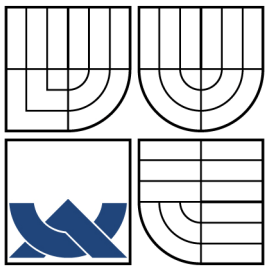
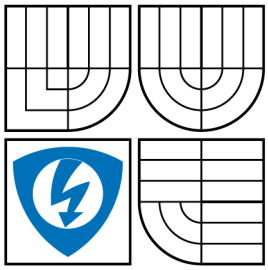


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE SÍŤOVÉHO PROTOKOLU DO PROSTŘEDÍ NETWORK SIMULATOR 2

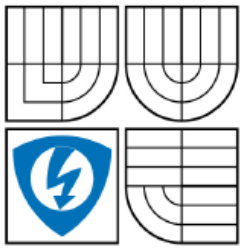
IMPLENETATION OF NETWORK PROTOCOL INTO NETWORK SIMULATOR 2 ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — Bc. ROBIN JANIGA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN KOUTNÝ
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Robin Janiga	***ID:***	84346
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Implementace síťového protokolu do prostřední network simulator 2

POKYNY PRO VYPRACOVÁNÍ:

Podrobně se seznamte se simulačním prostředím Network Simulator 2. Nastudujte problematiku multicastových přenosů ASM a SSM a jejich možnou implementaci do prostředí simulátoru. Na základě získaných znalostí navrhněte a realizujte knihovnu pro implementaci a simulaci protokolu hromadného sběru dat v síti SSM. Při realizaci se snažte pracovat modulárně tak, aby bylo možné protokol jednoduše rozšiřovat o další zprávy.

DOPORUČENÁ LITERATURA:

[1] Information Sciences Institute, "The Network Simulator - ns-2", [online], June 2004, dostupné <http://www.isi.edu/nsnam/ns/>
[2] M.A. SPORCTAK, Směrování v sítích IP, ComputerPress, a.s.,2004, ISBN: 80-251-0127-4

Termín zadání:	9.2.2009	***Termín odevzdání:***	26.5.2009

Vedoucí práce: Ing. Martin Koutný

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ANOTACE

Tato diplomová práce popisuje protokol pro systém hromadného sběru dat a jeho implementaci do prostředí Network simulator 2. Systém definuje dvě nové komunikační jednotky. Centrální jednotku CU a měřicí jednotku MU. Jednotky pak pracují podle pravidel definovaných komunikačním protokolem. Obsah této práce jsou následující. Na začátku je popsán simulační nástroj, konkrétně systém Ns-2 a nástroj pro vizualizaci simulačních výsledků, program NAM. Dále následuje popis navrženého komunikačního protokolu, jeho princip funkce, popis jednotek a komunikačních zpráv. Způsob komunikace mezi jednotkami. Především byl popsán multicast a jeho typy ASM a SSM. Jako doplnění byl popsán i unicastový princip komunikace. Dále následuje kapitola popisující způsob rozšíření simulátoru o vlastní protokol a podporu multicastové komunikace SSM. Přidání nového protokolu je reprezentováno naprogramováním agenta a aplikace a definici hlavičky nového protokolu. V této kapitole jsou také uvedeny nezbytné změny ve zdrojových souborech, nutné před rekompilací.

Hlavním cílem této diplomové práce je vlastní implementace navrženého protokolu. V programovacím jazyce C++ byli vytvořeni dva agenti, kteří reprezentují centrální a měřicí jednotku. Tito agenti byli zkompilováni do simulátoru a pomocí jednoduchého skriptu byla otestována jejich funkčnost. Simulační skript definoval 200 MU a jednu jednotku CU. Závěr práce je věnován simulaci zatížení společné linky mezi centrální jednotkou a ,,přístupovým,, uzlem. Bylo zjišťováno zda použití kumulativního způsobu potvrzovaní šetří přenosové kapacity linky oproti běžnému způsobu potvrzování.

Klíčová slova: implementace, síťový protokol, multicast, simulace, kumulativní potvrzení

ABSTRACT

This thesis describes a protocol for the multiple data collection system and his implementation into Network Simulator 2 environment. The system defines two communication units. CU central unit and measuring unit MU. The units operate according to the rules defined by communication protocol. The content of this work is as follows. At the beginning is described the simulation tool, namely a system NS-2 and a tool for visualization of simulation results, the NAM. This is followed by a description of the proposed protocol, his principle of functions, units description and communication messages. The method of communication between units. Mainly was described the multicast and the types of multicast ASM and SSM. Additionally, was described the principle of unicast communication. This is followed by chapter describing methods of enlargement simulator. Adding an own protocol and support of multicast communication SSM. Adding a new protocol is represented by programming a new agent, a new application and a new protocol header definition. In this chapter are also described the necessary changes in the source files that are need to the recompilation.

The main objective of this thesis is own implementation of the proposed protocol. In the programming language C++ were created two agents who represent a central and a measuring unit. These agents were compiled into a simulator and by using a simple script have been tested for functionality. The simulation script define MU 200 and one unit CU. Conclusion of work is devoted to simulation the load line between the central unit and "access" node. It was examined whether the use the method of cumulative acknowledgement saves the transmission capacity of line compared to normal method acknowledgement.

Keywords: implementation, network protocol, multicast, simulation, Cumulative Acknowledgement

Citace práce

JANIGA, R. *Implementace síťového protokolu do prostřední network simulator 2.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 49 stran a 2 přílohy. Vedoucí diplomové práce Ing. Martin Koutný.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma “Implementace síťového protokolu do prostředí Network simulator 2“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení §11 a následujícího autorského zákona č. 121.2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení §152 trestního zákona č. 140/1961 Sb.

V Brně dne …….. ……

(podpis autora)

OBSAH

1. Úvod

Následující diplomová práce se zabývá implementací dříve navrženého protokolu pro systém hromadného sběru dat, do simulačního prostředí Network simulator 2, dále Ns-2. Tento protokol představuje systém, který by měl v budoucnu sloužit pro automatizovaný sběr nejrůznějšího typu dat. Systém je tvořen centrální stanicí a mnoha měřicími jednotkami. Díky tomu, že systém využívá internet jako komunikační síť, mohou být jednotky rozmístěny kdekoliv na světě. Data z měřicích jednotek jsou posílány centrále, která je zpracovává nebo ukládá do databáze. Centrální a měřicí jednotky jsou členy jedné multicastové skupiny, a využívají k vzájemné komunikaci sítě SSM.

Cíle této práce jsou reprezentovány a zpracovány v několika kapitolách, které tvoří její vlastní náplň. V první kapitole je zevrubně představen zvolený simulační nástroj Ns-2. Následující kapitola popisuje způsoby komunikace v IP sítích, jako je unicast a multicast. Důraz je kladen zejména na problematiku multicastu a jeho typů ASM a SSM. Popis navrženého protokolu a jeho komunikačních zpráv je obsahem čtvrté kapitoly. Následující dvě kapitoly popisují obecné způsoby implementace nových agentů a multicastu SSM do Ns-2. Konkrétní návrh implementace výše zmíněného protokolu, jako realizace hlaviček, agentů a aplikací, je obsahem kapitoly sedmé. Poznatky z této kapitoly jsou pak využity k praktickému naprogramování nových agentů, reprezentujících funkci popisovaného protokolu. Zvolená řešení implementace jsou shrnuty do kapitoly 8. Nově vytvořený protokol je třeba simulovat za pomoci simulačních skriptů a zjistit případné nedostatky návrhu nebo implementace. Dosažené výsledky pak shrnuje poslední kapitola. Vlastní praktická implementace a simulace jsou pak hlavními cíli této diplomové práce.

2. Network simulator – ns2

Network simulator 2, dále ns-2, je simulátor diskrétních událostí, zaměřený na simulaci a výzkum sítí. Poskytuje značnou podporu pro simulaci známých síťových služeb. Jako jsou služby TCP, UDP, multicast i unicast protokoly, směřování na klasických drátových i bezdrátových sítích. Tento program je poskytován jako open source a je volně stažitelný z oficiálních webových stránek pro ns-2 [1], kde jsou rovněž pokyny a doporučení pro instalaci. Je třeba říci, že instalace není tak snadná jako u standardního GNU softwaru, tj. není ve stylu “configure, make, make install“, ale vyžaduje osobitější přístup, který je ovšem popsán na webu. Ns-2 je postaven pro běh na Unixových systémech, lze jej ovšem spustit i pod OS Windows a to za pomoci emulátoru Unixu, známého pod jménem Cygwin, který je opět volně stažitelný z webových stránek [2].

Ns-2 je simulátor, který využívá dvou programovacích jazyků, a to C++ a OTcl. Proč dva jazyky?

Vlastní program ns-2 je napsán v jazyce C++, jeho hlavní výhodou je rychlost při běhu programu, což je při simulacích v reálném čase potřeba. Nevýhodou a vlastně důvodem zavedení druhého programovacího jazyka je pomalost při vyhledávání a opravě chyb, změnám v kódu a rekompilaci.

Druhým jazykem je jazyk OTcl, který vyšel jako objektově orientovaný z jazyka Tcl. Je to jednoduchý a rychlý skriptovací jazyk, ve kterém se píší vlastní simulační skripty. Tyto skripty musí zahrnovat popis síťové topologie, komunikačních protokolů a událostí, tj. kdy a jaká mají být posílána vstupní data.

Ve skriptu lze dále specifikovat použitou frontu, její velikost, rozložení jednotlivých uzlů sítě, barevné odlišení jednotlivých datových toků … Vytváření dobrých simulačních skriptů je složitá úloha, vyžadující znalost vytvářeného modelu sítě, dostupných tříd simulátoru ns-2 a také znalost jazyka OTcl.

Pro svou činnost vyžaduje ns-2 dostatek operační paměti, a to zejména kvůli evidenci všech hlaviček paketů, které jsou právě v síti. Může se stát, že v rozlehlých síťových modelech, kdy se posílají sítí stovky tisíc paketů, může potřebná kapacita operační paměti dosahovat až do řádu gigabajtů. Pro snížení nároků na objem požadované paměti, existují příkazy, kterými omezíme počet hlaviček, jež budou pro každý paket evidovány.

remove-all-packet-headers

add-packet-header TCP IP

Ns-2 jak už bylo, je k dispozici jako open source, díky tomu umožňuje přidávání nových nebo modifikaci již používaných protokolů. V případě přidávání nového protokolu do ns-2 je zapotřebí přidat C++ kód, který specifikuje parametry a metody nového protokolu, a příslušný OTcl kód do datové základny simulátoru, díky němuž lze ze skriptů k těmto metodám a parametrům přistupovat.

NAM – Network AniMator

Nam je nedílnou součástí network simulátoru. Je to animační nástroj založen na Tcl/TK, který slouží pro vizualizaci simulovaných skriptů, na základě trasovacího souboru, vytvářeného v průběhu simulace samotným ns. Tento soubor obsahuje velké množství dat potřebných pro vizualizaci. Jsou to například topologické informace (uzly, agenti, spoje …) a také trasovací informace o přenesených paketech. Pro efektivní a rychle čtení těchto dat Nam ukládá jen minimální množství dat do paměti ale dle potřeby si je načítá ze souboru.

Po vytvoření trasovacího souboru se otevře animační okno Nam, které obsahuje několik ovládacích prvků pro podrobné zkoumání animace.

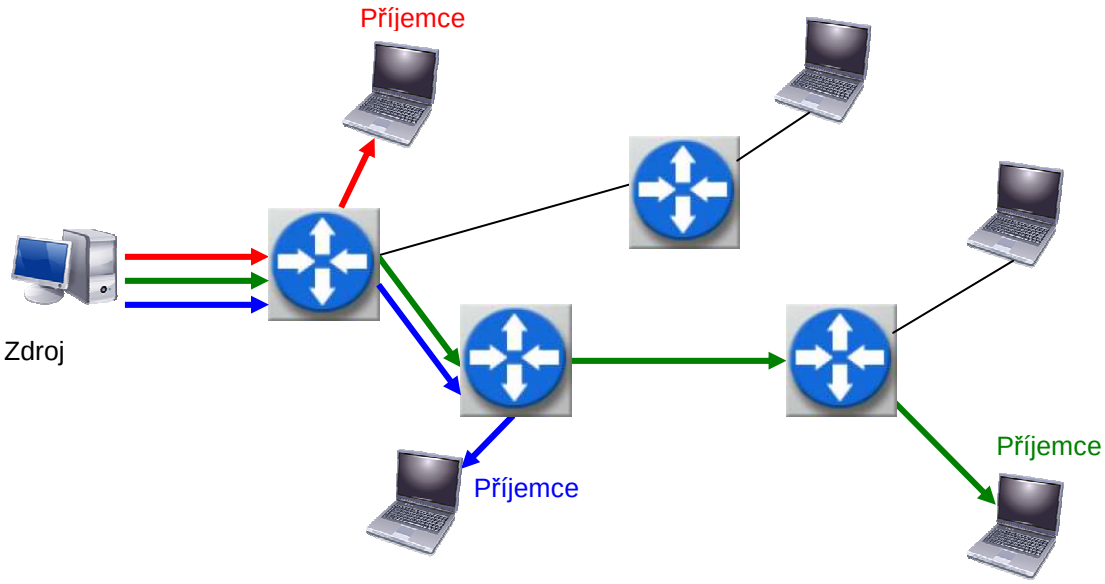
3. Způsoby komunikace v IP sítích

V IP sítích existují dva základní typy komunikace: unicast a multicast. Oba způsoby jsou ve svém principu rozdílné. V následujícím textu je popis obou typů přenosů s tím, že pro unicast bude zmíněn jen základní popis a princip, neboť tento není předmětem této práce.

3.1 Unicast

Unicast je první a tudíž nejstarší typ komunikace mezi jednotlivými uživateli používaný v IP sítích. Princip unicastu ukazuje **obr. 3.1**. Unicast je typ komunikace, který vychází z původní myšlenky komunikace IP sítí. Užilo se pro něj označení “Jeden k jednomu“. To znamená že data, defakto pakety, jsou zasílány od jednoho zdroje (odesílatele) k jednomu cíli (příjemci). Jak už bylo řečeno, tento typ komunikace je nejstarší a tudíž pro dnešní moderní multimediální přenosy nemá dostatečnou efektivitu. Vezměme třeba příklad, dnes hojně využívaného, internetového rádia či televize. Zde se vyžaduje přenos od jednoho zdroje k mnoha příjemcům. Realizace pomocí unicastu by vypadala

tak, že by bylo potřeba vytvořit pro každého příjemce (posluchače) vlastní unicastové spojení. Tato situace by byla přípustná jen pro malý počet posluchačů. S rostoucím počtem účastníků by počet nutných spojení mohl přesahovat několik tisíc, tím by docházelo ke značnému zatěžování sítě a použití unicastu by bylo neefektivní.



Obr.3.1.: Princip unicastového způsobu komunikace

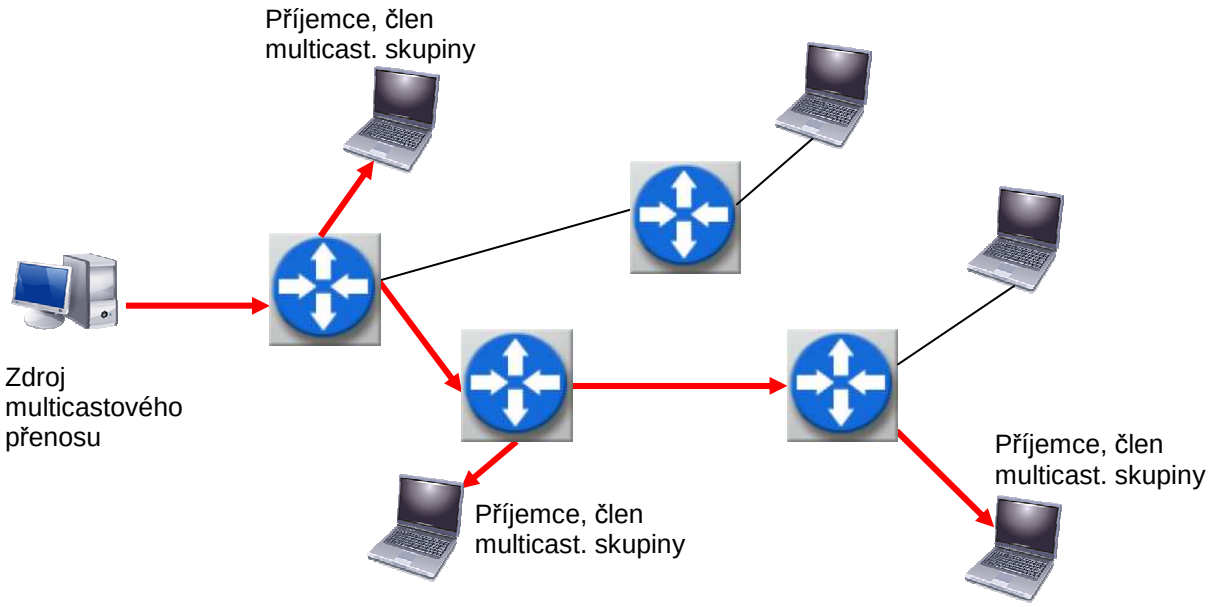
3.2 Multicast

Multicast je oproti unicastu mladší typ komunikace, který se zrodil začátkem osmdesátých let minulého století. Jeho tvůrcem a také průkopníkem byl Steve Deering, toho času student Stanfordské univerzity, který se zabýval prací na distribuovaném operačním systému schopném komunikace uvnitř jednoho ethernetového segmentu. Ve své práci zjistil, že komunikace pomocí ethernetových rámců je pro multicast nevhodná proti komunikaci pomocí IP paketů. Ve své doktorské práci, "*Multicast Routing in a Datagram Network*" [3], pak navrhnul způsob adresace multicastových paketů, jejich posílání sítí, navrhl také multicastovou verzi protokolů OSPF (MOSPF) a RIP (DVMRP) + základy protokolu IGMP. To bylo trochu historie a teď k principu multicastové komunikace.

Multicast je komunikační model, který využívá myšlenky “jeden (nebo více) k mnoho“. Na základě toho můžeme multicast rozdělit na dva případy:

- více zdrojů k více příjemcům, “mnoho k mnoho“ nazývané zkratkou ASM (Any Source Multicast)
- jeden zdroj k více příjemcům, “jeden k mnoho“ nazývané zkratkou SSM (Specific Source Multicast)

Vezmeme li si příklad z unicastu o internetovém radiovém nebo televizním vysílání, pak multicast umožňuje vysílat data pouze jednou s tím, že kopie vysílaných dat budou doručeny všem posluchačům. Příklad multicastového způsobu komunikace ukazuje **obr.3.2**.



