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

SYSTÉM PRO DETEKCI RÁMCE GPON

GPON FRAME DETECTION SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Martin Holík
AUTHOR

VEDOUCÍ PRÁCE — Ing. Tomáš Horváth, Ph.D.
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. Martin Holík ***ID:*** 164284
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Systém pro detekci rámce GPON

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je návrh a implementace řešení pro detekci chyb v rámcích GPON s využitím databázového systému SQL. V teoretické části se seznamte s problematikou databázových systémů a detailněji s rámci GPON a výskytem jejich chyb. V praktické části se zaměřte na systém Microsoft SQL. V tomto typu databázového systému dle pokynů navrhněte a implementujte databázovou strukturu rámců GPON. Výsledná aplikace či sada skriptů v jazyce Python by měla umožňovat čtení dat z navržené databáze SQL a analýzu vybraných chyb rámců GPON z teoretické části.

DOPORUČENÁ LITERATURA:

[1] LACKO, Ľuboslav. Mistrovství v SQL Server 2012: [kompletní průvodce databázového experta]. Brno: Computer Press, 2013. ISBN 978-80-2513-773-4.

[2] Microsoft. Data Mining Tutorials (Analysis Services), 2017, [online], Dostupné z: https://docs.microsoft.com/en-us/sql/analysis-services/data-mining-tutorials-analysis-services

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

Vedoucí práce: Ing. Tomáš Horváth, Ph.D.
Konzultant:

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

ABSTRAKT

Tato diplomová práce je zaměřena na systém pro detekci GPON rámců. V práci je teoreticky rozebrána problematika související se s optickými sítěmi, návrhem a správou databázového systému. Jako praktický výstup práce je navržen systému umožňující detekci rámců a vytvořen skript pro analýzu přenášeného provozu.

KLÍČOVÁ SLOVA

Databáze, databázový systém, GPON, Microsoft SQL, pasivní optická síť, Python, SQL

ABSTRACT

This diploma thesis deals with GPON frame detection system. Partial problems of designing databases, optical networks and management of database system are described in theoretical parts. Practical parts this thesis are focused on design of system for detecting GPON frames and script for analysing of traffic.

KEYWORDS

Database, database system, GPON, Microsoft SQL, passive optical network, Python, SQL

Bibliografická citace

HOLÍK, M. *Systém pro detekci rámce GPON* . Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. 94 s. Vedoucí diplomové práce Ing. Tomáš Horváth, Ph.D..

Prohlášení

Prohlašuji, že svou diplomovou práci na téma „Systém pro detekci rámce GPON“ jsem vypracoval samostatně pod vedením vedoucího semestrální 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 následku porušení ustanovení § 11 a následujících autorského zákona c. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonu (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“

V Brně dne ....

podpis autora

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

Poděkování

Chtěl bych poděkovat Ing. Tomášovi Horváthovi, Ph. D. a Ing. Václavu Oujezskému, Ph. D. za odborné vedení a připomínky k této diplomové práci. Dále bych rád poděkoval Bc. Veronice Foltýnové a Ing. Michalu Jurčíkovi za spolupráci.

V Brně dne …….. .....

podpis autora

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM PROGRAMOVÉHO KÓDU

ÚVOD

Celý svět kolem každého z nás je zdrojem nesmírného počtu informací, se kterými se dnes a denně setkáváme. S daty je nakládáno různým způsobem podle individuálních potřeb člověka, ale vždy jsou ukládány jako jednotlivé kolekce neboli databáze. Databáze nemohou být považovány jen za výdobytek dnešní doby, ale jsou známé již po celá staletí. I přes jejich dlouhodobou existenci jsou za opravdové předchůdce každé databáze považovány knihovny či lékařské kartotéky.

S rostoucím počtem informací jsou zvětšovány i kapacity úložišť, uchovávající stále větší množství databází, které musí být nějakým způsobem udržovány a doplňovány. S rychlým vývojem IT technologií je kladen důraz na rychlé a bezpečné ukládání dat do elektronické podoby, jejich rychlou, snadnou a bezpečnou dostupnost a přenositelnost odkudkoliv kamkoliv. S tímto vývojem je samozřejmě kladen i zvýšený důraz na bezpečnou distribuci, nežádoucí modifikaci informací z databází, jejich kvalitní uložení a zamezení tak případnému znehodnocení na základě poškození hardwarových technologií.

Při zvyšování nároků na rychlost a dostupnost dat jsou zvyšovány i požadavky na kvalitní a spolehlivé připojení k Internetu nejen ze stran firemních zákazníků či jiných organizací, ale v neposlední řadě i domácností i jednotlivých koncových uživatelů. S tímto trendem došlo v posledních deseti letech k obrovskému vzestupu přenosu informací po optických vláknech nejen v páteřních a distribučních sítích. Jejich působnost se čím dál více rozšiřována skrze pasivní optické sítě až do sítí přístupových. Tyto sítě jsou zakončeny optickými rozvaděči v bytových objektech nebo domácnostech.

S vývojem nových standardů, nejen v optických komunikacích, sice dochází ke zvýšení rychlosti a kvality, ale nastává i úskalí v komunikaci. V současné době neexistují žádné standardy pro komunikaci po optických vláknech, ale vše je definováno pouze doporučeními, které výrobci zařízení nemusí dodržovat. Už od začátku samotného vývoje je však tato architektura zcela odlišná od jiných technologií i referenčního modelu ISO OSI a až doposud neexistuje žádný způsob, jak její komunikaci monitorovat a analyzovat běžně dostupnými analyzátory. Pro analýzu sice mohou být použity komerční analyzátory, ale velkou otázkou je jejich vysoká cena s dalšími licenčními poplatky.

Tato práce je zaměřena na návrh systému pro detekci GPON rámce a analýzu přenášených dat prostřednictvím pasivních optických sítí. První kapitola se zabývá problematikou databázových systému a návrhů databází, která je později využita k návrhu

výsledné databáze. V druhé kapitole je pojednáno o rozvoji pasivních optických sítí a praktického nasazení. Dále je vysvětlen princip přenosu dat po GPON síti směrem ke koncovému uživateli.

Třetí kapitola pojednává o navržení databáze pro detekci a ukládání GPON rámců. V kapitole je dále rozebrána její implementace do databázového systému Microsoft SQL Server 2016 prostřednictvím Microsoft Server Management Studia. V neposlední řadě jsou zde rozebrány i některé možnosti konfigurace databázového systému, jeho správa, správa databází a ukázka spojení s databází prostřednictvím jazyka Python.

V poslední kapitole je popsán princip analýzy a správy dat zachycených sondou FPGA. S daty je v této práci pracováno na dvou úrovních, tj. na straně SLQ serveru prostřednictvím jazyka Transact SQL a na straně analyzačního skriptu napsaného v programovacím jazyce Python.

1 DATABÁZOVÉ SYSTÉMY

Všude kolem nás a zejména na poli výpočetní techniky jsou databázové systémy a báze dat na každém kroku implementovány do většiny služeb, aniž by si to koncový uživatel uvědomil. Typickým příkladem mohou být registry počítače, soubory na disku počítače, kontakty v emailovém adresáři atd.

Databáze je kolekce různě provázaných spolu souvisejících dat uspořádaných do nějakého snadno modifikovatelného, řízeného a přístupného systému. Databáze obvykle obsahuje sloučených několik celků bez redundance dat. Do databáze může přistupovat více uživatelů s jinými oprávněními [33].

Pro komunikaci s databází se používá tzv. dotazovací jazyk. Ten na základě dotazů získá odpověď od databáze. V praxi existují dva typy databázových systémů – univerzální a specializované. Rozdíl je v použití dotazovacího jazyka. V univerzálních databázových systémech je používán jazyk podle struktury, zatímco specializované systémy dotazovací jazyk nemají a jsou vytvářeny na zakázku tzv. „na klíč“ například pro bankovní systémy, mobilní operátory, rezervační systémy aj. [7].

1.1 Historický vývoj

Smysl sbírání dat je dán lidskou povahou a je znám již z období pravěku, odkud se dochovalo mnoho nálezů, jako jsou hliněné destičky nebo malby. Za první systém databáze je považován až systém kartotéky. Jde o kolekci karet, do které člověk zakládá materiály podle určitého uspořádaného systému a v případě potřeby lze daný záznam najít, modifikovat a zpět založit do kartotéky. V oblasti databází šlo o průlomové, ale s rostoucí velikostí značně pomalé řešení. S vývojem prvních počítačů byla data digitalizována a ukládána pomocí děrnoštítkového systému Hermana Hollerithe. S dalším rozvojem počítačů a prvním sériově vyráběným počítačem firmou IBM (International Business Machines Corporation) vznikla povinnost ukládat data do databází a strojový kód byl později nahrazen vyšším programovacím jazykem. Uzavřené datové systémy začaly být neefektivní a problematické. Data bylo potřeba sdílet mezi více systémy. Jako důsledek vznikly databázové modely [8].

1.2 Databázové modely

Databázové modely zobrazují způsob ukládání dat a logickou strukturu mezi položkami databáze. Definují, jak lze k záznamům přistupovat a ukazují jak databázi navrhnout. Většina modelů lze reprezentovat i grafem [30].

1.2.1 Hierarchický model dat

Databázový model založený na struktuře rodič-potomek. Jeden rodič může mít více potomků, ale každý potomek pouze jediného rodiče. Záznamy tak vytváří stromy a nejvyšší záznam označujeme jako kořen (root). Tento model byl vytvořen v 70. letech minulého století firmou IBM a prvně nasazen na sálových počítačích. V dnešní tobě má tento model mnoho nedostatků, které byly vyřešeny jinými modely. Problematický je i samotný přístup k sousedním potomkům – vždy je nutné jít přes rodiče. Dalším problémem je modifikace databáze, kde se zejména jedná o přidávání a rušení záznamů [7][8][30][34].

1.2.2 Síťový model dat

Síťový model vychází z nedostatků předchozího modelu a umožňuje lepší přístup k objektům databáze. Jsou vytvořeny mnohonásobné vazby mezi položkami databáze a tím odstraněna redundance položek v databázi. K objektům lze přistupovat přímo a už není nutné přistupovat přes jednoho nebo více rodičů [8][30][34].

1.2.3 Objektově orientovaný model dat

Databázový model vycházející z objektů a metod. Jednotlivé položky databáze (objekty) jsou vzájemně povázány, přístupny a spravovány pomocí metod. Často jsou používány v objektovém programování. Výhodou modelu je odstranění limitace ukládání údajů do tabulek, jako je tomu u relačního modelu a ukládat multimediální data [30].

1.2.4 Relační model dat

Jedná se dnes o hojně používaný a rozšířený datový model z roku 1970. Vychází ze vztahů tzv. relací, které si lze představit jako propojené tabulky s řádky a sloupci. Řádky tabulky jsou označovány jako záznamy (record) a jednotlivé sloupce jako vlastnosti. Každá buňka (atribut, položka) je definována hodnotou určitého datového typu, která je stejná pro celou vlastnost tj. sloupec tabulky. Při tvorbě databází je nutné dodržet základní tyto pravidla:

- Databáze musí být nezávislá na pořadí záznamů a atributů.

- Hodnoty atributů musí tvořit doménu – všechny atributy musí být stejného datového typu.
- Hodnoty v databázi musí být elementární, tzn. dále nedělitelné.
- Nesní obsahovat redundantní záznamy – každý záznam v databázi musí být jedinečný a označen klíčem, pomocí kterého je tabulka indexována.

Klíč v tabulce musí být alespoň jeden, který označujeme jako primární. Tabulku je možné setřídit i pomocí více klíčů, které nazýváme přídavné, nebo častěji sekundární. Při tvorbě relací je možné použít i primární klíč z jiné tabulky. Takový klíč pak nazýváme cizím klíčem [7][34].

Tabulky lze kromě samotných relací propojit i tzv. relačními tabulkami obsahující klíče uspořádané z více tabulek. Vztahy mezi tabulkami existují trojího typu[8]:

- 1:1 – při provázání dvou tabulek je přiřazen jednomu záznamu z první tabulky maximálně jeden záznam v druhé tabulce.
- 1:N – jeden záznam z první tabulky může být provázán s více záznamy. Například pokud první tabulka obsahuje seznam zákazníku e-shopu a druhá číselný seznam objednávek, pak platí, že jeden zákazník může mít více objednávek, ale každá objednávka patří právě jednomu zákazníkovi.
- N:M – více záznamům z první tabulky odpovídá více záznamů z druhé tabulky.

S relační databází uživatel komunikuje prostřednictvím strukturovaného dotazovacího jazyka (SQL – Structured Query Langurage), jehož syntaxe je velmi podobná slovům přirozené angličtiny. Příkazy jazyka lze rozdělit na 3 typy [7][8][27]:

- příkazy pro manipulaci s daty – SELECT, UPDATE, INSERT, DELETE,…
- příkazy pro řízení dat – GRANT, REVOKE,…
- příkazy pro definici dat – DROP, CREATE, ALTER,…

Na základě doporučení se pro přehlednost dotazů píší velkými písmeny příkazy a ostatní části dotazů, včetně parametrů tabulek, písmeny malé abecedy. V dotazech se může vyskytovat i zástupný znak hvězdička (*), který zastupuje celek.

Práce s tabulkami není jen o vytváření dotazů, ale je nutné definovat základní prostředek pro syntézu dat z tabulek tzv. relační algebru. Relační algebra určuje [28]:

- projekci tabulky – výběr položek (sloupců) – při projekci tabulky je z původní tabulky vytvořená tabulka nová tak, že jsou odstraněny položky A i B. Projekce tabulky umožňuje odstranit případnou duplicitu záznamů.

- selekci tabulky – výběr záznamů – z původní tabulky je vytvořena nová tabulka obsahující pouze takové záznamy, které splňují danou podmínku.
- spojení tabulek

1.3 Architektura databází

Architektura je způsob rozdělení zátěže databázového systému a jeho komponent na dostupných hardwarových kapacitách strojů a počítačové sítě. Kromě lokální architektury, kdy je vše soustředěno na jednom zařízení (např. MS Access), existují další tři typy architektur [2] [3] [4].

1.3.1 Centrální architektura

Jde o zastaralou technologii z období sálových počítačů, kdy zátěž běžela na jednom počítači. Uživatel prostřednictvím terminálové sítě komunikoval s centrálním počítačem a přes síť mu byla dopravovaná pouze data obsahující informace o rozložení údajů na obrazovce. Centrální počítače zpravidla musely obsluhovat více terminálů, a proto odezva systému nebyla okamžitá [2][3][4].

1.3.2 Architektura file-server

S rozvojem a snížením cen osobních počítačů a počítačových sítí, bylo možné rozložit zátěž mezi osobní počítač a centrální úložiště. Databáze je umístěna centrálně pro všechny klienty, ale systém řízení báze dat je instalován pouze na klientských stanicích. Uživatel tak komunikuje prostřednictvím dotazovacího jazyka se systémem řízení báze dat (SŘBD). Přes počítačovou síť jsou tak přenášeny pouze soubory a dotazy na soubory z úložiště. Všechno zpracování tak probíhá s ohledem na výkon klientského počítače. Nevýhodou architektury je nutnost vlastnit síť s dostatečnou přenosovou kapacitou [2][3][4].

1.3.3 Architektura klient-server

Architektura klient-server je obdobou předchozí architektury, která kladla minimální požadavky na samotný server udržující databázi. V této architektuře je největší výkon kladen právě na server, což je výhodné z hlediska sítě a klientů. Klienti nemusí disponovat velkým výpočetním výkonem a počítačovou sítí neprochází obrovské shluky dat. Komunikace začíná na klientské stanici, na které je formulován dotaz pro SŘBD a odeslán na server. Server s ohledem na svůj početní výkon a podle příchozích požadavků dotaz zpracuje a vrátí zpět

pouze klientem požadovaná data. Ve srovnání s předchozím modelem je výrazně snížena požadovaná přenosová kapacita sítě, ale za cenu mnohonásobně vyššího výkonu server. V praxi ale platí, že samotný databázový server může mít zátěž rozdělenou mezi více fyzických strojů, čímž je vytvořena tzv. distribuovaná databáze [2][3][4].

1.4 Systémy řízení báze dat

Databázový systém, neboli SŘBD, je softwarové vybavení počítače, specializující se na efektivní operace s bází dat. Výkonnost takového systému je závislá nejen na hardwarovém vybavení počítače či serveru, ale i na množství údajů v databázi a jejich vhodném uložení. Proto je nutné dbát i na správný návrh samotné databáze [6][8][9].

1.4.1 MySQL

Švédský multiplatformní databázový systém, založený na relačním modelu a architektuře klient-server. Je šířen pod bezplatnou licencí společností Sun Microsystems a považován za jeden z aktuálně nejrozšířenějších databázových systémů. Základní správa vychází z koncepce Linuxu, tj. lze relativně snadno spravovat skrze dotazovací jazyk z příkazové řádky a není potřeba žádných grafických nástrojů. Grafické komponenty jako je např. MySQL Workbench nebo phpMyAdmin je nutné doinstalovat zvlášť. Často je nasazován na servery v kombinaci zvané LAMP (Linux Apache MySQL PHP) a tvoří tak základ webových serverů. Vývoj projektu směřuje k maximálnímu možnému výkonu SŘBD, proto jsou některé funkce značně zjednodušené [22][23][32].

1.4.2 Microsoft SQL

Profesionální databázový software určený pro komerční sféru. Jedná se o placený produkt, ale obsahuje i velmi omezenou volnou edici. Není multiplatformní jako v případě MySQL. Je primárně určený pro servery Windows, ale je možné jej nasadit v dockeru (kontejnerová virtualizace) i v prostředí Linuxu konkrétně v distribucích Ubuntu 16.04, Red Hat Enterprise Linux 7 a SUSE Linux Enterprise Server 12. Microsoft SQL obsahuje několik edicí, které limitují maximální využitelné hardwarové parametry. Microsoft SQL resp. Edici Azure, je možné nasadit v cloudu a sdílet ji jako platformu (Platform As Service – PAAS). Kromě relační databáze nabízí i další analytické nástroje. Tento SŘBD je vhodný pro analýzy dat, datové sklady atd. S databází lze komunikovat pomocí SQL jazyka. Ačkoliv MS SQL

i MySQL jsou založeny na SQL jazyku, nemusí platit, že dotaz fungující v jedné databázi, musí fungovat i v druhé [19][20].

1.4.3 InterSystems Caché

Multiplatformní hierarchický systém řízení báze dat pracující s Caché objekty. Jeho vnitřní stromová struktura je výhodná např. pro systémy chorob v lékařství. Caché server obsahuje svůj vlastní jazyk, který neslouží pouze pro správu dat, ale umožňuje napsat i celé aplikace. Kromě tohoto jazyka je i definována podpora pro jazyk SQL. Práce s databázovými objekty je stejná jako v objektově orientovaném programování (OOP), a proto návrhy je možné vytvářet pomocí jazyka UML (Unified Modeling Language) [12][13].

1.4.4 Oracle DB

Jedná se o multiplatformní a nejdůvěryhodnější relační databázi pro komerční sféru, konkurující MS SQL. Její edice se liší opět hardwarovými omezeními a cenou. Správu databáze lze zajišťovat jazykem SQL. Databáze má oddělenou logickou a fyzickou architekturu, která jí činí velmi robustní, a proto ji je možné nasadit i na výpočetní gridy. Další výhodou Oracle DB je možnost používat ji globálně napříč rozsáhlými sítěmi. Komunikaci mezi různými technologiemi sítí zajišťuje integrovaná aplikace [5][25].

1.4.5 MariaDB

MariaDB je relační databáze, která vznikla jako další vývojová větev MySQL. První verze obsahovaly stejné doplňky a funkce jako původní MySQL. V pozdějších verzích vývojáři upustili od stejného číslování a vydali se rozdílným směrem. Oba systémy řízení báze dat mají stejné API, na kterém mohou vývojáři stavět své aplikace, a proto je relativně snadné migrovat data mezi těmito databázemi [1][29].

1.4.6 Mongo

Mongo je noSQL databáze, podobná relačním, ale data jsou ukládána do tabulek ve formátu JSON. Původně byla vyvíjena pro poskytování platformy jako služby (PAAS), umožňující nasazení v cloudu. Na software MongoDB se vztahuje licence GNU Affero General Public License a ovladače Apache Foundation [14][31].

2 PASIVNÍ OPTICKÁ SÍŤ

S rozvojem počítačů a Internetu došlo rovněž k průlomu v oblasti přenosových přístupových sítí. Optické sítě jsou rozděleny na pasivní a aktivní (AON, Active Optical Network), které vyžadují po cestě aktivní prvky pro zpracování signálu (např. zesilovače). Tyto sítě jsou základem transportních sítí[24].

Pasivní optické sítě (PON – Passive Optical Network) jsou nasazovány v přístupových sítích tj. mezi koncovým účastníkem a transportní sítí. Původní metalická vedení jsou pozvolna nahrazována optickými nebo kombinovanými sítěmi, lišícími se způsobem ukončení optického rozvodu. Jsou označovány:

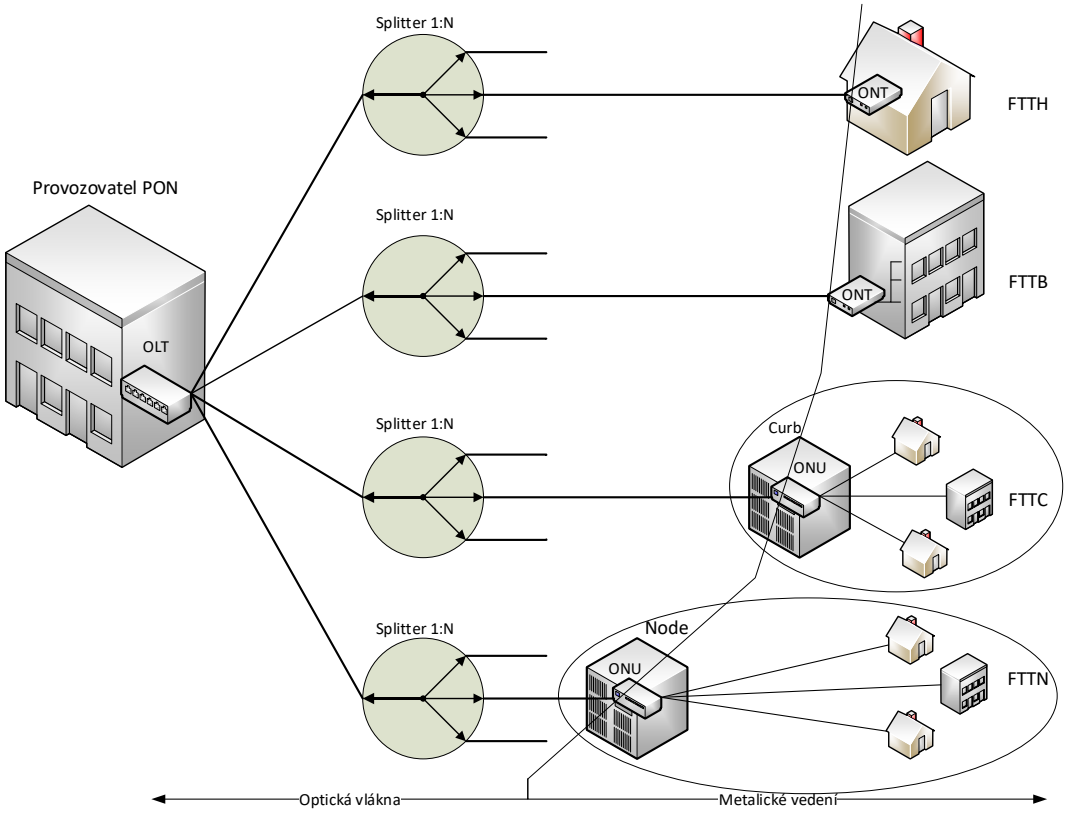
- FTTN (Fiber To The Node) – Vlákno do uzlu, rozvod je ukončen v optickém rozvaděči v okruhu několika kilometrů od koncového účastníka.
- FTTC (Fiber To The Curb) – Rozvod k „obrubníku“ optický rozvod je zakončen v okolí několika stovek metrů od zákazníka. Další rozvody jsou řešeny jinými technologiemi např. ADSL, VDSL,…
- FTTB (Fiber To The Building) – Vlákno do budovy, rozvod je ukončen v rozvaděči umístěným v budově např. bytové domy. Přípojka k zákazníkům je vedena jinou formou přenosu (WiFi, PLC, UTP rozvody,…)
- FTTH (Fiber To The Home) – Optická síť je ukončena na hranici bytové jednotky koncového zákazníka např. byt. Tímto zakončením je docíleno úplného optického spoje.
- FTTA (Fiber To The Antenna) – Jejich uplatnění nalezneme u sítí mobilních operátorů, nebo jiných vysokorychlostních bezdrátových systémů. Původní antény 3G a 4G sítí, včetně všech předchozích, měly rozvody mezi základnovou stanicí umístěnou na zemi a jednotkou obsluhující bezdrátový přenos vedeny koaxiálním kabelem. S rostoucími nároky na vysoké přenosové rychlosti a experimenty s 5G sítěmi je koaxiální kabel mezi jednotkami nahrazen optickým vláknem, které vede až do antény [10].

Od optických zakončení sítě jsou další rozvody opět metalické. Důvodem nahrazování metalických vedení optickými je zabránit zvyšování přenosových ztrát a tím i snižování přenosové rychlosti na delší vzdálenosti mezi účastníky [15][16][24].

