VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Pavel Rýdl

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

WEBOVÁ APLIKACE PRO MONITORING OPTICKÉ SÍTĚ
WEB APPLICATION TOOL FOR OPTICAL NETWORK MONITORING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Pavel Rýdl**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Martin Holík**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Pavel Rýdl ***ID:*** 165417
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Webová aplikace pro monitoring optické sítě

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a implementujte webový nástroj pro monitorování provozu GPON sítí. Na základě povahy GPON provozu definujte, jaká data by měla být uživateli zobrazena. Pro tato data implementujte vhodnou grafickou podobu do webového nástroje. Navržená aplikace by měla obsahovat i rozhraní pro spuštění a vypnutí zpracovávání dat, nastavení detailů analýzy a filtrování provozu, zobrazení statistik. Dále by měla být přehledná, jednoduchá a zabezpečená proti zneužití. Pro implementaci použijte programovací jazyk Python. K výslednému řešení připojte dokumentaci API rozhraní srovnejte možnosti použití Vašeho řešení i na současné doporučení pasivních optických sítí.

DOPORUČENÁ LITERATURA:

[1] D. Hood and E. Trojer, Gigabit-capable passive optical networks. Hoboken: Wiley, 2011

[2] Pilgrim, M. Ponořme se do Python(u) 3. CZ.NIC, z.s.p.o., 2010, ISBN: 978-80-904248-2-1

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Martin Holík

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

V rámci diplomové práce byla prostudována problematika gigabitových optických sítí a webových technologií vhodných pro implementaci webového nástroje. Na základě architektury systému byla implementována frontendová část v ReactJS a backendová část implementovaná v Tornadu. Data pro analýzu jsou čtena z datového proudu pomocí platformy Kafka. Výstupem je webový nástroj pro monitoring GPON rámců.

KLÍČOVÁ SLOVA

analýza, aplikace, GPON, JavaScript, Kafka, monitoring, Python, ReactJS, Tornado, webový nástroj, WebSocket

ABSTRACT

The problematics of gigabit optical networks as well as web technologies suitable for a web tool implementation were studied within this thesis. An experimental web application for monitoring GPON frames is developed based on the proposed system architecture. The frontend is implemented using ReactJS and the Tornado web framework is used for backend implementation. Data for analysis are read from the stream using the Kafka platform.

KEYWORDS

analysis, application, GPON, JavaScript, Kafka, monitoring, Python, ReactJS, Tornado, web tool, WebSocket

RÝDL, Pavel. *Webová aplikace pro monitoring optické sítě*. Brno, 2030, 69 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Martin Holík

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Webová aplikace pro monitoring optické sítě“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Martinu Holíkovi, za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Optické komunikační systémy jsou dnes široce využívány. S ohledem na rostoucí počet uživatelů internetu a služeb, které využívají, rostou požadavky na rychlost připojení. Také zvyšující se závislost ekonomik a podniků na nepřerušeném datovém připojení vede k vysoké poptávce po optických technologiích. Technologie gigabitové pasivní optické sítě na tyto požadavky reaguje a rozšiřuje hranice použití. Nový standard rychlosti umožňuje neustále doplňovat balíček poskytovaných služeb. Monitorování videa, telemetrie, zabezpečovací systémy ostrahy objektu a další služby jsou pomocí této technologie k dispozici. Firmy spoléhají na stabilitu sítě pro uspokojení potřeb a požadavků svých klientů, popřípadě zaměstnanců. Stabilita sítě je kritická především v odvětvích, jako je například medicína a nebo finanční sektor.

Problémy s výkonem nebo výpadky sítě jsou vysoce nákladné. Uvedeným problémům lze předcházet monitorováním síťového provozu a následnou analýzou dat. V souvislosti s tímto trendem však na trhu neexistuje velké množství open source řešení, která jsou vyhledávaná především malými a středními podniky. Téma této diplomové práce bylo zvoleno na základě osobních zkušeností s vývojem backendové části, zájmu o vývoj frontendové části webové aplikace a komunikačního protokolu, poskytujícího plně duplexní komunikační kanál. Implementace řešení založeného na analýze dat v reálném čase by mohla mít signifikantní pozitivní vliv na možnosti analýzy velkého množství dat. Samotné webové rozhraní aplikace pak není nijak limitováno z hlediska typu zobrazovaných dat.

Tato práce je rozdělena do čtyř kapitol. První kapitola je věnována teorii pasivních optických sítí, a podrobněji se zaměřuje na gigabitovou pasivní optickou síť. Obsahuje rozbor pasivních optických sítí, které předcházely technologii gigabitových optických sítí. Dále jsou zde popsány standardy, základní komponenty a nejčastější topologie, ze kterých gigabitová optická síť vychází. Navazující podkapitoly se zabývají přenosem rámců, jejich zapouzdřením, formáty pro sestupný směr, formátem jednotlivých zpráv a jejich funkcemi. Ve druhé kapitole jsou popsány technologie webových aplikací, způsoby komunikace a výměna dat mezi klientem a serverem. Je proveden souhrn kladů a záporů se zaměřením na komunikaci pomocí protokolu WebSocket. Třetí kapitola diplomové práce obsahuje popis praktické části, zaměřené na návrh a implementaci webové aplikace experimentálního nástroje pro analýzu gigabitové pasivní optické sítě. V podkapitolách je věnována pozornost návrhu grafického uživatelského rozhraní, architektury systému a návrhu formy přenášených dat. Tato diplomová práce je uzavřena kapitolou se souhrnem dosažených výsledků a návrhem na další možná řešení.

1 Pasivní optické sítě

Technologie pasivních optických sítí PON (Passive Optical Network) začala v 80. letech 20. století. Tato technologie využívá optická vlákna a pasivní prvky pro přenos signálů od zdroje ke koncovému zařízení. Aktivní prvky se vyskytují převážně jen na vysílacím a přijímacím zařízení.

Pasivní optické sítě využívají především síťovou strukturu typu bod-multibod. Technologie PON obsahuje velké množství standardů a je neustále aktualizována. Vývoj technologie xPON sahá od asynchronních pasivních optických sítí APON (ATM Passive Optical Network), širokopásmových optických pasivních sítí BPON (Broadband Passive Optical Network) až do pozdějších gigabitových pasivních optických sítí GPON (Gigabit Passive Optical Networks) a ethernetových pasivních optických sítí EPON (Ethernet Passive Optical Network). Všechny byly vyvinuty s různými přenosovými režimy a standardy v různých obdobích [1, 2].

1.1 Typy pasivních optických sítí

Od svého zavedení se technologie PON nadále vyvíjela a vznikaly nové typy sítě PON. Původní standardy pasivní optické sítě, APON a BPON, postupně ustoupily výhodám šířky pásma a celkového výkonu novějších verzí [1].

Asynchronní pasivní optické sítě (APON)

APON představuje první návrh pro paketovou komunikaci pomocí režimu asynchronního přenosu APON. Využívá centralizované a statistické multiplexování ATM (Asynchronous Transfer Mode) v kombinaci s efektem sdílení pasivních rozdělovačů na optických vláknech a terminálech optických linek [1].

Širokopásmové optické pasivní sítě (BPON)

Díky rychlému vývoji ethernetové technologie se APON již nepoužívá. Jako nástupce APON technologie byl navržen koncept širokopásmové pasivní optické sítě BPON. Je založen na protokolu ATM, s rychlostmi pro vzestupný a sestupný směr 155 Mbps a 622 Mbps. Přidává dynamické přidělování šířky pásma a ochranné funkce. Může poskytovat služby jako je ethernetový přístup, přenos videa a vysokorychlostní pronajaté linky [1].

Ethernetové pasivní optické sítě (EPON)

Vysoké náklady na realizaci optické sítě BPON vedly k dalšímu vývoji, z něhož vzešla optická sít EPON. EPON představuje pasivní optickou síť. Je založen na technologii

PON sítě Ethernet a kombinuje výhody těchto technologií, poskytuje různé služby přes Ethernet. Struktura sítě je bod-multibod [1].

Gigabitové pasivní optické sítě (GPON)

Jedná se o standard, který je definován sérií doporučení a je schválený mezinárodní organizací telekomunikační unií ITU-T G.984 (G.984.1 až G.984.7) podporující přenosové rychlosti vyšší než 1 Gbit/s. GPON standard vychází z dříve vydaných standardů APON a BPON) [3, 4].

Standardy G.984.x

Standardy GPON jsou založeny na předchozích specifikacích BPON. Specifikace jsou:

- G.984.1 - Dokument popisuje obecné charakteristiky pasivní optické sítě s gigabitovým připojením [5].
- G.984.2 - Dokument popisuje specifikaci vrstvy závislé na gigabitových pasivních optických sítích fyzických médií [6].
- G.984.3 - Dokument popisuje specifikaci vrstvy konvergence pasivního optického síťového přenosu s gigabitovým přenosem [4].
- G.984.4 - Dokument popisuje specifikaci OMCI (Passive Optical Network ONT Management and Control Interface) pasivní optické sítě [7].
- G.984.5 - Dokument definuje rozsahy vlnových délek, které mají být překryty pomocí multiplexování s vlnovou délkou, technologií WDM (Wavelength Division Multiplexing) v pasivních optických sítí PON. Pro maximalizaci hodnoty optických distribučních sítí (ODN) [8].
- G.984.6 - Dokument popisuje prodloužení dosahu [9].

1.2 Základní komponenty sítě GPON

Síť GPON se skládá se základních komponent, kterými jsou. Optické linkové zakončení OLT (Optical Line Termination), optická síťová jednotka ONU (Optical Network Unit) nebo optické síťové zakončení ONT (Optical Network Termination) a Splitter [3].

Optické linkové zakončení OLT

Jedná se o optické linkové zakončení, které se nachází na straně poskytovatele. Mezi hlavní funkce OLT patří překlad formátu zprávy ze síťového rozhraní interního rámce GPON. Správa jednotlivých ONU, včetně jejich registrace. Úpravy úrovní výkonu

a dalších parametrů komunikace např. řízení přístupu k mediím, včetně výpočtu zpoždění přenosu, přidělení šířky pásma, řízení rámců vzestupným a sestupným směrem [3].

ONU/ONT

Síťový prvek propojující zařízení pro přístup koncových uživatelů s optickou distribuční sítí. ONT se někdy nazývá optická síťová jednotka (ONU). V doporučení ITU-T G.984.3 považuje ONT za zvláštní případ ONU (pro jednoho uživatele). Z hlediska funkčnosti TC vrstvy jsou tyto dvě entity identické. Hlavní funkce ONU je převod signálů z optických vláken na elektrický signál. Základním signálem je Ethernet pro data a téměř každý z nich obsahuje jeden nebo více Ethernetových portů. Může mít také analogový signál pro hlasové porty pro starou telefonní službu (POTS) a jeden nebo více RF portů pro video. Další funkce zahrnují překlad formátu zpráv z uživatelského síťového rozhraní do rozhraní GPON interního rámce, řízení rámce v vzestupném a sestupném směru. ONU je napájeno napájecím zdrojem a může obsahovat i napájecí baterii [3, 10].

Splitter

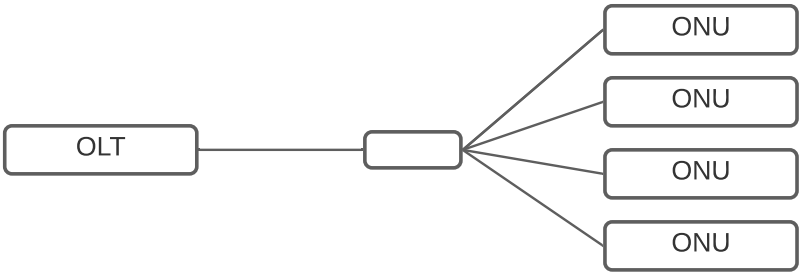
Splitter je pasivní zařízení, jehož hlavní funkcí je rozdělit optický výkon na N samostatných výkonů. Optické cesty se takto mohou rozdělit mezi 1 až 128 různých cest, obvykle (1:32). Z optického rozdělovače vede ke každému uživateli vlákno s jedním režimem tzv. (SM-siglemod). Vložení rozbočovače do sítě způsobí zvýšení vložného útlumu na trase. Běžný útlum pro splitter s rozdělením (1:32) je 17 dB. Útlumu je jedním ze základních parametrů limitujících vzdálenost mezi ONU a OLT [4, 11].

1.3 Topologie sítě GPON

GPON je postavena především na optické síti s jedním vláknem a topologií bod-multibod, známou také pod zkratkou P2MP (Point-To-Multipoint), je ve vztahu jedno OLT ke mnoha ONU. Na tuto topologii lze nahlížet jako na topologii stromu. Kořenovým prvkem je OLT a určitý počet ONU je k němu připojeno jako jeho následovníci. Vlákno je z OLT vedeno do Splitteru, který je umístěn v blízkosti odběratelů. Optická energie z OLT je rozdělena v odbočovacích bodech Splitterech. Toto rozdělení přiděluje každé větvi stejný podíl výkonu. Ze Splitteru je pak ke koncovému uživateli vedeno jedno vlákno, které končí v optické síťové jednotce ONU [3].

Stromová topologie

Stromová topologie uvedena na obrázku 1.1 patří k nejčastěji používaným přístupovým topologiím využívaných v sítích GPON. Používá jedno vlákno z OLT které je vedeno do mezilehlého bodu štěpení. Z tohoto bodu je pak rozděleno do více vláken a každé vlákno pak vede do samostatné jednoty ONU. Hlavní výhoda této topologie

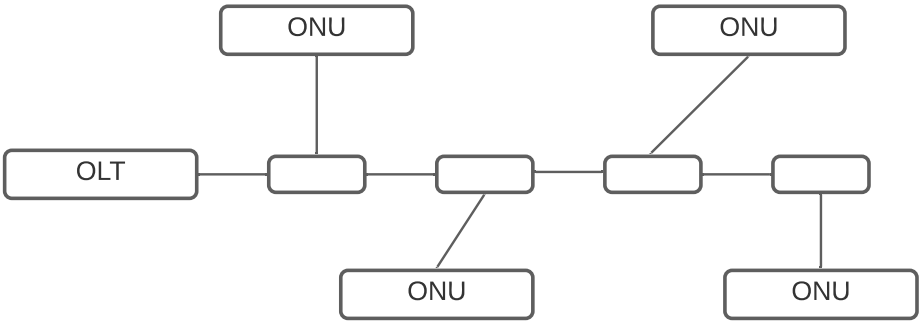


Obr. 1.1: Ukázka stromové topologie [10].

spočívají v soustředění rozdělení na jeden bod, čímž je snadné zjistit problémy se sítí. Další výhodou je, že všechny ONU obdrží přibližně stejnou kvalitu optického signálu [10].