Obr.3.2.: Princip multicastového způsobu komunikace

Multicastový model komunikace je tedy možné charakterizovat jako zvláštní případ všesměrového vysílání, kdy jednotlivé pakety jsou opatřeny mimo jiné multicastovou adresou, která specifikuje skupinu příjemců, kterým jsou pak pakety doručovány. Adresování paketů je spojeno s množstvím problémů, například je třeba zaručit aby se data neposílala do směrů, kde není posluchač (člen multicastové skupiny). Nastává zde problém s dynamickým přihlašováním do skupiny. Toto řeší tzv. distribuční stromy.

3.3 Distribuční stromy

Distribuční strom lze brát jako takový plán rozmístění jednotlivých členů multicastové skupiny. Vytváří a spravují si jej jednotlivé směrovací prvky sítě a na jeho základě jsou pakety doručovány příjemcům. Distribuční stromy můžeme rozdělit do dvou skupin na Zdrojový a Sdílený distribuční strom.

3.3.1 Zdrojový strom

Někdy také označován jako strom nejkratších cest. Tento typ stromu využívá SSM. Kořen stromu tvoří zdroj vysílaných dat, jednotliví posluchači tvoří listy tohoto stromu. Zdrojový strom je tvořen pouze pro jednu multicastovou skupinu. Existuje li více zdrojů posílajících data to stejné multicastové skupiny, musí se pro *n* zdrojů vytvořit dalších *n* zdrojových stromů.
Pro označení tohoto typu stromu se obvykle používá notace (S,G), kde S představuje adresu zdroje vysílání (Source address) a G označuje adresu multicastové skupiny (Multicast Group address).

3.3.2 Sdílený strom

Tento strom se od předchozího liší především v tom, že obsahuje cesty od více zdrojů k více příjemcům. Z toho je patrné, že tento typ stromu je využíván u ASM. Další odlišností je to, že kořen stromu nebývá jeden ze zdrojů ale některý

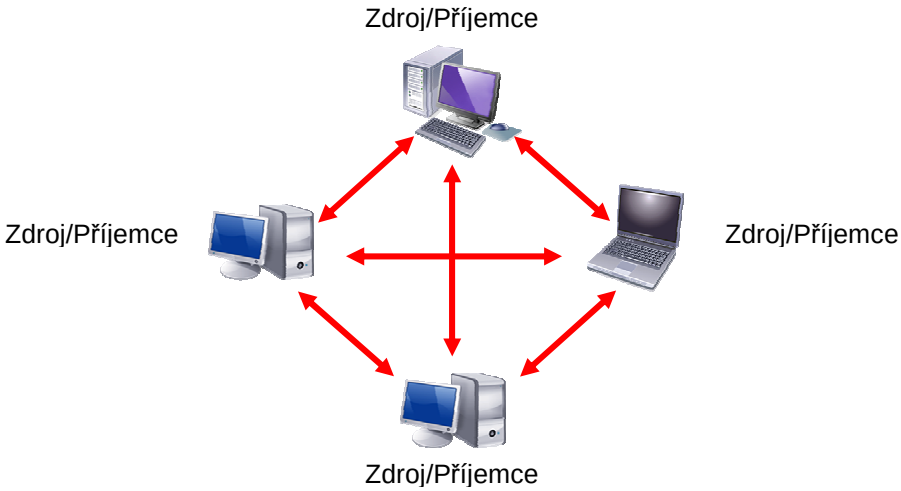
ze směrovacích prvků. Tomuto kořenu se pak říká randesvous point, zkráceně RP. Někdy se tyto stromy nazývají RP stromy.
Sdílené stromy se označují natací (*,G), kde hvězdička označuje, že strom není závislý na zdroji multicastu. G má obdobný význam jako u zdrojového stromu.

Pro oba typy distribučních stromů platí, že každá podsít, která obsahuje minimálně jednoho příjemce, je ve stromu reprezentována jako jedna větev. Pokud přijímač skončí svou účast v multicastové skupině, je tato větev z distribučního stromu vyčleněna. V místě dělení větvě do více směrů, jsou směrovacím prvkem vytvářeny kopie dat a posílány dále do vycházejících větví.

3.4 Základní typy multicastu

3.4.1 ASM (Any Source Multicast)

Jak už bylo naznačeno, tato metoda vychází hesla „mnoho k mnoha“. Pro tuto metodu je vytvářen sdílený distribuční strom. Všechny komunikační body v této síti mohou jak přijímat data od zdrojů tak je samy vysílat. Tzn, že každý z účastníků může být zároveň zdroj vysílání i posluchač. Princip ukazuje **obr.3.3**. Pakety jsou proto směrovacími prvky kopírovány a posílány do všech směrů. Pak záleží na každém účastníkovy zda paket příjme či zahodí. V případě, že se chce účastník přidat do probíhající konference (multikcastové skupiny), síť musí zjistit všechny potenciální zdroje a jejich data doručit novému účastníkovy. Všichni již připojení účastnící jsou o přidání nového bodu informováni.



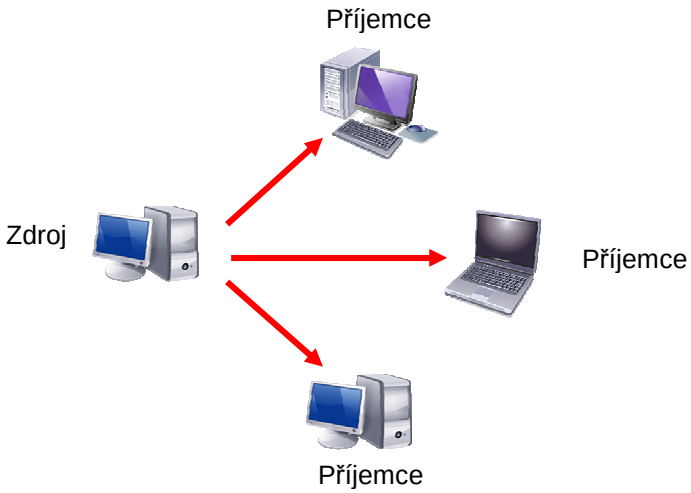
Obr.3.3.: ASM – „mnoho k mnoha“

Z předešlého je patrné, že pro metodu ASM je potřeba vytvořit složitou strukturu směřování. ASM se v praxi využívá například pro on-line hry, hlasové i obrazové konference.

3.4.2 SSM (Specific Source Multicast)

Druhou metodou realizace multicastu je SSM. Vychází z hesla „jeden k mnoha“. Princip komunikace ukazuje **obr.3.4**. Tato metoda využívá zdrojový distribuční strom. Z toho plyne, že SSM lze využít pouze v případě, kdy v jedné multicastové skupině existuje pouze jeden zdroj vysílaných dat. Ostatní členové skupiny musí být příjemci (posluchači). Případný jiný zdroj vysílání je příjemci ignorován.

Tohoto je dosaženo dvojicí adres (notace (S,G) u zdrojového stromu). Multicastovou adresu, což je adresa multicastové skupiny, ke které se přihlásil a jejíchž data chce přijímač přijímat. A unicastovou adresu, což je adresa zdroje od kterého bude přijímat data.



Obr.3.4.: SSM – „jeden k mnoha“

SSM má výhodu ve své jednoduchosti a v menší realizační náročnosti. Používá se například u již zmíněné internetové televize a rádia.

Při aplikacích metod ASM a SSM se využívají tři typy roubovacích protokolů. Zde popisuji jen dva.

3.5 Typy protokolů v multicastu

3.5.1 Dense mode protokoly

Protokoly této skupiny využívají zdrojové stromy k doručování SSM a pracují na tzv. push principu. Tento princip spočívá v předpokladu, že v každé podsíti se nachází příjemce multicastového provozu a data jsou tedy přenášena všude. Aby se zabránilo plýtvání pásma, každý list stromu, který neobsahuje žádného příjemce pro danou skupinu, pošle směrem ke kořenu tzv. zprávu **prune**. Směrovač, který tuto zprávu příjme, odřízne větev z něhož zpráva přišla. Tímto způsobem postupují všechny listy a zůstanou pak jen větvě,které obsahují nějakého příjemce. **Prune** zpráva má omezenou životnost, je nutné ji v časových intervalech opakovat.
V případě připojení nového posluchače je nadřazenému routeru poslána zpráva **graft** a směrovač ihned přidá větev do seznamu, kterým jsou data posílána.
Mezi dense mode protokoly patří například:

DVMRP (Distance Vector Multicast Routing Protocol) jeden z nejstarších protokolů, koncepcí připomíná unicastový RIP. Sám si sestavuje unicastovou roubovací tabulku, používán už jen z důvodu kompatibility.

PIM-DM (Protocol Independent Multicast – Dense Mode) mladší protokol, nesestavuje si sám unicastovou roubovací tabulku ale využívá ji od jiného protokolu, umí spolupracovat s libovolným protokolem.

3.5.2 Sparse mode protokoly

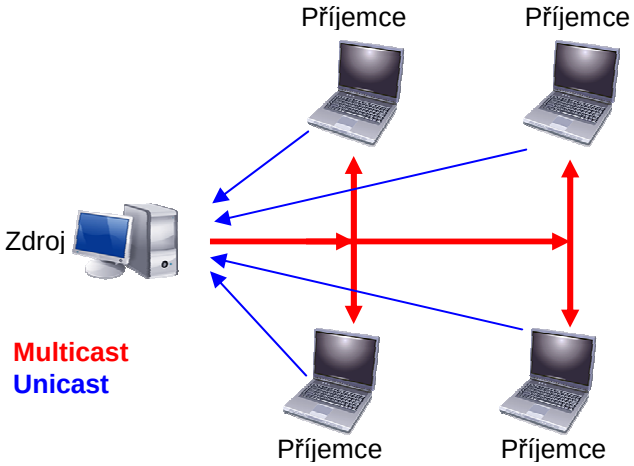
Tyto protokoly využívají sdílené distribuční stromy pro doručování ASM a využívají tzv. Pull mode. Model předpokládá posílání dat do sítě jedině na explicitní žádost. Přidání nového posluchače k multicastové skupině se děje odesláním zprávy **Join** ke kořenu stromu. Tím je vytvořena nová větev a data mohou proudit. Platnost zprávy **Join** je opět časově omezená. Pokud v některé větvi už nejsou posluchači, routek pošle zprávu **graft** pro uříznutí dané větve.
Do sparse mode protokolů patří například protokol PIM-SM

Sparse mode protokoly šetří, oproti dense mode protokolům, výpočetní výkon routerů. Spočívá to v tom, že pro více zdrojů se u desne mode musí pro každého přispívatele vypočítat vlastní strom. Toto u sparse mode odpadá. Ty ale mohou být méně efektivní, a je nutné dobře zvolit kořen RP.

4. Protokol hromadného sběru dat

4.1 Úvod

Následující text bude věnován definici protokolu, který popisuje chování a realizaci navrženého systému hromadného sběru dat [4]. Systém by měl být tvořen jednou centrální jednotkou a několika měřícími stanicemi. Tyto stanice sledují (měří) nějakou definovanou veličinu a takto naměřená data posílají v přesných časových intervalech centrální sběrné jednotce, která tato data ukládá do databáze.



Obr.4.1.:Definice hromadného sběru dat

Systém je založen na modifikovaném protokolu RTP/RTCP se zpětným signalizačním kanálem. Princip definovaného systému ukazuje **obr.4.1**. Jak lze vidět, systém obsahuje centrální bod, který z pohledu SSM komunikace zastává pozici zdroje. Komunikace s měřícími stanicemi probíhá pomocí multicastového přenosu SSM (odpovídá protokolu RTP). Centrální sběrná jednotka kromě ukládání naměřených dat rovněž zajišťuje jejich potvrzování a vzájemnou synchronizaci s měřicími jednotkami. Jak už bylo řečeno, komunikace a tudíž i potvrzování je řešeno pomocí multicastu. Aby však nedocházelo k nadměrnému zatěžování sítě, je potvrzování přijatých dat prováděno formou kumulativního

potvrzení. To znamená, že centrální jednotka potvrzuje v určitých časových intervalech několik správně přijatých dat.
Dále systém obsahuje měřící jednotky, které provádí vlastní měření a částečně se podílí na synchronizaci. Z pohledu SSM zastávají měřící stanice pozici přijímače, které znají multicastovou adresu skupiny, ke které náleží. Komunikace s centrální sběrnou jednotkou probíhá ve formě unicastu (odpovídá protokolu RTCP).
K zajištění správného fungování výše popsaného systému, bylo navrženo několik typů komunikačních zpráv.

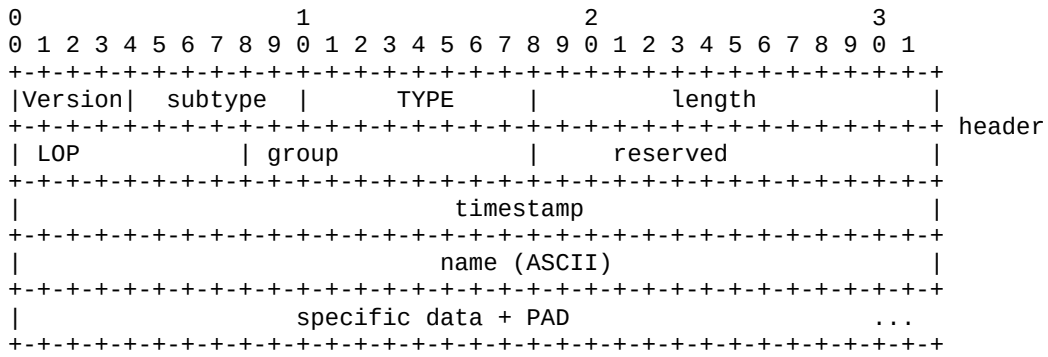
Typy komunikačních zpráv

V systému jsou definovány tyto typy zpráv. Z hlediska směru komunikace centrála – stanice je dělíme na:

- zprávy UM (unit message) ve směru komunikační jednotky -> centrála
- zprávy SM (server message) ve směru centrála -> komunikační jednotky

4.2 Zprávy typu UM (Unit Message)

Tyto zprávy slouží pro přenos ve směru komunikační jednotka -> centrála. Přenos těchto zpráv funguje na unicastovém principu a jsou adresovány na adresu centrály (zdroje) což odpovídá notaci (S,G). Jsou rozlišovány dva podtypy zpráv, jejichž formát je stejný a ukazuje ho **obr.4.2**.



Obr. 4.2: Formát UM zprávy

Význam jednotlivých polí zprávy UM:

Version – verze protokolu – 4 bity pro identifikaci verze protokolu.
Type – typ paketu. Jsou definovány dva základní typy – UM-MV a UM-SY .
Subtype – podtyp protokolu, pole je využito pro identifikaci konkrétních dat. V případě zprávy obsahující měřená data, bude pole subtype vyplněno nulami.
Length – celková velikost přenášených dat.
Timestamp – zobrazuje aktuální dobu odečtu. Pro tuto dobu je vyhrazeno 32 bitů. Čas je zde uložen v tzv. sekundovém unix formátu, který udává aktuální počet sekund od 1.1.1970 00:00. V případě, že se bude jednat o aplikační paket sloužící k synchronizaci, bude mít toto pole nulovou hodnotu.

Name – slouží pro identifikaci měřící jednotky, která je odesílatelem dané zprávy. Pro tuto hodnotu je vyhrazeno 32 bitů a toto číslo je pro každou jednotku unikátní. Tak je zaručeno, že se ve stejné síti nevyskytne jednotka se stejným identifikačním číslem.
LOP – označuje počet bitů, které musely být přidány pro zachování integrity násobku 32-bitové posloupnosti. 8 bitů je zvoleno z důvodu zachovaní integrity s ostatními typy zpráv.
Group – označuje specifickou skupinu měřiče (elektroměr, vodoměr, plynoměr, teploměr). Tento blok se vyznačuje délkou 10 bitů, které umožňují v jedné síti utvořit 1022 skupin. Nultý blok a poslední blok je určen pro speciální účely.
Reserved – v protokolu je vyhrazený prostor 16 bitů, který je určen pro další využití.
Specific data – posledním blokem jsou specifická data měřiče. Tento blok má předem danou délku a je dorovnáván na délku násobku 32 bitů.

4.2.1 Zpráva UM-MV (Unit Message – Measeured Values)

Tento podtyp zprávy UM slouží pro přenos naměřených dat od měřicích jednotek k centrále. Centrální jednotka tuto zprávu přijímá, data ukládá do databáze a pro každou zprávu generuje kumulativní potvrzení.

4.2.2 Zpráva UM-SY (Unit Message – Synchronization Message)

Tento podtyp zprávy slouží pro synchronizaci komunikační jednotky se sběrnou centrálou. Potřebuje li se komunikační jednotka zasynchronizovat, pošle centrále zprávu UM-SY a čeká na odpověď od centrály ve formě zprávy SM-SY.
Zpráva má definovány tři podtypy:

- UM-SY-I (Init) – inicializační zpráva pro zahájení synchronizace, která je zasílaná jednotkou směrem k centrále.
- UM-SY-C (Confirm) – zpráva potvrzující úspěšnou synchronizaci, kterou jednotka vyšle po úspěšném přijetí zprávy SM-SY-I.
- UM-SY-B (Bye) – ukončující zpráva, jednotka pomocí této zprávy může ukončit proces synchronizace.

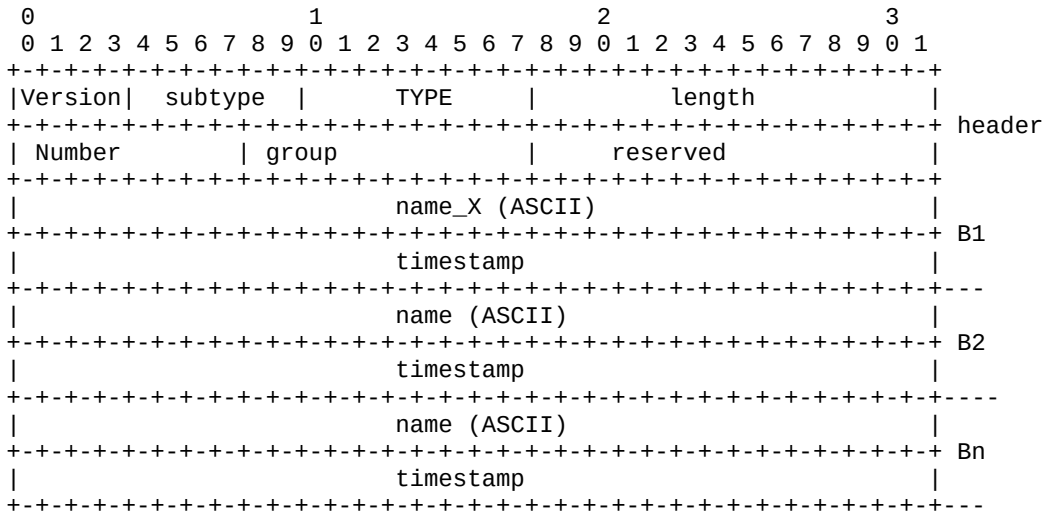
4.3 Zpráva SM (Server Message)

Tyto zprávy slouží ke komunikaci ve směru centrála -> měřicí jednotky. Komunikace probíhá pomocí multicastu. Jsou definovány tři typy SM zpráv.

