hlavičkou protokolu ARTP a vloženy do odesílacího bufferu. Předpokládá se asynchronní prostředí, takže funkce končí po vložení všech vytvořených paketů do bufferu a nečeká na jejich fyzické odeslání.

4. **Příjem datagramu**
   Formát:

   ```
   RECEIVE(identifikátor relace, adresa příjemce,
           port příjemce, přijatý datagram)
   ```

   Funkce `RECEIVE` slouží k získávání přijatých datagramů z přijímacího bufferu indikované relace. V případě, že se datagram skládá z více fragmentů, je nejprve složen a poté předán volající aplikaci. Volání této funkce je neblokující, takže pokud je buffer prázdný, funkce končí s definovaným chybovým kódem a nečeká na přijetí nějakého datagramu.

Kapitola 7

Implementace a testy

Nedílnou součástí této práce je také vzorová implementace navrženého protokolu ARTP, jejíž zdrojové kódy uvolněné pod BSD licencí lze nalézt na přiloženém CD. Součástí tohoto CD je také dokumentace k vytvořené implementaci (pouze v angličtině) a program demonstrující použití tohoto protokolu v praxi.

7.1 Vzorová implementace

Protokol je naprogramován formou knihovny v programovacím jazyce C, který byl zvolen především z důvodu své rychlosti. Knihovna byla vytvořena a testována na operačním systému Linux (konkrétně distribuce Debian s jádrem 2.4.26 a Mandrake s jádrem 2.4.23). Protože je v maximální možné míře použito volání standardů POSIX a ISO C, je možno knihovnu s minimálními změnami provozovat i na jiných unixových operačních systémech (např. *BSD, IRIX, Solaris a dalších). Komentáře k funkcím jsou psány ve formátu pro `doxygen`, pomocí kterého je také vygenerována přiložená dokumentace.

Knihovna implementuje veškerou funkčnost navrženého protokolu. Splňuje klíčové požadavky, které na ni byly vzhledem k předpokladu, že se bude protokol ARTP neustále vyvíjet, kladeny – jednoduchost, strukturovanost a čitelnost zdrojových kódů (díky těmto vlastnostem jsou usnadněny její další případné úpravy). Z těchto důvodů byla v případech, kdy bylo nutno vybírat mezi čitelností a rychlostí kódu, zvolena právě čitelnost. Popis architektury této implementace je uveden v přílohách, detailní popis rozhraní a implementovaných funkcí lze nalézt v dokumentaci na přiloženém CD.

7.2 Použití knihovny

Pro korektní spuštění protokolu ARTP je nutno inicializovat všechny potřebné struktury protokolu a spustit potřebná vlákna. K tomu slouží funkce `artp_init`, které je možno předat cestu ke konfiguračnímu souboru protokolu. Poté již lze protokol využívat pro zakládání nových spojení (funkce `artp_prepare_connection`, odesílání dat `artp_send_dgram` a další. Podrobnější popis všech funkcí rozhraní protokolu ARTP lze nalézt v dokumentaci. Modul volající uvedené funkce musí použít hlavičkový soubor `artp/artp.h`, který zajistí vložení všech nutných funkčních prototypů.

Při sestavování výsledného programu je potřeba připojit knihovnu protokolu `libartp`.

7.3 Testy

Funkčnost vytvořené implementace protokolu ARTP byla testována třemi testy – testem rychlosti, spolehlivosti přenosu a testem stability. Testy byly prováděny podle zapojení znázorněném na obrázku 7.1, konfigurace použitých počítačů jsou následující:

1. **Síť o rychlosti 1 Gbit/s**
   DELL PowerEdge 1600 SC, 2 × Intel Xeon CPU 2.8 GHz, paměť 1024 MB, síťové karty Intel PRO/1000 32 bit/66 MHz, operační systém Linux Debian Woody 2.4.26.

2. **Síť o rychlosti 100 Mbit/s**
   Intel Pentium 4 2.0 GHz, paměť 512 MB, síťová karta Intel PRO/100 VE 32 bit/33 MHz, operační systém Linux Debian Woody 2.4.26.

Obrázek 7.1: Zapojení počítačů pro testy protokolu ARTP

Všechny testy byly prováděny na síti o rychlosti 1 Gbit/s, test rychlosti byl navíc prováděn i na síti o rychlosti 100 Mbit/s. Pro test rychlosti byl vytvořen speciální testovací program, ostatní dva testy byly prováděny s využitím programu demonstrujícího funkčnost vytvořené implementace, který je přiložen na CD. Testování bylo ve všech případech prováděno s pseudonáhodnými daty (byla získána ze souboru `/dev/urandom`).

Konfigurace protokolu ARTP použitá při testech byla pro všechny testy jednotná – velikosti všech bufferů byly omezeny na 10 MB (výjimku tvoří test korektnosti fragmentace, kdy byly buffery neomezené – viz dále) a maximální velikost jednoho do sítě odesílaného segmentu byla nastavena na 2 KB.

7.3.1 Test rychlosti

Tento test byl zaměřen na zjištění maximální přenosové rychlosti vytvořené implementace a jejím srovnání s existujícím transportním protokolem (jako srovnávací protokol byl vybrán protokol TCP). V obou měřených konfiguracích (tj. 100Mbit síť a 1Gbit síť) byla také s využitím protokolu UDP změřena maximální propustnost linky[1].

