VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2018

Bc. Jaromír Polášek

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE PROTOKOLU HDLC V SÍŤOVÝCH SIMULÁTORECH
IMPLEMENTATION OF HDLC PROTOCOL IN NETWORK SIMULATORS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jaromír Polášek**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jiří Pokorný**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Jaromír Polášek ***ID:*** 164770
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Implementace protokolu HDLC v síťových simulátorech

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce bude prozkoumat možnosti implementace protokolu HDLC v dostupných síťových simulátorech (NS-2, NS-3, IT Guru, ...). Na základě poznatků bude zvolen nejvhodnější simulátor a následně navrženy simulační scénáře pro několik odlišných síťových komunikačních topologií v rámci Smart Grid infrastruktury (tj. přenos dat mezi elektroměrem a datovým koncentrátorem). Dále bude ověřena funkčnost vytvořených topologií a provedeno porovnání využití protokolu HDLC a adresace pomocí TCP/IP s dopadem na celkový charakter komunikace mezi zařízeními v Smart Grid infrastruktuře.

DOPORUČENÁ LITERATURA:

[1] MARTINI, Luca, et al. Encapsulation Methods for Transport of PPP/High-Level Data Link Control (HDLC) over MPLS Networks. 2006.

[2] LIMPHAPAYOM, Siwarat; PORA, Wanchalerm. An emulation of data concentrator units conformed to DLMS-HDLC protocols. In: Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2013 10th International Conference on. IEEE, 2013. p. 1-6.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: Ing. Jiří Pokorný
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato diplomová práce se zabývá možností využití HDLC (*High-Level Data Link Control*) protokolu pro komunikaci a adresování inteligentních měřících zařízení s datovým koncentrátorem. HDLC protokol je použitý ve dvou DLMS/COSEM (*Device Language Message Specification/Companion Specification for Energy Metering*) komunikačních profilech. Pro simulaci těchto komunikačních profilů je zvolen nejvhodnější simulační program, do kterého je implementován první komunikační profil a druhý je navržen. Implementace komunikačního profilu využívající TCP/IP (*Transmission Control Protocol/Internet Protocol*) byla plně realizována. Pro implementaci třívrstvého HDLC komunikačního profilu byly důsledně prozkoumány všechny možnosti. Pomocí zjištěných poznatků byl navrhnut postup, který slouží jako vodítko pro plnou implementaci. Pro první komunikační profil jsou změřeny kvalitativní parametry, které jsou zaneseny do grafů a zhodnoceny.

KLÍČOVÁ SLOVA

DLMS/COSEM, HDLC, inteligentní síť, komunikační profil, NS-2, NS-3, paket, propustnost, rámec, Riverbed, síťová simulace, šířka pásma, ztrátovost

ABSTRACT

This diploma thesis deals with the possibility of using HDLC (*High-Level Data Link Control*) protocol for communication and addressing of smart metering devices with a data concentrator. The HDLC protocol is used in two DLMS/COSEM (*Device Language Message Specification/Companion Specification for Energy Metering*) communication profiles. To simulate these communication profiles, the most appropriate simulation program is selected. Using this simulator, the first communication profile is implemented and the second one is designed. Communication profile based on TCP/IP (*Transmission Control Protocol/Internet Protocol*) has been fully implemented. To implement the three-layer HDLC communication profile, all options have been thoroughly explored. Using these findings, a process was designed to guide the full implementation. For the first communication profile the qualitative parameters are measured, which are then plotted and evaluated.

KEYWORDS

DLMS/COSEM, HDLC, SmartGrid, communication profile, NS-2, NS-3, packet, throughput, frame, Riverbed, network simulation, bandwidth, packet loss

POLÁŠEK, Jaromír. *Implementace protokolu HDLC v síťových simulátorech*. Brno, 2017, 95 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Jiří Pokorný,

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Implementace protokolu HDLC v síťových simulátorech“ 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/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Jiřímu Pokornému, a mému konzultantovi Ing. Pavlu Maškovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno ....

podpis autora

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

OBSAH

Úvod **13**

1 Inteligentní elektrická síť **14**
1.1 Architektura a technologie inteligentní elektrické sítě 14

2 Protokol HDLC **17**
2.1 Základní informace 17
2.2 Přenosové režimy 17
2.3 Rámce 18
2.3.1 Řídící pole pro informační rámce 19
2.3.2 Řídící pole pro řídící rámce 19
2.3.3 Řídící pole pro nečíslované rámce 20
2.4 HDLC DLMS/COSEM 20
2.4.1 Specifikace DLMS/COSEM obecně 21
2.4.2 Komunikační profily 22
2.4.3 Komunikační profil DLMS/COSEM využívající HDLC 23

3 Možnosti simulace datových sítí **26**
3.1 Simulátory síťového provozu 26
3.1.1 Simulátor NS-2 27
3.1.2 Simulátor NS-3 27
3.1.3 Simulátor Riverbed 27
3.2 Srovnání síťových simulátorů 28
3.2.1 Srovnání simulátorů NS-2 a NS-3 29
3.3 Úvod do simulátoru NS-2 30
3.3.1 Základní architektura 30
3.3.2 Struktura adresáře NS-2 31
3.3.3 Spuštění simulace NS-2 31

4 Implementace protokolu HDLC v simulátoru NS-2 **35**
4.1 Síťový model v simulátoru NS-2 35
4.1.1 Mechanismy přenosu dat 35
4.1.2 Adresace 36
4.2 Uzel 37
4.2.1 Architektura uzlu 38
4.2.2 Část základního uzlu 39
4.2.3 Rozšířená část mobilního uzlu 39
4.3 Paket v simulátoru NS-2 39

4.3.1 Architektura paketu 40
4.3.2 Záhlaví paketu 42
4.4 HDLC pro satelitní přenos 45
4.4.1 HDLC proces 46
4.4.2 Nevýhody satelitní implementace 53

5 Komunikační profily využívající protokol HDLC 54
5.1 LAN síť a uzel 54
5.1.1 Linková vrstva 54
5.1.2 MAC vrstva 55
5.1.3 Fyzická vrstva a kanál 56
5.1.4 Směrování v LAN síti 56
5.2 Komunikační profil HDLC využívající TCP/IP pomocí simulátoru NS-2 58
5.2.1 Úprava C++ kódu HDLC tříd 59
5.2.2 Úprava trasovacích tříd pro kabelové HDLC 62
5.3 Návrh třívrstvého komunikačního profilu HDLC pomocí simulátoru NS-2 65
5.3.1 Úprava C++ kódu třídy „sat-hdlc.cc“ 67
5.3.2 Návrh třívrstvého komunikačního profilu s využitím upraveného agenta 69
5.3.3 Návrh třívrstvého komunikačního profilu se zahrnutím klasifikátoru 70

6 Výsledky simulace získané pomocí komunikačních profilů HDLC 72
6.1 Kvalitativní parametry počítačových sítí 72
6.1.1 Šířka pásma 72
6.1.2 Propustnost 72
6.1.3 Zpoždění 73
6.1.4 Ztrátovost 74
6.1.5 Jitter 74
6.2 Návrh a implementace topologií v programu NS-2 74
6.2.1 Specifikace použitých topologií 74
6.2.2 Tcl implementace topologií 76
6.3 Simulace kvalitativních parametrů satelitní implementace HDLC . . . 78
6.3.1 Závislost průměrné propustnosti linky na šířce pásma 78
6.3.2 Závislost ztrátovosti linky na šířce pásma 79
6.4 Simulace kvalitativních parametrů HDLC komunikačního profilu využívajícího TCP/IP protokoly 81

6.4.1 Závislost ztrátovosti HDLC linky na zpoždění 82
6.4.2 Závislost ztrátovosti HDLC linky na šířce přenosového pásma 83

7 Závěr **85**

Literatura **87**

Seznam symbolů, veličin a zkratek **90**

Seznam příloh **93**

A Prostředky využité při implementaci **94**

B Obsah přiloženého DVD **95**

SEZNAM OBRÁZKŮ

1.1 Architektura a technologie inteligentní elektrické sítě. 14
1.2 Požadavky na komunikační rychlost a dosah pokrytí pro hierarchii inteligentní elektrické sítě. 15
2.1 Struktura rámců protokolu HDLC. 18
2.2 Struktura kontrolního pole jednotlivých rámců. 19
2.3 Generický komunikační profil DLMS/COSEM. 22
2.4 Porovnání dvou komunikačních profilů DLMS/COSEM využívajících HDLC. 24
3.1 Architektura simulátoru NS-2. 30
3.2 Struktura adresáře simulátoru NS-2 32
3.3 Srovnání obecných kroků simulace a kroků simulátoru NS-2. 33
4.1 Srovnání síťového modelu klasické sítě a simulátoru NS-2. 36
4.2 Diagram přenosu dat mezi jednotlivými vrstvami u protokolu TCP. . 37
4.3 Schéma mobilního uzlu. 38
4.4 Modelování paketu v NS-2. 41
5.1 Schéma konektivity v LAN síti. 55
5.2 Reálná konfigurace LAN srovnaná s konfigurací viděnou programem NS-2. 57
5.3 Komunikační profil HDLC využívající IP adresaci v simulátoru NS-2. 58
5.4 Napravo je uvedeno schéma třívrstvého komunikačního profilu využívající protokol HDLC v simulátoru NS-2, nalevo je uveden stejný profil, ale s přidaným klasifikátorem. 66
6.1 Topologie použité pro simulaci kvalitativních parametrů. 75
6.2 Graf závislosti průměrné propustnosti linky na šířce pásma. 79
6.3 Graf závislosti ztrátovosti rámců na šířce pásma. 81
6.4 Graf závislosti ztrátovosti paketů na zpoždění. 83
6.5 Graf závislosti ztrátovosti paketů na šířce pásma. 84

SEZNAM TABULEK

3.1 Obecné srovnání síťových simulátorů. 28
3.2 Srovnání simulátorů NS-2 a NS-3. 29
3.3 Formát řádků trasovacího souboru kabelových linek 34
4.1 Možnosti nastavení mobilních a satelitních uzlů [20]. 39
5.1 Vstupní parametry funkce `make-lan` 56
5.2 Část trasovacího souboru rozšířená protokolem HDLC 62
6.1 Závislost průměrné propustnosti linky na šířce pásma. 78
6.2 Závislost ztrátovosti rámců na šířce pásma. 80
6.3 Závislosti ztrátovosti paketů na zpoždění pro komunikační profil HDLC. 82
6.4 Závislost ztrátovosti paketů na šířce pásma linky pro komunikační profil HDLC. 84

SEZNAM VÝPISŮ

4.1 Deklarace třídy `Packet` . . . 42
4.2 Datový typ `struct hdr_cmn` . . . 43
4.3 Datový typ `struct hdr_ip` . . . 44
4.4 Datový typ `struct hdr_hdlc` . . . 45
4.5 Funkce `HDLC::recv(Packet *p, Handler*)` . . . 46
4.6 Funkce `HDLC::recvOutgoing(Packet *p)` . . . 47
4.7 Funkce `HDLC::sendUA(Packet* p, COMMAND_t cmd)` . . . 48
4.8 Funkce `HDLC::sendDown(Packet *p` . . . 49
4.9 Funkce `inSendBuffer(Packet *p, ARQstate *a)` . . . 50
4.10 Funkce `HDLC::recvIncoming(Packet *p)` . . . 50
4.11 Funkce `recvUframe(Packet *p)` . . . 51
4.12 Funkce `handleSAMBErequest(Packet *p)` . . . 52
5.1 Funkce pro vytvoření LAN sítě v programu NS-2 . . . 56
5.2 Funkce `next_hop(Packet *p)` . . . 57
5.3 Úprava dědičnosti HDLC třídy . . . 59
5.4 Funkce `sendUp(Packet *p)` . . . 60
5.5 Upravená funkce `HDLC::sendDown(Packet *p)` . . . 61
5.6 Výňatek upravené funkce `HDLC::recvOutgoing(Packet* p)` . . . 62
5.7 Funkce `Trace::recv(Packet *p, Handler *h)` . . . 62
5.8 Výňatek z funkce `Trace::format(int tt, int s, int d, Packet* p)` . . . 63
5.9 Formátování HDLC rámce typu S . . . 64
5.10 Inicializace funkce `SatTrace::format_hdlc(Packet *p, int offset)` 64
5.11 Upravená část souboru „vlan.tcl“ . . . 65
5.12 Upravená funkce `HDLC::recv(Packet *p)` . . . 67
5.13 Upravená funkce `HDLC::inSendBuffer(Packet *p, ARQstate *a)` . 68
5.14 Deklarace pro nastavení paketu do tvaru HDLC rámce . . . 69
5.15 Výpis nastavení portů HDLC agenta v souboru „ns-agent.tcl“ . . . 70
5.16 Výpis přidaných funkcí v souboru „ns-lib.tcl“. . . . 70
6.1 Implementace topologie pro HDLC komunikační profil využívající protokol TCP/IP . . . 77

ÚVOD

Jednou z nejrychleji se rozvíjejících sítí současnosti je inteligentní elektrická síť. Jelikož její vývoj začal až v 21. století její optimalizace je stále otevřený problém, jehož cílem je najít ideální protokolovou sadu, která by umožnila dosáhnout ideálních přenosových parametrů. Tato diplomová práce se konkrétně zabývá využitím protokolu HDLC pro komunikaci a adresaci chytrých měřících zařízení s datovým koncentrátorem. Protokol HDLC je využit ve dvou komunikačních profilech specifikace DLMS/COSEM.

Inteligentní elektrické sítě jsou většinou velmi složité a rozsáhlé. Proto je čím dál víc efektivnější před fyzickým sestaveném navrhovanou síť nasimulovat. Může se tak naleznout nejefektivnější rozložení síťových prvků nebo předejít krizovým situacím, které by mohly poškodit síťové komponenty. Z těchto důvodů se začali využívat simulační programy, které mohou žádanou síť virtuálně simulovat. V současnosti existuje velké množství simulačních programů z nichž každý má své výhody a nevýhody. Cílem této práce je prozkoumat vlastnosti různých simulačních programů a vybrat z nich ten nejvhodnější pro simulaci komunikačních profilů využívající protokol linkové vrstvy. Jím bude následně simulovaná jednoduchá topologie, na kterou budou komunikační profily nasazeny. Nakonec budou pomocí simulace zjištěny kvalitativní parametry získané pomocí komunikačních profilů. Kvalitativní parametry budou zhodnoceny a bude z nich odvozen výsledek.

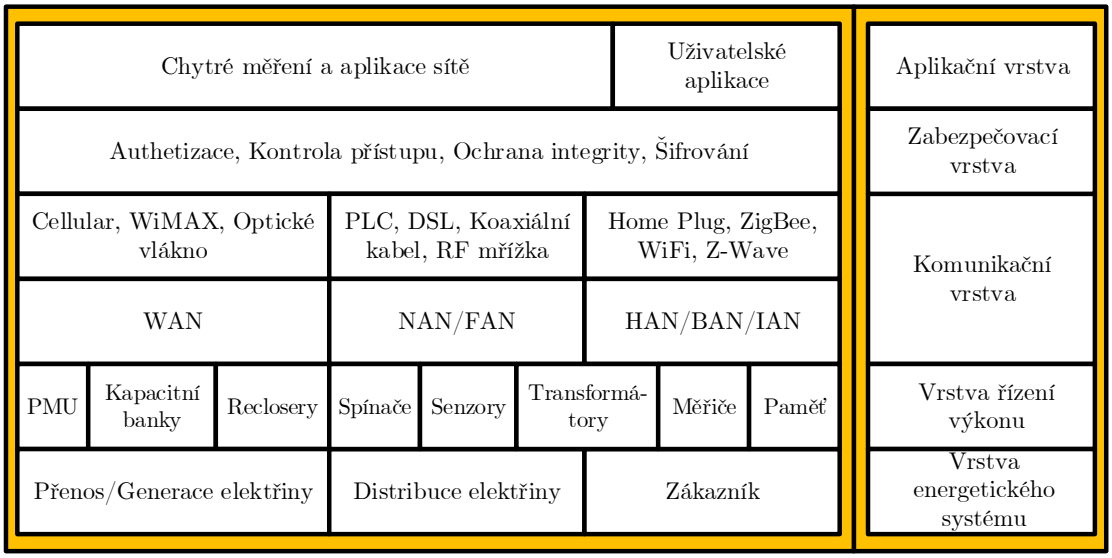
Diplomová práce je rozdělena do sedmi kapitol. V první kapitole jsou vysvětleny základy inteligentní elektrické sítě. Druhá kapitola popisuje protokol HDLC a specifikaci HDLC/COSEM. Nejdůležitější část této kapitoly je grafické zobrazení dvou komunikačních profilů využívajících protokol HDLC, které budou následně implementovány. Třetí kapitola popisuje simulační programy, které by mohli být zvoleny pro implementaci, kdy na jejím konci jsou popsány základy simulátoru. Čtvrtá kapitola je zaměřena na samotnou implementaci protokolu HDLC ve zvoleném simulátoru. Pátá kapitola se věnuje implementaci komunikačních profilů specifikace DLMS/COSEM využívajících protokolu HDLC. V šesté kapitole jsou diskutovány výsledky simulace získané při použití dvou typů komunikačních profilů. Sedmá kapitola slouží k shrnutí výsledků této diplomové práce.

1 INTELIGENTNÍ ELEKTRICKÁ SÍŤ

Inteligentní elektrická síť je definována jako modernizovaná energetická síť, která využívá informační a komunikační technologie pro shromažďování a zpracování informací pro zlepšení efektivity, spolehlivosti, ekonomiky, udržitelnosti výroby, přenosu a distribuce elektřiny. V této kapitole je popsána architektura této sítě a technologie, které využívá [1, 2, 3].

1.1 Architektura a technologie inteligentní elektrické sítě

Inteligentní síť je interaktivní platforma, která se skládá z vrstvy energetického systému, vrstvy řízení výkonu, komunikační vrstvy, vrstvy zabezpečení a aplikační vrstvy. Její struktura je vyobrazena na Obr. 1.1 [1, 2, 3].



Obr. 1.1: Architektura a technologie inteligentní elektrické sítě. [1].

Uvedená architektura zároveň představuje způsob implementace inteligentních sítí. Funkce zmíněných vrstev jsou následující:

1. Vrstva energetického systému se vztahuje na systémy výroby energie, přenosu, distribuce a zákazníků.
2. Vrstva řízení výkonu, která umožňuje monitorovat, řídit a spravovat inteligentní sítě.
3. Komunikační vrstva v rámci inteligentní elektrické sítě umožňuje obousměrnou komunikaci.

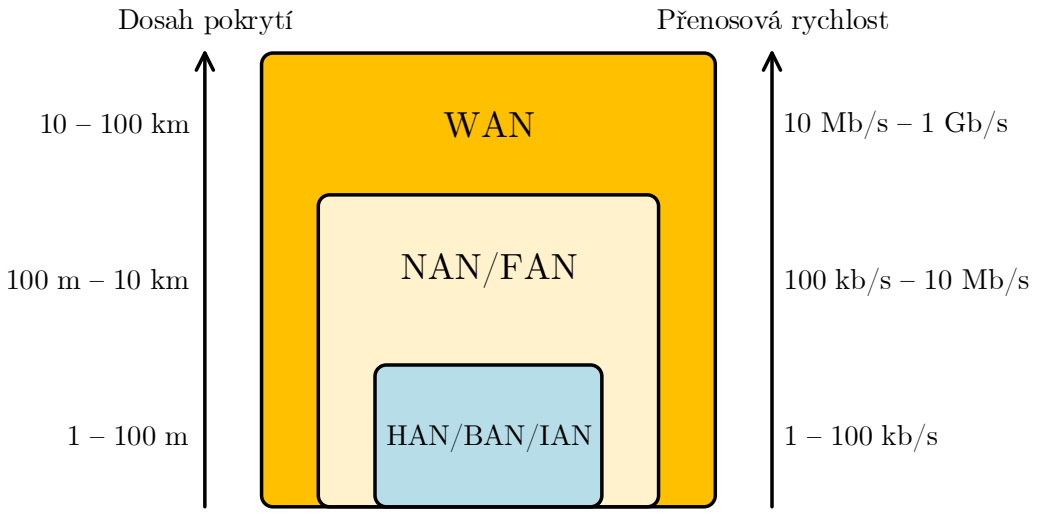
4. Bezpečnostní vrstva poskytuje důvěrnost, integritu, autentizaci a dostupnost dat.
5. Aplikační vrstva zákazníkům poskytuje různé inteligentní síťové aplikace a nástroje založené na existující informační infrastruktuře.

Například pro správnou funkci aplikace věnující se inteligentnímu měření musí mít inteligentní síť vrstvu energetického systému, která rozvádí elektrickou energii a dodává ji zákazníkům. Dále je potřebná řídící vrstva tvořená inteligentním měřičem, který umožňuje monitorování spotřeby energie. Komunikační vrstva je nezbytná pro přenos informací od zákazníka do měřiče nebo opačně. V neposlední řadě je pro správnou funkci inteligentního měřené bezpečnostní vrstva, která řeší problémy spojené s ochranou dat [1, 4, 5, 6].

Aby inteligentní síťová aplikace fungovala bez problémů musí využívat správnou komunikační vrstvu. V inteligentní elektrické síti může být komunikační síť reprezentována jako hierarchická vícevrstvá architektura, rozdělená podle přenosové rychlosti a dosahu pokrytí. Tato architektura obsahuje:

- Domácí síť (HAN (*Home Area Network*)), síť budov (BAN (*Building Area Network*)) a průmyslovou síť (IAN (*Industrial Area Network*))
- Síť sousedních (NAN (*Neighborhood Area Network*) a polních (FAN (*Field Area Network*) oblastí
- Rozsáhlou síť (WAN (*Wide Area Network*)

Přenosové rychlosti a dosah pokrytí těchto sítí jsou sumarizovány na Obr. 1.2.



Obr. 1.2: Požadavky na komunikační rychlost a dosah pokrytí pro hierarchii inteligentní elektrické sítě [1].

Aplikace HAN/BAN/IAN zahrnují automatizaci domácností a automatizaci budov, která souvisí s odesíláním a přijímáním elektrických údajů z přístroje k řadiči

v prostorech zákazníka. Tyto aplikace nevyžadují častý přenos dat a většina aplikací nachází uplatnění uvnitř obytných, komerčních a průmyslových budov. Tudíž požadavky na komunikaci pro HAN/BAN/IAN aplikace jsou: nízká spotřeba energie, nízké náklady, jednoduchost a bezpečná komunikace. Většinou stačí komunikační technologie, které dosahují přenosové rychlosti do 100 kb/s s dosahem do 100 m. Široce jsou za tímto účelem používány technologie ZigBee, WiFi, Z-Wave, PLC (Power Line Carrier), Bluetooth či Ethernet [1, 2, 7].

Příklady aplikací NAN/FAN jsou inteligentní měření, odpověď na požadavek zákazníka nebo automatizace distribuce. V těchto případech jsou požadovaná data přijímána od velkého množství zákazníků nebo polních zařízení do datového koncentrátoru nebo rozvodny a opačně. Proto tyto aplikace vyžadují komunikační technologie s vyšší přenosovou rychlostí (100 kb/s – 10 Mb/s) a vyším dosahem (100 m). NAN/FAN aplikace mohou být implementovány pomocí mřížkových ZigBee a WiFi sítí, PLC nebo pomocí technologií určených pro dálkový přenos jako jsou WiMAX, DSL (*Digital Subscriber Line*) a koaxiální kabel [1, 3, 4].

WAN aplikace jako je širokopásmové řízení, monitorování a zabezpečení vyžadují velmi častý přenos dat mezi velkým počtem vzdálených míst. Proto musíme u těchto aplikací použít přenosové technologie, které mohou dosáhnout velmi vysoké přenosové rychlosti (10 Mb/s – 1 Gb/s) na velkou vzdálenost (100 km). Díky své vysoké kapacitě a nízké latenci se běžně ke komunikaci mezi rozvodnými nebo distribučními rozvodnami a řídícím střediskem používají optické linky. Mobilní a WiMAX technologie nachází také časté použití a to kvůli jejich širokém rozsahu pokrytí a velké propustnosti. Satelitní komunikace může být také použita pro poskytnutí redundantního spojení v kritických místech [1, 2, 5].

2 PROTOKOL HDLC

V této kapitole je popsán protokol HDLC. Nejprve jsou uvedeny základní informace ohledně protokolu HDLC. Následně budou nastíněny jeho pracovní módy a do hloubky popsány jeho rámce. Nakonec bude popsána specifikace DLMS/COSEM.

2.1 Základní informace

V minulé kapitole byla probrána architektura inteligentních elektrických sítí. Z tohoto popisu víme, že inteligentní měření patří do aplikací typu NAN/FAN. Jedním z nejvíc využívaných komunikačních protokolů pro tyto aplikace je protokol HDLC, na který je další text zaměřený.

HDLC je bitově orientovaný protokol, který byl vyvinut organizací ISO. Spadá pod standardy ISO 3309 a ISO 4335. Jeho základní funkcí je stanovení paketizačních standardů pro sériové linky. HDLC podporuje polo-duplexní i plně duplexní přenosové linky, dále dvojbodové i vícebodové sítě a spínané i nespínané kanály. HDLC rovněž podporuje více pracovních režimů zahrnujících také prosté posuvné okno pro spolehlivý přenos. Poněvadž TCP/IP zajišťuje spolehlivý přenos na vyšších vrstvách, většina aplikací při použití HDLC používá nespolehlivý přenos [8].

2.2 Přenosové režimy

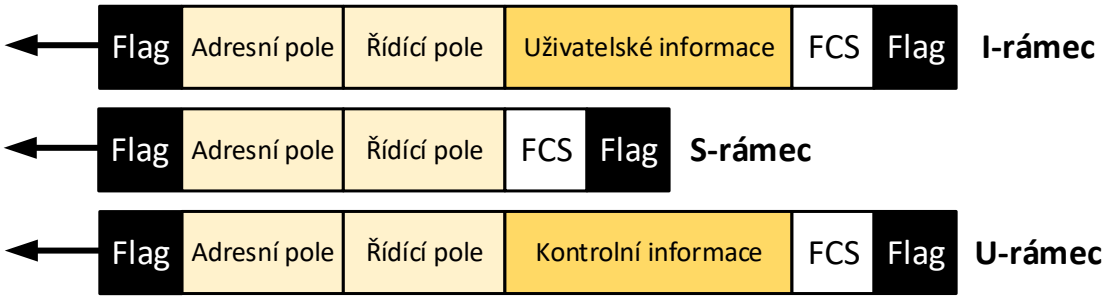
HDLC poskytuje tři základní přenosové režimy, které mohou být využity v různých konfiguracích. Tyto režimy jsou: režim normální odezvy NRM (*Normal Response Mode*), asynchronní vyvážený režim ABM (*Asynchronous Balanced Mode*) a režim asynchronní odezvy ARM (*Asynchronous Response Mode*). Při režimu NRM je konfigurace jednotlivých stanic nevyvážená. V síti se nachází jedna primární stanici a několik sekundárních stanic. Jen primární stanice může vysílat příkazy, sekundární stanice vysílají pouze odpovědi. Režim NRM se používá jak pro dvoubodové tak pro vícebodové linky [9].

Režim ARM pracuje také s nevyváženou konfigurací stanic, kdy na rozdíl od režimu NRM sekundární stanice mohou vysílat informace bez příkazu z primární stanice. Primární stanice, ale stále zodpovídá za inicializaci linky, logické rozpojeni a zotavení z chyb.

Poslední režim ABM pracuje s vyváženou konfigurací. Linka je dvoubodová a každá stanice může pracovat jako primární i sekundární. Tento režim je v současnosti nejvíc používaný [9, 10].

2.3 Rámce

Aby protokol HDLC byl dostatečně flexibilní pro všechny režimy a konfigurace, jdou definovány tři typy rámců: informační rámce (I-rámce), řídící rámce (S-rámce) a nečíslované rámce (U-rámce). Každý druh rámce slouží jako obálka, která přenáší určitý druh zprávy. Informační rámce jsou použity pro přenos uživatelských dat linkové vrstvy a řídících informací spojených s těmito daty. Řídící rámce jsou určeny pouze pro přenos řídících dat. Nečíslované rámce slouží pro přenos dat souvisejících s řízením samotné linky. Každý rámec protokolu HDLC může obsahovat až šest polí. Začínající značkovací pole, adresní pole, řídící pole, informační pole, pole s kontrolním součtem, a koncové značkovací pole. Při přenosu více rámců po sobě může koncové značkovací pole jednoho rámce sloužit jako začínající značkovací pole následující rámce. Struktura všech tří rámců je ukázána na Obr. 2.1 [9, 11, 8].

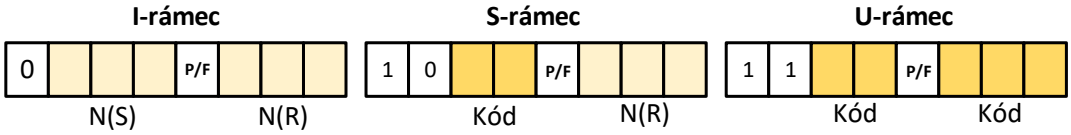


Obr. 2.1: Struktura rámců protokolu HDLC [9].