4.3.1 Zpráva SM-CA (Server Message – Cumulative Acknowledgement)

Tento typ zprávy je určen pro kumulativní potvrzování přijatých dat od měřicích jednotek. Centrální jednotka si vede frontu přijatých datových zpráv UM-MV. Po

jejím naplnění odešle multicastově zprávu SM-CA a frontu vynuluje. Navržená podoba této zprávy je na **obr. 4.3**. Komunikační jednotky pak potvrzovací zprávu přijímají a na základě položek Name (jméno stanice) a časového razítka (timestap) identifikují potvrzení svých dat. Toto řešení má nespornou výhodu v šetření komunikačního pásma na rozdíl od unicastového způsobu potvrzování.



Obr. 4.3: Formát SM-CA zprávy

Význam jednotlivých polí zprávy SM-CA je následující:

Version – verze protokolu – 4 bity pro identifikaci verze protokolu.
Type – typ paketu. Jsou definovány tři základní typy – SM-CA.
Subtype – podtyp protokolu, pole je využito pro identifikaci konkrétních dat. V případě zprávy obsahující měřená data, bude pole subtype vyplněno nulami.
Length – celková velikost přenášených dat.
Number – celkový počet potvrzení, které nese daná zpráva.
Group – označuje specifickou skupinu měřiče (elektroměr, vodoměr, plynoměr, teploměr). Tento blok se vyznačuje délkou 10 bitů, které umožňují v jedné síti utvořit 1022 skupin. Nultý blok a poslední blok je určen pro speciální účely.
Reserved – v protokolu je vyhrazený prostor 14 bitů, který je určen pro další využití.
Name – slouží pro identifikaci měřící jednotky, která je odesílatelem dané zprávy. Pro tuto hodnotu je vyhrazeno 32 bitů a toto číslo je pro každou jednotku unikátní. Tak je zaručeno, že se ve stejné síti nevyskytne jednotka se stejným identifikačním číslem.
Timestamp – zobrazuje aktuální dobu odečtu. Pro tuto dobu je vyhrazeno 32 bitů. Čas je zde uložen v tzv. sekundovém unix formátu, který udává aktuální počet sekund od 1.1.1970 00:00. V případě, že se bude jednat o aplikační paket sloužící k synchronizaci, bude mít toto pole nulovou hodnotu.

Za hlavičkou pak následují 64 bitové bloky kumulativního potvrzování označované jako B1 až Bn, kde n je celkový počet potvrzení v jedné zprávě. Ten byl stanoven na základě maximální délky ethernetového datového prostoru (1500B), délky hlaviček a potvrzovacích bloků na 183 [4], dle vztahu (1).

$$E_{\max} = \frac{[1500 - (hlavicky)]}{delka\_bloku\_B} = 183.$$

$$E_{\max} = \frac{[1500 - (20 + 8)]}{8} = 183. \quad (1)$$

Kde: hlavičky – IP hlavička + UDP hlavička = 8 B + 20 B
Delka_bloku_B – délka jednoho kumulativního bloku = 8B

4.3.2 Zpráva SM-SY (Server Message – Synchronization Message)

Tento typ zprávy bude sloužit k synchronizaci centrální jednotky a konkrétní měřicí stanice. Na tuto zprávu stanice čeká po vyslání zprávy UM-SY. Je zřejmé, že synchronizaci vždy inicializuje měřicí jednotka. Formát synchronizační zprávy SM-SY je odpovídající ke zprávě UM-SY, a ukazuje ho **obr.4.2**.
Zpráva SM-SY má definovány tři podtypy:

- SM-SY-I (Init) – inicializační zpráva pro potvrzení zahájené synchronizace, která je zasílaná centrálou směrem k jednotce.
- SM-SY-P (Parameters) – zpráva nesoucí řídící parametry potřebné pro počáteční synchronizaci.
- SM-SY-B (Bye) – ukončující zpráva, centrála pomocí této zprávy může ukončit proces synchronizace.

4.3.3 Zpráva SM-CI (Server Message – Control Informations)

Tento typ zprávy by měl sloužit k multicastovému přenosu řídících a informačních dat směrem ke všem členům skupiny. Informace nesené v této zprávě se mohou týkat například režijních informací o centrále, resynchronizace při posuvu času nebo třeba změny fyzické adresy centrální stanice. Formát zprávy je obdobný jako předchozí zprávy a ukazuje ho **obr.4.2**. Rozlišení jednotlivých typů zpráv je identifikováno v poli Type a jednotlivé podtypy zpráv v poli Subtype.
Zpráva SM-CI má definovány tyto čtyři podtypy:

- SM-CI-T (Time) – řídící zpráva obsahující aktuální čas sběrné centrály.
- SM-CI-CN (Client Number) – řídící zpráva obsahující aktuální počet komunikačních jednotek v síti *CN*.
- SM-CI-MT (Measure Time) – řídící zpráva nesoucí časový interval t_D, který definuje maximální dobu na odeslání naměřených dat.
- SM-CI-A (Alive) – řídící zpráva oznamující dostupnost centrály. Tato zpráva je vysílána vždy jednou za dobu $9 \cdot t_D$.

5. Multicast v ns2

Popis programu network simulator 2 už byl popsán v první kapitole. Bylo řečeno, že dokáže simulovat různé typy sítí, síťové služby a protokoly. A to jak v komunikaci unicastové tak multicastové. Tato kapitola popisuje podporu multicastových protokolů v Ns2.
Network simulator 2 byl vyvinut na základě dvou objektově orientovaný programovacích jazycích, C++ a OTcl. V jazyce C++ jsou psány ty prvky, které potřebují větší výpočetní výkon jako například procedury, funkce, třídy, atd. Kdežto v jazyce OTcl jsou psané prvky, u kterých je potřeba neustálé zdokonalování a několik výpočetních zpracování. Toto však při vývoji multicastu nebylo zcela dodrženo a téměř všechny multicastové zdrojové kódy jsou psané v jazyce OTcl. Tyto soubory lze nalézt ve složce “tcl/mcast“.

- ***NS-mcast.tcl,*** vytváří hierarchickou třídu odvozenou od třídy Simulátor. Tato třída popisuje speciální multicast příkazy, vyžadované uživatelem pro chod multicast simulátoru;
- ***McastProto.tcl****,* soubor, který definuje hlavní multicast třídy, McastProcotol, je to třída, která je kořenem všech níže uvedených multicast tříd.
- ***ST.tcl,*** popisuje chování multicast protokolu Simplified Sparse Mode.
- ***BST.tcl****,* popisuje chování multicast protokolu BST - Bi-directional Shared Tree Mode
- ***DM.tcl****,* popisuje chování multicast protokolu PIM-DM.

Ctr-MCAST, v této složce jsou tři soubory, které jsou zodpovědné za simulaci centralizovaného Multicastu.

- **CtrMcast.tcl,** popisuje chování multicast protokolu PIM-SM, že CtrMcast třídy odvozené formy McastProtocol.
- **CtrMcastComp.tcl**, reprezentuje směrovací funkce objektu.
- **CtrRPComp.tcl**, tato třída simuluje chování RP bodu v režimu PIM-SM;

Z toho je zřejmé, že network simulator verze 2 podporuje protokoly:
PIM-SM
PIM-DM
Simplified Sparse Mode
Bi-directional Shared Tree Mode

Protokol, který je v této práci popsán v předešlé kapitole, je definován pro multicastovou komunikaci SSM. Tento protokol však v NS2 není primárně zapouzdřen, ale musí se dodatečně přidat do zdrojových souborů Ns2 před jeho samotnou instalací.

5.1 Implementace SSM do Ns-2

Při implementaci protokolu PIM-SSM do network simulátoru se vycházelo s jisté podobnosti s již implementovaným protokolem PIM-SM. Vývoj SSM byl z tohoto důvodu založen na kódu centralizovaného multicastu. Mezi hlavní odlišnosti obou protokolů bylo:

- Join a Prune zprávy museli specifikovat zdroj (S – Source) a skupinu (Group - G) multicastového kanálu. Směšovače pak nemohou přijímat zprávy od více zdrojů dle notace (*, G) jako u PIM-SM protokolu.
- V SSM už není potřeba tzv. randevous point RP. Kořen stromu je pak zdroj (Sourse - S) což odpovídá SSM.

Chce li uživatel při simulaci využívat multicastový protokol, musí jej specifikovat voláním funkce mrproto.

```
> $ ns mrtproto (CtrMcast, DM, ST, BST)
```

,kde CtrMcast (Centralizovaný Multicast), DM (Dense Mode), ST (zjednodušená Sparse Mode) a BST (Bi-directional Shared Tree Mode) jsou podporované multicast protokoly.
Pro podporu SSM byly vytvořeny dvě nové třídy, které byly hierarchicky rovné již implementovaným protokolům. Třídy byly vytvořeny jako dva soubory, “SSMMcast“ a “SSMMcastComp“, psané v jazyce OTcl.
Po vytvoření požadovaných tříd bylo třeba přidat do souboru “ns-mcast.tcl“ kód, který zajistí přidání argumentu pro SSM do funkce mrproto. Tučně je vyznačen nový kód.

```
>    set MrtHandle_ ""
>    if { $mproto == "CtrMcast" } {
>         set MrtHandle_ [new CtrMcastComp $self]
>         $MrtHandle_ set ctrrpcomp [new CtrRPComp $self]
>    }
>    if { $mproto == "SSMMcast" } {
>         set MrtHandle_ [new SSMMcastComp $self]
>    }
>    if { $mproto == "BST" } {
>    …
```

Dále musela být upravena struktura třídy CtrMCast, která není kompatibilní s chováním SSM. Byla vytvořena třída se strukturou SSMMcast. Tato struktura obsahuje dvě pole SGList a MList.
SGList je realizován jako dvourozměrné pole. Toto umožňuje vytvoření několika multicastových skupin, které mohou mít jeden společný zdroj (Source), ale každá skupina má svůj vlastní SSM kanál, dle notace (S,G). Například může existovat (S,G1) a (S,G2).
MList je realizován jako trojrozměrné pole a udržuje informace o všech uzlech, které ve specifickou dobu, přijímají informace z určitého (S,G) kanálu.

Další úpravou pro soulad s protokolem SSM, spočívá v přidání nového argumentu protokolech IGMP a MLDV, které se využívají pro připojení respektive odhlášení posluchače od skupiny. Jako příklad jsou zprávy Join a Prune, které teď musí obsahovat na rozdíl od PIM-SM i argument zdroje.

```
>CtrMcast instproc leave-group  { group } {
     …
>SSMMcast instproc leave-group  { group src } {
     …
```

and

```
>CtrMcast instproc join-group  { group } {
     …
>SSMMcast instproc join-group  { group src } {
     …
```

Ve zprávách pak lze ověřit, že zprávy Prune a Join pak pro CtrMCast (SM) obsahují pouze jeden argument (Group), kdežto zprávy pro SSMMCast (SSM) obsahují argumenty dva (Src a Group).

5.2 Instalace

Výše zmíněné informace popisovali knihovny vytvořené pro podporu SSM v Ns2. Ta je k dispozici na webu [5] ve formě balíčku *ssm-ns-extension.tar.gz* . Tento balíček je psán pro Ns-2.27 nebo Ns-2.28.
Aplikace balíčku se provádí až po rozbalení balíčku vlastního simulátoru ns-2.27 (nebo ns-2.28) Balíček obsahuje script s názvem “patch_script.sh” který nakopíruje potřebné soubory tříd do patřičných složek.network simulátoru. Až poté je možno přistoupit k vlastní instalaci simulátoru. Přesný postup a formu příkazů pro instalaci ukazuje nasledující text:

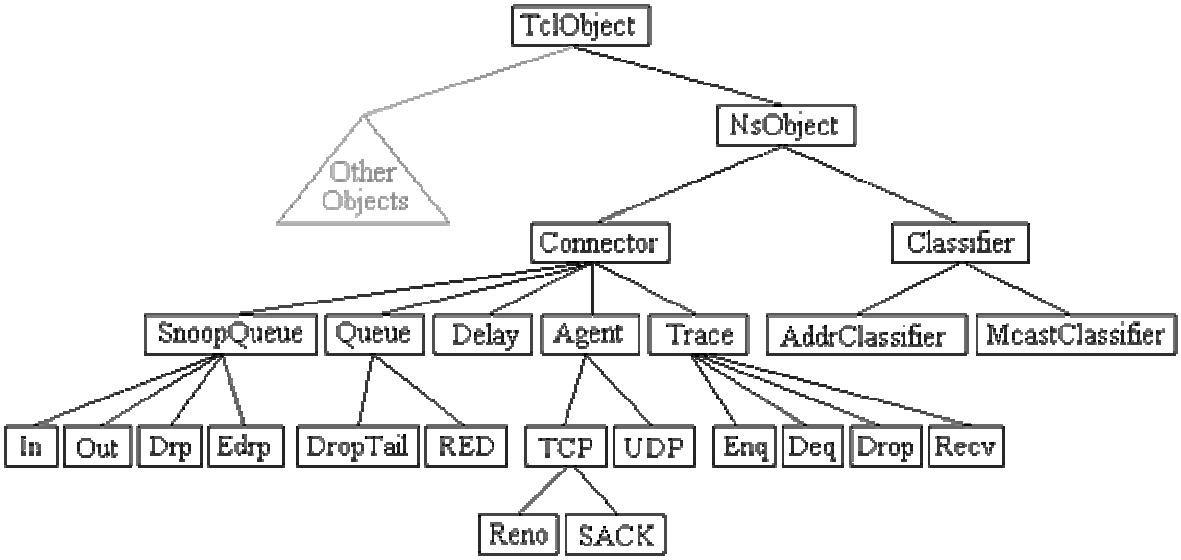
```
tar -xzvf ns-allinone-2.27.tgz [nebo tar -xzvf ns-allinone-
2.28.tgz]
tar -xzvf ssm-ns-extension.tar.gz
cd ssm-ns-extension/
./patch_script.sh
cd ../ns-allinone-2.27/ [nebo ns-allinone-2.28/]
./install
```

6. Princip implementace nového protokolu do ns-2

6.1 Struktura základních objektů

Network simulator byl již obecně popsán v předchozích kapitolách. Jak bylo řečeno, umožňuje simulaci nejrůznějších síťových služeb a protokolů. Ty mohou být už přímo součástí zdrojových kódů Ns-2 nebo si je uživatel může vytvořit dle vlastních představ. Ještě před vytvářením samotného rozšíření je vhodné, seznámit se blíže se strukturou Ns a jejich síťových komponent. Jako je hierarchie tříd, realizace uzlů a možnosti jejich směřování, definice linek mezi uzly a strukturu paketu.

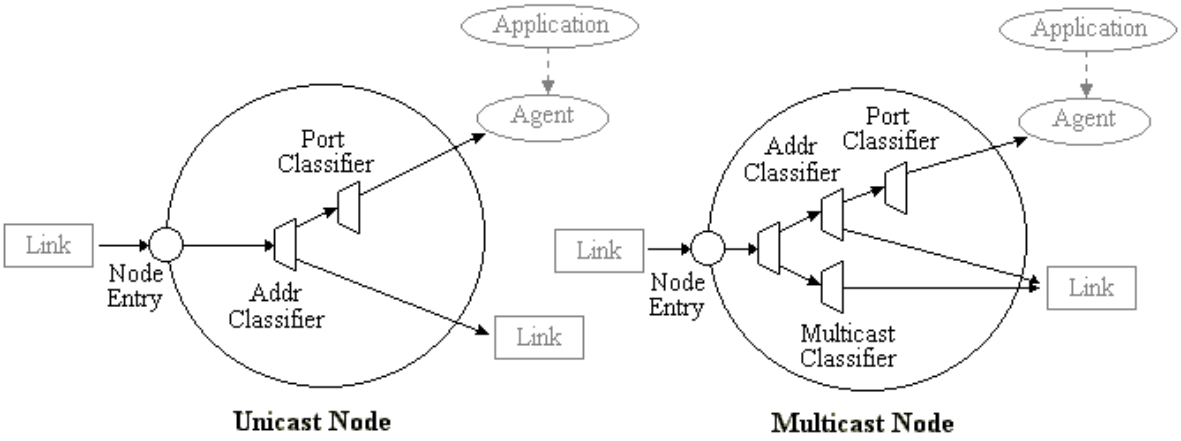
Obr. 6.1 ukazuje částečnou hierarchii tříd. Kořen hierarchického stromu tvoří třída TclObject, která je supertřídou pro všechny ostatních objektů, jako jsou plánovač paketů (používán například pro simulaci zpoždění paketů v síti), síťové komponenty, časovač a další. TclObject je předkem další významné třídy NsObject, která je supertřídou všech síťových komponent, které pracují s pakety. NsObject se pak dělí na dvě třídy podle počtu výstupní cest objektů. Třídu Konektor (Connector) a Klasifikátor (Classifier). Do třídy Konektor spadají základní síťové objekty, které mají pouze jednu výstupní datovou cestu. Do třídy Klasifikátor patří přepínací objekty, které umožňují realizaci více výstupních datových cest.



Obr.6.1: Hierarchická struktura tříd

6.1.1 Node – uzel

Uzel (Node) je jedním ze základních prvků Ns, **obr 6.2**. Je to složený objekt, který je složen ze dvou hlavních částí. Vstupu (Node Entry) a klasifikátoru (adresový a portový). Adresový klasifikátor (Addr Classifiier) určuje je li příchozí paket adresován aktuálnímu uzlu či nikoliv. Portový klasifikátor (Port Classifier) rozlišuje na základě čísla portu, kterému agentu respektive aplikaci daný paket náleží.



Obr.6.2: Struktura uzlu v Ns (unicastový a multicastový)

Existují dva typy uzlů. Unicastový, obsahující výše zmíněné prvky a multicastový, obsahující navíc i klasifikátor multicastové adresy (Multicast Classifier).
Unicastové uzly jsou při simulacích v Ns vždy nastaveny jako výchozí. Chceme li vytvořit multicastové uzly, musíme o tom simulátor informovat a dále můžeme pomocí již známého příkazu *mrtproto* definovat typ použitého protokolu.