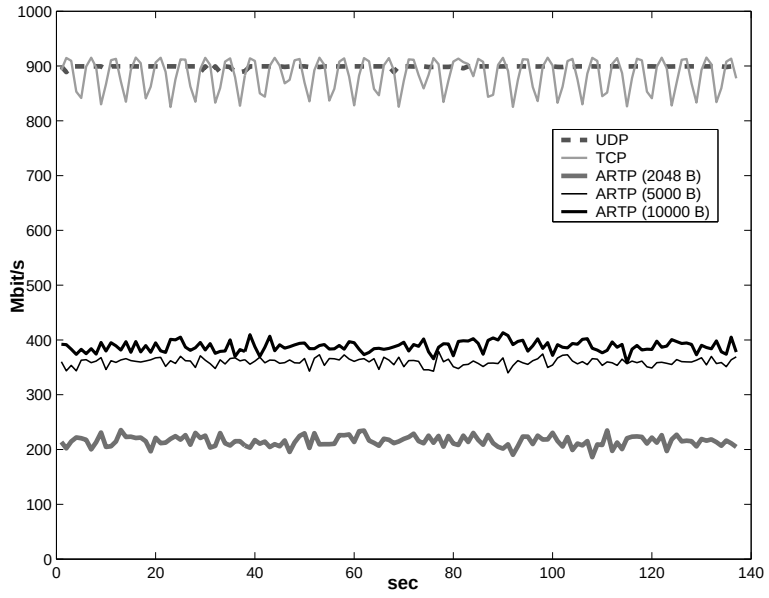
Oba protokoly (tj. ARTP a TCP) byly testovány na třech velikostech přenášených datagramů – 2000 B, 5000 B a 10000 B. V případě protokolu TCP je však v grafu zobrazen pouze naměřený průtok pro datagramy o velikosti 2000 B, poněvadž všechny tři testy dopadly téměř shodně a z důvodu přehlednosti nebyly do grafu zaneseny.

Dosažené výsledky testovaných protokolů na 1Gbit síti zobrazuje obrázek 7.2, na 100Mbit síti obrázek 7.3. V následující tabulce 7.1 jsou vyznačeny průměrné rychlosti testovaných protokolů spolu se směrodatnou odchylkou naměřených průměrných hodnot.

	1 Gbit/s (v Mbit)			100 Mbit/s (v Mbit)		
Datagram	**UDP**	**TCP**	**ARTP**	**UDP**	**TCP**	**ARTP**
2048 B	898,46 ± 2,49	882,94 ± 30,15	215,24 ± 9,45	91,75 ± 0,00	90,00 ± 0,00	84,01 ± 1,47
5000 B	–	881,63 ± 27,67	359,82 ± 7,18	–	89,53 ± 0,00	87,50 ± 1,43
10000 B	–	881,27 ± 32,42	387,98 ± 9,72	–	89,17 ± 0,00	87,80 ± 1,48

Tabulka 7.1: Průměrné rychlosti testovaných protokolů

[1] Ve všech měřeních se přenosová rychlost počítala pouze z přenesených dat – do jejich objemu nebyly zahrnuty velikosti hlaviček protokolů nižších vrstev (UDP, IP, atp.) ani hlavičky testovaných protokolů.



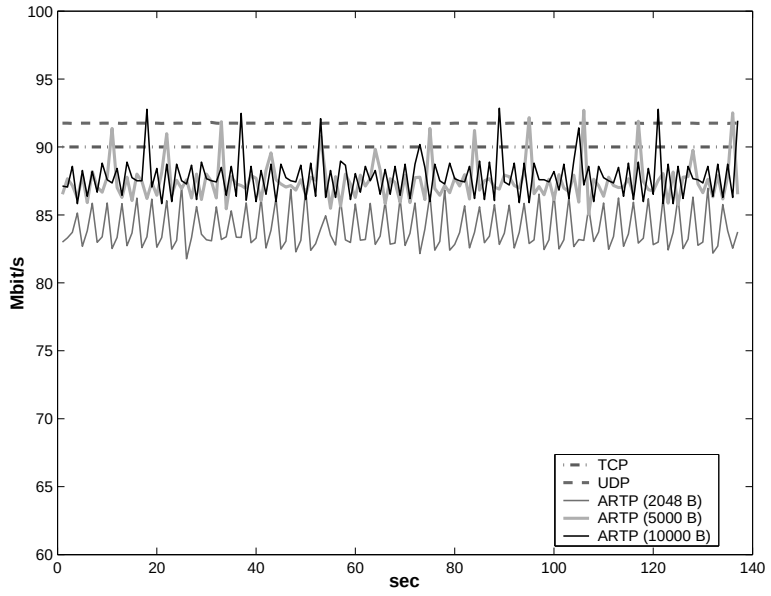
Obrázek 7.2: Srovnání rychlostí testovaných protokolů na 1Gbit síti

Z naměřených výsledků lze vyčíst, že vytvořená implementace protokolu ARTP je na 100Mbit síti konkurenceschopná, avšak na 1Gbit síti dosti zaostává. To může být způsobeno řadou faktorů – především tím, že vypracovaná implementace je zaměřena na přehlednost a strukturovanost kódu, nikoli na efektivitu.