Následně jsou podrobněji vysvětleny jednotlivé pole:

- **Značkovací pole** – Obsahuje synchronizační posloupnost bitů 01111110, která označuje začátek i konec rámce
- **Adresní pole** – Obsahuje adresu sekundární stanice. Pokud rámec vytvoří primární stanici obsahuje cílovou adresu. Jestli rámec vytvoří sekundární stanici pole obsahuje zdrojovou adresu. Adresní pole může mít dle typu sítě velikost jeden nebo více bytů.
- **Řídící pole** – Je použito pro kontrolu toku a chyb. Jeho délka je jeden nebo dva byty. Podrobné vysvětlení funkce pole je nastíněno níže.
- **Informační pole** – Obsahuje data uživatele ze síťové vrstvy nebo kontrolní informace. Délka tohoto pole se mění podle typu sítě
- **Pole kontrolního součtu** – Slouží protokolu HDLC ke kontrole při přenosu vzniklých chyb. Může obsahovat 2 nebo 4 bytové CRC (*Cyclic Redundancy Check*).

Řídící pole určuje druh rámce a jeho funkčnost. Proto se tomuto poli budeme věnovat podrobněji. Na Obr. 2.2 lze sledovat, že formát řídícího pole je dán typem rámce [9].



Obr. 2.2: Struktura kontrolního pole jednotlivých rámců [9].

2.3.1 Řídící pole pro informační rámce

Informační rámce jsou navrženy tak aby nesly uživatelská data ze síťové vrstvy. Navíc mohou zahrnovat kontrolní informace o toku a chybách (*Piggybacking*). Jednotlivé bity kontrolního pole definují následující funkce:

- První bit definuje typ rámce, kdy 0 značí rámec informační.
- Následující 3 bity se nazývají *N(S)* a definují sekvenční číslo rámce. Tři bity mohou definovat sekvenční číslo (0 až 7).
- Pokud je použito (Piggybacking), poslední 3 bity *N(R)* odpovídají potvrzovacímu číslu. Bit mezi *N(S)* a *N(R)* je takzvaný *P/F* bit. Pokud je nastaven na 1, bit může mýt dva významy: finále(*final*) nebo průzkum (*pool*). Bit znamená průzkum pokud je rámec zaslán primární stanicí a finále pokud je zaslán sekundární stanicí [9, 10].

2.3.2 Řídící pole pro řídící rámce

Řídící rámce jsou využity pro kontrolu toku a korekci chyb kdykoliv je (Piggybacking) nemožné nebo nevhodné. Řídící rámce nemají informační pole. Pokud první 2 bity řídícího pole jsou 10, tak to znamená, že rámec je řídící. Poslední 3 bity nazývané *N(R)*, odpovídají potvrzovacímu číslu (ACK (*Acknowledgement*)) nebo negativnímu potvrzovacímu číslu (NAK (*Negative-Acknowledgement*)) v závislosti na typu řídícího rámce. Dva bity, tzv. „kód“ jsou použity pro definici typu řídícího rámce. Pomocí těchto dvou bitů lze definovat čtyři typy řídících rámců [9, 11]:

- **Odpověď připravena (*RR*)** – Pokud jsou tyto dva bity 00, jedná se o řídící rámec typu RR (*Receive Ready*), který potvrzuje přijetí správného a nepoškozeného rámce nebo skupiny rámců. *N(R)* bity odpovídají potvrzovacímu číslu [9, 11].

- **Odpověď není připravena (*RNR*)** – Pokud jsou tyto dva bity 10, jedná se o řídící rámec typu RNR (*Receive Not Ready*). RNR rámec stejně jako RR rámec potvrzuje přijetí rámce nebo skupiny rámců, ale navíc oproti RR rámci oznamuje, že přijímač je zaneprázdněn a nemůže přijmout další rámce. Jedná se o formu kontroly proti zahlcení požádáním odesílatele rámců o zpomalení. *N(R)* bity odpovídají potvrzovacímu číslu [9].
- **Odmítnout (*REJ*)** – Pokud jsou dva bity „kódu“ 01, jedná se o kontrolní rámec typu REJ (*Reject*). Tento rámec je typu NAK, ale nedá se použít pro selektivní opakování (*Selective Repeat ARQ (Automatic Repeat-reQuest)*). Rámec se používá pro opakování s návratem (*Go-Back-N ARQ*), které zefektivní proces tím, že ještě před vypršením časovače je odesílatel rámce informován, jestli je rámec poškozen nebo ztracen. *N(R)* bity odpovídají negativnímu potvrzovacímu číslu [9].
- **Selektivní odmítnutí (*SREJ*)** – V případě, že jsou bity „kódu“ 11, jedná se o kontrolní rámec typu SREJ (*Selective Reject*). Rámec je typu NAK a je použit pro selektivní opakování. *N(R)* bity odpovídají negativnímu potvrzovacímu číslu.

2.3.3 Řídící pole pro nečíslované rámce

Nečíslované rámce jsou používány pro vyměňování informací nutných pro správu relací a kontrolních informací mezi spojenými zařízeními. Na rozdíl od řídících rámců, nečíslované rámce obsahují informační pole. To, ale není použito pro data uživatelů, nýbrž je použito pro informace nutné k řízení systému. Stejně jako u kontrolních rámců, je velká část přenášených informací nečíslovanými rámci obsažena v kontrolním poli. Pomocí pěti bitů tzv. „kódů“ lze vytvořit až 32 různých druhů nečíslovaného rámce. Rozložení těchto bitů je zobrazeno na Obr. 2.2 [9].

2.4 HDLC DLMS/COSEM

Tato sekce se zabývá specifikací DLMS/COSEM a využití protokolu HDLC v této specifikaci. První část této kapitoly se věnuje obecnému popisu specifikace DLMS/COSEM a jejich základních vlastností. Následně je popsán základ komunikačních profilů a poslední část této kapitoly se věnuje rozdílům v koncepci komunikačních profilů založené na modelu TCP-UDP/IP a koncepci založené pouze na protokolu HDLC [12, 13].

2.4.1 Specifikace DLMS/COSEM obecně

Specifikace DLMS/COSEM určuje model rozhraní a komunikační protokoly, které jsou použity při výměně dat mezi měřícím vybavením. Model rozhraní poskytuje přehled o funkčnosti a dostupnosti měrného zařízení. Komunikační protokoly definují, jak se k datům přistupuje a jakým způsobem se přenášejí [12].

Aplikační vrstva DLMS/COSEM specifikuje služby používané pro navázání spojení mezi klientem a serverem a služby pro přístup k atributům a metodám COSEM objektů. Specifické profily DLMS/COSEM komunikačních médií určují, jak mohou být zprávy aplikační vrstvy přepravovány přes různá komunikační média. Každý profil specifikuje sadu protokolů, které jsou uspořádány do různých vrstev ISO/OSI a podporují DLMS/COSEM specifikaci umístěnou v nejvyšší vrstvě [12].

Rozsáhlé zavádění inteligentních měřicích systémů vyžaduje odolné mechanismy využívané sloužící k zabezpečení informací a k ochraně soukromí odběratelů energie. DLMS/COSEM poskytuje předem připravené bezpečnostní mechanismy. Mezi tyto mechanismy patří identifikace a autentizace klientů a serverů, specifická přístupová práva k atributům a metodám COSEM objektů a zašifrovaná APDU (*Application Protocol Data Unit*) data pro ochranu přenášených zpráv mezi klientem a serverem [12].

Cílem DLMS/COSEM je stanovit standard pro rozhraní orientované na podnikovou sféru, objektový model pro měřicí zařízení, systémy a služby pro zajištění přístupu k objektům. Komunikační profily používané pro přenos zpráv prostřednictvím různých komunikačních médií jsou taktéž specifikovány. Model objektů COSEM je specifikován v DLMS UA 1000-1 Ed. 12:2014 the „Blue Book“. COSEM objekty poskytují přehled o funkčnosti měřicích přístrojů prostřednictvím jejich komunikačních rozhraní [12, 13].

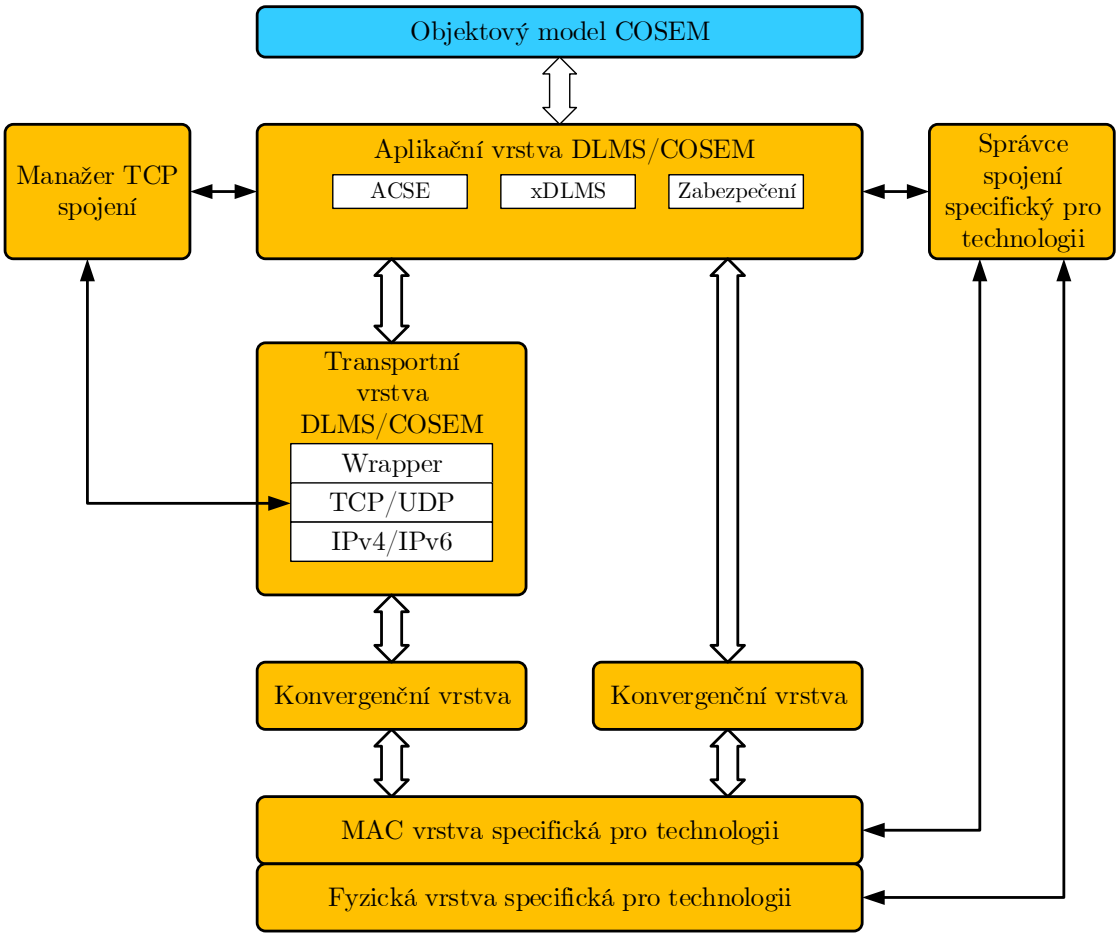
Klíčové charakteristiky výměny dat pomocí DLMS/COSEM jsou následující :

- Měřící přístroje mohou být přístupné pro různé uživatele, kdy kromě klientů se mohou mít k datům přístup i třetí strany.
- Mechanismy pro kontrolu přístupu k prostředkům měřících zařízení jsou k dispozici pomocí aplikační vrstvy DLMS/COSEM a COSEM objektů.
- Bezpečnost a soukromí jsou zajištěny aplikováním kryptografické ochrany na xDLMS (*extended Device Language Message Specification*) zprávy a COSEM data.
- Nízká režie a efektivita je zajištěna různými mechanismy včetně selektivního přístupu, kompaktního kódování a komprese.
- Na místě měření může zaznamenávat data jedno nebo více měřicích zařízení. V případě více měřících zařízení může být k dispozici pouze jediný přístupový bod.

- Výměna dat probíhá buď vzdáleně nebo lokálně. V závislosti na možnostech měřícího zařízení, lokální a vzdálená výměna dat může být prováděna současně.

2.4.2 Komunikační profily

Komunikační profily specifikují jak aplikační vrstva DLMS/COSEM a COSEM modely jsou podporovány zvolenými nižšími vrstvami. Komunikační profily se sestávají z více vrstev ISO/OSI. Každá vrstva má svůj specifický úkol, kdy k jeho plnění využívá služeb nižších vrstev a zároveň poskytuje své služby vrstvám vyšším. Klient a server používají služby nejvyšší vrstvy (aplikační vrstva DLMS/COSEM). Tato vrstva jediná obsahuje specifické COSEM objekty. Počet nižších vrstev závisí na typu použitého přenosového média [12].



Obr. 2.3: Generický komunikační profil DLMS/COSEM [12].

Na Obr. 2.3 je zobrazen generický komunikační profil, který obsahuje:

- Objektový model COSEM modelující aplikační procesy. Pro každou komunikační technologii jsou specifikována specifická rozhraní.
- Aplikační vrstvu DLMS/COSEM.
- Transportní vrstvu DLMS/COSEM zahrnutou v profilech podporující internet.
- Konvergenční vrstvu pojící MAC (*Media Access Control*) vrstvu s aplikační vrstvou DLMS/COSEM a to buď přímo nebo skrze transportní vrstvu.
- MAC vrstvu specifickou pro technologii.
- Komunikační manažery.

Fyzické zařízení může podporovat více komunikačních profilů, které umožňují výměnu dat různými komunikačními médii. V těchto případech je úkolem klientské strany aplikačního procesu rozhodnout, který komunikační profil je použit.

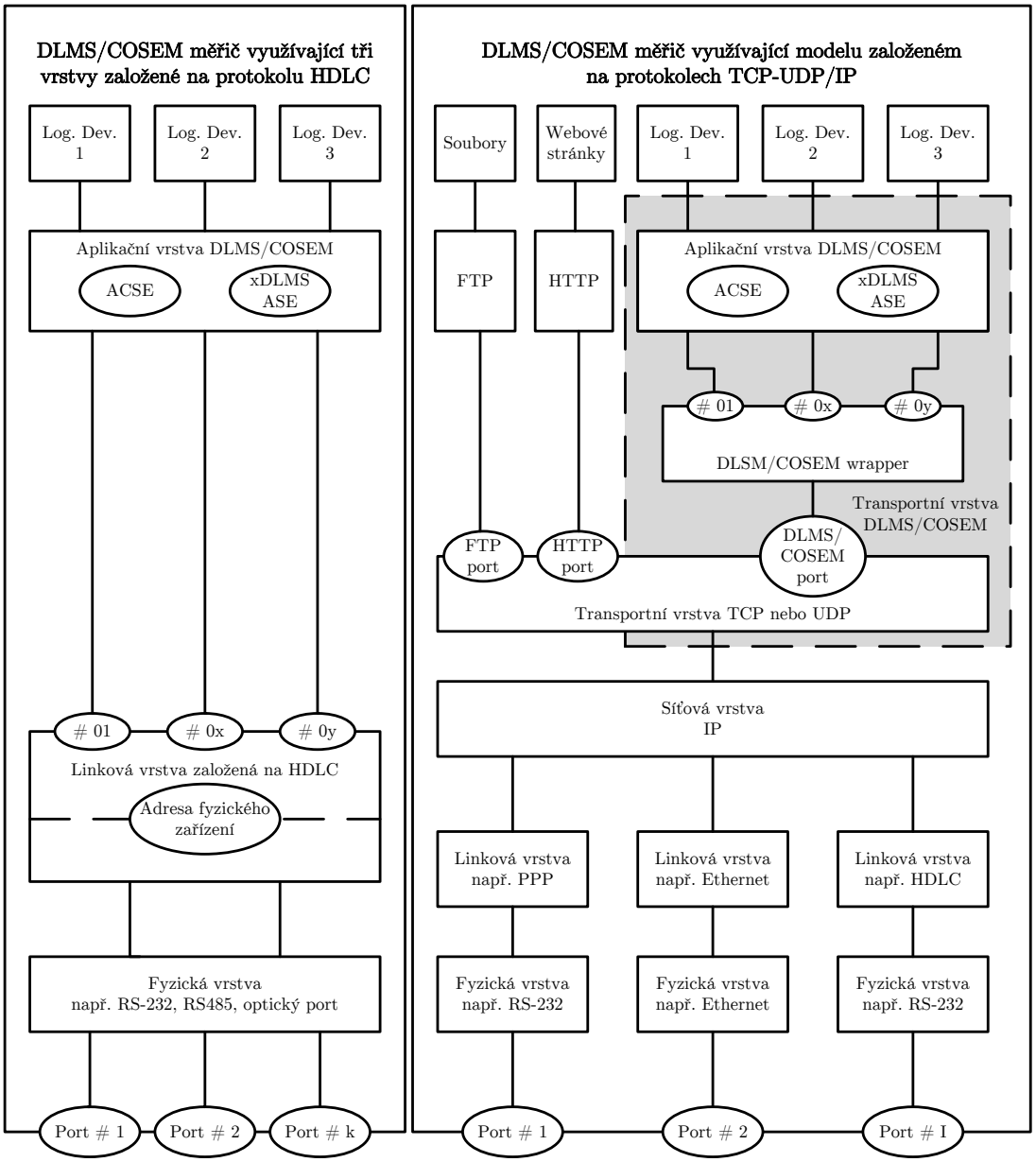
2.4.3 Komunikační profil DLMS/COSEM využívající HDLC

Na Obr. 2.4 je uvedeno srovnání dvou různých komunikačních profilů DLMS/COSEM serverů. První z nich využívá 3 vrstvy a je založen na protokolu HDLC. Druhý profil je založený na TCP-UDP/IP modelu.

Měřící zařízení na levé straně Obr. 2.4 obsahuje „n“ logických jednotek a tvoří i 3 vrství komunikační profil podporující HDLC. Aplikační vrstva DLMS/COSEM je podporována linkovou vrstvou založenou na protokolu HDLC. Její hlavní rolí je poskytovat spolehlivý přenos dat mezi jednotlivými vrstvami. Další úlohou této vrstvy je adresování mezi logickými jednotkami. Řídící logická jednotka má vždy adresu 0x01. K vytvoření místní sítě umožňující přístup k několika měřícím zařízení přes jeden přístupový bod je potřeba, aby měřící zařízení mělo přiřazenu další adresu. Tato adresa se nazývá fyzická adresa a je zprostředkována linkovou vrstvou. Pro úplnost adresa logické jednotky bývá označována jako vrchní HDLC adresa a fyzická adresa jednotky se nazývá nižší HDLC adresa [12].

Fyzická vrstva zajišťuje podporu linkové vrstvy sériovým přenosem bitů mezi fyzickými zařízeními hostujícími klientské a serverové aplikace. Díky tomu mohou různá rozhraní jako je například RS 232 a RS 485 přenášet data skrze různé sítě [12].

Měřící zařízení na pravé straně Obr 2.4 obsahuje „m“ logických jednotek a obsahuje komunikační profil využívající protokolů TCP-UDP/IP. Aplikační vrstva DLMS/COSEM je podporována transportní vrstvou DLMS/COSEM, která zahrnuje TCP nebo UDP (*User Datagram Protocol*) protokol a wrapper. Hlavní úlohou wrapperu je adaptovat servisní sadu ISO/OSI poskytovanou transportní vrstvou



Obr. 2.4: Porovnání dvou komunikačních profilů DLMS/COSEM využívajících HDLC [12].

DLMS/COSEM k volání funkcí TCP a UDP. Jeho další úlohou je přiřazování adres logickým jednotkám, které dále přiřazuje/asociuje na jednotlivé porty. Poslední úlohou wrapperu je poskytování informace o délce přenesené datové jednotky, aby mohl uživatel zjistit její konec. Tento proces je nezbytný kvůli povaze protokolu TCP [12, 13].

Skrze wrapper je aplikační vrstva DLMS/COSEM vázána na TCP a UDP porty, které jsou použity DLMS/COSEM aplikacemi. Přítomnost TCP a UDP protokolů

umožňuje začlenění dalších internetových aplikací, jako FTP (*File transfer Protocol*) nebo HTTP (*Hypertext Transfer Protocol*), vázaných na jejich standardní porty. TCP je podporováno síťovou vrstvou pomocí protokolu IP, který je následně podporován další sadou nižších vrstev specifických podle zvoleného přenosového média [12, 13].

3 MOŽNOSTI SIMULACE DATOVÝCH SÍTÍ

V této kapitole jsou popsány základy simulace a tři vybrané simulátory. U každého simulátoru jsou diskutovány jeho dobré a špatné vlastnosti, kdy k komplexnímu srovnání všech tří simulátorů slouží tabulky 3.1 a 3.2. Poslední část této kapitoly se důkladněji věnuje základům simulátoru NS-2 (*Network Simulator-2*).

3.1 Simulátory síťového provozu

Simulace je velmi důležitá pro moderní technologii, jelikož simulace pomocí počítače může být použita pro modelování a studování hypotetických a reálných objektů. Počítačové simulace mohou být využity pro pomoc při modelování velkého rozsahu náročných vědeckých problémů. Při výzkumu komunikace pomocí počítačových sítí je síťová simulace technikou, kde program předpovídaní chování sítě buď výpočtem interakce mezi různými síťovými entitami (síťové prvky, pakety apod.) nebo analyzováním dat z už existující sítě. Chování sítě a různé aplikace nebo služby, které podporuje, lze posléze studovat v testovacích laboratořích. Rozličné atributy simulované sítě mohou být upraveny kontrolovaným způsobem, což pro různé podmínky umožňuje předpovídat chování simulované sítě [14, 15].

Síťové simulace umožňují vědcům testovat návrhy, které se v důsledku vysoké finanční náročnosti nedají testovat pomocí fyzických komponent. To je obzvlášť užitečné při testování nových neověřených síťových protokolů nebo změny už existujících složitých protokolů. V síťovém simulátoru se síťová topologie navrhuje pomocí virtuálních síťových prvků (stanice, rozbočovače, přepínače, směrovače a mobilní stanice). Topologie sítě je tedy pouze soubor parametrů simulace, které lze jednoduše změnit, což umožňuje studium mnoha odlišných druhů síťového provozu [15].

Síťové simulátory mohou být rozděleny pomocí několika parametrů. Jedno z nejzákladnější dělení je rozdělení podle typu uživatelského rozhraní:

- **Grafické aplikace** – Umožňují uživateli jednoduše graficky zobrazit síťovou topologii.
- **Textově založené aplikace** – Většinou poskytují více možností při nastavování parametrů síťové topologie než grafické aplikace.
- **Aplikace zaměřené na programování** – Pracuje programovací strukturu pomocí, které se dají programovat různé složité síťové simulace.

Reálné síťové simulátory jsou v mnoha případech tvořeny kombinací všech výše zmíněných typů. V této kapitole se budeme věnovat třem simulátorům: NS-2, NS-3 (*Network Simulator-3*) a Riverbed.

3.1.1 Simulátor NS-2

NS-2 je simulátor diskrétních událostí zaměřený na síťový výzkum. Poskytuje podporu pro simulaci TCP, směrovacích a multicastových protokolů. NS-2 byl vyvinut pod projektem VINT (*Virtual Inter Network Testbed*) v roce 1995 spojeným úsilím Kalifornské univerzity v Berkeley, informačního institutu Univerzity Jižní Kalifornie, Národní Laboratoře Lawrence Barkeleye a Výzkumného střediska Xerox Palo Alto. NS-2 podporuje velké množství odlišných síťových protokolů jako jsou například: TCP, UDP, CBR (*Constant Bit Rate*), FTP a HTTP. Při práci s NS-2 je využito dvou jazyků a to je C++ a OTcl (*Object Tool command language*) [15, 14].

3.1.2 Simulátor NS-3

NS-3, nástupce simulátoru NS-2, je simulátor diskrétních událostí zaměřený na internetové systémy primárně pro výzkum a výuku. Projekt NS-3 začal v roce 2006 a je volně dostupný simulátor pod licencí GNU GPLv2. Hlavní rozdíly mezi NS-2 a NS-3 jsou [15, 14]:

- **Rozdílné softwarové jádro** – Jádro NS-3 je napsáno pomocí jazyku C++ a rozhraním pro vytváření skriptů Python. NS-2 používá dvojici jazyku C++ a OTcl [15].
- **Důraz na realizmus** – Protokolové entity NS-3 jsou navrženy tak aby kladly mnohem větší důraz na skutečný svět.
- **Integrace softwaru** – NS-3 podporuje zahrnutí většího množství volně dostupného síťového softwaru. To redukuje potřebu přepisovat simulační modely.
- **Podpora virtualizace**

Rozdílům simulátoru NS-2 a NS-3 se podrobněji věnuje podkapitola 3.2.

3.1.3 Simulátor Riverbed

Simulátor Riverbed původně nazývaný OPNET (*Optimized Network Engineering Tool*), byl vydán v roce 1987 jako první komerční simulační nástroj určený pro modelování komunikačních sítí. Poskytuje komplexní vývojářské prostředí pro simulaci a analýzu výkonu komunikačních sítí. Riverbed dokáže simulovat sítě od velikosti jedné lokální sítě LAN (*Local Area Network)* až po globální síť, která obsahuje stovky různých sítí. Nejdůležitější vlastnosti simulátoru Riverbed jsou [16, 17]:

- **Modelovací a simulační cyklus** – Ulehčuje uživateli postup třemi fázemi návrhu. Tyto fáze jsou: návrh komunikační sítě, simulace sítě a analýza výsledků [16, 14].
- **Hierarchické modelování** – Možnost jednoduše analyzovat jednotlivé části simulovaného projektu.

- **Specializace na komunikační sítě** – Podpora už existujících protokolů a jejích další možná modifikace uživatelem.
- **Automatické generování výsledků simulace**

3.2 Srovnání síťových simulátorů

V této sekci budou srovnány tři síťové simulátory NS-2, NS-3 a Riverbed, se zaměřením hlavně na simulátory s volně dostupným kódem (tzv. Open-Source). Tyto simulátory jsou NS-2 a NS-3. Jak bylo zmíněno v minulé podkapitole Riverbed je komerční nástroj. Výhodou simulátorů s volně dostupným kódem je to, že jeho zdrojový kód je volně dostupný, tudíž každý uživatel se může zapojit do opravy případných chyb. Rovněž jeho rozhraní je otevřené pro možnost budoucích vylepšení. Simulátor je pak velmi flexibilní a reflektuje nejnovější vývoj počítačových sítí rychleji než komerční nástroje. Díky tomuto se tyto simulátory výborně uplatňují ve vědeckých a výukových nasazeních [18, 15].

Riverbed jako jediný používá grafické rozhraní pro návrh topologií. NS-2 a NS-3 pracují s textovým rozhraním. Výhoda grafického rozhraní je přehlednost a snadná, rychlá a okamžitá změna topologie. Textové rozhraní má výhodu v tom, že většinou jím lze nastavit více parametrů sítě. Simulátor Riverbed podporuje jediný operační systém Windows. NS-2 a NS-3 podporuje více operačních systémů: Linux, Windows, Mac OS a Free BSD (*Berkeley Standard Distribution*). Všechny popsané rozdíly jsou přehledně uvedeny v Tab. 3.1 [18].

Tab. 3.1: Obecné srovnání síťových simulátorů [18].

Simulátor	**Rozhraní**	**Licence**	**Programovací jazyky**	**Platforma**	**Poslední verze**
NS-2	Textové	Open-source	C++, Otcl	Windows, Linux, Mac OS, Free BSD	NS-2.35
NS-3	Textové	Open-source	C++, Python	Windows, Linux, Mac OS, Free BSD	NS-3.27
Riverbed	Grafické	Komerční	C, C++	Windows	9.6.1

3.2.1 Srovnání simulátorů NS-2 a NS-3

Základní rozdíly mezi simulátorem NS-2 a NS-3 byly popsány už dříve (viz. podkapitola 3.1.2). V této části budou tyto dva simulátory popsány podrobněji pomocí Tab. 3.2.

Tab. 3.2: Srovnání simulátorů NS-2 a NS-3 [18].