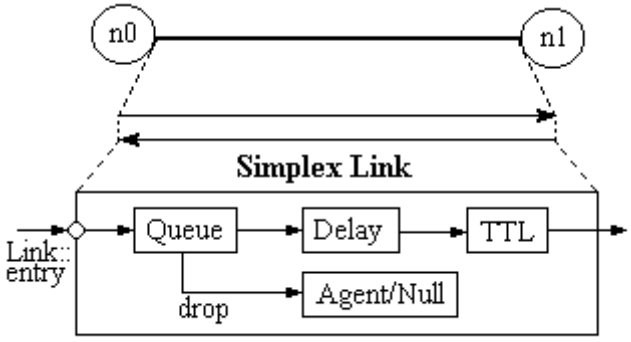
```
set ns [new Simulator -multicast on]
.
.
$ns mrtproto Typ
```

Pro definici unicastového protokolu se používá příkaz *rtproto*.

```
$ns rtproto Typ
```

6.1.2 Link – spoj

Je dalším důležitým složeným objektem v Ns. Ns primárně podporuje simplexní spojení, vytvoří li uživatel mezi dvěma uzly duplexní spojení, je realizováno jako dvě simplexní spojení. **Obr.6.3** ukazuje vnitřní realizaci objektu Link.

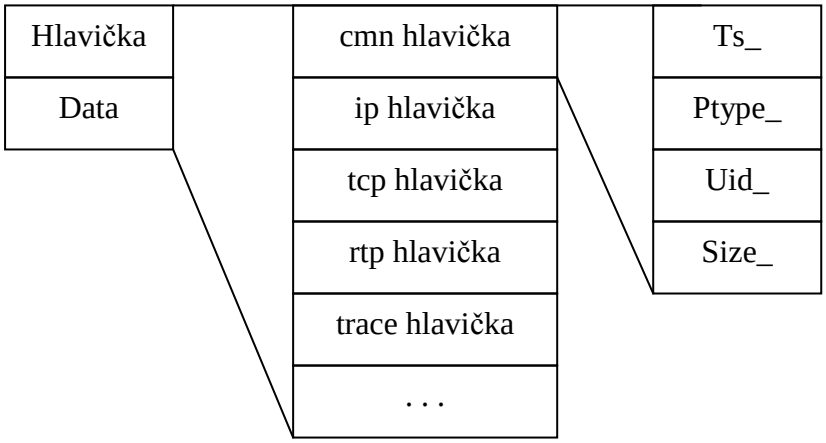


Obr.6.3: Struktura simplexního spoje v Ns

Pakety vstupují do fronty (Queue). Na jejím výstupu je na ně aplikována simulace zpoždění (Delay) a ztráta paketu (Agent/Null). A konečně, blok TTL zjistí, aktualizuje paramert TTL v hlavičce každého průchozího paketu.
Tento jednoduchý model může být doplněn také bloky pro monitorování průchodu paketů pomocí Monitoru fronty (Queue Monitor) a nebo bloky pro vytvoření tzv. trasovacího souboru, definovaného ve vlastním scénáři simulace.

6.1.3 Paket

Paket je asi nejdůležitějším objektem v Ns. Představuje datovou jednotku, díky níž je možné simulovat a sledovat chování datových jednotek v síti. Formát objektu Packet znázorňuje **obr.6.4**.



Obr.6.4: Formát paketu v Ns

Jak je vidět, paket se skládá ze zásobníku registrovaných hlaviček (headers) a nepovinného datového prostoru. Nepovinného z toho důvodu, že při non-real-time simulacích nejsou nějaké data potřeba. Mnoho agentů simulovaných protokolů tento datový prostor vůbec nevyužívá.
Zásobník hlaviček obsahuje všechny dostupné a registrované typy hlaviček. Například společná hlavička (cmn), hlavičku IP, TCP, RTP a nebo také hlavičku pro trasování (trace). Jednotlivé hlavičky v zásobníku mají přesně definovaný a zaznamenaný odstup, a k libovolné hlavičce se lze dostat pomocí odpovídající hodnoty offsetu.
Formát hlavičky paketu je vždy inicializován při vytvoření objektu Simulator, kde je také definován zásobník všech registrovaných hlaviček a pro každou je zaznamenán její offset.
Definici obecného paketu v jazyce C++ je k dispozici v souboru packet.h (a packet.cc) a soubor všech registrovaných hlaviček v souboru packet.tcl.

6.2 Vlastní implementace v Ns-2

Ns umožňuje simulaci již předdefinovaných síťových protokolů a služeb, pomocí agentů (UDP, TCP …) a aplikací (RTP, FTP …). Tím, že je k dispozici s otevřeným zdrojovým kódem, může uživatel upravovat původní nebo dokonce napsat vlastní služby dle svých představ. V následujících odstavcích je popsán princip vytváření nového agenta a aplikace a tudíž vytváření nového protokolu. Popis je pouze ukázkový, podrobné rozebrání jednotlivých řádků kódu je na webu [6].

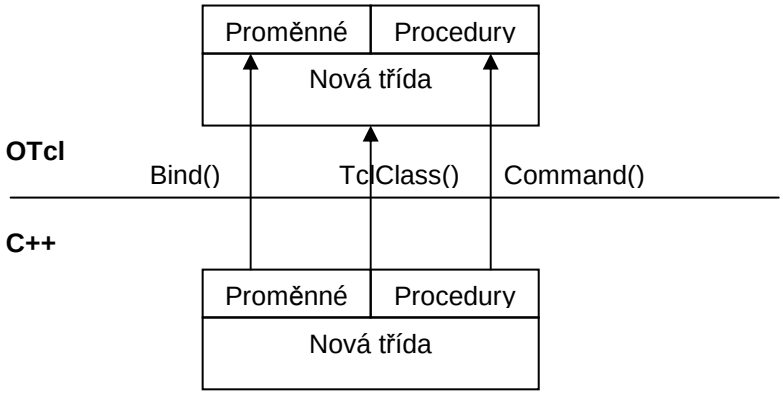
6.2.1 Přidání nového agenta

Agent je základním objektem v Ns. Jako takový vlastně realizuje požadované chování námi simulovaného protokolu, nebo služby. Používají se i aplikace, které jsou z hlediska jejich vytváření téměř shodné. Rozdíl v nich bývá ten, že agent pracuje na transportní vrstvě a tudíž obsahuje deklaraci struktury nového paketu, funkce pro příjem a odesílání paketů. Kdežto aplikace, jak už z názvu vyplívá, představuje vlastní aplikaci, pracující na aplikační vrstvě. Aplikace pak řeší problémy jako, reakce na příchozí paket, jeho zpracování, odesílání nebo třeba časování.
Agent, nebo aplikace, není nic jiného než vytvoření nové třídy, například NovyAgent, která je odvozena od rodičovské třídy Agent. Ve třídě NovyAgent je pak deklarován její konstruktor, funkce command(), viz. dále, a nějaké funkce a proměnné realizující vlastní činnost agenta.

```
class NovyAgent : public Agent {
public:
        NovyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int    promenna1;
        double promenna2;
        void   vypis(void);
};
```

6.2.2 Spojeni C++ / OTcl

Již bylo řečeno, že Ns používá dva programovací jazyky, C++ a OTcl. V jazyce C++ se píší agenti a aplikace. Jsou to součásti, které se v průběhu simulace nemění a vyžadují rychlé zpracování. V jazyce OTcl se píší vlastní simulační scénáře. Aby bylo možné ze simulace přistupovat k ,,céčkovým,, proměnným a funkcím, využívá se tzv. vazba mezi C++ a OTcl. Ta je pro proměnné realizována pomocí funkce *bind()* a pro funkce pomocí funkce *command()*. Vazba se musí vytvořit i mezi novými třídami pomocí funkce *TclClass()*. Jak ukazuje **obr.6.5**.



Obr.6.5: Vazba mezi C++ a OTcl

Následující výřezy zdrojového kódu osvětlují použití zmíněných vazeb..

TclClass()

Aby bylo možné vytvořit instanci agenta přímo ze simulace, musíme je nějakým způsobem svázat. K tomu slouží právě funkce TclClass().

C++:

```
static class NovyAgentClass : public TclClass {
public:
        NovyAgentClass() : TclClass("Agent/NovyAgentOtcl") {}
        TclObject* create(int, const char*const*) {
                return(new NovyAgent());
        }
} class_novy_agent;
```

OTcl:

```
set  novyAgent [new Agent/NovyAgentOtcl]
```

Příklad ukazuje vytvoření vazby mezi třídami v C++ a Otcl a způsob vytvoření nového agenta ve scénáři simulace.

Bind()

Tato funkce slouží pro vytvoření vazby mezi proměnnými. Příklad ukazuje použití funkce bind() a následné nastavení hodnoty proměnné ze simulačního scénáře.

Funkce bind() se používá v konstruktoru třídy agenta, neboť je logické že by měli být přístupné už od vytvoření objektu agenta.

C++:

```
NovyAgent::NovyAgent() : Agent() {
       bind("promenna1_otcl", &promenna1);
       bind("promenna2_otcl", &promenna2);
}
```

OTcl:

```
$novyAgent set promenna1_otcl 12
$novyAgent set promenna2_otcl 3
```

Command()

A konečně funkce command(). Ta vytváří vazbu mezi funkcemi, definovanými ve třídě vytvářeného agenta. Zde konkrétně pro funkci *vypis()* třídy NovyAgent. Vlastní funkce command() je ve třídě NovyAgent rovněž definována jako chráněná (protected).

C++:

```
int NovyAgent::command(int argc, const char*const* argv) {
      if(argc == 2) {
           if(strcmp(argv[1], "vypis_otcl") == 0) {
                  vypis();
                  return(TCL_OK);
           }
      }
      return(Agent::command(argc, argv));
}
```

OTcl:

```
$novyAgent vypis_otcl
```

6.2.3 Před rekompilační úpravy

Příklad prezentovaný výše byl jen jednoduchou ukázkou vytvoření agenta, který vlastně sám o sobě nic nedělal, a vytvoření potřebných vazeb mezi C++ a OTcl. V praxi, kdy potřebujeme do Ns implementovat složitější protokol, jako v tomto případě, je potřeba vytvořit jak nové agenty tak jejich aplikace, ve formě zdrojových souborů. Při implementaci se tak často definují vlastní formáty hlaviček pro používané pakety a další pomocné třídy. Tyto nové objekty se pak musí zaregistrovat do hlavičkových souborů Ns, aby s nimi bylo možné pracovat.
Takový případ i podrobný popis ukazuje příklad v [6], proto jsou v následujícím textu vyjádřeny jen doporučené úkoly, které je nutné provést před vlastní rekompilací Ns a spuštěním simulace.
Před vlastní rekompilací by měly být provedeny následující kroky:

1. První krok předpokládá naprogramované zdrojové soubory pro agenta/y a aplikaci. Ty se píší v programovacím jazyku C++ a jsou typu *.cc a *.h.

Tyto soubory se uloží do složky ns-2, jež obsahuje veškeré zdrojové c++ soubory.

2. Registrace nové hlavičky případně hlaviček, jsou li nějaké nové vytvořeny. Tato registrace se provádí ve dvou souborech, packet.h, což je C++ soubor, a NS-packet.tcl, který je v OTcl.
3. Registrace nových funkcí. Toto je potřeba jen v některých případech. Tyto funkce jsou sice definovány v nové třídě, která je odvozena od rodičovské, ale mohou být volány i třídou rodičovskou. Tehdy by nastal problém při sestavování kódu. Proto je řešením, přidat takové funkce do veřejných v dané rodičovské třídě.
4. Dalším celkem důležitým krokem je nastavení defaultnich hodnot pro proměnné s vazbou pomocí funkce bind(). Tyto proměnné se dají nastavovat přímo ze simulačního scénáře. Pokud je ovšem nezadáme, může dojít k potížím s nedefinovanou hodnotou a simulace může skončit s chybou. Přidání defaultních hodnot do souboru ns-default.tcl pak nastaví proměnné na hodnoty z tohoto souboru.
5. Upravit soubor “Makefile“ podle potřeby, přidání odkazů na soubory *.o

Po provedení všech pěti předchozích kroků je možné přistoupit k rekompilaci Ns pomoci příkazu “make“. Ještě před ní je doporučeno použít “make clean“ a “make depend“.

7. Návrh implementace protokolu

Multiple Data Collection Protocol (MDCP) – protokol hromadného sběru dat.

Tato kapitola je věnována samotnému návrhu implementace výše definovaného protokolu hromadného sběru dat do prostředí network simulátoru. Návrhy se vztahují zejména k realizaci centrální jednotky a měřicích jednotek , návrhu realizace hlaviček dvou nově definovaných paketů a vytvoření časového diagramu komunikace mezi centrální jednotkou a měřicími jednotkou.

7.1 Centrální a měřicí jednotky

V systému navrženého hromadného sběru dat, existují dvě základní jednotky. Jednotka měřicí, dále MU (Measuring Unit) a jednotka centrální, dále CU (Central Unit). Tyto jednotky plní každá rozdílnou úlohu. MU složí pro unicastové zasílání naměřených dat do CU a inicializuje rovněž synchronizaci s CU. Těchto jednotek může být v systému několik, a z funkčního hlediska jsou identické. Jednotka CU ej v dané skupině (multicastové skupině) pouze jedna a má za úkol shromažďovat data od MU a multicastově rozesílat potvrzení přijatých dat. Tato jednotka je z funkčního hlediska od MU zcela odlišná.

Realizace těchto jednotek v Ns 2 bude prostřednictvím dvou agentů a dvou aplikací. Agenti mají především realizovat funkce spojené s příjmem a odesíláním paketů a definici nových paketových hlaviček. Aplikace by pak z hlediska funkce měly odpovídat chování MU a CU.

Pro agenty je možné buď napsat zcela nové zdrojové soubory, nebo využít faktu, že CU je ve funkci obdobná protokolu RTP a MU protokolu RTCP.

7.2 Hlavičky nových paketů

V protokolu MDCP bylo navrženo několik typů zpráv pro komunikaci mezi CU a MU. Tyto zprávy využívají dvou nově definovaných formátů paketů a tudíž i hlaviček, jak ukazují **obr.4.2** a **4.3**.
Oba pakety mají definovánu délku hlavičky 64 bitů. Při pohledu na jejich strukturu vypadají obě dvě zcela identicky. Rozdíl v nich je pouze v pojmenování a významu jedné části hlavičky. Konkrétně 8 bitové pole LOP u formátu UM zpráv a pole Number u formátu potvrzovací zprávy SM konkrétně podtyp SM-CA.

K vlastní realizaci je možné vytvořit buď jednu univerzální hlavičku a rozdílnou položku (Number/LOP) pak identifikovat na základě podtypu dané zprávy (položka Subtype) nebo vytvořit dvě nové hlavičky, které budou jen minimálně rozdílné.

Následující obrázek, **obr.7.1**, ukazuje, realizaci nové hlavičky jako novou strukturu, defakto bitové pole. Hlavičky se definují zpravidla v hlavičkovém souboru vytvářeného agenta (*.h). Například v agentu pro CU. Pro použití této struktury v ostatních agentech (agent MU) a aplikacích je třeba tento hlavičkový soubor přidat, použitím direktivy *#include *.h*.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  subtype  |      TYPE     |         length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ header
| Numer / LOP   | group             |     reserved              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
typedef struct {
     unsigned int Version   : 4;
     unsigned int Subtype   : 6;
     unsigned int TYPE      : 8;
     unsigned int Length    : 14;
     unsigned int Nemer/LOP : 8;
     unsigned int Group     : 10;
     unsigned int           : 14;   //zbylých 14 rezervovaných
bitů