2.1 Topologie sítě

Pasivní optická síť je spojení typu Point-To-Multipoint. Hlavní myšlenkou pasivní optické sítě je přenášet signál bez dalšího upravovaní (zesilovaní), a proto je síť složena ze základních prvků:

- ONT (Optical Network Terminate) – Optické síťové zakončení, ukončuje optickou trasu u uživatele a zajišťuje adaptaci protokolů mezi uživatelskou a optickou sítí.
- ONU (Optical Network Unit) – Optická síťová jednotka. Plní stejnou funkci jako jednotka ONT, ale nachází se v optických rozvodnicích odkud je signál šířen jinými technologiemi.
- OLT (Optical Line Terminate) – Optické linkové zakončení se nachází na straně poskytovatele (provozovatele) PON. OLT zajišťuje kromě konverze protokolů mezi páteřní a pasivní optickou síti také dohled a správu nad jednotkami ONU a ONT tj. určuje čas pro vysílání jednotlivých koncových jednotek.
- Splitter – Pasivní prvek umožňující větvení pasivní sítě. Přenášený signál je rozbočován v poměru 1:N, kde N může být 2, 4, 8, … 64, 128. Cena těchto prvků závisí na počtu portů, na které je signál dále větven. Protože ani splittery nejsou aktivní zařízení, vkládají do cesty signálu útlum a tím zhoršují přenos. S rostoucím dělícím poměrem roste celkový útlum vedení [15][16][24].



Obr. 1: Topologie PON

Výhodou pasivních sítí jsou provozní a pořizovací náklady. Protože síť je složena z pasivních prvků, které nepotřebují napájení, lze náklady minimalizovat a rovnoměrně rozdělit mezi jednotlivé účastníky.

Každý prvek v síti do cesty vkládá útlum. Tím dochází k omezení i maximální vzdálenosti, kterou lze optickou sítí překlenout. Tyto parametry jsou upřesněny postupně vyvíjenými standardy APON (ATM Based PON), BPON (Broadband PON), GPON (Gigabit-capable PON) atd. [11] [15][16][24]

2.2 Standard GPON

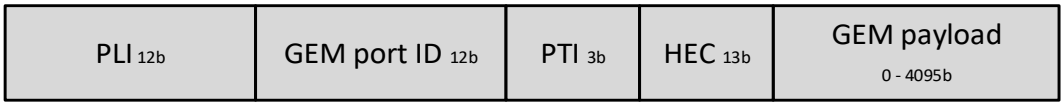
Standard G984.1 schválený organizací mezinárodní telekomunikační unií ITU (International Telecomunication Unit). Použití tohoto standartu, garantuje dostatečnou kapacitu pro dnes běžné používané služby a zároveň poskytuje i dostatečnou rezervu pro služby budoucí.

Standard GPON vychází z dříve vydaných standardů APON a BPON, ale využívá mírně upravený vrstvový model, čímž není docíleno zpětné kompatibility. Rámcování vychází z časového dělení jako u ATM. Přenášeny jsou jednotlivé GPON rámce, které nejsou fixně definovány bitovou velikostí, ale časovým intervalem 125 µs. Podle přenosové rychlosti

sítě je definováno kolik bitů obsahuje jeden GPON rámec. Ve standardu GPON jsou definovány dvě přenosové rychlosti 1,244 a 2,488 Gbit/s [11][24].

Pro komunikační sítě je důležitá obousměrná (full-duplexní) komunikace. V případě pasivních optických sítí je pro každý směr nezbytné použít záření o jiných vlnových délkách. S tím by bylo nutné vybudovat dvojitou síť – jedno vlákno pro odchozí směr druhé pro příchozí. Aby nedocházelo k duplikování sítě, a tím i růstu ceny na realizaci a údržbu, využívá se vlnového multiplexování (WDD – Wavelength Division Multiplex) umožňujícího přenos pouze po jednom společném optickém vlákně [24].

Další rozdíl je ve způsobu přenosu informací v sestupném a vzestupném směru. Přenos uživatelských dat v PON je možný pouze po zapouzdření do GEM rámců (GPON Encapsulation Mode).



Obr. 2: Struktura GEM rámce

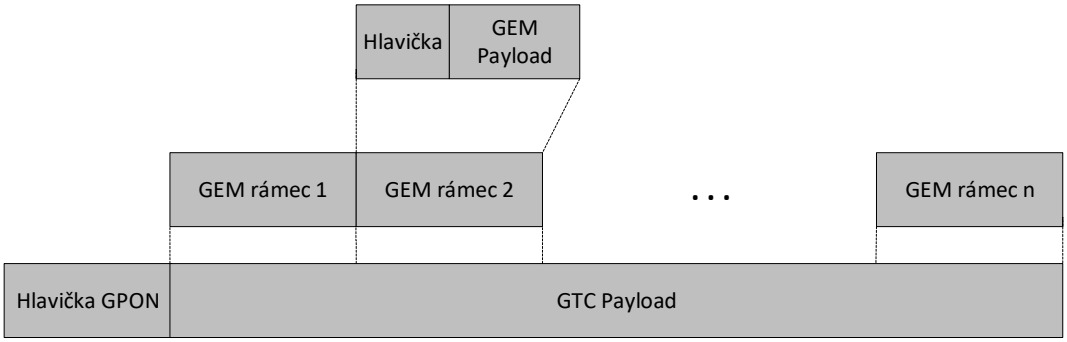
Na obr. 2 je zobrazena struktura GEM rámce. Velikost GEM rámců není pevně stanovena, protože každý GEM rámec se skládá z hlavičky rámce o fixní velikosti 5 bajtů a přenášených dat o velikosti mezi 0 – 4095 bajty. Pokud jsou uživatelská data větší velikosti, než je maximální definovaná velikost GEM rámce, jsou data fragmentována do více rámců. Každý rámec je složen z bloků [11]:

- PLI (Payload Lenght Indicator) – Indikátor délky pole uživatelských dat je blok o velikosti 12 bitů. Hodnota pole definuje proměnou délku velikosti přenášených uživatelských dat.
- GEM port ID – Blok z 12 bitů, označující identifikátor provozu a pomocných informací pro multiplexování.
- PTI (Payload Typ Indicator) – Blok 3 bitů nesoucí informaci o typu uživatelských dat a kompletnosti GEM rámce (jestli uživatelská data nejsou fragmentována). Pokud není rámec kompletní, určuje pořadí rámce (první nebo poslední).
- HEC – Ochranné kódování hlavičky (13 bitů). Kvůli variabilní proměnné délce GEM rámců musí být úspěšně dekódována hlavička. Bez správně dekódovaného rámce nelze zjistit začátek dalšího rámce.
- GEM payload – uživatelská data s proměnnou délkou.

2.2.1 Provoz na GPON v sestupném směru

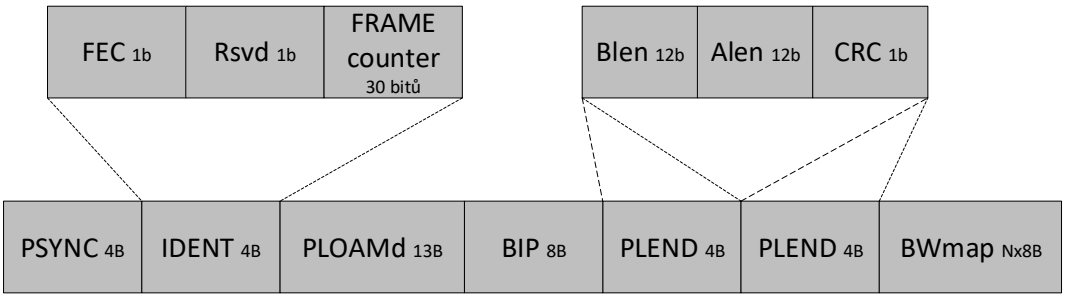
Ze schématu FTTx sítí je patrné, že do cesty jsou vkládány optické splittery, které síť rozvětvují. Komunikace probíhá od řídící jednotky OLT směrem k uživateli tj. jednotkám ONU/ONT. Z podstaty splitterů, které stojí v cestě ke koncovým jednotkám, dochází k duplikování GPON rámců. Každá koncová jednotka tak obdrží i rámce, které pro ni nejsou určeny. Takto snadno získané rámce by mohly vést k snadným a především nežádoucím odposlechům komunikace. Proto bývá komunikace šifrována blokovou šifrou AES (Advanced Encryption Standard).

Sestupný směr je důležitý i pro směr vzestupný, protože podle hodnot z GPON záhlaví je nastavována doba, kdy mohou jednotlivé jednotky odesílat svá data. Jak bylo dříve uvedeno, aby bylo možné přenášet uživatelská data, musí být zapouzdřena. Zapouzdření dat se prování na několika úrovních vycházejících z GPON modelu [11][24].



Obr. 3: Zapouzdřování v GPON

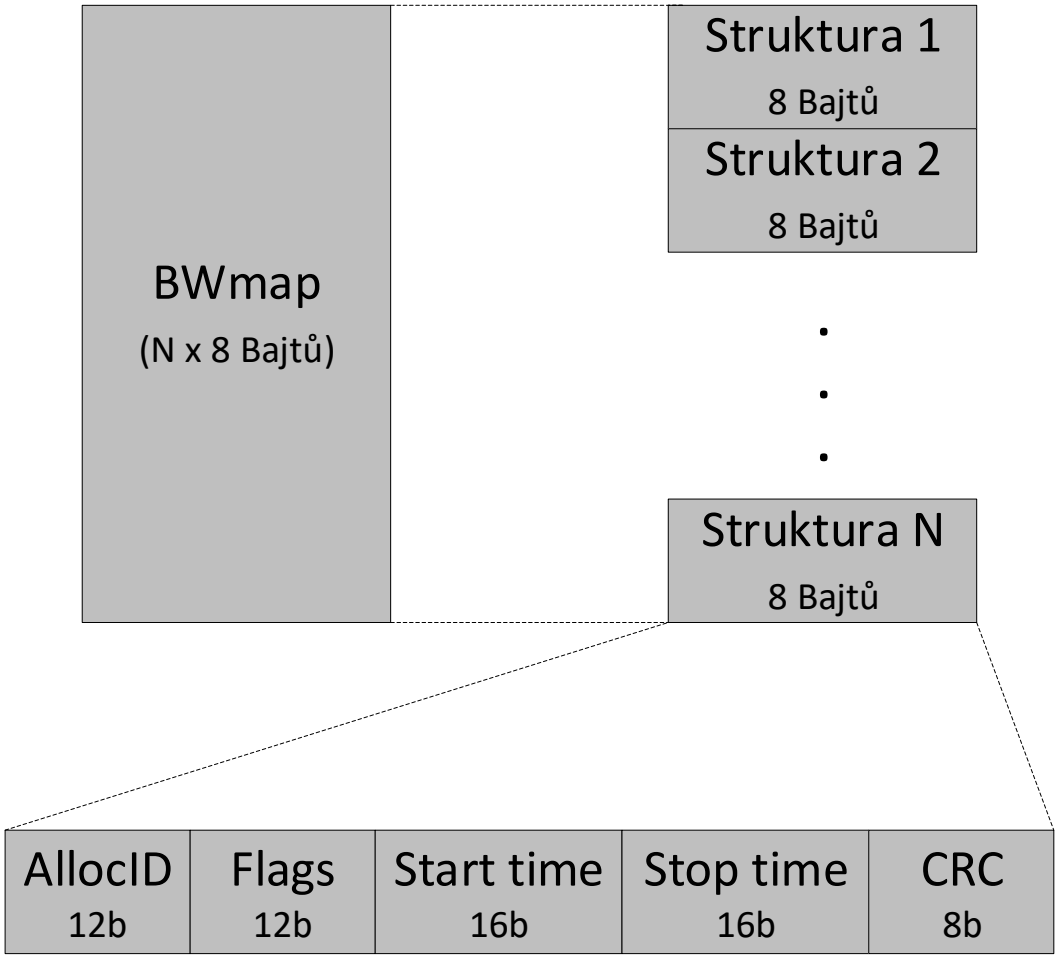
Standard GPON využívá při přenosu časového multiplexu (TDM – Time Division Multiplex). GEM rámce, obsahující uživatelská data, jsou zapouzdřovány do GPON rámců. Hlavička každého GPON rámce se skládá z polí jak je uvedeno na následujícím obrázku 4 [11]:



Obr. 4: Struktura hlavičky GPON rámce

- PSYNC – daná sekvence 32 bitů. Vyskytuje se na začátku každého rámce a zajišťuje synchronizaci fyzické vrstvy.
- IDENT – 32 bitů, určuje vlastnosti a nastavení hlavičky
    - FEC – Definuje nastavení proti chybovému kódování. Pokud je bit nastaven na hodnotu 1, je ochrana použita.
    - Rsvd (Reserved) – Volný a nepoužívaný jeden bit.
    - Frame counter – Počítadlo rámců. Každý rámec má svoje číslo, které je s každým dalším inkrementováno o jedničku. Po vyčerpání všech 30 bitů, tj. přetečení čísla 2^{30} začíná číslování od nuly.
- PLOAM (Physical Layer Operation and Maintenance) – 13 bajtová zpráva. Informace obsažené v PLOAM buňce obsahují řídící a pomocné zprávy.
- BIP (Bit Interleaved Parity) – Buňka obsahuje 8 bajtů prokládaného kódování. Tyto bity ověřují všechny bitové chyby vyskytující se v hlavičce GPON rámce od předchozího pole BIP s výjimkou paritních bitů.
- PLEND (Payload Length fielD) – Buňka o velikosti 4 bajty. Obsahuje informace o poli BWmap, které je důležité pro správný odchozí provoz sítě. Pro zajištění přenosu zprávy v nepoškozeném tvaru, je pole odesíláno dvakrát za sebou a zabezpečena kontrolním součtem.
    - BLen – Pole o velikosti 12 bitů definuje velikost pole BWmap.
    - Alen – Pozůstatek z přenosu ATM buněk. Dnes se nepoužívá a přenáší se nulové bity.
    - CRC – kontrolní součet.
- BWmap – Pole nabývá velikosti podle počtu koncových klientských stanic. S rostoucím počtem stanic je v poli více 8 bajtových struktur. Každá ONU/ONT jednotka má zde definovány parametry pro upstream.
    - AllocID – 12 bitové pole, definující, pro kterou jednotku ONU nebo ONT je definovaná daná 8 bitová struktura.
    - Flags – Pole flags definuje parametry odchozího provozu. Příznakové pole je složeno z 12 bitů, ale použito je zatím pouze 5 bitů.
    - Start time a Stop time – v buňkách je vyhrazeno 16 bitů, které definují dobu od začátku GPON rámce pro zahájení a ukončené přenosu koncovou jednotkou.

- CRC (Cyclic Redundancy Check) – Pole CRC obsahuje 8 bitů pro zabezpečení bezpečného přenosu. Poškozením pole BWmap může být zničena veškerá síťová komunikace.



Obr. 5: Schéma pole BWmap

2.2.2 Aktivační proces

Jak bylo uvedeno dříve, každá pasivní optická síť je složena z jednotek koncových (ONU/ONT) a řídích (OLT). Po fyzickém připojení do sítě je nutnou částí navázat komunikaci s aktivní sítí resp. provést její synchronizaci, přidělení identifikačních údajů a provést její aktivaci v síti. Tato komunikace, nazývající se aktivační proces, je prvním a zároveň nejdůležitějším stavem pro správné fungování koncové jednotky v síti. Během svého působení v síti jednotka prochází, resp. může procházet několika operačními stavy [11]:

- O1: Initial state
- O2: Standby
- O3: Serial number

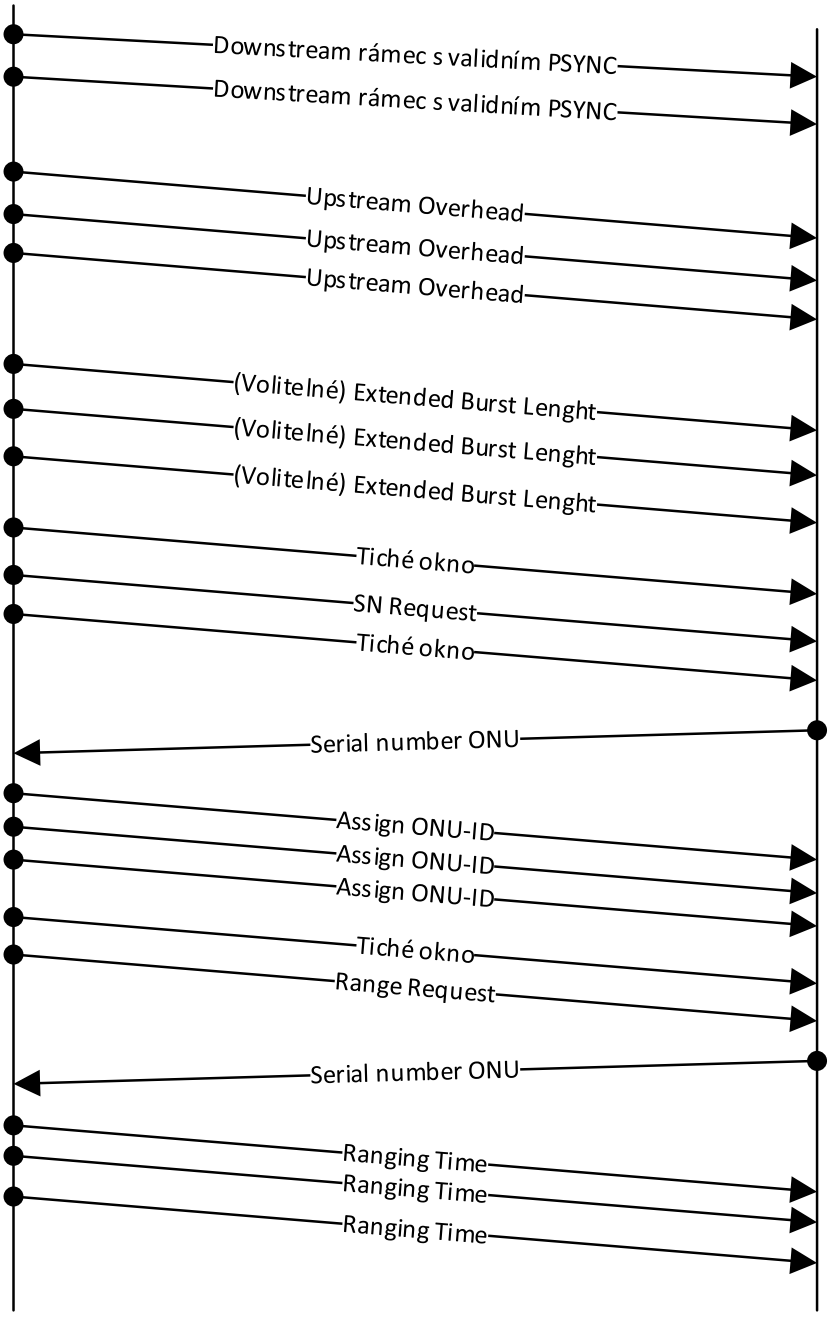
- O4: Ranging
- O5: Operation
- O6: PopUp state
- O7: Emergency state

Aktivační proces je definován prvními čtyřmi definovanými operacemi, ale lze také rozdělit na fáze:

- Parameter learning – získání provozních parametrů
- Serial number acquisition – přiřazení ONU-ID koncové jednotce
- Ranging

První fáze může být označena jako pasivní a je nutná pro další komunikaci. Po zapnutí musí koncová jednotka provést synchronizaci svého přijímače. Synchronizace je prováděna sledováním síťové komunikace a detekcí začátků GPON rámců (tzv. HUNT STATE). Úspěšná synchronizace je považována při detekci M po sobě jdoucích polí PSYNC, kde M je dáno výrobcem (avšak minimálně M je rovno třemi). Po prvním úspěšně detekovaným začátku rámce jednotka inkrementuje čítač a přesune se do předsynchronizačního stavu (PRE-SYNC), ve kterém setrvá do detekování dalšího rámce. Při správné detekci M–1 po sobě následujících rámců je jednotka synchronizována a přejde do stavu Sync. Při nesprávné detekci jednoho z rámců jednotka opět přechází do stavu hunt state a musí být znovu synchronizována. Do režimu hunt state přejde i při ztrátě synchronizace v pozdějších stavech. V této situaci jsou vymazány i dříve vyjednané parametry (Alloc-ID, ONU-ID atd.).

Po synchronizování je jednotka schopna detekovat začátky záhlaví a čeká na zahájení komunikace a vyjednávání potřebných parametrů, fáze O2, O3 a část O4 resp. Serial number acquisition. V této fázi dochází k výměně posloupnosti PLOAM zpráv viz obr. 6.



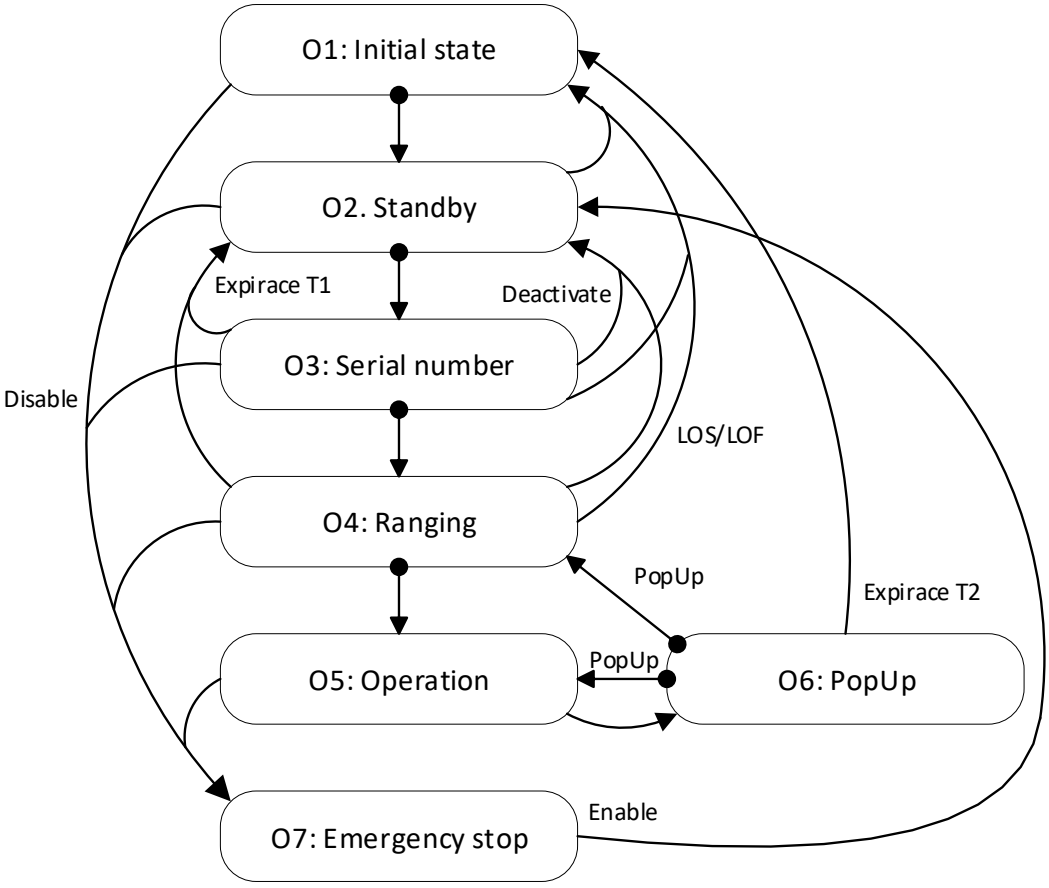
Obr. 6: Aktivační proces [11]

Aktivace je zahájena všesměrovou PLOAM zprávou Upstream_Overhead definující globální parametry. Pro zamezení možnosti výskytu chyb v přenosovém kanálu je odeslána vícekrát. Upstream_Overhed je následována volitelnou, vícekrát odeslanou všesměrovou zprávou Extended_Burst_Lenght definující preambule v následujících stavech.

Po nastavení a upřesnění globálních parametrů je OLT všesměrově zašle žádost o odeslání sériového čísla jednotky ONU zprávou SN_Request. Aby nemohlo během získávání sériového čísla dojít ke kolizi s normálním provozem je vytvořeno před a po žádosti tzv. Quiet Window (tiché okno), Quiet Window je reprezentováno GPON rámcem

s prázdným polem BWmaptrvajícím 250 µs. Na tuto žádost jednotka odpoví PLOAM zprávou Serial_Number_ONU, obsahující sériové číslo jednotky. Na základě sériového čísla je OLT schopno přiřadit konkrétní jednotce konkrétní ONU-ID všesměrovou zprávou Assign_ONU_ID. Po přidělení vlastního ONU-ID, jsou OLT a ONU schopny mezi sebou komunikovat unicast zprávami a ONU se přesouvá do stavu O4 tj. ranging.

Během této fáze je jednotka synchronizována pro upstream přenos. Hlavním problémem je různá vzdálenost mezi ONU resp. jejich čas pro přenos. Fáze je zahájena přenosem Quiet Window s trváním 202 µs a následována přenosem Range_Request, na které ONU odpoví opětovným odesláním PLOAM zprávy Serial_Number_ONU. Na základě těchto zpráv je OLT schopno vypočítat vzdálenost a zpoždění, se kterým má ONU odesílat GPON rámce. Tohle zpoždění označené jako equalization-delay je koncové jednotce ONU oznámeno PLOAM zprávou Ranging_Time. Po nastavení zpoždění je aktivační proces dokončen, jednotka přechází do operačního stavu (O5) a může začít komunikovat a přenášet další provoz.



Obr. 7: Stavy jednotky ONU [15]

2.2.3 Deaktivace ONU

K deaktivaci ONU jednotky může dojít z mnoha důvodů. Každá jednotka obsahuje alarm upozorňující o ztrátě optického signálu (LOS/LOF), jehož spuštění má za následek okamžité přesušení upstream komunikace až do jeho vyřešení. Pokud je alarm spuštěn během aktivačního procesu, dojde i ke ztrátě synchronizace a jednotka automaticky přechází do stavu O1 viz obr 7. Do stavu O2 jednotka přejde během aktivace při vypršení časovače T1, který stanovuje maximální dobu na přesun do dalšího stavu.

Stav O6: PopUp je stav, do kterého jednotka přejde při spuštění alarmu za běžného provozu (stav O5). Vyvolaný alarm spouští časovač T2, během kterého má jednotka možnost provést obnovení optického signálu synchronizování s GTC rámcem a opětovného návratu do stavu O5. Pokud dojde k jeho vypršení a jednotka neobdrží ani PopUp zprávu od OLT, ONU automaticky přechází do stavu O1. Během svého provozu může jednotka ONU obdržet cílenou nebo všesměrovou PopUp zprávu, která ji nařídí přejít do jednoho z předchozích stavů. Pokud jsou detekovány OLT jednotkou kolize GTC rámce v upstream přenosu. Odesílá všesměrovou PopUp zprávu, vracející jednotky ONU do ranging (stav O4). Po nastavení equalisation delay se jednotka ONU může opět vrátit do běženého režimu (O5). Při získání cílené zprávy PopUp jednotka ONU přechází do Stanby stavu (O2) a musí být zahájen aktivační přenos.