Z výsledků dosažených na 1Gbit síti vyplývá ještě jedna skutečnost – rychlost protokolu ARTP se mírně zvyšuje s velikostí přenášeného datagramu. Tento jev je pravděpodobně způsoben menší manipulací s pamětí při alokaci a dealokaci paměťových bloků na přijímající straně protokolu (místo pěti sérií kroků nutných ke zpracování datagramu o velikosti 2000 B je v případě datagramu velikosti 10000 B provedena pouze jedna taková série).

7.3.2 Test spolehlivosti přenosu

Úkolem tohoto testu bylo ověření, zda vytvořená implementace přenáší data spolehlivě beze ztrát, duplicit či jiných možných defektů.



Obrázek 7.3: Srovnání rychlostí testovaných protokolů na 100Mbit síti

Test byl rozdělen na dvě části. První část ověřovala, zda protokol k příjemci skutečně přenese všechna data beze ztrát, druhá navíc testovala, zda protokol přenášená data správně fragmentuje.

1. **Bezeztrátový přenos**
   Pro účely tohoto testu byl vytvořen soubor s náhodnými daty, který byl v datagramech o velikosti 10000 B přenášen příjemci. Velikost vytvořeného souboru byla 10,15 GB. Protokol celý soubor řádně přenesl. Testování zdrojového a cílového souboru na shodný obsah dopadlo podle očekávání neúspěšně díky tomu, že vlastností protokolu ARTP je nezachovávání pořadí přijatých datagramů.

   Za zmínku také stojí, že se doba testu významně lišila podle toho, zda byla na přijímající straně data zapisována na disk či nikoli (a byla počítána pouze jejich velikost). Zatímco v druhém případě byl celý soubor přenesen za 3 minuty a 56 sekund (tj. průměrná rychlost 352,33 Mbit/s), v druhém případě se tento čas v důsledku čekání na diskové operace prodloužil na 20 minut a 7 sekund (tj. průměrná rychlost 69 Mbit/s).

2. **Korektnost fragmentace**
   Korektnost fragmentace byla opět zjišťována s využitím přenosu souboru. V tomto testu byl vytvořen soubor s náhodnými daty o velikosti 102,57 MB, který byl celý načten do řetězce a jako jeden datagram předán protokolu[2]. Ten z něj musel vytvořit fragmenty, přenést je příjemci a tam je opět složit do původního datagramu.

   I nyní protokol celý soubor řádně přenesl (velikosti zdrojového i cílového souboru byly totožné). Kromě samotného přenosu bylo potřeba přenesený soubor zapisovat na disk, proto byla doba přenosu dosti dlouhá – 4 minuty a 31 sekund (tj. průměrná rychlost 3 Mbit/s). Korektnost fragmentace byla ověřena s použitím hashovacího algoritmu MD5, jehož výsledky se na obou souborech shodovaly. Proto lze konstatovat, že vytvořená implementace fragmentuje korektně.

7.3.3 Test stability

Cílem tohoto testu bylo zjistit, zda vytvořená implementace snese dlouhodobý chod při maximální zátěži, především zda správně a korektně pracuje s pamětí při ukládání a uvolňování odesílaných i přijímaných paketů.

V tomto testu bylo úspěšně přeneseno 9,71 TB náhodných dat, délka testu byla přesně 75 hodin (tj. průměrná rychlost 301,68 Mbit/s). Na žádné z komunikujících stran nenastal problém s pamětí, všechna alokovaná paměťová místa byla správně uvolněna. Proto lze i tento test vyhodnotit jako úspěšný.

[2] Pro účely tohoto testu byla použita neomezená velikost bufferů.

Kapitola 8

Závěr

Cílem práce bylo navrhnout a implementovat transportní protokol použitelný jako komunikační vrstva aktivního směrovače vyvíjeného na Masarykově univerzitě v Brně. Navržený protokol měl vyhovovat hlavním požadavkům tohoto směrovače, zejména měl zajistit spolehlivý přenos dat bez nutnosti zachovávat jejich pořadí. Cíl práce byl splněn, poněvadž navržený protokol ARTP všem dříve uvedeným požadavkům vyhovuje.

K protokolu byla také vytvořena jeho vzorová implementace, která zahrnuje veškerou funkčnost navrženého protokolu. Její chování a efektivita byly testovány na dvou různě rychlých sítích – 100 Mbit/s a 1 Gbit/s. Zatímco na 100Mbit síti byl protokol konkurenceschopný existujícím transportním protokolům (UDP, TCP), na 1Gbit síti dosti zaostával (některé důvody tohoto jevu byly uvedeny v předchozí kapitole). A právě směrem k dosažení větší efektivity by se mohla ubírat další práce na protokolu a vytvořené implementaci. Možné způsoby, jak dosáhnout vyššího výkonu na vysokorychlostních sítích, jsou tyto:

- zefektivnění práce s pamětí a struktur použitých ve vytvořené implementaci – zejména odesílacích a přijímacích bufferů. Tohoto kroku lze díky modulární architektuře programu dosáhnout velmi snadno.
- navržení a důkladné otestování lepších způsobů manipulace s oknem zahlcení (*congestion window*).
- zavedení protokolu do vrstvy jádra operačních systémů Linux/Unix.