Sběrnicová topologie

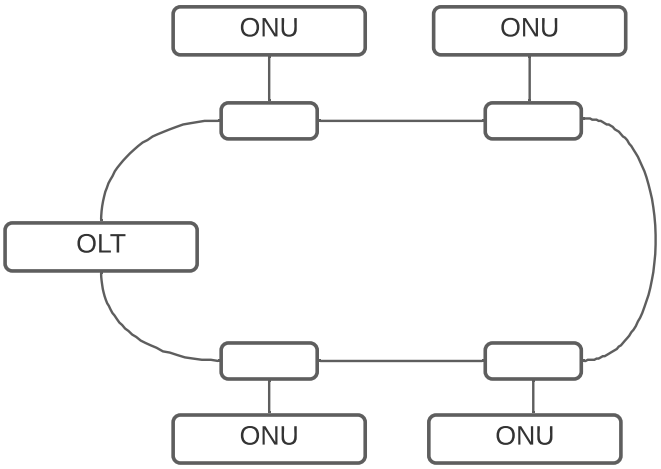
Sběrnicová topologie 1.2 používá jedno vlákno podobně jako stromová topologie. Jedná se však o horší případ, při kterém vznikají problémy s využitou kapacitou. Každý koncový uživatel je k této topologii připojen pomocí odbočující spojky, která extrahuje malou část energie vysílanou z OLT. Výhoda této topologie spočívá v použití vždy minimálního množství vlákna (pokud jsou ONU připojeny rovnou k odbočce). Což umožňuje jednodušší nasazení při připojování nové jednotky ONU. Hlavní problémy této sítě spočívají v tom, že při každém průchodu signálu odbočkou je tento signál utlumen. Proto ONU, které jsou umístěné ve větší vzdálenosti od OLT, mají slabší signál [10].



Obr. 1.2: Ukázka sběrnicové topologie [10].

Kruhová topologie

Kruhová topologie 1.3 je používána převážně v metropolitních sítích, protože nabízí schopnost odolnosti v případě poruchy jednoho z vláken. Vyžaduje však použití dvou vláken pro OLT a tím i složitější zařízení na každém ONU. Toto zařízení musí mít možnost přepínání, aby bylo možné odesílat a přijímat signály přenášené ze dvou směrů prstencového vlákna. Tento typ zapojení vykazuje stejné problémy



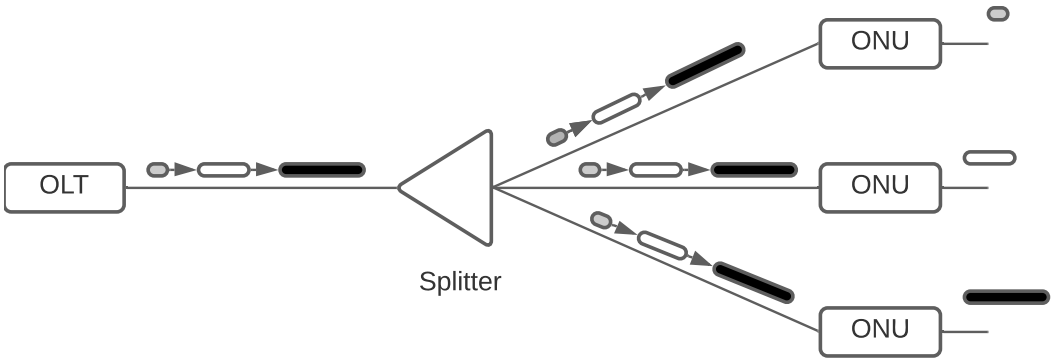
Obr. 1.3: Ukázka kruhové topologie [10].

jako je tomu u topologie sběrnicové, zejména se jedná o energetickou náročnost. Při průchodu každou ONU je optický signál degradován a utlumen. Tento faktor je

nejvíce omezující z hlediska přenosových schopností a omezení počtu připojených ONU ke kruhové topologii [10].

1.4 Přesnos rámců v GPON

GPON používá dva mechanismy přenosu dat. V sestupném směru od OLT k uživatelům jsou datové pakety přenášeny všesměrovým vysíláním pomocí multiplexu s časovým dělením (Time Division Multiplex – TDM), kde časový interval odpovídá 125 µs. Datové pakety tedy nemají definovanou fixní bitovou velikost. OLT vysílá data pro všechny ONU, která jsou k němu připojeny. Každá ONU selektuje data určená právě jemu, tak jak je uvedeno na obrázku 1.4. K zabránění odposlechu se používá šifrováním symetrickým algoritmem (AES – Advanced Encryption Standard). Pro vzestupný směr od uživatelů k OLT, jsou datové pakety přenášeny způsobem



Obr. 1.4: Přenos rámců s časovým dělením.

TDMA (Time-division multiple access). Tento provoz, například pro přenos Ethernet lze popsat podle následujících kroků. Při potřebě přenést data ze servisní sítě provozovatele do uživatelské sítě prostřednictvím GPON dochází k následnému přenosu.

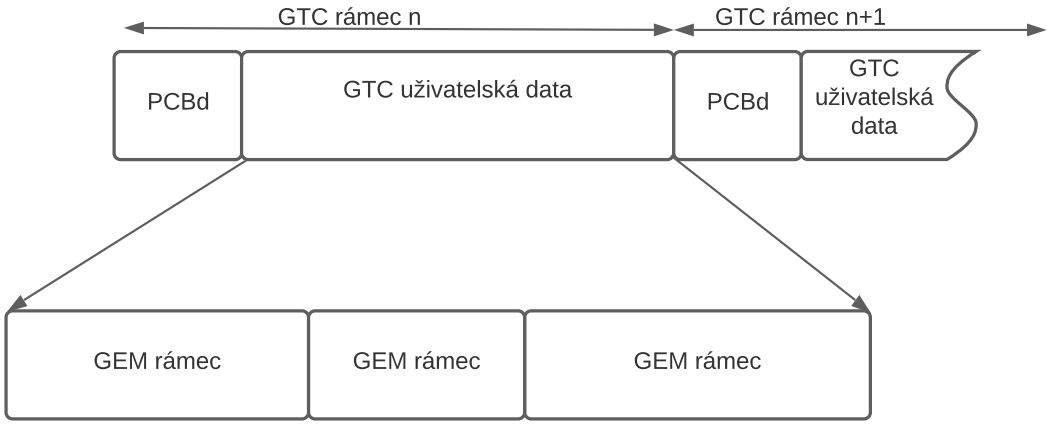
1. OLT přijímající Ethernetový provoz ze servisní sítě poskytovatele kontroluje cílovou MAC adresu.
2. OLT zkontroluje vyhledávající tabulku, aby se podle adresy MAC získalo související ID portu. Další nezbytná pole jsou zahrnuta v záhlaví rámce GEM.
3. OLT takto zabalí několik rámců dohromady.
4. OLT za tento rámec připojí záhlaví PCBd, která souvisí s řídící zprávou, s mapou předělení šířky pásma a dalších polí pro ovládání rámce. Tyto všechny informace tvoří rámec GTC, který vysílá všesměrově do všech ONU jednotek.

5. GTC rámec přijme jenom konkrétní ONU, pro kterou je určen.
6. ONU jednotka zanalyzuje PCBd pole zkontroluje správnost přečte PLOAM informaci a vzestupnou šířku alokovaného pásma.

1.5 Zapouzdření GEM

Gigabitové pasivní optické sítě využívají metodu zapouzdření, tzv. metodu GEM (Gigabit Encapsulation Method), díky které dokážou zapouzdřit různé datové typy uživatelských datových rámců na fragmenty proměnné velikosti. GEM obecně obsahuje rámec záhlaví, část přenášených dat (payload) a někdy i patičku (trailer). Zapouzdření GEM rámců slouží dvěma funkcím jedná se to multiplexování GEM portů a fragmentace dat uživatelského obsahu. Podporuje přenos různých druhů rámců jako SDUs (service data units), někdy nazývané jako rámce někdy jako pakety, zejména pokud se jedná o Ethernet. Každý rámec SDU je zapouzdřen v jednom nebo více GEM rámcích pro přenos po PON síti.

Formát rámce GTC v sestupném směru se zapouzdřením GEM rámců je zobrazen na obrázku 1.5.

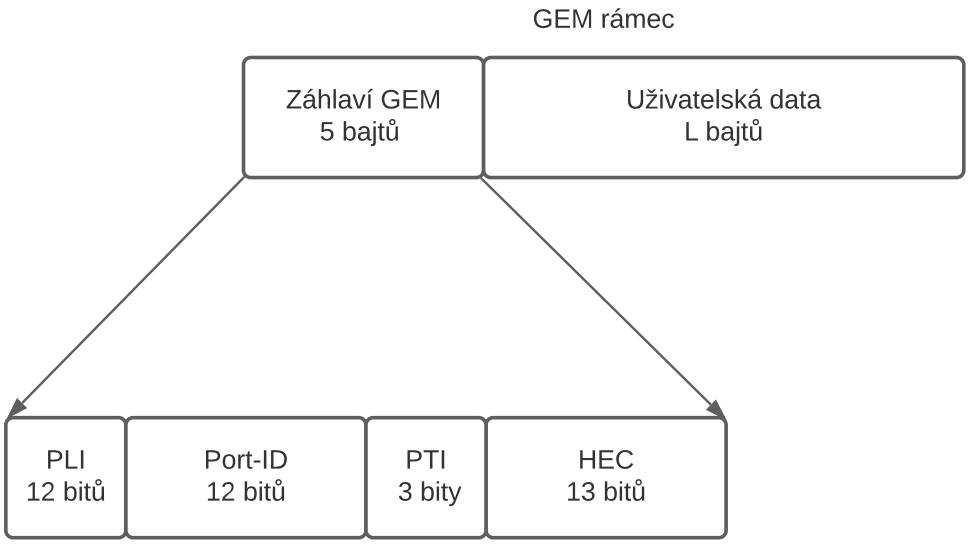


Obr. 1.5: Formát GTC v sestupném směru se zapouzdřením GEM rámců [4].

Nečinné rámce GEM jsou definovány pro příležitosti, kdy k přenosu není k dispozici žádné jiné užitečné zatížení. Souvislá sekvence pro sestupné GEM rámce, spolu s jeho režií, se nazývá rámec konvergenční vrstvy přenosu G-PON (GTC), které se opakují v intervalech 125 μs [1, 3, 4].

Hlavička GEM rámce

Hlavička GEM rámce obsahuje čtyři pole, která jsou zobrazena na obrázku 1.6 a jsou popsána v textu níže.



Obr. 1.6: Formát hlavičky GEM a struktura rámce [4].

- **PLI** - 12bitový indikátor délky datového obsahu PLI (Payload Length Indicator) v bajtech, Maximální délku může být tedy 4095 bajtů. Jakákoli delší struktura delší než 4095 bajtů musí být fragmentována na menší části.
- **PORT-ID** - 12bitové ID portu umožňuje přiřadit 4096 možných portů pro multiplexování provozu.
- **PTI** - 3bitový indikátor datového obsahu PTI (Payload Type Indicator) označuje, zda datový obsah obsahuje rámce uživatelských dat nebo rámce GEM OAM. Rovněž označuje, že se jedná o poslední fragmentem.
- **HEC** - 13bitové pole řízení chyb záhlaví HEC (Header Error Control) slouží k zabezpečení přenášeného záhlaví rámce GEM. Jedná se o speciální kód typu BCH (39, 12, 2) kombinovaný s bitovou paritou [4].

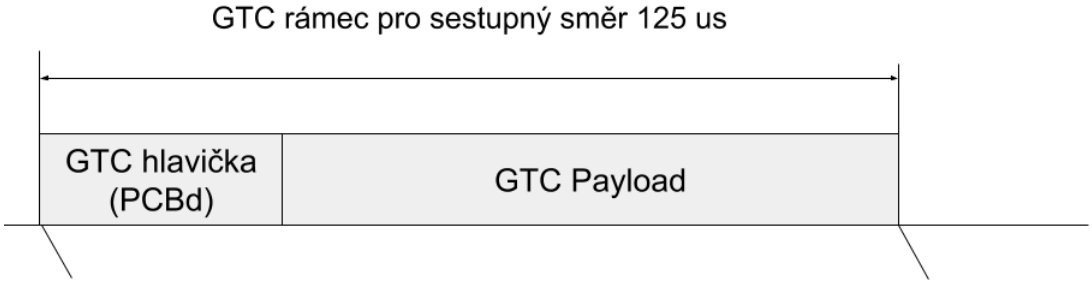
1.6 Formát GPON rámce v sestupném směru

Stahovací GTC rámec trvá 125 μs a je dlouhý 38880 bajtů, což odpovídá rychlosti stahování 2, 48832 Gbps nebo 19440 bajtů při přenosové rychlosti 1,24416 Gbps. Struktura rámce GTC se skládá z fyzického řídícího bloku pro sestupný směr hlavičky

PCBd (Physical Control Block downstream) a GTC datového obsahu, viz obrázek 1.7.

Délka záhlaví rámce PCBd je vždy shodná pro obě přenosové rychlosti. Rozsah délek PCBd závisí na počtu alokačních struktur na rámec. PCBd obsahuje informace o řízení přístupu k mediím v šířce pásma pro vzestupný směr BW mapa.

Záhlaví se skládá z pevné části a proměnné části. Jako pevnou část obsahuje pole pro fyzickou synchronizaci, pole Ident a pole PLOAM. Tato pole jsou chráněna jedním bajtem CRC. Čtyřbajtová nešifrovaná fyzická synchronizace označuje začátek sestupného rámce. Pole Ident označuje, zda se používá FEC. Kromě toho je také implementován 30bitový zalamovací čítač tzv. superframe, který lze použít k zajištění synchronizace s nízkou rychlostí referenčního signálu. Pole PLOAM se využívá pro komunikaci s ONU. Funkce PLOAM zahrnují registraci a zrušení registrace ONU, rozsah, vyrovnávání výkonu, kryptografické aktualizace klíčů, zprávy o chybách fyzické vrstvy atd. Podrobnosti o poli typu PLOAM bude pojednáno v textu níže [4, 1].