Stav O7: Jednotka ONU může být násilně deaktivována (např. při poruše) do nouzového reřimu (Emergency Stop) jednotkou OLT. Toho je docíleno po přijetí zprávy Deactivate_ONU_ID s parametrem disable. V nouzovém režimu jednotka ONU nesmí odesílat žádná data, ani jinak komunikovat po optické síti. Toho je docíleno nuceným vypnutím laseru. V tomto stavu jednotka musí setrvat až do odstranění příčiny poruchy. Po jejím odstranění je opět aktivována zprávou Disable_ONU_ID s parametrem allow a přechází do stavu Standby (O2) [15].

Zvláštním způsobem deaktivace ONU jednotky je odesláním PLOAM zprávy Dying Gasp. Zprávou je OLT informována o ukončení komunikace po síti a nenásledujícího pokusu na obnovení synchronizace a navázání opětovného spojení. Příčinou odeslání zprávy Dying Gasp může být násilné odpojení od zdroje napájení. Po opětovném připojení je zahájen aktivační proces.

3 KONFIGURACE SYSTÉMU MS SQL

Pro realizaci praktické části bylo použito systému MS SQL. Monitorování provozu a analýza GPON sítě je náročná činnost, která vyžaduje stabilní a dostatečně výkonné řešení. Databázový systém Microsoft SQL je pokročilý software, umožňující optimální rozložení zátěže na daném hardwaru a využíti pokročilých nástrojů pro analýzu.

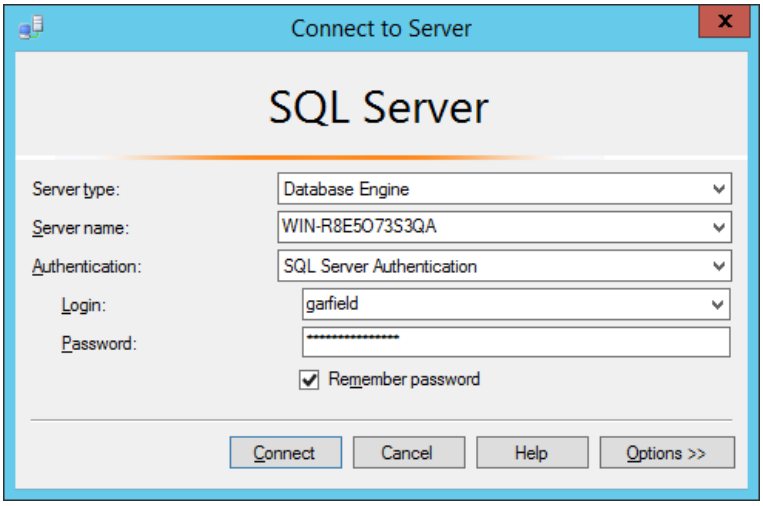
3.1 Server management studio

Microsoft SQL Server Management Studio (SSMS) je integrované prostředí pro správu a řízení databáze, nabízející nejen nástroje a grafické rozhraní, ale i editor pro tvorbu SQL dotazů či skriptů.

3.1.1 Autentizace uživatele

Po spuštění prostřední SSMS je uživatel vyzván k výběru serveru a ke své autorizaci, kterou je možná provést několika způsoby. Úroveň autentizace má dva módy – SQL server autentizace a Windows Autentizace nebo jen Windows Autentizace. Ve výchozím nastavení je zapnuta pouze autentizace s nastaveními Windows. Uživatel má k dispozici několik možností, jak se do systému přihlásit. První tři jsou kombinace s nastavením pro Active Directory (AD) server a poslední možností je přihlásit se s údaji uživatelského účtu ve Windows. Ve druhém módu je kromě zmíněných možností umožněno přihlásit se s uživatelským účtem pro SQL server. Výhodou použití SQL serveru je vytvořit uživatele umožňujícího se přihlásit pouze do databáze a upravovat jí podle nastavených práv. Uživatel nemá možnost se s těmito údaji přihlásit do sítě Windows, nebo k počítači.

Další položkou při přihlašování do systému MS SQL je nutné vybrat, k jaké službě se chce uživatel připojit. K nastavování a úpravě databáze (případně databází) je nezbytné se připojit k databázovému enginu. Další možností je vybrat si analyzační služby (Analysis services), integrační služby (Integration services), Reporting services nebo příhlášení ke cloudovému uložišti Azure.



Obr. 8: Přihlašovací okno

Po úspěšném přihlášení do systému je možné provádět, podle nastavených práv, různé databázové operace – import, export, vytváření, editaci a mazání databází, upravovat schémata, vytvářet uživatele, přidělovat jim oprávnění atp.

3.1.2 Fyzické uložení souborů

Každý databázový systém musí ukládat data na disky. Databáze vytvořená systémem MS SQL může být tvořena ze třech typů souborů – databázového primárního souboru (.mdf), databázového sekundárního souboru (.ndf) a transakčního logu (.ldf). Každá fungující databáze je tvořena minimálně primárním souborem a transakčním logem.

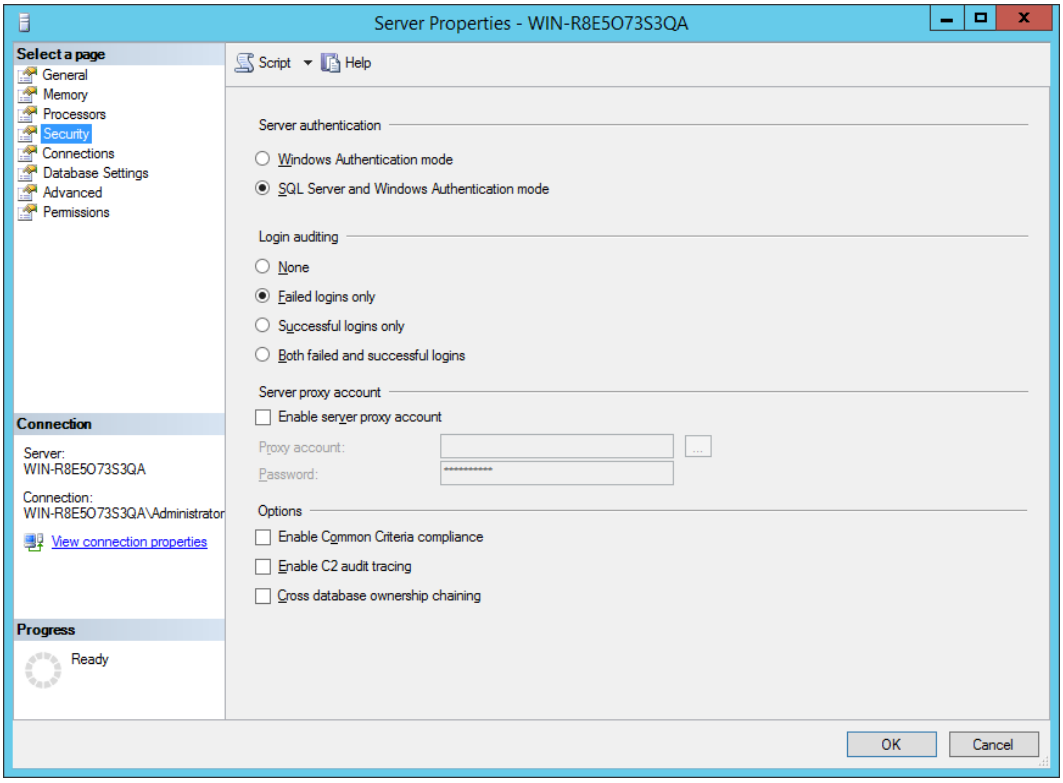
Data uložená v databázi jsou obsažena v primárním datovém souboru, jehož velikost může být systémem MS SQL limitována. Naopak transakční log bývá výrazně menší a obsahuje informace o všech operacích s databází. Při tvorbě databáze na serveru je výhodné ukládat transakční log na jiné diskové pole, protože pomocí něj lze obnovit data z havarované databáze. Tyto soubory obsahuje každá databáze nejvýše jednou, ale může obsahovat více sekundárních souborů. Do sekundárních souborů mohou být uživatelem přesměrovány často přístupné tabulky z primárního souboru. Rozmístěním těchto souborů lze rovnoměrněji rozdělit zatížení serveru mezi více disků, polí, případně strojů.

3.1.3 Základní nastavení MS SQL

Cílem práce je zachytávat data z FPGA sondy umístěné na jiném serveru. Zachycená data ze sondy jsou zpracovávána skriptem a následně ukládána do databáze. Aby bylo možné data do

databáze uložit, je nutné povolit v MS SQL management studiu ověřování uživatelů SQL serverem a vytvořit uživatele, který má do databáze práva zápisu.

Základní nastavení SQL serveru lze provést v SSMS otevřením nabídky *properties* enginu databáze. Pro povolení ověřování SQL serverem je nutné přejít do záložky *security* a zatrhnout v bloku *Server Authentitication* volbu *SQL server and Windows authentication mode*. Aby bylo možné tato pravidla po uložení změn aplikovat, je nezbytné restartovat databázi spravujícího agenta opět přes kliknutí na engine databáze a zvolit restart. SQL agent je služba, většinou spouštěná se startem serveru a umožňuje vykovávat naplánované úlohy např. replikaci či údržbu. Jejím restartováním jsou dočasně všechny databáze nedostupné a při novém startu agenta jsou načtena změněná nastavení.

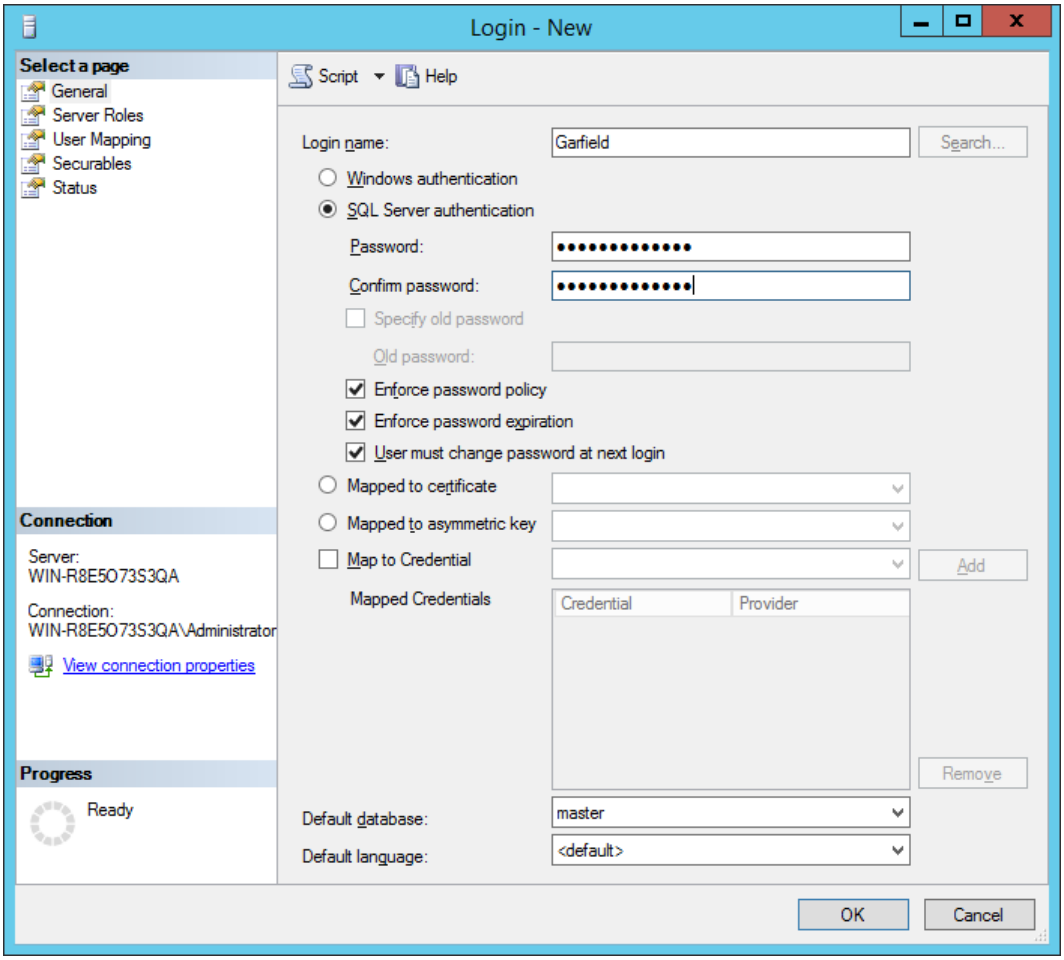


Obr. 9: Vastavení ověřování uživatelů

V záložce *properties* lze zjistit základní nastavení serveru resp. jazyková nastavení, jméno serveru, jakou velikostí paměti RAM systém disponuje atp. V dalších záložkách okna lze nastavit počet aktivních spojení na SQL server, výchozí nastavení kam ukládat nové databáze, nebo nastavit oprávnění k databázovému enginu pro jednotlivé uživatele či jejich skupiny.

Po rozbalení balíku databázového enginu je možné zasahovat do struktury SŘBD. Jednotlivé balíky upravují nastavení pro jiné oblasti – replikaci dat *(Replication)*, zabezpečení databázového enginu *(Security)* nebo správu jednotlivých databází *(Databases)*.

Po povolení SQL autentizace je možné vytvářet nové uživatele nebo skupiny v záložce *security*. Další struktura opět odpovídá intuitivně svému obsahu. Vytváření nových uživatelů je přístupné pod kontextovou nabídkou složky *Logins* a volbou *New Login*. V nově otevřeném okně se záložkou *General* musí být doplněno uživatelské jméno, nastaven způsob autentizace uživatele, výchozí jazyk a výchozí databáze. Pokud neexistuje žádná vytvořená databáze, nastavuje se šablona pro databáze – *master*. Volba šablony umožní uživatelům se připojit ke všem uživatelským databázím, které jsou specifikovány v záložce *User Mapping*. V záložce *status* je nezbytné nastavit povolení k připojení a přihlášení na engine databáze pro nového uživatele. Po jeho vytvoření musí být zkontrolována přidělená práva rozkliknutím databázového enginu a záložky oprávnění *(Permissions)*.

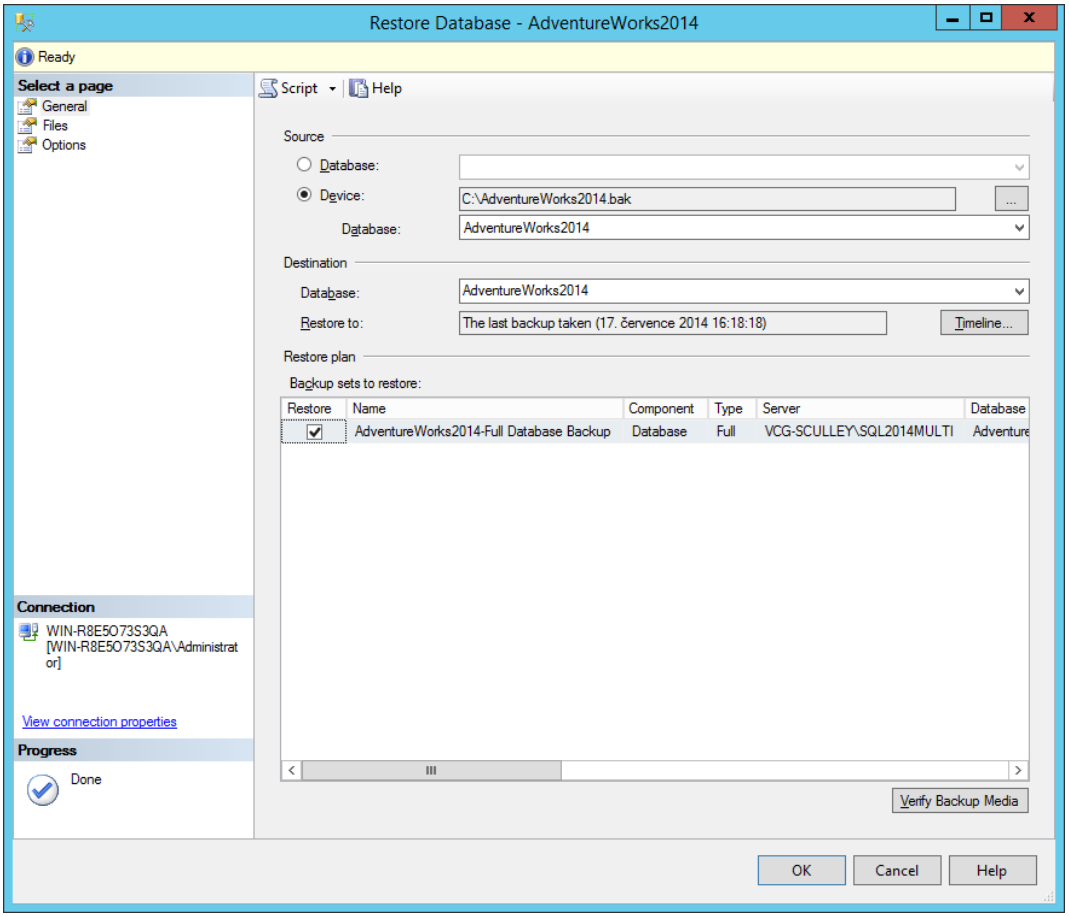


Obr. 10: Vytvoření uživatele v MS SQL

Mezi uživatelskými účty je i účet systémového administrátora, označen jako *sa*. Pro zvýšení bezpečnosti a stability systému je doporučeno tento účet nepoužívat a zakázat. Povolení účtu je při útocích z internetu výhoda pro útočníka, kterému už zbývá jen prolomit heslo.

3.1.4 Import testovací databáze

Základem práce s MS SQL systém bylo importovat libovolnou databázi a seznámit se s prostřením management studia. Nejvhodnější způsob, jak se seznámit s takovým systém, je experimentovat na nějaké obsáhlejší vzorové databázi volně stažitelné z internetu. Po stažení databáze AdventureWorks2014 a jejím rozbalením z komprimovaného souboru na disku serveru, musí být databáze uživatelem obnovena ze záložního souboru s příponou .bak. Obnovení databáze se provádí otevřením nabídky nad polem *database* v databázovém enginu a spuštěním volby *Restore Database*. V nově otevřeném okně musí být vybrán jako zdroj databáze zařízení a specifikována cesta k databázi. Po potvrzení tlačítkem ok je o úspěšně dokončeném importu uživatel informován dialogovým oknem.



Obr. 11: Obnovení databáze

3.1.5 Vytvoření databáze FLOW

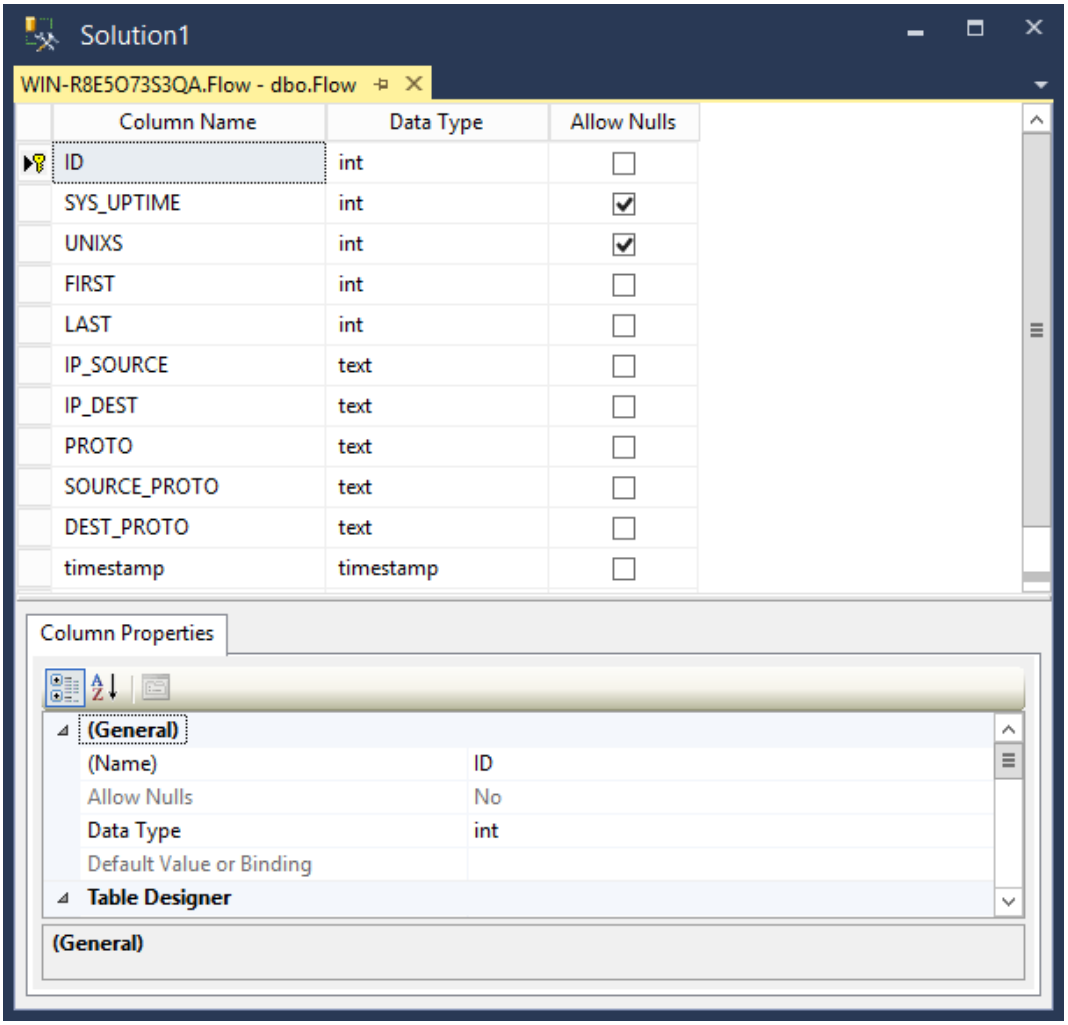
Po instalaci SQL serveru je několik databází v SŘBD již obsaženo. Jedná se o tzv. snímky (*snapshots*), ve kterých jsou uloženy předchozí stavy databází např. body obnovení

a systémové databáze s předdefinovanými počátečními nastaveními. Jsou to systémové databáze:

- Master – obsahující informace o konkrétním SQL serveru
- Model – sloužící jako šablona pro uživatelsky vytvořené databáze.
- Msdb – obsahující informace pro SQL server agenta.
- Tempdb – sloužící jako dočasné úložiště.

Uživatelské databáze je možné vytvářet přes pravé tlačítko na *Databases* a zvolením nabídky *New Database*. V nově otevřeném okně se stanoví počáteční nastavení, jméno databáze, její vlastník a vlastnosti včetně maximální povolené velikosti na pevném disku primárního souboru, sekundárních souborů a transakčního logu. Nastavením položky *compatibility level* v záložce možnosti *(Options)* je definováno, s jakou nejstarší verzí Microsoft SQL serveru bude vytvářená databáze kompatibilní. Po odklepnutí tlačítka OK je nová uživatelská databáze vytvořena a zbývá ji jen nastavit strukturu.

MS SQL je relační databáze založená na tabulkách. Tabulku lze přidat otevřením nabídky *table* příslušné databáze, dále *new* a *new table*. V nově otevřeném okně jsou nastaveny parametry tabulky tj. názvy vlastností (první sloupec), datové typy (druhý sloupec), které jednotlivé položky obsahují, viz obr. 12. Poslední sloupec nastavuje nutnost zápisu do buňky. Nejčastěji bývá vynucován zápis do buněk obsahující primární nebo sekundární klíč. Podle pravidel relační databáze nesmí existovat v databázi žádný záznam, který obsahuje prázdný nebo duplicitní primární klíč.



Obr. 12: Parametry tabulky FLOW

3.1.6 Connector na databázi

Databáze je uložená na vzdáleném serveru a je nutné do ní ukládat data. Po povolení SQL autentizace a vytvoření nového uživatele je možné vytvořit program napsaný v pythonu umožňující zápis a čtení do databáze. Python je multiplatformní programovací jazyk, který se hodí pro vzájemnou kompatibilitu mezi operačními systémy.

Základní koncept vychází z python balíku *pyodbc* obsahujícího metody pro připojení k Microsoft SQL databázi. Tato knihovna je volně ke stažení z pypi repositáře. Instalace je velmi jednoduchá, přičemž stačí do příkazového řádku nebo PowerShellu Windows (případně konzole v OS Linux) zadat příkaz *pip install pyodbc*.

Správa připojení je zobrazena na následujících řádcích:

Programový kód 1: Ukázka připojení k databázi

```
1  conParam=["localhost", "Flow", "garfield", "**********"]
2
3  connection = pyodbc.connect('DRIVER={ODBC Driver 13 for SQL
   Server};SERVER='+conParam[0]+';DATABASE='+conParam[1]+';UID
   ='+conParam[2]+';PWD='+conParam[3])     #vytvoreni spojeni
   cursor = connection.cursor()
```

První řádek ukládá do pole *conParam* řetězce s parametry pro připojení. Jsou to v pořadí parametry názvu nebo IP adresy serveru, kde je spuštěn databázový systém, název připojované databáze, jméno a heslo uživatele, oprávněného pracovat s databází. Na druhém řádku je otevřeno spojení s definovanými parametry. Hodnota pole DRIVER určuje do jaké verze systému MS SQL je vytvářeno spojení. Po úspěšném vytvoření spojení je vytvořen ukazatel na připojení a lze s připojením pracovat. Po ukončení práce je spojení ukončeno příkazem *connection.commit()*

Programový kód 2: Ukázka dotazu SELECT

```
1  cursor.execute("SELECT * FROM
   ["+conParam[1]+"].[dbo].[Flow]")
2  item = cursor.fetchone()
```

Program s databází komunikuje prostřednictvím SQL jazyka. Metoda *execute()* obsahuje jako parametr, jehož obsahem je SQL dotaz, uložený jako textový řetězec. Odpověď od SQL serveru zpracovává metoda *fetchone()*. Jejím zavoláním na ukazatel spojení jsou uloženy do proměnné získané hodnoty z databáze FLOW. Pro ukládání do databáze je použit následující příkaz.