Kromě kroků vedoucích k větší efektivitě protokolu jej lze vyvíjet i v dalších směrech:

- přenesení protokolu do vrstvy jádra si vynutí úpravu vytvořené implementace vedoucí k podpoře více aplikací současně.
- do protokolu bude jistě potřeba implementovat další, doposud neznámé, vlastnosti vyvíjeného aktivního směrovače, které vyplynou z jeho dalšího výzkumu a vývoje.

Navržený protokol je jako komunikační vrstva vyvíjeného aktivního směrovače prakticky použitelný. Ačkoli výkon vypracované vzorové implementace lze z hlediska vysokorychlostních sítí hodnotit jako průměrný, je její rychlost pro širokou třídu aplikací zcela dostatečná. V první fázi vývoje směrovače je totiž hlavním požadavkem kladeným na všechny jeho vrstvy jejich plná funkčnost a čistota kódu, což implementovaná komunikační vrstva splňuje.

Literatura

[1] ISO OSI 7 Layer model and other models. May 2004. `http://www.hackerscenter.com/KnowledgeArea/papers/download/ISO_OSI_7_Layer_model_and_other_models.htm`

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, April 1999.

[3] S. Bhattacharjee, K. Calvert, and E. Zegura. Active Networking and the End-to-End Argument, 1997.

[4] T. Bova and T. Krivoruchka. Reliable UDP protocol. Internet-Draft, February 1999.

[5] Stefan Covaci (Ed.). *Active Networks*. Springer, GmbH & Co.KG, Tiergartenstrasse 17, Heidelberg, 1999.

[6] Atanu Ghosh, Michael Fry, and Glen MacLarty. An Infrastructure for Application Level Active Networking. 2000.

[7] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeXCompanion*. AW, 75 Arlington Street, Suite 300, Boston, 2003.

[8] Eva Hladká. Aktivní sítě – jedna z možných budoucností. Zpravodaj ÚVT MU, April 2000.

[9] Eva Hladká. User Empowered Collaborative Environment – Active Network Support, January 2004.

[10] Eva Hladká and Zdeněk Salvet. An Active Network Architecture: Distributed Computer or Transport Medium. In *First International Conference on Networking*, volume 2094 of *Lecture Notes in Computer Science*, pages 612–619. Springer, July 2001.

[11] Information Sciences Institute, University of Southern California. Internet Protocol. RFC 791, September 1981.

[12] Information Sciences Institute, University of Southern California. Transmission Control Protocol. RFC 793, September 1981.

[13] Van Jacobson and Michael J. Karels. Congestion Avoidance and Control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.

[14] U. Legedza, D. Wetherall, and J. Guttag. Improving the performance of distributed applications using active networks, 1998.

[15] D. Murphy. Building an Active Node on the Internet. Technical Report MIT/LCS/TR-723, 1997.

[16] J. Postel. User Datagram Protocol. RFC 768, August 1980.

[17] K. Psounis. Active Networks, Applications, Safety, Security, and Applications. *IEEE Communication Surveys Magazine*, 1999.

[18] Roberto Rinaldi and Marcel Waldvogel. Routing and data location in overlay peer-to-peer networks. Research Report RZ-3433, IBM, July 2002.

[19] Jiří Rybička. *LaTeXpro začátečníky*. Konvoj, spol. s r.o., Berkova 22, 612 00, Brno, 2003.

[20] B. Schwartz, W. Zhou, A. Jackson, W. Strayer, D. Rockwell, and C. Partridge. Smart Packets for Active Networks, January 1998.

[21] W. Richard Stevens, Bill Ferner, and Andrew M. Rudoff. *Unix Network Programming – The Sockets Networking API*. AW, 75 Arlington Street, Suite 300, Boston, 2004.

[22] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

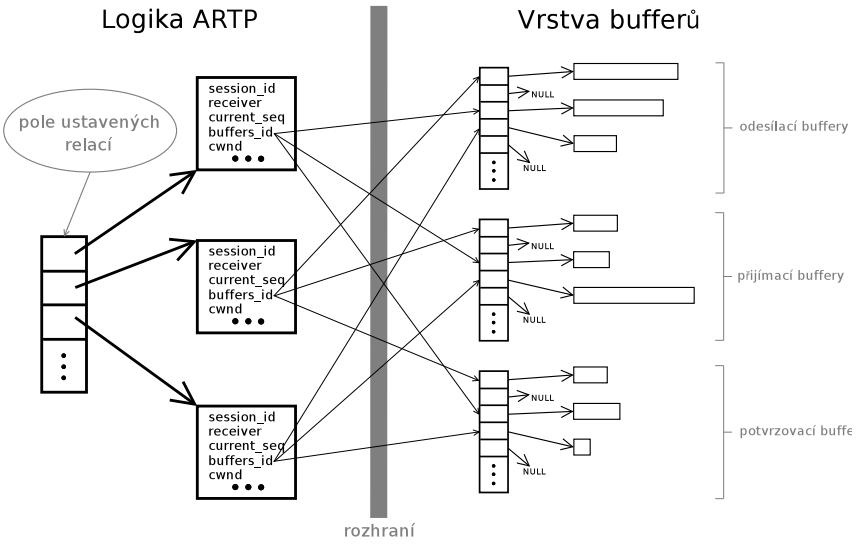
[23] D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE Conference on Open Architectures and Network Programming*, pages 117–129, April 1998.