     // Přístupové funkce hlavičky
        static int offset_;
        inline static int& offset() { return offset_; }
        inline static hdr_mdcp* access(const Packet* p) {
                return (hdr_mdcp*) p->access(offset_);
        } hdr_mdcp;
};
```

Obr.7.1: Implementace nové hlavičky

Nově definované hlavičky se pak musí zaregistrovat do souborů packet.h a ns-packet.tcl , jak bylo naznačeno v podkapitole 6.2.3 (předrekompilační úpravy).

7.3 Časový diagram průběhu komunikace

Pro ucelený pohled na komunikaci mezi CU a připojenými MU, ukazuje Příloha A časový digram jejich komunikace. Je zde vidět průběh synchronizace MU, přenosem synchronizačních zpráv (a jejich podtypů), unicastový přenos zpráv s naměřenými daty od MU k CU, v definovaných časových intervalech a jejich multicastové potvrzení. Z důvodu přehlednosti jsou jednotlivé důležité úseky označeny čísly. Tyto úseky jsou pak vysvětleny, viz. níže.

Vysvětlení zkratek

Podrobný popis průběhu spojení je k dispozici v literatuře [4]. Zde jsou vysvětleny některé pojmy.

t_D – doba pro odeslání naměřených dat vypočtená na základě počtu klientů, maximální doby pro odeslání dat a na charakteru sítě. CU tuto informaci posílá MU, která podle ní řídí odesílání svých dat a čekání na potvrzení. $t_D = t_{UM} + t_{SM}$, viz dále.

t_{UM} – 4/5 z doby t_D. Čas MU pro odeslání svých dat k CU.

t_{SM} – 1/5 z doby t_D. Čas kdy MU už nevysílá ale čeká na potvrzení odeslaných dat.

1. Komunikace začíná připojením měřicí jednotky MU-1. Připojení je realizováno formou vzájemné synchronizace obou jednotek. CU předává jednotce MU-1 data potřebné k vlastní komunikaci a přenosu naměřených dat. Jako jsou aktuální počet připojených jednotek, vypočtenou dobu pro odeslání naměřených dat t_d, aktuální čas centrály,

2. Po každém připojení nového člena skupiny, je ještě před spuštěním vlastní doby na odeslání a potvrzení dat, nutné multicastove informovat ostatní členy o změně počtu členů a přepočtu doby t_d , pomocí řídících zpráv SM-CI.

3. Nyní zná stanice, nebo více stanic, potřebné informace a může začít vysílat svá naměřená data centrále. Ta je přijímá a hromadně potvrzuje zprávami SM-CA. Mohou nastat dvě základní situace:
   Počet připojených jednotek MU je malý a tudíž příjde jen malý počet naměřených dat a po skončení doby t_{UM} je odesláno poloprázdné kumulativní potvrzení SM-CA.
   Počet stanic MU ve velký a dojde li k zaplnění kapacity zprávy pro potvrzování (více než 183 potvrzení), je toto potvrzení odesláno a ostatní potvrzení jsou vložena do další zprávy SM-CA. Může se tedy stát, že během jedné doby t_{UM} je odesláno několik plných potvrzení. V čase t_{SM}, pro potvrzování, se pak odešle zbývající potvrzení, i kdyby jich mělo byt jen par nebo jedno.

Pozn.: Po každém cyklu, době t_D, je nutno přepočítat hodnotu t_D a pomocí řídících zpráv SM-CI informovat všechny MU. V případě, že nedojde k žádným vyjímečným situacím, jako přihlášení nové MU, uplynutí doby $9t_D$ (nutno odeslat SM-CI-A) nebo přílišné zahlcování CU přijatými daty, může další cyklus pracovat za stejných podmínek (parametrů) a není tedy nutno přenášet žádný typ řídící zprávy a zatěžovat síť.

4. V síti nedošlo k ničemu, co by si vyžadovalo řídící zprávy, tudíž je spuštěna další doba t_{D2}, která je rovná předchozí.

5. V průběhu druhého cyklu odesílání dat, požádala o připojení nová MU-2. Synchronizace proběhla obdobně jako na začátku u MU-1. Byly jí předány komunikační parametry ale do vysílání se může zapojit až od dalšího cyklu.

6. Po synchronizaci nové jednotky MU, je třeba přepočítat t_D, aktualizovat počet klientů a toto oznámit všem připojeným jednotkám MU, pomocí zpráv SM-CI.

7. Nový odesílací a potvrzovací cyklus t_{D3}, ke kterému se zapojila už i jednotka MU-2.

Další způsob komunikace mezi CU a jednotlivými MU je obdobný, s několika specifickými situacemi:

- V době $9t_D$ – CU posílá řídící zprávu SM-CI-A o své existenci.
- Nedojde li potvrzení do doby $5t_D$ (ztráta potvrzení) – MU žádá re-synchronizaci.
- Neodpovídá li CU, případně nedošla zpráva SM-CI-A ve stanovené lhůtě – MU má 5 pokusů pro re-synchronizaci, jinak přechází do pohotovostního režimu.
- Pohotovostní režim – MU v něm čeká na obnovení činnosti CU, tj. zprávu SM-CI-A.

8. Praktická implementace

Předchozí kapitola byla spíše teoretickou přípravou pro vlastní praktickou implementaci výše zmíněného protokolu do Ns-2. Jak už to tak bývá, i v tomto případě se při praktické části objevily překážky, které v průběhu návrhu nebyly zřejmé.

Ns-2, kam s ním?

Jak už bylo zmíněno v některé z prvních kapitol této práce, Ns-2 je určen pro unixové systémy a tudíž běží buď pod linuxem, což je asi nejlepší řešení, ovšem ne každý má linux jako hlavní operační systém a tak existují i možnosti pro uživatele OS Windows. Zde běží pomocí emulátorů unixu (cygwin) či pomocí virtuálních nástrojů (VMware, Virtual Box atd.) Já jsem zvolil variantu s emulátorem cygwin.

Jak přidávat vlastní zdrojová data

Program cygwin je však pouze jedním nástrojem. Díky němu je možné Ns-2 spustit, pracovat v něm a simulovat nejrůznější skripty již implementovaných protokolů. Nicméně v našem případě potřebujeme přidat nový protokol, respektive, jak už bylo naznačeno v kapitole 8, nové agenty realizující funkci nového protokolu. Vlastní programování agentů probíhá v programovacím jazyce C++ a simulace pomocí skriptů v jazyce OTcl. Nejdříve je nutné opatřit patřičný nástroj pro programování v C++.

Jaké vývojové prostředí?

Bohužel jsem nenalezl vývojové prostředí určené přímo pro Ns-2. Existuje však několik alternativ. V literatuře [1] v sekci "*Developer information*" je popsán postup pro využití programu Eclipse. Obdobně lze nalézt postupy i pro další programy, jako např. Dev C++. A právě v ten jsem zvolil, i když mi nefungoval *debuger*, díky čemuž jsem programoval ,,naslepo" a ověření funkčnosti kódu bylo možné až kompilací v cygwinu, příkazem *make* a spuštěním jednoduchého simulačního skriptu. Jak je zřejmé, pro testování funkčnosti některých funkcí, které budou stručně popsány dále, je tento postup neefektivní. V takových případech jsem pro testy daných funkcí využíval program Borland C++.

8.1 Změny v návrhu

V průběhu vlastního programování agentů došlo také k několika změnám.

Omezeni počtu zdrojových souborů

Především jsem nakonec upustil od návrhu programovat pro CU nebo MU jak agenta tak aplikaci a naprogramoval sem pro každou jednotku pouze agenta. Ve výsledku to vůbec není na škodu. Funkční celek měřicí jednotky nebo centrály je pak v kupě v jednom souboru a při simulaci není třeba k uzlu připojovat jak agenta tak ještě aplikaci.

V konečném důsledku existují soubory mdcp.h, CU.cc a MU.cc, kde mdcp.h je hlavičkový soubor, společný pro oba typy jednotek a obsahuje definici struktury paketové hlavičky protokolu hromadného sběru dat (MDCP). A dále soubory CU.cc a MU.cc obsahují zdrojový kód jednotlivých jednotek.

Vynechání synchronizační zprávy MU-SY-P

Tato zpráva měla v původně zamýšleném návrhu protokolu předávat jednotce MU potřebná provozní data. Zejména počet klientů přihlášených do multicastové skupiny, dobu td, aktuální čas centrály atd. Tato zpráva je ovšem z praktického hlediska zbytečná, neboť po ukončení synchronizace jednotka čeká na řídící zprávu SM-CI, která nese tytéž informace.

Sjednocení zpráv SM-CI

Dalším snížením provozní zátěže je sjednocení čtyř řídících zpráv do jedné. Ta je označena jako SM-CI a její hodnoty *subtype* je 0. Namísto čtyř zpráv o velikosti 4B , je odesílána jedna 12B. I slabý matematik může namítnout že 4x4B není 12B. Ovšem původní zpráva SM-CI-A nenese žádnou informaci, ale je to jen hlavička s parametrem *subtype* = A. Nově definovaná, společná, řídící zpráva SM-CI pak obsahuje pouze jednu hlavičku a data z původních zpráv SM-CI-T , SM-CI-CN a SM-CI-MT. Příznak zprávy SM-CI-A je pak vyřešen nastavením položky *subtype* v hlavičce paketu na “A”.

Pozn.: Řídící zpráva je centrální jednotkou vysílána pouze v případě vyjímečné situace (např. přihlášení nového klienta, změna td v důsledku velkého zpoždění v síti atd.) a jednou za dobu 9td a to s příznakem “A”.

8.2 Popis jednotek

V následujících odstavcích budou stručně popsány jednotliví agenti a některé specifické funkce v nich použité. Nebude zde uváděn jejich rozsáhlý zdrojový kód, neboť ten je k dispozici okomentovaný ve zdrojových soubore *.cc na přiloženém CD.

Ve vlastním programování jsem vycházel z již implementovaných agentů v ns-2. Především se jednalo o agenty protokolu RTP, UDP a CBR což je aplikace pro odesílání paketů s nastavenou konstantní přenosovou rychlostí. Tito a vesměs všichni agenti implementovaní v Ns-2 mají obdobnou strukturu. Je to dáno tím, že simulátor je značně rozsáhlý, obsahuje množství zdrojových a hlavičkových souboru, které jsou mezi sebou provázány. Existuje zde však rodičovská hierarchie, kdy jednotliví agenti jsou odvozeni od nějakého rodičovského základu. Celou strukturu a provázanost souborů je možné projít na webu v [10].

Rodičovský základ pro agenty představují soubory Agent.cc a Agent.h. Samozřejmě že i tyto soubory jsou potomky nadřazenějších tříd. V těchto souborech jsou definovány funkce, které tvoří kostru všech agentů. Vlastní tělo jednotlivých funkcí může obsahovat libovolný zdrojový kód, dle požadavků programátora. Také není nezbytné aby ten či onen agent obsahoval všechny následující funkce. Mezi základ používaných funkcí patří:

`JmenoAgenta();`

Jedná se o konstruktor daného agenta. Jeho kód je prováděn při inicializaci nové instance, tzn. při vytvoření nového agenta v simulačním skriptu.

Většinou se v něm inicializují proměnné, potřebné již od počátku běhu agenta. Například typ paketu, který bude daný agent využívat, nebo k inicializaci časovače, je-li využíván.

```
virtual void Start();
virtual void Stop();
```
Tyto funkce se používají především ve spojení s funkcí command(). Představují tedy úkoly které se musí provést po spuštění nebo ukončení běhu agenta.

Samozřejmě je možné je volat kdykoliv za běhu programu, je-li to nutné.

```
void  Sendpkt()
{
1:   Packet* p = allocpkt();
2:   hdr_mdcp* mh = hdr_mdcp::access(p);
3:   hdr_cmn *cmnh = hdr_cmn::access(p);
4:   PacketData* data = new PacketData(velikost_dat);

// tělo funkce

5:   p->setdata(data);
6:   target_->recv(p);
}
```

Jedná se o jednu z nejpoužívanějších funkcí v agentu. Jak už název napovídá, jedná se o funkci pro vlastní odeslání paketu. Zde je vhodné podotknout, že pakety lze odesílat buď bez dat, tzn. pouze hlavičky, nebo s připojenými daty, což je tento případ. Přenos paketů jen s hlavičkovými informacemi je nejčastější případ, který již implementovaní agenti, reprezentující různé protokoly, využívají. Přenos dat je vyjímečná záležitost.

U této funkce je uvedena i její základní struktura. Význam jednotlivých řádků je následující:

1: vytvoření nového paketu a jeho inicializace pomocí funkce *allocpkt()*. Tato funkce je definována v rodičovské třídě Agent v souboru Agent.h . Vyhradí paměťové místo pro nový paket a vyplní základní hlavičky (hlavičku IP a CMN) defaultními informacemi.

2: ukazuje vytvoření ukazatele na hlavičku protokolu hromadného sběru dat, náležící výše definovanému paketu. Umožňuje pak editaci dané hlavičky.

3: obdoba řádku 2. Zde se jedná o ukazatel na společnou hlavičku. Řádky 2 a 3 nejsou ve funkci *Sendpkt()* povinné, pokud nepotřebujeme nebo nechceme editovat hodnoty hlaviček.

4: vytvoření ukazatele na datový typ *PacketData* a dynamická alokace paměti, pomocí operátoru *new*.

5: předání ukazatele dat paketu.

6: vlastní odeslání paketu.

```
virtual void  timeout(int);
```

Tato funkce obsahuje kód, který se vykoná po vypršení doby vyměřené časovačem.

```
virtual void  recv(Packet* paket, Handler*)
{
1:   hdr_ip *iph = hdr_ip::access(paket);
2:   PacketData* data = (PacketData*)paket->userdata();
```

```
// tělo funkce

3:   Packet::free(paket);
}
```

Funkce pro příjem paketu. Je automaticky volána vždy, když danému agentu příjde jakýkoliv paket. Opět je zde uvedena základní struktura funkce.

1: vytvoření ukazatele na ip hlavičku. Lze využít pro zjištění směrovacích údajů paketu.

2: ukazatel na paketová data, kterému je následně přiřazena adresa datové části přijatého paketu.

3: volání funkce pro uvolnění paketu z paměti.