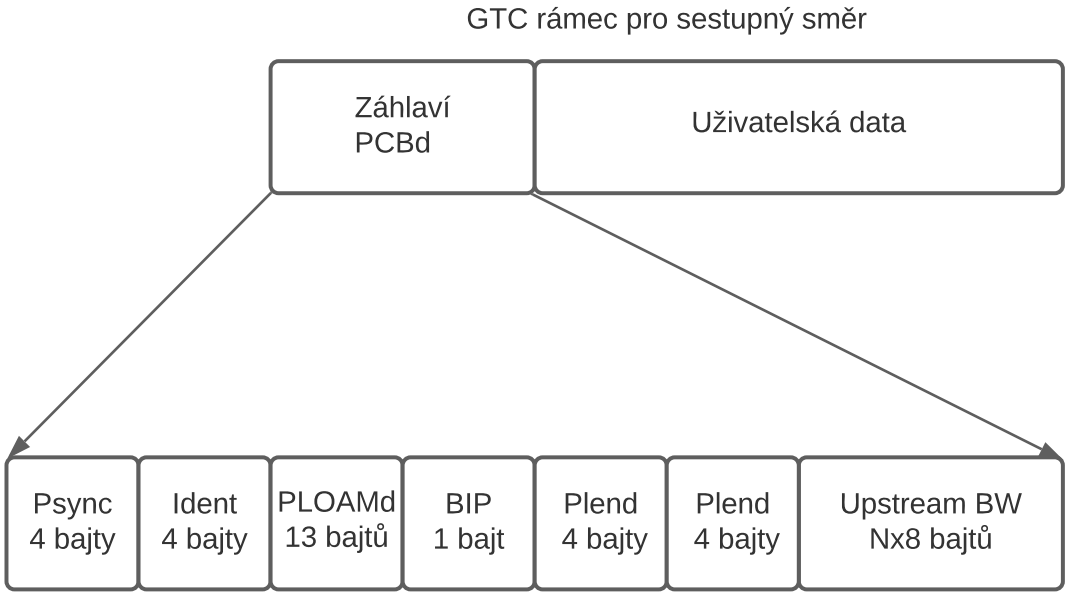


Obr. 1.7: Formát GTC v sestupném směru [4].

Variabilní část záhlaví PCBd obsahuje duplikovanou délku užitečného zatížení pro sestupný směr (PLend) deskriptor, který určuje délku pro vzestupnou komunikaci. Mapa šířky pásma a mapa oddílu ATM v T-CONT, která určuje přidělení šířky pásma pro vzestupný směr proudu s přístupovými položkami. Každá položka má 8 bajtů a obsahuje Alloc-ID T-CONT (Transmission Container), začátek času a čas ukončení přenosu T-CONT pro vzestupný směr a 12bitový příznak označující, jak má být alokace použita. Zbytek rámce obsahuje datový obsah [1].

Záhlaví GTC rámce PCBd

Hlavička GTC rámce PCBd obsahuje několik polí. Ta jsou popsána v textu níže a mají následují formát viz obrázek 1.8.



Obr. 1.8: Formát záhlaví rámce GTC (PCBd) [4].

- **Psync** - Synchronizační pole Psync (Physical Synchronization) je definované jako 4 bajtová posloupnost, která koncové jednotce ONU/ONT slouží pro správnou detekci počátku rámce a odvození rámcové synchronizace.
- **Ident** - Identifikační pole (Ident field) obsahuje první bit pro identifikaci FEC (Forward Error Correction) kódování, druhý bit je rezervovaný pro pozdější použití. Zbylých 30 bitů slouží pro číslování rámců v sestupném směru a při odeslání se automaticky inkrementuje. Při vyčerpání rozsahu se čítač nuluje a rámce se počítají opět od počátku.
- **PLOAMd** – Zprávy PLOAM jsou dlouhé 13 bajtů, obsahující řídící a pomocné zprávy. Každý rámec v sestupném směru obsahuje tuto zprávu. Pokud OLT nemá zprávu, kterou by poslal, posílá zprávu typu *no_message* viz dále v textu.
- **BIP** - Pole BIP (Bit Interleaved Parity) slouží pro indikaci chyb v PCBd záhlaví o velikosti 8 bajtů. Obsahuje bitové prokládání paritním bitem. Tyto bity mohou sloužit k monitorování výkonu.

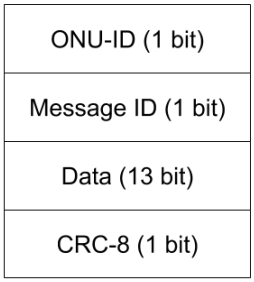
- **PLend** - Pole PLend (Payload Length downstream) o velikosti 4 bajty. Obsahuje informaci o délce posledního pole záhlaví, pole s informacemi o přidělených vysílacích kapacitách ve vzestupném směru a rovněž udává délku ATM (Asynchronous Transfer Mode) sekce v části uživatelských dat. Toto pole je v záhlaví odesláno dvakrát po sobě. Aby byl zajištěn jeho bezchybný příjem, obsahuje navíc zabezpečení vlastního obsahu kódem CRC (Cyclic Redundand Check).
- **Upstream BWmap** - Obsahuje vlastní informace o přidělené vysílací kapacitě vysílané v vzestupném směru. Jeho velikost je dána počtem jednotek připojených v síti. Obecně se jedná o Nx8 bajtů. Těchto 8 bajtů pro každou jednotku obsahuje identifikaci jednotky ONU/ONT, identifikaci přiděleného vzestupného rámce T-CONT, řídící zprávy (informace o nastavení vysílacích úrovní, informaci o způsobu kódování FEC, nastavené schéma pro odesílání požadavků), dále vlastní interval určený pro vysílání a bajt zabezpečení CRC [3, 4].

1.7 Fyzická vrstva řízení a údržby

Fyzická vrstva řízení a údržby PLOAM (Physical Layer Operation Maitenance). Slouží jako kanál pro správu, který je založený na zprávách mezi zakončením optické linky (OLT) a jednotkami optické sítě (ONU), které podporují funkce správy PON TC vrstvy, včetně aktivace ONU, správy a kontroly ONU, založení kanálu (OMCC), konfigurace šifrování, správa klíčů a signalizace alarmů [4].

Formát zprávy PLOAM se přenáší do 13 bajtového pole zprávy PLOAM v režijní sekci následného GTC rámce a výchozí přidělené Alloc-ID. Formát zprávy PLOAM je znázorněn na obrázku 1.9. Struktura zprávy PLOAM se skládá z polí:

- **ONU-ID** - Pole sloužící k adresaci konkrétní ONU. Během protokolu je ONU přiřazeno číslo z rozsahu 0 až 253. Pro vysílání do všech ONU je nastaveno toto pole na hodnotu 0xFF. Potřebný čas pro zpracování všech zpráv v sestupném směru je do 750 ms, což je potřebný čas proto aby ONU mohla zpracovat zprávy, které obdržela a připravila se na odpověď.
- **Message-ID** - Označuje typ zprávy viz v textu níže.
- **Message data** - Tyto oktety se na používají data zpráv GTC.
- **CRC** - Pole slouží pro kontrolu rámce. Zpráva je zahozena pokud je CRC nesprávný [3, 4].



Obr. 1.9: Formát PLOAM zprávy [4].

Význam jednotlivých kontrolních PLOAM zpráv pro sestupný směr

Ve standardu G.984 pro G-PON, OLT odesílá zprávy třikrát kvůli zlepšení šance na úspěšné doručení. ONU generuje událost po přijetí alespoň jedné platné zprávy. Platná zpráva se vyznačuje platným polem CRC.

- **Upstream_Overhead** - Tato zpráva instruuje ONU, ke kterému je předem přiřazené zpoždění a počet bajtů preambule které je použit pro vzestupný směr. ONU optický výkon je definovaný. Tato událost je spuštěna vždy když začíná aktivační proces. Výsledek je poté že ONU nastaví předem přidělené zpoždění.
- **Assign_ONU-ID** - Používá se pro propojení volného onu ID čísla se sériovým číslem. Událost se spouští když OLT najde sériové číslo unikátního ONU. Výsledkem je, že ONU s daným sériovým číslem nastaví ONU-ID a nastaví výchozí Alloc-ID. ID zprávy je 0x14.
- **Ranging_Time** - Slouží k indikaci hodnoty, která je vyjádřena počtem vzestupných bytů. Konkrétní ONU musí vyplnit tuto hodnotu pro registrování vyrovnávacího zpoždění. Událost nastává Když OLT vyhodnotí, že údaj o zpoždění musí být aktualizován. ONU poté vyplní vyrovnávací zpoždění a zaregistruje tuto hodnotu. ID zprávy je 0x04.
- **Deactivate_ONU-ID** - Jedná se o zprávu, která má poskytnou informaci ONU s ONU-ID aby, přestala vysílat zprávy a resetovala se. Tato zpráva může být vyslána i všesměrově.
- **Disable_Serial_Number** - Slouží pro povolení nebo zakázání ONU s daným sériovým číslem.
- **Encrypted_Port-ID** - Zpráva slouží pro nastavení šifrování přímo pomocí GEM portu. Hodnota GEM portu obsahuje celkem dva bajty (12bitů a poté

následují 4 nuly).

- **Request_Password** - Zpráva slouží pro žádost hesla z ONU za účelem ověření. OLT obsahuje lokální tabulku s namapovanými hesly k připojeným ONU. ID zprávy je 0x09.
- **Assign_Alloc-ID** - Pomocí instrukce ONU zjistí stav obsazení konkrétního ID. ID zprávy je 0x04.
- **No_message** - Žádná zpráva není k dispozici při přenosu PLOAM pole. Nastává při prázdné frontě. Struktura rámce G-PON je definována tak, že OLT odesílá pokaždé zprávu PLOAM. ID zprávy je 0x14.
- **POPUP** - Zpráva slouží pro znovu uvedení ONU do provozního stavu, pokud dojde ke ztrátě signálu nebo synchronizace. ID zprávy je 0x0C.
- **Request_Key** - Požadavek od OLT posílá požadavek ONU, aby vygeneroval nové hesla. OLT obsahuje lokální tabulku, kde jsou namapována hesla k připojeným ONU.
- **Configure Port-ID** - Zpráva, která umožňuje pomocí kódů změnit port OMCC GEM. Porty GEM jsou konfigurovány pouze prostřednictvím OMCI. ID zprávy je 0x09.
- **Physical_Equipment_Error (PEE)** - Výstražná zpráva pro fyzickou poruchu zařízení. ID zprávy je 0x0F.
- **Change_Power_Level** - Méně využívaná zpráva. Jednotky ONU vždy vysílají svůj plný jmenovitý výkon. ID zprávy je 0x10.
- **PST message** - Slouží pro přepínání ochrany zprávy. ID zprávy je 0x11.
- **BER** - Definuje akumulační interval, který je vyjádřen počtem sestupných rámců pro následné sestupné bitové chyby.
- **Key_Switching_Time** - OLT indikuje, kdy ONU začíná používat nový šifrovací klíč. ID zprávy je 0x13.
- **Extended_Burst_Length** - Zpráva je vyslána s počátkem aktivačního procesu a slouží, pro poskytnutí informace ONU s číslem typu 3 bajty preambule k použití pro vzestupný režimu. ID zprávy je 0x14 [4].

Rozhraní pro správu a ovládání ONU – OMCI

Rozhraní správy a ovládání ONU zpráv je využíváno pro objevování ONU, řízení a kontrolu. Tento speciální druh zpráv jsou zasílány přes dedikované GEM porty stanoveným mezi OLT a ONU. OMCI protokol povoluje OLT následující.

- Stanovení a zrušení spojení s ONT.
- Správu UNIs na jednotce ONT.
- Požádat o konfigurační informace a statistiky výkonu.
- Samostatně upozorňovat na události, například selhání spojení [4].

2 Technologie webových aplikací

Web známý jako WWW (World Wide Web). Vznikl poprvé v roce 1989, kdy slavný vědec a inženýr Tim Berners-Lee přišel s účinným mechanismem sdílení zdrojů mezi vědci z celého světa. Webové prohlížeče, často nazývané pouze prohlížeče, umožňují zobrazit všechny zdroje, které jsou součástí WWW. Jsou založeny na architektuře klient-server. Klientem je prohlížeč a server si lze představit jako kombinaci softwaru a hardwaru, který přijímá požadavky klienta a poté klientovi pošle požadovaný prostředek. Prohlížeč odešle požadavek na server při zadání adresy nazývané URL (Uniform Resource Locator), poté načte a zobrazí vše, o co uživatel požádal. Klientská část nazývaná frontend označuje všechny části webu, které může uživatel vidět na své obrazovce a komunikovat s nimi a servrová část nazývaná backend odkazuje na pravý opak toho. Zahrnuje skryté mechanismy, které vytvářejí funkci webové stránky. Typický uživatel obecně neví, co se děje na backendu. Webové aplikace jsou vyvíjeny pomocí sady pravidel, technik a nástrojů používaných v procesu komunikace mezi různými typy zařízení přes internet. Proces vytváření webových stránek je založen na řadě kroků, kterými jsou například.

- úvodní analýza,
- vytvoření časového plánu a rozpočtu projektu,
- návrhu architektury,
- návrhu grafického designu webové prezentace,
- vlastní kódování webových stránek,
- uvedení webových stránek do provozu.

2.1 Tornado

Jedná se o ucelený soubor tématicky zaměřených knihoven pro implementaci neblokujícího webového serveru implementovaný v programovacím jazyku Python [1], který podporuje asynchronní síťové knihovny. Tornado [2] je založen na architektuře webového serveru a byl vyvinut s ohledem na extrémně vysoký výkon se zaměřením na vyřešení tzv. problému C10K viz dále v textu 2.1. Poprvé byl vytvořen firmou Facebook [3] v rámci projektu FriendFeed. Neblokující architektura webového serveru Tornado umožňuje škálování desítek tisíc současně otevřených připojení, což je ideální pro dlouhé dotazování, webové sokety a další aplikace, které vyžadují dlouhodobé připojení jednotlivých uživatelů. Proto je Tornado vhodným nástrojem pro webové aplikace pracující v reálném čase [12, 13].

[1] `https://www.python.org/`

[2] `https://www.tornadoweb.org/en/stable/`

[3] `https://www.facebook.com/`

Aplikace Tornado jsou vytvářeny s využitím Model-View-Controller (MVC) architektury. Každá aplikace poskytuje jednoduché mapování adres URL nebo vzorů adres URL na konkrétní třídy. Tyto třídy mohou definovat metody pro zpracování požadavků známé pod zkratkou CRUD (Create, Read, Update, Delete), které využívají klasické HTTP metody GET, POST, PUT, PATCH, DELETE, na adresu URL. Pokud klient odešle požadavek na server, webový rámec Tornado se podívá na požadovaný URL a určí, která třída by měla na požadavek reagovat. Potom volá metody třídy get() nebo post(), v závislosti na metodě požadavku HTTP [14].

Tornado lze rozdělit do čtyř hlavních komponent a to do:

- webového rámce (včetně třídy RequestHandler, který je podtřídou pro vytváření webových aplikací, a různé podpůrné třídy),
- implementace HTTP na straně klienta a serveru (HTTPServer a AsyncHTTPClient),
- asynchronní síťové knihovna zahrnující třídy IOLoop a IOStream, které slouží jako stavební bloky pro komponenty HTTP a lze je použít k implementaci dalších protokolů,
- Knihovna coroutine (tornado.gen), která umožňuje psát asynchronní kód přímějším způsobem než řetězení zpětných volání [14].