Programový kód 3: Ukázka dotazu INSERT

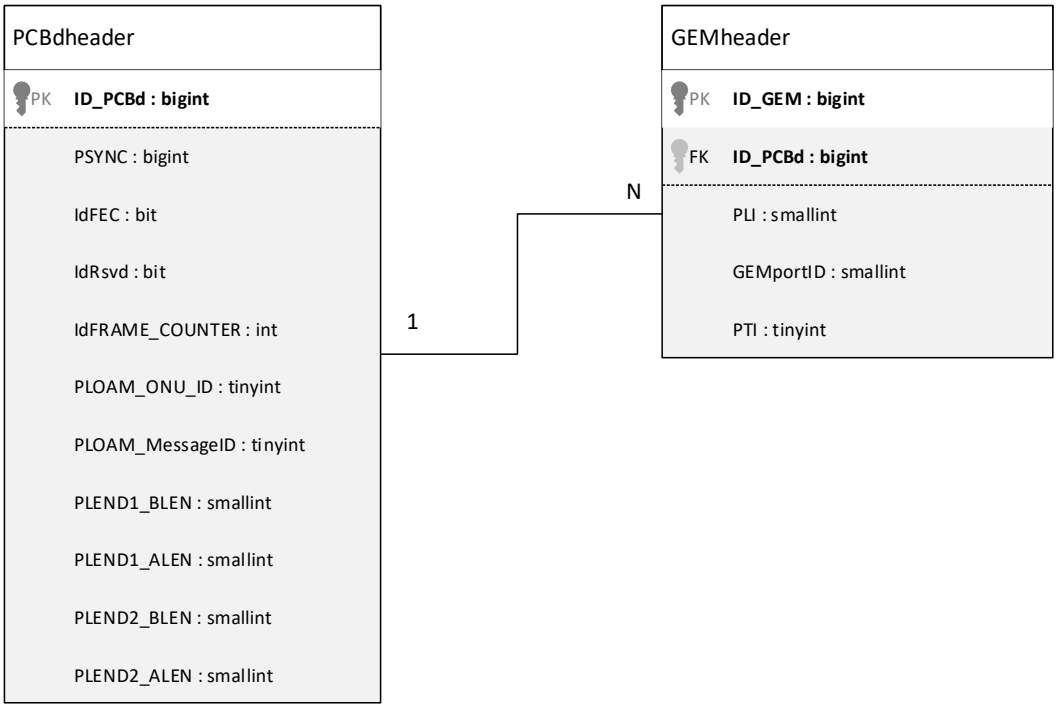
```
record=[str(123456),str(12345), str(1234), str(123),
"text1", "text2", "text3", "text4", "text5"]

cursor.execute("INSERT INTO [dbo].[Flow](SYS_UPTIME, UNIXS,
FIRST, LAST, IP_SOURCE, IP_DEST, PROTO, SOURCE_PROTO,
DEST_PROTO) VALUES ("\

+record[0]+", "+record[1]+", "+record[2]+", "+record[3]+","
\ +record[4]+", "+record[5]+", "+record[6]+",
"+record[7]+"," \ +record[8]+")"
```

Funkce *cursor.execute()* znovu zasílá dotaz SQL serveru. Do parametru metody je vložen dotaz na vložení záznamu do tabulky INSERT. Parametry jednotlivých buněk tabulky jsou uloženy v proměnné *record*.

3.2 První návrh databáze

Zachycená data sondou FPGA je potřeba uložit pro další zpracování a analýzu. Jakákoliv data, ukládaná na pevný disk, musí mít předem definovanou strukturu a formát. Získaná data lze ukládat do souborů, ale nevýhodou tohoto způsobu ukládání je další práce s daty. Při srovnání souborů s databázovým systémem je zpracování souborů složitější, méně bezpečné a neopatrnou manipulací s textovým soborem může dojít k úplné ztrátě dat. Výhodou použití databázového systému je nadefinování struktury pro ukládání dat a nastavení oprávnění jednotlivým uživatelům. Pravidla, definovaná jednotlivým uživatelům, mohou uživatelům stanovit různá bezpečnostní opatření například čtení, ukládání, mazání, editaci atd.

Pro danou problematiku byla navržena databáze GPON, skládající se ze dvou tabulek PCBdheader a GEMheader, obsahujících informace o zachyceném provozu. Ze zachycených dat jsou odděleny informace týkající se záhlaví a dat přenášených GPON rámcem. Obsah hlavičky rámce, označovaný jako PCBd, je ukládán do tabulky PCBDheader. Přenášená data rámcem (GTC payload) jsou dále rozkódována do podoby přenášených GEM rámců a opět rozdělena na přenášená data (GEM payload) a hlavičku (GEM header). Data ze záhlaví GEM rámce jsou uložena v tabulce GEMheader. Do obou tabulek jsou ukládány pouze užitečné informace pro uživatele bez informací o kódovacích a paritních bitech. Strukturu tabulek lze vidět na následujícím obrázku:



Obr. 13: Návrh databáze

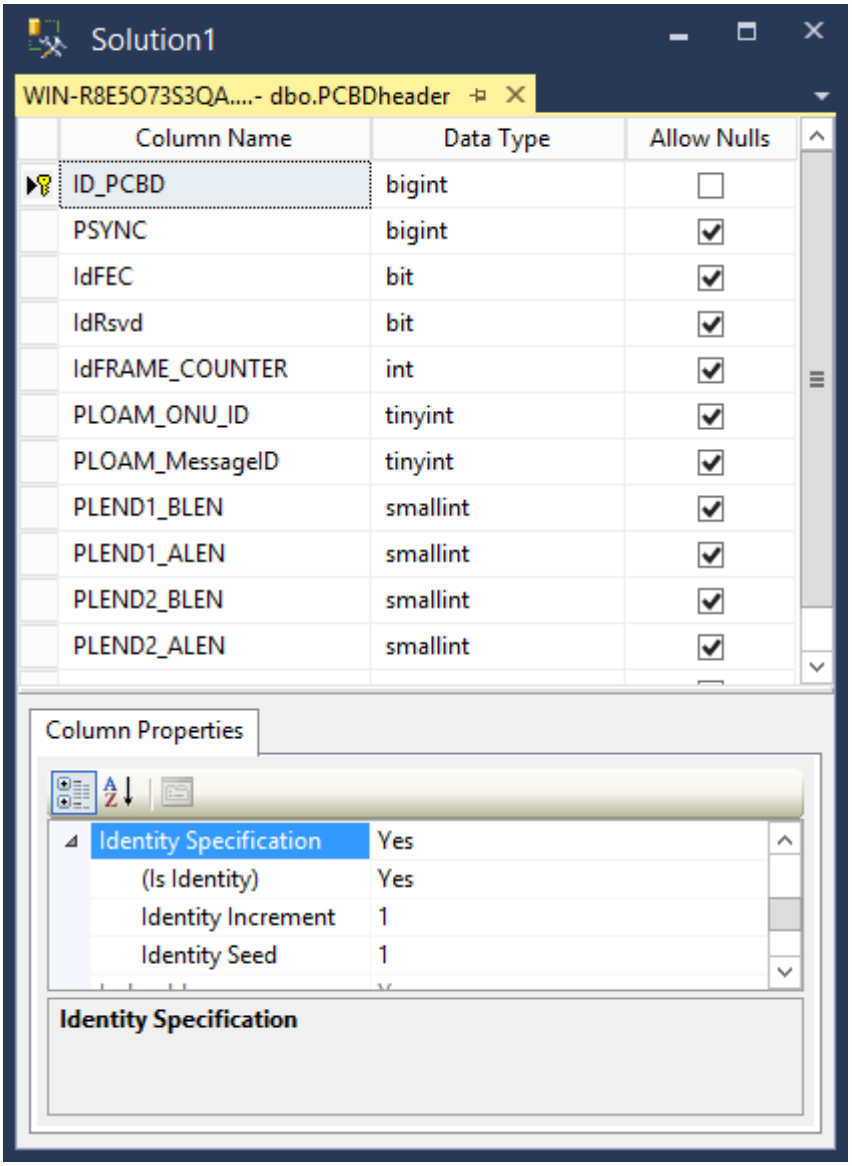
Tabulka se záhlavím GPON rámce obsahuje na každém řádku informace z polí PSYNC, IDENT, PLOAM a obou polí PLEND. Hodnoty jsou ukládány do celočíselných proměnných v decimálním tvaru. Maximální velikost proměnné je definována s ohledem na maximální ukládanou hodnotu. Prvním sloupcem tabulky je automaticky inkrementovaný primární klíč ID_PCBd stanovující pořadí záznamu v tabulce. Druhý sloupec, obsahující položku PSYNC, označuje informace o začátku rámce. Pokud by hodnoty v tomto sloupci byly různé, jedná se o chybu při dekódování dat. Pole IDENT je rozděleno do třech sloupců. Pole IdFEC může nabývat maximální hodnoty 1. Odpovídá tak logickému datovému typu. Pokud je v rámci použito zabezpečení FEC kódováním, je údaj v tabulce nastaven na jedničku a odpovídá logické hodnotě true. Další bit je rezervní a měl by být koncovými jednotkami na síti ignorován a nastaven na nulu. Poměrně důležitým údajem z pole IDENT je počítadlo rámců, označující číslo přenášeného rámce. Ze zachycených dat pak lze snadno detekovat chyby při zachytávání, protože každá hodnota tohoto pole by měla být s každým rámcem inkrementována o jedničku. Ze zprávy PLOAM jsou pro downstream důležité hodnoty ONU-ID, které identifikuje danou ONU a pole messageID definující typ řídící zprávy. Zbylé sloupce se týkají polí PLEND. Při bezchybném přenosu by měla být data sobou polí stejná. Hodnota pole BLEN určuje velikost pole BWmap a tím i konec PCBd záhlaví. Další pole, bezvýznamné pro přenos, je ALEN. Těchto 12 bitů by mělo být opět ignorováno koncovými

jednotkami a nastaveno na nuly. Důvod zachytávání těchto polí je tuto skutečnost ověřit, případně detekovat nestandartní chování.

Tabulka GEMheader je v porovnání s PCBdheader obsahuje méně informací a je z podstaty provozu na GPON síti v relaci 1:N. Jeden GPON rámec obsahuje více přenášených GEM rámců. Parametry tabulky jsou tak tvořeny ze dvou klíčů, a to primárního ID_GEM a cizího ID_PCBd. První klíč je automaticky inkrementován určuje jednoznačnost záznamu v tabulce. Druhý klíč je převzat z tabulky PCBdheader a určuje, ke kterému PCBd záhlaví, resp. ve kterém GPON rámci byla data přenesena. Dále následuje trojice sloupců s informacemi se záhlavím GEM rámce. V poli PLI je stanoveno, kolik bajtů je přenášeno v těle rámce a polem PTI je upřesněno, zdali je rámec kompletní. Poslední hodnota GEMportID ukládá informace o provozu. Na základě této hodnoty je stanoven virtuální komunikační kanál mezi OLT a ONU. Koncové jednotky očekávají od OLT definovanou hodnotu GEMportID. Pokud je příchozí hodnota neodpovídá očekávané, bývá automaticky zahozena. Tímto mechanismem je zajištěno přiřazování správných dat do koncových sítí za koncovou optickou jednotkou.

3.2.1 Implementace

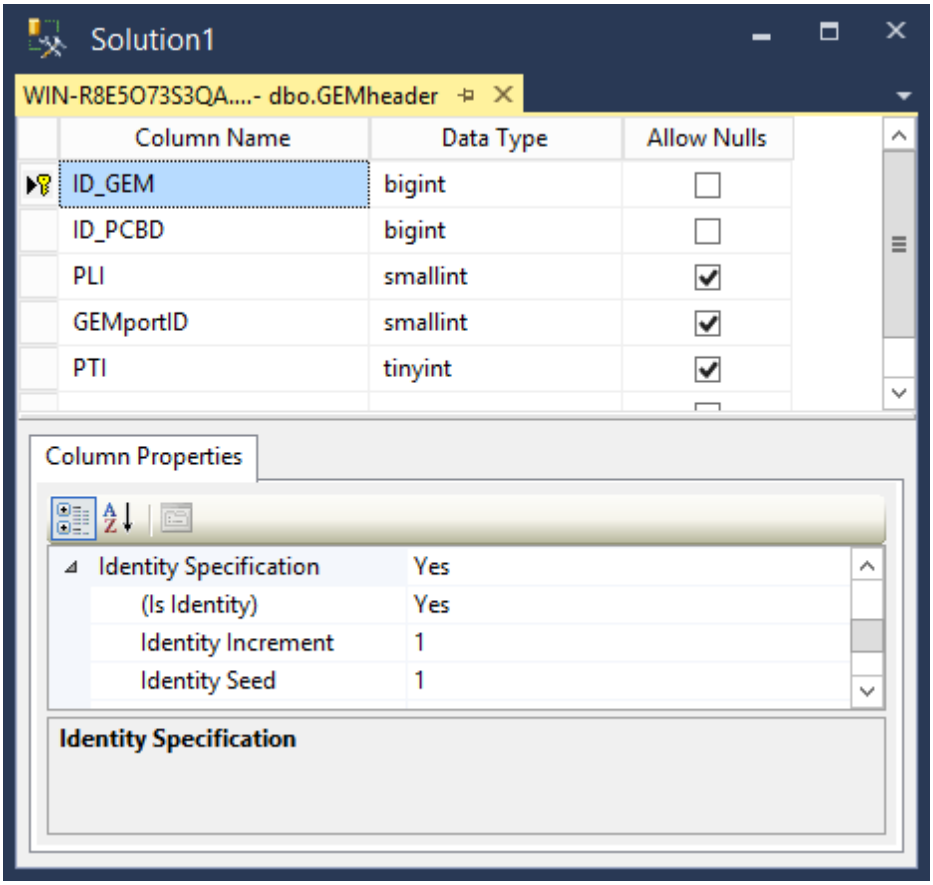
Po vytvoření UML modelu databáze následuje implementace do MS SQL systému. Databáze GPON je vytvořena stejným postupem jako databáze FLOW. Po vyvolání dialogového okna pro vytvoření databáze *(Database – New Database)*, je vyplněn název databáze GPON a databáze vytvořena stiskem tlačítka OK. Nabídkou pro vytváření tabulek (*Table – New – Table)* jsou postupně definovány obě tabulky. Nastavení parametrů tabulky PCBdheader ukazuje obr. 14.



Obr. 14: Nastavení parametrů tabulky PCBdheader

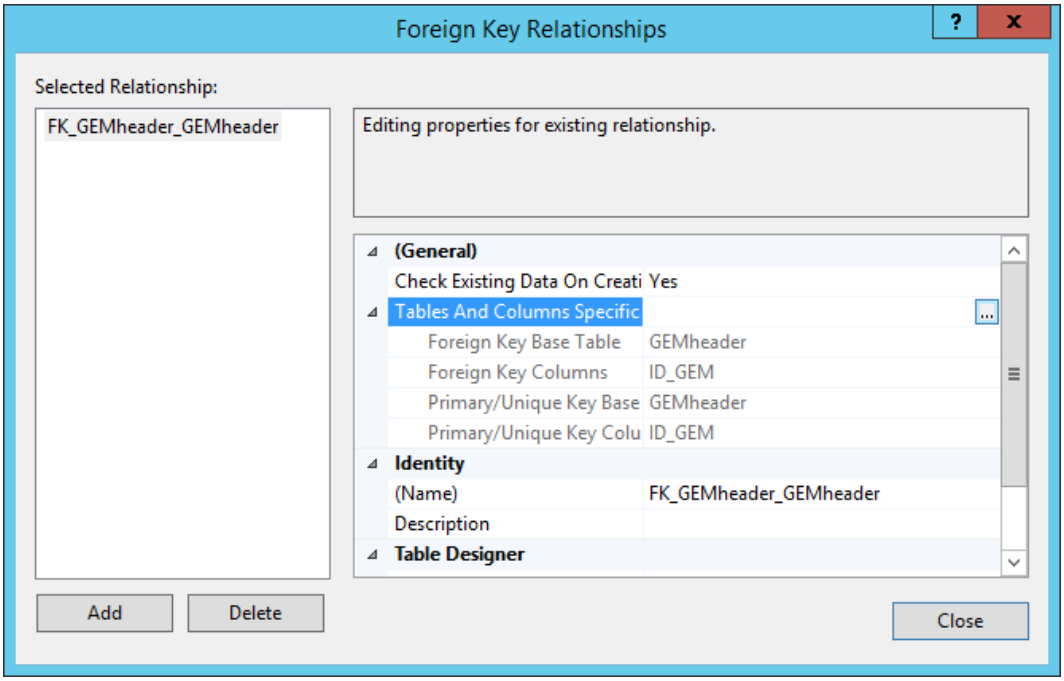
Po vyplnění parametrů je nezbytné definovat primární klíč tabulky. Nastavení klíče je provedeno označením položky (ID_PCBd) vyvoláním nabídky přes pravé tlačítko myši a volbou *Set Primary Key*. Pro nastavení automatické inkrementaci slouží nabídka *Identity Specification* v záložce *Column Properties*. Obr. 14 ukazuje i nastavení velikost automatické inkrementace klíče. Po uložení je vytvořena a zobrazena tabulka v mezi tabulkami databáze GPON.

Stejným způsobem je vytvořena i druhá tabulka GEMheader, jejíž nastavení zobrazuje obr. 15.



Obr. 15: Nastavení parametrů tabulky GEMheader

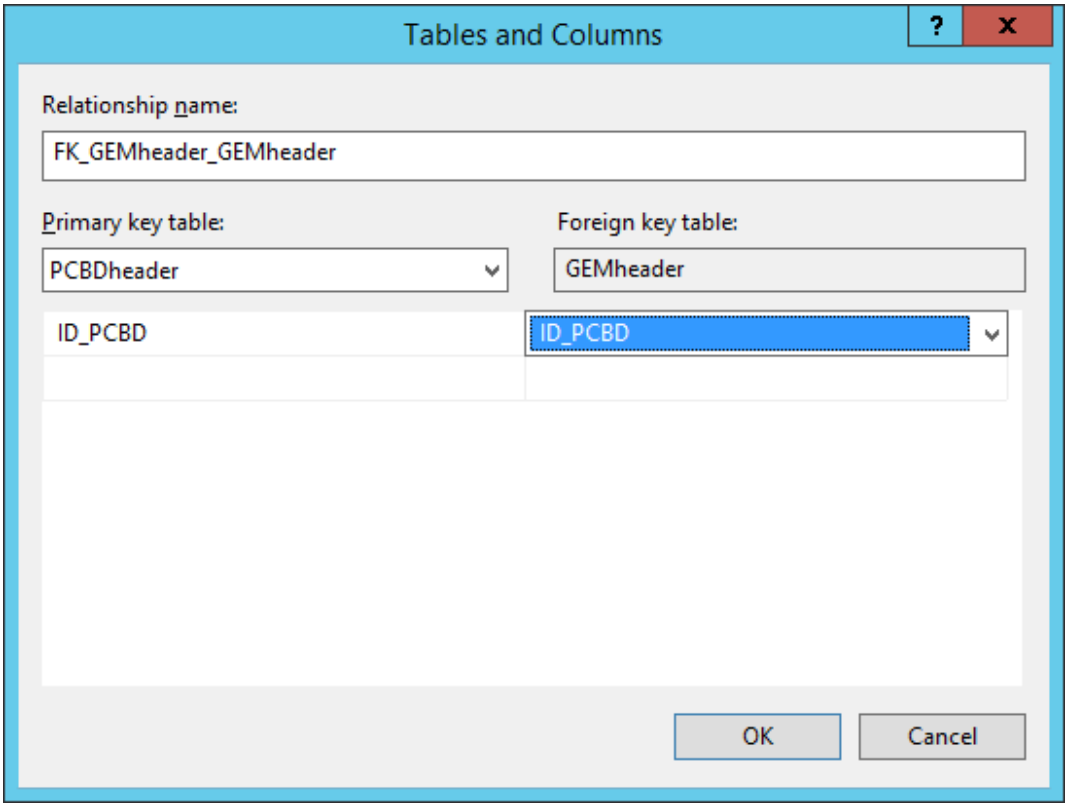
V uložené tabulce je nutné nadefinovat cizí klíč (ID_PCBd) z předchozí tabulky. Nastavení klíče vytváří relaci 1:N mezi tabulkami a tím znemožní zápis hlavičky GEM rámce bez přiřazení ke ID_PCBd klíči. Tímto spřažením je nastaveno, který GPON rámec daný GEM rámec přenášel. Po rozbalení tabulky GEMheader v *Object Exploreru* jsou zobrazeny všechny vlastnosti příslušné tabulky. Složka obsahuje všechny nadefinované klíče tabulky. Vyvoláním kontextové nabídky nad složkou *Keys* a zvolením nabídky *New Foreign Keys*, jsou zobrazeny vlastnosti klíče (viz obr. 16).



Obr. 16: Zobrazení parametrů cizího klíče

Kliknutím na tlačítko s tečkami je v dalším okně zobrazeno samotné nastavení, kde pole *Relationship name* obsahuje název cizího klíče. Tento klíč může být libovolně definován uživatelem, ale pro lepší orientaci je uváděn ve tvaru *FK_aktualniTabulka_zdrojovaTabulka*. V dalším kroce musí uživatel definovat, ze které tabulky má být použit primární klíč *(Primary key table)* a stanovit, které sloupce budou propojeny. Nastavení pro tento klíč je zobrazuje obr. 15. Potvrzením akce tlačítkem OK jsou aplikovány změny na nadřazené okno (obr. 16). Před potvrzením nastavení jsou nutná nastavit pravidla *Delete Rule* a *Update Rule* v sekci *Table Designer – INSERT And Update specific* pro chování cizího klíče při mazání a vkládání záznamů do tabulky. Z výběrové nabídky jsou k dispozici možnosti [26]:

- Žádná akce *(No action)* – Uživateli není povoleno akci provést. O situaci je informován dialogovým oknem.
- Kaskáda *(Cascade)* – Odstraní všechny záznamy s cizím klíčem.
- Nastavit hodnotu Null *(Set Null)* – Nastaví hodnotu záznamu na Null, pokud to zápis do položky umožňuje.
- Nastavit výchozí *(Set default)* – Nastaví výchozí hodnotu daného sloupce.

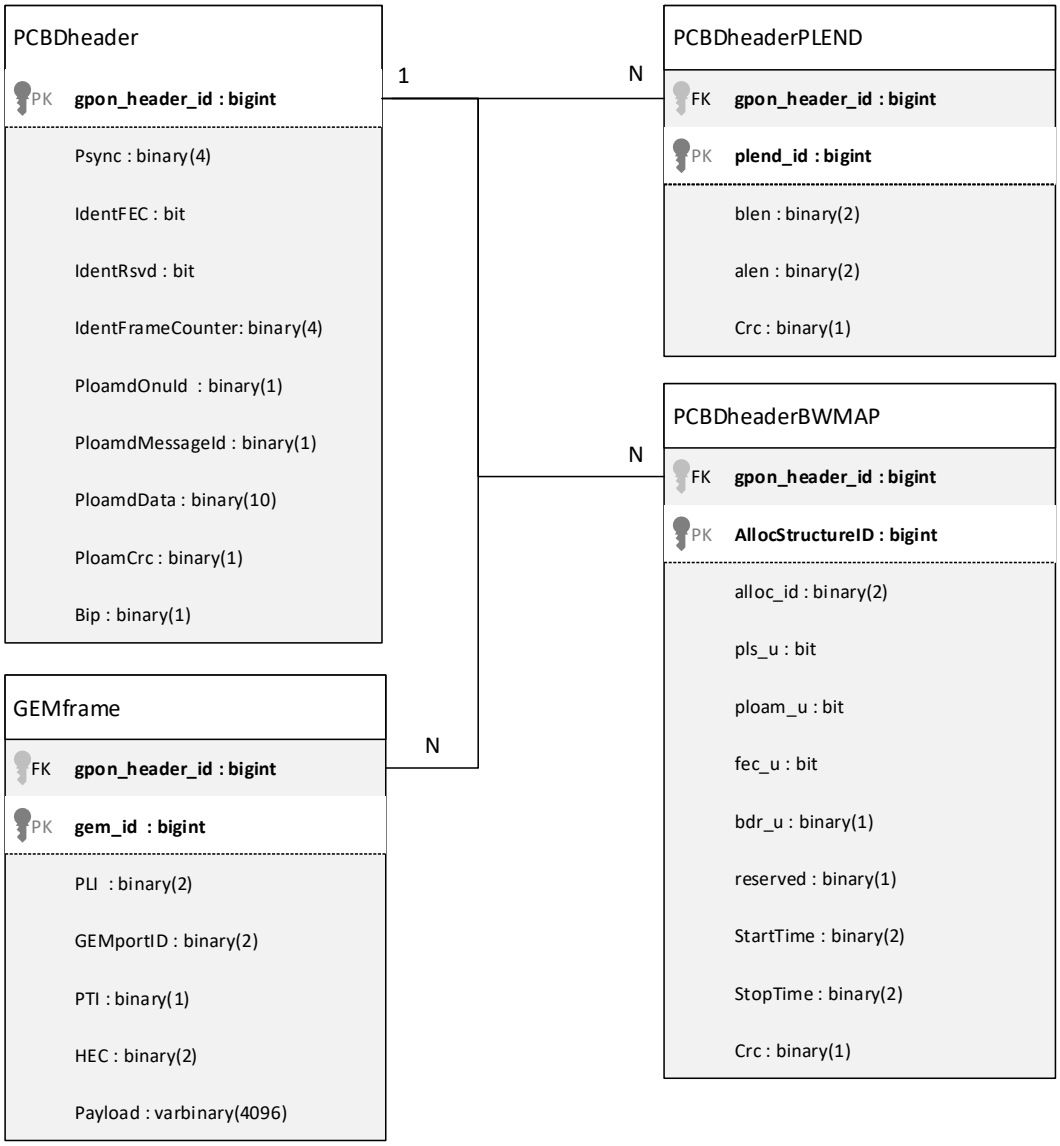


Obr. 17: Nastavení cizího klíče

3.3 Finální návrh databáze

Po implementaci a při pokusu použít porovnání návrh bylo odhaleno několik zásadních nedostatků. Navržená databáze byla navržena tak, aby uchovávala pouze čitelné informace pro člověka a nepočítala s žádnou podrobnou analýzou na úrovni celých rámců. Její účel byl pouze pro zobrazení přenášených dat a analýzu o tom jaká spojení jsou vytvářena. Proto bylo potřebné upravit resp. vytvořit nový návrh, který by tyto nedostatky eliminoval a umožnil data ukládat binárně, aby je bylo možné dalšími nástroji a nejen SQL serverem zpracovávat.