```
int  command(int argc, const char*const* argv);
```

Tato funkce byla představena již dříve v kapitole 6.2.2. Umožňuje spuštění ,,céčkových,, funkcí ze simulačního skriptu. Nejpoužívanějšími funkcemi jsou zpravidla start() a stop().

Mimo výše zmíněných funkcí samozřejmě musí agent obsahovat již zmíněné funkce jako *bind()* a *TclClass()*. Poslední zmíněná funkce spolu s konstruktorem jsou jediné, která jsou pro chod agenta bezpodmínečně nutné.

Dalším důležitým souborem nebo lze také říci protokolem, je IP.h. Jedná se o protokol síťové vrstvy a definuje proměnné a hlavičku jenž jsou potřebné pro směrování. Příklad využití IP protokolu je například v centrální jednotce, která musí umět odesílat pakety jak na všesměrovou adresu tak jednotlivým měřicím jednotkám.

Následující podkapitoly se budou zabývat realizací měřicí a centrální jednotky. U každé bude stručný popis funkce jednotky a některých funkcí v ní použité. Pro lepší představu o chování složitějších funkcí budou doplněny jednoduchým vývojovým diagramem.

8.2.1 Měřicí jednotka – agent MU

Měřicí jednotka, představuje vlastního klienta, který se pomocí synchronizačních zpráv přihlašuje centrále. Ta jej po úspěšné synchronizaci přidá do multicastové skupiny. MU se tak po obdržení řídící zprávy SM-CI může zapojit do relace. Ta sestává z odesílání naměřených dat pomocí zprávy UM-MV v daných časových intervalech, daných dobou td, a očekávání zprávy kumulativního potvrzování. Pro odpočet času má jednotka MU definovány dva časovače. Jeden měří dobu td, respektive tum a tsm a druhý slouží pro odpočet náhodné doby pro vlastní odeslání dat, tzn. aby například při výpadku centrály se všichni klienti nezačali resynchronizovat ve stejnou dobu.

MU obsahuje několik funkcí, které budou dále charakterizovány a řadu proměnných, které tyto funkce využívají. Jejich popis je uveden ve zdrojovém souboru MU.cc na přiloženém CD. Pro lepší orientaci v několika následujících diagramech uvádím popis nejdůležitějších proměnných.

double ack[5] – pole o pěti prvcích do něhož se při každém odeslání zprávy UM-MV zapíše její čas a při nalezení potvrzení je patřičná hodnota vymazána. Zajišťuje se pomocí ní nutnost potvrzení zprávy do doby 5td.

Short CU_ready – při startu je nastavena na hodnotu 0. Po každém cyklu td je inkrementována a po přijetí řídící zprávy s podtypem ‘A’ je nulována.

Funkce Start()

Spouští se jako první funkce při rozběhnutí agenta MU a je také volána během vlastního běhu programu. Například pro rozběhnutí dalšího cyklu, vždy po skončení doby tum . Její prioritní funkcí je nastavení obou časovačů. První (hlavní) časovač je nastaven na dobu 4/5 td, druhý (odesílací) časovač je nastaven na náhodný zlomek tc doby hlavního časovače dle vztahu (2) navrženého v [4].

$$t_{C_i} = \frac{\frac{4}{5} t_D}{CN} \cdot R \,, \tag{2}$$

kde td je celková doba jednoho cyklu, CN je aktuální počet klientů dané multicastové relace a R je náhodně generované číslo z rozsahu $R \in \langle 1; CN \rangle$.

Pozn: doba td a CN je defaultně nastavena a to td na hodnotu 5s a CN na hodnotu 1. Aktualizuí se po synchronizaci přijetím řídící zprávy SM-CI.

Funkce Stop()

Opačná obdoba předešlé funkce. Má za úkol zastavit oba časovače.

Funkce Send_timeout(int)

Je volána vždy po přetečení odesílacího časovače. Volá funkci *sendpkt(…)* pro odeslání příslušného paketu. Funkce může být volána ve dvou různých stavech jednotky MU. Prvním je její volání při startu, tedy při odeslání synchronizační zprávy UM-SY-I. Druhý stav představuje klasický běh jednotky MU, kdy je volána pro odesílání zpráv s naměřenými daty UM-MV.

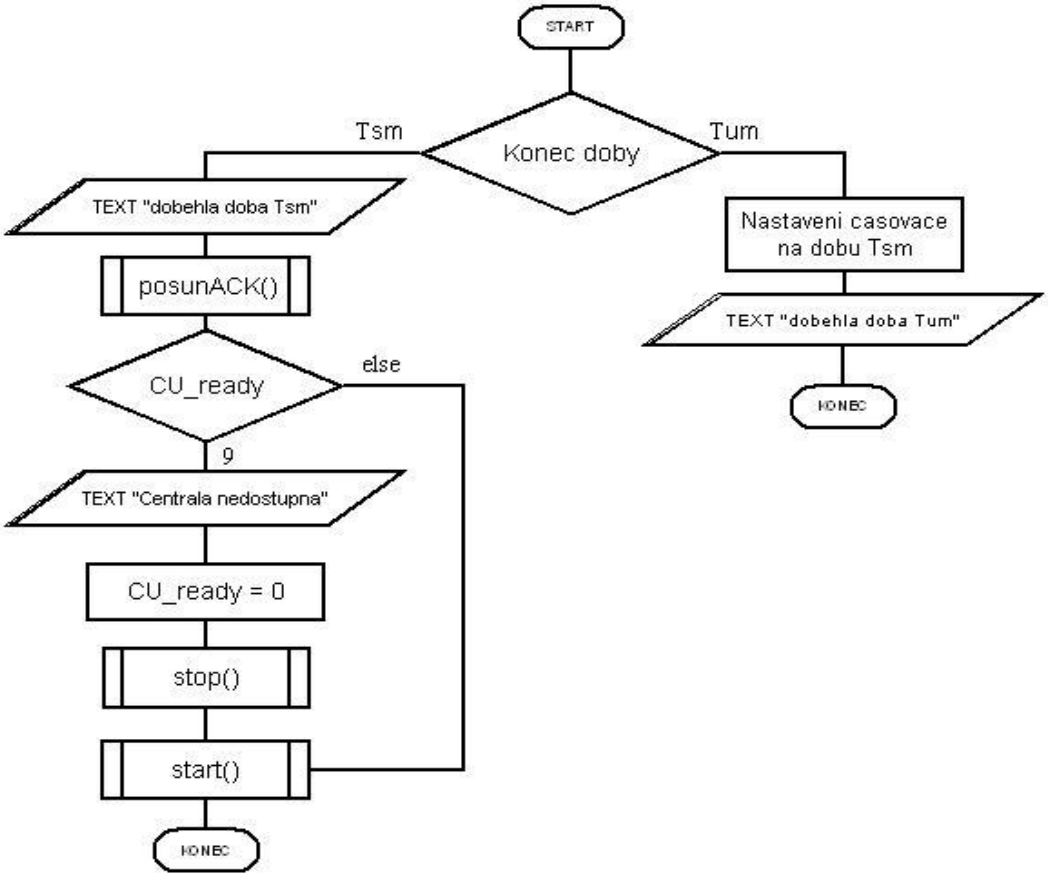
Funkce timeout(int)

Tato funkce je podobná předchozí. Je volána při přetečení hlavního časovače. Její funkci ukazuje diagram na obr.8.1.

Funkce Sendpkt(unsigned int _type, unsigned int _subtype)

Funkce sloužící pro odeslání paketu. Má definovány dva vstupní parametry, pomocí nichž se nastaví odeslání patřičného paketu. MU odesílá jen dva typy paketu, měřicí a synchronizační a ze simulačního hlediska, kdy měřicí paket nenese žádné skutečně naměřené data, ale jen informaci o svém jménu a čase, jsou oba pakety rozdílné pouze v hlavičce.

Pozn: Protože se jedná pouze o simulaci, vyplňuje tato funkce jen potřebné hlavičkové informace. *Type* a *subtype*.



Obr.8.1.: Vývojový diagram funkce timeout(int)

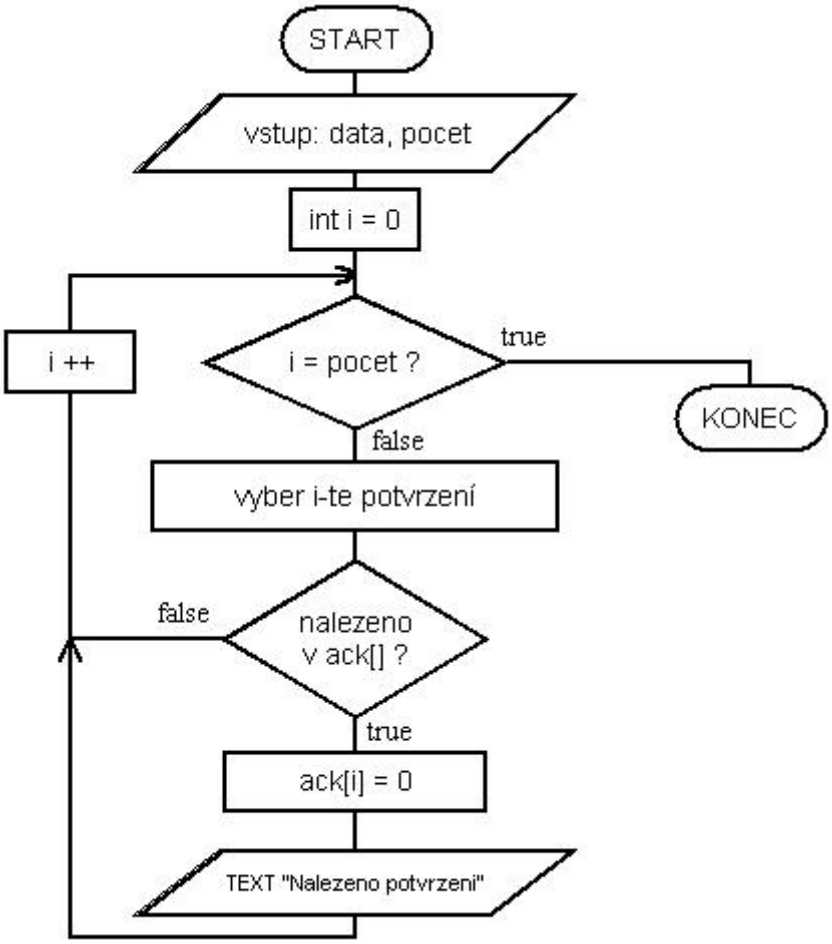
Funkce recv(Packet* p, Handler* h)

Tato funkce, jak už bylo zmíněno výše, zpracovává přijatý paket. Způsob zpracování je závislý na typu a podtypu zprávy, který je vybrán z hlavičky paketu. Na synchronizační zprávy se reaguje jednoduše voláním funkce *sendpkt(…)* s parametry podle synchronizačního schématu. Z řídící zprávy jsou získána provozní data jako td, počet klientů a aktuální čas centrály. A konečně data zprávy kumulativního potvrzení jsou spolu s počtem potvrzení předány funkci *zpracujCA(…)*.

Funkce zpracujCA(unsigned char* data, unsigned int pocet)

Funkce pro zpracování kumulativních potvrzení. Je volána z funkce *recv(…)* při příjmu zprávy SM-CA. Ta z ní ,,vytáhne,, soubor potvrzení a ukazatel na ně předá funkci jako první parametr. Strukturu potvrzení v datech paketu ukazuje obr.4.3.

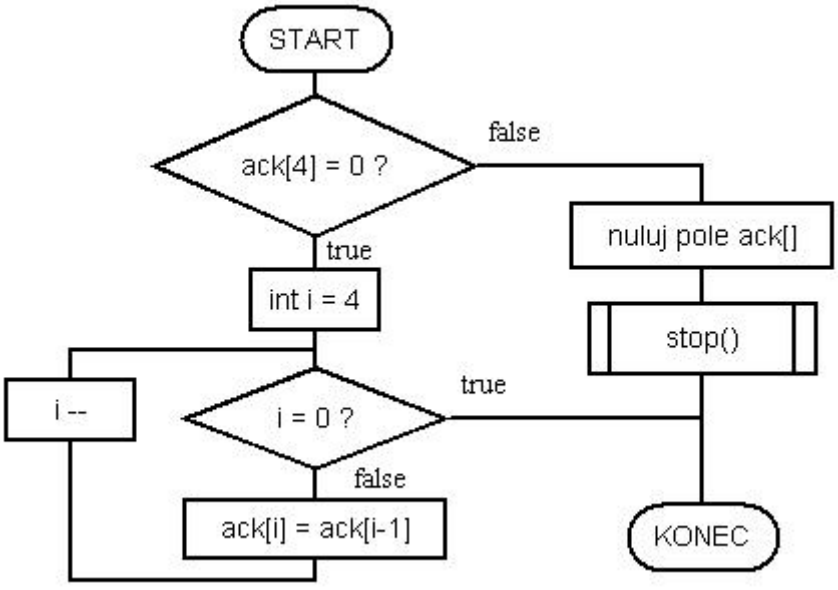
Druhým parametrem je počet potvrzení ve zprávě, získaný z hlavičky paketu SM-CA, konkrétně z položky *Member*. Pomocí cyklu pak prohledává jednotlivá potvrzení a nalezne li potvrzení se shodným jménem, porovná položky pole *ack[]* s hodnotou *timestamp*. V případě shody je potvrzení nalezeno a položka v *ack[]* nulována (kvůli volání funkce *posunACK()* na konci tsm) Vývojový diagram ukazuje obr.8.2.



Obr.8.2.: Vývojový diagram funkce zpracujCA(unsigned char data, unsigned int pocet)*

Funkce posunACK()

Jedná se o poslední funkci jednotky MU. Pracuje s polem ack[], kdy po každém cyklu jsou jednotlivé položky pole posunuty na následující hodnotu. Zároveň se kontroluje zda li už některá zpráva nedosáhla posledního prvku. Pokud ano, došlo k situaci nedoručeného potvrzení. Diagram funkce ukazuje obr.8.3.



Obr.8.3.: Vývojový diagram funkce posunACK()

8.2.2 Centrální jednotka – agent CU

Centrální jednotka je hlavní funkční jednotkou navrženého protokolu hromadného sběru dat. Má za úkol provést synchronizaci s MU, inicializovanou ze strany MU. Dále připojit klienty do multicastové skupiny a v definovaných časových intervalech přijímat naměřená data a pomocí zprávy kumulativního potvrzení tato data potvrzovat. Popis funkce CU zde už byl několikrát zmíněn, nicméně konkrétní řešení dílčích problémů je popsáno v následujícím textu. Nejdříve bude popsán princip funkce jednotky CU a následně jednotlivé použité funkce.

Start centrály je provázen nastavením časovače na hodnotu td, která v tomto případě reprezentuje synchronizační dobu po kterou CU vyčkává na připojení klientů. Po skončení této doby, za předpokladu připojení alespoň jednoho klienta, je časovač přednastaven na dobu 4/5 td a začíná doba tum, následně tsm a tak dokola.

Pozn: čas td je obdobně jako u MU nastaven na defaultní hodnotu 5s a je možné jej nastavit přímo ze simulačního skriptu.

V době tum CU přijímá zprávy UM-MV a přidává je do seznamu. V případě překročení 183 potvrzení je generována a odeslána zpráva kumulativního potvrzení SM-CA. V následné době tsm se odesílají zbylá potvrzení a řídící zpráva SM-CI.

Zdrojový kód CU stejně jako MU obsahuje několik funkcí a proměnných, zajišťující chod centrály. Jejich popis obsahuje přímo zdrojový soubor CU.cc. Pro lepší představu o jejich funkci, především o způsobu vytváření kumulativních potvrzení a přidávání, jsou v následujícím textu vývojové diagramy a popis některých funkcí.

Funkce start()

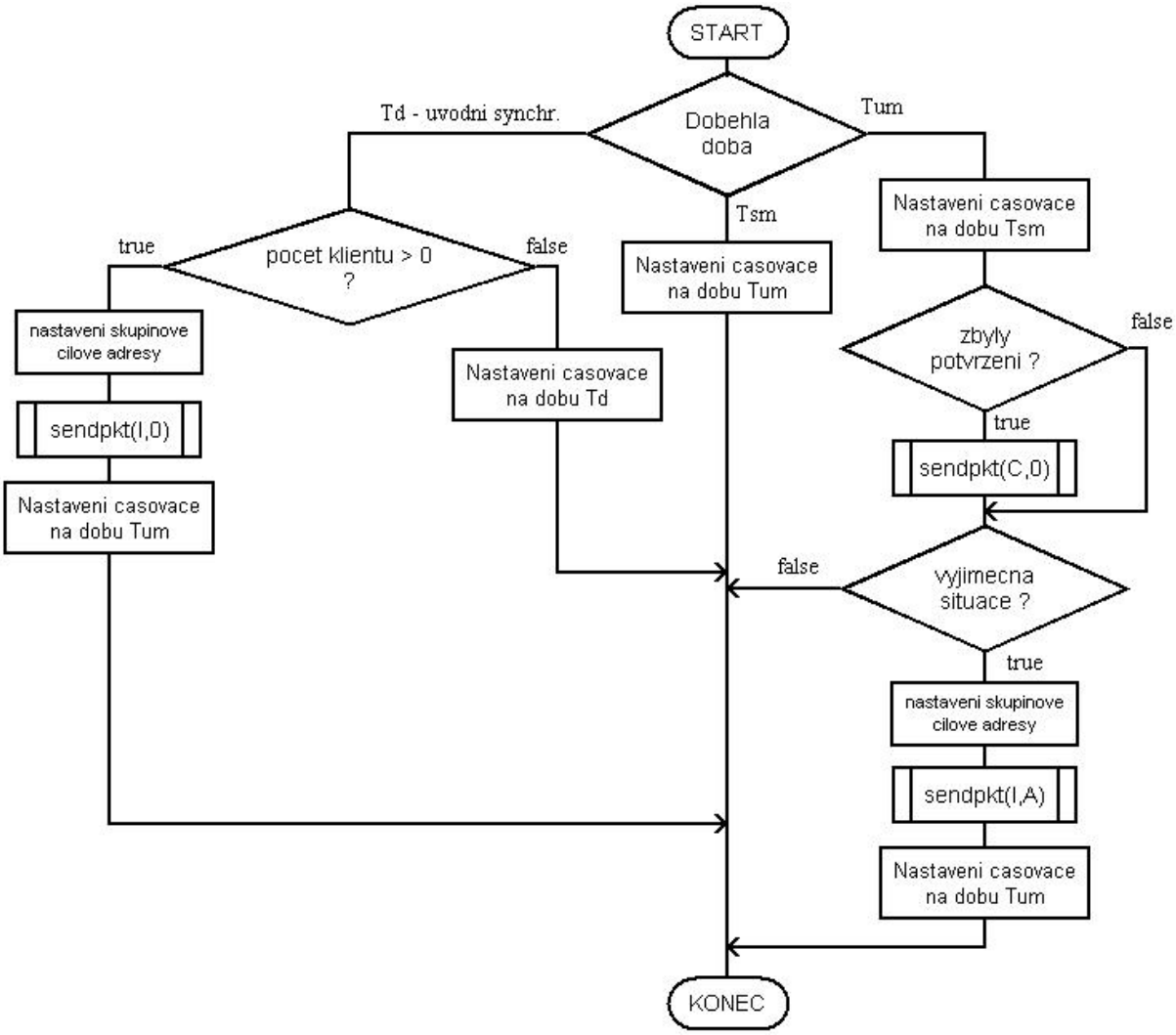
Jednoduchá funkce spouštěná při startu centrály. Spouští časovač na dobu td a nastavuje proměnné na úvodní hodnoty. Například počet klientů na 0.

Funkce stop()

Ukončuje činnost časovače.

Funkce timeout(int)

Funkce spouštěná při přetečení časovače. Její princip je patrný z vývojového diagramu na obr.8.4.



Obr.8.4: Vývojový diagram funkce timeout(int)

Funkce recv(Packet* paket, Handler*)

Stejná funkce jako u MU. Zjednodušeně řečeno. Zpracovává všechny přijaté pakety, z každého vyjme data (hodnotu *Name* což odpovídá jménu klienta a *Timestamp* což je časové razítko odeslání této zprávy) a z hlavičky typ příchozího paketu. Na jeho základě je paket zpracován. Zpráva typu M (UM-MV) je zpracována přidáním do seznamu kumulativního potvrzení, viz. funkce *pridej (…)*. Na synchronizační zprávy se reaguje dle podtypu zprávy, odesláním příslušné odpovědní synchronizační zprávy.

Synchronizační zprávy se posílají unicastově jednotlivým klientům, kdežto ostatní zprávy se odesílají na multicastovou adresu. Proto se u CU před odesláním paketu musí upřesnit směrovací údaje. Ty se nastavují pomocí předdefinované funkce agenta *daddr()* případně *dport()*. První uvedená funkce představuje aktuální cílovou adresu agenta, druhá cílový port. Toho se využívá v případě, že je na jednom uzlu připojeno více agentů. Pro multicastové odesílání se jednoduše zmíněné funkci nastaví hodnota skupinové adresy. Pro unicast se adresa klienta vytáhne z ip hlavičky přijatého paketu.

Příklad nastavení cílové adresy na hodnotu adresy zdrojové z ip hlavičky paketu.

```
daddr() = iph->saddr();
```

Důležitým úkolem této funkce je přidání klienta do multicastové skupiny. Provádí se to voláním tcl funkce přímo ze zdrojového kódu, za pomocí speciální funkce *tcl.eval()* viz. obr.8.5. Tcl funkce pak musí být definována s samotném simulačním skriptu jak ukazuje obr.8.6.
Pozn: Obr.8.5. odpovídá bloku „*Pridani klienta do multicastove sklupiny*“ ve vývojovém diagramu obr.8.7.

```
1:  char uzel[5];
2:  uzel[0] = '_';
3:  uzel[1] = 'o';
4:  sprintf(uzel+2,"%d",Name);
5:  char out[100];
6:  sprintf(out, "%s joinToGroup %d",uzel,groupAddr,Name());
7:  Tcl& tcl = Tcl::instance();
8:  tcl.eval(out);
```

Obr.8.5.: Část zdrojového kódu funkce recv() v souboru CU.cc, ukazující princip připojení klienta k multicastové skupině.

Add. Obr.8.5. Předem je vhodné říct, jak funkce tcl.eval() funguje. Funkce přebírá vstupní parametry pomocí pole znaků. Hodnoty v tomto poli jsou uspořádány dle vzorce:

(JménoAgenta NázevFunkce ParamertyFunkce).

Jméno agenta je ns-2 definováno ve tvaru *o_číslo* (např. o_154). Paket však z důvodu snadnější implementace přenáší pouze číslo z tohoto jména, proto řádky kódu 1-4 rekonstruují jméno do správného tvaru. Název funkce je v tomto případě *joinToGroup* a parametry funkce jsou *groupAddr*, což je globální proměnná CU obsahující multicastovou adresu a Name(), což je funkce vracející jméno CU agenta. Řádky 5 a 6 vytváří výše zmíněné pole, které je pak použito jako parametr funkce *tcl.eval()* jak ukazuje řádek 8.