Tornado WebSocket modul

Tornado poskytuje třídu *WebSocketHandler* jako součást web socketového modulu. Třída poskytuje události a metody pro komunikace s připojeným klientem. Metoda *open* je volána při otevření nového připojení a metody *on_message* a *on_close* jsou volány, když připojení přijme novou zprávu nebo je zavřeno klientem. Metoda *on_message* je vyvolána pokaždé, když obslužná rutina obdrží novou zprávu od klienta. Třída *WebSocketHandler* navíc poskytuje metodu textitwrite_message pro odesílání zpráv klientovi a metodu *close* pro ukončení připojení [14].

Objekt Application

Je objekt zodpovědný za globální konfiguraci, včetně směrovací tabulky. Umožňuje mapovat žádosti do manipulátorů tzv. handlers. Směrovací tabulka někdy nazývaná routovací tabulka je seznam tzv. URLSpec objektů (nebo jen seznamů), každý obsahuje regulární výraz a handler třídu. Na pořadí hodnot v seznamu záleží, protože se provede první položka seznamu, která bude odpovídat pravidlům a použije se [14].

Problém C10K

Jedná se o studii zkoumající poskytování služeb a optimalizací síťových socketů, které by umožnily souběžné zpracování velkého počtu připojení. Řádově až deseti tisíc připojeních klientů nebo HTTP požadavků na webový server. Název C10K je odvozen z číselného názvu (ten thousand connections) [12].

Běžné servery založené na vláknech, jako je například Apache [4], udržují vlákna operačního systému pro příchozí připojení. Apache přiřazuje každému HTTP připojení jedno vlákno a vytvoří nové, dokud nejsou všechna existující vlákna vyčerpána (v závislosti na systému). Aktualizace v reálném čase, jako například aktualizace dat nebo změny a upozornění vyžadují, aby klient udržoval otevřené připojení. Architektura Apache se však při zatížení nepředvídatelně mění a udržuje velké skupiny otevřených připojení, z nichž každá čeká na svá data. Tímto způsobem však může dojít k rychlému vyčerpání paměti daného serveru [12].

Asynchronní servery jsou navrženy tak, aby zmírňovaly dopady omezení webových serverů založených na vláknech. Servery jako Node.js [5] a Tornado řeší tento problém pomocí kooperativního multitaskingu. To znamená, že asynchronní server přebírá kontrolu nad nevyřízenými požadavky a pokud je aktuální požadavek, čeká na data z jiného zdroje např. (databázový dotaz nebo požadavek HTTP). Běžný vzor, který asynchronní servery používají k obnovení pozastavené operace je vyvolání zpětného volání, když jsou příslušná data připravena [12].

Tornado se liší od většiny webových frameworků napsaných v jazyku Python tím, že není založen na tzv. WSGI (Web Server Gateway Interface), které je obvykle spuštěné pouze s jedním vláknem na proces. Obecně většina frameworků vyžaduje pro svůj produkční běh použití serveru, jako například Apache nebo NGINX [6]. V Tornadu je tento server integrovaný. Tornado však disponuje určitou podporou WSGI. V Tornadu je integrovaná standardní modul asyncio a sdílí stejnou smyčku událostí [12].

2.2 ReactJS

ReactJS [7] je populární JavaScriptová knihovna s otevřeným softwarem (open-source) používaná k vytváření uživatelských rozhraní. ReactJS byl vytvořen firmou Facebook pro řešení některých problémů spojených s rozsáhlými webovými stránkami založenými na zpracování velkého množství dat. Konvence React jsou zcela unikátní.

[4] `https://httpd.apache.org/`
[5] `https://nodejs.org/en/`
[6] `https://www.nginx.com/`
[7] `https://reactjs.org/docs/hooks-effect.html`

Základní vlastností je vytváření tzv. komponent. Komponenta přestavuje různé znovupoužitelné bloky HTML elementů se zapouzdřenou funkcionalitou. Z takto sestavených komponent pak vzniká komplexní grafické uživatelské rozhraní GUI (Graphical User Interface) aplikace [15].

Vytvoření jednoduché webové aplikace zahrnuje implementaci HTML, CSS (Cascading Style Sheets) a JavaScriptový kód. Tři různé technologie jsou použity za účelem oddělení logiky. Jedna osoba může strukturovat obsah pomocí HTML a stylizoval jej pomocí CSS a další osoba může implementovat dynamické chování různých prvků pomocí JavaScriptu. Webová aplikace většinou není považována za kolekci jednotlivých webových stránek . Taková webová aplikace s jedinou webovou stránkou se nazývá jednostránková aplikace (SPA – Single Page Application. Jakmile jsou načteny webové stránky ve webovém prohlížeči, vytvoří se objektový model dokumentu (DOM – Document Object Model). DOM představuje webovou stránku ve stromové struktuře a odráží strukturu rozložení, které bylo vytvořeno pouze pomocí značek HTML. To se děje bez ohledu na to, zda se vytváří tradiční webová stránka nebo SPA. Pokud se jedná o tradiční webovou stránku, je nutné dokončit vytváření rozložení webové stránky. Na druhou stranu, pokud se implementuje SPA, je potřeba začít vytvářet další prvky manipulací s DOM pomocí JavaScriptu. Webový prohlížeč poskytuje JavaScript DOM API [8] [16].

Jednostránková aplikace

Jednostránková aplikace SPA (Single Page Application) je webová stránka, která dynamicky interaguje s webovým prohlížečem a přepisuje aktuální webové stránky s daty získanými z webového serveru. Uživatelské rozhraní UI (User Interface) nabízí velice dobrou uživatelskou zkušenost UE (User Experience) napodobováním přirozeného prostředí prohlížeče a eliminováním čekací doby a opětovného načítání stránek. SPA jsou ve většině případů rychlejší, protože všechny zdroje webové stránky jsou v aplikaci načteny pouze jednou a tak jediným přenášeným obsahem jsou data. Při poslání požadavků ukládá efektivně všechna data do lokální cache paměti, a proto je může využít i v offline stavu. Zjednodušuje a zefektivňuje vývojové aktivity, protože eliminuje potřebu psát kód pro vykreslování stránek na serveru. SPA je obtížné a složité optimalizovat pro SEO (Search Engine Optimization), protože jeho obsah je načítán pomocí AJAX (asynchroní JavaScript a XML). SPA vyžaduje aby uživatelé měli povolený JavaScript pro správné načítání aplikací a akcí. Není tedy vhodný pro případy, kdy může být na straně uživatele deaktivován. I když je uživatelská zkušenost se SPA za běhu rychlá, prvotní stahování může být pomalejší a lze

[8] `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model`

ji také zpomalit pokud dojde k úniku paměti v JavaScriptu. Není proto vhodný pro aplikace s velkým obsahem dat.

Mezi nejznámější SPA webové aplikace patří například, Gmail [9], Mapy Google [10], Facebook [11], GitHub [12] atd [17].

Vícestránková aplikace

Více stránková aplikace MPA (Multi-Page Applications) fungují tradičním způsobem, které znovu načtou celou stránku a zobrazí novou při interakci uživatele s webovou aplikací.

Výhody vícestránkových aplikací:

- ve vyhledávači fungují dobře,
- poskytují uživateli vizuální mapu webové aplikace.

Mezi nevýhody vícestránkových aplikací patří:

- velice obtížný vývoj,

Uživatelské rozhraní vícestránkové aplikace tvoří větší části aplikace s velkým množstvím obsahu, takže uživatelská zkušenost je ve srovnání s jednostránkovými aplikacemi omezená. Mezi vícestránkové aplikace můžeme zařadit například eBay [13] a Amazon [14] [17].

2.3 Přenos dat mezi klientem a serverem

Stávající techniky sloužící k oboustranné komunikaci mezi servery a klienty. Web byl původně vytvořen pro zobrazování textových dokumentů. V dnešní době je ale využíván s řadou dalších funkcí jako zobrazování multimediálního obsahu, funkce umístění(polohy) a to obnáší přenášet zcela odlišný druh dat [18].

Webové stránky byly postaveny na modelu HTTP (Hypertext Transfer Protocol). HTTP je bezstavový protokol, což znamená, že komunikace mezi dvěma stranami se skládá z nezávislé dvojice požadavku a odpovědi. Přesněji klient odešle požadavek na server a server vrátí odpověď nazpět klientu, tímto klient obdrží informace od serveru. Nevýhody tohoto typu komunikace jsou, že posílají záhlaví HTTP, čímž

[9] `https://gmail.com/`
[10] `https://www.google.com/maps/`
[11] `https://www.facebook.com/`
[12] `http://github.com/`
[13] `https://www.ebay.com/`
[14] `https://www.amazon.com/`

se zvětšuje celková velikost souboru, typ komunikace je poloduplexní, což znamená, že každá strana musí počkat na dokončení akce toho druhého, webový server tak spotřebová více zdrojů. V dnešní době mají uživatelé telefony v kapsách s plnohodnotnými webovými prohlížeči, které v některých případech nabízejí více než desktopové prohlížeče. Mimo jiné senzory GPS (Global Positioning System) a dotykové rozhraní [18].

V minulosti byla interakce mezi uživatelem a webovou aplikací velmi jednoduchá. Když chtěl uživatel, aby jeho prohlížeč načetl data nasměroval jej pomocí URL adresy, aby získala obsah pro prohlížeč. Pokud by chtěl napsat blogový příspěvek musel by si prohlížeč načíst formulář a tlačítkem odeslat. Pokud chtěl tento komentář vidět musel udělat stejné kroky (tedy načíst celou stránku). Tato funkce se již změnila Web již nemůže čekat až uživatelé přejdou na správnou adresu URL. Webová stránka musí uživatele kontaktovat, ať je kdekoliv. Paradigma tohoto modelu se změnila kde byl web ve středu interakce. Nyní všechny interakce začínají a končí u uživatele, ať už se jedná o návštěvy webu nebo zasílání krátkých zpráv (SMS) [13].

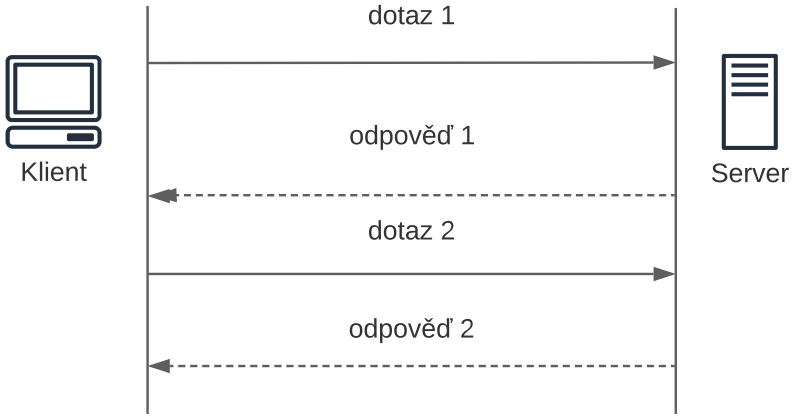
Zprávy typu HTTP

Je běžně používaný mechanismus výměny dat mezi serverem a klientem viz obrázek 2.1. Existují dva typy zpráv. Požadavky zaslané klientem ke spuštění akce na serveru, zpráva typu žádost (request) a následnou odpovědí ze serveru, zpráva typu odpověď (responce). Tento proces je řízen vždy interakcí ze strany klienta, jako je například kliknutí na tlačítko na webové stránce. Historicky docházelo k obnově celé stránky avšak s příchodem AJAXu [15] bylo umožněno načítat dynamicky jen některé části webové stránky. Zprávy HTTP se skládají z textových informací kódovaných v ASCII (American Standard Code for Information Interchange) [19].

Dotazování

Dotazování známé jako polling je synchronní metoda, která provádí periodické požadavky, bez ohledu na to, zda existují data pro přenos. Klient zasílá po sobě jdoucí žádosti po stanoveném časovém intervalu. Sever na tyto požadavky pokaždé odpoví a to bez ohledu na to, zda se data změnila či nikoliv. Tato metoda je však velice náročná na zdroje pro moderní webové aplikace. Chování způsobuje zbytečné dotazování na server, otevírání a následné ukončování spojení [18, 19].

[15] `https://api.jquery.com/jquery.ajax/`



Obr. 2.1: Cyklus HTTP dotaz/odpověď.

Dlouhé dotazování

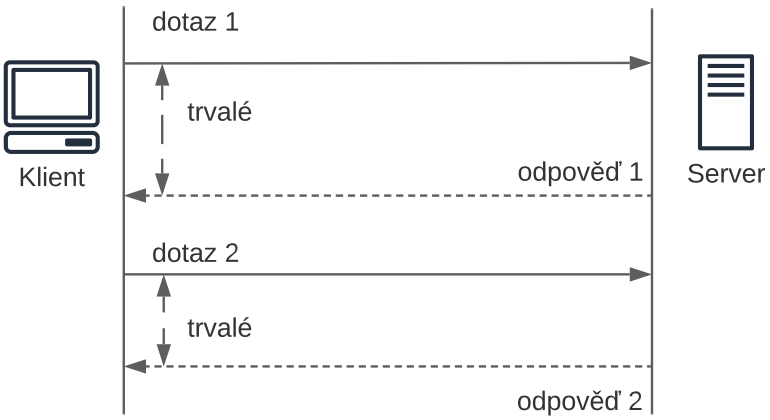
Dlouhé dotazování (pong polling) viz obrázek 2.2 jedná se o podobnou techniku jako předchozí metoda polling. Klient otevře připojení a server udržuje připojení aktivní dokud nebudou načtena data nebo nedojde k vypršení časového limitu. Klient pak může začít znovu a provést sekvenční požadavek. Dlouhé dotazování je tak zlepšením výkonu oproti dotazování, ale neustálé dotazování může zpomalit chod serveru [18, 20].

Streamování

Streamování se zdálo jako nejlepší volba pro přenos dat v reálném čase. Při použití streamování, klient provede požadavek a server udržuje připojení otevřené na dobu neurčitou nebo na nastavený časový interval. Data jsou průběžně aktualizována. Jedná se o skvělé řešení pro nepředvídatelná (reálná) data ale server nikdy nesignalizuje dokončení své odpovědi HTTP. I když se jedná o velké zlepšení streamování stále obsahuje záhlaví HTTP, které zvětšují velikost dat a způsobují zbytečné zpoždění .Všechny problémy se kterými se potýkají staré metody vedly k řešení, kterými jsou webové sokety [18, 20, 19].