Po odhalení nedostatků byl vytvořen návrh, který umožňuje nejen ukládat celé GPON rámce, ale hlavně ukládat je v binární podobě. Nevýhodou proti prvnímu návrhu je skutečnost, že není uvažováno ukládat data ihned v podobě GEM rámců, proto s nimi není v práci pracováno a práce je zaměřena pouze na rámce GPON. GEM rámce byli vyloučeny z důvodu dalšího nutného parsování GPON dat, které už parser GPON rámců neobstarává a je nutné pro tohle parsování vytvořit parser jiný.



Obr. 18: Finální návrh databáze

Na obrázku 18 je uvedeno schéma nového návrhu. Je složeno ze tří základních tabulek pro ukládání dat a to PCBDheader, PCBDheaderPLEND a PCBDheaderBWMAP. Protože parser pracuje sekvenčně, jsou data do tabulek ukládána v tomto pořadí. Nejdříve jsou uloženy údaje do tabulky PCBDheader, která si vygeneruje automaticky primární klíč gpon_header_id a s tímto klíčem jsou pole PLEND a BWMAP propojena v následujících tabulkách vazbou 1:N. Tato vazba je zavedena z důvodu většího polí PLEND a BWMAP. Každý rámec přenáší dvě pole PLEND a jedno pole BWMAP. Údaje z pole BWMAP jsou ukládána po jednotlivých strukturách, proto je i zde zavedena vazba 1:N.

Tabulka PCBDheader je nejdůležitějším z těchto tří. První pole definuje primární klíč definovaný přes celý databázový systém a je pro něj zaveden datový typ bigint. Tento obrovský typ je zaveden záměrně, aby nedošlo k vyčerpání primárních klíčů při ukládání.

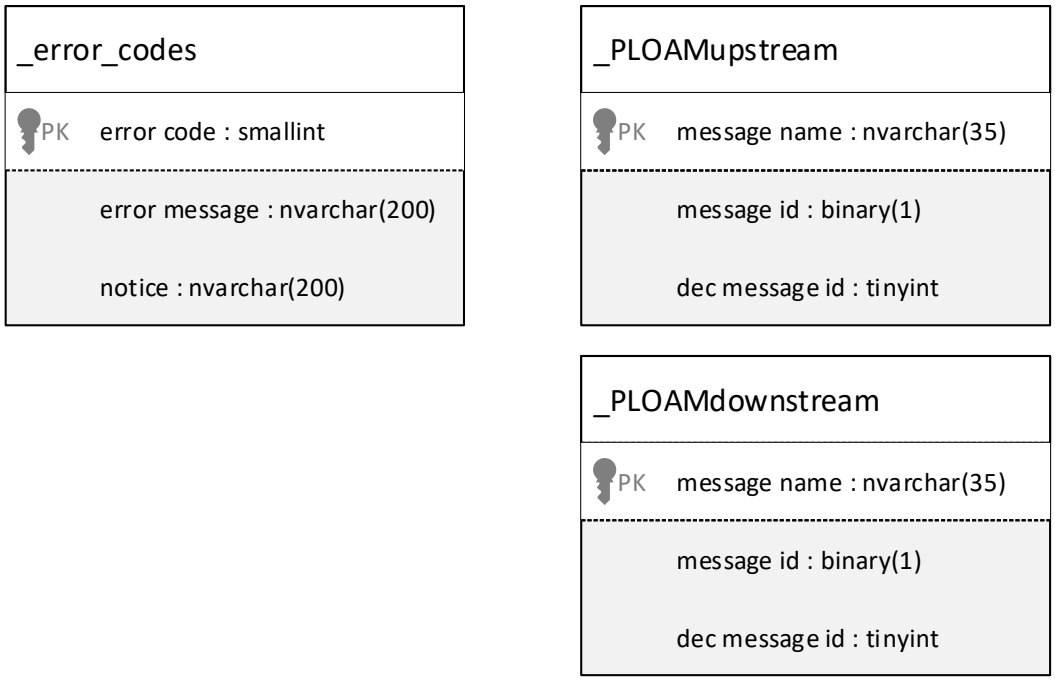
S použitím datového typu int by k vyčerpání klíčů došlo při rychlosti 8000 rámců za sekundu přibližně po 149 hodinách, zatímco u datového typu bigint je tento rozsah nevyčerpatelný, protože lze vytvořit až 2^{63}-1 klíčů. Druhé pole tabulky ukládá informaci o synchronizační posloupnosti PSYNC podle standardu jde o posloupnost 4 bajtů, která je do MS SQL databáze uložena jako datový typ binary o délce 4 bajty. Po synchroznizační posloupnosti je do třech sloupců ukládáno pole IDENT jsou to sloupce IdentFEC, IdentRsvd IdentFrameCounter. IdentFEC a IdentRsvd jsou reprezentovány datovým typem bit. Tento datový typ uložňuje ukládat fyzicky pouze jeden bit do paměti počítače. Ukládání funguje na principu vyčlenění jednoho bajtu v paměti počítače a do něj lze uložit vždy jeden takový bit. Po naplnění bajtu naplněn osmi různými bity, je vytvořen další bajt pro devátý bit. Pole IdentFrameCounter je ukládáno jako 4 bajty, ale ve skutečnosti je ukládáno pouze 30 bitů a zbylé dva jsou doplněny nulami. Další čtyři sloupce v tabulce uchovávají sloupce o PLOAM zprávách, které jsou opět logicky rozděleny po významných bajtech – jeden bajt pro ONU-ID ve sloupci PloamdOnuId, jeden bajt pro uložení o jaký typ PLOAM zprávy se jedná ve sloupci PloamdMessageId, deset bajtů dat obsažených v PLOAM zprávě do sloupce PloamdData a poslední bajt uchovávající ve sloupci PloamCrc informaci z kontrolního součtu celého pole PLOAM. Poslední sloupec celé tabulky Bip je vyhrazen pro jeden bajt informující o použitém dopředném kódování BIP.

Druhá tabulka uchovává informace u polích PLEND ze záhlaví GPON rámce. Položka gpon_header_id je cizí klíč z tabulky PCBDheader. Druhý klíč plend_id je datového typu bigint a slouží jako primární klíč tabulky. Za ním následují sloupce blen, alen, Crc identifikující položky BLEN, ALEN a CRC v poli PLEND. Pro ukládání je opět použit binární datový typ binary s odpovídající velikostí.

Poslední tabulka ukládá jednotlivé alokované struktury z pole BWMAP, identifikovány primárním klíčem AllocStructureID opět jako bigint a cizím klíčem gpon_header_id z tabulky PCBDheader. Po nich následují sloupce pro jednotlivé položky alokované struktury pole BWMAP. Sloupec alloc_id o velikosti dvou bajtů definuje pro jakou ONU je alokovaná struktura určena. Skupina dalších čtyřech sloupců (pls_u, fec_u, bdr_u a reserved) je definována pro uložení příznaků. Po příznacích jsou vyčleněny dva sloupce StartTime a StopTime pro definování vysílacího času ONU jednotky. Poslední sloupce je stejně jako v předešlých tabulkách rezervován pro jeden bajt CRC kódování.

Databáze GPON obsahuje také několik tabulek, které jsou použity až při zpracovávání dat. Jejich struktura je zobrazena na obrázku 19. Tabulka _error_codes uchovává přehled návratových kódů, které jsou vráceny vykonanými procedurami. Zbylé dvě tabulky

_PLOAMdownstream a _PLOAMupstream uchovávají informace o PLOAM zprávách definovaných podle standardu ITU-T G983.4 z roku 2014.



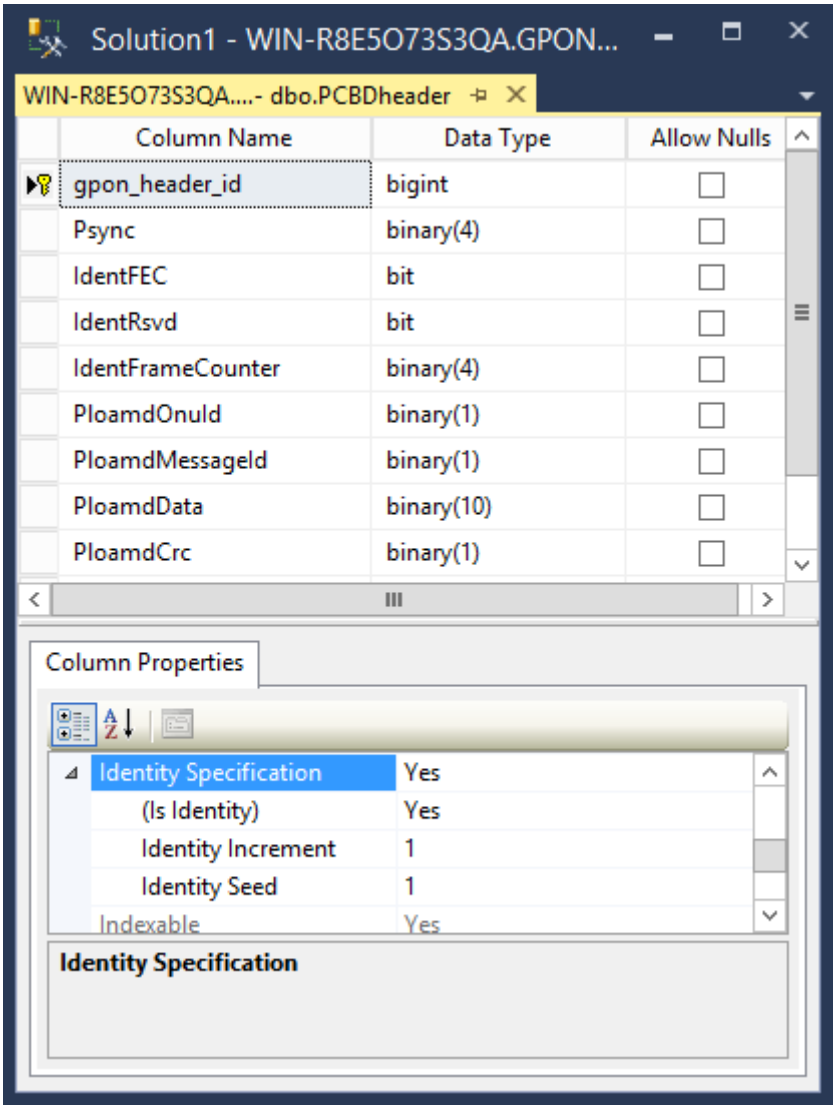
Obr. 19: Struktura tabulek nastaveními procedur

Poslední výraznou změnou prošla i tabulka uchovávající GEM rámce viz obrázek 18. Návaznost na tabulku PCBDheader zůstala zachována, ale způsob ukládání byl změněn také na binární datový typ binary a přibyla sloupec Payload pro ukládání dat. Tento sloupec je definován jako datový typ varbinary o maximální délce 4096 bajtů.

Většina hodnot byla převedena na datový typ binary. Ačkoliv se nejedná o dynamický datový typ je v tomto případě lepšípro ukládání dat, protože data jsou definována právě na nastavenou velikost, resp. nelze uložit jen určitý počet bitů. Jedinné využitý datového typu varbinary je při ukládání přenášených dat, protože jejich délka je opravdu proměnlivá. Rozdíl mezi typy je v ukládání do paměti počítače. Datový typ binary uloží do paměti hodnotu o stejné velikosti, jak je v databázi definováno. U proměnlivé délky datového typu varbinary jsou ke stanovené velikosti přičteny pokaždé dva bajty uchovávající údaj o tom, jaká je skutečná velikost hodnoty.

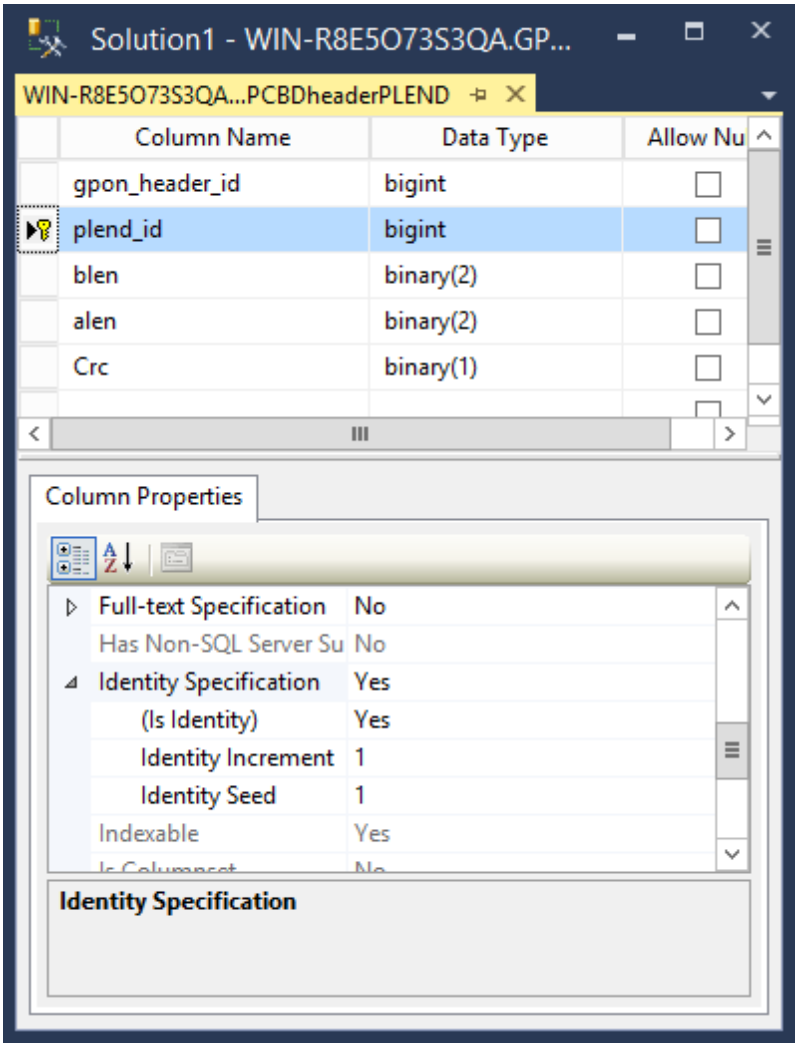
3.3.1 Implementace

S upraveným modelem bylo jednodušší původní tabulku PCBdheader smazat SQL příkazem *DROP TABLE PCBdheader* a vytvořit novou v průzkumníku databáze GPON kliknutím na vyvoláním nabídky *(Tables – New – Table…)* a vytvořit nové formáty podle obrázku 20.

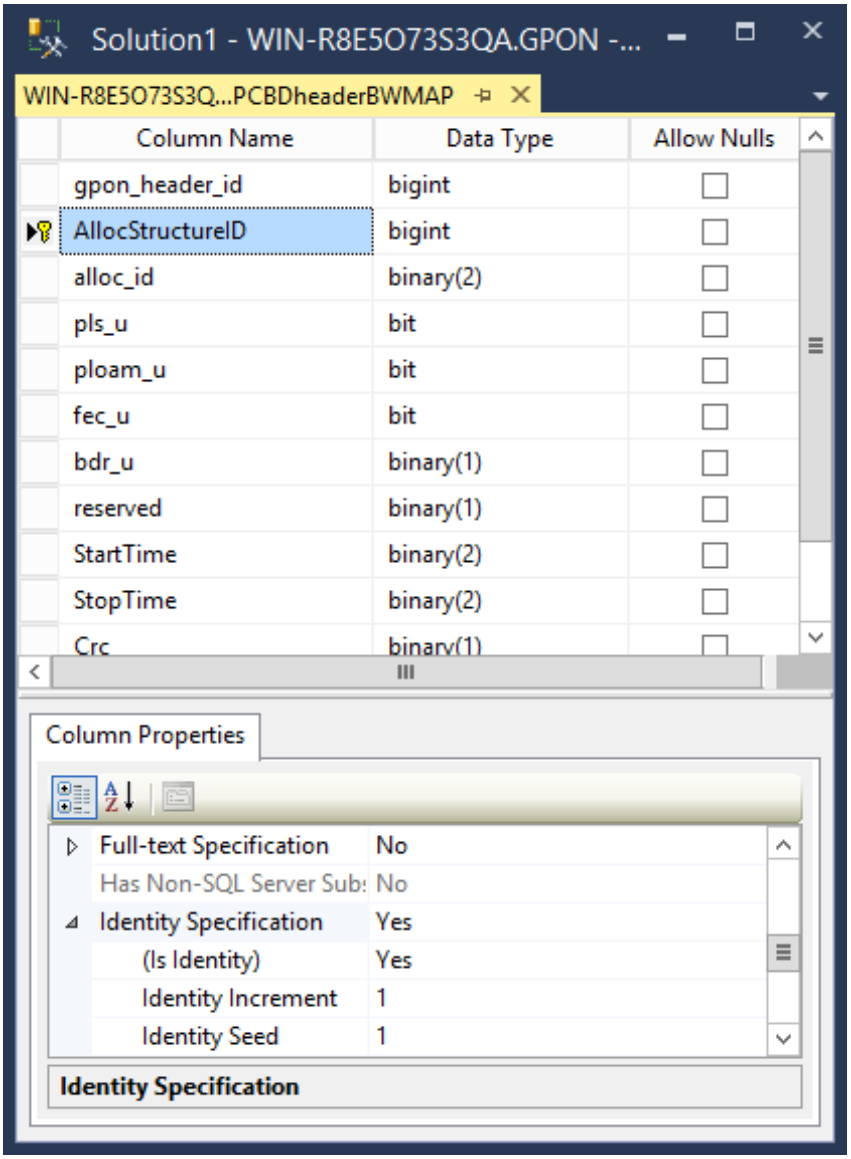


Obr. 20: Vytvoření tabulky PCBDheader

Stejným způsobem byly vytvořeny i tabulky pro zachytávání zbylé části GPON rámce PCBDheaderPLEND a PCBDheaderBWMAP viz obrázky 21 a 22.

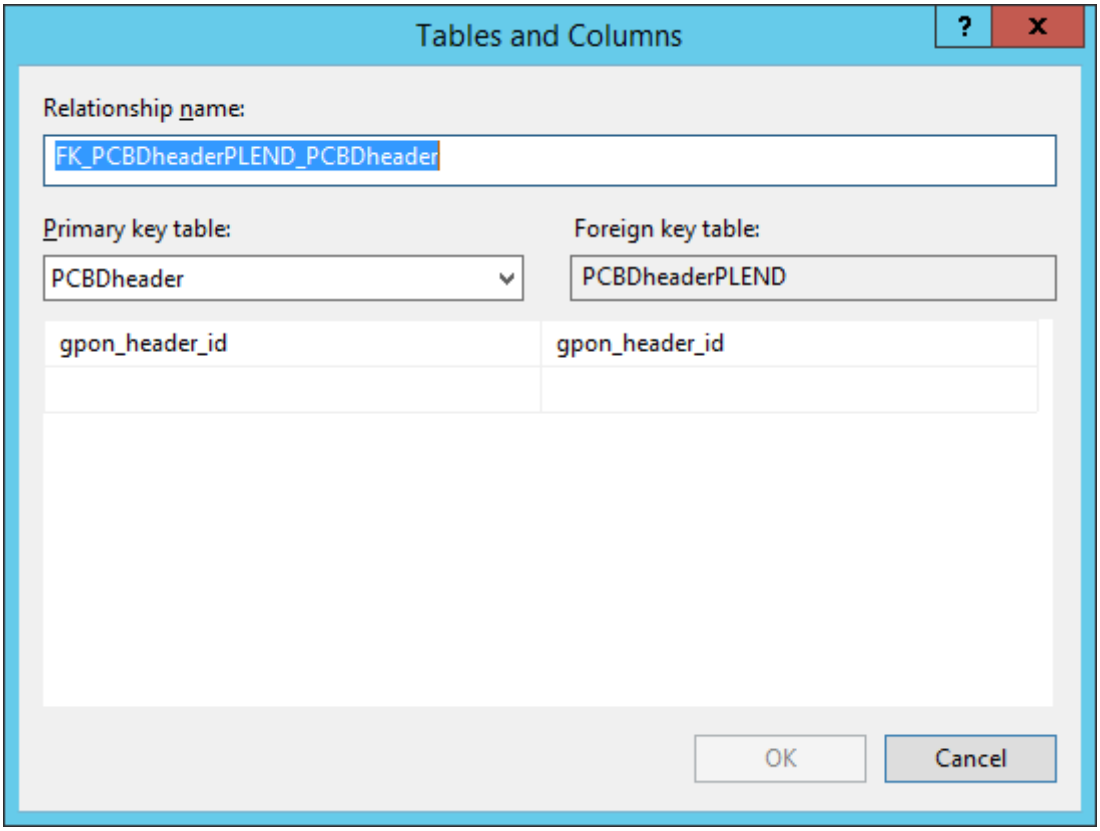


Obr. 21: Vytvoření tabulky PCBDheaderPLEND

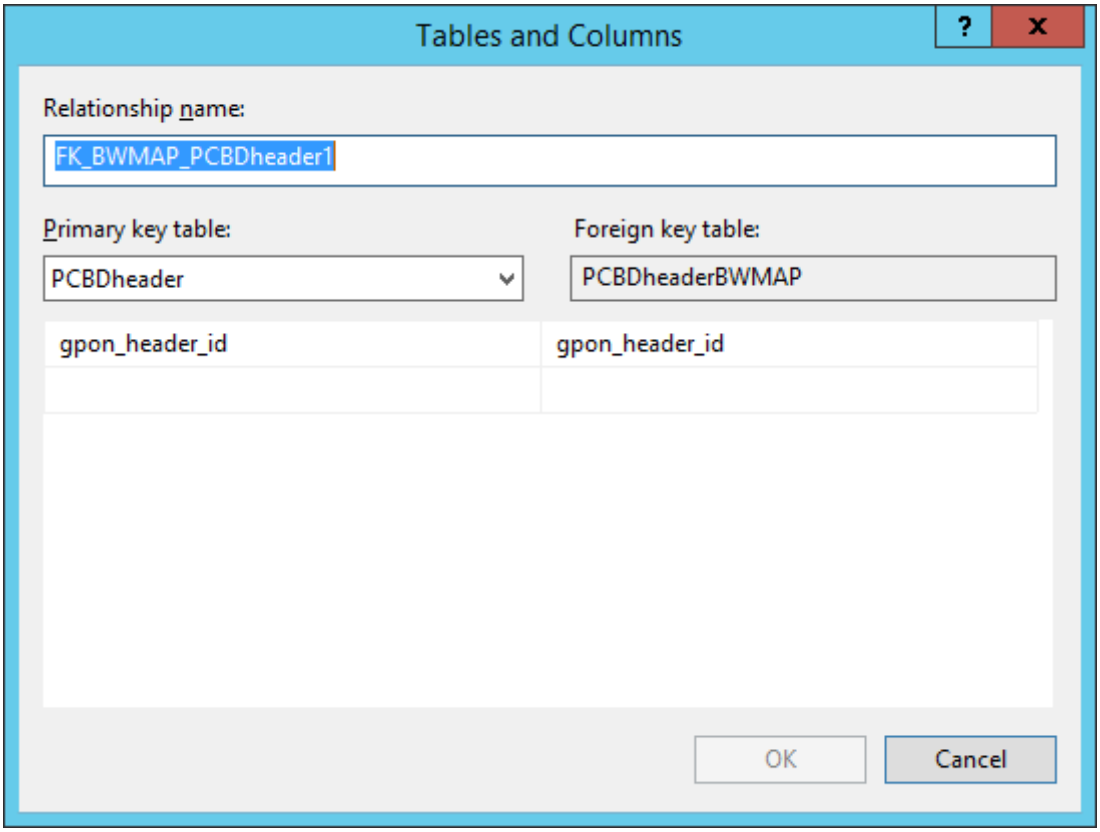


Obr. 22: Vytvoření tabulky PCBDheaderBWMAP

Po vytvoření posledních dvou uvedených tabulek je nutné je propojit cizím klíčem gpon_header_id a vytvořit tak relace 1:N mezi tabulkami. Postup nastavení cizího klíče je stejný jako v kapitole 3.2.1, rozdílné jsou pouze hodnoty v posledním kroku postupu viz obrázky 23 a 24.



Obr. 23: Nastavení cizího klíče v tabulce PCBDheaderPLEND



Obr. 24: Nastavení cizího klíče v tabulce PCBDheaderBWMAP

Při práci s databází a spouštění jednotlivých procedur a funkcí budou postupně dynamicky vytvářet další tabulky. Struktura těchto tabulek je dána jednotlivými procedurami, které tyto tabulky vytváří, ale ve většině případů bude jejich struktura a datové typy převzaty z původní tabulky PCBDheader.