[24] D. Wetherall, U. Legedza, and J. Guttag. Introducing new internet services: Why and How, 1998.

Příloha A

Architektura implementace

Architektura vytvořené implementace protokolu ARTP je důsledně rozdělena do několika vrstev, které mezi sebou komunikují přesně definovaným rozhraním. Mezi nejdůležitější vrstvy patří logika ARTP protokolu sloužící pro řízení protokolu a vrstva bufferů pro ukládání dat. Obrázek A.1 názorně ilustruje základní architekturu těchto dvou vrstev.



Obrázek A.1: Základní architektura protokolu

Pole ustavených relací v logické (ovládací) části popisované implementace ARTP protokolu obsahuje ukazatele na paměťové místo, ve kterém jsou uloženy všechny potřebné informace o založených relacích. Z důvodu jeho efektivního prohledávání je setříděno podle identifikačních čísel jednotlivých ustavených relací. Pole ukazatelů bylo zvoleno díky použití více vláken, která k tomuto poli přistupují. Vzhledem k tomu, že se uvnitř proto-

kolu předávají pouze ukazatele, není problémem toto pole kdykoli přetřídit či z něj nějakou relaci smazat. Každá relace má pak implementován jednoduchý *garbage collector*, který v momentě, kdy na ni neukazuje žádný ukazatel (tj. už není ani v poli ustavených relací), tuto relaci uvolní z paměti.

Jak lze vidět, logika protokolu je důsledně oddělena od pomocných ukládacích struktur (bufferů). Díky tomuto oddělení lze kdykoliv kteroukoliv pomocnou strukturu nahradit jinou (například efektivnější). Jedinou nutností je zachování definovaného rozhraní.

Příloha B

Implementované struktury

V této části budou popsány některé zajímavé struktury vytvořené implementace ARTP protokolu. Jedná se o tři druhy bufferů – odesílací, přijímací a potvrzovací. Budou zde vysvětleny úvahy vedoucí k jejich návrhu a základní principy práce s nimi.

Základem všech znázorněných struktur je pole ukazatelů, které ukazují na paměťové místo držící informace o bufferech jednotlivých založených relací. Důvod, proč byla tato struktura implementována právě takto, vyplyne z popisu výhod a nevýhod dalších možných řešení.

Všechny informace o bufferech by bylo možno vložit do dynamického pole, čímž by byl zachován přístup k potřebným informacím v konstantním čase (podle indexu v tomto poli). Problémy by však nastaly při správě paměti – při mazání relací by začaly vznikat díry, které by nebylo možno zmenšením pole odstranit, dokud by na jeho konci existovaly nějaké neukončené relace. Pole by tak zabíralo zbytečně mnoho paměti.

Další možností by bylo umístění informací o bufferech do spojového seznamu, který by uvedené problémy s pamětí zcela vyřešil. Tento přístup by však vedl ke ztrátě přímého přístupu k informacím o indexovaném bufferu (zvýšila by se časová náročnost díky prohledávání seznamu).

Všechny tyto nedostatky řeší implementovaná struktura – nabízí přímý přístup k informacím o žádaném bufferu se zanedbatelnou paměťovou náročností. Velikost vytvořeného pole ukazatelů se nezmenšuje, takže jeho velikost v daném časovém okamžiku odráží maximální počet relací, které byly do té doby najednou ustaveny. Jednosměrnost tohoto růstu však není problémem, poněvadž pole obsahuje pouze ukazatele (například při 1000 najednou ustavených relacích bude na 32-bitové architektuře v paměti zabírat 4 KB paměti, což lze zanedbat).

V odesílacích a přijímacích bufferech jsou implementovány seznamy, které jsou ukončeny zarážkou – prvkem, který má stejnou strukturu jako všechny ostatní prvky seznamu, avšak pomocí speciálního parametru je indikováno, že informace v něm obsažené jsou neplatné. Díky těmto prvkům není potřeba řešit výlučný přístup mezi přidáváním a vybíráním prvků

do jednotlivých seznamů (stačí pouze řešit výlučný přístup mezi více čtecími, resp. zapisovacími vlákny navzájem). Odstranění nutnosti výlučného přístupu je docíleno speciálním způsobem přidávání nových položek do seznamu – nový prvek je nejprve připojen za původní zarážku, přičemž je zároveň označen jako nová zarážka. Poté jsou informace v něm obsažené zkopírovány do původní zarážky, která je v dalším kroku označena jako platný prvek.