Webové sokety

Webové sokety jsou plně duplexní formou komunikací mezi klientem (webovým prohlížečem) a serverem. Tento druh komunikace je trvalý, což znamená, že server musí



Obr. 2.2: Cyklus dlouhého dotazování..

zvládnout velké množství otevřených spojení najednou. Na straně serveru je potřeba postupovat opatrně aby některá připojení nevyužívala stejné zdroje, které poté následně mohou být omezeny. Toto se týče například pokud webový server otevře příliš mnoho připojení k databázi, která je limitovaná počtem připojení.Webové sokety jsou součástí HTML5 standardu. Nejnovější verze všech prohlížečů podporují sokety. Webové sokety komunikují pomocí TCP vrstvy. Připojení je navázáno přes HTTP a jedná v podstatě o mechanismus potřesení rukou (hanshake) mezi klientem a serverem. Po následném handshake, je připojení povýšeno na TCP viz obrázek 2.3.

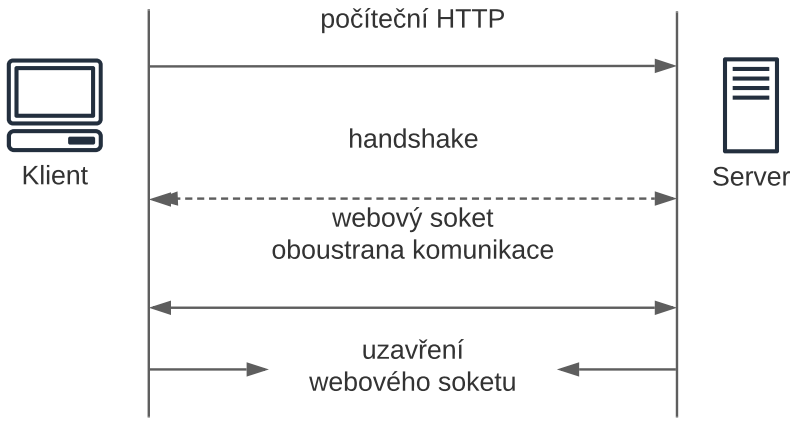
Tak jako protokol HTTP vyžaduje vlastní schéma url adresy tak i soketový protokol má svoje vlastní schéma url. Jednou změnou je předpona ws (nebo wss pro zabezpečené připojení SSL) zbytek URL adresy je podobný obyčejné adresy pro HTTP[18].

Další metody komunikace

Mezi další méně známé metody komunikace řadíme následující typy.

Server-Sent Event - SSE narozdíl od webových soketů nepoužívá obousměrné připojení. Jakmile klient naváže spojení se serverem, server automaticky odesílá data klientovy. Tato komunikace vyžaduje připojení s dlouhou životností.

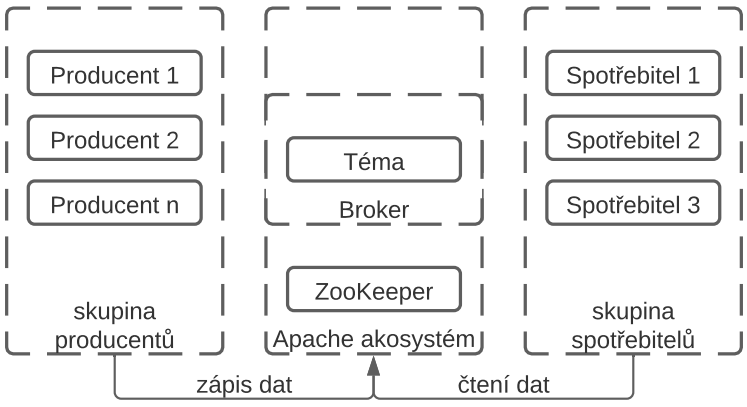
WebRTC na rozdíl od webových soketů a SSE je komunikace mezi prohlížeči a serverem pomocí implementace protokolu, která umožňuje webovým aplikacím přenášet video, audio a datové toky mezi klientem a serverem [21].



Obr. 2.3: Cyklus webového soketu.

2.4 Kafka

Implementace s využitím Kafky umožňuje spravovat streamování dat mezi aplikacemi v reálném čase ze zdrojů událostí, jako jsou databáze, senzory, mobilní zařízení, cloudové služby a softwarové aplikace.



Obr. 2.4: Architektura technologie Kafka.

Na obrázku 2.4 je zobrazena základní architektura technologie Kafka kombinující tři klíčové funkce:

- Posílání a příjímání dat do streamového proudu,
- trvale a spolehlivě ukládat streamy událostí,
- zpracovávat proudy událostí v reálném čase nebo i zpětně.

Všechny tyto funkce jsou poskytovány distribuovaným, vysoce škálovatelným, odolným proti chybám a bezpečným způsobem. Kafka je distribuovaný systém skládající se z producentů, spotřebitelů a událostí, které jsou organizovány a trvale uloženy v tématech. Producenti jsou klientské aplikace, které publikují (zapisují) události do Kafky. Spotřebitelé se přihlašují k odběru (čtení a zpracování) událostí. V Kafce jsou producenti a spotřebitelé navzájem plně odděleni. ZooKeeper je centralizovaná služba pro údržbu konfiguračních informací, pojmenování, poskytování distribuované synchronizace a poskytování skupinových služeb s jednoduchým rozhraním [22, 23].

3 Návrh a Implementace

Kapitola se zabývá popisem návrhu webového nástroje a jeho implementačním řešením. Navazující sekce obsahují prvotní myšlenky návrhu grafického uživatelského rozhraní, návrhu architektury aplikace a návrhu dat. Na základě návrhů je implementována klientská část, která obsahuje uživatelské rozhraní, ověřování identity na serveru, routování ve frontendové části, vytvoření funkcí pro správu webových soketů a funkcí zobrazují data. Serverová část je implementována s využitím webových soketů a Kafky spotřebitele, který zachycuje streamovací data. Součástí implemetace je ověřování uživatele a správa routování, kde pro běžné HTTP požadavky může být routování společené jak pro server tak i pro klienta. Technologie pro klienta byla zvolena knihovna ReactJS v jazyce JavaScript. Technologie pro server byl zvolen modul Tornado v jazyce Python. Obě tyto technologie mají dobrou a přehlednou dokumentaci s rozsáhlou komunitou, která je spravuje. Webová aplikace je implementována formou webového nástroje, který slouží uživatelům jako nástroj pro monitorování provozu GPON sítí. Uživateli umožňuje základní práci a analýzu pomocí intuitivního vizuálního rozhraní. Prvotní návrhy webového nástroje byly vytvořeny tužkou na papír pomocích tzv. drátových modelů. V pozdější fázi došlo k přepracování do digitální podoby. Jedná se o minimalistické návrhy prvotních myšlenek tak, aby vynechávaly určité aspekty, například obrázky, barvy atd. Jde o nezbytnou součást procesu návrhu výsledného produktu. Tato diplomová práce nezahrnuje způsob jakým jsou data získávána z optické sítě, ani způsob jakým jsou zpracovávána pomocí technologie Kafka.

Pro vývoj aplikace bylo použito vývojové prostředí PyCharm [1] pro Python vyvíjené českou firmou JetBrains [2]. PyCharm obsahuje všechny potřebné nástroje pro vývoj aplikací na jednom místě, jako je správa virtuálních prostředí, správu verzování atd.

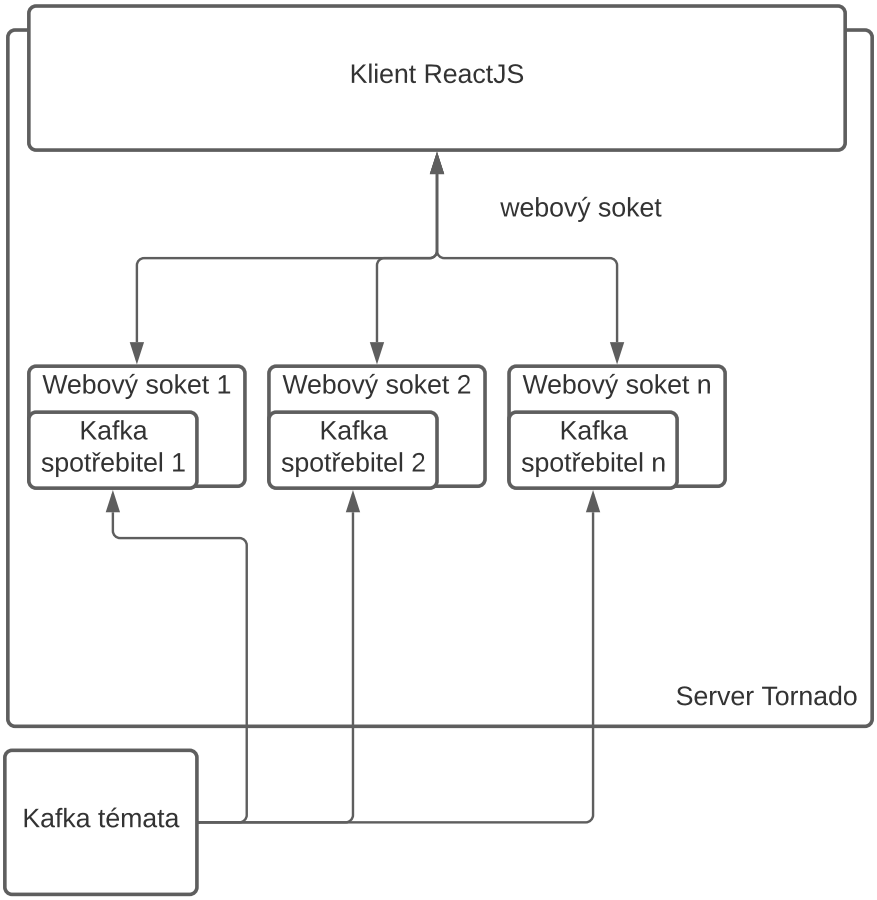
3.1 Návrh architektury systému

Návrh architektury umožňující renderování klientské části na serveru, částečně řeší problémy, které vznikají při přenosu velkého množství dat v reálném čase, kdy je kladen důraz na celkový výkon webové aplikace. Byl navržen systém webového nástroje fungujícího jako jedna samostatná aplikace, která komunikuje přes lokální síť. Obecně je přenos po lokální síti rychlejším řešením, než přenos dat přes celou internetovou síť. Na obrázku 3.1 je zobrazen základní návrh řešení, které umožní

[1] `https://www.jetbrains.com/pycharm/`

[2] `https://www.jetbrains.com/`

zpracovat a odeslat data mezi klientskou částí a serverem. Pro komunikaci mezi klientem a serverem je zvolen komunikační kanál pomocí protokolu WebSocket.



Obr. 3.1: Diagram návrhu architektury systému.

V každém webovém soketu je pak navázáno spojení na streaming dat z Kafky spotřebitele. Jakmile je navázáno spojení mezi klientem a serverem jsou data zpracována a přeposlána na klientskou část aplikace. Klientská část aplikace poté data vhodným způsobem prezentuje uživateli.

Návrh přenášených dat

Velký objem dat, který je přenášen v reálném čase, zatěžuje výpočetní schopnosti aplikace. Proto veškeré zpracování dat probíhá přímo na platformě Kafka. Tím je významně snížena zátěž na samotný server i na klientskou část zobrazující data. Použitý formát je typu JSON, tento formát je běžně používán k přenosu dat ve webových aplikacích.

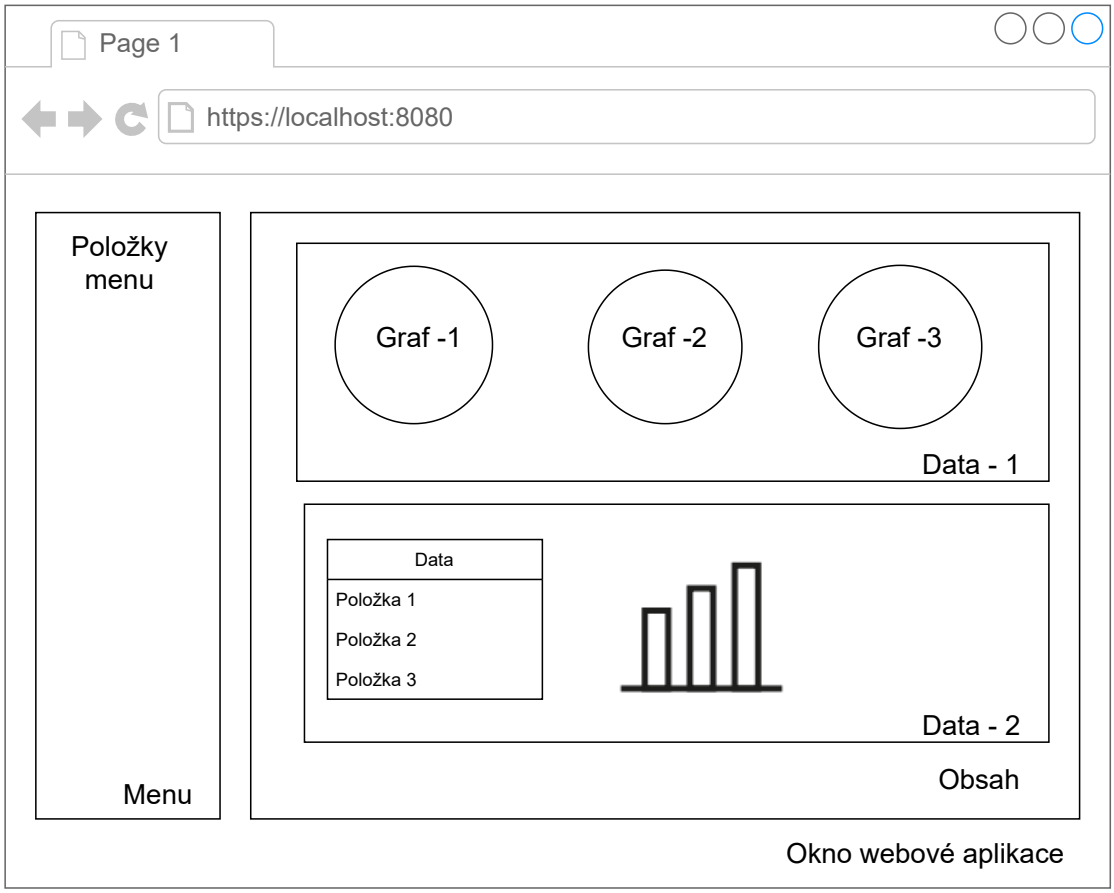
Byly navrženy následující struktury pro analýzu. První strukturou uvedenou v příloze A.1 je struktura, která umožňuje zobrazení aktivních a neaktivních ONU s časovým údajem, kdy došlo k aktivaci nebo deaktivaci. Struktura obsahuje pole objektů, kde každý objekt je tvořen dvěma parametry. První parametr je *onu_id* s číselnou hodnotou typu *int*. Druhý parametr je *log*, jehož hodnotou je pole objektů obsahující dva parametry. Jedná se o parametr *state* s hodnotou typu *bool* vyjadřující aktivaci nebo deaktivaci. Druhým parametrem je *time_stamp* časová známka vyjadřující, kdy došlo ke změně stavu. Druhou strukturou uvedenou v příloze viz A.2 je struktura, která umožňuje zobrazení unikátních PLOAM zpráv a jejich počet, zaslané ke konkrétní ONU. Struktura obsahuje pole objektů, kde každý objekt obsahuje parametry *onu_id* s číselnou hodnotou typu *int* a parametr *message*, vyjadřující údaj o typu PLOAM zprávy. Parametr *message* je pole objektů. Každý objekt obsahuje parametry *id*, *name*, *bin*, *count*. Poslední struktura odpovídá struktuře záhlaví rámce GPON viz příloha A.3.