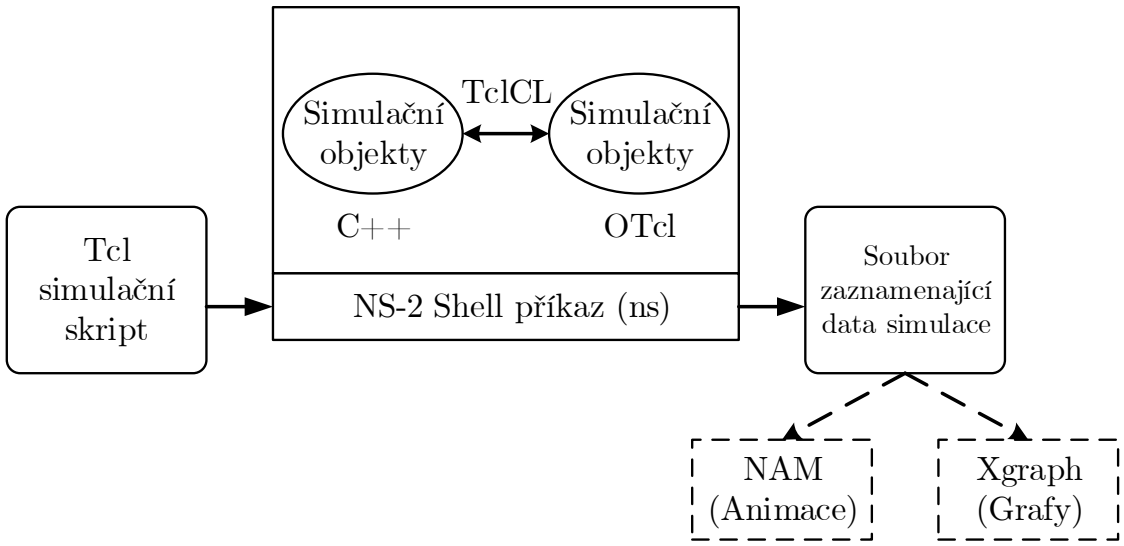
	NS-2	**NS-3**
Začátek vývoje	1995	2006
První vydání	1996	2008
Typ license	Volně dostupný software	Volně dostupný software
Financováno	DARPA VINT,SAMAN a NSF CONSER	NSF CISE a INRIA
Založeno na	NS-1 a simulátoru REAL	NS-2, GTNets, YANS
Současná podpora	Dobrovolníci, USC ISI a webová stránka Sourceforge	Dobrovolníci, NSF, INRIA, GTa a WashU
Architektura	OTcl a C++	C++ a volitelné python skripty
Skriptovací jazyk	Otcl	Python
Vizualizace výsledků	NAM	ns3-viz, pyviz, NetAnim
Správa paměti	NS-2 požaduje základní funkce na správu paměti jazyku C++	Protože NS3 je implementován v jazyce C++, všechny normální C++ funkce správy paměti jako jsou new, delete, malloc a free jsou stále dostupné
Čas simulace	Celkový čas simulace je delší než u NS-3	Celkový čas simulace se škáluje lépe u NS-3 než u NS-2
Kompatibilita	NS-2 je částečně kompatibilní z NS-3 (C++ kód)	NS-3 není zpětně kompatibilní s NS-2
Škálování	Sekvenční simulace	MPI pro distribuované simulace
Pakety	Paket obsahuje dvě odlišné oblasti. První nese záhlaví a druhá údaje o užitečném zatížení	Paket se skládá z jednoho bloku bytů a volitelné sbírky malých značek obsahující data
Přispěné kódy a modely	Mnoho různých organizací přispělo modely a kódy pro NS-2	Oproti NS-2 je počet přispěních modelů a kódů velmi malý
Aplikační vrstva	Ping, vat, telnet, FTP, multicast, generátory síťového provozu	P2P, Ping, generátor síťového provozu, Echo, příjemce dat, čtečky vstupních topologií
Transportní vrstva	TCP, UDP, SCTP, XCP, TFRC, RAP, RTP, PGM, SRM, RLM	Emulace TCP zásobníku (Linux, BSD), DDCP, další vysokorychlostní varianty TCP, UDP
Síťová vrstva	IP, MIP, DV, LS, IPinIP, SR, SRM, MANET: AODV, DSR, DSDV, TORA, IMEP, DiffServ, RED, WFQ, DropTail	Plná podpora IPv4 a IPv6, NAT, BGP, OSPF, RIP, IS-IS, PIM-SM, IGMP/MLD, statický (Dijkstra) unicast, statický multicast, DSDV, Nix-vector, DSR, VANET, Click, MANET: OLSR, AODV
Linková vrstva	ARP, HDLC, GAF, MPLS, LDP, CSMA, 802.11b, 802.15.4, satellite Aloha, Drop Tail, RED, RIO, SRR, WFQ, REM	Nový 802.11 model, Wifi 802.11 linky, Mesh 802.11s, 802.11 varianty (mesh, QoS), WiMAX 802.16, TDMA, CDMA, GPRS, CSMA, PPP, Zigbee, MPLS
Fyzická vrstva	Satelitní opakovač, bezdrátové stínování, energetický model, širokopásmová anténa	IEEE 802 fyzické vrstvy, GSM, Friis, Reyleighovi a Ricianovi zeslabující kanály
Podpora a nástroje	Generátor náhodných čísel, matematická podpora, animace, chybový model, trasování a monitorování	Generátor náhodných čísel, animace, chybový model, trasování a monitorování, logování
Emulační mód	Integrace s reálnými sítěmi	Integrace s reálnými sítěmi

3.3 Úvod do simulátoru NS-2

V následující části je popsán úvod do simulátoru NS-2. Nejprve je popsána architektura simulátoru NS-2, posléze se kapitola věnuje hlavním krokům prováděných při jeho použití.

3.3.1 Základní architektura

Obr. 3.1 zobrazuje základní architekturu simulátoru NS-2. Simulátor NS-2 poskytuje uživateli spustitelný příkaz „`ns`", který přijímá jeden vstupní argument. Tento argument je název skriptovacího Tcl souboru. Ve většině případů je vytvořen soubor zaznamenávající data simulace, která se dále mohou použít pro vytvoření grafů nebo animace [19, 20].



Obr. 3.1: Architektura simulátoru NS-2 [19].

NS-2 obsahuje dva programovací jazyky C++ a OTcl. Zatímco C++ jazyk definuje interní mechanizmy simulace (tzv. *backend*), jazyk OTcl sestavuje a nastavuje objekty a plánuje diskrétní informace (tzv. *frontend*). C++ a OTcl jsou spolu svázány pomocí TclCL. Proměnné v doméně OTcl namapované na C++ objekt jsou nazvány (tzv. *handles*). Konceptuálně handle je pouze řetězec v doméně OTcl a neplní žádnou funkci. Na místo toho funkce (např. přijetí paketu) je definována v namapovaném C++ objektu. V OTcl doméně handle může pracovat jako frontend, který komunikuje s uživateli a ostatními objekty. Může dokonce definovat svoje vlastní procedury a proměnné, které následně zprostředkovávají komunikaci [19].

NS-2 nabízí velké množství vestavěných C++ tříd, které se pomocí Tcl skriptu používají pro sestavení simulací.Pokročilí uživatelé mohou nalézt tyto třídy nedostatečné. Musí proto navrhnout svoje vlastní C++ třídy a použít OTcl konfigurační rozhraní pro spojení objektů vytvořených pomocí těchto tříd [19, 20].

Výstupem simulátoru NS-2 po simulaci jsou textová data. Výsledky se následně mohou graficky vykreslit pomocí nástrojů jako je NAM nebo Xgraph. Pro analyzování konkrétního chování sítě mohou uživatelé vyextrahovat jen potřebná textová data.

3.3.2 Struktura adresáře NS-2

Vysvětlení v dalším textu pracuje s předpokladem, že simulátor NS-2 je nainstalován v adresáři `nsallinone-2.35`. Na Obr. 3.2 je zobrazena stromová struktura tohoto adresáře. Adresář `nsallinone-2.35` je na první úrovni adresářové struktury. Na druhé úrovni se nachází adresář `tclcl-1.20`, který obsahuje třídy TclCL (např. `Tcl`, `TclObject` a `TclClass`). Všechny simulační moduly simulátoru NS-2 jsou umístěny v adresáři `ns-2.35` taktéž na druhé úrovni. Na třetí úrovni jsou moduly v interpretované hierarchii v adresáři `tcl`. Nejčastěji používané moduly v adresáři `tcl` jsou uloženy v adresáři `lib`, který se nachází na čtvrté úrovni [19]. Simulační moduly v kompilované hierarchii jsou zařazeny do adresářů na třetí úrovně. Mezi tyto adresáře patří:

- `tools` – Zahrnuje různé pomocné třídy, jako je například generátor náhodných čísel [20].
- `common` – Obsahuje základní moduly pro práci s pakety jako je simulátor, plánovač nebo konektor.
- `queue` – Obsahuje moduly pro obsluhu front.
- `tcp` – Obsahuje moduly protokolu TCP.
- `trace` – Obsahuje moduly používané ke trasování.

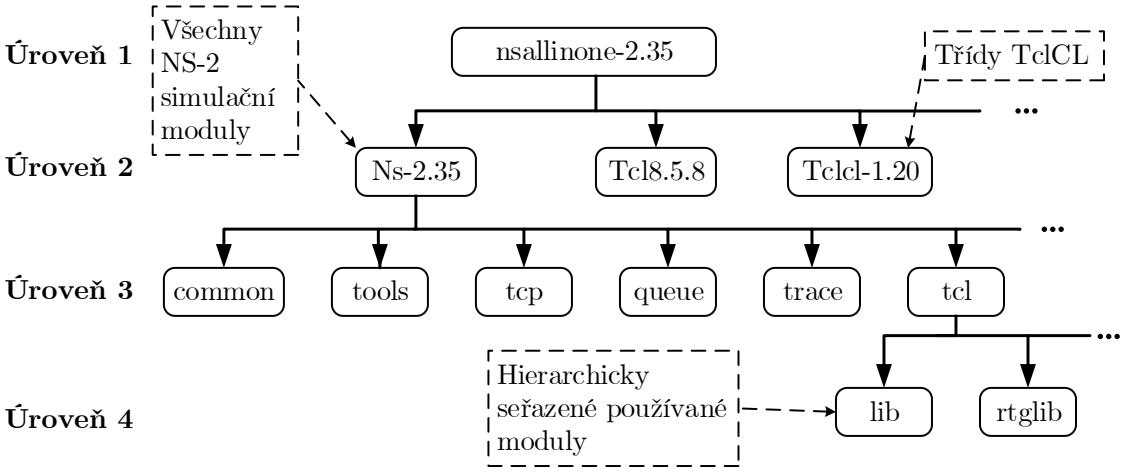
3.3.3 Spuštění simulace NS-2

Tato sekce se věnuje invokaci simulátoru NS-2 a popisu jednotlivých kroků simulačního procesu.

Vyvolání simulátoru NS-2

Po instalaci je v domovském adresáři simulátoru NS-2 vytvořen spustitelný soubor „`ns`". NS-2 může být posléze vyvolán příkazem zadaným v příkazovém řádku:

```
»ns [<file>] [<args>]
```



Obr. 3.2: Struktura adresáře simulátoru NS-2 [19].

kde `<file>` a `<args>` jsou volitelné vstupní argumenty. Pokud není zadán žádný argument, příkaz vyvolá pracovní prostředí NS-2, kde NS-2 čeká na interpretování příkazů zadaných pomocí klávesnice. Pokud je zadán první argument `<file>`, NS-2 provede vstupní skript `<file>` na základě Tcl syntaxe programu [19].

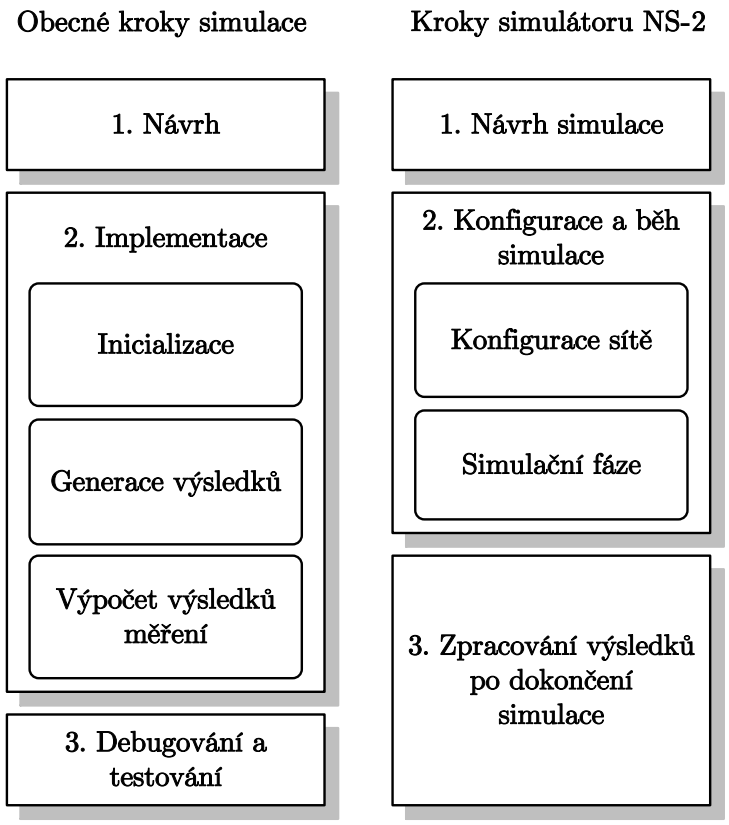
Hlavní kroky simulace NS-2

Na Obr. 3.3 jsou názorně zobrazeny rozdíly mezi obecnou simulací a simulací vytvořenou v simulátoru NS-2. Z obrázku je patrné, že obecný návrh může být upraven tak, aby vyhovoval simulátoru NS-2. Hlavní kroky simulace v simulátoru NS-2 jsou:

1. **Návrh simulace** – První krok v simulaci sítě je zaměřen na návrh simulace. V tomto kroku by uživatel zadává/určuje smysl simulace, rozvržení sítě, měřené veličiny a typ výsledku.
2. **Konfigurace a běh simulace** – Tento krok implementuje návrh z prvního kroku. Je rozdělen do dvou částí:
   - **Konfigurace sítě** – V této fázi jsou vytvořeny a nakonfigurovány síťové komponenty (např. TCP klient, UDP klient a uzly) na základě návrhu z prvního kroku. Zároveň se zde nastavuje kdy mají být spuštěny události v sítí (např. přenos dat).
   - **Provedení simulace** – Tato fáze spustí simulaci, která byla v předchozí fázi nakonfigurována. Časově plánuje simulace a provádí události chronologicky. Fáze většinou trvá dokud simulační hodiny nedosáhnou hranici stanovenou v nastavení.

   Ve většině případů je efektivní nadefinovat simulační scénář ve souboru Tcl (tzv. `<file>`) a simulovat tento scénář pomocí příkazu „`ns <file>`".

3. **Zpracování výsledků po dokončení simulace** – Trasovací soubor programu NS-2 detailně zaznamenává pohyb paketů v simulované síti. NS-2 poskytuje dva typy trasovacího souboru: textový a NAM soubor.



Obr. 3.3: Srovnání obecných kroků simulace a kroků simulátoru NS-2 [19].

Textově založené trasování

Textově založené trasování zaznamenává detaily paketů procházející skrze uzly a fronty. Tab. 3.3 zobrazuje strukturu jednoho řádku trasovacího souboru kabelové sítě. Každý řádek obsahuje 12 sloupců. Sloupec identifikátoru typu paketu odkazuje na čtyři možné události, které mohou paket při přenosu potkat:

- `r` – Přijetí.
- `+` – Zařazení do fronty.
- `-` – Opuštění fronty.
- `d` – Zahození.

Sloupec času zaznamenává čas uskutečnění události. Sloupce tři a čtyři jsou zdrojový a cílový uzel linky, kde se událost uskutečnila. Další sloupec je série značek

Tab. 3.3: Formát řádků trasovacího souboru kabelových linek

ID typu	Čas	Zdrojový uzel	Cílový uzel	Název paketu	Velikost paketu	Značky	ID Toku	Zdrojová adresa	Cílová adresa	Sekvenční číslo	Unikátní ID paketu

indikující abnormální chování. Pokud má zaznamenaný výstup v sedmém sloupci následují tvar „----", nebyla zaznamenáno žádné abnormální chování případně vyjádřené značkou. Následující sloupec zaznamenává identifikátor toku. Devátý a desátý sloupec označují zdrojovou a cílovou adresu ve formě `uzel.port`. Správnost složení zprávy u koncové stanice NS-2 je zaručena záznamem sekvenčního čísla (předposlední sloupec souboru). Poslední sloupec označuje unikátní identifikátor paketů. Při trasování satelitních linek řádek obsahuje o dva sloupce navíc, které zaznamenávají zeměpisnou délku a šířku daného cílového uzlu [20].

Trasovací soubor nemá žádnou výpovědní hodnotu dokud není provedená smysluplná analýza zaměřená na některý ukazatel výkonu sítě. Většinou ze trasovacího souboru je vyjmut podstatný blok dat, který nás zajímá a ten se dále analyzuje. Například pro výpočet průměrné propustnosti vybrané linky se používají pouze řádky asociované s vybranou linkou podle sloupců 3 a 4. Dva nejvíce používané programovací jazyky pro extrakci dat jsou AWK a Perl.

Textově založené trasování je spuštěno pomocí příkazu „`$ns trace-all $file`" kde `ns` je handle simulátoru a `file` je handle asociována se trasovacím souborem. Tento příkaz informuje NS-2, že má trasovat všechny pakety. Po zadaní tohoto příkladu a vytvoření objektu je současně vytvořen trasovací objekt, který shromažďuje informace o procházejících paketech. Proto musí být příkaz „`trace-all`" zadán před vytvářením objektů souvisejících s přenosem [19].

Trasování pomocí NAM

Trasovací soubor NAM ukládá detaily do textového souboru a ten následně používá pro vytvoření grafické simulace. Trasovací soubor NAM je aktivován pomocí příkazu „`$ns namtrace-all $file`" kde `ns` je handle simulátoru a `file` je handle asociována se trasovacím souborem NAM. Po získání trasovacího souboru NAM muže být animace spuštěna pomocí příkazu „`$ns namtrace-all $file`" [19, 20].

4 IMPLEMENTACE PROTOKOLU HDLC V SIMULÁTORU NS-2

Tato kapitola je zaměřená na praktickou implementaci protokolu HDLC v simulátoru NS-2. Nejdříve jsou vysvětleny skutečnosti, potřebné pro správnou implementaci HDLC protokolu. Následně je popsána původní implementace protokolu HDLC na satelitních linkách.

4.1 Síťový model v simulátoru NS-2

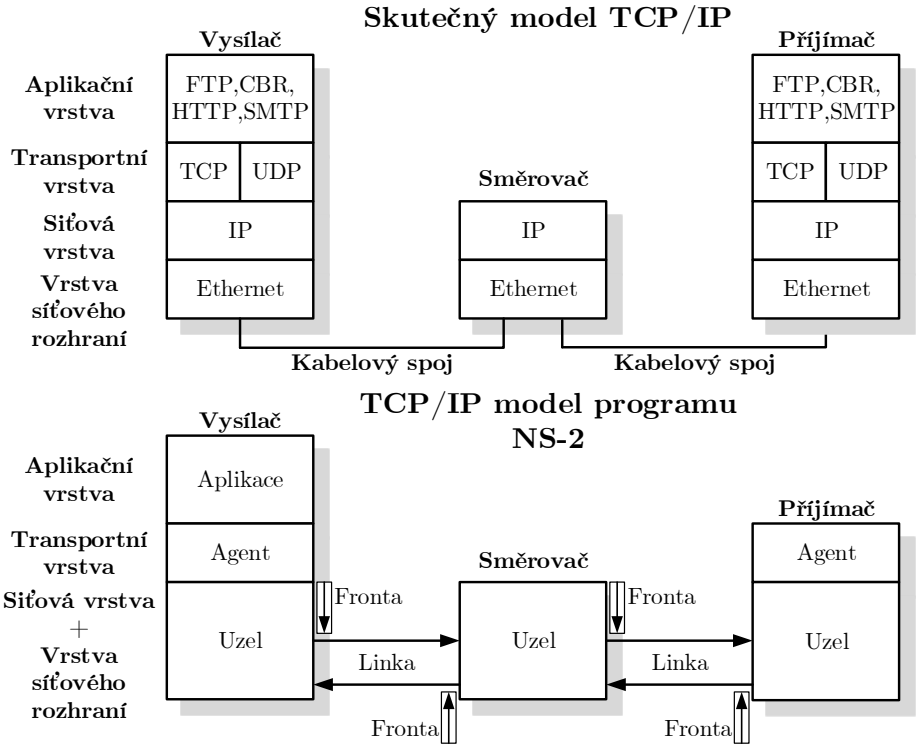
První důležitá část simulátoru NS-2, která musí být vysvětlena je jím používaný síťový model a jeho srovnání se skutečným modelem TCP/IP. Na Obr. 4.1 je uvedeno srovnání modelu TCP/IP s modelem použitým v simulátoru NS-2, kdy lze pozorovat několik zásadních rozdílů, které jsou diskutovány dál v textu [19, 20].

Aplikační vrstvu v simulátoru NS-2 zprostředkovávají různě definované aplikace jako je například FTP, Telnet a generátor síťového provozu. Transportní vrstva je realizována agentem, který vytváří pakety podle pokynů přicházejících z aplikace. Na rozdíl od klasického modelu TCP/IP není u simulátoru NS-2 od sebe oddělena síťová vrstva a vrstvy síťového rozhraní. Tyto vrstvy jsou u simulátoru NS-2 zprostředkovány uzlem. Vytvořený síťový provoz jde z uzlu do fronty a z této fronty pak putuje přes linku k příjemci. Uzlu se podrobněji věnuje podkapitola 4.2. Druhá významná odlišnost síťového modelu simulátoru NS-2 spočívá v přijímači, který v tomto případě nemusí simulovat příjem paketů. Přijímač proto neobsahuje aplikační vrstvu a pakety jsou v něm rovnou ničeny. Detaily ohledně přenosu a vytváření síťového provozu pomocí protokolu TCP jsou uvedeny níže [19].

4.1.1 Mechanismy přenosu dat

V minulé sekci bylo lehce nastíněno, jak v simulátoru NS-2 probíhá vytváření síťového přenosu. Tento základní popis je pro upravení protokolu HDLC nedostatečný, proto se tato sekce podrobně věnuje mechanismu přenosu dat. Mechanismus přenosu dat je názorně vysvětlen na příkladu, který je uveden na Obr. 4.2. Lze na něm pozorovat mechanismus přenosu dat v simulátoru NS-2 využívající transportní protokol TCP a aplikační protokol FTP.

Proces začíná když aplikace (`FTP`) informuje TCP vysílač (`TCPAgent`) o uživatelském požadavku pro vysílaní dat voláním funkce `sendmsg(nbytes)` třídy `TcpAgent`. TCP vysílač vytvoří paket a pošle jej směrem dolů na zvolený objekt pomocí funkce `target_->recv(p,h)`. Nižší vrstvy sítě následně doručí paket do cílového uzlu,



Obr. 4.1: Srovnání síťového modelu klasické sítě a simulátoru NS-2 [19].

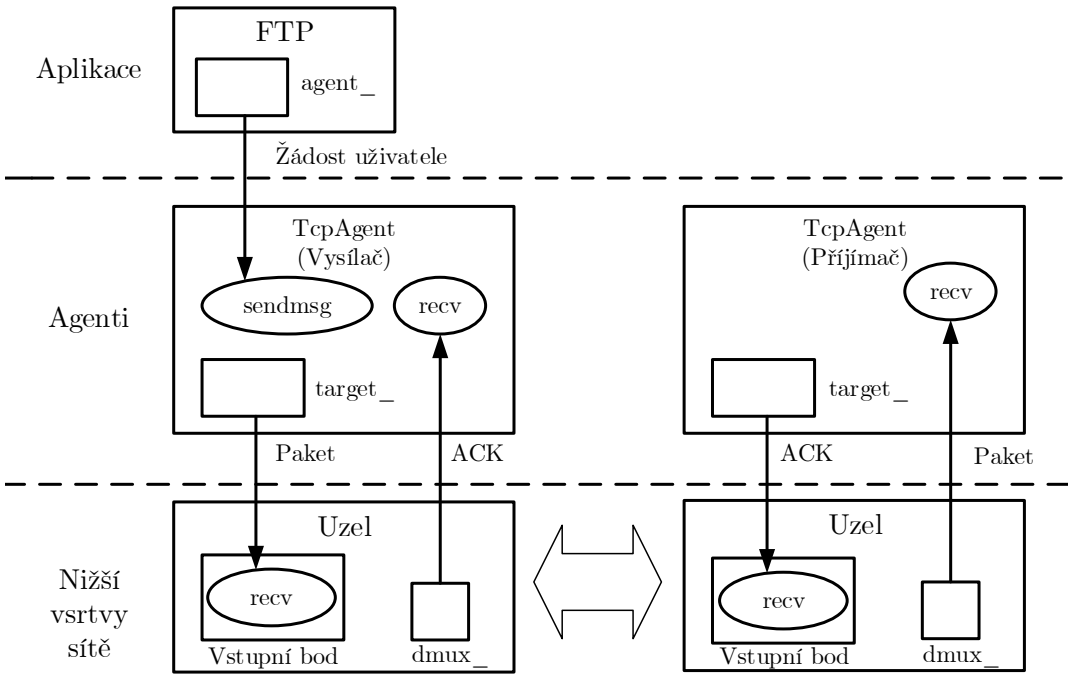
který je vázán na TCP přijímač (`TcpSink`). Cílový uzel odešle paket do TCP přijímače,který je nastavený v de-multiplexeru `dmux_` invokací funkce `recv(p,h)`. Po přijetí paketu TCP přijímač vytvoří ACK paket a odešle jej zpátky TCP vysílači pomocí funkce `target_->recv(p,h)`, kde `p` je ukazatel na vytvořený ACK paket. Nižší vrstvy sítě doručí tento ACK paket vysílajícímu uzlu, který pošle ACK paket TCP vysílači pomocí de-multiplexeru [19, 21].

Pokud je TCP paket nebo ACK paket ztracen (nebo zdržen na dlouhou dobu) TCP vysílač předpokládá, že je paket ztracen. V tomto případě TCP vysílač paket znovu vytvoří a následně podle stejného principu přenese znovu.

4.1.2 Adresace

Adresace v simulátoru NS-2 je velmi odlišná od standardní adresace v síťovém modelu TCP/IP. Adresní prostor v simulátoru NS-2 existuje jako souvislé pole `n` bitů, kde `n` se může lišit podle požadavků simulace. Typicky má `n` velikost 31 bitů, což je i jeho výchozí a maximální hodnota. Obě tyto hodnoty jsou definovány ve složce: ~ns//tcl/lib/ns-default.tcl [20].

V simulátoru NS-2 existuji dva typy adresování: výchozí a hierarchické. U hierarchického adresování jsou rozlišovány další dva typy [20].



Obr. 4.2: Diagram přenosu dat mezi jednotlivými vrstvami u protokolu TCP [19].

Výchozí adresování
Výchozí nastavení alokuje 31 nižších bitů pro identifikátor portu, 1 vyšší bit pro multicast a zbývajících 30 vyšších bitů pro identifikátor uzlu. Pokud není jinak explicitně zadáno, výchozí nastavení je vždy nastaveno při vytvoření simulace [20].

Hierarchické adresování V simulátoru NS-2 existují dvě možnosti nastavení hierarchického adresování:

- **Výchozí hierarchické nastavení** – Velikost identifikátoru portu zůstává stejná (31 bitů), ale identifikátor uzlu je rozdělen do tří úrovní s rozdělením bitů (9 11 11). Hierarchické nastavení může být nastaveno příkazem `$ns set-address-format hierarchical`. Pokud je nastaven multicast, jednotlivé úrovně rozděleny na (8 11 11) bitů [20].
- **Specifické hierarchické nastavení** – Pomocí tohoto nastavení lze nastavit u identifikátoru uzlu kolik bitů bude obsahovat každá úroveň.