B.1 Struktura odesílacích bufferů

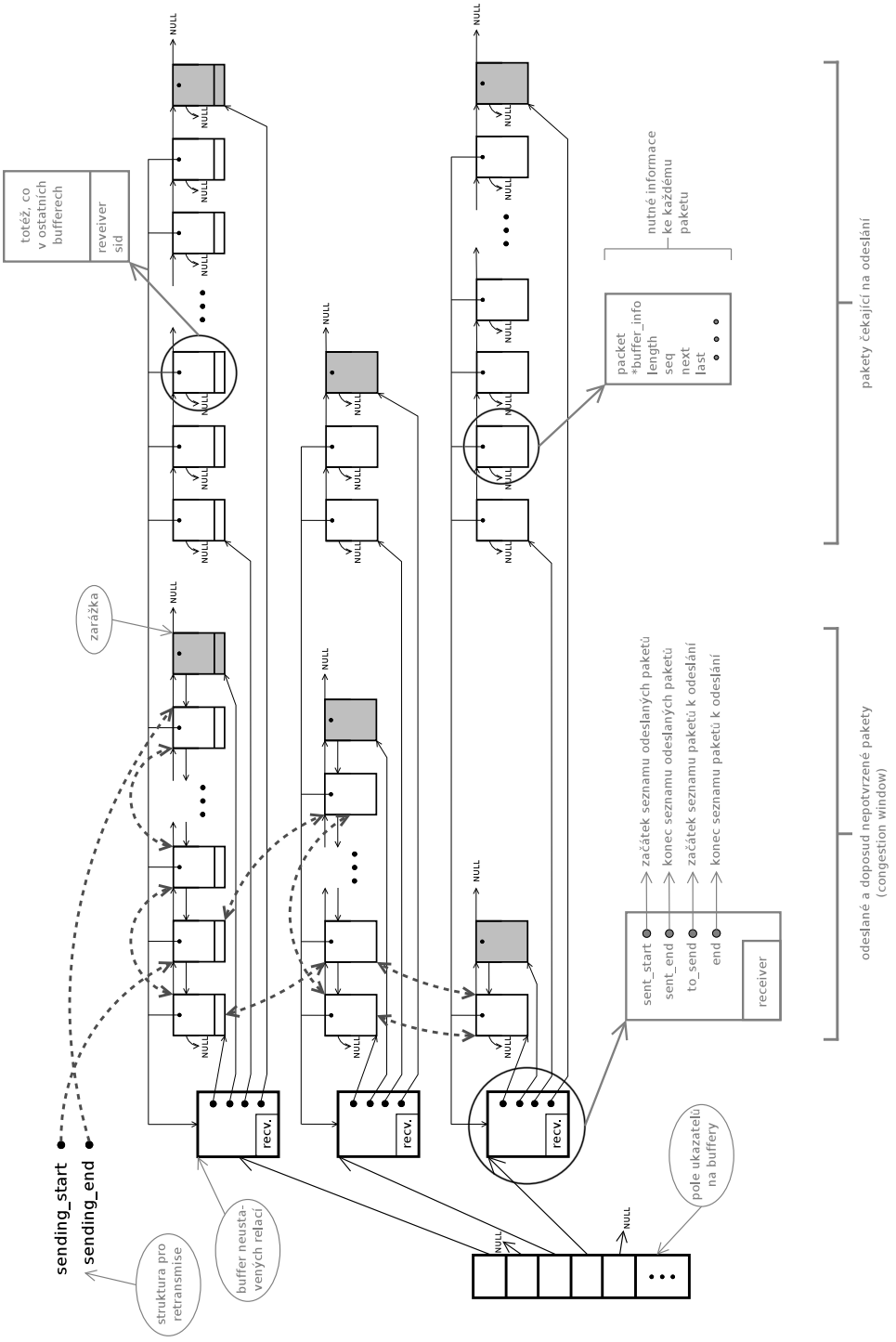
Tyto buffery slouží pro ukládání paketů, které jsou určeny k odeslání. Jejich struktura je znázorněna na obrázku B.1.

Vytvoření nového odesílacího bufferu znamená nalezení první volné pozice v poli založených relací. Pokud je nalezena, je vrácena jako identifikátor vytvořeného bufferu. Pokud nalezena není, je pole o tento nový prvek zvětšeno. Po nalezení správné pozice nového bufferu jsou vytvořeny a inicializovány všechny potřebné struktury.

Každý odesílací buffer ustavené relace se skládá ze dvou spojových seznamů – jeden je použit pro uložení odeslaných a doposud nepotvrzených paketů (*congestion window*), druhý pro uložení paketů čekajících na odeslání. Prvky obou seznamů obsahují nezbytné informace o uložených paketech, v případě neustavené relace je v nich navíc uložena adresa příjemce daného paketu (pro ustavené relace je součástí každého prvku ukazatel na strukturu popisující odesílací buffer dané relace, kde je její příjemce uložen). Informace o příjemci je v paketech nezbytná pro retransmise, jak bude vysvětleno dále.

Součástí odesílacích bufferů je také globální struktura pro retransmise, která je použita pro účely přeposílání nepotvrzených paketů. Touto strukturou jsou obousměrně provázány všechny odeslané a doposud nepotvrzené pakety všech relací. Ve struktuře jsou setříděny podle času svého nutného přeposlání, takže při zjišťování, zda je potřeba nějaký paket přeposlat, stačí nahlížet pouze na její vrchol. Zjišťování příjemce daného paketu určeného k přeposlání je z důvodu globální provázanosti této struktury prováděno s pomocí ukazatele na informace o celém bufferu, ve kterém se paket nachází[1] (v bufferu neustavených relací je příjemce uložen přímo ve struktuře popisující uložený paket).

[1] Struktura ukazuje pouze paket, který je nutno přeposlat – informaci, ve kterém bufferu je uložen, tak není možno zjistit.



Obrázek B.1: Struktura odesílacích bufferů

Základní funkce, které odesílací buffer poskytuje, jsou následující:

1. **Přidání paketu do bufferu**
   Nový paket se přidává na konec seznamu dané relace, ve kterém jsou uloženy pakety čekající na své odeslání. Díky zarážce probíhá přidávání podle dříve popsaného algoritmu.