3.2 Návrh grafického uživatelského rozhraní

Ve spojitosti s webovými aplikacemi se jedná především o grafické nebo textové prvky a jejich rozmístění. Tyto ovládací prvky umožňují uživateli práci s aplikací, získávají od uživatele potřebná vstupní data, reagují na výsledky zpracované aplikací a ty pak prezentují uživateli. Návrh jasného a moderního uživatelského rozhraní je základním aspektem pro následný vývoj a spokojenost uživatelů. Základní návrh GUI aplikace je zobrazen na obrázku 3.2. GUI aplikace obsahuje pevné vertikální menu sloužící k navigaci, které je umístěno na levé straně webového nástroje. Jednotlivé položky menu, tak získají dostatečné množství prostoru a díky vertikálnímu rozložení je umožněn navýšit jejich počet, aniž by byla omezena plocha pro zobrazování dat. Položky menu slouží pro přepínání obsahu celé aplikace. Zbylá plocha je poté využita pro komponenty umožňující prezentaci dat uživateli.

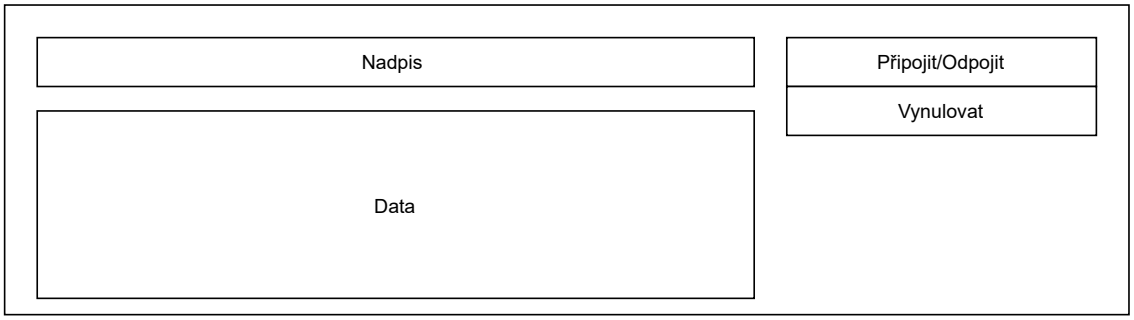
Detail plochy pro zobrazování dat

Komponenty jsou nezávislé a opakovaně použitelné části kódů, aby bylo plně využito výhod ReactJS, je navržena komponenta, zobrazena na obrázku 3.3. Tato komponenta bude obsahovat společné rozhraní. Tlačítka s funkcí pro připojení, odpojení k webovým soketům a vynulování dat. Funkce umožňují uživateli poskytnout kontrolu nad správou připojení. Dále bude obsahovat popisovou část, a oblast pro zobrazení dat. Podle návrhu 3.1 byly zpracovány tři sady dat. Pro každou sadu dat byl vytvořen vhodný návrh zobrazení. První sada obsahuje grafické zobrazení pomocí kruhových grafů, sloužící pro zobrazování unikátních PLOAM zpráv, které



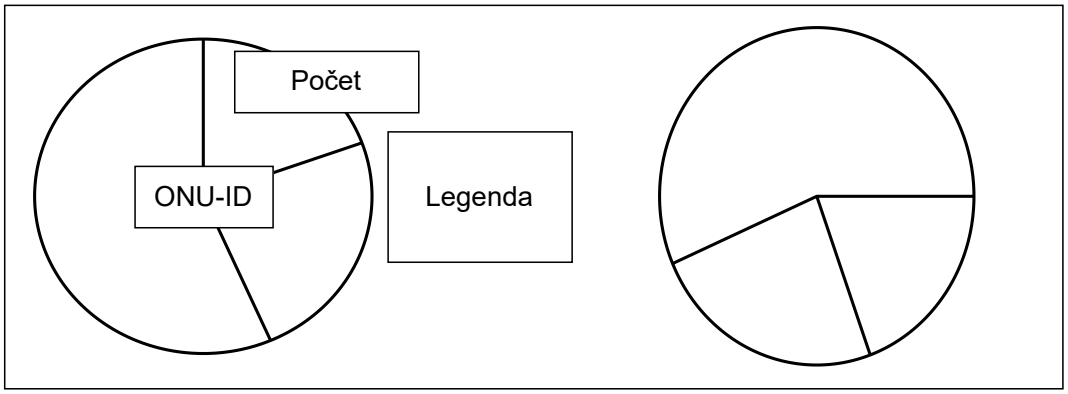
Obr. 3.2: Ukázka návrhu GUI.

jsou adresovány konkrétní ONU. Následující obrázek zobrazuje grafický návrh 3.4, který vychází z dřívějšího návrhu dat A.2.



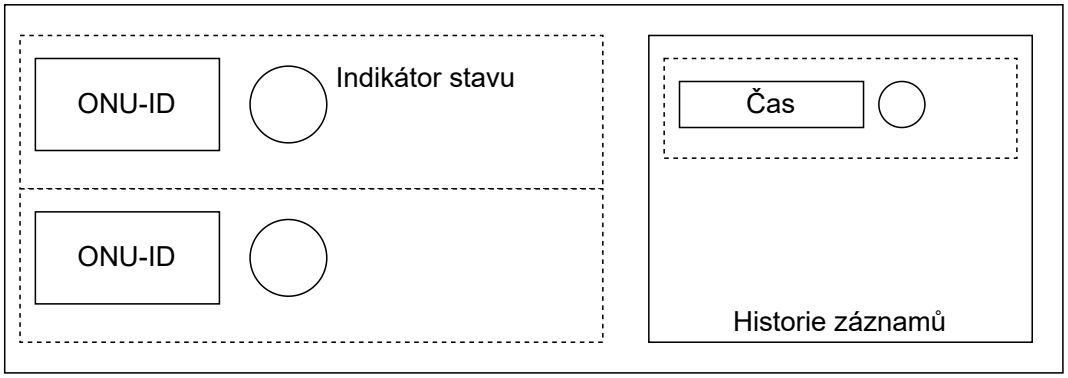
Obr. 3.3: Ukázka návrhu komponenty pro zobrazování dat.

Každý kruhový graf obsahuje ve středu popisek, který slouží pro jednoznačnou identifikaci ONU. Legenda slouží pro zobrazení všech unikátních zpráv a jejich počet. Druhá sada vychází z předešlého návrhu dat podle A.1 grafické znázornění návrhu



Obr. 3.4: Ukázka návrhu GUI pro zobrazení unikátních PLOAM zpráv.

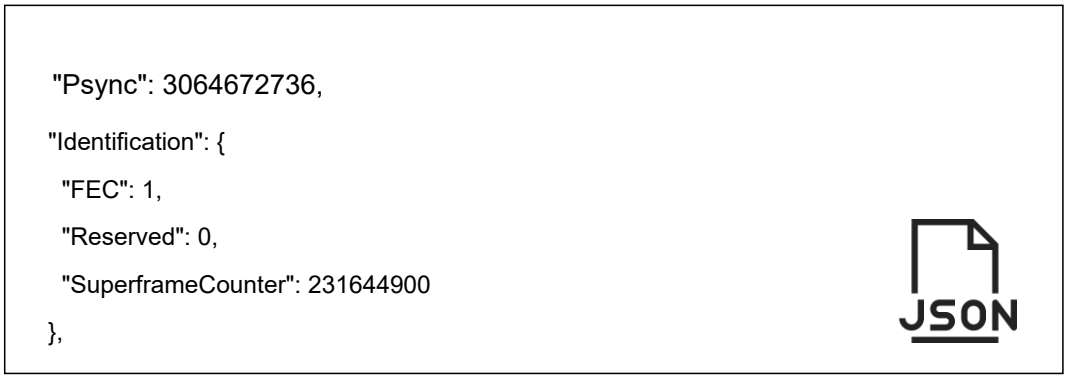
viz obrázek 3.5 umožňuje zobrazovat aktivní a neaktivní ONU pomocí parametru ONU-ID a indikátoru, který mění zbarvení v závislosti na stavu. Dále zobrazuje historii záznamů ve formě času, kdy došlo ke změně. Poslední sada obsahuje záznam celého záhlaví GPON rámce viz návrh 3.6.



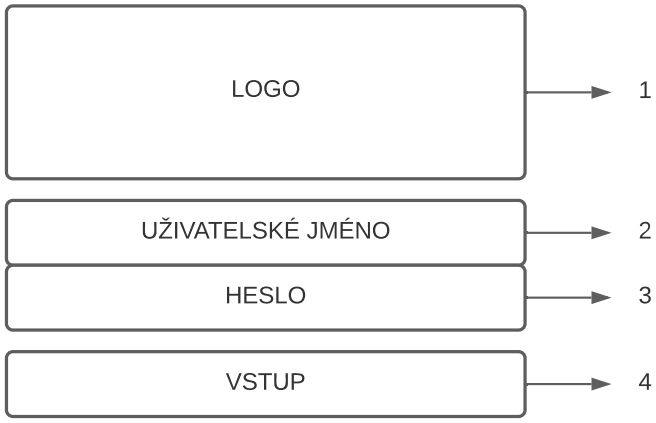
Obr. 3.5: Ukázka návrhu GUI pro zobrazení aktivních a neaktivních ONU.

Návrh přihlašovacího formuláře

Webový nástroj vyžaduje identifikaci a autentizaci uživatele pro vstup do systému. Z tohoto důvodu byl navržen přihlašovací formulář, viz obrázek 3.7, který je používán k zadání ověřovacích údajů pro přístup na webovou stránku, která obsahuje zabezpečený obsah. Obsahuje logo, vstupní pole pro uživatelské jméno a heslo. Po odeslání hodnot z přihlašovacího formuláře pomocí tlačítka jsou vstupní pole zkontrolována a odeslána na server pomocí REST API viz podkapitola 2.3. Na serveru jsou následně tyto udaje ze vstupních polí ověřeny. Pokud server vyhodnotí správnost přihlašovacích údajů, umožní uživateli přístup do webové aplikace.



Obr. 3.6: Ukázka návrhu GUI pro zobrazování záhlaví rámců GPON.



Obr. 3.7: Ukázka návrhu přihlašovacího formuláře.

3.3 Implementace serveru

Jak již bylo probráno v teoretické části 2.1, Tornado dokáže zpracovávat webové sokety a nevyžaduje externí konfiguraci. Na základě návrhu, bylo implementováno řešení, umožňující renderování klientské aplikace na serveru. Metodou pro posílání dat mezi serverem a klientem probíhá pomocí protokolu WebSocket formou endpointů, které umožňují po navázání spojení výměnu dat.

Popis hlavního modulu

Výpisu programového kódu níže 3.1, zobrazuje implementaci potřebnou pro spuštění Tornado aplikace na systému Windows 10.

Výpis 3.1: Hlavní modul pro běh celé serverové aplikace

```
1  if __name__ == '__main__':
2      # for windows
3      asyncio.set_event_loop_policy(
4          asyncio.WindowsSelectorEventLoopPolicy()
5      )
6      app = make_app()
7      app.listen(port=app.config.server.PORT)
8
9      my_ioloop = tornado.ioloop.IOLoop.current()
10     my_ioloop.start()
```

Spouštění celé serverové aplikace probíhá pomocí hlavního skriptu. Soubory v Pythonu se nazývají moduly a jsou identifikovány příponou souboru .py. Modul může definovat funkce, třídy, proměnné. Když tedy Python interpret spustí modul s názvem *run_server.py* a proměná *___name___* je nastavena na *___name___* je tento modul modul spuštěný jako hlavní program. Unitř toho modulu je volána funkce *make_app* pro nastavení aplikace. Dále je potřeba nastavit port, na kterém server bude naslouchat metodou *listen*. Port je nastaven v souboru conf 3.3 na hodnotu 8080 a je načten ze souboru .env. Volání metody *listen* server ještě nespustí. Je třeba vytvořit vstupně-výstupní smyčku, která dokáže naslouchat požadavkům a vracet odpovědi pomocí *tornado.ioloop.IOLoop*. Tento typ konfigurace je užitečný pro běh aplikace v cloudu, skript lze spustit několikrát na stejném serveru a každý na jiném portu. Výpis programového kódu níže 3.2, představuje implementaci funkce pro vytvoření Tornado serveru.
Při implementaci funkce pro aplikaci Tornado bylo potřeba nastavit mapování adres URL. Z tohoto důvodu bylo nutné namapovat veškeré statické soubory jako CSS, JS, obrázky včetně celé přeložené aplikace Reactu nacházející se ve složce build. Složka build vznikně při vytváření produkční verze. V tomto případě konfigurace obsahuje nastavení *static_path* pomocí modulu *StaticFileHandler* na řádku 12, které slouží pro renderování klientské části. Toto nastavení se používá k informování modulu Tornado o úplné cestě k souborovému systému pro statické prostředky. Další nastavení, které Tornado nabízí je nastavení tajného klíče, který se využívá při zabezpečení souborů cookie a přihlašovací adresy URL, na kterou jsou uživatelé přesměrováni pokud nemají potřebnou autentizaci.

Výpis 3.2: Metoda pro vytvoření Tornado serveru

```
def make_app() -> Application:
    """
    Method creates tornado application.
    """
    config: Config = Config()

    app: Application = Application(
        service_name=config.server.NAME,
        debug=config.server.DEBUG,
        handlers=[
            # Render build from RectJS
            (r"/build/(.*)",
            StaticFileHandler,
            {"path": config.server_static.react_path}),
            # Custom endpoints
            (r"/login", handlers.LoginHandler),
            # WebSocket endpoints
            (r'/gpon-frames', GponFramesWebSocket),
            (r'/connected-onus', ConnectedOnuWebSocket),
            (r'/unique-ploam-mess', UniquePlomMessagesWebSocket),
            (r"/.*", handlers.MainPageHandler),
        ], )

    app.router = load_router()
    app.config = config

    return app
```

Popis enpointů Websocket

Implementace zahrnovala vytvoření společné třídy *WebSocketHandler* viz ukázky kódu 3.3 a třídy pro jednotlivé endpointy např. pro zobrazení aktivních a neaktivních ONU viz příloha F.1.