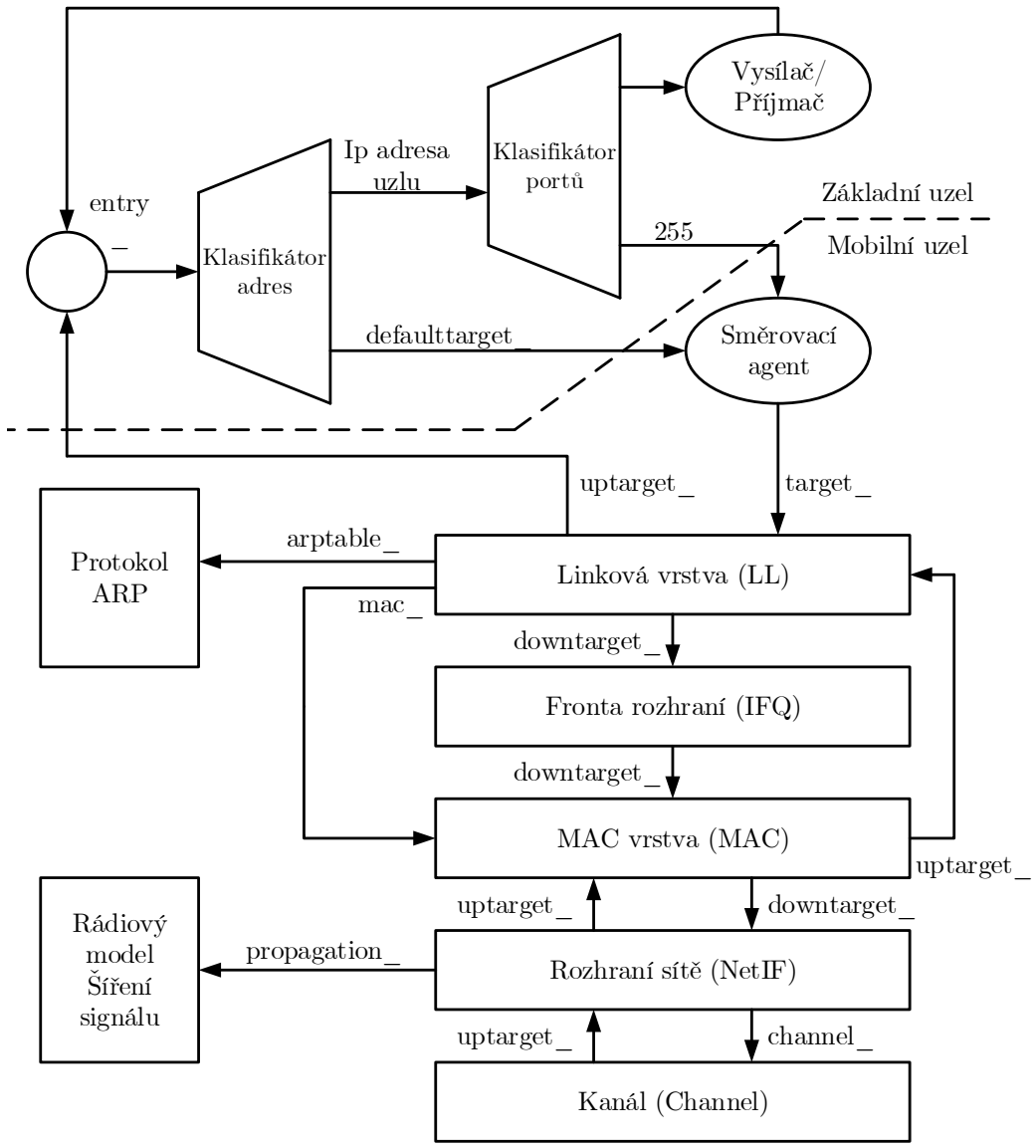
4.2 Uzel

V simulátoru NS-2 uzel zastává funkci dvou vrstev síťového modelu a to síťové vrstvy a vrstvy síťového rozhraní. Jedná se o hlavní prvek, který realizuje síťový provoz. V simulátoru NS-2 existuje více druhů uzlů. Základním typem je `RegularNode`.

Z tohoto uzlu následně vychází další dva uzly, kdy jeden je určený pro satelitní přenos a druhý pro bezdrátovou komunikaci. Oproti základním typu uzlu jsou tyto uzly rozšířeny o další vlastnosti, které umožňují simulovat žádaný přenos. Konstrukce uzlu pro satelitní přenos a uzlu pro mobilní komunikaci jsou si velmi podobné [19, 21].

4.2.1 Architektura uzlu

Pro vysvětlení architektury uzlů bude zvolen mobilní uzel, který obsahuje spolu s částí základního uzlu i rozšiřující část. Obr. 4.3 ukazuje jeho kompletní architekturu.



Obr. 4.3: Schéma mobilního uzlu [19].

4.2.2 Část základního uzlu

Část základního uzlu obsahuje dvě hlavní komponenty: klasifikátor adres (instvar `classifier_`) a klasifikátor portů (instvar `dmux_`). Obě komponenty mají jeden vstupní bod a několik směrovacích cílů. Klasifikátor adres slouží jako router, který obdrží paket z vyšší vrstvy a pošle ho do směrovacího agenta, který s přiloženou adresou v záhlaví paketu, pošle paket do nižších vrstev. Klasifikátor portů funguje jako most transportní vrstvy, který přijímá pakety z klasifikátoru adres (v tomto případě jsou pakety určeny tomuto konkrétnímu uzlu) a posílá je dál určitému agentovi, který je s ním svázán [19, 20].

4.2.3 Rozšířená část mobilního uzlu

Tato část je rozšířením regulárního uzlu za účelem podrobnějšího simulování síťových technologií. Zahrnuje všechny komponenty v dolní části obr. 4.3. Každý komponent je specifikován určitými možnostmi, kdy nejdůležitější volby jejich nastavení jsou jak pro mobilní, tak pro satelitní uzel uvedeny v tabulce 4.1. Všechny možnosti nastavení se dají dohledat v souboru „~ns/tcl/lib/ns-lib.tcl.“ Rozšířená část uzlu bude podrobněji popsána při rozboru LAN uzlu v kapitole 5 [19, 20].

Tab. 4.1: Možnosti nastavení mobilních a satelitních uzlů [20].

Možnost	Dostupné hodnoty	Výchozí nastavení
Obecné nastavení		
Typ adresy	výchozí, hierarchické	výchozí
MPLS	ON, OFF	OFF
Nastavení pro satelitní i mobilní uzly		
wiredRouting	ON, OFF	OFF
Typ ll	LL, LL/Sat, LL/Sat-hdlc	
Typ MAC	Mac/802_11, Mac/Csma/Ca, Mac/Sat, Mac/Sat/UnslottedAloha, Mac/Tdma	
Typ ifq	Queue/DropTail, Queue/DropTail/PrQueue	
Typ phy	Phy/WirelessPhy, Phy/Sat	

4.3 Paket v simulátoru NS-2

Obecně paket obsahuje záhlaví a přiložená data. Záhlaví paketu nese atributy (např. zdrojovou a cílovou IP adresu) nezbytné pro přenos a přiložené data, které přenášejí

data uživatelů. I když je tento princip v praxi typický, program NS-2 modeluje pakety odlišně.

Ve většině případech NS-2 získává informace z přiložených dat a tyto informace ukládá do záhlaví paketů. Tím odpadá nutnost zpracovávat data při běhu simulátoru. Například místo počítaní počtu bitů v paketu, NS-2 ukládá velikost paketu do proměnné `hdr_cmn::size_`. S touto proměnnou pak pracuje při simulaci [19].

4.3.1 Architektura paketu

Obr. 4.4 zobrazuje architekturu paketového modelu v simulátoru NS-2, který je složený ze 4 hlavních částí:

- **Skutečný paket**: Skutečný paket označuje část paměti,ve které je uloženo záhlaví paketu a uživatelská data. Simulátor NS-2 nepřistupuje přímo do této paměti, ale využívá proměnné „`bits_`“ a „`data_`“ třídy `Packet` k přístupu k záhlaví paketu a uživatelským datům.
- **Třída `Packet`**: Hlavní třída reprezentující pakety. Její deklarace je zobrazena na kódu 4.1. Obsahuje tyto proměnné a funkce [19]:
  - **C++ proměnné třídy `Packet`**

    `bits_` String obsahující záhlaví paketu.

    `data_` Ukazatel na objekt `AppData`, který obsahuje uživatelsk8 data.

    `fflag_` Nastaven jako `true` pokud je paket referován ostatními objekty, jinak je nastaven jako `false`.

    `free_` Ukazatel na začátek listu volných paketů.

    `ref_count_` Počet objektů, které právě odkazují na paket.

    `next_` Ukazatel na další paket v listu paketů.

    `hdr_len_` Délka záhlaví paketu.
  - **C++ funkce třídy `Packet`**

    `init(p)` Vymazání záhlaví `*bits_` vstupního paketu `*p`.

    `copy()` Vrácení ukazatele na duplikovaný paket.

    `refcopy()` Zvýšení počtu objektů referujících na paket o jeden.

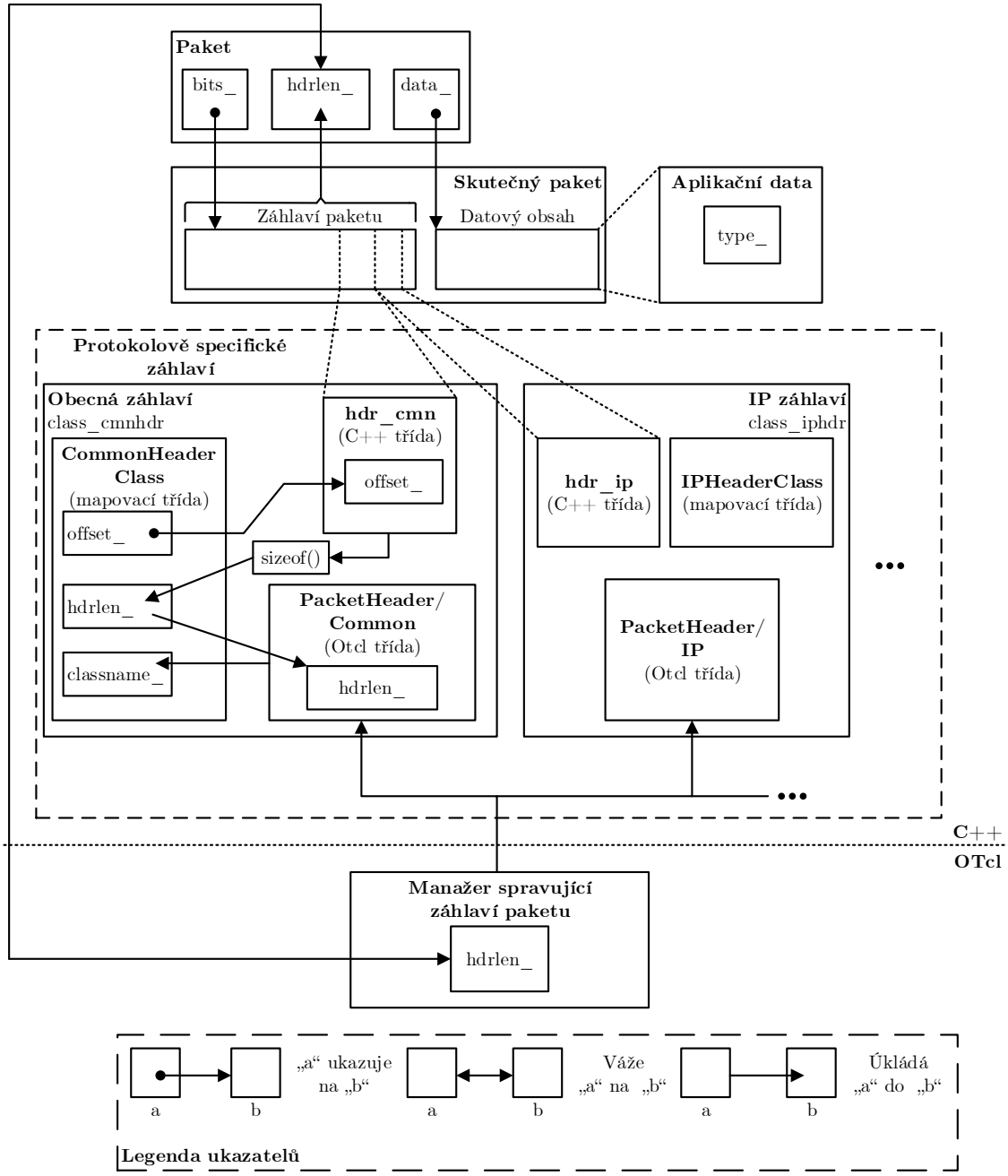
    `alloc()` Vytvoří paket a na něj ukazatel.

    `alloc(n)` Vytvoří paket s „`n`“ byty uživatelských dat a vrátí ukazatel na vytvořený paket.

    `allocdata(n)` Alokuje „`n`“ bytů dat do proměnné `data_`.

    `free(p)` Uvolní paket „`p`“.

    `access(off)` Získá referenci na specifický bod proměnné „`bits_`“.
- **Protokolově specifické záhlaví**: Jak je znázorněno na Obr. 4.4 záhlaví paketu skládá z několika protokolově specifických záhlaví. Každé z nich používá pro uložení svých atributů spojitou část záhlaví paketu. Stejně jako u většiny



Obr. 4.4: Modelování paketu v NS-2 [19].

OTcl objektů, existují tři třídy související s daným záhlavím [19]:

- C++ třída (např. `hdr_cmn`) zprostředkuje strukturu pro uložení atributů paketu.
- OTcl třída (např. `PacketHeader/Common`) sloužící jako rozhraní do OTcl domény. NS-2 používá tuto třídu pro konfiguraci záhlaví paketu z OTcl domény.

– Mapovací třída (např. *CommonHeaderClass*) spojuje C++ třídu s OTcl třídou.

Jednotlivá protokolově specifická záhlaví budou probrány v sekci 4.3.2.

- **Manažer spravující záhlaví paketu**: Manažer spravující záhlaví paketu udržuje list aktivních protokolů a konfiguruje všechny aktivní protokolově specifická záhlaví k nastavení správného záhlaví paketu [19].

Výpis 4.1: Deklarace třídy `Packet`

```
1  class Packet : public Event {
2  private:
3    unsigned char* bits_;
4    AppData* data_;
5    static void init(Packet*);
6    bool fflag_;
7  protected:
8    static Packet* free_;
9    int ref_count_;
10 public:
11   Packet* next_;
12   static int hdrlen_;
13
14   // Alokace a delokace paketů
15   Packet() : bits_(0), data_(0), ref_count_(0), next_(0) { }
16   inline unsigned char* bits() { return (bits_); }
17   inline Packet* copy() const;
18   inline Packet* refcopy() { ++ref_count_; return this; }
19   inline int& ref_count() { return (ref_count_); }
20   static inline Packet* alloc();
21   static inline Packet* alloc(int);
22   inline void allocdata(int);
23   inline void initdata() { data_  = 0;}
24   static inline void free(Packet*);
25
26   // Přístup k paketu
27   inline unsigned char* access(int off) const {
28     if (off < 0)
29       abort();
30     return (&bits_[off]);
31   }
```

4.3.2 Záhlaví paketu

Jako část paketu, záhlaví obsahuje atributy nutné k přenosu informací jako je například unikátní identifikátor paketu a IP adresy. V praxi paket obsahuje pouze relevantní protokolové záhlaví. Paket v simulátoru NS-2 naopak obsahuje všechna protokolově specifická záhlaví bez ohledu na jeho typ. Záhlaví každého paketu používá stejné množství paměti. To je uloženo v proměnné „`hdrlen_`“ třídy `Packet` (viz. 12 řádek kódu 4.1) a nemá žádný vztah k velikosti paketu při simulaci. Například TCP a UDP pakety mohou mít odlišné velikosti. Hodnoty uložené v korespondující

proměnné `hdr_cmn::size_` mohou být odlišné, ale hodnoty uložené v proměnné `Packet::hdrlen_` jsou pak pro TCP i UCP stejné [19, 20, 21].

4.3.2.1 Obecné záhlaví paketu

Obecné záhlaví paketu obsahuje atributy paketu, které jsou běžné pro všechny druhy paketů. K indikaci způsobu uložení atributů se používá `struct hdr_cmn`. Zjednodušená struktura tohoto datového typu je zobrazena na výpisu kódu 4.2. Mezi důležité proměnné `hdr_cmn` patří:

`ptype_` Typ přenášených dat.

`size_` Velikost paketu v bytech. Během simulace NS-2 používá tuto proměnnou jako velikost paketu.

`uid_` Identifikátor unikátní pro každý jednotlivý paket.

`dir_t` Směr, kterým se paket pohybuje. Paket se může pohybovat „`UP`“ (`dir_t = 1`) směrem k vyšším vrstvám nebo „`DOWN`“ (`dir_t = −1`) směrem k nižším vrstvám. Výchozí nastavení `dir_t` je „`DOWN`“.

Výpis 4.2: Datový typ `struct hdr_cmn`

```
1  struct hdr_cmn {
2    enum dir_t { DOWN= -1, NONE= 0, UP= 1 };
3    packet_t ptype_;
4    int size_;
5    int uid_;
6    int iface_;
7    dir_t direction_;
8    static int offset_;
9
10   nsaddr_t prev_hop_;
11   nsaddr_t next_hop_;
12   int       addr_type_;
13   nsaddr_t last_hop_;
14
15   inline static hdr_cmn* access(const Packet* p) {
16     return (hdr_cmn*) p->access(offset_);
17   }
18   inline packet_t& ptype() { return (ptype_); }
19   inline int& size() { return (size_); }
20   inline int& uid() { return (uid_); }
21   inline dir_t& direction() { return (direction_); }
22 };
```

4.3.2.2 IP záhlaví paketu

IP záhlaví paketu je reprezentováno C++ datovým typem `struct hdr_ip`. IP záhlaví paketu obsahuje informace o zdroji a cíli paketu. Kód 4.3 ukazuje část `struct hdr_ip` deklarace. IP záhlaví obsahuje následujících pět hlavních proměnných:

`src_` Adresa zdrojového uzlu paketu

`dst_` Adresa cílového uzlu paketu

`ttl_` Doba života paketu

`fid_` Identifikátor toku paketu

`prio_` Úroveň priority paketu

K uložení adresy uzlu, NS-2 používá datový typ `ns_addr_t` definovaný v souboru ~ns/config.h. Datový typ `struct ns_addr_t` obsahuje dva členy: `addr_` a `port_`. Oba členy jsou typu `int32_t`, což je alias datového typu `int`. Zatímco `addr_` určuje adresu uzlu, `port_` identifikuje připojený port (pokud existuje) [19, 20, 21].

Výpis 4.3: Datový typ `struct hdr_ip`

```
1  struct hdr_ip {
2    ns_addr_t src_;
3    ns_addr_t dst_;
4    int    ttl_;
5    int    fid_;
6    int    prio_;
7    static int offset_;
8    inline static int& offset() { return offset_; }
9    inline static hdr_ip* access(const Packet* p) {
10     return (hdr_ip*) p->access(offset_);
11   }
12
13   ns_addr_t& src() { return (src_); }
14   nsaddr_t& saddr() { return (src_.addr_); }
15   int32_t& sport() { return src_.port_;}
16   ns_addr_t& dst() { return (dst_); }
17   nsaddr_t& daddr() { return (dst_.addr_); }
18   int32_t& dport() { return dst_.port_;}
19   int& ttl() { return (ttl_); }
20   int& flowid() { return (fid_); }
21   int& prio() { return (prio_); }
22 };
```

4.3.2.3 HDLC záhlaví paketu

Reprezentováno C++ datovým typem `struct hdr_hdlc`. HDLC záhlaví paketu obsahuje informace potřebné pro funkci protokolu HDLC, kdy sdílí adresní prostor se záhlavím linkové vrstvy `struct hdr_ll`. Kód 4.3 ukazuje celý datový typ `struct hdr_hdlc`. Důležité proměnné obsažené v HDLC záhlaví jsou:

`saddr_` Zdrojová MAC adresa uzlu

`daddr_` Cílová MAC adresa uzlu

`hdlc_fc_` Kontrolní HDLC rámec

`fc_type_` Typ HDLC rámce

Výpis 4.4: Datový typ `struct hdr_hdlc`

```
1  struct hdr_hdlc {
2    int saddr_;
3    int daddr_;
4    HDLCControlFrame hdlc_fc_;
5    HDLCFrameType fc_type_;
6    int padding_;
7
8    inline int saddr() {return saddr_;}
9    inline int daddr() {return daddr_;}
10 };
```

4.4 HDLC pro satelitní přenos

V této sekci bude probrána implementace protokolu HDLC použitá v programu NS-2. Hlavní důvod, proč byl simulátor NS-2 vybrán, je předem implementovaný protokol HDLC. Ten je v tomto případě navržen pro satelitní přenos. To znamená, že využívá satelitní uzly a C++ třídy spojené se satelitním přenosem. C++ kód třídy HDLC je umístěn v souboru ~ns/satellite/sat-hdlc.cc, kdy ve stejné složce je uložen i jeho hlavičkový soubor ~ns/satellite/sat-hdlc.h. HDLC protokol v simulátoru NS-2 funguje na LL vrstvě rozšířené části uzlu.

Protokol HDLC může pracovat ve více různých nastaveních, kdy v simulátoru NS-2 je navržený za použití následujících parametrů:

- Typ stanice – Kombinovaná stanice schopná vydávání příkazů a reagování na přijaté příkazy.
- Konfigurace linek – Skládá se z kombinovaných stanic a podporuje poloduplexní přenos.
- Režim přenosu dat – Je použit rozšířený asynchronní vyvážený režim (ABME (*Asynchronous Balanced Mode Extended*)), kdy každá stanice může zahájit přenos bez povolení od druhé kombinované stanice. V tomto režimu jsou rámce očíslovány a příjemce musí potvrdit jejich převzetí. HDLC definuje rozšířený režim s podporou 7 bitů (128bit modulo číslování) využitých pro sekvenční číslování. Ve výchozím nastavení je využito 31 bitů.
- Inicializace přenosu – Odesílatel odešle rámec typu U nesoucí příkaz SABME (*Set Asynchronous Balanced Mode Extended*), který nastaví asynchronní vyvážený rozšířený režim. Přijímač odpoví rámcem typu U nesoucí příkaz UA (nečíslované číslo) nebo může odeslat příkaz DM, čímž odmítne inicializaci připojení.
- Přenos dat – Prováděn pomocí rámců typu I, které odesílá vysílač. Přijímač odešle zpět I rámec typu RR, který indikuje, že je očekáván další I rámec. Rámec typu S se používají pro řízení chyb a kontrolu toku. Pokud přijímač

zaznamená chybějící rámec, vyšle S rámec typu REJ, který vysílači přikazuje, aby se vrátil zpět a odeslal chybějící rámec.

- Ukončení spojení – HDLC kombinovaná stanice může zahájit odpojení pomocí U rámce typu DISC (*Disconnect*).

4.4.1 HDLC proces

V předchozí části bylo popsáno obecné fungovaní protokolu HDLC implementovaného v simulátoru NS-2. Pro úpravu tohoto protokolu je nutné důkladně prostudovat jeho vnitřní proces v C++ třídě „sat-hdlc.cc“. Proto bude v následujícím textu dopodrobna popsán algoritmus třídy „sat-hdlc.cc“ při odeslání prvního paketu a jeho následné přijetí.

Při odeslání prvního HDLC rámce pracuje ve třídě „sat-hdlc.cc“ celkem 5 funkcí. Průběh HDLC procesu jse poposán pomocí následujících kroků:

1. `HDLC::recv(Packet *p, Handler* )` – Při začátku procesu HDLC protokolu je paket přijat touto funkcí. Ta nejdříve inicializuje obecné záhlaví tohoto paketu a jestli je, na řádku 15 kódu 4.5 směr v proměnné obecného záhlaví dir_t nastaven `„UP"`, paket je brán jako příchozí. Jinak je směr v proměnné obecného záhlaví dir_t nastaven `„DOWN"`, paket přechází do funkce `HDLC::recvOutgoing(Packet* p)`.

Výpis 4.5: Funkce `HDLC::recv(Packet *p, Handler*)`

```
1  void HDLC::recv(Packet* p, Handler* )
2  {
3    hdr_cmn *ch = HDR_CMN(p);
4    assert(initialized());
5    if (ch->direction() == hdr_cmn::UP) {
6      if (ch->ptype_ == PT_ARP)
7        arptable_->arpinput(p, this);
8      else
9        recvIncoming(p);
10     return;
11   }
12   ch->direction() = hdr_cmn::DOWN;
13   recvOutgoing(p);
14 }
```

2. `HDLC::recvOutgoing(Packet *p)` – Kód této funkce je zobrazen na výpisu 4.6. Na druhém řádku se inicializuje datový typ protokolu HDLC `struct ARQstate`. `ARQstate` u HDLC implementace slouží k řízení spojení. Čtvrtý řádek zjišťuje pomocí funkce `getRoute(Packet *p)` následující skok paketu. Na 5 řádku je zjištěno jestli už bylo dříve vytvořeno HDLC spojení. Protože se jedná o první přenášený HDLC rámec, spojení ještě není vytvořené. Z pohledu vysílače se vytvoří na řádku 7. Pokud je na řádku 8 splněna podmínkaposílání

prvního rámce, je na dalším řádku volána funkce `HDLC::sendUA(Packet* p, COMMAND_t cmd)` s parametrem `SAMBE`, který je na dalším řádku nastaven na hodnotu 1, protože ještě není jasné jestli byl rámec s tímto příkazem potvrzen příjemcem. Na řádku 14 je paket umístěn funkcí `inSendBuffer(Packet *p, ARQstate *a)` do odchozí fronty.

Výpis 4.6: Funkce `HDLC::recvOutgoing(Packet *p)`

```
1  void HDLC::recvOutgoing(Packet* p)
2  {
3    ARQstate *a;
4    int next_hop = getRoute(p);
5    a = checkState(next_hop);
6    if (a == 0)
7      a = createState(next_hop);
8    if (!(a->SABME_req_) && a->t_seqno_ == 0) {
9      sendUA(p, SABME);
10     a->SABME_req_ = 1;
11     set_rtx_timer(a);
12   }
13
14   inSendBuffer(p, a);
15   if (a->SABME_req_ == 2)
16     sendMuch(a);
17 }
```

3. `HDLC::sendUA(Packet *p, COMMAND_t cmd)` – Kód této funkce je zobrazen na výpisu 4.7. Na 3. řádku kódu je alokován nový paket, který bude použit jako HDLC rámec. Následně jsou na řádcích 5 až 8 inicializovana obecná a IP záhlaví obou paketů. Pro HDLC rámec vytvořený na 3. řádku je na Radcích 9 a 10 inicializováno HDLC záhlaví. Jeho zdrojová a cílová IP adresa se zjistí z IP záhlaví vstupního paketu, kdy na řádcích 15 až 20 se nastaví jeho obecné záhlaví. Na řádku 22 se nastaví, že se jedná o rámec typu U. Následně se podle typu příkazu `COMMAND_t` rozhodne, jak se má s vytvořeným rámcem pracovat. Při prvním průchodu, funkce pracuje s příkazem `SAMBE`, a proto se na řádcích 27 až 30 nastaví zdrojová a cílová adresa rámce na stejnou hodnotu jako u paketu. Nakonec řádek 43 volá funkci `sendDown(Packet *p)`.

Výpis 4.7: Funkce `HDLC::sendUA(Packet* p, COMMAND_t cmd)`

```
1  void HDLC::sendUA(Packet *p, COMMAND_t cmd)
2  {
3    Packet *np = Packet::alloc();
4
5    hdr_cmn *ch = HDR_CMN(p);
6    hdr_cmn *nch = HDR_CMN(np);
7    hdr_ip *ih = HDR_IP(p);
8    hdr_ip *nih = HDR_IP(np);
9    struct hdr_hdlc* nhh = HDR_HDLC(np);
10   struct U_frame* uf = (struct U_frame *)&(nhh->hdlc_fc_);
11
12   nsaddr_t src = (nsaddr_t)Address::instance().get_nodeaddr(ih->saddr());
13   nsaddr_t dst = (nsaddr_t)Address::instance().get_nodeaddr(ih->daddr());
14
15   nch->addr_type() = ch->addr_type();
16   nch->uid() = uidcnt_++;
17   nch->ptype() = PT_HDLC;
18   nch->size() = HDLC_HDR_LEN;
19   nch->error() = 0;
20   nch->iface() = -2;
21
22   nhh->fc_type_ = HDLC_U_frame;
23
24   switch(cmd) {
25   case SABME:
26   case DISC:
27     nih->daddr() = ih->daddr();
28     nih->saddr() = ih->saddr();
29     nhh->daddr_ = dst;
30     nhh->saddr_ = src;
31     break;
32   case UA:
33     nih->daddr() = ih->saddr();
34     nih->saddr() = ih->daddr();
35     nhh->daddr_ = src;
36     nhh->saddr_ = dst;
37     break;
38   default:
39     fprintf(stderr, "Unknown type of U frame\n");
40     exit(1);
41   }
42   uf->utype = cmd;
43   sendDown(np);
44 }
```

4. `HDLC::sendDown(Packet *p)` – Kód této funkce je zobrazen na výpisu 4.8. Funkce `HDLC::sendDown(Packet *p)` slouží k odesílaní HDLC rámců do nižších vrstev sítě. Na řádku 3 a 4 je inicializováno obecné záhlaví a MAC záhlaví. Na řádku 7 je volána funkce `getRoute(Packet *p)`, která u satelitní linky najde další skok. Na řádcích 8 až 11 je nastavena zdrojová a cílová MAC adresa a její typ. Následuje nastavení správné MAC adresy dalšího skoku rámce, které se provádí na řádcích 25 až 40. Funkce `find_peer_mac_addr_(int dst)` na

řádku 30 slouží k vyhledání správné MAC adresy cílového uzlu, který je připojen na satelitní kanál. Řádek 46 inicializuje plánovač využitý pro odesílaní rámců. Plánovač na následujícím řádku zařadí rámec do fronty a nastaví jeho odeslání podle zadaného zpoždění.