```
Agent/MU instproc joinToGroup {group src} {
          $self instvar node_           //získání uzlu MU
          set uzel $node_
          $src instvar node_            //získání uzlu CU
          $uzel join-group $self $group $node_
          }
```

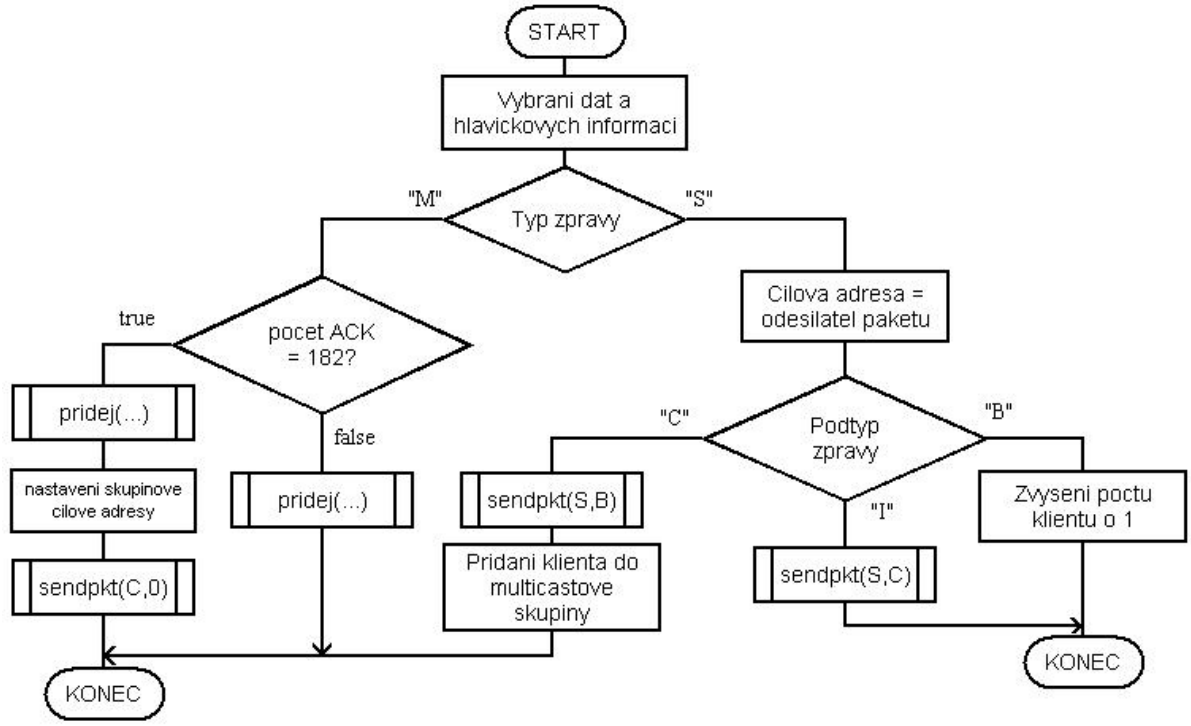
Obr.8.6.: Část kódu ze skriptu CUMU.tcl, ukazuje definici Tcl funkce pro připojení agenta do skupiny.

Obr.8.6 ukazuje zápis funkce v simulačním skriptu. Jedná se vlastně jen o funkci volající tcl funkci Join-group. Její syntaxe je:

Uzel_MU ***join-group*** *jménoAgentaMU skupinováAdresa uzelCU*

kde *Uzel_MU* představuje číslo uzlu připojovaného MU agenta, *jménoAgentaMU* pak jeho jméno, *skupinovaAdresa* je jasná a *uzelCU* představuje jméno uzlu zdroje mcastového vysílání, tedy uzlu s připojeným agentem CU. Poslední parametr ve funkci Join-group se používá pouze u SSM.

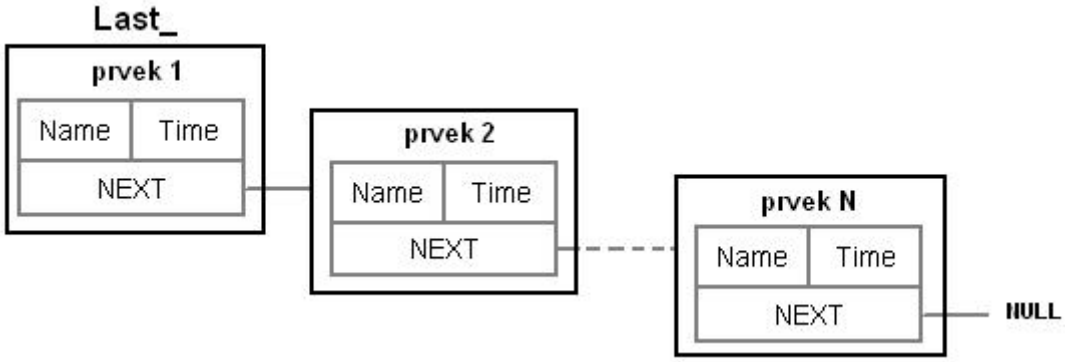
O této funkci jsem se rozepsal trochu více ale způsob připojení klientů do multicastové skupiny pokládám za stěžejní část. Obr.8.7. pak ještě ukazuje vývojový diagram celé funkce *recv(…).*



Obr.8.7.: Vývojový diagram funkce recv(Packe p, Handler* h)*

Funkce pridej(data **last_, unsigned int name_, double timestamp_)

Tato funkce slouží pro přidávání hodnot k celkovému seznamu kumulativních potvrzení. Základem je jednosměrný lineární seznam, reprezentovaný nově definovanou třídou *data*. Třída obsahuje proměnné *name_* a *timestamp_* , které představují jeden blok potvrzení a proměnnou (ukazatel) *next* typu *data*, která ukazuje na další instanci třídy *data*. Tento poněkud kostrbatý popis lépe vystihuje obr.8.8.



Obr.8.8.:Struktura seznamu potvrzení

Prvek třídy je vytvořen pomocí konstruktoru, obsahující dva vstupní parametry (jméno a časové razítko). Funkce pro mazání není implementována, jednotlivé prvky jsou mazány pomocí operátoru *delete* ve funkci *createData(…)*. Třída obsahuje ještě destruktor a funkci *number_ACK()* , která vrací počet položek seznamu.

Samotná funkce *pridej(…)* pak má tři vstupní parametry. Prvním je ukazatel na ukazatel proměnné *last_*, která představuje základní prvek seznamu,

definovaný jako globální proměnnou v agentu CU. Další dva parametry jsou předány konstruktoru třídy *data*, tím je vytvořen nový prvek seznamu. Konkrétní řešení je přímo v souboru CU.cc.

Funkce createData(unsigned char *_data, unsigned int *_pocetACK, data **last_)

Tato funkce vytváří ze seznamu potvrzení, představeného výše, paketová data vhodná pro odeslání. Bývá volána z funkce *sendpkt(…)* pro zprávu s typem C. Jako vstupní parametry jsou jí předány:

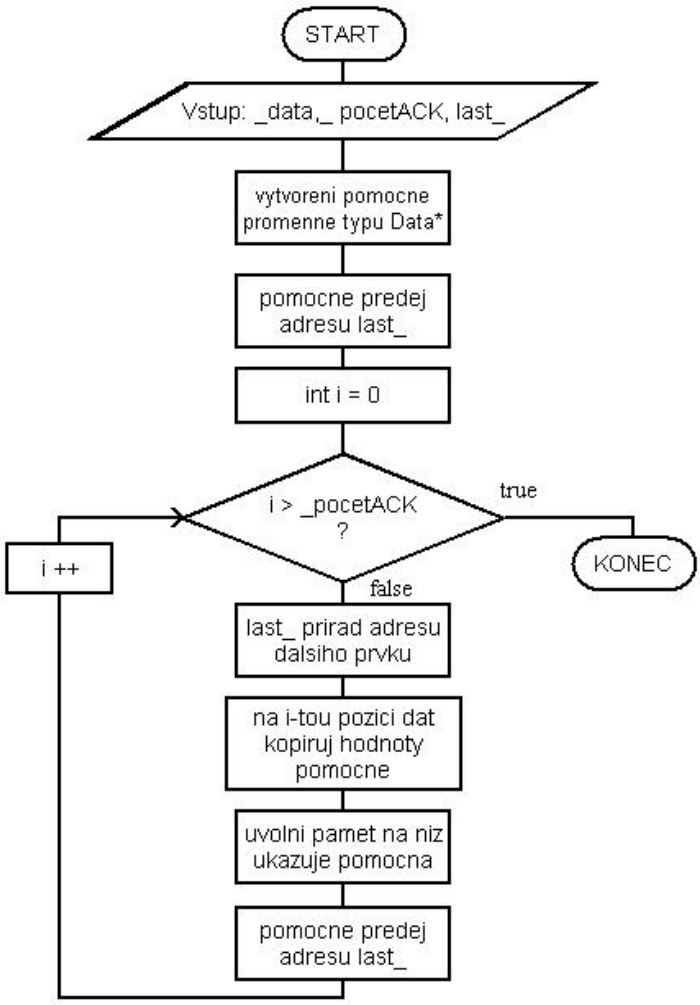
- ukazatel na paketová data (PacketData jsou defakto typ unsigned char*) , jinak řečeno vyhrazené místo v paměti kam budou jednotlivé prvky seznamu předány.
- počet prvků v seznamu
- a ukazatel na poslední prvek seznamu

Pozn: název proměnné *last_* je trochu matoucí. Jedná se o jednosměrný seznam a prvek s označením *last_* zde ve skutečnosti představuje první prvek seznamu, z hlediska pořadí (v pořadí poslední prvek má hodnotu Next = null) ale poslední prvek z hlediska jeho přidání. Označení prvku *last_* ukazuje obr.8.8.

Principielní schéma funkce ukazuje diagram na obr.8.9.

Funkce sendpkt(unsigned int _type, unsigned int _subtype)

Funkce je obdobná jako u jednotky MU. Opět má dva vstupní parametry definující typ odesílané zprávy. CU však musí odesílat tři druhy zpráv z nichž každá obsahuje svá specifická data. V jednoduchosti. Zpráva SM-CA, tedy kumulativní potvrzení, data pro paket vytváří výše zmíněná funkce *createData(…)*. Synchronizační zprávy nenesou žádné data ale tato možnost zde je dostupná okomentováním příslušných řádků kódu. A konečně řídící zpráva SM-CI nese jako data informace o aktuálním čase, době td a počtu klientů.



Obr.8.9.: Vývojový diagram funkce createData(…).

9. Simulace

V předchozí kapitole byla stručně naznačena funkce jednotlivých jednotek a jejich realizace. Tato kapitola by měla pomocí simulačních scénářů ověřit funkčnost naprogramovaných agentů a odhalit případné nedostatky buď v návrhu nebo vlastní realizaci navrženého protokolu.

9.1 Ověřeni funkce

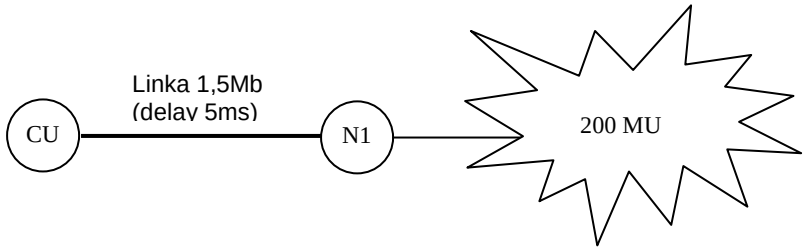
Dostali jsme se do bodu, kdy máme vytvořeny nové agenty. Tito byly dle pokynů podkapitoly 6.2.3 zkompilovány do Ns-2. Nyní stačí vytvořit simulační skript ověřující, zda se chování naprogramovaných agentů shoduje s teoretickými předpoklady.

Simulační skripty se píší za pomoci jazyka *Otcl*, jak už bylo zmíněno v některé z prvních kapitol této práce. Skript pro ověření funkce , se jménem *basic.tcl*, je okomentovaný k dispozici na přiloženém CD. Skript představuje topologii jedné jednotky CU a 5 jednotek MU. Jedná se pouze o jednoduchou simulaci s minimálním počtem jednotek aby byla zřejmá funkčnost agentů. Pro lepší představu a také doložení, že daná implementace funguje, ukazuje Příloha B výpis simulátoru. Výpis je omezen pouze na dobu úvodní synchronizace a jeden cyklus doby t_d (t_{um} + t_{sm}). Je to především z hlediska jeho rozsáhlosti.

Popis přílohy B:
- Výše zmíněný výpis začíná startem jednotek MU v čase 0,2s. Doba t_d je defaultně nastavena na 5s, tzn. že následuje pětivteřinová synchronizační doba, v níž se mohou jednotky MU hlásit centrále.
- Jednotky MU se přihlašují. Jsou zobrazeny odesílací časy, centrála informuje o připojení klienta do multicastové skupiny a navyšuje se počet klientů této skupiny. Z důvodu menšího rozsahu byly při simulaci omezeny synchronizační zprávy. MU se přihlásí zprávou UM-SY-C a centrála odpoví SM-SY-B.
- V čase 5.2s končí úvodní synchronizace. CU odesílá řídící zprávu a po příjmu všem začíná doba t_{um}.
- Jednotky MU odesílají měřicí zprávy centrále Ta má nastaven maximální počet potvrzení ve zprávě SM-CA na 3, tzn. že po příjmu třetí zprávy UM-MV centrála odesílá kumulativní potvrzení, kterým jednotkám potvrdí jejich naměřené údaje.
- Do konce t_{um} jsou ještě přijaty dvě měřicí zprávy od zbylých jednotek MU
- V čase 9,2s končí doba t_{um} a nastává t_{sm}. CU odesílá zprávu SM-CA se zbylými potvrzeními, které jednotky MU zpracují.
- V čase 10,2s dobíhá i doba t_{sm}. Centrále se za předchozí cyklus nikdo jiný nepřihlásil, proto nebyla odeslána řídící zpráva a může začít nový cyklus se stejnými parametry jako předchozí.

9.2 Zatížení linky

Předchozí část ověřovala funkčnost implementovaného systému. V této, bude provedena simulace, která bude zkoumat vliv použití či nepoužití kumulativního potvrzování na zatížení linky mezi centrálou a přípojným uzlem do sítě. Simulační topologii ukazuje obr.9.1. a opět popisuje soubor *basic.tcl* přiložený na CD.



Obr.9.1.: Simulovaná topologie odpovídající skriptu basic.tcl pro 200 MU.

Topologie se nyní sestává z jedné centrální jednotky CU, přístupového uzlu N1 a 200 měřicích jednotek MU. Maximální počet potvrzení ve zprávě SM-CA je nastaven na 183. Doba t_d je nastavena na 5s.

Při simulaci bez použití CA je třeba agenty upravit aby potvrzovali jednotlivým klientům po každé přijaté měřicí zprávě. Řešení bylo velmi jednoduché. Stačilo nastavení maximálního počtu potvrzení na hodnotu 1 a zprávu směrovat nikoliv na multicastovou adresu, ale přímo odesílateli.

Měření a zpracování dat
Měřená data představují provoz na lince mezi CU a N1. Ty je možné pomocí následujících tcl příkazů, použitých přímo v simulačním skriptu, vypsat do souboru a to jak ve směru CU – N1 tak ve směru N1 – CU.

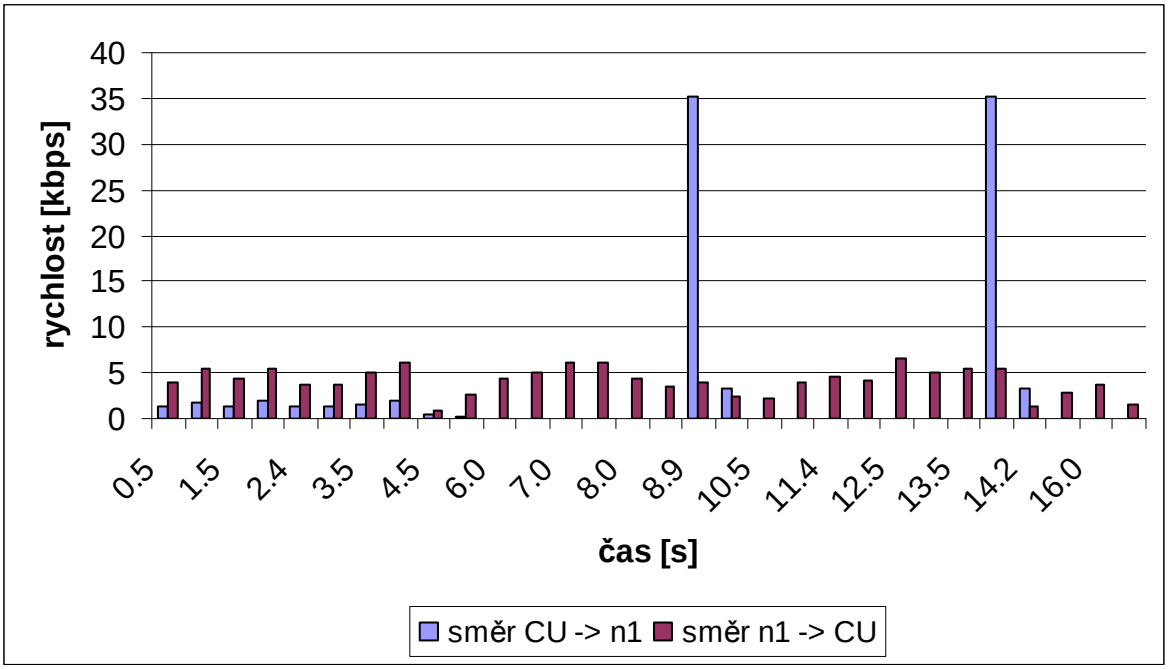
```
$ns trace-queue $n0 $n1 $f01
$ns trace-queue $n1 $n0 $f10
```

N0 zde představuje uzel ke kterému je připojen agent CU. V získaným souborech je spousta informaci, jako například číslo odesílatele, příjemce, čas odeslání, příjmu, velikost paketu atd. Pro náš účel budou dále zpracovány čas příjmu paketu daným uzlem a jeho velikost.
Zatížení linky je pak počítáno jako počet přenesených bitů za časovou jednotku z následujícího vztahu (3). Inspiraci k výpočtu byla získána v článku [11].

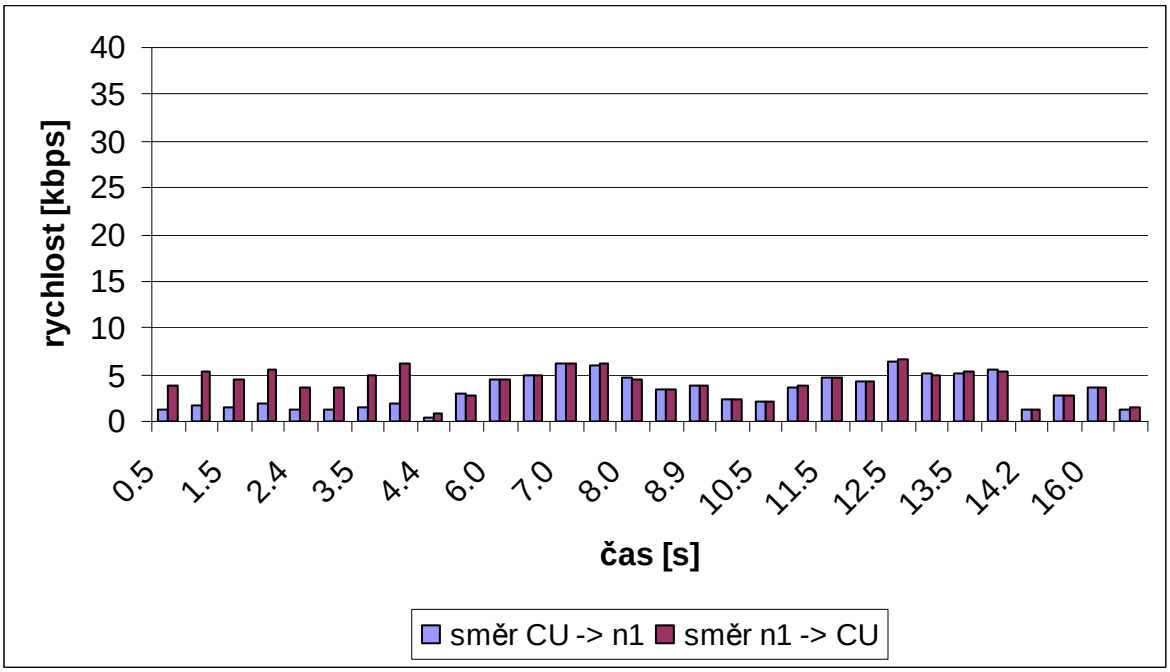
$$kbps = \left(\frac{bw * 8}{time} \right) / 1000 , \qquad (3)$$

kde *kbps* představuje potřebnou přenosovou rychlost v jednotlivých měřených intervalech, *time* je interval během něhož jsou sčítány přenášené bajty a *bw* je suma bajtů za dobu *time*.

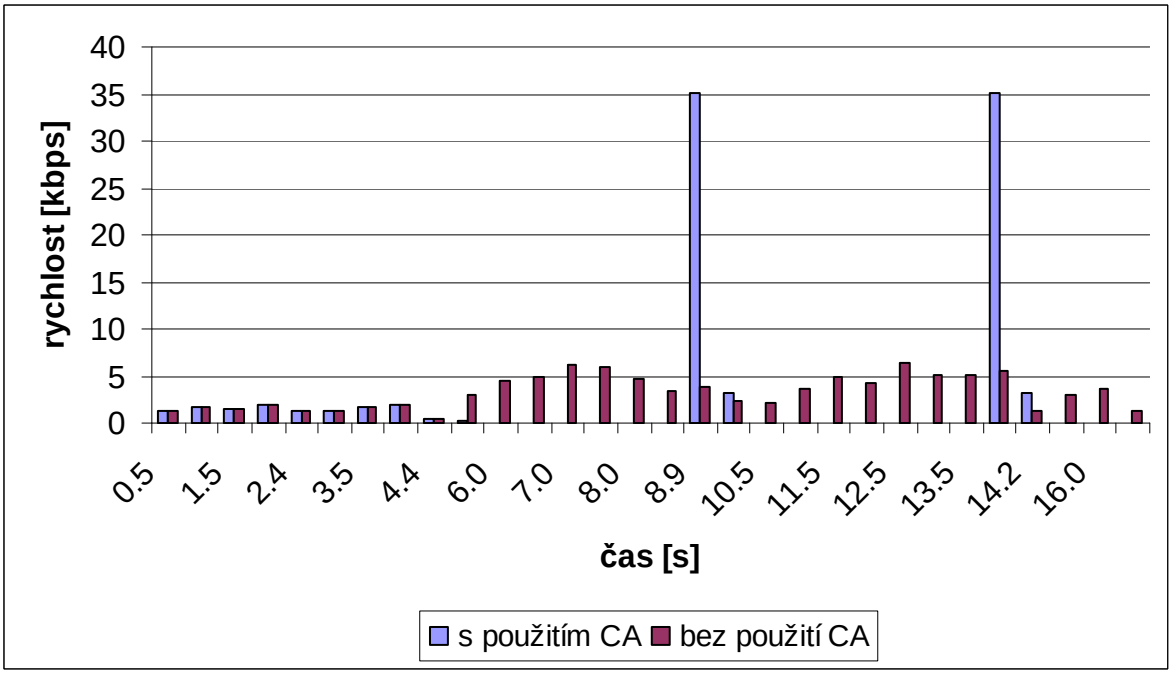
Dosažené výsledky ukazují následující grafy. Jejich časové rozložení (synchronizační doba, t_{um} a t_{sm}) je obdobné, jak popisuje příloha B s tím, že jsou vyobrazeny dva cykly. Grafy na obr.9.2 a 9.3 ukazují závislost potřebné přenosové v závislosti na aktuálním čase simulace. Rozdíl je, že první graf počítá s použitím kumulativního potvrzení a druhý graf již ne. Je patrné že použití kumulativního potvrzení nemá vliv na přenos ve směru N1 – CU, proto je v grafu na obr.9.4 vynechán a je zobrazeno jen porovnání pro systém s a bez použití kumulativního potvrzování.



Obr.9.2.: Závislost přenosové rychlosti na čase pro systém s použitím kumulativního potvrzování.



Obr.9.3.: Závislost přenosové rychlosti na čase pro systém bez použití kumulativního potvrzování.



Obr.9.4.: Závislost přenosové rychlosti na čase pro systém s a bez použití kumulativního potvrzování pro směr CU – N1.

Předpokládané ušetření přenosové kapacity není z dosažených výsledků nijak oslnivé, jak ukazuje poslední graf. Ovšem je zřejmé že při použití CA dochází k ojedinělým špičkovým nárůstům přenosové rychlosti, kdežto při jednotlivém klasickém potvrzování je potřebná přenosová rychlost mnohem menší, ale je vyžadována po celou dobu t_{um}.

10. Závěr

V této diplomové práci byl popsán princip,návrh, implementace a simulace již dříve navrženého protokolu pro systém hromadného sběru dat. Tento systém měl být implementován do některého vhodného simulačního prostředí. Konkrétně byl zvolen Network Simulator 2. Systém pak mohl od návrhové fáze přikročit k fázi simulační.

Jednotlivé cíle představené v úvodu jsou řešeny v několika samostatných kapitolách. Byl popsán jak samostatný protokol, jeho princip funkce, struktura a význam jednotlivých zpráv, tak komunikační metody, které tento systém využívá. Především byl popsán multicast a jeho typy jako ASM a SSM. Ovšem stručně byl čtenář seznámen i unicastovým typem komunikace. Dále zde byly popsány obecné postupy pro implementaci nových protokolů a rozšíření do prostředí Ns-2. Na tyto poznatky pak bylo navázáno v části o návrhu a následně při praktické realizaci. Podařilo se naprogramovat dva agenty, představující svou funkcí navržené jednotky CU a MU systému hromadného sběru dat. Jejich funkčnost byla ověřena pomocí simulačních skriptů. Rovněž byly vytvořeny simulace, které měli otestovat vliv a možné výhody či nevýhody, navrženého kumulativního potvrzování. Z dosažených výsledků vyplynulo, že při používání kumulativního potvrzení jsou prostředky linky využívány v určitých časových intervalech, daných dobou cyklu. Bez použití CA je pak linka zatěžována po celou dobu příjmu měřicích zpráv a velikost zatížení odpovídá množství těchto zpráv.

Z celkového hlediska, byly jednotlivé předdefinované cíle splněny. Vytvořený simulační model je naprogramován v celku jednoduše a umožňuje,na základě dalších simulací nebo v případě potřeby, snadnou úpravu či přidání nových funkcí, zpráv atd.

11. Použitá literatura

[1] *The Network Simulator - Ns-2* [online]. 29 December 2005 , 12:53, 21 November 2008 [cit. 2009-05-20]. Oficiální stránky Ns-2. Anglicky. Dostupný z WWW: <http://www.isi.edu/nsnam/index.php/User_Information>.

[2] *Cygwin* [online]. 2000 [cit. 2009-05-20]. Oficiální stránky emulátoru Cygwin. Anglicky. Dostupný z WWW: <http://cygwin.com/>.

[3] FILIP, Ondřej. Úvod do IP multicastu. *Lupa* [online]. 2004 [cit. 2009-05-20]. Dostupný z WWW: <http://www.lupa.cz/clanky/uvod-do-ip-multicastu/>.

[4] KOUTNÝ, Martin. Základ hromadného sběru dat. [2008]. 18 s.

[5] *Source Specific Multicast Extension to Ns-2* [online]. 2004 [cit. 2009-05-20]. SSM v Ns-2. Anglicky. Dostupný z WWW: <http://eden.dei.uc.pt/~tandre/ssm_extension/>.

[6] CHUNG, Jae, CLAYPOOL, Mark. *NS by Example* [online]. WPI, 1995-2008 [cit. 2009-05-20]. Příklady práce s Ns2. Anglicky. Dostupný z WWW: <www.nile/wpi.edu/NS/>.

[7] KOMOSNÝ, Dan. Nové směry vývoje protokolu RTP/RTCP pro rozsáhlé konference v Internetu. *Elektrorevue* [online]. 2004 [cit. 2009-05-20]. Dostupný z WWW: <http://www.elektrorevue.cz/clanky/04052/index.html>.

[8] FILIP, Ondřej. Úvod do IP multicastu. *Lupa* [online]. 2004 [cit. 2009-05-20]. Dostupný z WWW: <http://www.lupa.cz/clanky/uvod-do-ip-multicastu-dil-treti/>.

[9] CAMILO, Tiago, SÁ SILVA, Jorge, BOAVIDA, Fernando. *Enabling Ns-2 whith SSM environments*. [s.l.] : [s.n.], [2004?]. 6 s. Dostupný z WWW: <http://eden.dei.uc.pt/~tandre/ssm_extension/>.

[10] *Ns-2.26SourcesOriginal documentation* [online]. 2004 [cit. 2009-05-20]. Dokumentace k Ns-2.26. Anglicky. Dostupný z WWW: <http://www-rp.lip6.fr/ns-doc/ns226-doc/html/index.htm>.

[11] POLÁŠEK, Roman. Kdo měří rychlost připojení k internetu nejlépe? 2.díl. *Digitálně* [online]. 2008 [cit. 2009-05-20]. Dostupný z WWW: <http:// digitalne.stahuj.centrum.cz/kdo-meri-rychlost-pripojeni-k-internetu-nejlepe-2-dil/>.

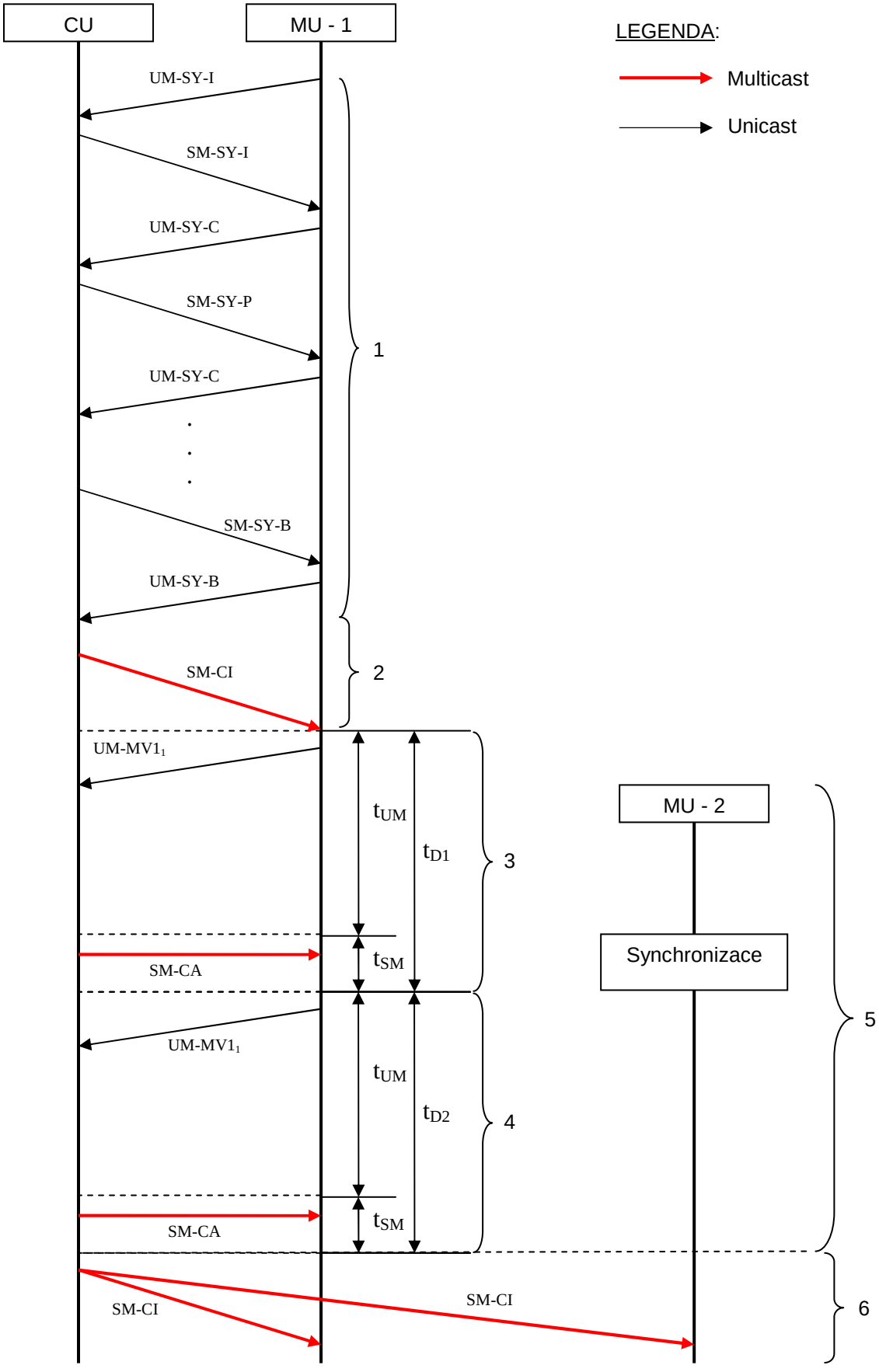
12. Seznam použitých zkratek a symbolů

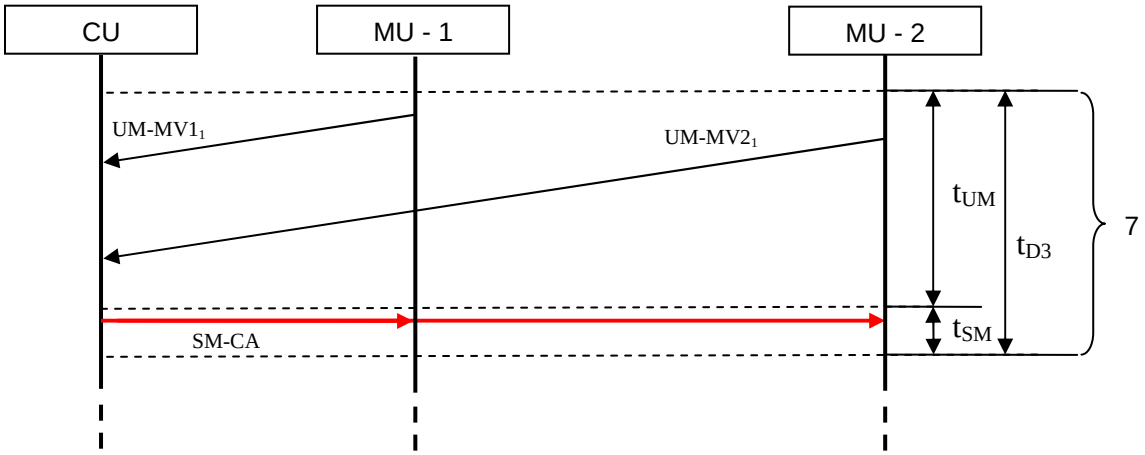
SSM – typ multicastové komunikace „jeden k mnoha“ (Specific Source Multicast)
ASM – typ multicastové komunikace „mnoho k mnoha“ (Any Source Multicast)
Multicast – způsob spojení účastníků, kdy je možná komunikace jednoho (nebo více) s mnoha dalšími současně.
Unicast – nejstarší typ komunikace představující komunikaci pouze jednoho k jednomu.
RTP – protokol aplikační vrstvy pro přenos multimediálních dat v reálném čase
UDP – protokol transportní vrstvy. Nespojově orientovaný, nespolehlivý.
CBR – v tomto případě představuje generátor dat s konstantní přenosovou rychlostí
MU – označení měřicí jednotky (Measure Unit)
CU – označení centrální jednotky (Central Unit)
MDCP – pracovní název navrženého protokolu (Multiple Data Collection Protokol).
T_d – doba jednoho cyklu dána jako $t_d = T_{um} + T_{sm}$. Využívaná i jako doba úvodní synchronizace po startu CU.
T_{sm} – jedná se o 1/5 doby t_d a představuje dobu kdy MU neodesílají data a CU odesílají zbylá potvrzení, přepočítávají t_d a odesílají řídící zprávu.
T_{um} – jedná se o dobu 4/5 t_d. V této době jednotky MU odesílají centrále zprávy s naměřenými daty.
Operátor new – dynamická alokace paměti.
Operátor delete – po dynamické alokaci je třeba paměť uvolnit tímto příkazem.
PacketData – datový typ definovaný v Network Simulatoru 2.
Unsigned int* – ukazatel na celočíselný, nezáporný datový typ. Rozsah 2^{32}.
Unsigned char* – ukazatel na místo v paměti o velikosti znaku (0 .. 255).
OTcl – objektově orientovaná forma jazyka Tcl, což je jednoduchý skriptovací jazyk.
NAM – součást simulačního programu Ns-2, slouží k vizualizaci výsledků simulačních skriptů (Network Animator).
NS-2 – jedná se o simulátor diskrétních událostí, zaměřený na simulaci a výzkum sítí (Network Simulator 2).

13. Seznam příloh

Příloha A: Časový diagram komunikace dle protokolu MDCP
Příloha B: Výpis programu Ns-2 pro skript basic.tcl

Příloha A: Časový diagram komunikace dle protokolu MDCP




CU
MU - 1
MU - 2
UM-MV1$_1$
UM-MV2$_1$
t_{UM}
t_{D3}
t_{SM}
SM-CA
7

Příloha B: Výpis programu Ns-2 pro skript basic.tcl