Výpis 3.3: Ukázka třídy pro konkretní endpoint

```
1  class WebSocketHandler(ws.WebSocketHandler):
2
3      config: Config = Config()
4      stop = False
5
6      @gen.coroutine
7      def on_message(self, message):
8          if message == "stop":
9              yield gen.sleep(0.01)
10             self.stop = True
11
12     def open(self):
13         ioloop.IOLoop.instance().add_timeout(
14             datetime.timedelta(seconds=1),self.send_data)
15
16     # typical for 403
17     def check_origin(self, origin):
18         return True
19
20     def on_close(self):
21         pass
```

Společná třída využívá modulu *WebSocketHandler* obsahující metody, pro správu soketu. Konkrétní endpointy pak využívají dědičnosti ze společné třídy. Metoda *open* slouží pro otevření a navázání komunikačního kanálu. Ve funkci *open* bylo nastaveno sekundové zpoždění, jedná se o časový interval, který je potřebný k otevření komunikace. Po úspěšném navázání komunikace dojde k připojení Kafky spotřebitele na datový tok. Pokud existují data v datovém toku je zahájen přenos dat mezi klientem a serverem metodou *send_data*. Metoda *send_data* společně s metodou *write_message* slouží pro odesílání zpráv klientovy. Metoda *on_close* slouží pro uzavření komunikačního kanálu. Spojení může být uzavřeno ze strany serveru pokud neexistují žádná data pro zobrazení. Ze strany klienta může být uzavřeno spojení posláním zprávy *stop* s využitím metody *on_message*. Metody *send_data* a *on_message* využívali dekoratér pro asynchronní zpracování, aby bylo možné přerušit Kafku spotřebitele. Po uzavření spojení dochází i k opuštění tématu Kafky.

V této funkci bylo potřeba uzavřít i Kafku spotřebitele. V příloze E je uvedena dokumentace koncových bodů API.

Popis config souboru

Tento soubor obsahuje globální konfigurační nastavení pro server. Struktura tohoto souboru je členěna do tříd. Jednotlivé třídy jsou pak přiřazeny k hlavní třídě s názvem *Config*, která je zobrazena na následujících řádcích v příloze D.1. Každá třída spravuje určitou část nastavení, například pro nastavení serveru slouží třída *ServerConfig* zobrazena v příloze D.3.

Výpis 3.4: Dílčí konfigurace statických cest

```
@dataclass()
class StaticConfig:
    """
    Static configuration
    """
    BASE_DIR = os.path.dirname(
        os.path.dirname(os.path.abspath(__file__))
    )

    img_path: str = os.path.join(BASE_DIR, '..', 'img/')
    react_path: str = os.path.join(BASE_DIR, '..', 'build/')
    css_path: str = os.path.join(BASE_DIR, 'css/')
    js_path: str = os.path.join(BASE_DIR, 'js/')
    html_path: str = os.path.join(BASE_DIR, 'html/')
```

Pro nastavení statických cest slouží třída *StaticConfig*, která je zobrazena na následujícím kódu 3.4 nebo nastavení Kafky třída *KafkaConfig* viz příloha D.2. Třídy, které obsahují citlivé údaje jako jsou například port, na kterém bude spuštěná serverová část, volba ladícího režimu nebo tajný klíč jsou uloženy v *.env* souboru. Tento soubor slouží pro ukládání individualních uživatelských proměnných jako jsou právě klíče nebo data, která nemají být uložena na repozitáři.

Implementace Kafka spotřebitele

Z návrhu byla implementována třída pro Kafku spotřebitele umožnující načítat data ze streamu. Spotřebitel je nakonfigurován přijímat tři témat. První téma s názvem GPONFrames obsahuje stream zachycených rámců. Tyto data nejsou žádným způsobem filtrována ani upravena. Jedná se čistě o zobrazení zachycených rámců s jejich parametry viz příloha B.1. Druhým tématem je ConnectedOnus viz příloha B.2, toto

téma je schopno zobrazovat dvě pole objektů , první pole slouží pro ukládání aktivních ONU, u kterých byla zachycena aktivační zpráva s konkrétním časem, kdy byla zpráva zachycena. Druhé pole slouží pro ONU, u kterých byla zachycena deaktivační zpráva. Výhodou této struktury je zobrazování předešlých dat. V daný moment je k dispozici výčet ONU-ID, který nikdy nepřesáhne počet větší jak 255 objektů v jednom nebo druhém poli. Posledním tématem je UniquePloamMessages viz příloha B.3. Tato data nesou informaci o unikátních PLOAM zprávách, jejich počtu, binární číslo pro danou zprávu a ONU-ID, kterému byla zpráva zaslána. I tato struktura dat byla navržena s ohledem na ukládání předešlých stavů.

Použitá knihovna kafka-python je navržena tak, aby fungovala podobně jako oficiální systém Apache Kafka. Implementaci Kafka spotřebitele zahrnovalo vytvoření globální konfigurace viz příloha D.2 a implementaci třídy, která obstarává základní správu obsluhy streamovaných dat. Následující pseudokód 3.5 umožňuje číst streamovaná data z tématu vytvořeného producentem.

Výpis 3.5: Základní nastavení Kafka spotřebitele

```
consumer = KafkaConsumer(
    config.kafka.TOPIC
    bootstrap_servers=[config.kafka.FULL_ADRESS],
    auto_offset_reset=config.kafka.AUTO_OFFSET_RESET,
    enable_auto_commit=config.kafka.AUTO_COMMIT,
    group_id=group_id,
    consumer_timeout_ms=config.kafka.TIMEOUT,
    value_deserializer=lambda data: loads(data.decode('utf-8')),
    api_version=config.kafka.API_VERSION
```

Pro inicializaci objektu spotřebitele a je nutné poskytnout třídě základní argumenty.

- téma (topic), ze kterého spotřebitel čte data,
- ID skupiny spotřebitelů (group_id), které je spotřebitel součástí (může být libovolný řetězec),
- bootstrap_servers - obsahuje ip adresu serveru a port,
- auto_offset_reset - s hodnotou 'earliest' je přidána k načtení událostí od samého začátku,
- enable_auto_commit - umožňuje pravidelné potvrzování zpráv na pozadí,
- onsumer_timeout_ms - umožňuje odpojení od streamovacích dat pokud vyprší časový interval.
- value_serializer slouží pro deserializaci dat

Spotřebitel je nakonfigurován tak aby používal JSON deserializátor, který slouží pro převod bajtového toku zpátky na formát UTF-8. Data lze ve formatu JSON

serializovat a de-serializovat před odesláním a přijímání. Data ve formátu JSON mohou být zasílána producentem Kafky a čtena spotřebitelem Kafky pomocí modulu json. Pro opuštění skupiny témat je nutné Kafku spotřebitele vypnout.

3.4 Implementace klienta

Pro tvorbu grafického uživatelského rozhraní byla použita knihovna Ant Design [3]. Jedná se o populární knihovnu, která obsahuje již připravené komponenty pro ReactJS a lze je snadno přizpůsobit pro implementaci webového nástroje. Tyto komponenty značně urychlují vývoj celé aplikace. Podle grafického návrhu viz v textu výše 3.2, bylo implementováno rozhraní aplikace. Skládá se z levého navigačního menu a hlavní plochy, která slouží pro zobrazování dat. Aplikace disponuje ověřováním uživatele prostřednictvím API, využitím access tokenu, který je vygenerován na straně serveru. Při každém požadavku klienta je na serveru ověřena platnost tokenu. Z důvodu množství dat, která se přenáší v reálném čase je nevýhodné použití klasického přístupu dotaz/odpověď. Proto byla vytvořena komunikace, kde klient naváže spojení pomocí webových soketů. Po vytvoření spojení získává data ze serveru viz teorie webových soketů 2.3. Vytvořená aplikace nezpracovává backendovou logiku ani databáze. Slouží pouze pro vytvoření frontendové části, která jde použít s jakýmkoliv backendem.

Vytvoření React aplikace

Create React App je prostředí pro vytváření nové jednostránkové aplikace v Reactu bez nutnosti konfigurací, které jsou skryté uživateli a díky kterým je možné se soustředit na vytváření aplikace místo na nastavování prostředí pro vývoj. Tato funkce se postará o nastavení vývojového prostředí a umožní tak využívat nejnovější funkce JavaScriptu. Poskytuje vhodné prostředí pro vývojáře a optimalizuje aplikaci pro nasazení pro produkci. Pomocí příkazů 3.6 lze vygenerovat a spustit aplikaci React a začít tak implementovat frontendovou část.

Výpis 3.6: Vytvoření aplikace v React

```
1 npx create-react-app my-app
2 cd my-app
3 npm start
```

[3] https://ant.design/

Správa stavu v rámci celé aplikace

Pro správu stavu je použit React-Redux, který dokáže uchovávat stav celé aplikace oproti Reactu, který dokáže uchovávat stav konkrétní komponenty. Redux je samostatná knihovna JS a byl speciálně navržen tak, aby fungoval s Reactem. React-Redux vyžaduje nastavení konkrétního úložiště. V celé aplikaci by mělo být použito pouze jedno úložiště. Proto je potřeba použít komponentu *Provider*, do které je vložena celá aplikace *App*. Tímto způsobem získá celá aplikace v případě potřeby přístup do úložiště. Následující pseudokód 3.7 ilustruje implementaci použití.

Výpis 3.7: Ukázka implemetace správy stavu v rámci celé aplikace

```
1 const store = createStore(reducer, applyMiddleware(thunk))
2 const App = () => {
3     return (
4         <Provider store={store}>
5             <ApplicationRoutes/>
6         </Provider>
7     );
8 };
9 export default App;
```

Komponenta *App* je počáteční komponentou pro inicializaci aplikace. Funkce *createStore* na prvním řádku vytvoří uložiště Redux, který obsahuje kompletní stavový strom aplikace. Argumenty pro tuto funkci jsou:

- reducer - funkce, která vrací další stavový strom , daný aktuální stavový strom a akci ke zpracování,
- applyMiddleware (thunk) - Thunks jsou doporučeným middlewarem pro základní logiku, včetně komplexní synchronní logiky, která vyžaduje přístup do úložiště a jednoduché asynchronní logiky, jako jsou požadavky AJAX.

Funkce *createStore* vrací objekt, který obsahuje úplný stav aplikace. Jediným způsobem, jak změnit stav, je odeslání akce. Každá komponenta v aplikaci má možnost číst a využívat stav úložiště a aktualizovat tak uživatelské rozhraní.

Základní rozložení aplikace

Komponenta *Layout* slouží pro rozložení stránky na jednotlivé části, které lze nazvat kontejnery. Aplikace je rozdělena do dvou kontejnerů, kde v prvním, levém kontejneru je komponenta postranního panelu *Sider* a v druhé části se nachází *Layout*, ve kterém jsou definované komponenty horního rozložení *Header*, rozložení obsahu *Content* a spodní rozložení *Footer* jak je patrné z obrázku 3.8.

Výpis 3.8: Ukázka základního rozložení aplikace pomocí Ant Design komponent

```
<Layout>
    <Sider></Sider>
    <Layout>
        <Header></Header>
        <Content></Content>
        <Footer></Footer>
    </Layout>
</Layout>
```

Navigační menu aplikace

Uvnitř komponenty *Sider* je volána komponenta *SideNav* umožňující přepínat obsah. Komponenta *Router* umožňuje použít React hooks, které využívají přístup ke stavu routeru a navigaci uvnitř komponent. V tomto případě je využit hook s názvem *useHistory*. Celé GUI viz obrázek výše 3.8 je zabaleno do komponenty *Router*, které je nastavena instance history. Uvnitř komponenty *Content* je pak použita komponenta *Switch* společně s *Route* umožňující přepínat obsah. V pseudokódu 3.9 je zobrazen příklad řešení pro přepínání obsahu pomocí společného souboru pro klientskou a serverovou část.

Výpis 3.9: Ukázka pseudokódu pro přepínaní obsahu

```
<Switch>
    <Route exact path="/login" component={LoginJSX}/>
    {token ? routerConfig.routes.map((route, index) => {
        return (
            <Route
                key={index} exact path={route.path}
                component={
                    componentRegistry[route.component]
                }
            />
        );
    }) : <Redirect to="/login"/>}
    <Route path="*"><Page404/></Route>
</Switch>
```

Ve výchozím nastavení je ověřována přítomnost tokenu. Pokud klient neobdrží token od serveru je přesměrován na přihlašovací stránku. V opačném případě je umožněn přístup k routám. Při pokusu dostat se na jinou routu, než je uvedena ve společném routovacím souboru je klient přesměrován na stranu s obsahem 404, která značí, že tato routa nebyla nalezena.

Implementace komponenty pro zobrazování dat

V rámci znovupoužitelnosti byla vytvořena komponenta s názvem *Feature*. Tato komponenta díky vhodnému návrhu umožňuje vícenásobné použití pro různé typy dat, které však mají rozdílnou strukturu a vyžadují proto rozdílné funkce pro jejich zpracování. Společnými prvky komponenty jsou funkce pro správu komunikačního kanálu, zobrazení titulního nadpisu a plochy pro prezentaci data.

Výpis 3.10: Ukázka použití komponenty Feature

```
<Feature
    title={"Show active and deactive ONU by ONU-ID"}
    child={<Connected/>}
    uri={socket_conn_onu}
    />
```

Komponenta vyžaduje pouze tři vstupní parametry tak jak je zobrazeno na ukázce 3.10. Jedná se o title, child, a *uri*. Parametr *title* je textový řetěz. Vstup pro data je umožněn parametrem *child*. Jedná o vstupní funkci, kde uvnitř každé funkce dochází k zpracování dat. Každé zobrazení dat má svoji vlastní funkci, ve které bylo potřebné data zpracovat. Posledním argumentem je *uri* jedná se o URL adresu endpointu, ke kterému se komponenta připojuje viz dokumentace k API endpointům E.