Výpis 4.8: Funkce `HDLC::sendDown(Packet *p`

```
1  void HDLC::sendDown(Packet* p)
2  {
3    hdr_cmn *ch = HDR_CMN(p);
4    char *mh = (char*)p->access(hdr_mac::offset_);
5    int peer_mac_;
6    SatChannel* satchannel_;
7    getRoute(p);
8    mac_->hdr_src(mh, mac_->addr());
9    mac_->hdr_type(mh, ETHERTYPE_IP);
10
11   nsaddr_t dst = ch->next_hop();
12   if (dst < -1) {
13     printf("Error: next_hop_ field not set by routing agent\n");
14     exit(1);
15   }
16
17   switch(ch->addr_type()) {
18   case NS_AF_INET:
19   case NS_AF_NONE:
20     if (IP_BROADCAST == (u_int32_t) dst)
21       {
22       mac_->hdr_dst((char*) HDR_MAC(p), MAC_BROADCAST);
23       break;
24     }
25     if (dst == arpcachedst_) {
26       mac_->hdr_dst((char*) HDR_MAC(p), arpcache_);
27       break;
28     }
29     satchannel_ = (SatChannel*) channel();
30     peer_mac_ = satchannel_->find_peer_mac_addr(dst);
31     if (peer_mac_ < 0 ) {
32       printf("Error: couldn't find dest mac on channel ");
33       printf("for src/dst %d %d at NOW %f\n",
34           ch->last_hop_, dst, NOW);
35       exit(1);
36     } else {
37       mac_->hdr_dst((char*) HDR_MAC(p), peer_mac_);
38       arpcachedst_ = dst;
39       arpcache_ = peer_mac_;
40       break;
41     }
42   default:
43     printf("Error: addr_type not set to NS_AF_INET or NS_AF_NONE\n");
44     exit(1);
45   }
46   Scheduler& s = Scheduler::instance();
47   s.schedule(downtarget_, p, delay_);
48 }
```

5. `inSendBuffer(Packet *p, ARQstate *a)` – Kód této funkce je zobrazen ve výpisu 4.9. Funkce `inSendBuffer(Packet *p, ARQstate *a)` slouží vytváření HDLC I rámců z příchozích paketů. Na 3 až 5 řádku kódu z výpisu 4.9 se tyto pakety v případě plné HDLC fronty zahodí. Řádek 7 až 9 inicializuje potřebné záhlaví. Následuje nastavení HDLC záhlaví na řádcích 15 až 18. Řádek 19 přičte velikost HDLC záhlaví k původní velikosti vstupního paketu. Tímto krokem vzniká HDLC rámec typu I, který se na řádku 20 zařadí do fronty. Tímto krokem se ukončí proces odeslání prvního HDLC rámce.

Výpis 4.9: Funkce `inSendBuffer(Packet *p, ARQstate *a)`

```
1  void HDLC::inSendBuffer(Packet *p, ARQstate *a)
2  {
3    if (a->sendBuf_.length() >= queueSize_) {
4      drop(p, "HDLC queue full");
5      return;
6    }
7    hdr_cmn *ch = HDR_CMN(p);
8    hdr_ip *ih = HDR_IP(p);
9    hdr_hdlc *hh = HDR_HDLC(p);
10   struct I_frame* ifr = (struct I_frame *)&(hh->hdlc_fc_);
11
12   nsaddr_t src = (nsaddr_t)Address::instance().get_nodeaddr(ih->saddr());
13   nsaddr_t dst = (nsaddr_t)Address::instance().get_nodeaddr(ih->daddr());
14
15   hh->fc_type_ = HDLC_I_frame;
16   hh->saddr_ = src;
17   hh->daddr_ = dst;
18   ifr->send_seqno = a->seqno_++;
19   ch->size() += HDLC_HDR_LEN;
20   a->sendBuf_.enque(p);
21 }
```

V minulých pěti krocích byl popsán proces odeslání prvního rámce. Rámec projde nižšími vrstvami a přichází do cílového uzlu. V následujících krocích je popsán proces jeho přijetí:

1. `HDLC::recv(Packet *p, Handler* )` – Proces přijetí HDLC rámce opět začíná funkcí `HDLC::recv(Packet* p, Handler*)`. Kód této funkce je zobrazen ve výpisu 4.5. Přicházející paket má nastavenu proměnnou dir_t na hodnotu „`UP`". Toto nastavení se provede ve fyzické vrstvě ve objektu `Channel`. Rámec proto plní podmínku `if` funkce na řádku 5. Následně je na řádku 9 volána funkce `HDLC::recvIncoming(Packet *p)`.
2. `HDLC::recvIncoming(Packet *p)` – Kód této funkce zobrazuje 4.10. Úkolem funkce `HDLC::recvIncoming(Packet *p)` je zavolat další funkce v závislosti na typu přijatého rámce. Funkce `switch`, která tento úkol provádí, je zobrazena na řádku 7 až 20. První přijatý rámec je typu U proto se provede funkce `recvUframe(Packet *p)`.

Výpis 4.10: Funkce `HDLC::recvIncoming(Packet *p)`

```
1  void HDLC::recvIncoming(Packet* p)
2  {
3    if (hdr_cmn::access(p)->error() > 0)
4      drop(p);
5    else {
6      hdr_hdlc* hh = HDR_HDLC(p);
7      switch (hh->fc_type_) {
8      case HDLC_I_frame:
9        recvIframe(p);
10       break;
11     case HDLC_S_frame:
12       recvSframe(p);
13       break;
14     case HDLC_U_frame:
15       recvUframe(p);
16       break;
17     default:
18       fprintf(stderr, "Invalid HDLC control type\n");
19       exit(1);
20     }
21   }
22 }
```

3. `recvUframe(Packet *p)` – Kód této funkce je uveden ve výpisu 4.11. Funkce volá další funkce v závislosti na příkazu, který je přenášen v HDLC záhlaví rámce. To je inicializováno na třetím řádku. Samotná rozdělující funkce `switch` má svou deklaraci na řádcích 5 až 19. První přijatý rámce obsahuje příkaz `SAMBE`, proto se jako další provede funkce `handleSAMBErequest(Pa cket *p)`.

Výpis 4.11: Funkce `recvUframe(Packet *p)`

```
1  void HDLC::recvUframe(Packet* p)
2  {
3    hdr_hdlc* hh = HDR_HDLC(p);
4    struct U_frame *uf = (struct U_frame*)&(hh->hdlc_fc_);
5    switch(uf->utype)
6    {
7    case SABME:
8      handleSABMErequest(p);
9      break;
10   case UA:
11     handleUA(p);
12     break;
13   case DISC:
14     handleDISC(p);
15     break;
16   default:
17     fprintf(stderr, "Unknown type of U frame\n");
18     exit(1);
19   }
20 }
```

4. `handleSAMBErequest(Packet *p)` – Kód této funkce zobrazuje výpis 4.12, kdy jejím úkolem je vyřízení příkazu `SAMBE` a vytvoření spojení z pohledu příjemce. Na třetím řádku je inicializováno obecné záhlaví rámce. Následně 5. řádek slouží pro zjištění, pomocí datové struktury `ARQstate`, jestli pro vysílající uzel už existuje navázaný přenos. Pokud přenos už existuje, HDLC spojení se restartuje a vytvoří se nové. Pokud spojení neexistuje, vytvoří se na řádku 14. Následně se na řádku 16 volá funkce `HDLC::sendUA(Packet *p, COMMAND_t cmd)` s příkazem `UA`. Po skončení této funkce se rámec na řádku 17 uvolní.

Výpis 4.12: Funkce `handleSAMBErequest(Packet *p)`

```
1  void HDLC::handleSABMErequest(Packet *p)
2  {
3    hdr_cmn *ch = HDR_CMN(p);
4    int last_hop = ch->last_hop_;
5    ARQstate *a = checkState(last_hop);
6  #ifdef RESET_HDLC
7    if (a != 0) {
8      printf("Got SABME req for existing connection; resetting\n");
9      reset(a);
10   }
11   a = createState(last_hop);
12 #else
13   if (a == 0)
14     a = createState(last_hop);
15 #endif
16   sendUA(p, UA);
17   Packet::free(p);
18 }
```

5. `HDLC::sendUA(Packet *p, COMMAND_t cmd)` – Kód této funkce zobrazuje výpis 4.7. Funkce `HDLC::sendUA(Packet *p, COMMAND_t cmd)` pracuje obdobně jako při odesílaní prvního rámces jediným rozdílem v příkazu obsaženém v U rámci. Místo příkazu `SAMBE` nese rámec příkaz typu `UA`. Díky němu se provedou příkazy na řádcích 33 až 37, které nastaví nově vytvořenému rámci adresu vysílače. Následuje volání funkce `sendDown(Packet *p)`.
6. `HDLC::sendDown(Packet *p)` – Kód této funkce je zobrazen na výpisu 4.8. Funkce pracuje stejně jako při odesílaní prvního rámce.

4.4.2 Nevýhody satelitní implementace

HDLC protokol implementovaný pro satelitní proces funguje v souladu s komunikačním profilem využívajícím TCP/IP zobrazeným na obr. 2.4. Satelitní implementace ale obsahuje nevýhody spojené ze satelitním přenosem, které díky následujícím důvodům brání její implementaci pro navržený komunikační profil:

- Nemožnost nastavení zpoždění u satelitní linky.
- Odlišnost simulačních výsledku, ke kterým dochází započítávaní vzdálenosti mezi satelity a ztrát spojených ze satelitním přenosem.
- Komunikační profily DLMS/COSEM využívané u inteligentních elektrických sítí nejsou většinou zamýšleny pro satelitní přenos.

5 KOMUNIKAČNÍ PROFILY VYUŽÍVAJÍCÍ PROTKOL HDLC

Tato kapitola se zaměřuje na implementaci komunikačních profilů DLMS/COSEM využívajících protokol HDLC v simulátoru NS-2. Dříve bude probrán LAN uzel, který bude využit pro implementaci obou komunikačních profilů. Následně bude popsána implementace komunikačního profilu HDLC využívající IP v simulátoru NS-2. Poslední část této kapitoly se věnuje návrhu implementace třívrstvého komunikačního profilu využívající protokol HDLC.

5.1 LAN síť a uzel

Kvůli důvodům zmíněným v minulé kapitole není pro naše účely vhodné využít satelitní implementaci protokolu HDLC, kterou simulátor NS-2 obsahuje. Protože satelitní implementace protokolu HDLC využívá satelitní uzly a linky, které obsahují funkce nevhodné ve vyžadovaném komunikačním profilu, musí být v simulátoru NS-2 nalezena alternativní možnost. Jelikož ideální implementace počítá s kabelovými spoji mezi jednotlivými zařízeními je potřeba využít uzel simulátoru NS-2, který podporuje kabelové spoje a zároveň obsahuje rozšiřující část uzlu podle obr. 4.3. Obě tyto podmínky jsou splněny v implementaci LAN sítě, kdy po menších úpravách lze dosáhnou požadovaného komunikačního profilu. Z těchto důvodů byla pro implementaci protokolu HDLC zvolena právě LAN síť.

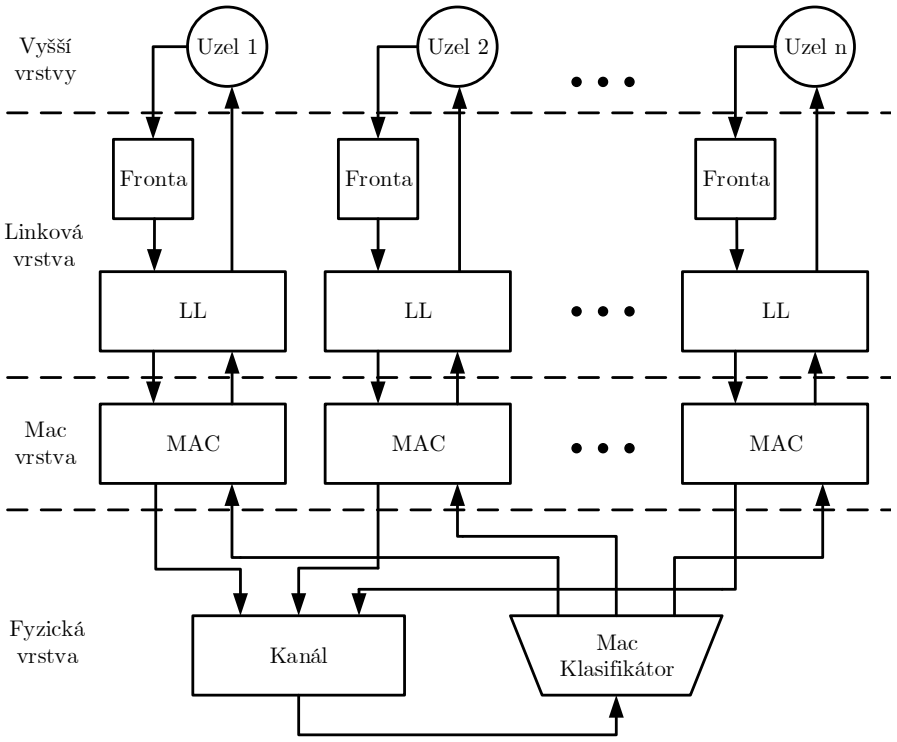
Obr. 5.1 zobrazuje konektivitu v LAN síti. Na obrázku je vidět, že LAN síť podporuje rozšiřující nižší vrstvy podobně jako mobilní či satelitní uzel. Paket poslaný směrem dolů nejprve projde linkovou vrstvou (`Fronta` a `LL`) poté MAC vrstvou (`Mac`) a nakonec fyzickou vrstvou (`Kanál` do `Mac Klasifikátoru`). Následně prochází opačným směrem zpátky do vyšších vrstev.

5.1.1 Linková vrstva

Díky velkému počtu funkcí, které může linková vrstva zastávat, je linková vrstva rozdělená na dvě hlavní části: fronta definována objektem `Queue` a samotná linková vrstva definována objektem `LL`. `Queue` objekt simuluje frontu rozhraní podle předem daných parametrů. `LL` objekt implementuje zvolený linkový protokol, jako je například ARQ nebo používaný protokol HDLC [20].

Důležitou funkcí linkové vrstvy je nastavení cílové MAC adresy v MAC záhlaví paketu. V simulátoru NS-2 se tento úkol sestává ze dvou kroků:

1. Směrování – Hledání adresy následujícího skoku paketu.
2. ARP – Přeložení nalezené IP adresy na správnou MAC adresu.



Obr. 5.1: Schéma konektivity v LAN síti [20].

Pro jednoduchost je v simulátoru NS-2 IP adresa a MAC adresa stejná. Pomocí Otcl je možné u C++ třídy `LL` nastavit:

`delay_` Zpoždění linky

5.1.2 MAC vrstva

V závislosti na typu fyzické vrstvy, MAC vrstva musí obsahovat určitou sadu funkcí, jako je detekce přenosu či detekce kolizí a předcházení kolizí. Protože tyto funkce ovlivňují odesílající i přijímací stranu, jsou tyto funkce implementovány v jediném `Mac` objektu [20].

`Mac` objekt vytvořený z `Mac` třídy simuluje protokoly MAC vrstvy, které jsou nezbytné ve sdíleném přenosovém prostředí jako jsou například LAN nebo bezdrátové sítě. Na vysílající straně je `Mac` objekt zodpovědný za nastavení MAC záhlaví paketu a jeho doručení do přenosovému kanálu. Na straně příjemce `Mac` objekt asynchronně přijímá pakety z `Mac` klasifikátoru fyzické vrstvy. Tyto přijaté pakety jsou zde pomocí MAC protokolu zpracovány a odeslány do linkové vrstvy. Pomocí Otcl je možné u C++ třídy `Mac` nastavit [20]:

`bandwidth_` Modulační rychlost MAC vrstvy

`label_` MAC adresu

5.1.3 Fyzická vrstva a kanál

Fyzická vrstva je reprezentována dvěma simulačními objekty: kanál a Mac klasifikátor. Kanál simuluje sdílené médium a podporuje mechanismy středního přístupu Mac vrstvy na straně odesílatele. Na straně příjemce Mac klasifikátor odpovídá za doručování a případnou duplikaci paketů [20].

Třída `Channel` vytváří kanál, který simuluje skutečný přenos paketů na fyzické vrstvě. Základní kanál implementuje sdílené médium s podporou mechanismů sporu. Umožňuje Mac vrstvě vykonávat detekci kolizí. Pokud se více než jeden přenos překrývá v čase, kanál indikuje kolizi pomocí kolizního příznaku. Kontrolou tohoto příznaku může Mac objekt implementovat detekci a řešení kolizí. Pomocí Otcl je možné u C++ třídy `Channel` nastavit:

`delay_` Propagační zpoždění kanálu

5.1.3.1 Vytvoření LAN sítě

Rozhraní sloužící pro vytvoření a konfiguraci LAN sítě se liší od standardních linek simulátoru NS-2. OTcl třída `Simulator` exportuje novou metodu `make-lan`, jejíž vstupní parametry jsou podobné těm, které se používají pro vytvoření duplexních linek `duplex-link`. Na výpisu 5.1 je uveden příkaz použitý pro vytvoření LAN sítě v simulátoru NS-2. Vstupní parametry tohoto příkazu jsou vysvětleny v Tab. 5.1 [20].

Výpis 5.1: Funkce pro vytvoření LAN sítě v programu NS-2

```
1 SimulatSimulator instproc make-lan {nodes bw delay lltype mactype chantype}
```

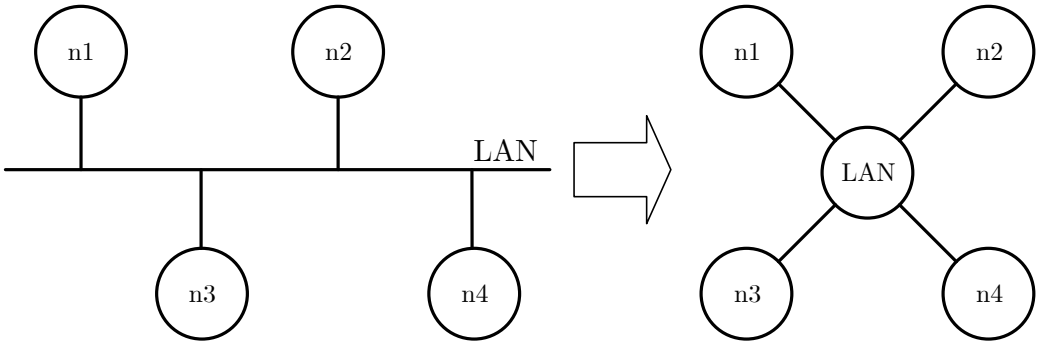
Tab. 5.1: Vstupní parametry funkce `make-lan`

Vstupní parametr	**Význam parametru**
`nodes`	List uzlů využitých při vytvoření LAN sítě
`bw`	Propustnost v LAN síti
`delay`	Zpoždění v LAN síti
`lltype`	Typ linkové vrstvy
`ifqtype`	Typ fronty
`mactype`	Typ MAC vrstvy
`chantype`	Typ přenosového kanálu (fyzická vrstva)

5.1.4 Směrování v LAN síti

Když vytvoříme LAN síť pomocí příkazu `make-lan` nebo `newLan`, vytvoří se také „virtuální LAN uzel“ `LanNode`. `LanNode` spojuje všechny sdílené položky v LAN

síti: `Channel`, `Classifier/Mac` a `LanRouter`. Následně je pro každý uzel v LAN síti vytvořen objekt `LanIface`. `LanIface` obsahuje všechny ostatní objekty nezbytné pro fungování individuálních uzlů. Je nutné si uvědomit, že uzel `LanNode` je použit pouze pro směrovací algoritmy. Obr. 5.2 zobrazuje srovnání reálné LAN konfigurace s konfigurací, kterou vidí program NS-2. Z obrázku vyplývá, že program NS-2 vidí `LanNode` jenom jako další uzel, který je se všemi ostatními uzly v LAN síti spojený „vituálními“ linky (Vlink). Protože je zpoždění nastaveno v linkové vrstvě a propustnost v MAC vrstvě, linky `Vlink` nevykazují žádné zpoždění a mají nekonečnou propustnost [20].



Obr. 5.2: Reálná konfigurace LAN vůči konfiguraci viděnou programem NS-2 [20].

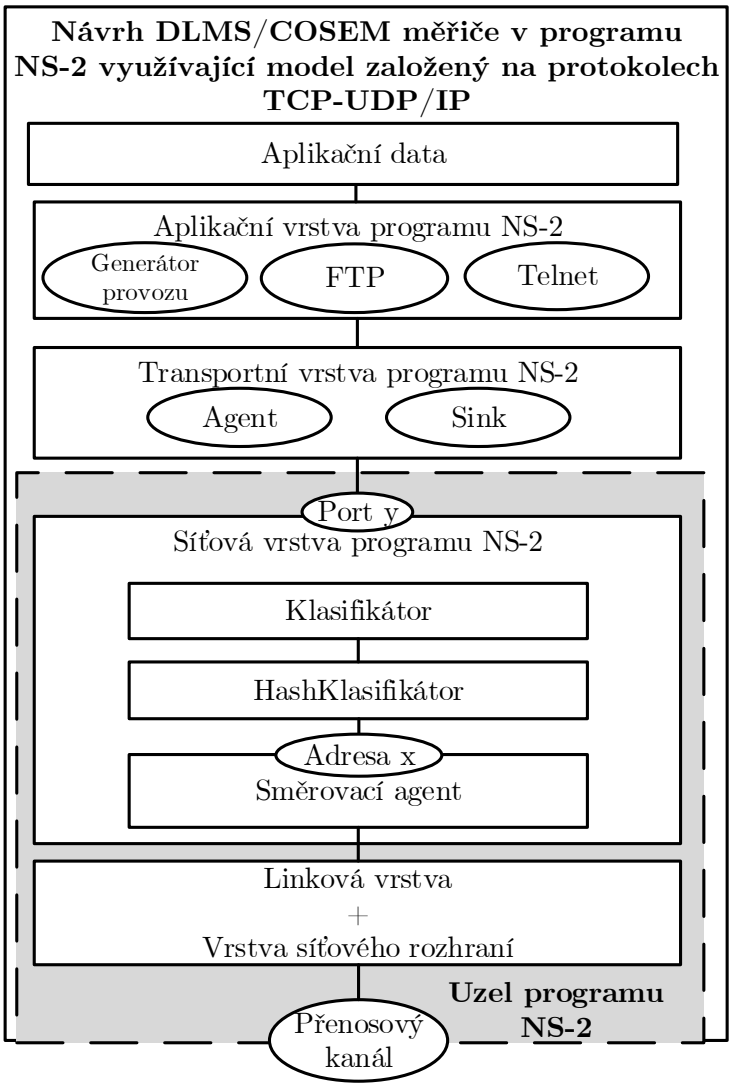
Směrování v LAN síti zprostředkovává objekt `LanRouter`. Ve výchozím nastavení obsahuje síť LAN pouze jeden objekt `LanRouter`, který je vytvořen při inicializaci `LanNode`. Pro každý uzel v LAN má objekt linkové vrstvy (`LL`) ukazatel na `LanRouter`, díky čemu je schopen najít další skok pro paket, který je v této síti odeslán. `LanRouter` hledá další skok pro paket pomocí funkce `next_hop(Packet *p)`, ve které se dotazuje, na další skoky paketu pomocí třídy `RouteLogic`. Kód funkce `next_hop(Packet *p)` je zobrazen ve výpisu 5.2 [20].

Výpis 5.2: Funkce `next_hop(Packet *p)`

```
1 int LanRouter::next_hop(Packet *p) {
2     int next_hopIP;
3     if (enableHrouting_) {
4         routelogic_->lookup_hier(lanaddr_, adst, next_hopIP);
5     } else {
6         routelogic_->lookup_flat(lanaddr_, adst, next_hopIP);
7 }
```

5.2 Komunikační profil HDLC využívající TCP/IP pomocí simulátoru NS-2

V kapitole 2.4 byly diskutovány komunikační profily DLMS/COSEM využívajících HDLC. Tato sekce je zaměřena na komunikační profil DLMS/COSEM využívající model založený na protokolech TCP-UDP/IP, který ke své adresaci využívá protokoly IP a TCP. Tento profil se používá v případech, kdy je potřeba přenést data médiem (internet) kde by adresace pouze pomocí HDLC nebyla možná. Jelikož simulátor NS-2 nepoužívá standardní síťový model TCP/IP, návrh komunikačního profilu pro NS-2 musí zahrnovat různé úpravy, které tyto rozdíly řeší. Přizpůsobený komunikační profil je detailně zobrazen na obr. 5.3.



Obr. 5.3: Komunikační profil HDLC využívající IP adresaci v simulátoru NS-2.

Nejvyšší síťovou vrstvu v simulátoru NS-2 zastupují aplikační data, která jsou definována ve třídě `Appdata`, která rovněž slouží k nastavení typu přenášených dat. Velikost a typ těchto data se pak nastaví v transportní vrstvě agentem do záhlaví paketů. Aplikační vrstva je v simulátoru NS-2 tvořena třemi aplikacemi:

1. `FTP` – Aplikace simulující FTP přenos.
2. `Telnet` – Aplikace simulující přenos protokolem Telnet.
3. `Generátor síťového provozu` – Generátor síťového provozu, který může dle daného schématu data odesílat.

Aplikační model DLMS/COSEM by se dal do NS-2 implementovat právě na této vrstvě, což ale nebylo cílem této práce.

Transportní vrstva je v simulátoru NS-2 reprezentována agentem a jeho protistranou. Agent generuje pakety podle parametrů, které obdrží od připojené aplikace. Program NS-2 obsahuje dva hlavní typy agentů TCP a UDP, kdy agent typu TCP kromě generace paketu zajišťuje i jeho spolehlivý přenos. Mechanismu přenosu paketů se věnovala podkapitola 4.1.1.

Agent je vázán k uzlu jehož ukazatel se nazývá port. Uzel v simulátoru NS-2 zastupuje funkci síťové vrstvy, linkové vrstvy a vrstvy síťového rozhraní. IP adresaci obstarávají tři komponenty: Klasifikátor, HashKlasifikátor a Směrovací agent. Klasifikátor bude diskutován samostatně v kapitole 5.3. HashKlasifikátor slouží k hledání cílových adres paketů. Směrovací agent má za úkol získat všechny informace a topologii sítě. Ty dále použije pro výpočet směrovacího schéma, kdy k tomuto úkolu používá zvolený směrovací protokol. Linková vrstva je v navrženém komunikačním profilu realizována protokolem HDLC.

5.2.1 Úprava C++ kódu HDLC tříd

Aby mohl být komunikační profil představený na obr. 5.3 pomocí LAN sítě implementován na kabelové spoje, musí se satelitní implementace HDLC upravit tak, aby nebyla závislá na satelitních uzlech a linkách. Většina úprav se provede uvnitř C++ třídy `sat-hdlc.cc`. Nejprve se musí změnit dědičnost třídy `HDLC()` ze satelitní třídy `SatLL()`na dědičnost ze třídy `LL()`. Algoritmus definující rozšířenou část satelitního uzlu se nalézá v C++ třídě „satlink.cc“. `LL()` se používá jako výchozí `LL` třída u implementace LAN sítě. Provedená úprava je uvedena ve výpisu 5.3. Dále se musí do hlavičkového souboru „sat-hdlc.h“ přidat příkaz `#include <ll.h>`, aby pod procesor při kompilaci třídy „sat-hdlc.h“ počítal i s třídou `ll.h`.

Výpis 5.3: Úprava dědičnosti HDLC třídy

```
1 HDLC::HDLC() : LL(), list_head_(0)
2 {
3   bind("window_size_", &wnd_);
```

```
4     bind("queue_size_", &queueSize_);
5     bind_time("timeout_", &timeout_);
6     bind("max_timeouts_", &maxTimeouts_);
7     bind("selRepeat_", &selRepeat_);
8     bind("delAck_", &delAck_);
9     bind("delAckVal_", &delAckVal_);
10    wndmask_ = HDLC_MWM;
11  }
```

Dále je potřeba upravit funkce využívající proměnné nebo funkce definované v satelitní třídě `SatLL()`:

`HDLC::sendDown(Packet *p)` – `satchannel_`, `find_peer_mac_addr()`, `arpcache_`, `arpcachedst_` a `getRoute(Packet *p)`

`HDLC::recvOutgoing(Packet* p)` – `getRoute(Packet *p)`

`HDLC::reset(ARQstate *a)` – `sendUp(Packet *p)`

`HDLC::goBackNMode(Packet* p)` – `sendUp(Packet *p)`

`HDLC::selectiveRepeatMode(Packet* p)` – `sendUp(Packet *p)`

Poslední tři uvedené funkce využívají ze satelitní třídy `SatLL()` stejnou funkci `sendUp(Packet *p)`. Tato funkce je obsažena i ve třídě LL, kdy syntaxe je u obou tříd stejná a `sendUp(Packet *p)` se tedy upravovat nemusí. Kód této funkce je zobrazen ve výpisu 5.4. Na 3. řádku výpisu dochází k inicializaci plánovače, který na 7. řádku naplánuje poslání paketu do vyšší vrstvy (`uptarget_`) v závislosti na zvoleném zpoždění (`delay_`).