3.4 Procedury a funkce

Silnou funkcionalitou databázového systému je možnost používání procedur a funkcí. Jsou to bloky kódu, umožňující sekvenční zpracovávání a jsou uloženy na SQL serveru. Do procedur a funkcí lze zapsat mnohonásobné a různě provázané SQL dotazy, jejichž funkčnost může být modifikovaná různými vstupními parametry. Rozlišujeme procedury a funkce podle jejich výstupu na dva typy:

- Skalární – výstupem je jedna skalární hodnota, např. číslo
- Tabulkové – výstupem je tabulka

Výhodou jejich použití je vytvoření pokročilých dotazů, které mohou být volány jedním příkazem a provedou pokročilé transakce či různé manipulace s tabulkami a daty aniž by uživatel musel tyto dotazy vytvářet a volat prostřednictvím SQL serveru sám. V tělech procedur lze použít i některé běžné programátorské syntaxe jako jsou podmínky a cykly.

Další jejich nespornou výhodou procedur a funkcí je zvýšení bezpečnosti při manipulace s daty. Jejich tělo může být šifrovaně na MS SQL serveru uloženo a uživatel tak nemá přehled o jejím vnitřním kódu a při vhodném omezení práv pro vzdálené volání procedur lze snadno vytvořit i aplikační rozhraní.

4 ZPRACOVÁNÍ DAT

Každý databázový systém je navržen jako základna pro ukládání dat. Tato data je důležité vhodně uložit a ve většině případů jsou data i dále zpracovávána, tříděna, přesouvána atp.

Pro uložení dat a následné další zpracování jsou data v této práci ukládávána v surové formě tj. ve formě binárních řetězců. Výhodou ukládání dat do binární podoby je ve skutečnosti snížení výpočetní kapacity pro následující práci s externími programy, skripty atd. V případě parseru by převádění přijatých dat z binární podoby na čitelnou hodnotu zbytečně prodloužilo čas zpracování a i když jde o početní operace prováděné na výkonných strojích s více procesory a poli SSD disků. Výhodou databázového systému MS SQL je možnost zobrazit binárně uložená data v přijatelné podobě jako jsou řetězce s hexadecimálními hodnotami. Tyto data lze v určité míře zpracovávat a převádět, pokud je to nezbytně nutné, do decimálního, pro člověka snadno čitelného tvaru. V některých případech je tento převod při přímém ukládání spíše nežádoucí protože pro uložení jednoho bitu musíme použít např. buňku s celočíselným datem, jehož velikost pro uložení na disku počítače již není jeden bit, ale je mnohonásobně větší. Po uložení většího množství takto převedených dat dochází i k obrovskému nárůstu velikosti databáze. Za jednu vteřinu je v sestupném směru GPON sítě přeneseno 8000 GPON rámců a při této stejné rychlosti je přeneseno za jednu hodinu 28 milionů GPON rámců. Proto při práci s daty je nutné vzít tyto číselné údaje v úvahu.

Navržená databáze ukládá data ze sestupného směru GPON sítě. V praktické části práce je s těmito daty prováděno filtrování dat a analýza vztažená na celou komunikaci všech jednotek nebo konkrétní jednotky ONU.

Analyzační část je složena ze dvou celků a to podle způsobu zpracování. Nejvíce náročné operace na zpracování jsou prováděny na SQL serveru, zatímco zobrazení vyfiltrovaných dat je prováděno jazykem Python. Cílem tohoto rozdělení bylo vytvořit na MS SQL serveru jednoduché aplikační rozhraní za pomocí uložených procedur a funkcí, které bude z daty provádět obsáhlé transakce. Skrze toto rozhraní jsou data předávána k finálnímu Python skriptu. Předpoklad využití tohoto skriptu je cílen spíše na běžnou počítačovou stanici, která data zobrazí uživateli a nemusí vždy disponovat takovým početním výkonem jako MS SQL server.

Dalším důvodem pro vytvoření rozhraní bylo zvýšení rychlosti zpracování, zpřístupnění dat a jejich vizualizace do přehlednější podoby nejen pro další budoucí aplikace např. při tvorbě jiného softwaru.

4.1 Zpracování dat T-SQL

Zpracování na MS SQL serveru je realizováno procedurami a funkcemi. Vytvořené procedury a funkce jsou uloženy na serveru a běžný uživatel nemá přístup ke kódu funkce. Princip práce je založen na zavolání procedury nebo funkce, ta vykoná vnitřně definovanou sekvenci příkazů a provede navrácení hodnoty nebo některé funkce uloží zpracovaná data do nové nebo upravené již vytvořené tabulky. Některé procedury a funkce, zejména ty které používají jako výstup tabulku, vrací jako návratovou hodnotu číselnou hodnotu informující o chybovém stavu. Navrácená chybová hodnota s označením nula symbolizuje úspěšně vykonaný příkaz. Chybové hodnoty jsou označeny, ačkoliv trochu nelogickým způsobem, zápornou celočíselnou hodnotou. Záporná hodnota byla při vytváření využita záměrně, z důvodu odlišení od běžné hodnoty. Jazyk T-SQL vrací jako výstupný hodnoty parametrů a procedur ve většině případů požadovanou hodnotu. Pokud však došlo k chybě při zpracování např. při nevalidním SQL dotazu, je uživatel o této skutečnosti vrácen chybovou hláškou. Problém však nastává u uživatelských procedur a funkcí. I když je kód vykonán v pořádku, nemusí být vypsána validní hodnota. Dále pro ošetření kritických stavů, například pokud procedura hledá údaje, které v tabulce nejsou nalezeny, jsou navráceny právě tyto záporné hodnoty. Přehled navrácených a aktuálně definovaných hodnot je uložen v tabulce s názvem _error_codes.

Vytvořené procedury, funkce i tabulky mají specifické označování názvů za účelem bližší identifikace. Se současným a zvláště i z předpokládaným budoucím nárůstem objektů je počítáno s jistou nepřehledností při pojmenování. Ačkoliv na funkčnost serveru či kódu pojmenování nemá zásadní vliv je navržená struktura oddělována logicky na interní a obecné objekty. Za objekty interní jsou na serveru označeny všechny názvy obsahující na začátku podtržítko, zatímco běžné tabulky jsou pojmenovány bez této syntaxe. Jako interní objekty jsou označeny ty objekty, které ukládají data pro zpracování, například chybové kódy (tabulka _erroc_codes) a standardem definované PLOAM zprávy (tabulky _PLOAMdownstream a _PLOAMupstream). Tato analogie byla přejata ze syntaxe objektově orientovaného programování v Pythonu.

Všechny výstupy z procedur a funkcí jsou ukládány do stejné databáze GPON. Některé procedury využívají pro ukládání dočasných dat dočasné tabulky pojmenované pod označením temporary_table_x, kde x je libovolné celé číslo. Tyto tabulky jsou pravidelně po ukončení SQL procedury odstraněny a jejich odstranění je kontrolováno i před spouštěním procedur. Pokud dočasná tabulka nebyla odstraněna na konci běhu poslední procedury, je odstraněna okamžitě po spuštění procedury aktuální, aby nedošlo k poškození vyfiltrovaného

výstupu nevhodnými daty. Pokud dojde během vykonávání SQL procedury k neočekávanému stavu a následnému pádu, dočasná tabulka v databázi zůstane, ale její přítomnost v žádném případě neovlivní žádné další vykonávané instrukce SQL serverem.

4.1.1 Ukládání dat

Ukládání dat je prvním krokem pro další manipulaci s daty nejen na úrovni SQL serveru. Podle základní syntaxe jazyka SQL jsou data ukládána vkládacím příkazem se syntaxí podobnou *INSERT INTO table_name (sloupec1, sloupec2, ...)VALUES (hodnota1, hodnota2,...)*. Příkaz je složen z povinné a nepovinné části. Do výčtu sloupců a hodnot je povinné uvést všechny hodnoty, které jsou uvedeny jako NOT NULL bez DEFAULT hodnoty. Pokud jsou výchozí hodnoty nastaveny, není takové položky do příkazu uvádět.

Ukládání do databáze je prováděno přes tři tabulky PCBDheader, PCBDheaderPLEND a PCBDheaderBWMAP, z důvodu zamezení výskytu duplicitních sloupců v databázi a oddělení spolu souvisejících datových struktur. Nevýhodou rozdělení je složitější ukládání do databáze. Kvůli provázání tabulek relacemi, jsou všechny tabulky propojeny skrze primární klíč tabulky PCBDheader a proto je nutné uvádět tento dynamicky generovaný klíč i ve zbývajících tabulkách. Problém nastává při vytváření příkazu *INSERT*, protože pro zbývající dvě tabulky se stává povinným parametrem.

Pro ukládání dat do navrženého systému jsou použity procedury se vstupními parametry:

- saveGponHeader
- savePlendRecord
- saveBwMapAllocStructure

Funkční část každé procedury je složena z modifikovaného dotazu insert. Pro první tabulku je výkonný kód zobrazen ve výpisu programového kódu č. 4. Jako vstupní parametry jsou vkládány hodnoty datového typu bajt nebo bit.

Programový kód 4: Výkonná kód procedury saveGponHeader

```
1  CREATE PROCEDURE dbo.saveGponRecord
2       @Psync binary(4),
3       @IdentFEC bit,
4       @IdentRsvd bit,
5       @IdentFrameCounter binary(4),
6       @PloamdOnuId binary(1),
7       @PloamdMessageId binary(1),
8       @PloamdData binary(10),
9       @PloamdCrc binary(1),
10      @Bip binary(1)
11 AS
12 BEGIN
13      INSERT INTO [dbo].[PCBDheader]
14            ([Psync],
15             [IdentFEC],[IdentRsvd],[IdentFrameCounter],
16             [PloamdOnuId],[PloamdMessageId],
17             [PloamdData],[PloamdCrc],[Bip])
18      VALUES
19            (@Psync,
20              @IdentFEC,@IdentRsvd, @IdentFrameCounter,
21              @PloamdOnuId,  @PloamdMessageId, @PloamdData,
22              @PloamdCrc,
23              @Bip)
24 END
```

Zbylé dvě metody ukládající data do tabulek PCBDheaderPLEND a PCBDheaderBWMAP obsahují principiálně stejný kód pouze s obměněnými parametry a s volitelnou vstupní proměnou *@gpon_header_id*. Ačkoliv tento parametr je důležitý pro správné ukládání, jeho doplnění je ošetřeno ve vnitřním kódu procedury, viz programový kód 5. Zadáním specifického parametru je možné uložit do tabulky doplňující, resp. jiné hodnoty a provázat je s původním GPON rámcem.

Programový kód 5: Vnitřní kód funkce returnLastIndex()

```
1  CREATE PROCEDURE [dbo].[returnLastIndex]
2       @table NVARCHAR(max) = 'PCBDheader'
3  AS
4  BEGIN
5       SELECT IDENT_CURRENT (@table) AS lastIndex;
6  END
```

Ve výše uvedeném úryvku kódu je ukázána vnitřní část procedury *returnLastIndex* s volitelným parametrem *@table* pro zadávání názvu tabulky, ve které má být nalezen poslední použitý klíč. Vstupní parametr je datového typu nvarchar a jako výchozí hodnota je nastaven název tabulky PCBDheader.

4.1.2 Analýza provozu

Analýza resp. filtrování zachyceného provozu je zpracovávána procedurou *trafficAnalyzing*. Procedura zpracovává všechna zachycená data a na základě vstupního parametru provádí filtrování a ukládání požadovaného provozu do nové tabulky. Filtrování podle daného vstupního parametru provádí modifikaci filtrovacího příkazu *SELECT INTO*. Doplnění druhého klíčového slova v dotazu způsobuje ukládání získaných dat do dočasné temporary_table_2. Uložená data jsou řazena podle identifikačního čísla záhlaví rámce gpon_id_header a řazena postupně podle zachycení a uložení do databáze.

Programový kód 6: Procedura trafficAnalyzing()

```
CREATE PROCEDURE trafficAnalyzing(@ploam_onuId
binary(1)=NULL)
AS
BEGIN
     DECLARE @newTableName NVARCHAR(30);
     DECLARE @broadcast binary(1) = 0xFF
     IF @ploam_onuId IS NULL
               SET @newTableName = 'trafficAnalyzing_ALL';
     ELSE
          BEGIN
               /*Print ONU-ID in hex*/
               SET @newTableName = 'trafficAnalyzing_'+
master.sys.fn_varbintohexstr(@ploam_onuId);
               /*Print ONU-ID in dec*/  /*
               SET @newTableName = 'trafficAnalyzing_'+
CAST(CAST(@ploam_onuId AS INT) AS VARCHAR(max));
               */
          END
     DROP TABLE IF EXISTS temporary_table_2;
     DECLARE @sql varchar(max);
     SET @sql = 'DROP TABLE IF EXISTS [GPON].[dbo].['+
     @newTableName+']';
     EXEC (@sql);
     IF @ploam_onuId IS NULL
          BEGIN
               SELECT Psync,
                  IdentFEC, IdentRsvd, IdentFrameCounter,
                  PloamdOnuId, PloamdMessageId,
                  PloamdData, PloamdCrc,
                  Bip, gpon_header_id
               INTO temporary_table_2
               FROM PCBDheader
               ORDER BY gpon_header_id ASC;
```

```
END
     ELSE
          BEGIN
               SELECT Psync,
                 IdentFEC, IdentRsvd, IdentFrameCounter,
                 PloamdOnuId, PloamdMessageId, PloamdData,
                 PloamdCrc,
                 Bip, gpon_header_id
               INTO temporary_table_2
               FROM PCBDheader
               WHERE (PloamdOnuId = @ploam_onuId
                     OR
                      PloamdOnuId = @broadcast) AND
gpon_header_id >= dbo.firstMessageOfGpon(@ploam_onuId)
               ORDER BY gpon_header_id ASC;
          END
     EXEC sp_rename 'temporary_table_2', @newTableName;

     RETURN 0;
END
```

Vstupní parametr *@ploam_onuId* definuje způsob zpracování. Jeho výchozí hodnota je nastavena na NULL a pokud není při volání změněna, jsou všechna zachycená data zapsána do nové tabulky. Vstupní hodnota je definována jako binární vstup o velikosti jednoho bajtu a zadáním hodnoty ve tvaru 0xXX, kde X je hexadecimální vyjádření ONU-ID lze definovat, pro jakou ONU jednotku má být analýza komunikace vyfiltrována. Vstupní parametr také ovlivňuje způsob pojmenování výstupní tabulky. Jako prefix tabulky je pro všechny výstupy nastaven řetězec trafficAnalyzing_, zatímco sufix je definován na ALL pro hodnotu NULL nebo po přetypování systémovou funkcí *master.sys.fn_varbinarytohexstr()* na hexadecimální řetězec obsahující číslo jednotky ONU jednotky. Jako alternativní možností je nastavit sufix pomocí přetypování *CAST(CAST(@ploam_onuId AS INT) AS VARCHAR(max))* na celé číslo, ale tato možnost není nastavena.

Nastavením názvu nové tabulky je nutno ověřit absenci nové i dočasné tabulky temporary_table_2. Odstranění tabulky provádí příkaz *DROP TABLE IF EXISTS*

nazev_tabulky. Při testování a mazání tabulky s vygenerovaným názvem dle vstupního parametru nelze provést mazání přímo, ale použit tzv. dynamicky generovaný SQL dotaz. MS SQL nedokáže zpracovat část názvu tabulky uloženého do proměnné, proto je dotaz nejprve celý sestaven jako řetězec, uložen do proměnné *@sql* a následně vykonán příkazem *EXEC(@sql)*.

Po připravení prostředí je vykonáno samotné filtrování. Příkaz *SELECT* pro analýzu celého provozu vytvoří identickou tabulku s tabulkou PCBDheader. Při zadání specifického zařízení je upravena filtrovací podmínka *WHERE*. Jsou vybrány všechny rámce obsahující v poli PloamOnuId danou ONU-ID nebo rámce obsahující broadcastové ONU-ID rovno 255, zadanou v hexadecimálním tvaru 0xFF ale od prvního gpon_header_id od připojení dané jednotky. Vyfiltrování tohoto prvního GPON rámce provádí funkce *firstMessageOfGpon()* se vstupním parametrem *@ploam_onuId*.

Vyfiltrovaná data jsou v posledním kroku přejmenována funkcí *sp_rename* na název tabulky definované v proměnné *@newTableName* a vrácen bechybový návratový kód 0.

4.1.3 Aktivace jednotky ONU

Filtrování aktivačního procesu je složeno z procedury *onuActivatingProcess* a funkce *firstMessageOfGpon()* jejichž kódy jsou obsáhlejší a proto jsou uvedeny v příloze 2 a 3.

Funkce *firstMessageOfGpon()* je základem celého vyhledávání aktivačního procesu. Ve svém těle vykonává sekvenci příkazů, které vyhledávají zprávy používané během aktivačního procesu v databázi a jako návratovou hodnotu vrací *@gpon_frame_id* prvního uloženého GPON rámce obsahujícího některou z aktivačních PLOAM zpráv.

Struktura funkce je podobná jako u procedury analyzující zachycenou komunikaci, kde na základně jednoho volitelného parametru *@ploam_onuId* , se stejnými možnostmi jako bylo uvedeno dříve, je definováno, co má být obsahem vyhledávání. Zavoláním funkce bez parametru je vráceno gpon_header_id prvního řádku z tabulky PCBDheader. Přidáním parametru s číslem ONU jednotky je ověřena návaznost na tabulku _PLOAMdownstream uchovávající názvy PLOAM zpráv s odpovídajícími ID PLOAM zpráv. Formou otestování jsou načteny do proměnných identifikátory zpráv, používaných dále při zpracování – Ranging_Time, Assign_ONU-ID, Extended_Burst_Lenght a Upstream_Overhead. Po úspěšném načtení, skript pokračuje k filtrování. V opačném případě je navrácen záporný chybový kód.

Filtrování začíná otestováním, zdali v zachyceném procesu jsou přítomny zprávy s ONU-ID dané jednotky a zároveň existuje i PLOAM zpráva Ranging_Time. V opačném

případě je vrácen první řádek tabulky a funkce končí. V kladném případě je do proměnné *@filtered* představující tabulku, uložen vzestupně podle gpon_header_id všechen provoz od začátku přenosu po poslední zprávu Ranging_Time adresované dané ONU. V dalším kroku vyhledáváno id předchozí zprávy Assign_ONU-ID ve vyfiltrovaném obsahu tabulky *@filtered* a při nalezení uloženo do proměnné *@tmp*. Následně je ověřeno, jestli bylo id nalezeno, tzn. hodnota *@tmp* není rovna NULL. Po ověření skriptu pokračuje dále vyhledáváním další zprávy nebo se ukončí a vrátí předchozí hodnotu uloženou v *@gpon_id*. Vyhledávání zprávy Assign_ONU-ID závisí na obsahu dat PLOAM zprávy, konkrétně na prvních osmi bitech, které definují číslo ONU-ID jednotky. Protože PLOAM data jsou v tabulce uloženy jako celých deset bajtů, je první bajt získán kombinací volání funkcí *varbinaryToStringOfBits()* převádějící celých deset bajtů PLOAM dat na řetězec jednotlivých bitů a funkcí *stringOfBitsToVarbinary()* převádějící zpět tento řetězec na binární číslo. Obě tyto funkce jsou popsány v kapitole 4.1.6. V těle druhé metody je volána funkce *SUBSTRING*, která vybírá z textového řetězce bitů právě požadovaných prvních 8 bitů dat. Pokud je i tato zpráva nalezena, procedura může přestoupit k posledním kroku a to k vyhledávání zpráv Extended_Burst_Lenght a Upstream_Overhead. První z těchto zpráv je definována jako volitelná, proto nemusí být vždy nalezena. Pokud je nalezena, stejným způsobem je uloženo do dočasné proměnné její id. Podobný postup je i v případě druhé zprávy. Skript je ukončen s navrácením hodnoty obsahující gpon_header_id poslední nalezené zprávy.

Pokud při vyhledávání zpráv některá z hledaných zpráv není nalezena je vždy navrácena hodnota gpon_header_id poslední nalezené zprávy. Na základě takto nalezeného identifikátoru lze pak následně sestavit celý aktivační proces.

4.1.4 Deaktivace jednotky ONU

Jak bylo vysvětleno dříve, jednotka může být deaktivována dvěma způsoby:

- Samovolně – například odpojením zařízení od zdroje napájení což je signalizováno odesláním zprávy Dying_Gasp
- Jednotkou OLT – která provádí odpojení jednotky odesláním zprávy Deactivate_ONU-ID.

Filtrování komunikace při deaktivaci jednotky je zajištěno procedurou *onuDeactivatingProcess* s volitelným parametrem *@ploam_onuId* viz programový kód 7.

Programový kód 7: Procedura onuDeactivatingProcess()

```
CREATE PROCEDURE onuDeactivatingProcess (@ploam_onuId
binary(1) = NULL)
AS
     /* Checking input params */
     IF EXISTS (SELECT [message id] FROM _PLOAMdownstream
     WHERE [message name]='Deactivate_ONU-ID')
          DECLARE @deactivate_onu_id binary(1) = (SELECT
          [message id] FROM _PLOAMdownstream
          WHERE [message name]='Deactivate_ONU-ID');
     ELSE
          BEGIN
               /*PRINT 'Upstream_Overhead not exist in
_PLOAMdownstream';*/
               RETURN -1;
          END
     DECLARE @newTableName NVARCHAR(30);
     IF @ploam_onuId IS NULL
          SET @newTableName ='deactivatingProcessOnu_ALL';
     ELSE
          BEGIN
               /*Print ONU-ID in hex*/
               SET @newTableName =
'deactivatingProcessOnu_'+
master.sys.fn_varbintohexstr(@ploam_onuId);

               /*Print ONU-ID in dec*/  /*
               SET @newTableName =
'deactivatingProcessOnu-'+ CAST(CAST(@ploam_onuId AS INT)
AS VARCHAR(max));
               */
          END
     DROP TABLE IF EXISTS temporary_table_1;
     DECLARE @sql varchar(max);
```

```
34      SET @sql = 'DROP TABLE IF EXISTS [GPON].[dbo].['+
35 @newTableName+']';
36      EXEC (@sql);
37      SELECT PloamdMessageId, PloamdOnuId,PloamdData,
38           gpon_header_id
39      INTO temporary_table_1
40      FROM PCBDheader WHERE (
41           PloamdMessageId=@deactivate_onu_id
42      ) ORDER BY gpon_header_id ASC;
43      IF @ploam_onuId IS NOT NULL
44           BEGIN
45                DELETE FROM temporary_table_1
46                WHERE (
47                     PloamdMessageId = @deactivate_onu_id
48                     AND
49                     PloamdOnuId != @ploam_onuId)
50           END
51      EXEC sp_rename 'temporary_table_1', @newTableName;
52      return 0;
```

Procedura na základě zvoleného parametru provádí filtrování komunikace a zápis do příslušné tabulky, stejně jako tomu bylo u procedury *onuActivatingProcess.*

V prvním kroce po spuštění procedury jsou načteny informace o zprávě Deactivating_ONU-ID z tabulky _PLOAMdownstream. Pokud zpráva není definována, nebo její některé její parametry, procedura končí s navrácením chybového kódu, reprezentovaným záporným celým číslem.

Po načtení hodnot je na základě vstupního parametru nastaveno pojmenování výstupní tabulky, do které budou uložena filtrovaná data. Princip je stejný jako o u procedury *onuActivatingProcess*, ale je použitý jiný prefix tabulky – deactivatingProcessOnu_. Nastavením jména procedura, ověří neexistenci dočasné tabulky temporary_table_1 a tabulky definované názvem uložené v proměnní *@newTableName*. Pokud neexistují, jsou vytvořeny případně staré smazány a vytvořeny nové příkazem *DROP IF EXISTS*. Vytvoření tabulky s názvem uloženém v proměnné je definováno pomocí dynamického SQL.

Samotné uložení dat do dočasné tabulky provádí příkaz *SELECT INTO,* který vyfiltruje a uloží všechny zprávy Deactivate_ONU-ID. Při zvoleném vstupním parametru s číslem konkrétní onu, jsou z této tabulky příkazem *DELETE* odstraněny všechny zprávy, které této jednotce nepatří.

Procedura je dokončena úspěšným přejmenováním tabulky na požadovaný název příkazem *sp_rename* a navrácením chybového kódu, reprezentující úspěšné dokončení vykonané procedury.

4.1.5 Počet zpráv

Pro zobrazení počtu přijatých zpráv je použita funkce *numberOfReceivedMessages()* s volitelným parametrem číslem ONU-ID a návratovou hodnotou reprezentovanou datovým typem bigint.

Výkonné části kódu předchází deklarace výstupního parametru *@result* a definováním broadcastového ONU-ID *@broadcast*, obsahujícího hexadecimální hodnotu 0xFF uloženou jako datový typ binary. Po deklaraci je jako o většiny metod na základě vstupního parametru vykonán blok kódu založen na spojení příkazu *SELECT* a agregační funkce *COUNT()*. S volaným defaultním parametrem funkce provede součet všech řádků tabulky PCBDheader a hodnoty uloží hodnotu do proměnné *@result*. Se specifikací konkrétní jednotky ve vstupním parametru je použit stejný dotaz, ale je doplněna filtrovací podmínka *WHERE*. V podmínce je definováno, že musí být vybrány všechny zprávy od gpon_header_id většího nebo rovnému první přijaté zprávě v celé komunikaci (zjištěno funkcí *firstMessageOfGpon()*) a zároveň musí vybrat všechny zprávy, které obsahují konkrétní ONU-ID jednotky nebo zprávy označené jako broadcastové.

Na závěr je provedeno otestování, jestli jsou některé hodnoty nalezeny. Problém nastává při volání funkce příkazu *SELECT*. Pokud příkaz *SELECT* nenajde shodu s žádným řádkem v tabulce, není navrácena hodnota 0 ale NULL. T-SQL při použití procedur a funkcí musí mít nadefinován i návratový datový typ. I když dokáže navrátit hodnotu NULL, je to v této situaci spíše nežádoucí, protože při dalším zpracování např. python skriptem by bylo nezbytné otestovat, zdali není navrácená hodnota NULL a zaměnit ji za hodnotu nula. Mnohem jednodušší je otestovat tuto možnost přímo v SQL kódu funkce. Pokud není nalezen žádný řádek obsahující definovanou podmínku *WHERE* a proměnná *@result* tak obsahuje hodnotu NULL, je v těle podmínky *IF* nahrazena hodnota NULL za číslo nula. Nakonec je vrácen odpovídající počet řádků přijatých PLOAM zpráv.