2. **Získání paketu určeného k odeslání**
   Paket určený k odeslání je získáván z vrcholu seznamu obsahujícího pakety čekající na odeslání. Po jeho skutečném odeslání[2] je potřeba zavolat příslušnou funkci, která paket přemístí na konec seznamu odeslaných paketů. Tam bude uložen do té doby, než na něj přijde potvrzení o doručení. Zároveň je tento odeslaný paket vložen na správné místo ve struktuře pro retransmise.

3. **Získání paketu určeného k přeposlání**
   Paket určený k přeposlání je vybírán z čela struktury pro retransmise. Spolu s ním je nutno předat i čas jeho prvního odeslání, aby mohla vyšší vrstva protokolu detekovat rozpadlá spojení. Po jeho opětovném odeslání je podle času své další retransmise přesunut na nové místo ve struktuře.

4. **Smazání paketu, na nějž přišlo potvrzení o doručení**
   Paket, který byl správně doručen příjemci, je ze seznamu odeslaných paketů nutno smazat. Hledání se provádí na základě potvrzovaného sekvenčního čísla pouze v rámci bufferu dané relace (již nikoli globálně přes strukturu pro retransmise). V případě neustavených relací je hledání prováděno nejen podle sekvenčního čísla, ale i podle adresy příjemce a identifikace relace, ke které potvrzený paket náleží[3]. Před fyzickým smazáním je paket navíc vyjmut ze struktury pro retransmise.

B.2 Struktura přijímacích bufferů

Přijímací buffery jsou určeny k ukládání přijatých paketů. Jejich přesnou strukturu znázorňuje obrázek B.2.

Přijímací buffer se opět skládá ze dvou seznamů. Tentokrát je první seznam určen pro uložení přijatých úplných datagramů a druhý pro uložení

[2] Po získání paketu nemusí být odeslání uskutečněno například z důvodu zábrany zahlcení příjemce.

[3] Sekvenční čísla v bufferu pro neustavené relace nemusí být unikátní.

přijatých neúplných datagramů. Na konci seznamu úplných datagramů je z dříve popsaných důvodů implementována zarážka. V seznamu neúplných datagramů se však nevyskytuje, poněvadž zde vzájemné vyloučení není potřeba (k tomuto seznamu vždy přistupuje právě jedno vlákno).

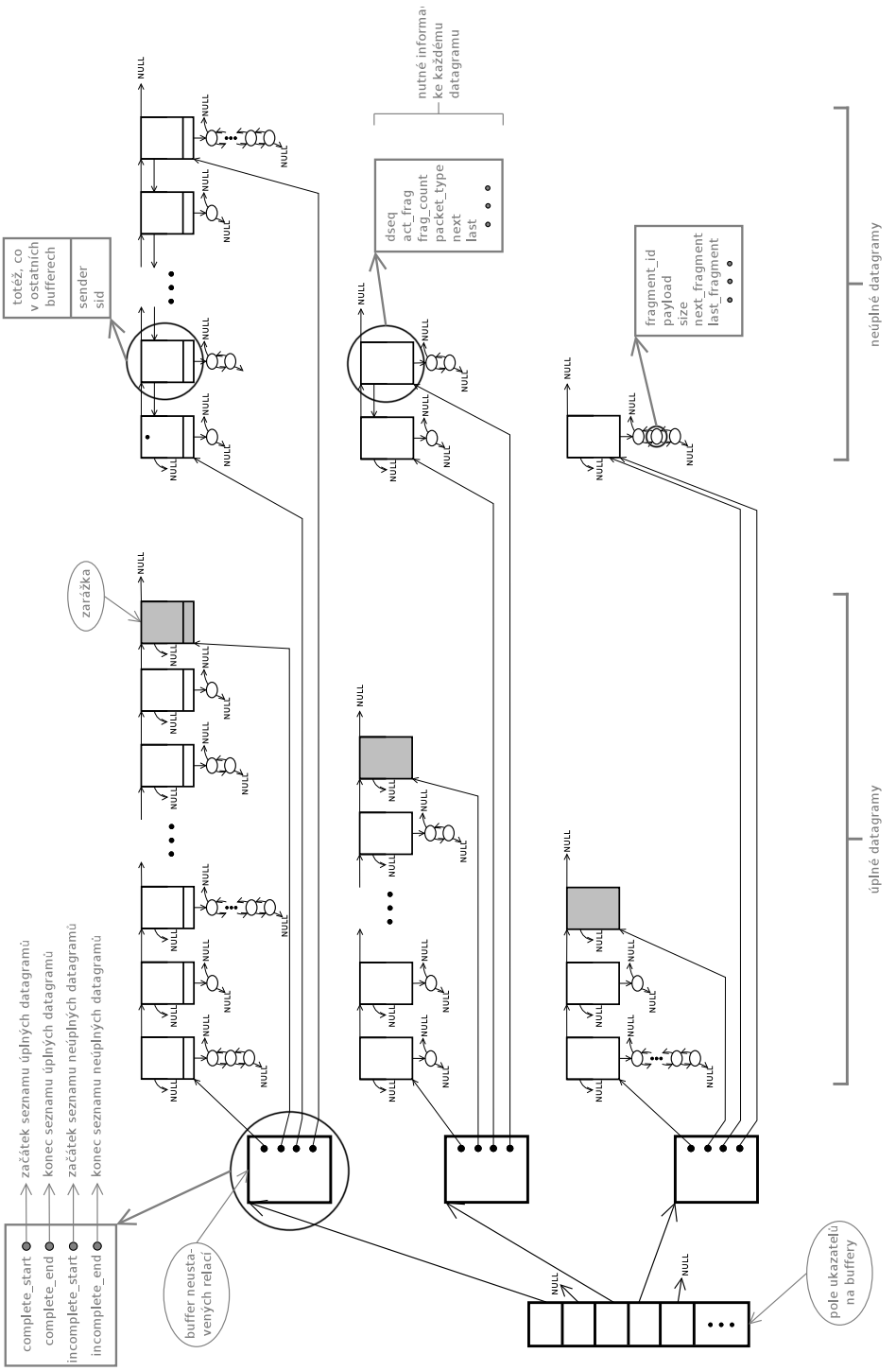
Základní funkce přijímacího bufferu jsou tyto:

1. **Přidání paketu do bufferu**
   Vkládání paketu do přijímacího bufferu je mírně odlišné podle toho, zda je tento paket fragmentem nějakého většího datagramu nebo zda popisuje celý datagram (tj. nebyl fragmentován).

   V případě vkládání datagramu, který nebyl fragmentován, je tento datagram s využitím zarážky ihned vložen do seznamu úplných datagramů.

   V případě vkládání paketu, který je fragmentem nějakého datagramu, je nalezena správná položka v seznamu neúplných datagramů popisující daný datagram (pokud taková neexistuje, je vytvořena nová). Položky jsou v tomto seznamu setříděny podle datového sekvenčního čísla uložených datagramů. V případě, že přišel poslední fragment, je celý datagram přesunut do seznamu úplných datagramů.

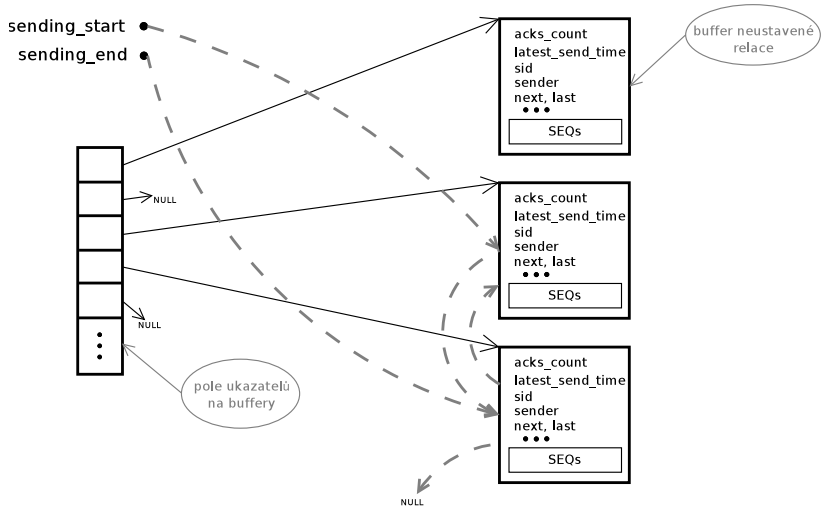
2. **Výběr úplného datagramu z bufferu**
   Pro získání úplného datagramu je vždy nahlíženo na vrchol seznamu úplných datagramů. Pokud tam nějaký datagram existuje, je složena jeho datová část a poté je předán vyšší vrstvě protokolu.



Obrázek B.2: Struktura přijímacích bufferů

B.3 Struktura potvrzovacích bufferů

Potvrzovací buffery slouží pro ukládání sekvenčních čísel přijatých paketů, která jsou v definovaných časových periodách vybírána a potvrzovacími pakety zasílána odesílateli.



Obrázek B.3: Struktura potvrzovacích bufferů

Základní struktura je opět totožná s předchozími – pole ukazatelů ukazuje na strukturu popisující daný potvrzovací buffer. Ten se však již neskládá z více částí, ale přímo obsahuje vyhrazené paměťové místo pro ukládání potvrzovaných sekvenčních čísel.

Ke každému vkládanému sekvenčnímu číslu je určen čas nejpozdějšího odeslání potvrzovacího paketu, ve kterém musí být obsaženo. Čas prvního vloženého sekvenčního čísla dané relace určuje její pořadí ve struktuře pro posílání potvrzovacích paketů, která všechny relace zřetězuje. Jakmile je dosažen maximální povolený počet sekvenčních čísel uložených v bufferu nějaké relace, je ve zmíněné struktuře přesunuta na začátek bez ohledu na čas, kdy měl být její paket odeslán (a tudíž bude odeslán okamžitě).

Sekvenční čísla určená k potvrzení se získávají v pořadí, v jakém jsou buffery jednotlivých relací uloženy ve struktuře pro přeposílání. Přednostně jsou však odesílány potvrzovací pakety neustavených relací.

Díky charakteru přijímacího algoritmu není při přístupu k těmto bufferům potřeba jakýkoli výlučný přístup, poněvadž k nim vždy přistupuje právě jedno vlákno.