Výpis 5.4: Funkce `sendUp(Packet *p)`

```
1  void LL::sendUp(Packet* p)
2  {
3    Scheduler& s = Scheduler::instance();
4    if (hdr_cmn::access(p)->error() > 0)
5      drop(p);
6    else
7      s.schedule(uptarget_, p, delay_);
8  }
```

Funkce `HDLC::sendDown(Packet *p)` a `HDLC::recvOutgoing(Packet *p)` musejí být upraveny ve větším měřítku. V následujících sekcích je popsána úprava těchto funkcí.

Úprava funkce `HDLC::sendDown(Packet *p)`

Výpis kódu 5.5 přísluší upravené funkci `HDLC::sendDown(Packet *p)`. V původní funkci byla pro nalezení cílového uzlu využita funkce `getRoute(Packet *p)`. U upravené verze bude využito IP záhlaví, které má ve svých proměnných `daddr()` a `saddr()` uložený cílový a zdrojový uzel, který byl nastaven směrovacím agentem. Inicializace IP záhlaví je provedena na 4. řádku. Získání proměnných s uloženými uzly

je popsáno na 6. a 7. řádku. Ty jsou následně na řádcích 10 a 11 uloženy do proměnných obecného záhlaví `next_hop_` a `last_hop_`. Poté funkce `switch` za použití typu adresy určí, jak dál s MAC hlavičkou paketu pracovat, kdy v našem případě bude vždy pracovat z možností `default`. Na řádku 36 lze vidět příkaz, který nastavuje `LanRouter` jako další skok rámce, kdy `LanRouter` zajistí další směrování k cíli. Následuje uložení MAC adresy, které v závislosti na proměnné `IPnh` může probíhat různými způsoby.

Výpis 5.5: Upravená funkce `HDLC::sendDown(Packet *p)`

```
1  void HDLC::sendDown(Packet* p)
2  {
3    hdr_cmn *ch = HDR_CMN(p);
4    hdr_ip *ih = HDR_IP(p);
5
6    nsaddr_t dst = (nsaddr_t)Address::instance().get_nodeaddr(ih->daddr());
7    nsaddr_t src = (nsaddr_t)Address::instance().get_nodeaddr(ih->saddr());
8    char *mh = (char*)p->access(hdr_mac::offset_);
9
10   ch->next_hop_ = dst;
11   ch->last_hop_ = src;
12   int next_hop = ch->next_hop_;
13
14   mac_->hdr_src(mh, mac_->addr());
15   mac_->hdr_type(mh, ETHERTYPE_IP);
16   int tx = 0;
17   switch(ch->addr_type()) {
18
19   case NS_AF_ILINK:
20       mac_->hdr_dst((char*) HDR_MAC(p), ch->next_hop());
21       break;
22   case NS_AF_INET:
23   case NS_AF_NONE:
24
25     if (IP_BROADCAST == (u_int32_t) dst)
26       {
27       mac_->hdr_dst((char*) HDR_MAC(p), MAC_BROADCAST);
28       break;
29       }
30     if (arptable_) {
31           tx = arptable_->arpresolve(dst, p, this);
32           break;
33         }
34   default:
35
36     int IPnh = (lanrouter_) ? lanrouter_->next_hop(p) : -1;
37     if (IPnh < 0)
38       mac_->hdr_dst((char*) HDR_MAC(p),macDA_);
39     else if (varp_)
40       tx = varp_->arpresolve(IPnh, p);
41     else
42       mac_->hdr_dst((char*) HDR_MAC(p), IPnh);
43     break;
44   }
45 if (tx == 0) {
```

```
46    Scheduler& s = Scheduler::instance();
47    s.schedule(downtarget_, p, delay_);
48  }
49  }
```

Úprava funkce `HDLC::recvOutgoing(Packet* p)`

Výpis kódu 5.6 ukazuje důležitý výňatek z upravené funkce `HDLC::recvOutgoing (Packet *p)`, která byla přizpůsobena stejným způsobem jako funkce `HDLC::send Down(Packet *p)`. Funkce `getRoute (Packet *p)` je znovu nahrazena metodou využívající IP záhlaví, jejíž princip byl popsán v předchozím paragrafu.

Výpis 5.6: Výňatek upravené funkce `HDLC::recvOutgoing(Packet* p)`

```
1   hdr_ip *ip = HDR_IP(p);
2
3   nsaddr_t dst = (nsaddr_t)Address::instance().get_nodeaddr(ip->daddr());
4   nsaddr_t src = (nsaddr_t)Address::instance().get_nodeaddr(ip->saddr());
5
6   ch->next_hop_ = dst;
7   ch->last_hop_ = src;
```

5.2.2 Úprava trasovacích tříd pro kabelové HDLC

V části 3 byly zmíněny základy trasovacích souborů v simulátoru NS-2. Tab. 3.3 zobrazuje strukturu trasovacího souboru klasické duplexní linky. Implementovaný protokol HDLC přidává za klasický výpis další informace spojené s svým fungováním. V Tab. 5.2 jsou uvedeny tyto přidané informace.

Tab. 5.2: Část trasovacího souboru rozšířená protokolem HDLC

Zdroj HDLC paketu	Cíl HDLC paketu	Typ HDLC rámce	Pořadové číslo paketu	Typ přenášeného příkazu

Formát trasovacího souboru je nastaven v C++ třídě „trace.cc“, kdy o jeho vypsání do souboru se stará nadřazená C++ třída „basetrace.cc“. V OTcl doméně se dá nastavit, aby paket odcházející z určité vrstvy (např. `LL`) nebyl odeslán do vrstvy nižší (`Mac`), ale do třídy `Trace()`, která ho přijme funkcí funkcí `Trace::recv(Packet *p, Handler *h)`. Kód této funkce se nachází ve výpisu 5.7.

Výpis 5.7: Funkce `Trace::recv(Packet *p, Handler *h)`

```
1  void Trace::recv(Packet* p, Handler* h)
2  {
3    format(type_, src_, dst_, p);
4    pt_->dump();
```

```
5     callback();
6     pt_->namdump();
7     if (target_ == 0)
8       Packet::free(p);
9     else
10      send(p, h);
11  }
```

Z výpisu lze vyčíst, že hned druhý řádek volá funkci `format(type_, src_, dst_, p)`, jejíž vstupní parametry jsou:

`type_` Typ paketu

`src_` Zdrojový uzel paketu

`dst_` Cílový uzel paketu

`p` Zpracovávaný paket

Funkce `Trace::format(int tt, int s, int d, Packet* p)` slouží k vytvoření správného formátu trasovacího souboru podle typu vstupního paketu a nastavených trasovacích parametrů. Na kódu 5.8 je možné pozorovat výňatek z této funkce, který se u klasického trasovacího souboru používá k nastavení jeho formátu. Druhý řádek zobrazuje instanci třídy `BaseTrace() pt_`, která je vytvořena při inicializaci třídy `Trace()`, ukládá proměnné určené k dalšímu použití do své vyrovnávací paměti. Ta je okamžitě použita voláním funkce `pt_->dump()` na 4 řádku výpisu 5.7. Funkce `pt_->dump()` vysazuje vyrovnávací paměť do trasovacího souboru.

Výpis 5.8: Výňatek z funkce `Trace::format(int tt, int s, int d, Packet* p)`

```
1  else if (!show_tcphdr_) {
2    sprintf(pt_->buffer(), "%c "TIME_FORMAT" %d %d %s %d %s %d %s.%s %s.%s %d %d",
3    tt,
4    pt_->round(Scheduler::instance().clock()),
5    s,
6    d,
7    name,
8    th->size(),
9    flags,
10   iph->flowid()
11   src_nodeaddr,
12   src_portaddr,
13   dst_nodeaddr,
14   dst_portaddr,
15   seqno,
16   th->uid());
17 }
```

Při simulaci satelitního přenosu v simulátoru NS-2 se definuje speciální trasovací třídu `Sattrace()`, která je využívána místo klasické třídy `Trace()`. Tato třída obsahuje HDLC rozšíření ve formě funkce `SatTrace::format_hdlc(Packet *p, int offset)`, která funguje na stejném principu jako funkce `Trace::format(int tt, int s, int d, Packet *p)`, kdy vkládá parametry uvedené v tabulce 5.2 na konec

vyrovnávací paměti. Funkce `SatTr ace::format_hdlc(Packet *p, int offset)` obsahuje speciální formátování pro každý druh HDLC rámce. Ve výpisu kódu 5.9 je uvedeno formátování HDLC rámce typu S.

Výpis 5.9: Formátování HDLC rámce typu S

```
1 sprintf(pt_->buffer() + offset,
2   "[%d %d S %d %s]",
3   hh->saddr(),
4   hh->daddr(),
5   sf->recv_seqno,
6   sf->stype == RR ? "RR" :
7   sf->stype == REJ ? "REJ" :
8   sf->stype == RNR ? "RNR" :
9   sf->stype == SREJ ? "SREJ" :
```

Funkce `SatTrace::format_hdlc(Packet *p, int offset)` se inicializuje v hlavní formátující funkci `SatTrace::format(int tt, int s, int d,Packet* p)`, jejíž kód je uveden ve výpisu 5.10. Na prvním řádku kódu se do proměnné `offset` uloží délka dat uložených ve vyrovnávací paměti proměnné `pt_`. Tuto proměnnou funkce `SatTrace::format _hdlc(Packet *p, int offset)` používá ke vkládání nových dat na správné místo vyrovnávací paměti. Na 2. až 4. řádku figuruje funkce `if`, která provede HDLC formátování pokud je vstupní paket typu HDLC.

Výpis 5.10: Inicializace funkce `SatTrace::format_hdlc(Packet *p, int offset)`

```
1 offset = strlen(pt_->buffer());
2  if (t == PT_HDLC){
3     format_hdlc(p, offset);
4      }
```

LAN sítě používají pro vytvoření trasovacího souboru třídu `Trace()`. Jak bylo uvedeno výše, formátovaní HDLC obsahuje satelitní třída `Sattrace()` a je tedy potřeba ho do LAN sítě doplnit. Funkce `SatTrace::format_hdlc(Packet *p, int offset)` se musí přesunout do třídy `Sattrace()`. Po přesunu je změněn název funkce na `Trace::format_hdlc(Packet *p, int offset)` a inicializace této funkce 5.10 je vložena do funkce `Trace::format(int tt, int s, int d, Packet* p)`. Aby funkce správně fungovala, musí být definována ve hlavičkovém souboru „trace.h“. Posledním krokem úpravy je Tcl natavení LAN sítě tak, aby správně potvrzovala přijaté pakety. LAN síť má v základu nastaveno potvrzování přijatých paketů (formátování) ve vrstvě `iface`, která je nadřazená linkové vrstvě. Protože se HDLC rámce nikdy do této vrstvy nedostanou, je potřeba, aby potvrzení paketů nastalo jeho odeslání z MAC do linkové vrstvy. Této změny lze dosáhnout úpravou konfiguračního Tcl souboru `vlan.tcl`. Upravená část tohoto konfiguračního souboru je ukázána ve výpisu 5.11. První až pátý řádek slouží k vytvoření trasovacích událostí

(např. příjem nebo zahození paketu). Nejdůležitější jsou řádky 17 a 18, ve kterých je nastaveno potvrzení přijetí paketu na výstup z MAC vrstvy. Po těchto úpravách je trasovací soubor v sítích LAN nastaven tak, aby zobrazoval stejné udaje o HDLC protokolu jako trasovací soubor u satelitního přenosu.

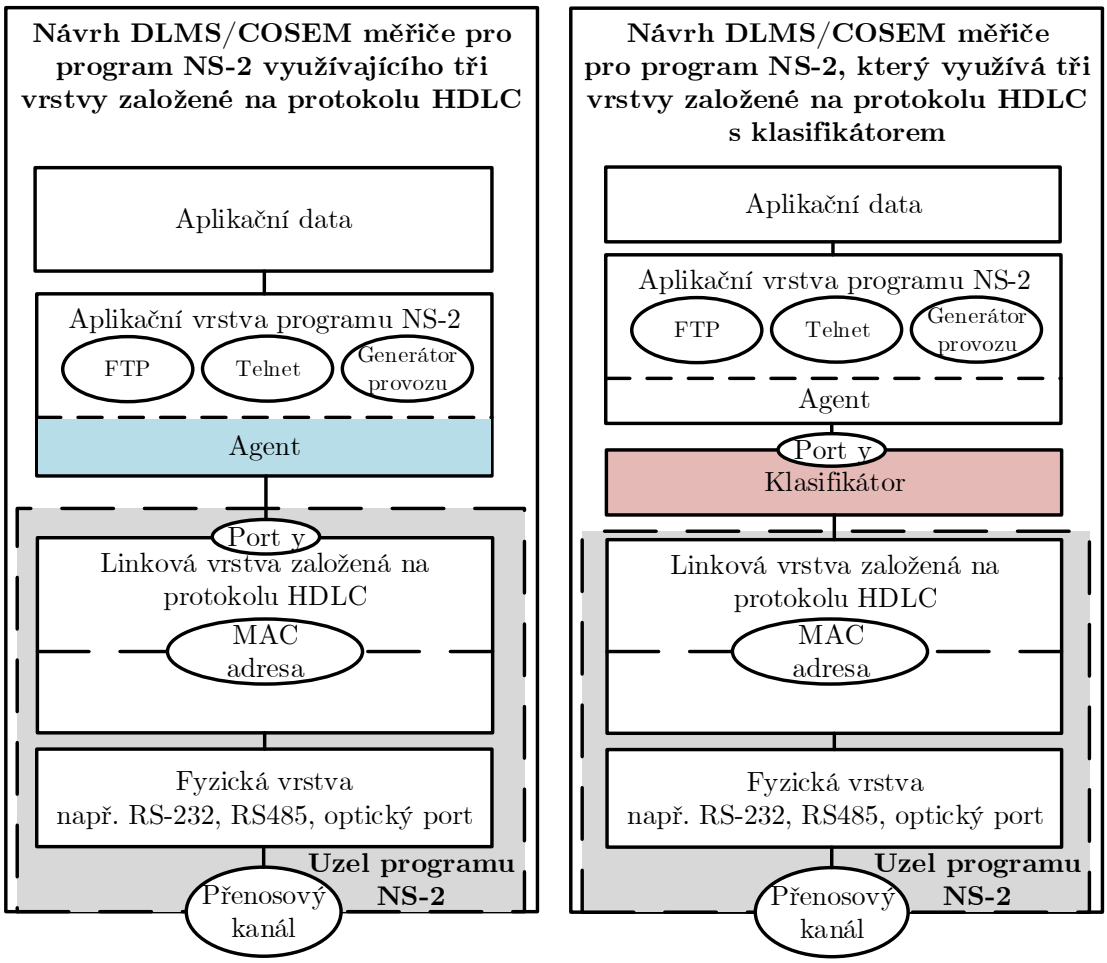
Výpis 5.11: Upravená část souboru „vlan.tcl“

```
1   set hopT_ [$ns create-trace Hop   $f $node_ $lan_  $op]
2   set rcvT_ [$ns create-trace Recv  $f $lan_  $node_ $op]
3   set enqT_ [$ns create-trace Enque $f $node_ $lan_  $op]
4   set deqT_ [$ns create-trace Deque $f $node_ $lan_  $op]
5   set drpT_ [$ns create-trace Drop  $f $node_ $lan_  $op]
6         if {[string compare $mactrace_ "true"] == 0} {
7       #  For Mac level collision traces
8       set macdrpT_ [$ns create-trace Collision $f $node_ $lan_ $op]
9       set macdrophead_ [new Connector]
10      $mac_ drop-target $macdrophead_
11      $macdrophead_ target $macdrpT_
12  }
13
14  $hopT_ target [$entry_ target]
15  $entry_ target $hopT_
16
17  $rcvT_ target [$mac_ up-target]
18  $mac_ up-target $rcvT_
19
20  $enqT_ target [$ll_ down-target]
21  $ll_ down-target $enqT_
22
23  $deqT_ target [$ifq_ target]
24  $ifq_ target $deqT_
25
26  $drpT_ target [$drophead_ target]
27  $drophead_ target $drpT_
28 }
```

5.3 Návrh třívrstvého komunikačního profilu HDLC pomocí simulátoru NS-2

Druhý navrhovaný komunikační profil využívá ke své adresaci pouze protokol HDLC. Obr. 5.4 zobrazuje návrhy třívrstvého komunikačního profilu využívající protokol HDLC pomocí simulátoru NS-2. Tento komunikační profil je realizací DLMS/COSEM profilu zobrazeného na Obr. 2.4. Jak jde na Obr. 5.4 vidět, že tento komunikační profil vůbec neobsahuje transportní ani síťovou vrstvu.

Aplikační data zastávají stejnou úlohu jako při předchozím implementovaném komunikačním profilu využívající TCP/IP. Velký rozdíl je přesun agenta z transportní vrstvy do aplikační. V simulátoru NS-2 existují dva základní druhy agentů: TCP a UDP agent. Ti však tyto agenti pracují podle protokolů transportní vrstvy



Obr. 5.4: Napravo je uvedeno schéma třívrstvého komunikačního profilu využívající protokol HDLC v simulátoru NS-2, nalevo je uveden stejný profil, ale s přidaným klasifikátorem.

a nelze je použít. Z tohoto důvodu musí být vytvořen nový agent, který bude HDLC rámce vytvářet a bez použití IP adresy dál posílat do linkové vrstvy.

V linkové vrstvě bude probíhat adresace založená na obecném a HDLC záhlaví rámců, kdy HDLC bude také odpovídat za spolehlivé doručení rámců. MAC vrstva a fyzická vrstva budou realizovány stejně jako u komunikačního profilu využívající TCP/IP protokoly (viz. Obr. 5.3).

Při tvorbě požadovaného komunikačního profilu v simulátoru NS-2 se bude vycházet z implementace popsané v sekci 5.2. Úprava této implementace se dá rozdělit do dvou hlavních částí:

1. Úprava HDLC třídy „sat-hdlc.cc“ tak, aby při své adresaci nevyužívala IP protokol.
2. Úprava tříd spojených s adresováním v simulátoru NS-2.

5.3.1 Úprava C++ kódu třídy „sat-hdlc.cc“

První krok nutný k úspěšné implementaci třívrstvého komunikačního profilu využívajícího HDLC je změna C++ třídy protokolu HDLC „sat-hdlc.cc“. Úpravou je v tomto případě myšleno odstranění závislosti protokolu HDLC na IP adresách přenášených v IP záhlaví.Celkově se bude muset ve třídě HDLC() upravit 7 funkcí, kdy cílem je, aby se IP adresy nastavovaly do obecného záhlaví. Protože komunikace probíhá pouze mezi dvěmi stanicemi je možné napevno nastavit cílové a zdrojové adresy. Toto nastavení bude probíhat ve třídě `HDLC::recv(Packet* p)`. V následujících bodech budou pro všech 7 funkcí popsány provedené úpravy:

1. `HDLC::recv(Packet *p, Handler*)` – Kód této funkce je zobrazen ve výpisu 5.12. Upravená funkce `HDLC::recv(Packet *p, Handler*)` slouží k nastavení adres dvou uzlů do obecného záhlaví. Tato adresace je provedena pomocí funkce `if` (viz řádky 6 až 14). je provedena pomocí funkce `if`. Adresa vysílajícího uzlu je při vytvoření nastavena jako 0 a adresa přijímajícího uzlu je nastavena 1. Proměnná `ch->next_hop_` slouží jako ukazatel uzelu, který právě zpracovává rámec. Při vytvoření rámce v agentu je výchozí hodnota `ch->next_hop_` rovna 0, čímž je při prvním přenosu vždy splněna podmínka `ch->next_hop_== 0` a jsou provedeny příkazy na řádcích 7 až 9, které nastaví adresu přijímacího uzlu do obecného záhlaví.

Výpis 5.12: Upravená funkce `HDLC::recv(Packet *p)`

```
1  void HDLC::recv(Packet* p, Handler* )
2  {
3    printf("recv \n");
4    hdr_cmn *ch = HDR_CMN(p);
5
6    if (ch->prev_hop_ == 0 ) {
7      ch->next_hop_ = 1;
8      ch->last_hop_ = 0;
9      ch->prev_hop_ = 1;
10   }
11   else {
12     ch->next_hop_ = 0;
13     ch->last_hop_ = 1;
14     ch->prev_hop_ = 0;
15   }
16   assert(initialized());
17   if (ch->direction() == hdr_cmn::UP) {
18     if (ch->ptype_ == PT_ARP)
19       arptable_->arpinput(p, this);
20     else
21       recvIncoming(p);
22     return;
23   }
24   ch->direction() = hdr_cmn::DOWN;
25   recvOutgoing(p);
26 }
```

2. `HDLC::recvOutgoing(Packet *p)` – Původně byla ve funkci `HDLC::recvOutgoing(Packet *p)` prováděna IP adresace pro rámce. Protože tuto funkci nyní splňuje funkce `HDLC::recv(Packet *p, Handler*)` byly příkazy spjaté s IP adresací odstraněny. Stejnou změnou prošla funkce `HDLC::sendDown(Packet *p)`.
3. `HDLC::sendRR(Packet *p, ARQstate *a)` – Kód upravené funkce je zobrazen ve výpisu 5.13. První změnou oproti původní funkci `HDLC::sendRR(Packet *p, ARQstate *a)` je, že neinicializujeme IP záhlaví pro původní ani pro nově vytvořený rámec. Druhá provedená změna zaručuje, že se do obecného záhlaví ihned nastavuje adresa původního rámce. Stejným způsobem byly upraveny funkce `HDLC::sendREJ(Packet *p, ARQstate *a)`, `HDLC::sendSREJ(Packet *p, ARQstate *a)`, `HDLC::sendUA(Packet *p, COMMAND_t cmd)` a `HDLC::inSendBuffer(Packet *p, ARQstate *a)`.

Výpis 5.13: Upravená funkce `HDLC::inSendBuffer(Packet *p, ARQstate *a)`

```
1  void HDLC::sendRR(Packet *p, ARQstate *a)
2  {
3    printf("sendRR \n");
4    Packet *np = Packet::alloc();
5    hdr_cmn *ch = HDR_CMN(p);
6    struct hdr_hdlc *hh = HDR_HDLC(p);
7    hdr_cmn *nch = HDR_CMN(np);
8    struct hdr_hdlc *nhh = HDR_HDLC(np);
9    struct S_frame *sf = (struct S_frame*)&(nhh->hdlc_fc_);
10
11   nch->addr_type() = ch->addr_type();
12   nch->uid() = uidcnt_++;
13   nch->ptype() = PT_HDLC;
14   nch->size() = HDLC_HDR_LEN;
15   nch->error() = 0;
16   nch->iface() = -2;
17   nch->prev_hop_  = ch->prev_hop_ ;
18   nch->next_hop_  = ch->next_hop_ ;
19   nch->last_hop_  = ch->last_hop_ ;
20   nhh->fc_type_ = HDLC_S_frame;
21
22   if (selRepeat_)
23     sf->recv_seqno = a->nextpkt_;
24   else
25     sf->recv_seqno = a->recv_seqno_;
26   sf->stype = RR;
27   nhh->saddr_ = hh->daddr();
28   nhh->daddr_ = hh->saddr();
29   sendDown(np);
30 }
```

Pro úplné odstranění IP adresace je kromě uvedených úprav funkcí třídy „sat-hdlc.cc“ potřeba odstranit IP záhlaví z konfigurace simulace pomocí Tcl kódu. To se udělá příkazem `remove-packet-header IP`, který pro správnou funkčnost musí proběhnout před inicializací simulátoru příkazem `set ns [new Simulator]`.

5.3.2 Návrh třívrstvého komunikačního profilu s využitím upraveného agenta

První návrh ke zprovoznění komunikačního profilu, zobrazený na levé straně Obr. 5.4, počítá s vytvoření nového HDLC agenta, který pracuje pouze na aplikační vrstvě. Toho bude docíleno tím, že agent bude přinucen, aby pakety bez použití adresace IP záhlavím posílal přímo do linkové vrstvy. Agent bude vázán na uzel pomocí portu bez IP adresy. V následující krocích jsou popsány úkony vedoucí k vytvoření nového agenta:

1. V adresáři „/ns-2.35“ se vytvoří složka s názvem „HDLCagent“.
2. Ve složce „/HDLCagent“ se vytvoří následující třídy potřebné k fungování agenta:
   (a) „`HDLCagent.h`“ – Poskytuje záhlaví souboru. Skládá se z deklarace funkcí a proměnných, které mají být za tímto účelem použity.
   (b) „`HDLCagent.cc`“ – Poskytuje definici funkcí, které agent používá, a které jsou deklarovány ve výše uvedeném souboru záhlaví. Důležité je, aby agent vytvářel pakety ve tvaru HDLC rámce. Toho lze docílit deklarací zobrazenou na výpisu 5.14, který obsahuje výňatek kódu agenta.

   Výpis 5.14: Deklarace pro nastavení paketu do tvaru HDLC rámce

```
1 p = allocpkt();
2 hdr_cmn *ch = HDR_CMN(p);
3 ch->ptype() = PT_HDLC;
4 ch->size() = HDLC_HDR_LEN;
```

3. Modifikuje se inicializační soubor simulátoru NS-2 „Makefile.in“. Ten obsahuje soubory objektů všech agentů a protokolů, které jsou v simulátoru NS-2 dostupné. Objekt patřící Novému agentu musí být v tomto souboru vytvořen. Objekt se vytváří přidáním řádku `HDLCagent/HDLCagent.o` v deklaraci objektů `OBJ_CC =`.
4. Modifikace hlavičkového souboru „packet.h“, ve kterém se definují nové typy agentů a paketů, které agenti používají. HDLC agent nepotřebuje definovat nový paket, protože použije paket už implementovaný satelitní implementací protokolu HDLC. V hlavičkovém souboru „packet.h“ se pak HDLC agent definuje příkazem `name_[PT_HDLC]= "HDLC";`.

5. Modifikace souboru s definicemi příkazu nutných k vytvoření OTcl objektů „ns-lib.tcl“. Pro vytvoření nového HDLC agenta se musí definovat dvě funkce (viz výpis 5.15). První funkce slouží k inicializaci nového HDLC agenta. Druhá funkce `instproc create-newagent-agent {node}` slouží k jeho zavolání při vytváření simulace.

Výpis 5.15: Výpis nastavení portů HDLC agenta v souboru „ns-agent.tcl“

```
1  HDLCAGENT {
2  set ragent [$self create-HDLCagent-agent $node]
3  }
4  Simulator instproc create-HDLCagent-agent { node }
5  {
6  set ragent [new Agent/HDLCAGENT [$node node-addr]]
7  $self at 0.0 "$ragent start"
8  $node set ragent_ $ragent
9  return $ragent
10 }
```

6. Modifikace souboru nastavujícího porty, na které se má HDLC agent vázat, „ns-agent.tcl“. Kód tohoto nastavení je zobrazen ve výpisu 5.16

Výpis 5.16: Výpis přidaných funkcí v souboru „ns-lib.tcl“.

```
1 Agent/HDLCAGENT instproc init args {
2 $self next $args
3 }
4 Agent/HDLCAGENT set sport_ 0
5 Agent/HDLCAGENT set dport_ 0
```

7. Modifikace souboru „ns-packet.tcl“. Pro správnou funkčnost nového agenta je nutné do tohoto souboru přidat kód `HDLCagent`, čímž končí proces vytvoření nového agenta v simulátoru NS-2.

Bohužel kvůli struktuře simulátoru HDLC není možné posílat pakety z agenta přímo linkové vrstvy. Pakety musí z agenta nejprve odejít do klasifikátoru, který pošle paket do hash klasifikátoru, který provede klasifikaci přicházejících paketů (Přidělí jím cílové a zdrojové IP adresy). Další postup by měl být vytvoření vlastního klasifikátoru, který by neposílal pakety do hash klasifikátoru ale do linkové vrstvy kde proběhne adresace.

5.3.3 Návrh třívrstvého komunikačního profilu se zahrnutím klasifikátoru

Jak bylo zjištěno v minulé sekci, vytvoření nového agenta k simulaci navrhovaného komunikačního profilu nestačí. Musí být proto prozkoumány další možnosti jako je vytvoření vlastního klasifikátoru, který bude pracovat spolu s navrženým agentem.

Třívrstvý komunikační profil s klasifikátorem je zobrazen na pravé straně Obr. 5.4. Klasifikátor je na obrázku označen červenou barvou. Klasifikátor bude pracovat jako mezivrstva mezi aplikační a linkovou vrstvou.