Programový kód 8: Výpočet počtu přijatých zpráv

```
1  CREATE FUNCTION numberOfReceivedMessages
2  (ploam_onuId binary(1)=NULL)
3  RETURNS BIGINT
4  AS
5  BEGIN
6       DECLARE @result bigint;
7       DECLARE @broadcast binary(1) = 0xFF
8       IF @ploam_onuId IS NULL
9            /*All catch messages on DOWNSTREAM*/
10           SELECT @result = COUNT(*) FROM PCBDheader;
11      ELSE
12           SELECT @result = COUNT(*) FROM PCBDheader
13           WHERE(
14 gpon_header_id >= dbo.firstMessageOfGpon (@ploam_onuId)
15           AND
16 (PloamdOnuId = @broadcast OR PloamdOnuId = @ploam_onuId)
17 );
18      IF @result IS NULL
19           SET @result = 0
20      RETURN @result;
21 END
```

4.1.6 Převodní funkce

T-SQL obsahuje mnoho definovaných mozností jak převádět data na jiná dat. Typickým příkladem mohou být ve skriptech použité funkce *CAST(@hodnota AS datovy_typ)*, nebo funkce *CONVERT()*. Při vytváření skriptů došlo k problému s extrahováním části PLOAM dat, protože T-SQL neumí pracovat s řetězcem bitů. Během aktivačního procesu je nezbytné přečíst pár bitů z PLOAM dat a následně je převést zpět do datového typu binary. To zajišťují funkce *varbinaryToStringOfBits()* a *stringOfBitsToVarbinary()*.

První funkce slouží k převodu binárně uložené hodnoty danou jako vstupní parametr funkce do řetězce bitů. Kvůli omezení SQL serveru funkce umí převádět datový typ varbinary o maximální délce 2000 bajtů, ale to pro současnou aplikaci je plně dostačující. Po ověření maximální velikosti vstupní proměnné, je tento vstup převeden na řetězec s hexadecimální

hodnotou ve tvaru 0x…, kde za tečkami následují symboly hexadecimální soustavy. Po inicializaci potřebných proměnných je přestoupenu do cyklu *WHILE* s opakováním dokud iterační proměnná *@i* je menší nebo rovna délce řetězce. V okamžiku kdy jsou si tyto dvě hodnoty rovny je vrácen řetězec obsahující jednotlivé bity. Iterování cyklu začíná od hodnoty *@i*=3, protože první dva znaky řetězce jsou prefixem pro označení binární hodnoty. V každém kroku cyklu dochází deklaraci dočasné proměnné *@tmp* a vložení do funkcí *substring* části řetězce *@hex,* mezi indexy *@i a @i+1,* které v tomto případě obsahuje jeden konkrétní znak hexadecimální soustavy. V druhém kroku cyklu je do výstupní proměnné *@string* doplněn převod na jednotlivé bity. Rozhodování v příkazu *SET* zajišťuje příkaz *CASE,* v jehož těle jednotlivé podmínky definují, na základě jaké hodnoty má být do řetězce *@string* doplněn binární výstup. V posledním kroku je o jedničku inkrementována iterační proměnná *@i.* Po splnění podmínky cyklu je jako výstup funkce vrácen výsledek uložený v proměnné *@result.*

Programový kód 9: Převodní funkce varbinaryToStringOfBits()

```
1  CREATE FUNCTION varbinaryToStringOfBits
2  ( @bin varbinary(max) )
3  RETURNS VARCHAR(max)
4  AS
5  BEGIN
6       IF LEN(@bin) > 2000
7            RETURN -5;
8       DECLARE @hex varchar(max)=
9       master.sys.fn_varbintohexstr(@bin);
10      DECLARE @i int = 3;
11      DECLARE @string VARCHAR(max) = '';
12      WHILE @i <= LEN(@hex)
13      BEGIN
14           DECLARE @tmp varchar(1) = SUBSTRING(@hex, @i,
15 @i+1)
16           SET @string = (
17           CASE
18                WHEN @tmp = '0' THEN @string + '0000'
19                WHEN @tmp = '1' THEN @string + '0001'
```

```
20                  WHEN @tmp = '2' THEN @string + '0010'
21                  WHEN @tmp = '3' THEN @string + '0011'
22                  WHEN @tmp = '4' THEN @string + '0100'
23                  WHEN @tmp = '5' THEN @string + '0101'
24                  WHEN @tmp = '6' THEN @string + '0110'
25                  WHEN @tmp = '7' THEN @string + '0111'
26                  WHEN @tmp = '8' THEN @string + '1000'
27                  WHEN @tmp = '9' THEN @string + '1001'
28                  WHEN @tmp = 'a' THEN @string + '1010'
29                  WHEN @tmp = 'b' THEN @string + '1011'
30                  WHEN @tmp = 'c' THEN @string + '1100'
31                  WHEN @tmp = 'd' THEN @string + '1101'
32                  WHEN @tmp = 'e' THEN @string + '1110'
33                  WHEN @tmp = 'f' THEN @string + '1111'
34              END )
35              SET @i = @i+1;
36      END
37      RETURN @string;
38  END
```

Druhá funkce provádí inverzní operaci k první funkci, tj. převod řetězce s vyjádřením bitů zpět na textový řetězec obsahující hodnotu datového typu varbinary. V prvním kroku skriptu je ověřena délka vstupního řetězce. Pokud délka *@inputString* není zbytku dělitelná čtyřmi, je do proměnné *@string* před hodnotu *@inputString* doplněn chybějící počet nulových bitů. V opačném případě, tj. hodnota je dělitelná beze zbytku, jsou tyto dvě hodnoty identické a dále je pracováni pouze s proměnou *@string*. Po verifikaci jsou deklarovány inicializační parametry zejména na začátek výstupní proměnné, kde je doplněn prefix *0x*, označující v T-SQL začátek binární hodnoty. S deklarovanými parametry lze přestoupit ke zpracovávání dat cyklem *WHILE*, který iteruje za stejných podmínek jako u předchozí funkce. Jednotlivé iterující kroky jsou také principiálně stejné, ale vyhledávání probíhá po 4 znacích řetězce. Při převodu mezi binární a hexadecimální soustavou jsou totiž nezbytné pro převod potřeba právě čtyři bity. I ukládání do výstupní proměnné *@result* je založeno na stejném principu s příkazem *CASE*, ale odpovídající čtyři bity jsou v podmínce *WHEN* převedeny na příslušný znak hexadecimální soustavy a doplněny do výsledného řetězce *@result*.

V posledním kroku cyklu je zvýšena inkrementační proměnná o čtyři, což zajišťuje posunutí vytvoření podřetězce také o čtyři znaky v dalším kroku cyklu. Po splnění podmínky nutné pro běh cyklu je cyklus ukončen a jako výstup funkce navrácen textový řetězec s převedenou hodnotou v proměnné *@result*.

Programový kód 10: Funkce stringOfBitsToVarbinary() – ošetření vstupu

```
1  SET @string = (
2  CASE
3       WHEN LEN(@inputString)%4 = 0 THEN @inputString
4       WHEN LEN(@inputString)%4 = 1 THEN '000' + @inputString
5       WHEN LEN(@inputString)%4 = 2 THEN '00' + @inputString
6       WHEN LEN(@inputString)%4 = 3 THEN '0' + @inputString
7  END
```

Programový kód 11: Funkce stringOfBitsToVarbinary() – upravená podmínka

```
1  SET @result = (
2            CASE
3                 WHEN @tmp = '0000' THEN @result + '0'
4                 WHEN @tmp = '0001' THEN @result + '1'
5                 WHEN @tmp = '0010' THEN @result + '2'
6                 WHEN @tmp = '0011' THEN @result + '3'
7                 WHEN @tmp = '0100' THEN @result + '4'
8                 WHEN @tmp = '0101' THEN @result + '5'
9                 WHEN @tmp = '0110' THEN @result + '6'
10                WHEN @tmp = '0111' THEN @result + '7'
11                WHEN @tmp = '1000' THEN @result + '8'
12                WHEN @tmp = '1001' THEN @result + '9'
13                WHEN @tmp = '1010' THEN @result + 'A'
14                WHEN @tmp = '1011' THEN @result + 'B'
15                WHEN @tmp = '1100' THEN @result + 'C'
16                WHEN @tmp = '1101' THEN @result + 'D'
17                WHEN @tmp = '1110' THEN @result + 'E'
18                WHEN @tmp = '1111' THEN @result + 'F'
19           END )
```

4.2 Zpracování dat v Pythonu

Výstupy volaných procedur a funkcí jsou dále zpracovány python skriptem a výstupem ze skriptu zobrazeny koncovému uživateli. Výstupy jsou zobrazeny uživateli dvojím způsobem:

- Textovým – zpracované hodnoty jsou uživateli zobrazeny prostřednictvím textového výstupu z konzole jazyka python nebo
- Tabulkovým – výstupy jsou převedeny do tabulky a uloženy do HTML souboru, který může uživatel otevřít libovolným webových prohlížečem.

Spouštění analýzy je provedeno spuštěním skriptu gpon.py. Pro úspěšné spuštění a správné dokončení skriptu je nutné do systému nainstalovat nejnovější balíky pip, pandas, pyodbc a s nimi i jejich důležité závislosti například přes příkazovou konzoli powershell příkazy:

Programový kód 12: Instalace závislostí z pipu

```
1  .\python.exe -m pip install --upgrade pip
2  .\python.exe -m pip install pandas pyodbc
```

Analyzační skript využívá pro zpracování knihovnu pandas, která obsahuje nástroje a funkce pro analýzu a práci s daty. Výhodou použití knihovny pandas jsou její mocné nástroje pro práci s poli. Další knihovny jsou importovány ze složky lib. Jde především o jednoduché knihovny umožňující skriptu nějakou specializační funkci např. vytvoření připojení k databázovému enginu MS SQL (s podporou balíku pyodbc), zápis do HTML souboru atp. Všechny výkonné funkce jsou deklarovány v hlavní části skriptu. Po spuštění skriptu jsou průběžně generovány výsledky na textový výstup python terminálu a v podobě HTML souborů do složky output.

4.2.1 graphic.py

Mezi jednotlivými kroky analýzy a po spuštění skriptu jsou do textového výstupu vypisovány grafické výstupy z knihovny graphic.py. V této knihovně jsou definovány části, které se opakují a znepřehlednily by zbytečně hlavní kód. Každá funkce je složena ze série příkazů *print()*, jak je uvedeno v ukázce programového kódu 13.

Programový kód 13: Ukázka kódu v grafické knihovně

```
1  def boldLine():
2       print('==========================================
3  ==========================================================
4  ===')
5
6  def gk_gponFrame():
7      print('<--------------- 125 us -------------->')
8      print
9      print('---------------------------------------')
10     print('| PCBd header |         Payload       |')
11     print('---------------------------------------')
12
13 def gk_ident():
14     print('-------------------------------------------')
15     print('| FEC (1b) | RSVD (1b) | FRAME COUNTER (30b)|')
16     print('-------------------------------------------')
```

4.2.2 MsSqlConnector.py

V prvním kroku hlavního skriptu je otevřeno spojení do databázového enginu. To je zajištěno knihovnou lib/MsSqlConnector.py, jejíž koncept vychází z connectoru na databázi uvedeného v kapitole 3.1.6. Upravený databázový connector je zobrazen v ukázce programového kódu 14.

Pod importem balíku pyodbc jsou definovány dvě funkce zajišťující připojení a odpojení od databáze. Funkce *databaseConnection()* má nastaveny v záhlaví volitelné parametry a na začátku kódu i defaultní parametry pro připojení k databázi. Pokud nejsou volitelné parametry nastaveny, nedojde ke změně volitelných parametrů a v dalším kroku dojde k otevření spojení do databáze příkazem *pyodbc.connect()*. Po vytvoření spojení je definován i cursor ukazující na dané spojení, pomocí kterého mohou být následně vykonávány jednotlivé SQL dotazy v databázi. Metoda vrací jako návratový parametr spojení *connection* a ukazatel *cursor*.

Metoda *closeDatabaseConnection()* s povinným parametrem *connection* spojení na databázi ukončuje příkazem *connection.commit()*.

Programový kód 14: Upravený MsSQL connector

```
1  import pyodbc
2  def databaseConnection(ip=None, user=None, password=None):
3      #defaultni nastaveni
4      _IP = "localhost"
5      _USER = "garfield"
6      _PASSWORD = "********"
7      _DATABASE = "GPON"
8
9      if ip is not None:
10         _IP = str(ip)
11     if user is not None:
12         _USER = str(user)
13     if password is not None:
14         _PASSWORD = str(password)
15
16     connection = pyodbc.connect('DRIVER={ODBC Driver 13 for
17 SQL Server};SERVER=' + _IP + ';DATABASE=' + _DATABASE + \
18 ';UID=' + _USER + ';PWD=' + _PASSWORD)
19     cursor = connection.cursor()
20     return connection, cursor
21
22 def closeDatabaseConnection(connection):
23     if connection is not None:
24         connection.commit()
25     else:
26         print("ERR: spojeni nelze uzavrit")
```

4.2.3 htmlCreator.py

O výpis do HTML souboru se stará knihovna lib/htmlCreator.py. Obsahuje funkci *createHtml()* pro vytvoření a zápis do souboru. Má zadefinovány dva povinné a dva volitelné vstupní parametry. Mezi povinné patří *fileName,* definující nastavení jména souboru a *data,* obsahující HTML výstup tabulky z knihovny pandas. Za volitelné parametry jsou parametry

css definující odkaz na CSS styly pro formátování tabulky a *notice*, který je použit při doplnění výstupu z knihovny pandas například statistikami. Po vstupu do funkce je ověřeno, zdali mají být použity výchozí kaskádový styly pro z bootstrap knihovny nebo uživatelsky definované. Po ověření je vytvořen nový soubor v cestě path s názvem souboru z proměnné *fileName*. Nový soubor je otvírán s parametrem ´w´, který umožňuje zápis do soubory se zahozením celého předešlého obsahu. Do otevřeného souboru je pomocí příkazů *file.write()* po řádcích zapsán nový obsah. Po ukončení zápisu je otevřený soubor uzavřen příkazem *file.close()*.

	Message name	Message-ID	Binary message ID
0	Assign_Alloc-ID	10	b'\n'
1	Assign_ONU-ID	3	b'\x03'
2	BER_Inertval	18	b'\x12'
3	Configure_Port-ID	14	b'\x0e'
4	Configure_VP/VC	7	b'\x07'
5	Deactivate_ONU-ID	5	b'\x05'
6	Disable_Serial_Number	6	b'\x06'
7	Encrypted_Port-ID	8	b'\x08'

Obr. 25: Ukázka vygenerovaného přehledu PLOAM zpráv

4.2.4 convert.py

Posledním modulem skriptu je modul lib/covert.py, definující velice jednoduché převodní funkce *byteToInt()* a *byteToStringOfBites()*. Jde o velice jednoduché funkce, které v nejsou v pythonu definovány. První funkce umožňuje převést přijatou hodnotu datového typu byte z databáze a použít ji jako celé číslo. I když python datový typ byte při výpisu zobrazí jako celé číslo, do vytvářeného souboru je vypsán jako posloupnost nesmyslných znaků. Metoda prochází cyklem *for* bit po bitu v celém bajtu a každý bit přetypovává na celé číslo, které přičítá k výsledku *number*.

Druhá funkce *byteToStringOfBytes()* převádí vstup z datového typu byte na celé číslo a to následně pomocí vestavěné funkce *bin* převede na posloupnost bitů.

4.2.5 Analyzační skript gpon.py

Skript gpon.py je spouštěcí skript celé analýzy. V záhlaví skript jsou importovány nezbytné knihovny. Jednotlivé řádky importují knihovny a přiřazují jim zkratky pro použití v kódu.

Pod importovanými knihovnami se nachází jednotlivé funkce specializující se na jednotlivé analýzy. Tyto funkce jsou uvozeny podtržítkem podle podobné analogie jako je tomu u objektově orientovaného programování v pythonu. I když jde pouze o sadů jednoduchých skriptů a objektově orientované programování není použito, znakem podtržítko je míněno, že uvedená funkce se nevyskytuje nikde mimo skript, ve kterém je zapsána. To platí i pro skripty ve složce lib. Pod deklarací a funkčním kódem jednotlivých funkcí je zapsán samotná výkonný kód spouštějící jednotlivý části analýzy. Tento blok kódu je označen komentářem *# MAIN SCRIPT*.

Každá analyzační funkce ve skriptu gpon.py vytváří svůj vlastní výstup do HTML souboru a vrací výsledek analýzy do hlavního bloku, kde je vypsán do terminálového výstupu.

4.2.5.1 Příprava HTML souboru

Ačkoliv data jsou do HTML souboru zapisována funkci z modulu htmlCreator.py, je nezbytné tato data pro výstup do souboru připravit. Tuto přípravu provádí metody *_createHtml()* a *_mark_errors()*.

Metoda *_createHtml()* sice obsahuje tři vstupní parametry, ale svůj výkon potřebuje pouze uvedený dataframe výstup z knihovny pandas označovaný jako df. Jako dataframe analytici používající pandas označují dvourozměrné pole s hodnotami, zpracovávané touto knihovnou. Na tento výstup jsou aplikovány styly umožňující barevně formátovat výslednou podobu. Pro formátování a aplikaci barev lze použít metody *apply()*, která lze aplikovat jen na některé sloupce, nebo metodu *applymap()* aplikující barevné formátování na celé dvourozměrné pole. Jako atribut těchto funkcí je uvedena funkce *_mark_errors()*. Další metodou aplikovanou nad styly je metoda *set_table_attributes()*. Jako vstupní parametr funkce je definován řetězec obsahující HTML tagy, které mají příslušet vygenerované tabulce. V tomto případně je přidán řetězec přiřazující jaké třídy kaskádových stylů mají být na tabulku použity. Po aplikaci uvedených stylů a doplňujících parametrů je vykonána nad dataframem metoda *render()*, která zajistí převod celého dataframu s parametry do podoby HTML kódu. Tento vygenerovaný kód je předám spolu i se zbylými dvěma parametry do htmlCreatoru.

Metoda *_mark_errors()*, jak bylo uvedeno, je volána z dříve vysvětlené metody a představuje filtr provádějící barevné podbarvení tabulky. Jako parametr se předávají jednotlivě buňky pole. V prvním kroku je nastavena výchozí černá barva písma. Další kód je definován pro datový typ řetězec. Cyklus *for* prochází postupně celý řetězec a hledá jeden ze znaků uvedených v množině *chars*. Při nalezení takového znaku je barva buňky zaměněna na červenou a cyklus ukončen. Zvolená barva je vrácena a metodou *applymap()* aplikována na prozkoumanou buňku.

	PSYNC	FEC	Rsvd	FrameCounter	OnuID	Ploam-ID	Ploam message name	PloamData	PloamCRC	BIP	GPON_ID
0	3064672736	1	0	850499514	0	*32	Undefined message!	b'\x01\xffY\xab\xaa\x00\x00\x00 \x83'	96	0	1
1	3064672736	1	0	850499515	0	*32	Undefined message!	b'\x01\xffY\xab\xaa\x00\x00\x00 \x83'	96	0	2
2	3064672736	1	0	850499516	0	*32	Undefined message!	b'\x01\xffY\xab\xaa\x00\x00\x00 \x83'	96	0	3
3	3064672736	1	0	850539434	0	*32	Undefined message!	b'\x01\xffY\xab\xaa\x00\x00\x00 \x83'	96	0	7
4	3064672736	1	0	850539435	0	*32	Undefined message!	b'\x01\xffY\xab\xaa\x00\x00\x00 \x83'	96	0	8
5	3064672736	1	0	850539436	0	*32	Undefined message!	b'\x01\xffY\xab\xaa\x00\x00\x00 \x83'	96	0	9
6	3064672736	1	0	882631484	0	*32	Undefined message!	b'\x01\xffY\xab\xaa\x00\x00\x00 \x83'	96	0	12

Obr. 26: Zobrazení detekce chyb v HTML souboru

4.2.5.2 Zjištění všech jednotek na GPON portu

Ke zjištění všech dostupných jednotek na pasivní optické síti slouží metoda *onuIds()*. Jedná se o metodu bez vstupních parametrů a jako výstup je vráceno pole obsahující jednotlivé ONU.

Metoda na prvním řádku vytvoří prázdné pole *onus* a poté zavolá nad dotaz *SELECT*, který prohledá tabulku PCBDheader a seskupí všechny hodnoty podle ONU-ID jednotky. Vrácenou odpověď uloží do proměnné response a projde ji cyklem *for* pořádcích. V každém řádku zjistí ONU-ID a uloží ji jako pole *onu* jako binární a celé číslo. Celé pole *onu* je pak připojeno do dynamického pole *onus* a po skončení cyklu jsou smazány proměnné *response* a *onu*.

Výsledné pole *onus* je pak předáno knihovně pandas, která jej metodou *DataFrame()* upraví do podoby tabulky a definuje mu přidělené názvy zadáním parametrem *columns*. Celý výstup je uložen do proměnné dataframe, který je vypsán do textové konzole a předán do metody *_createHtml()* za účelem vytvoření HTML souboru s názvem ollOnus.

4.2.5.3 Výpis všech existujících zpráv dle standardu ITU-T G983.4

Řídící informace pro ONU jednotky se nazývají PLOAM zprávy. Přehled všech zpráv definovaných standardem ITU-T G983.4 je uložen v databázi v tabulce _PLOAMdownstream. O jejich výtah z databáze a výpise se stará metoda *_ploamMessagesOverview()*. Metoda neobsahuje žádný vstupní parametr a ihned po spuštění je vytvořen SQL dotaz *SELECT*, který získá obsah těchto zpráv ze serveru a seřadí je podle názvu zprávy s abecedním pořadí. Získaná odpověď je zpracovávána stejně jako u metody *onuIds()* po řádcích, převedena na tabulku knihovnou pandas, vytištěna na obrazovku a zapsána do souboru ploamOverview.html metodou *_createHtml()*. Ukázka HTML výstupu viz obrázek 25.

4.2.5.4 Analýza provozu

K analýze celého nebo části provozu pro konkrétní jednotku ONU je určena metoda *allTraffic()*. Na základě vstupního parametru jsou se podobnými parametry zavolány procedury na SQL serveru, které vytvoří analýzu provozu a uloží ji do nových tabulek. Z těchto tabulek je odpověď získat SQL dotazem *SELECT* a následnou odpověď zpracovat a předat ke zpracování grafickému rozhraní.

Po přijetí odpovědi je provedeno ověření, zdali není odpověď prázdná a případně je skript ukončen s výpisem zprávy informujícím uživatele. V opačném případě skript může začít analýzu zpracovávat. Odpověď od serveru s obdrženými daty je procházena po řádcích cyklem *for*. Na začátku cyklu je vytvořeno prázdné pole, do kterého jsou postupně skládány v jednotlivých krocích části záhlaví GPON rámce z odpovědi serveru. V těle cyklu je také aplikován aparát pro detekci chyb přenášených zpráv. Po přiřazení do proměnné *gponFrame* PLOAM ONU-ID, je prováděno testování, zdali přijatá zpráva odpovídá některé z definovaných zpráv ze standardu ITU-T. Tyto zprávy jsou uloženy v poli *ploamMessages* a byli načteny z tabulky _PLOAMdownstream po startu skriptu. ID zpráva nalezená v této proměnné je připojeno do pole *gponFrame* a hned za ním je připojen i její název. Pokud zpráva v databázi neexistuje, je před hodnotu s identifikátorem PLOAM zprávy vepsán znak ′*′, a připojena hláška o oznamující podezřelou zprávu. Tyto upravené hodnoty jsou později při výpisu do HTML souboru označeny červeně. Při detekci chyby je také inkrementována proměnná *errorCounter*, která počítá počet chybných zpráv. Celý sestavený GPON rámec je následně v posledním kroku uložen do proměnné *gponFrames*, která v sobě uchovává všechny získané rámce z databáze.

Po ukončení běhu cyklu jsou dočasné proměnné *tmp* a *frame* smazány a pole *gponFrames* je předáno funkci *DataFrame* z knihovny pandas, která provede sestavení tabulky s definovanými názvy sloupců a aplikaci stylů na jednotlivé buňky z tabulky. Analyzovaná data knihovnou pandas jsou na konci analýzy vypsána do terminálu a předána metodě *_createHtml(),* která provede vytvoření HTML souboru a na jeho konec zapíše i obsah proměnné *errorCounter,* zobrazujícím uživateli počet chybných GPON rámců.

4.2.5.5 Aktivace a deaktivace jednotky

O analýzu přenosu obsahujícího zprávy z aktivačního procesu nebo při deaktivaci jednotky analyzuje metoda *onuProcess(),* která jako povinný parametr *proces* obsahuje řetězec definující, jakou operaci má provést. Sloučení těchto dvou procesů do jedné funkce je provedeno z důvodů analogie obou dále volaných procedur na SQL serveru. Analogií mezi procedurami *onuActivatingProcess* a *onuDeactivatingProcess* je myšleno jejich použití. Ačkoliv obě metody vrací různá data, jejich výstupy a vstupní parametry mají stejnou strukturu.

Po vstupu do metody jsou přečteny vstupní parametry a na jejich základě je nastaven odpovídající výstup. Podle zvoleného parametru proces je vybrána vhodná metoda a tabulka, ze které se mají data analyzovat. Na základě volitelného parametru *onuId* je zahájena analýza pro specifickou ONU.

S nastavenými vstupními parametry pro SQL volání, metoda předává název tabulky uložený v *tableName* metodě *_formatOutputActDeact().* Unitř této metody jsou získány příkazem *SELECT* data z tabulky jejíž název byl předán z nadřazené metody. Získaná data jsou převedena do výsledné podoby cyklem *for,* který vykonává velice podobné instrukce včetně detekce chyb stejně jako v kapitole 4.2.5.4. Jediným rozdílem je pouze zpracování menšího počtu polí v záhlaví GPON rámce.