```
running nam...
MU228 : start OK - cas: 0.200000
MU229 : start OK - cas: 0.200000
MU230 : start OK - cas: 0.200000
MU231 : start OK - cas: 0.200000
MU232 : start OK - cas: 0.200000

MU228 :odesilam synchronizacni zpravu - cas: 0.201358  ... odeslano
CU: prijem paketu S od 228 - cas: 0.211486
CU: klient 228 pridan do mcastove skupiny.. pocet klientu 1

MU230 :odesilam synchronizacni zpravu - cas: 0.256819  ... odeslano
CU: prijem paketu S od 230 - cas: 0.266947
CU: klient 230 pridan do mcastove skupiny.. pocet klientu 2

MU231 :odesilam synchronizacni zpravu - cas: 0.552491  ... odeslano
CU: prijem paketu S od 231 - cas: 0.562619
CU: klient 231 pridan do mcastove skupiny.. pocet klientu 3

MU232 :odesilam synchronizacni zpravu - cas: 1.964694  ... odeslano
CU: prijem paketu S od 232 - cas: 1.974822
CU: klient 232 pridan do mcastove skupiny.. pocet klientu 4

MU229 :odesilam synchronizacni zpravu - cas: 2.423523  ... odeslano
CU: prijem paketu S od 229 - cas: 2.433651
CU: klient 229 pridan do mcastove skupiny.. pocet klientu 5

CU : dobehla doba td - uvodni synchronizace - cas: 5.200000
CU : paket I odeslan - cas: 5.200000
 :paket I, pocet klientu 5 a new td 5
MU230 : start OK - cas: 5.210171
 :paket I, pocet klientu 5 a new td 5
MU231 : start OK - cas: 5.210171
 :paket I, pocet klientu 5 a new td 5
MU232 : start OK - cas: 5.210171
 :paket I, pocet klientu 5 a new td 5
MU228 : start OK - cas: 5.210171
 :paket I, pocet klientu 5 a new td 5
MU229 : start OK - cas: 5.210171

MU228 :odesilam merici zpravu - cas: 6.146192  ... odeslano
CU: prijem paketu M od 228 - cas: 6.156320

MU231 :odesilam merici zpravu - cas: 6.458835  ... odeslano
CU: prijem paketu M od 231 - cas: 6.468963

MU229 :odesilam merici zpravu - cas: 7.759134  ... odeslano
CU: prijem paketu M od 229 - cas: 7.769262

CU : paket CA odeslan - cas: 7.769262
MU231 :potvrzena zprava s timestampem = 6.458835
MU228 :potvrzena zprava s timestampem = 6.146192
MU229 :potvrzena zprava s timestampem = 7.759134


MU230 :odesilam merici zpravu - cas: 8.398665  ... odeslano
```

CU: prijem paketu M od 230 - cas: 8.408793

MU232 :odesilam merici zpravu - cas: 8.467771 ... odeslano

CU: prijem paketu M od 232 - cas: 8.477899
CU : dobehla doba t_UM - cas: 9.200000
CU : paket CA odeslan - cas: 9.200000

MU230 :konec t_um zacal t_sm - cas: 9.200000
MU231 :konec t_um zacal t_sm - cas: 9.200000
MU232 :konec t_um zacal t_sm - cas: 9.200000
MU228 :konec t_um zacal t_sm - cas: 9.200000
MU229 :konec t_um zacal t_sm - cas: 9.200000

MU230 :potvrzena zprava s timestampem = 8.398665
MU232 :potvrzena zprava s timestampem = 8.467771

CU : dobehla doba t_SM - cas: 10.200000
MU230 :konec t_sm zacal t_um - cas: 10.200000
MU230 : start OK - cas: 10.200000
MU231 :konec t_sm zacal t_um - cas: 10.200000
MU231 : start OK - cas: 10.200000
MU232 :konec t_sm zacal t_um - cas: 10.200000
MU232 : start OK - cas: 10.200000
MU228 :konec t_sm zacal t_um - cas: 10.200000
MU228 : start OK - cas: 10.200000
MU229 :konec t_sm zacal t_um - cas: 10.200000
MU229 : start OK - cas: 10.200000

Další cyklus …