Klasifikátor je objekt, který směruje pakety k několika připojeným cílům. Klasifikace v něm probíhá podle předem daných kritérií (cílová adresa, číslo portu), kdy pakety spadající do stejné kategorie jsou poslány ke stejnému objektu. NS-2 implementuje klasifikátory použitím zástupných symbolů pro ukazatel na objekt, tzv. slotů (slot). Když paket dorazí do klasifikátoru, klasifikátor rozhodne o jeho typu a pošle ho do NS-2 objektu, jehož ukazatel je instalován v daném slotu. V novém návrhu třívrstvého komunikačního profilu by proto úkolem nově navrženého klasifikátoru bylo posílat všechny pakety odpovídající HDLC rámci, které byly vytvořené v navrženém agentu, přímo do linkové vrstvy, kde by jejich adresaci zprostředkoval protokol HDLC. V následujících krocích je stručně popsán obecný návrh nového klasifikátoru:

1. Návrh – Definovat zpracování paketu podle určitého kritéria (`criterion, slot, NSObject`. Pokud se paket shoduje s definovaným kritériem, paket se pošle do objektu, který je instalován v správném slotu `slot_[slot]` .
2. Konstrukce třídy – Odvodit novou C++ třídu (např. třída `HDLCClassifier`) z původní C++ třídy `Classifier` a vytvořit stínové OTcl třídy.
3. Interní mechanismy – Nahradit funkci `classify()` podle definice určené v prvním kroku.
4. Konfigurace – Nainstalovat, v Otcl doméně, `NSobject` do číslovaného slotu `slot` vytvořeného klasifikátoru `HDLCClassifier`.

Tento návrh se v rámci této diplomové práce v důsledku nedostatku času nepodařilo realizovat a předchozí kroky slouží jako ukázka dalšího možného postupu řešení implementace třívrstvého komunikačního profilu HDLC do NS-2 simulátoru.

6 VÝSLEDKY SIMULACE ZÍSKANÉ POMOCÍ KOMUNIKAČNÍCH PROFILŮ HDLC

Tato kapitola se zaměřuje na analýzu získaných výsledku simulace navržených komunikačních profilů. První část kapitoly popisuje základní kvalitativní parametry používané při simulaci sítí. V následující části kapitoly jsou popsány použité topologie pro simulaci satelitní implementace protokolu HDLC a HDLC komunikačního profilu využívajícího protokolů TCP/IP. Další část této kapitoly se věnuje zhodnoceni kvalitativních parametrů u satelitní implementace HDLC. V poslední část kapitoly se nachází zhodnocení kvalitativních parametrů HDLC komunikačního profilu využívajícího protokolů TCP/IP.

6.1 Kvalitativní parametry počítačových sítí

Protokoly vyšších vrstev, které používají služby poskytované síťovou vrstvou, předpokládají obdržení dokonalé služby, což však není v reálných podmínkách dosažitelné. Proto pro určování kvality komunikační sítě využíváme několik parametrů, mezi které patří například šířka pásma, zpoždění nebo propustnost [9].

6.1.1 Šířka pásma

Šířka pásma je jeden z parametrů, který určuje výkon počítačové sítě. Může být vyjádřena dvěma způsoby, v hertzech nebo v bitech za sekundu.[9].

- **Šířka pásma [Hz]** – Šířka pásma v hertzech udává rozsah frekvencí obsažených v kompozitním signálu nebo rozsah frekvencí, kterými může kanál přenášet. Například šířka pásma uživatele telefonní linky je 4 kHz [9].
- **Šířka pásma [b/s]** – Šířka pásma může také vyjadřovat počet přenesených bitů za sekundu kanálu, linky nebo dokonce sítě. Příkladem může být šířka pásma rychlé ethernetové sítě, která je maximálně 100 Mb/s.
- **Souvislost** – Mezi šířkou pásma vyjádřenou v hertzech a šířkou pásma vyjádřenou v bitech za sekundu existuje přímá úměra. Tedy zvýšíme-li jednu, zvýší se i druhá a naopak. Tato souvislost ještě závisí na použitém přenosu, tj. jestli používáme přenos s modulací nebo v základním pásu [9].

6.1.2 Propustnost

Propustnost je ukazatel s jakou rychlostí lze data posílat skrze počítačovou síť. Na první pohled se může zdát, že šířka pásma vyjádřená pomocí bitů za sekundu je stejná jako propustnost, ale není tomu tak. Linka může mít šířku pásma B b/s, ale

poslat může pouze T b/s, kdy platí $T < B$. Jinými slovy šířka pásma je potenciální měřítko linky a propustnost je reálné měřítko linky. Příkladem může být linka se šířkou pásma 1 Mb/s, na kterou jsou připojeny přístroje s přenosovou rychlostí 200 kb/s. To znamená, že lze skrz linku poslat pouze 200 kb/s, což je v tomto případě i hodnota její propustnosti [9].

6.1.3 Zpoždění

Zpoždění udává časový úsek, ze který, začínajíce odesláním prvního bitu, dorazí celá zpráva na místo určení. Lze říct, že se zpoždění skládá ze čtyř částí: čas propagace, čas přenosu, čas čekání a zpoždění způsobené zpracováním zprávy.

- **Čas propagace** – Čas propagace je doba, kterou bit potřebuje, aby ze zdroje dorazil k cíli. Můžeme ho definovat jako podíl vzdálenosti a propagační rychlosti.

$$\text{Čas propagace} = \frac{\text{Vzdálenost}}{\text{Propagační rychlost.}} \tag{6.1}$$

  Propagační rychlost u elektromagnetických signálů závisí na frekvenci signálu a přenosovém médiu.Maximální propagační rychlost $3{\cdot}10^8$ m/s signál dosahuje ve vakuu.Při přenosu vzduchem tato rychlost klesá, kdy v kabelech je propagační rychlost ještě nižší [9].
- **Čas přenosu** – V datových komunikacích neposíláme pouze jeden bit, ale celou zprávou. Přenos prvního i posledního bitu může odpovídat času propagace, ale nedochází k němu současně, a proto je nutné definovat časový úsek mezi odesláním prvního bitu a příchodem posledního bitu k příjemci, tzv. čas přenosu. Ten závisí na velikosti zprávy a šířce přenosového pásma linky.

$$\text{Čas přenosu} = \frac{\text{Velikost zprávy}}{\text{Šířka přenosového pásma.}} \tag{6.2}$$

- **Čas čekání** – Třetí částí zpoždění je čas čekání. Je to čas, po který je zpráva před zpracováním držena daným zařízením přenosové linky. Čas čekání není fixní mění se dynamicky se zatížením sítě, kdy platí přímá úměra. Zařízení slouží jako prostředník (např. router), příchozí zprávy zpracovává po jedné. Pokud tedy přijme mnoho zpráv najednou dochází ke zpoždění [9].
- **Čas zpoždění způsobené zpracováním zprávy** – Poslední část zpoždění vzniká při zpracování zprávy. Jedná se o čas, který při přijetí paketu síťový prvek potřebuje k odstranění hlavičky, provedení procedury pro odhalení chyb a doručení paketu na cílový port nebo protokolům vyšší vrstvy. Tato část zpoždění může být u každého paketu odlišná, a proto se většinou udává průměrná hodnota.

6.1.4 Ztrátovost

Dalším parametrem, který značně ovlivňuje výkon komunikační sítě, je počet paketů ztracených během přenosu. Pokud během zpracování paketu směrovač přijme jiný, přijatý paket musí být uschován ve vyrovnávací paměti. Tu má ale směrovač omezenou a může nastat případ, kdy je vyrovnávací paměť plná a následující příchozí paket musí být zahozen. Aby byl přenos spolehlivý, zahozený paket musí být znovu odeslán, což snižuje celkový výkon komunikační sítě. Ztrátovost procentuálně určuje kolik paketů je při přenosu ztraceno [9].

6.1.5 Jitter

Jitter velmi blízce souvisí se zpožděním. Obecně platí, že jitter je kolísání zpoždění přicházejících paketů. Negativní vliv tohoto parametru je nejprominentnější u aplikací, které jsou velmi citlivé na časovou změnu (například audio a video data přenášená v reálném čase) [9].

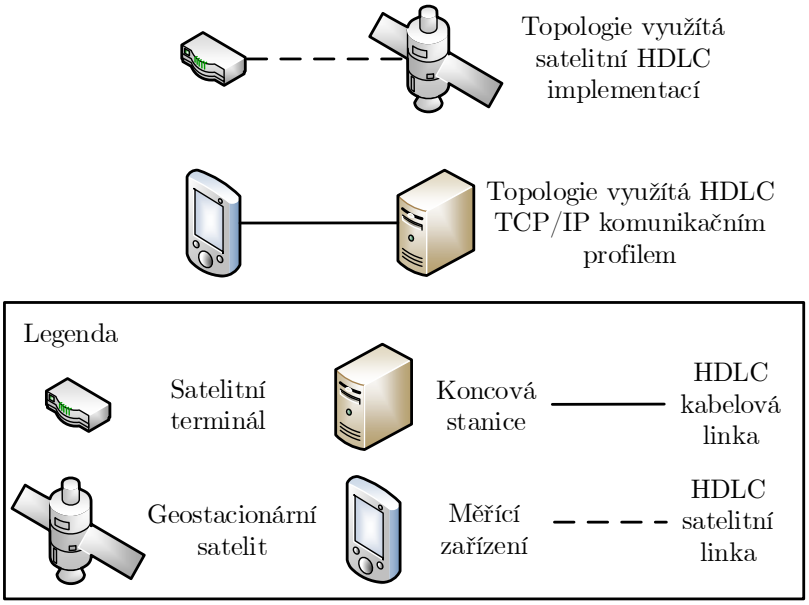
6.2 Návrh a implementace topologií v programu NS-2

V této sekci jsou popsány dvě různé topologie, které byly použity při testování parametrů protokolu linkové vrstvy HDLC. První testovaná topologie byla využita pro simulování parametrů satelitní implementace HDLC a druhá topologie byla využita pro simulování parametrů HDLC komunikačního profilu využívajícího protokolu TCP/IP pomocí kabelových linek. U této topologie je následně popsán Tcl kód využitý pro implementaci topologie v simulátoru NS-2. Nakonec jsou zmíněny rozdíly v kódu satelitní implementace. Navržené topologie jsou graficky zobrazeny na Obr. 6.1.

6.2.1 Specifikace použitých topologií

V této sekci budou popsaný použité topologie, které byly použity pro simulaci kvalitativních parametrů.

Topologie pro satelitní implementaci protokolu HDLC
Topologie použitá pro satelitní implementaci se skládá ze dvou uzlů, satelitního terminálu a geostacionárního satelitu. Tyto dva uzly jsou spojeny pomocí satelitní HDLC linky, na které se testují následující parametry:



Obr. 6.1: Topologie použité pro simulaci kvalitativních parametrů.

- **Průměrná propustnost linky** – U přenosu dat pomocí FTP na lince testujeme průměrnou propustnost v závislosti na šířce pásma linky. Průměrná propustnost se testuje i v závislosti na velikosti okna TCP.
- **Ztrátovost linky** – Druhý parametr, který na lince testujeme je ztrátovost paketů v závislosti na šířce pásma a velikosti okna TCP. Parametr se opět testuje pomocí protokolu FTP.

Satelitní terminál se nachází v městě Boston. To znamená 42,3° zeměpisné šířky a −71,1° zeměpisné délky. Geostacionární satelit obíhá zemi ve výšce 35 888 km nad rovníkem a má zeměpisnou délku −95°. Navrhovaná topologie je graficky zobrazena na Obr.6.1.

Topologie pro HDLC komunikační profil využívající protokol TCP/IP
Topologie HDLC komunikačního profilu využívajícího protokolu TCP/IP se skládá z koncové stanice a měřícího zařízení. Obě zařízení jsou teoretická, kdy při simulaci jsou tvořena uzlem programu NS-2, který pouze plní jejich funkci. Uzly spolu komunikují pomocí pozemní kabelového komunikace. Byl zkoumán následující kvalitativní parametr:

- **Ztrátovost linky** – Na lince pomocí protokolu FTP testujeme průměrnou ztrátovost v závislosti na zpoždění nebo šířce pásma.

6.2.2 Tcl implementace topologií

V této sekci bude dopodrobna popsáná druhá implementovaná topologie využívající HDLC komunikační profil používající TCP/IP protokol. Stručný návrh této topologie můžeme vidět na Obr. 6.1. Měřící stanice bude fungovat jako vysílač a koncová stanice bude přijímač. Simulace této topologie byla naprogramována pomocí Tcl skriptu (viz výpis 6.1).

Na prvním řádku kódu je inicializován hlavní objekt simulátoru NS-2, který má tři hlavní úlohy:

- Inicializuje formát paketu.
- Vytváří plánovač, který v simulaci řídí posílání paketů.
- Nastavuje výchozí adresní formát.

Na řádcích 3 až 9 jsou nastaveny parametry simulace jako je délka front. Řádek 10 pro HDLC protokol nastavuje režim opakování rámců. Pokud je hodnota proměnné `selRepeat` ve svém výchozím nastavení 0, HDLC používá opakování rámců s návratem (tzv. Go-Back-N). Při hodnotě proměnné `selRepeat` 1, protokol HDLC využívá selektivní opakování rámců.

Řádky 12 až 16 slouží k nastavení trasovacích funkcí simulátoru NS-2. Na 12. a 13. řádku se inicializuje program NAM, který slouží ke grafickému zobrazení nasimulovaných sítí. Následující tři řádky (14 – 16) slouží k nastavení textového trasovaní probíhající simulace. Na řádku 14 se nastavuje výstupní soubor, do kterého se ukládají vybraná data simulace. Řádky 15 a 16 simulátoru říkají, že má zaznamenávat všechny události, ke kterým v síti došlo. Řádky 18 až 24 tvoří deklarace funkce `finish()`. Tato funkce se provede po skončení simulace, kdy primárně slouží k výmazu trasovaných dat uložených v paměti.

Řádky 25 a 26 vytvářejí uzly `n1` a `n2`, z nichž je na řádku 28 vytvořena LAN síť. Funkce pro vytvoření sítě LAN byla podrobněji vysvětlena v kapitole 5. Tímto řádkem se vytvoří zkoumaná linka využívající HDLC protokol. Na řádcích 30 až 37 následuje vytvořeni a konfigurace TCP vysílače (agent) a přijímače (sink). Na řádku 30 se nastaví jaký typ TCP přenosu má být použit. Řádek 33 připojí agenta na uzel `n1` a řádek 35 přípojí přijímacího agenta (sink) na uzel `n2`. Logické spojení mezi vysílajícím a přijímajícím agentem se vytvoří na řádku 36.

Řádek 40 slouží k vytvoření FTP aplikace, která bude použita pro vytváření přenášených dat, kdy následující řádek ji připojí k vysílajícímu TCP agentu. Řádky 43 až 44 slouží jako ukazatel pro plánovač, který mu říká, kdy má spustit a vypnout FTP přenos. Celá simulace končí ve stejný čas jako FTP přenos a to spuštěním funkce `finish ()`. Simulace se spouští na řádku 47 příkazem `$ns run`.

Výpis 6.1: Implementace topologie pro HDLC komunikační profil využívající protokol TCP/IP

```
set ns [new Simulator]

Mac set bandwidth_ 1Mb
set val(bdp) 1500
LL/Sat/HDLC set window_size_ $val(bdp)
LL/Sat/HDLC set queue_size_ $val(bdp)
LL/Sat/HDLC set timeout_ 0.26
set qlen $val(bdp)
set val(ifqlen) $qlen
LL/Sat/HDLC set selRepeat_ 0

set nf [open graficky.nam w]
$ns namtrace-all $nf
set tracefd     [open Vys.tr w]
$ns trace-all $tracefd
$ns eventtrace-all

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
}
set n1 [$ns node]
set n2 [$ns node]

set lan [$ns newLan "$n1 $n2" 1Mb 10ms LL/Sat/HDLC Queue/DropTail MAC Channel]

set tcp [new Agent/TCP/Newreno]
Agent/TCP/Newreno set window_ $val(bdp)
$tcp set class_ 2
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink/Sack1]
$ns attach-agent $n2 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

$ns at 0.0 "$ftp start"
$ns at 50.0 "$ftp stop"
$ns at 50.0 "finish"

$ns run
```

Princip konfigurace satelitní implementace protokolu HDLC zůstává stejný jako u předchozí implementace. Konfigurace se liší pouze v detailech jako je například vytváření uzlů, které musí být u satelitní komunikace speciální. Tyto speciální uzly slouží k nastavení parametrů spojených se satelitním přenosem (zeměpisná šířka).

6.3 Simulace kvalitativních parametrů satelitní implementace HDLC

Tato sekce se zabývá simulací kvalitativních parametrů topologie využívající satelitní implementaci protokolu HDLC. Topologie byla navržena v programu NS-2. Studované kvalitativní parametry linky HDLC, které byly simulovány v závislosti na šířce pásma linky HDLC a velikosti okna TCP, jsou průměrná propustnost a ztrátovost HDLC rámců (paketů). Pro přenos dat byl použit prokol FTP. Velikost TCP okna byla nastavena 1500 paketů, kdy v případě chybného paketu mohou nastat maximálně tři opakování. Fronta HDLC linky byla nastavena do módu zahození konce (DropTail). Hodnoty šířky pásma, pro které byla simulace provedena, jsme zvolily tak, aby šlo sledovat snižování ztrátovosti paketů až na hodnotu 0 %. Hodnoty velikosti okna TCP se nastavovali tak, aby bylo možné sledovat podstatné změny kvalitativních parametrů.

6.3.1 Závislost průměrné propustnosti linky na šířce pásma

Výsledné průměrné hodnoty propustnosti linky jsou pro různé hodnoty šířky pásma uvedeny v Tab. 6.1, kdy testovány byly tři druhy satelitní linky:

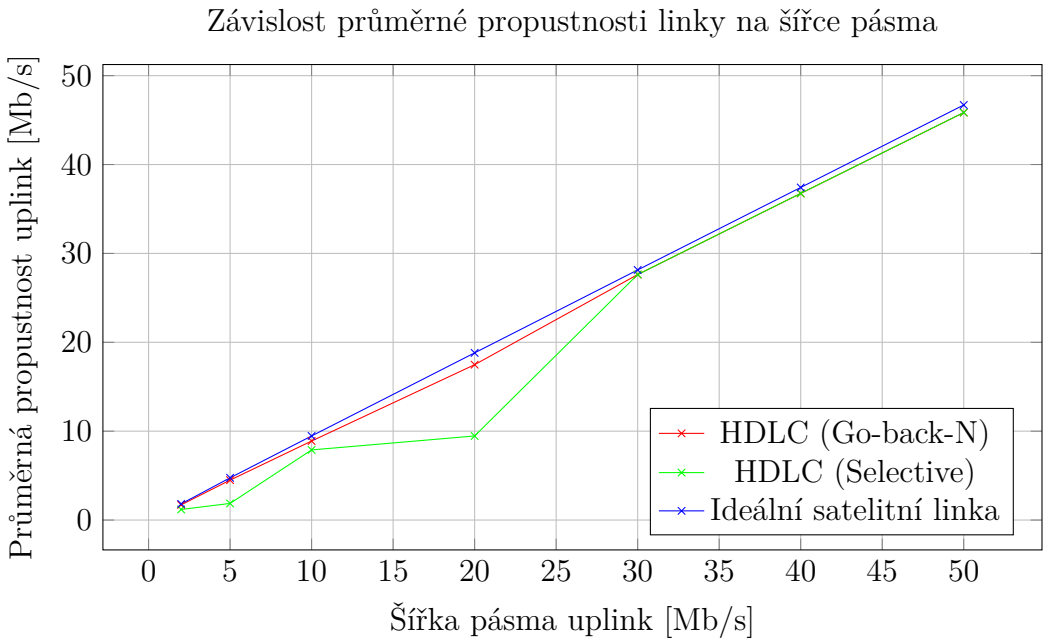
- HDLC využívající opakování rámců s návratem (Go-Back-N ARQ).
- HDLC využívající selektivní opakování rámců (Selective ARQ).
- Ideální Satelitní linka.

Je nutné si uvědomit, že propustnost byla simulována pouze ve směru odesílání. Data byla odesílána pomocí protokolu FTP, který přenáší po lince nekonečný soubor. Další faktor ovlivňující přesnost získaných hodnot je čas navázání TCP spojení (jedná se cca 2 s). Délka simulace byla u všech simulací uvedených v Tab. 6.1 50 s, kdy byly testovány tři druhy linek.

Tab. 6.1: Závislost průměrné propustnosti linky na šířce pásma.

Šířka pásma uplink [Mb/s]	**Průměrná propustnost uplink [Mb/s]**		
	HDLC (Go-back-N)	**HDLC (Selective)**	**Ideální satelitní linka**
2	1,71	1,19	1,82
5	4,49	1,87	4,75
10	8,89	7,89	9,46
20	17,48	9,46	18,81
30	27,62	27,62	28,14
40	36,75	36,75	37,42
50	45,85	45,85	46,7

Výsledky z Tab. 6.1 jsou vyneseny v grafu 6.2. Z grafu lze vyčíst, že nejlepších výsledků dosahuje ideální satelitní linka, která má propustnost skoro stejnou jako je příslušná šířka pásma. Ideální satelitní linka vykazuje pouze ztráty na fyzické vrstvě spojené se satelitním přenosem. V našem případě je ztrátovost pro všechny velikosti přenosového pásma nulová. Linka využívající protokol HDLC s opakováním rámců s návratem (Go-Back-N) dosahuje hodnoty průměrné propustnosti blízké ideální satelitní lince. Z grafu na Obr. 6.2 rovněž vyplývá, že linka využívající protokol HDLC se selektivním opakováním rámců (Selective) vykazuje menší hodnoty průměrné propustnosti a to hlavně na malých velikostech šířky pásma. U šířky pásma 30 Mb/s je průměrná propustnost všech použitých linek téměř shodná. Pokud je ztrátovost linky nulová (viz Tab. 6.2) je průměrná propustnost linky u obou HDLC linek prakticky totožná.



Obr. 6.2: Graf závislosti průměrné propustnosti linky na šířce pásma.

6.3.2 Závislost ztrátovosti linky na šířce pásma

Výsledky simulace ztrátovosti linky pro různé šířky pásma jsou uvedeny v Tab. 6.2. Ztrátovost byla opět simulována pro tři druhy satelitní linky a přenos dat pomocí protokolu FTP.

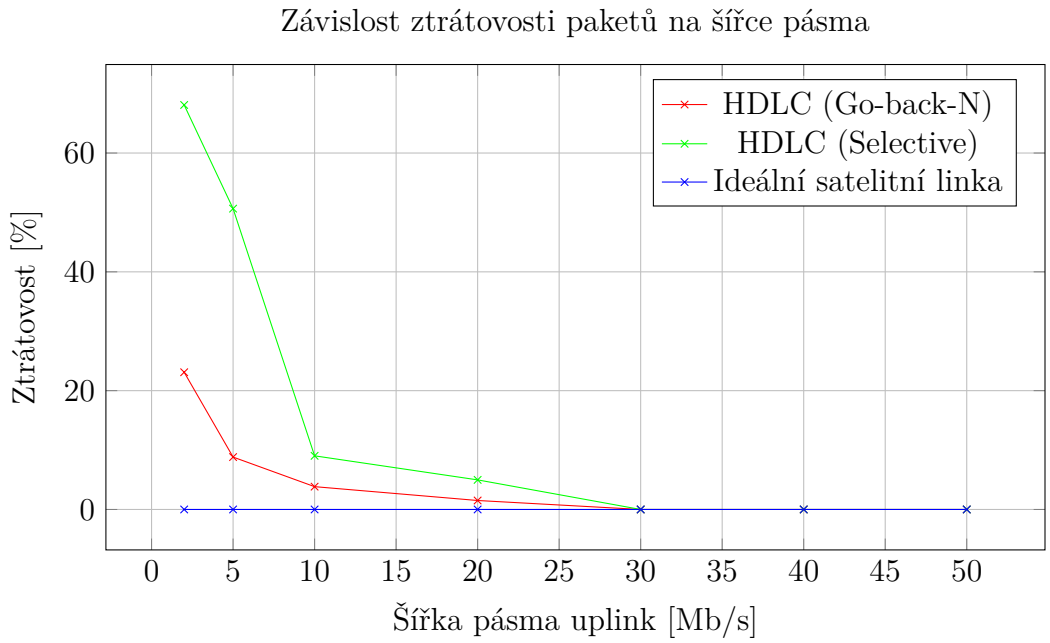
Výsledky simulace z Tab. 6.2 jsou vyneseny v grafu 6.3. Jak bylo uvedeno v sekci 6.3.1, ztrátovost ideální satelitní linky je pro všechny šířky pásma nulová. Linky využívající HDLC na malých šířkách pásma mají velmi ztrátovost vysokou.

Tab. 6.2: Závislost ztrátovosti rámců na šířce pásma.

Šířka pásma uplink [Mb/s]	**Ztrátovost [%]**		
	HDLC (Go-back-N)	**HDLC (Selective)**	**Ideální satelitní linka**
2	23,1	68,09	0
5	8,82	50,64	0
10	3,84	9,03	0
20	1,51	4,98	0
30	0	0	0
40	0	0	0
50	0	0	0

Ta však začíná s rostoucí šířkou pásma klesat, kdy pro 30 Mb/s a více je u obou technik opakování rámců už nulová. Vysoká ztrátovost při nižších hodnotách šířky pásma je způsobena kombinací vysokého propagačního zpoždění satelitu s omezenou velikostí bufferu. Ten se po určitém čase zaplní a začnou se zahazovat pakety. Ty se musí opakovaně přenést, což ale z důvodu vysokého propagačního zpoždění nějakou chvíli trvá a ztrátovost linky tím roste. Tento efekt s rostoucí šířkou pásma klesá, kdy nakonec je přenos paketů natolik rychlý, že fronta HDLC nemá problém všechny pakety přicházející v TCP odbavit a ztrátovost klesá na nulu.

V souladu s výsledky propustnosti (viz Obr. 6.2) je u malých hodnot šířky pásma ztrátovost HDLC se selektivním opakováním rámců mnohem vyšší než ztrátovost HDLC s opakováním rámců s návratem. Tento rozdíl je způsoben rozdílným mechanismem opakování špatných paketů, kdy HDLC s opakováním rámců s návratem při zaznamenání špatného přenosu opakuje bloky dat. HDLC se selektivním opakováním opakuje pouze chybné pakety a data si ukládá do bufferu.



Obr. 6.3: Graf závislosti ztrátovosti rámců na šířce pásma.

6.4 Simulace kvalitativních parametrů HDLC komunikačního profilu využívajícího TCP/IP protokoly

Tato sekce se věnuje diskuzi výsledků simulací kvalitativních parametrů HDLC komunikačního profilu využívajícího TCP/IP protokoly, kdy použitá topologie byla navržena v programu NS-2. Obdobně jako u simulace využívající satelitní implementaci byla i v případě použití komunikačního profilu s TCP/IP protokolem simulována ztrátovost rámců, která v tomto případě závisí na zpoždění a šířce pásma linky HDLC. Výsledky těchto dvou simulací se spolu přímo srovnávat nedají, jelikož u satelitní implementace neexistuje možnost nastavení zpoždění, které se v jejím případě musí dopočítávat z propagačního zpoždění a vzdálenosti satelitů.

Zkoumaný kvalitativní parametr linky HDLC je ztrátovost rámců. Ztrátovost rámců je simulována v závislosti na zpoždění a šířce pásma linky HDLC. Pro přenos dat simulace používá prokol FTP. Velikost TCP okna byla nastavena 1500 paketů, kdy jsou povoleny maximálně tři opakování chybného paketu. Fronta HDLC linky byla nastavena do módu zahození konce (DropTail) a měla velikost 1500 rámců. Protokol HDLC měl při simulaci nastavený časový limit pro opakovaní rámců (*timeout*) na hodnotu 260 ms.

Hodnoty zpoždění byly zvoleny tak, aby jejich změny byly dobře rozlišitelné a to

až do hodnoty, kdy se ztrátovost rámců ustálí na konstantní úrovni cca 1 Mb/s. Obdobná filozofie byla použita také pro výběr hodnot šířky pásma linky.

6.4.1 Závislost ztrátovosti HDLC linky na zpoždění

Výsledky získané simulací ztrátovosti linky v závislosti na zpoždění jsou zaznamenány v Tab.6.3. Při simulaci byl po lince protokolem FTP přenášen nekonečný soubor, kdy její délka byla nastavena na 50 s. Ztrátovost rámců byla simulována pro šířky pásma 0,1 až 1 Mb/s u kterých je nastaveno zpoždění 10 až 260 ms.