4.2.5.6 Počet přijatých zpráv

V některých případech je vhodné znát i počet přenesených, resp. přijatých zpráv konkrétní jednotkou ONU. Získávání počtu zpráv je zařízeno metodou *numberOfReceivedMessages()* s volitelným parametrem *onuID*. Při použití výchozí hodnoty parametru, je zavolána na SQL serveru stejnojmenná procedura vracející počet všech zachycených GPON rámců. Po zadání ONU-ID v hexadecimálním tvaru je navrácen vždy požadovaný výstup.

Výstup z knihovny je renderován s upravenou stylizací knihovnou pandas a předán jako v předchozích případech metodě *_createHtml().* Upravenou stylizací jsou do sloupce

SUM výstupního HTML souboru přidány statistiky, zobrazující statistiky mezi celkovým počtem přijatých zpráv a počtem přijatých zpráv jedinou jednotkou ONU.

	OnuID	SUM
0	ALL onus	44
1	0	21
2	18	23

Obr. 27: Ukázka statistik přijatých zpráv

4.2.5.7 Hlavní část skriptu

Hlavní část skriptu je uvozena komentářem *#MAIN SCRIPT* a skládá se z jednotlivých bloků, spouštějící jednotlivé analýzy. Jejich zakomentováním a odkomentováním lze ovlivňovat výstup analýzy. Ve výchozím nastavení skript provádí analýzu pro všechny jednotky ONU, které detekoval po spuštění metody *onuIds()*. Kromě této analýzy vytváří výstup i pro celkovou komunikaci např. všechny zachycené aktivační procesy atp. Mezi jednotlivými výstupy jsou volány metody z modulu graphic.py, které provádí jednoduché grafické oddělení v terminálu.

Po spuštění skriptu a počátečním navázání spojení s databází je nastavena knihovna pandas pro optimální zobrazení v terminálu příkazem *pd.set_option('display.width', 1024)*, kde celé číslo představuje, kolik znaků může být vypsáno na jeden řádek terminálu. Po tomto nastavení jsou z grafické knihovny vypsány základní údaje o skriptu včetně jednoduchého nastínění struktury GPON rámce a následně je spuštěna analýza. Výstup analýzy do terminálového výstupu je vyobrazen na obrázcích 28 a 29.

```
----------------------------------------------------------------------------------------------------
Analyza celeho provozu: 18
        PSYNC  FEC  Rsvd  FrameCounter  OnuID Ploam-ID  Ploam message name                                  PloamData  PloamCRC  BIP  GPON_ID
0   3064672736    1     0     850500705     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0         4
1   3064672736    1     0     850500706     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0         5
2   3064672736    1     0     850500707     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0         6
3   3064672736    1     0     850540624     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        10
4   3064672736    1     0     850540625     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        11
5   3064672736    1     0     882632675     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        15
6   3064672736    1     0     882632676     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        16
7   3064672736    1     0     882632677     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        17
8   3064672736    1     0     882712514     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        20
9   3064672736    1     0     882752511     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        22
10  3064672736    1     0     624545954     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        23
11  3064672736    1     0     624545955     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        24
12  3064672736    1     0     624545956     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        25
13  3064672736    1     0     624585873     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        29
14  3064672736    1     0     624585874     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        30
15  3064672736    1     0     624585875     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        31
16  3064672736    1     0     624625793     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        32
17  3064672736    1     0     624665713     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        33
18  3064672736    1     0      18292829     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        37
19  3064672736    1     0      18292830     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        38
20  3064672736    1     0      18292831     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        39
21  3064672736    1     0      18412589     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        43
22  3064672736    1     0      18452508     18      *30  Undefined message!  b'\x14\xff\x00\x00\x00\x00\x00\x00\x00'         96    0        44
----------------------------------------------------------------------------------------------------
Pocet chyb detekovanych chyb behem prenosu:23
Pozn: * oznaceni chyby

====================================================================================================
```

Obr. 28:Analýza provozu jednotky v terminálu

```
----------------------------------------------------------------------------------------------------------------------
Deaktivacni process pro ONU: 18
WARNING: Nebyla vracena zadna data
======================================================================================================================
Pocet prijatych zprav:
----------------------------------------------------------------------------------------------------------------------
[['ALL onus', 44], ['0', 21], ['18', 23]]
======================================================================================================================
```

Obr. 29: Výpis chybového výstupu a počtu zpráv do terminálu

ZÁVĚR

Jak je známo, pasivní optické sítě jsou špičkou pro připojení účastníků v poslední míli. V dnešní době jde o dobře se rozšiřující trend, disponující dostatečnými rezervami pro budoucí využití. Vývoj technologie vychází z dřívějších technologií, ale využívá moderní přenosová média – optické kabely.

Pasivní optické sítě nejsou novým pojmem v současných trendech. Pro přenos dat po těchto sítích jsou stanovena pouze doporučení ITU G.989/984/987/989 a neexistují žádná pevně definovaná pravidla pro přenos. Vzhledem k absenci standardu prozatím existují jen velmi drahé komerční nástroje pro monitorování provozu a zjištění skutečné komunikace GPON rámci.

Tato diplomová práce je zaměřena na problematiku zachytávání a ukládání dat z pasivní optické sítě FPGA sondou. Obsahem praktické části bylo seznámení s relačním systémem řízení báze dat Microsoft SQL Server a provést návrh struktury pro ukládání zpracovaných dat z FPGA sondy. Po vytvoření návrhu byla tato struktura implementována jako databáze GPON na MS SQL server 2016. Po navržení a implementaci prvního návrhu databáze se objevily problémy s nedostatkem získávaných dat a jejich uložením z FPGA sondy. Po vytvoření druhého návrhu modelu a jeho implementaci bylo vytvořeno rozhraní na SQL serveru, které umožňuje parseru, zpracovávajícímu data z FPGA sondy, do databáze úspěšně a jednoduše ukládat.

Do navržené a implementované databáze byla parserem uložena data se kterými je prováděna analýza z hlediska správného přenosu a základních dějů odehrávající v komunikaci na GPON síti. Typickým příkladem je aktivace a deaktivace jednotky ONU, analýza komunikace, detekce všech aktivních jednotek na pasivní optické síti atp.

Jako praktický výstup je vytvořen obecný Python connector do databázového enginu MS SQL s využitím knihovny pyodbc, umožňující zápis a čtení z databáze.

Jako hlavní výstup práce je vytvořen skript jazyce Python a rozhraní na MS SQL serveru, umožňující provádět filtrování a analýzu provozu na základě zachycených dat. Výsledný Python skript, s využitím upraveného connectoru na databázi a balíku pandas, provádí analýzu dat. Data jsou získávána a filtrovaná skrze procedury uložené na SQL serveru. Výsledky analýzy jsou zobrazovány do konzolového výstupu programovacího jazyka Python, a také ukládány do formy stylizovaných HTML tabulek s využitím bootstrap kaskádových stylů.

Během práce bylo naráženo na několik nedostatků, mezi něž lze zařadit problém s balíkem pandas. Ten je sice považován za mocný nástroj pro práci s poli v jazyce Python, ale při jeho použití a následné manipulaci s již načtenými daty vždy došlo k automatickému přetypování celého pole na datový typ byte.

Druhým a hlavním nedostatkem práce bylo nedostatečné množství zachycených dat a jejich velikost. Na serveru je vytvořeno kromě rozhraní pro práci s GPON rámci i struktura pro ukládání GEM rámců. Ale vzhledem k povaze dat a odchylce jejich struktury od standardu ITU-T G983.4, parser umožnil dekódovat pouze část záhlaví. Parser v některých případech dekódoval i PLOAM zprávy, které nejsou definovány standardem.

Na základě informací získaných se může, ale nemusí tak jednat o nežádoucí provoz ven z pasivní optické sítě do internetu nebo opačně.

LITERATURA

[1] *About MariaDB* [online]. MariaDB, 2017 [cit. 2017-11-29]. Dostupné z: https://mariadb.org/about/

[2] *Architektura databází* [online]. VSB [cit. 2017-11-29]. Dostupné z: http://books.fs.vsb.cz/MSSQLServer/MSSQL_soubory/SQL_index3.htm

[3] *Architektury databázových systémů* [online]. Praha: J. Lažanský, 2014 [cit. 2017-11-29]. Dostupné z: http://labe.felk.cvut.cz/vyuka/A3B33OSD/Tema-13-ArchitekturyDistribDBMS-OSD-4.pdf

[4] *Database Architecture* [online]. W3schools Online Quality Education [cit. 2017-11-29]. Dostupné z: https://www.w3schools.in/dbms/database-architecture/

[5] *Database Concepts* [online]. Redwood City: Oracle, 2017 [cit. 2017-11-29]. Dostupné z: https://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm

[6] *Database management system (DBMS)* [online]. Newton: Margaret Rouse, 2014 [cit. 2017-11-29]. Dostupné z: http://searchsqlserver.techtarget.com/definition/database-management-system

[7] *Databáze a jazyk SQL* [online]. Brno: Interval.cz, 2000 [cit. 2017-11-29]. Dostupné z: https://www.interval.cz/clanky/databaze-a-jazyk-sql/

[8] *Databáze* [online]. Brno [cit. 2017-11-29]. Dostupné z: http://www.databaze.chytrak.cz/

[9] *Databázové systémy – trocha teorie* [online]. Brno: MUNI [cit. 2017-11-29]. Dostupné z: http://www.ped.muni.cz/wtech/03_studium/cvt4/databaze.pdf

[10] *Fiber-to-the-Antenna (FTTA, Fiber to the Antenna)* [online]. Kanada: EXFO, 2017 [cit. 2017-12-07]. Dostupné z: http://www.exfo.com/glossary/fiber-to-the-antenna

[11] HOOD, Dave a Elmar. TROJER. *Gigabit-capable passive optical networks*. Hoboken: Wiley, 2011.

[12] *InterSystems Caché* [online]. Brno: Martin Holoubek [cit. 2017-11-29]. Dostupné z: http://www.fi.muni.cz/~kripac/PV136/holoubek.pdf

[13] *InterSystems Caché* [online]. San Francisco: Wikipedia, 2017 [cit. 2017-11-29]. Dostupné z: https://en.wikipedia.org/wiki/InterSystems_Cach%C3%A9

[14] *Introduction to MongoDB* [online]. mongoDB, 2017 [cit. 2017-11-29]. Dostupné z: https://docs.mongodb.com/manual/introduction/

[15] ITU-T G.984.1 *Gigabit-capable passive optical networks (G-PON): General characteristics.*[online]. Ženeva, Švýcarsko, 2008 [cit. 2018-05-01]. Dostupné z URL: <https://www.itu.int/rec/T-REC-G.984.1-200803-I/en>.

[16] ITU-T G.984.2 *Gigabit-capable passive optical networks (G-PON): Physical Media Dependent (PMD) layer specification.*[online]. Ženeva, Švýcarsko, 2003 [cit. 2018-05-01]. Dostupné z URL: <https://www.itu.int/rec/T-REC-G.984.1-200803-I/en>.

[17] ITU-T G.984.3 *Gigabit-capable passive optical networks (G-PON): Transmissionconvergence layer specification.*[online]. Ženeva, Švýcarsko, 2014 [cit. 2018-05-01]. Dostupné z URL: <https://www.itu.int/rec/T-REC-G.987.1-201001-I>.

[18] LACKO, Ľuboslav. Mistrovství v SQL Server 2012: [kompletní průvodce databázového experta]. Brno: Computer Press, 2013. ISBN 978-80-2513-773-4.

[19] *Microsoft SQL Server* [online]. Newton: TechTarget, 2017 [cit. 2017-11-29]. Dostupné z: http://searchsqlserver.techtarget.com/definition/SQL-Server

[20] *Microsoft SQL Server* [online]. San Francisco: Wikipedia, 2017 [cit. 2017-11-29]. Dostupné z: https://en.wikipedia.org/wiki/Microsoft_SQL_Server

[21] Microsoft. Data Mining Tutorials (Analysis Services), 2017, [online], Dostupné z: https://docs.microsoft.com/en-us/sql/analysis-services/data-mining-tutorials-analysis-services

[22] *MySQL - Introduction* [online]. Hyderabad: tutorialspoint, 2017 [cit. 2017-11-29]. Dostupné z: https://www.tutorialspoint.com/mysql/mysql-introduction.htm

[23] *MySQL* [online]. Newton: TechTarget, 2013 [cit. 2017-11-29]. Dostupné z: http://searchoracle.techtarget.com/definition/MySQL

[24] *Optické přístupové sítě* [online]. Brno: e-Publi [cit. 2017-11-29]. Dostupné z: https://publi.cz/books/185/11.html

[25] *Oracle Database* [online]. Newton: Wikipedia, 2017 [cit. 2017-11-29]. Dostupné z: https://en.wikipedia.org/wiki/Oracle_Database

[26] *Postup: Zakázat omezení cizí klíč s INSERT a UPDATE prohlášení (databáze Visual nástroje)* [online]. Washington: Microsoft, 2017 [cit. 2017-12-10]. Dostupné z: https://technet.microsoft.com/cs-cz/library/ms175041(v=sql.100).aspx

[27] *Query Language* [online]. Techopedia [cit. 2017-11-29]. Dostupné z: https://www.techopedia.com/definition/3948/query-language

[28] *Relační algebra* [online]. Ostrava: Vysoká Škola Báňská [cit. 2017-11-29]. Dostupné z: http://homen.vsb.cz/~s1i95/ISVDAS/IS/IS_relace.htm

[29] *What comes in between MariaDB now and MySQL 5.6?* [online]. MariaDB, 2012 [cit. 2017-11-29]. Dostupné z: http://mariadb.org/what-comes-in-between-mariadb-now-and-mysql-5-6/

[30] *What is a Database Model* [online]. South Jordan: Lucidchart [cit. 2017-11-29]. Dostupné z: https://www.lucidchart.com/pages/database-diagram/database-models

[31] *What is MongoDB?* [online]. mongoDB, 2017 [cit. 2017-11-29]. Dostupné z: https://www.mongodb.com/what-is-mongodb

[32] *What is MySQL?* [online]. Redwood City: Oracle, 2017 [cit. 2017-11-29]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html

[33] *Základní informace o databázích* [online]. Dostupné z: https://support.office.com/cs-cz/article/Z%C3%A1kladn%C3%AD-informace-o-datab%C3%A1z%C3%ADch-a849ac16-07c7-4a31-9948-3c8c94a7c204

[34] *Základy relačních databází* [online]. Brno: Lenka Hrušková, 2011 [cit. 2017-11-29]. Dostupné z: https://www.sspbrno.cz/~lenka.hruskova/maturita/temata/v4i-2011/12-zaklady-relacnich-databazi.pdf

SEZNAM POUŽITÝCH ZKRATEK

AD	Active Directory
AES	Advanced Encryption Standard
AON	Active Optical Network
API	Application Programming Interface
APON	ATM-base PON
ATM	Asynchronous Transfer Mode
BPON	Broadband PON
CRC	Cyclic Redundand Check
DB	Databáze (database)
FTTA	Fiber To The Antenna
FTTB	Fiber To The Building
FTTC	Fiber To The Curb
FTTN	Fiber To The Node
FTTX	Fiber To The x
GEM	GPON Encapsulation Mode
GPON	Gigabit PON
IBM	International Business Machines Corporation
ITU	International Telecomunication Unit
LAMP	Linux Apache MySQL PHP (někdy Perl, Python)
MS	Microsoft
OLT	Optical Line Termination
ONT	Optical Network Termination
ONU	Optical Network Unit
OOP	Objektově Orientované Programování
PAAS	Platform As Service
PHP	Hypertext PreProcesor
PON	Passive Optical Network
RAM	Random Access Memory
SQL	Structured Query Langurage
SŘBD	Systém řízení báze dat
SSMS	Microsoft SQL Server Management Studio
TDM	Time Division Multiplex
UML	Unified Modeling Language
WDD	Wavelength Division Multiplex

SEZNAM PŘÍLOH

PŘÍLOHA 1 – PYTHON CONNECTOR NA DATABÁZI

```
import pyodbc

#definice parametru (serverName, dbName, UID, pass)
conParam=["localhost" , "Flow", "garfield", "mon21dEli8"]
#-----------------------------------------------------

connection = pyodbc.connect('DRIVER={ODBC Driver 13 for SQL
Server};SERVER='+conParam[0]+';DATABASE='+conParam[1]+';UID
='
+conParam[2]+';PWD='+conParam[3])      #vytvoreni spojeni
cursor = connection.cursor()           #ukazatel na spojeni
print("Cist nebo ukladat do databaze? "
      "(moznosti: c-cist,u-ukladat)")
tmp = input("c/u: ")
if (tmp =="c"):
    cursor.execute("SELECT * FROM
         ["+conParam[1]+"].[dbo].[Flow]")
    i = 0

print("[ID],[SYS_UPTIME],[UNIXS],[FIRST],[LAST],[IP_SOURCE]
,[IP_DEST],[PROTO],[SOURCE_PROTO],[DEST_PROTO],[timestamp]"
)    while True:
        item = cursor.fetchone()
        if not item:
            break
        print(item)
        if i >= 1000:
            break
        i = i+1
    print("Cteni dokonceno")
```

```
else:
#record=[[SYS_UPTIME],[UNIXS],[FIRST],[LAST],[IP_SOURCE],
#[IP_DEST],[PROTO],[SOURCE_PROTO],[DEST_PROTO]]
record=[str(123456),str(12345),str(1234),str(123),"text1",\
"text2","text3","text4","text5"]      #vytvoreni zaznamu

cursor.execute("INSERT INTO [dbo].[Flow](SYS_UPTIME, UNIXS,
FIRST, LAST, IP_SOURCE, IP_DEST, PROTO, SOURCE_PROTO,
DEST_PROTO) VALUES ("\

+record[0]+","+record[1]+","+record[2]+","+record[3]+",'
"+record[4]+"','"+record[5]+"','"+record[6]+"','"+record[7]
+"','"+record[8]+"')")
    print ("Zaznam ulozen")
connection.commit()
print("Spojeni ukonceno")
```

PŘÍLOHA 2 – AKTIVAČNÍ PROCES

```
1  CREATE PROCEDURE onuActivatingProcess(
2  @ploam_onuId binary(1) = NULL)
3  AS
4       IF EXISTS (SELECT [message id] FROM _PLOAMdownstream
5       WHERE [message name]='Upstream_Overhead')
6            DECLARE @upstream_overhead_id binary(1) =
7                 (SELECT [message id] FROM _PLOAMdownstream
8                   WHERE [message name]='Upstream_Overhead');
9       ELSE
10           RETURN -1;
11      IF EXISTS (SELECT [message id] FROM _PLOAMdownstream
12      WHERE [message name]='Extended_Burst_Length')
13           DECLARE @extended_burst_lenght_id binary(1) =
14             (SELECT [message id] FROM _PLOAMdownstream
15              WHERE [message name]='Extended_Burst_Length');
16      ELSE
17           RETURN -2;
18      IF EXISTS (SELECT [message id] FROM _PLOAMdownstream
19      WHERE [message name]='Assign_ONU-ID')
20           DECLARE @assign_onuId_id binary(1) =
21           (SELECT [message id] FROM _PLOAMdownstream
22           WHERE [message name]='Assign_ONU-ID');
23      ELSE
24           RETURN -3;
25      IF EXISTS (SELECT [message id] FROM _PLOAMdownstream
26      WHERE [message name]='Ranging-Time')
27           DECLARE @ranging_time_id binary(1) =
28           (SELECT [message id] FROM _PLOAMdownstream
29           WHERE [message name]='Ranging-Time');
30      ELSE
31           RETURN -4;
32      DECLARE @newTableName NVARCHAR(30);
```

```
33      IF @ploam_onuId IS NULL
34           BEGIN
35           SET @newTableName = 'activatingProcessOnu_ALL';
36      ELSE
37           BEGIN
38                /*Print ONU-ID in hex*/
39                SET @newTableName = 'activatingProcessOnu_'+
40 master.sys.fn_varbintohexstr(@ploam_onuId);
41           END
42      DROP TABLE IF EXISTS temporary_table_0;
43      DECLARE @sql varchar(max);
44      SET @sql = 'DROP TABLE IF EXISTS
45            [GPON].[dbo].['+ @newTableName+']';
46      EXEC (@sql);
47      SELECT PloamdMessageId, PloamdOnuId,PloamdData,
48           gpon_header_id
49      INTO temporary_table_0
50      FROM PCBDheader
51      WHERE (
52           (PloamdMessageId=@upstream_overhead_id) OR
53           (PloamdMessageId=@extended_burst_lenght_id) OR
54           (PloamdMessageId=@assign_onuId_id) OR
55           (PloamdMessageId=@ranging_time_id)
56      ) ORDER BY gpon_header_id ASC;
57      IF @ploam_onuId IS NOT NULL
58           BEGIN
59                DELETE FROM temporary_table_0
60                WHERE (PloamdMessageId=@ranging_time_id
61                     AND PloamdOnuId != @ploam_onuId
62                )
63           END
64      EXEC sp_rename 'temporary_table_0', @newTableName;
65      RETURN 0;
```

PŘÍLOHA 3 – ZJIŠTĚNÍ PRVNÍHO RÁMCE ONU

```
1  CREATE FUNCTION firstMessageOfGpon
2  ( @ploam_onuId binary(1))
3  RETURNS BIGINT
4  AS
5  BEGIN
6       IF @ploam_onuId IS NULL
7            RETURN (SELECT TOP 1 gpon_header_id FROM
8  PCBDheader);
9       DECLARE @broadcast binary(1) = 0xFF
10      IF EXISTS (SELECT [message id] FROM _PLOAMdownstream
11      WHERE [message name]='Ranging-Time')
12           DECLARE @ranging_time_id binary(1) =
13           (SELECT [message id] FROM _PLOAMdownstream
14           WHERE [message name]='Ranging-Time');
15      ELSE
16           RETURN -4;
17      IF EXISTS (SELECT [message id] FROM _PLOAMdownstream
18      WHERE [message name]='Assign_ONU-ID')
19           DECLARE @assign_onuId_id binary(1) =
20           (SELECT [message id] FROM _PLOAMdownstream
21           WHERE [message name]='Assign_ONU-ID');
22      ELSE
23           RETURN -3;
24      IF EXISTS (SELECT [message id] FROM _PLOAMdownstream
25           WHERE [message name]='Extended_Burst_Length')
26           DECLARE @extended_burst_lenght_id binary(1) =
27                (SELECT [message id] FROM _PLOAMdownstream
28           WHERE [message name]='Extended_Burst_Length');
29      ELSE
30           RETURN -2;
31      IF EXISTS (SELECT [message id] FROM _PLOAMdownstream
32      WHERE [message name]='Upstream_Overhead')
```

```
33          DECLARE @upstream_overhead_id binary(1) =
34          (SELECT [message id] FROM _PLOAMdownstream
35          WHERE [message name]='Upstream_Overhead');
36     ELSE
37          RETURN -1;
38     IF (SELECT COUNT(*) FROM PCBDheader
39     WHERE PloamdMessageId=@ranging_time_id AND
40     PloamdOnuId=@ploam_onuId ) > 0
41          BEGIN
42               DECLARE @gpon_id bigint =
43               (SELECT TOP 1 gpon_header_id
44               FROM PCBDheader ORDER BY gpon_header_id
45                    DESC);
46               DECLARE @tmp bigint;
47               DECLARE @filtered TABLE(
48                    gpon_header_id bigint,
49                    PloamdOnuId binary(1),
50                    PloamdMessageId binary(1),
51                    PloamdData binary(10)
52               )
53               INSERT INTO @filtered
54               SELECT gpon_header_id, PloamdOnuId,
55                    PloamdMessageId, PloamdData
56               FROM PCBDheader
57               WHERE (gpon_header_id <= @gpon_id
58                    AND PloamdOnuId=@broadcast)
59               ORDER BY gpon_header_id DESC;
60               SET @tmp = (SELECT TOP 1
61                    filt1.gpon_header_id
62               FROM (SELECT gpon_header_id, PloamdData
63                    FROM @filtered
64                    WHERE PloamdMessageId=@assign_onuId_id
65               )filt1
66               WHERE (dbo.stringOfBitsToVarbinary(
```

```
67                    SUBSTRING(dbo.varbinaryToStringOfBits(
68                         filt1.PloamdData),
69                    )
70               ) = @ploam_onuId
71               ) ORDER BY gpon_header_id DESC)
72               IF @tmp IS NOT NULL
73                    BEGIN
74                         SET @gpon_id = @tmp;
75                         SET @tmp = (SELECT TOP 1
76                         filt2.gpon_header_id FROM (
77                              SELECT gpon_header_id,
78                                   PloamdMessageId,
79                                   PloamdOnuId
80                              FROM @filtered
81                              WHERE
82               PloamdMessageId=@extended_burst_lenght_id
83                         ) filt2
84                         WHERE(
85                         (filt2.gpon_header_id <= @gpon_id)
86                         AND (filt2.PloamdOnuId=@broadcast)
87                         ) ORDER BY gpon_header_id DESC)
88                    END
89               IF @tmp IS NOT NULL
90                    BEGIN
91                         SET @gpon_id = @tmp;
92                         SET @tmp = (SELECT TOP 1
93                         filt3.gpon_header_id FROM (
94                              SELECT gpon_header_id,
95                              PloamdMessageId, PloamdOnuId
96                              FROM @filtered
97                              WHERE
98                    PloamdMessageId=@upstream_overhead_id
99                         ) filt3
100                        WHERE(
```

```
                          (filt3.gpon_header_id <= @gpon_id)
                          AND(filt3.PloamdOnuId=@broadcast)
                          ) ORDER BY gpon_header_id DESC);

                          IF @tmp IS NOT NULL
                               SET @gpon_id = @tmp
                     END

               RETURN @gpon_id;
          END
     IF EXISTS (SELECT gpon_header_id FROM PCBDheader
     WHERE PloamdOnuId = @ploam_onuId)
          BEGIN
               SET @gpon_id = (SELECT TOP 1 gpon_header_id
                               FROM PCBDheader
                               WHERE (
                               (PloamdOnuId=@broadcast)
                               OR
                               (PloamdOnuId=@ploam_onuId)
                              ) ORDER BY gpon_header_id ASC
               );
               RETURN @gpon_id;
          END
     RETURN NULL;
END
```