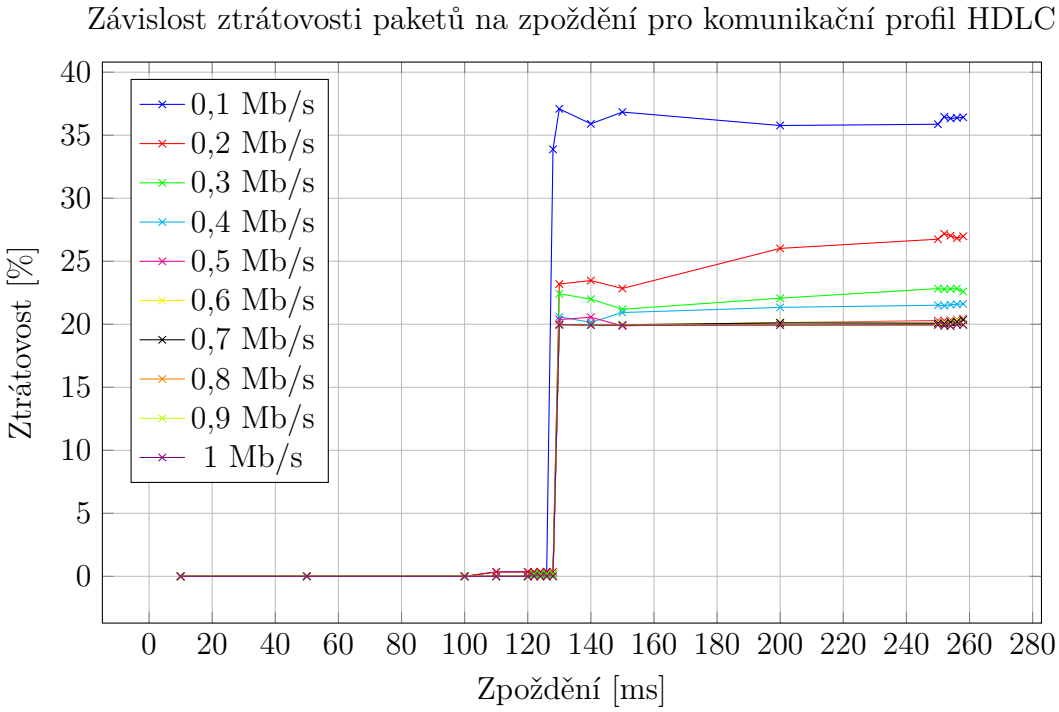
Tab. 6.3: Závislosti ztrátovosti paketů na zpoždění pro komunikační profil HDLC.

Šířka pásma [Mb/s]	**0,1**	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**	**1**
Zpoždění [ms]	**Ztrátovost [%] (Za 50 s)**									
10	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0	0
110	0,35	0,35	0	0	0	0	0	0	0	0
120	0,35	0,35	0	0	0	0	0	0	0	0
122	0,35	0,35	0,18	0	0	0	0	0	0	0
124	0,35	0,35	0,18	0	0	0	0	0	0	0
126	0,35	0,36	0,18	0	0	0	0	0	0	0
128	0,36	0,35	0,19	0	0	0	0	0	0	0
130	33,88	23,19	22,42	20,59	20,37	19,99	20,22	19,97	19,97	19,96
140	37,09	23,47	22	20,15	20,56	19,93	19,94	19,95	19,95	19,94
150	35,9	22,84	21,19	20,93	19,87	19,91	19,96	19,94	19,94	19,92
200	36,84	26,02	22,07	21,34	20,12	20,15	20,11	19,92	19,93	19,94
250	35,77	26,75	22.83	21,51	20,29	20,18	20,08	19,95	19,95	19,95
252	35,87	27,18	22.78	21,47	20,29	20,16	20,05	19,89	19,88	19,88
254	36,46	27,03	22,8	21,56	20,33	20,26	20,17	19,87	19,87	19,86
256	36,33	26,83	22,84	21,58	20,37	20,32	20,2	19,98	19,99	19,97
258	36,38	26,98	22,6	21,62	20,45	20,39	20,35	19,96	19,96	19,95
260	-	-	-	-	-	-	-	-	-	-

Výsledky simulací jsou graficky znázorněny na Obr. 6.3. Z něho lze vyčíst, že na hodnotě zpoždění 130 ms dochází k skokovému nárůstu ztrátovosti paketů. Skokový charakter nárůstu je potvrzen i snížením kroku simulace, který je v této oblasti (120 až 130 ms) 2 ms. Po skoku na 130 ms se hodnoty ztrátovosti ustáli na konstantní hodnotě, kdy simulace pro hodnoty vyšší než 260 ms selhává, jelikož dochází k překročení nastaveného časového limitu pro opakování HDLC rámců.

Nastavení šířky pásma výsledky simulace ovlivňuje jen při zpožděních vyšších než 130 ms a šířkách pásma menších než 0,6 Mb/s, kdy obdobně jako u satelitní

implementace s rostoucí šířkou pásma klesá ztrátovost, kdy její minimum je v tomto případě kolem 20 %.



Obr. 6.4: Graf závislosti ztrátovosti paketů na zpoždění.

6.4.2 Závislost ztrátovosti HDLC linky na šířce přenosového pásma

Výsledky závislosti ztrátovosti linky na zpoždění jsou zaznamenány v Tab.6.4.Data byla odesílána pomocí protokolu FTP, který přenáší po lince nekonečný soubor. Ztrátovost rámců byla simulována pro šířky pásma 0,01 až 1 Mb/s. Opět byla použita délka simulace 50 s, kdy byly analyzovány tři různé scénáře:

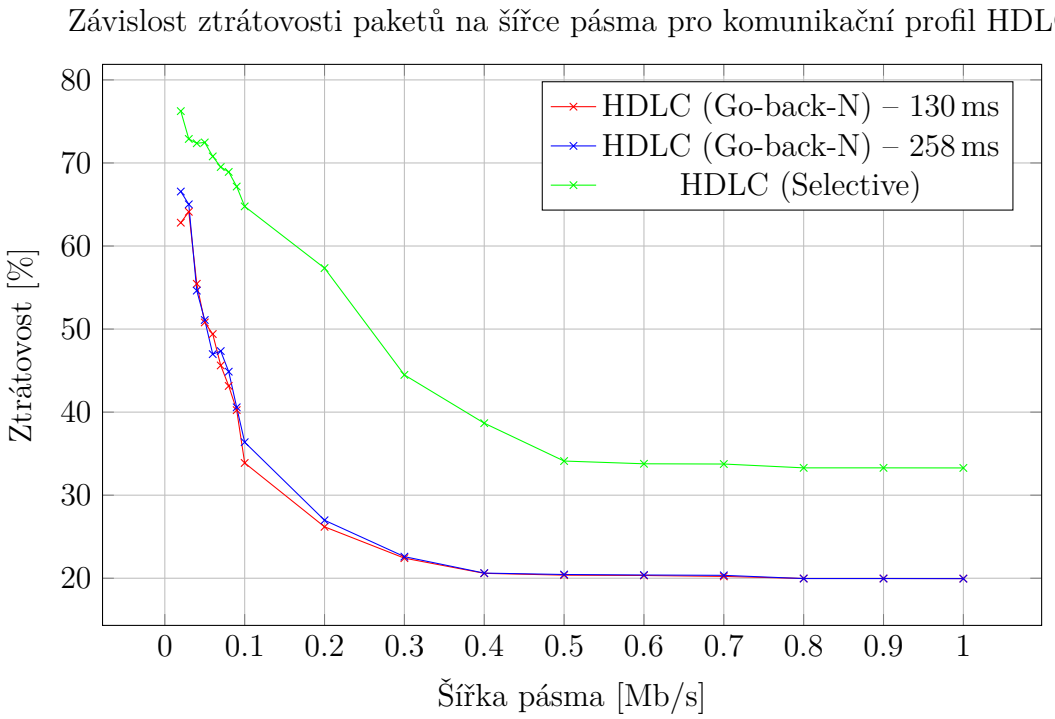
1. HDLC využívající opakování rámců s návratem (Go-Back-N ARQ) se zpožděním 130 ms.
2. HDLC využívající selektivní opakování rámců (Go-Back-N ARQ) se zpožděním 258 ms.
3. HDLC využívající selektivní opakování rámců (Selective ARQ).

Volba nastavení prvních dvou scénářů má za úkol určit, do jaké míry při nízkých šířkách pásma ovlivňuje volba zpoždění velikost ztrátovosti. Pro lepší ilustraci jsou získané hodnoty vyneseny do grafu na Obr. 6.5. Z grafu je na první pohled jasné, že rozdíly ve ztrátovosti těchto dvou scénářů jsou malé ($< 3\,\%$) a vliv šířky pásma je po

Tab. 6.4: Závislost ztrátovosti paketů na šířce pásma linky pro komunikační profil HDLC.

Šířka pásma [Mb/s]	**0,01**	**0,02**	**0,03**	**0,04**	**0,05**	**0,06**	**0,07**	**0,08**	**0,09**	**0,1**	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**	**1**
HDLC režim	**Ztrátovost [%] (Za 50 s)**																		
HDLC (Go-back-N) 130 ms	-	62,81	64,12	52,45	50,8	49,4	45,62	43,15	40,24	33,88	26,19	22,42	20,59	20,37	20,33	20,22	19.97	19,97	19,96
HDLC (Go-back-N) 258 ms	-	66,56	65,02	54,63	51,09	46,98	47,35	44,87	40,58	36,38	26,98	22,6	20,62	20,45	20,39	20,35	19.96	19,96	19,95
HDLC (Selective)	-	76,24	72,89	72,37	72,47	67,79	69,51	68,92	67,16	64,77	57,34	44,48	38,67	34,11	33,78	33,75	33,29	33,29	33,28

překročení hranice zpoždění 130 ms zanedbatelný. Třetí scénář dosahoval mnohem horších výsledků než předchozí dva scénáře a to řádově o 15 – 20 %. Tento rozdíl je dán rozdílným mechanizmem opakovaní rámců v Go-Back-N ARQ a v Selective ARQ, který byl už vysvětlen v podkapitole 6.3.



Obr. 6.5: Graf závislosti ztrátovosti paketů na šířce pásma.

7 ZÁVĚR

Cílem diplomové práce bylo seznámit se s problematikou inteligentních elektrických sítí a využití protokolu HDLC ke komunikaci a adresaci inteligentních měřících zařízení. Tato rešerše byla použita ke zvolení nejvhodnějšího síťového simulátoru, do kterého byly implementovány dva typy HDLC komunikačních profilů. První profil využívá ke své adresaci protokolu TCP/IP a protokol HDLC plní pouze funkci linkové vrstvy. Druhý profil je založen na třech vrstvách, kde adresaci měřících zařízení zprostředkovává přímo HDLC protokol v linkové vrstvě. Použitím jednoduché topologie byly pro první komunikační profil nasimulovány a vyhodnoceny závislosti ztrátovosti rámců na velikosti zpoždění a šířky pásma simulované linky.

Z dostupných simulačních programů byl na základě výhod a nevýhod zvolen ten nejvhodnější, a to program NS-2. Jeden z hlavních důvodů proč byl simulační program zvolen, byla již existující implementace protokolu HDLC v satelitních sítích. Po jejím důkladném prostudování bylo zjištěno, že z důvodu nemožnosti nastavení vyžadovaného zpoždění satelitní linky, tato implementace není vhodná pro realizaci komunikačních profilů využívajících HDLC. Proto musela být upravena tak, aby odpovídala prvnímu HDLC komunikačnímu profilu využívající TCP/IP protokol. Díku tomu, že uzel využívaný k vytváření LAN sítí obsahuje rozšířenou část uzlu a podporuje kabelové spoje, byl vybrán jako uzel použitý pro implementace navrhovaných komunikačních profilů. Ve druhém kroku úpravy byla změněna C++ třída „sat-hdlc.cc“, aby nevyužívala žádné části spojené se satelitním přenosem (např. směrování). Následně byla ještě upravena trasovací třída „trace.cc“ tak, aby speciální trasovací formát používaný při trasování protokolu HDLC byl použit i při trasování kabelových spojů využívajících protokolu HDLC v LAN síti.

Rovněž byly prozkoumány možnosti implementace třívrstvého HDLC komunikačního profilu v programu NS-2 a na základě těchto možností byl navržen postup jeho implementace. Postup byl rozdělen na dva hlavní kroky. Prvním krokem, byla úprava C++ třídy „sat-hdlc.cc“, HDLC komunikačního profilu využívající TCP/IP protokol, aby nepoužívala IP adresaci, ale adresaci zprostředkovanou HDLC protokolem s využitím obecného záhlaví. Následně byl navržen třívrstvý HDLC komunikační profil využívající nového agenta v programu NS-2. Po prozkoumání tohoto návrhu bylo shledáno, že navržený profil nebude funkční, proto byl navržen nový třívrstvý HDLC komunikační profil doplněný o klasifikátor, který bude pracovat spolu s agentem. Z důvodu časové náročnosti implementace obou HDLC profilů nebylo možné tento navržený profil realizovat, ale postup pro realizaci tohoto profilu je popsán v této práci.

Pro satelitní implementaci protokolu HDLC byly odsimulovány závislosti ztrátovosti rámců a průměrné propustnosti na velikosti šířky pásma. Pro HDLC komu-

nikační profil využívající TCP/IP protokol byly odsimulovány závislosti ztrátovosti paketů na zpoždění a velikosti šířky pásma. Z výsledků simulací vyplývá, že HDLC je efektivnější při vyšší šířce pásma. Při malých šířkách pásma vykazovala vysokou ztrátovost (například pro šířku pásma 0.02 Mb/s vykazují výsledky simulace ztrátovost 35, 9 % při zpoždění 150 ms). Ze dvou použitých režimů opakování rámců (Go-Back-N a HDLC se selektivním opakováním rámců) měl lepší výsledky režim Go-Back-N. Selektivní opakování rámců vykazovala řádově o 15 – 20 % vyšší ztrátovost oproti opakovaní rámců (Go-back-N). Z výsledků pro HDLC profil vyplývá, že pokud bylo zpoždění menší než 130 ms nevykazuje linka žádnou ztrátovost. Po dosažení zpoždění o velikosti 130 ms nastane skokový nárůst ztrátovosti o přibližně 20 % při šířce pásma 1 Mb/s. Tato hodnota byla poloviční vůči hodnotě časového limitu pro opakovaní rámců, který byl nastaven na 260 ms. To znamená, že do poloviční hodnoty časového limitu pro opakovaní rámců nedocházelo k žádnému zahazovaní paketů, proto linka nevykazovala žádnou ztrátovost. Pokud byla hodnota nastaveného zpoždění linky rovná hodnotě časového limitu pro opakovaní rámců (260 ms), neproběhla inicializační výměna rámců a komunikace mezi vysílačem a přijímačem se ani nenaváže.

LITERATURA

[1] KUZLU, Murat, Manisa PIPATTANASOMPORN a Saifur RAHMAN. Communication network requirements for major smart grid applications in HAN, NAN and WAN. *Computer Networks [online]*. 2014, 67, 74-88 [cit. 2018-05-16]. DOI: 10.1016/j.comnet.2014.03.029. ISSN 13891286. Dostupné z: `http://linkinghub.elsevier.com/retrieve/pii/S1389128614001431`

[2] JIANG, Jing a Yi QIAN. Distributed Communication Architecture for Smart Grid Applications. *IEEE Communications Magazine* [online]. 2016, **54**(12), 60-67 [cit. 2018-05-16]. DOI: 10.1109/MCOM.2016.1600321CM. ISSN 0163-6804. Dostupné z: `http://ieeexplore.ieee.org/document/7786112/`

[3] CHU, Chia-Chi a Herbert Ho-Ching IU. Complex Networks Theory For Modern Smart Grid Applications: A Survey. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* [online]. 2017, **7**(2), 177-191 [cit. 2018-05-16]. DOI: 10.1109/JETCAS.2017.2692243. ISSN 2156-3357. Dostupné z: `http://ieeexplore.ieee.org/document/7908993/`

[4] SAMAD, Tariq a Anuradha M. ANNASWAMY. Controls for Smart Grids: Architectures and Applications. *Proceedings of the IEEE* [online]. 2017, **105**(11), 2244-2261 [cit. 2018-05-16]. DOI: 10.1109/JPROC.2017.2707326. ISSN 0018-9219. Dostupné z: `http://ieeexplore.ieee.org/document/7967650/`

[5] NASIRI, Baktash, Christian WAGNER, Ulf HÄGER a Christian REHTANZ. Distribution grid planning considering smart grid technologies.*CIRED - Open Access Proceedings Journal* [online]. 2017, **2017**(1), 2228-2232 [cit. 2018-05-16]. DOI: 10.1049/oap-cired.2017.0991. ISSN 2515-0855. Dostupné z: `http://digital-library.theiet.org/content/journals/10.1049/oap-cired.2017.0991`

[6] U. Ahsan and A. Bais, Distributed Smart Home Architecture for Data Handling in Smart Grid. *Canadian Journal of Electrical and Computer Engineering*, 2018, vol. 41, no. 1, pp. 17-27 [cit. 2018-05-16]. DOI: 10.1109/CJECE.2017.2776975. Dostupné z: `https://ieeexplore.ieee.org/document/8341516/`

[7] XU, Shengjie, Yi QIAN a Rose QINGYANG HU. Reliable and resilient access network design for advanced metering infrastructures in smart grid. *IET Smart Grid* [online]. 2018, **1**(1), 24-30 [cit. 2018-05-16]. DOI: 10.1049/iet-stg.2018.0008. ISSN 2515-2947. Dostupné z: `http://digital-library.theiet.org/content/journals/10.1049/iet-stg.2018.0008`

[8] PAL, A. *DATA COMMUNICATION AND COMPUTER NETWORKS*. PHI Learning, 2013. ISBN 9788120348455.

[9] FOROUZAN, Behrouz A. *Data communications and networking*. 5th ed. New York: McGraw-Hill, c2013. ISBN 978-0073376226.

[10] MARTINI, L., ROSEN, E., HERON, G., and MALIS A., *Encapsulation Methods for Transport of PPP/High-Level Data Link Control (HDLC over MPLS Networks"*, RFC 4618, September 2006.

[11] LIMPHAPAYOM, Siwarat a Wanchalerm PORA. An emulation of data concentrator units conformed to DLMS-HDLC protocols. In: *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology* [online]. IEEE, 2013, 2013, s. 1-6 [cit. 2018-05-16]. DOI: 10.1109/ECTICon.2013.6559622. ISBN 978-1-4799-0545-4. Dostupné z: `http://ieeexplore.ieee.org/document/6559622/`

[12] COSEM Architecture and Protocol - DLMS User Association *Companion Specification for Energy Metering* [online]. 2018 [cit. 2018-11-02]. Dostupné z: `http://dlms.com/documents/archive/Excerpt_GB4.pdf`

[13] VYAS, Divyang Advance Metering Infrastructure and DLMS/COSEM Standards for Smart Grid. V: *International Journal of Engineering Research & Technology (IJERT)*. 2012. 12., 1.

[14] MISHRA, Vinita and JANGALE, Smita *Analysis and comparison of different network simulators*. In: International Journal of Application or Innovation in Engineering & Management. 2014.

[15] GUPTA, Suraj G.; PARAG, D. Thakare a Dr. P. M. Jawandhiya. Open-Source Network Simulation Tools: An Overview. V: *International Journal of Advanced Research in Computer Engineering & Technology*. 2012. l., 2(4). ISSN 2278 – 1323.

[16] RAHMAN, Muhammad Azizur, Algirdas PAKŠTAS a Frank Zhigang WANG. Network modelling and simulation tools. *Simulation Modelling Practice and Theory* [online]. 2009, **17**(6), 1011-1031 [cit. 2018-05-16]. DOI: 10.1016/j.simpat.2009.02.005. ISSN 1569190X. Dostupné z: `http://linkinghub.elsevier.com/retrieve/pii/S1569190X09000197`

[17] SULIMAN, Islam Abdelmuti a Ghassan Mohammed Taha ABDALLA. Comparative Study of wireless LAN using OPNET and NS-2. In: *2016 Conference of Basic Sciences and Engineering Studies (SGCAC)* [online]. IEEE, 2016, 2016,

s. 196-200 [cit. 2018-05-16]. DOI: 10.1109/SGCAC.2016.7458029. ISBN 978-1-5090-1812-3. Dostupné z: http://ieeexplore.ieee.org/document/7458029/

[18] PATEL, Rajan and KAMBOJ, Pariza *Investigation of Network Simulation Tools and Comparison Study: NS3 vs NS2*. In: Transactions. 2014. 14. 15.

[19] ISSARIYAKUL, Teerawat. a Ekram HOSSAIN. *Introduction to network simulator NS2*. 2nd ed. New York: Springer, c2012. ISBN 1461414059.

[20] The Network Simulator - ns-2 [online]. [cit. 2018-5-07]. Dostupné z: https://www.isi.edu/nsnam/ns/

[21] NAYAK, Ajit Kumar, Satyananda Champati RAI a Rajib MALL. *Computer Network Simulation Using NS2* [online]. Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742: CRC Press, 2016 [cit. 2018-05-16]. ISBN 978-1-4987-6854-2.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ABME	Asynchronous Balanced Mode Extended – rozšířený asynchronní vyvážený mód
ACK	Acknowledgement – potvrzení
AODV	Ad hoc On-Demand Distance Vector Routing – směrovací protokol pro mobilní sítě
APDU	Application Protocol Data Unit – aplikační datová jednotka
ARM	Asynchronous Response Mode – režim asynchronní odezvy
ARQ	Automatic Repeat-reQuest – automatické opakování
BAN	Building Area Network – síť budov
BGP	Border Gateway Protocol – dynamický směrovací protokol
BSD	Berkeley Standard Distribution – operačního systému Unix distribuovaný Kalifornskou univerzitou v Berkeley
CBR	Constant Bit Rate – konstantní bitový tok
CDMA	Code Division Multiple Access – kódový multiplex
COSEM	Companion Specification for Energy Metering – specifikace pro inteligentní měření
CRC	Cyclic Redundancy Check – kontrolní součet
CSMA	Carrier Sense Multiple Access – pravděpodobnostní protokol přístupu k médiu
DDCP	Datagram Congestion Control Protocol – protokol transportní vrstvy
DISC	(Disconnect – rámec s příkazem odpojení
DLMS	Device Language Message Specification – specifikace aplikačních zpráv
DSDV	Destination-Sequenced Distance Vector routing – směrovací schéma pro mobilní sítě
DSL	Digital Subscriber Line – technologie přenosu dat
DV	Distance-Vector – dálkově vektorové směrovací protokoly
DSR	Dynamic Source Routing – směrovací protokol pro mobilní sítě
FAN	Field Area Network – síť polních oblastí
FTP	File Transfer Protocol – protokol pro přenos dat
GPRS	General Packet Radio Service – univerzální paketová rádiová služba
GSM	Groupe Spécial Mobile – globální systém pro mobilní komunikaci
HAN	Home Area Network – domácí síť
HDLC	High-Level Data Link Control – protokol vysokoúrovňového řízení datového spoje
HTTP	Hypertext Transfer Protocol – protokol pro výměnu hypertextových dat
IAN	Industrial Area Network – průmyslová síť

IGMP	Internet Group Management Protocol– protokol správy internetové skupiny
IP	Internet Protocol – internetový protokol
IS-IS	Intermediate System to Intermediate System – směrovací protokol
LAN	Local Area Network – lokální síť
LDP	Label Distribution Protocol – protokol pro přepínání sítí
LS	Link-State – druh směrovacích protokolů
MAC	Media Access Control – podvrstva linkové vrstvy
MANET	Mobile Ad Hoc Network – směrovací protokol pro mobilní sítě
MIP	Mobile Internet Protocol – mobilní internetový protokol
MPI	Message Passing Interface – rozhraní pro přenos zpráv
MPLS	Multiprotocol Label Switching – multiprotokolové přepojování podle návěští
NAK	Negative Acknowledgement – negativní potvrzení
NAM	Network Animator – animační doplněk programu NS-2
NAN	Neighborhood Area Network – síť sousedních oblastí
NAT	Network Address Translation – překlad síťových adres
NRM	Normal Response Mode – režim normální odezvy
NS-2	Network Simulator 2 – síťový simulátor verze 2
NS-3	Network Simulator 3 – síťový simulátor verze 3
OPNET	Optimized Network Engineering Tool – optimalizovaný nástroj pro síťovou techniku
OSPF	Open Shortest Path First – hierarchický interní směrovací protokol
OTcl	Object Tool command language – objektově založený programovací jazyk
PGM	Pragmatic General Multicast – protokol pro přenos dat u technologie multicast
PIM-SM	Protocol Independent Multicast – směrovací protokol určený pro multicast
PLC	Power Line Carrier – technologie přenosu dat
PPP	Point-to-Point Protocol – protokol bod-bod
P2P	Peer-to-peer – klient-klient
RAP	Route Access Protocol – protokol pro distribuci směrovacích informací
RED	Random Early Detection – náhodná brzká detekce
REJ	Reject – odmítnout
RIP	Routing Information Protocol– směrovací protokol
RLM	Receiver-driven Layer Multicast – protokol pro spolehlivý multicast
RNR	Receive Not Ready – odpověď není připravena
RR	Receive Ready – odpověď připravena

RTP	Real-time Transport Protocol – protokol pro přenos dat v reálném čase
SABME	(Set Asynchronous Balanced Mode Extended – nastavit rozšířený asynchronní vyvážený mód
SCTP	Stream Control Transmission Protocol – protokol pro kontrolu telefonní signalizace
SR	Selective Repeat – selektivní opakování
SREJ	Selective Reject – selektivní odmítnutí
SRM	Scalable Reliable Multicast – protokol pro spolehlivý multicast
SRR	Single Ring Recovery Protocol – protokol pro obnovu sítě
TDMA	Time Division Multiple Access – časový multiplex
TCP	Transmission Control Protocol – spolehlivý protokol transportní vrstvy
TFRC	TCP Friendly Rate Control – protokol pro kontrolu přenášených dat
TORA	Temporally Ordered Routing Algorithm – směrovací protokol pro mobilní sítě
UDP	User Datagram Protocol – nespolehlivý protokol transportní vrstvy
VINT	Virtual Inter Network Testbed – virtuální interní testovací síť
WAN	Wide Area Network – rozsáhlá síť
WFQ	Weighted Fair Queueing – vážená fronta
xDLMS	extended Device Language Message Specification – rozšířená specifikace aplikačních zpráv
XCP	Universal Measurement and Calibration Protocol – protokol pro spojení kalibračních jednotek
YANS	Yet Another Network Simulator – síťový simulátor
bit	Jednotka digitální velikosti
Mb	Jednotka digitální velikosti
m	Jednotka délky
km	Jednotka délky
m/s	Jednotka rychlosti
Hz	Jednotka frekvence
kHz	Jednotka frekvence
b/s	Jednotka datového přenosu
kb/s	Jednotka datového přenosu
Mb/s	Jednotka datového přenosu
Gb/s	Jednotka datového přenosu
s	Jednotka času
ms	Jednotka času
pk	Paket

SEZNAM PŘÍLOH

A Prostředky využité při implementaci **94**

B Obsah přiloženého DVD **95**

A PROSTŘEDKY VYUŽITÉ PŘI IMPLEMENTACI

- **Program Oracle VM VirtualBox**
  - Verze použitého programu: 5.2.0 r118431 (Qt5.6.2).
  - Program dostupný z : `https://www.virtualbox.org/wiki/Downloads`
- **Operační systém Ubuntu**
  - Verze použitého operačního systému: 16.04 LTS.
  - Operační systém dostupný z : `http://releases.ubuntu.com/16.04/`
- **Vývojové prostředí Eclipse**
  - Verze použitého vývojového prostředí: Oxygen.3 Release (4.7.3RC3p)
  - Operační systém dostupný z : `https://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/oxygen3a`
- **Simulační program NS-2**
  - Verze simulačního programu: 2.35.
  - Operační systém dostupný z : `https://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.35/`

B OBSAH PŘILOŽENÉHO DVD

- `Diplomová práce` – Adresář s PDF verzí diplomové práce.
- `Zdrojové kódy` – Adresář obsahující upravené zdrojové kódy.
  - `Trasovací třída` – Adresář obsahující zdrojové kódy upravené trasovací třídy.
  - `Satelitní implementace HDLC` – Adresář obsahující zdrojové kódy původní satelitní implementaci protokolu HDLC.
  - `LAN` – Adresář obsahující zdrojové kódy upravené třídy využívané při tvoření LAN sítě.
  - `HDLC only` – Adresář obsahující zdrojové kódy HDLC třívrstvého komunikačního profilu.
  - `HDLC with IP` – Adresář obsahující zdrojové kódy HDLC komunikačního profilu využívajícího protokolu TCP/IP.

Aplikování navržených komunikačních profilů v simulátoru NS-2

1. Přenesení upravené třídy „vlan.tcl“ do složky „~ns/tcl/lan“, ve které bude nahrazen původní třída „vlan.tcl“.
2. Přenesení upravené třídy „trace.cc“ a jeho hlavičkového souboru „trace.h“ do složky „~ns/trace“, ve které bude nahrazena původní třída.
3. Přenesení třídy vybraného komunikačního profilu „trace.cc“ a jeho hlavičkového souboru „trace.h“ do složky „~ns/satellite“, ve které bude nahrazena původní třída.
4. Nová kompilace simulátoru NS-2 s parametrem `make`.