Výpis 3.11: Ukázka implementací funkcí pro připojení/odpojení k endpointům

```
const handlConnectWebSocketClick = (client, setClient, uri) => {
    try {
        if (!client)
            setClient(new W3CWebsocket(uri))
    } catch (err) {
        console.log(err)
        console.log("cant connect maybe server is turn-off")
    }
}

const handlDisconectWebSocketClick = (client, setClient) => {
    if (client)
        client.send("stop")
        client.close()
        setClient(null)
}
```

Pro implementace JS protokolu WebSocket byla použita knihovna *w3cwebsocket*, tato knihovna je kompatibilní se serverovou částí. Následně byly implementovány

funkce, které umožňovaly připojení popřípadě odpojení od webového soketu viz příklad kódu 3.11. Komponenta *Feature* zahrnovala deklaraci stavových proměnných pro ukládání dat a pro ukládání stavu pro komunikační protokol WebSocket. Tyto stavové proměně byly implementovány s využitím hooku *useState*. Tento hook vrácí dvojici hodnot. Aktuální stav a funkci, která jej aktualizuje. Komponenta *Feature* využívá dalšího hooku a to *useEffect*. Použití *useEffect* uvnitř komponenty umožňuje přístup ke stavové proměnné a tím je zajištěno vykreslování při změně stavu. Ukázka části kódu viz 3.12, zobrazuje implementaci React hooků. Na 11. řádků jsou zpracována příchozí data ve formátu JSON a následně je aktualizována proměnná pro ukládání dat. Komunikační kanál je uzavřen pokud dojde k neočekávané výjimce.

Výpis 3.12: Ukázka implementací useState a useEffect pro komponentu Feature

```
1  const [result, setResult] = useState([])
2  const [client, setClient] = useState(null)
3
4  useEffect(() => {
5
6      if (client) {
7          client.onopen = () => {}
8          client.onclose = (event) => {
9              setClient(null)
10         }
11         client.onmessage = (message) => {
12             let data = JSON.parse(message.data)
13             setResult(data)
14         }
15         client.onerror = err => {
16             console.log(err)
17             client.close()
18         }
19     }
20 })
```

Vytvoření produkční aplikace

Pokud je aplikace připravená na nasazení do produkce. Spuštěním příkazu 3.6 dojde k vytvoření optimalizované aplikace.

Výpis 3.13: Sestavení klientské části pro produkci

```
1  npm run build
```

4 Dosažené výsledky

Aplikace byla testována na datech z reálné sítě, která byla uložena v souboru na lokálním úložišti. Pro demonstraci testování byly některé záhlaví rámce GPON vhodně upraveny, protože sada testovacích dat obsahovala převážně stejné typy záznamů. Vlastnosti platformy Kafka umožňují data předem zpracovat a poskytovat pomocí témat. Díky těmto vlastnostem je webový nástroj odlehčen od výpočetní složitosti, která by jinak neumožňovala zpracování dat přenášených v reálném čase. Z testování vyplynula potřeba nastavení časového intervalu. Tento časový interval byl nastaven na hodnotu jedné sekundy, tak aby byl systém schopný přerendrovat obsah dat. Při implementaci byl kladen důraz na znovupoužitelnost kódu, tak aby bylo možné intuitivně doprogramovat systém. Jedná se zejména o implementaci endpointů formou webových soketů, poskytující data na straně serveru. Na straně klienta jde o implementovanou komponentu, která umožňuje správu připojení k endpointům a výsledné zobrazení dat.

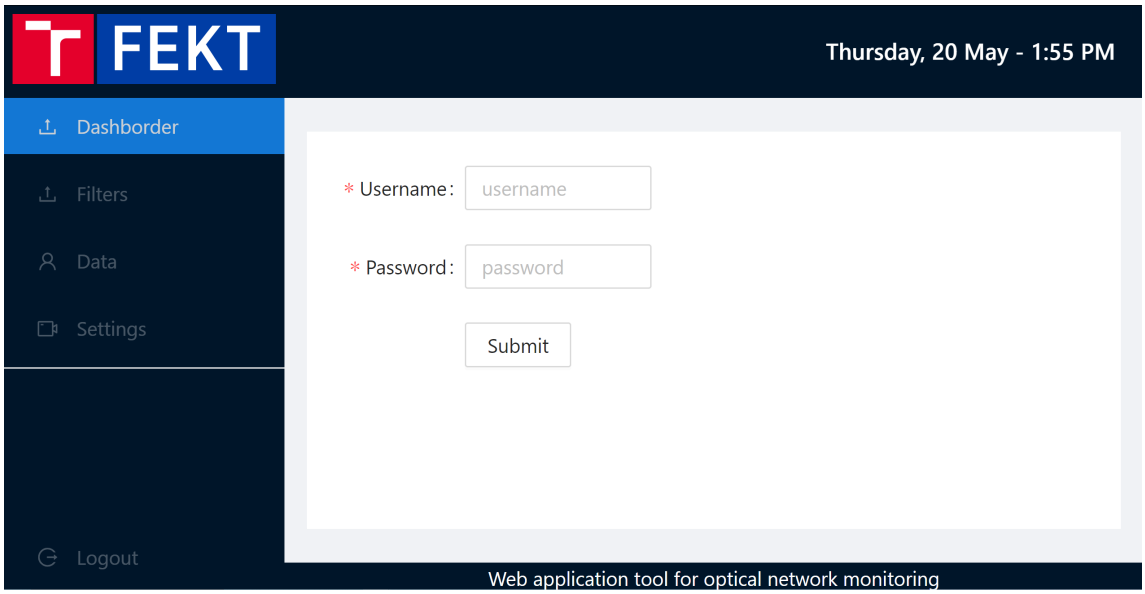
Konečná architektura systému tak není omezená pouze na sítě typu GPON a lze ji tak rozšířit i na ostatní standardy pasivních optických sítí. Díky vhodnému návrhu a implementaci tak lze webový nástroj doimplementovat o nové standardy pasivních optických sítí.

Výsledné grafické uživatelské rozhraní aplikace

Na obrázku 4.1 je zobrazen výsledek implementace webového nastroje pro monitorování sítě po načtení aplikace. Aplikace vyžaduje ověřovací údaje pro vstup do systému. Navigační menu pro přepínání ploch je ve výchozím nastavení deaktivované. Po zadání ověřovacích údajů je umožněn vstup do systému s možností přepínání ploch pomocí levého menu. Při odhlášení se aplikace vrátí do výchozího stavu, kdy vyžaduje opětovné zadání ověřovacích údajů.

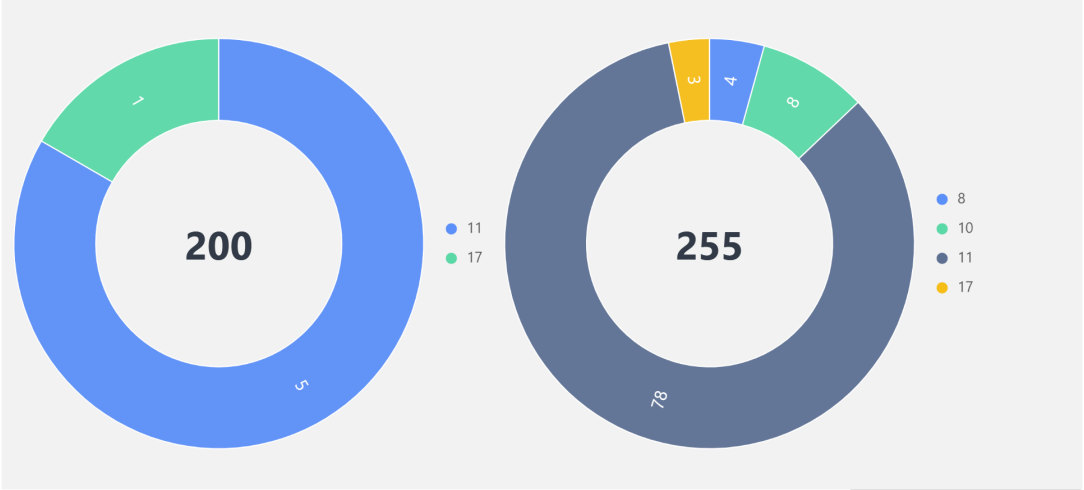
Výsledné zobrazení dat

První příklad zpravování dat zobrazuje obrázek 4.2. V tomto případě se jedná o unikátní zprávy PLOAM, obsažené v záhlaví rámce GPON. Výsledné řešení zobrazuje uživateli typ zpráv a jejich počet formou kruhového grafu. Uvnitř grafu je zobrazen popisek ONU-ID, kterému byly zprávy zaslány. Dalším příkladem zpracování dat je forma náhledu, která je zobrazena na obrázku 4.3. Data jsou zde prezentována formou JSON se strukturou plně odpovídající struktuře záhlaví rámce GPON. Uživateli je umožněn náhled jednotlivých polí a jejich hodnot s informací o datovém typu. Hodnoty objektů, polí, řetězců lze rozbalit a sbalit. Struktura je například vhodná pro zobrazení nestandardních dat a umožňuje tak zkontrolovat jednotlivé



Obr. 4.1: Ukázka výsledného GUI po načtení webového nástroje.

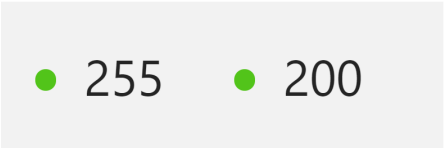
hodnoty obsažené v záhlaví rámce GPON. Podobné zobrazení dat existuje například ve webovém prohlížeči v nástroji DevTools.



Obr. 4.2: Ukázka výsledného GUI, detail zpracování dat unikátní zprávy PLOAM

Cílem posledního náhledu, který je zobrazen na obrázku 4.4, bylo umožnit elementární přehled o aktivní nebo neaktivních ONU, které se v síti nacházejí. K tomuto zobrazení byl použit indikátor stavu společně s parametrem ONU-ID.

Obr. 4.3: Ukázka výsledného GUI, detail zpracování celé hlavičky rámce GPON.



Obr. 4.4: Ukázka výsledného GUI, detail aktivních a neaktivních ONU.

Závěr

Cílem práce bylo nastudovat gigabitové pasivní optické sítě, přenos rámců, formát rámců a význam jednotlivých polí. Na základě povahy GPON provozu definovat data vhodné pro zobrazení uživateli. Následně pro tato data implementovat webový nástroj. Aplikace by měla být přehledná, intuitivní, zabezpečená proti zneužití a měla by obsahovat připojenou dokumentaci k API.

Studium bylo zaměřeno na technologie Tornado, ReactJS a Kafka. Serverová část aplikace byla vytvořena v programovacím jazyku Python pomocí modulu Tornado, který podporuje připojení pomocí protokolu WebSocket, včetně asynchronních operací. Hlavní výhodou modulu Tornada je využití integrovaného serveru, který umožňuje spuštění produkční veze bez nutnosti serverů jako jsou Apache nebo NGINX. Klientská část byla implementována v ReactJS s využitím knihovny Ant Design pro grafický design, který lze snadno přizpůsobit pro implementaci webového nástroje. Výsledná klientská část webové aplikace je přeložena do produkční verze a spuštěna jako statická stránka na serveru. Komunikační kanál mezi klientským a serverovým rozhraním probíhá pomocí webového soketu, kdy při úspěšném navázání dojde k připojení na konkrétní téma Kafky, která umožňuje načíst data ve formátu JSON. Tato diplomová práce nezahrnuje způsob jakým jsou data získávána z optické sítě, ani způsob jakým jsou zpracovávána pomocí platformy Kafka. Výsledné řešení webového nástroje pro monitorování sítě GPON je demonstrativního charakteru s využitím moderních technologií. Možným vylepšením aplikace je rozšíření funkcionality o offline analýzu a o analýzu dat zpětně, včetně možnosti exportování dat v csv formátu.

Literatura

[1] Cedric F. Lam. *Passive optical networks: principles and practice*. Elsevier/Academic Press, 2007. URL: http://gen.lib.rus.ec/book/index.php?md5=dd7c7d5dea7726e94930fec5d6f4a6bc.

[2] Kevin Bourg James Farmer, Brian Lane and Weyl Wang (Auth.). *FTTx Networks. Technology Implementation and Operation*. Morgan Kaufmann, 1st edition edition, 2016. URL: http://gen.lib.rus.ec/book/index.php?md5=72d3a8269e281117e29cd5a011106932.

[3] Elmar Trojer(auth.) Dave Hood. *Gigabit-Capable Passive Optical Networks*. 2012. URL: http://gen.lib.rus.ec/book/index.php?md5=c53b233f2d912852e5dfeb98596b1b3c.

[4] ITU-T G.984.3. Gigabit-capable passive optical networks (g-pon): Transmission convergence layer specification. Technical report, January 2014. URL: https://www.itu.int/rec/T-REC-G.984.3/en.

[5] ITU-T G.984.1. Gigabit-capable passive optical networks (g-pon): Enhancement band. Technical report, March 2008. URL: https://www.itu.int/rec/T-REC-G.984.1/en.

[6] ITU-T G.984.2. Gigabit-capable passive optical networks (gpon): Physical media dependent (pmd) layer specification. Technical report, September 2019. URL: https://www.itu.int/rec/T-REC-G.984.2/en.

[7] ITU-T G.984.4. Gigabit-capable passive optical networks (g-pon): Ont management and control interface specification. Technical report, February 2008. URL: https://www.itu.int/rec/T-REC-G.984.4/en.

[8] ITU-T G.984.5. Gigabit-capable passive optical networks (g-pon): Enhancement band. Technical report, May 2014. URL: https://www.itu.int/rec/T-REC-G.984.5/en.

[9] ITU-T G.984.6. Gigabit-capable passive optical networks (gpon): Reach extension. Technical report, March 2008. URL: https://www.itu.int/rec/T-REC-G.984.6/en.

[10] Josep Prat Josep Prat. *Next-Generation FTTH Passive Optical Networks - Research Towards Unlimited Bandwidth Access*. Springer, 2008. URL: http://gen.lib.rus.ec/book/index.php?md5=e2f61bee56055377499459b6f23b1ecf.

[11] Dirk Trossen George C. Polyzos (auth.) Ioannis Tomkos Christos J. Bouras Georgios Ellinas Panagiotis Demestichas Prasun Sinha (eds.) Nikos Fotiou, Pekka Nikander. *Broadband Communications, Networks, and Systems: 7th International ICST Conference, BROADNETS 2010, Athens, Greece, October 25–27, 2010, Revised Selected Papers*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 66. Springer-Verlag Berlin Heidelberg, 1 edition, 2012.

[12] Brendan Berg Michael Dory, Adam Parrish. *Introduction to Tornado: Modern Web Applications with Python*. O'Reilly Media, 2012.

[13] Ted Roden. *Building the Realtime User Experience: Creating Immersive and Interactive Websites*. O'Reilly Media, 1 edition, 2010. URL: `http://gen.lib.rus.ec/book/index.php?md5=aeaa2ed3f930e85613b37fb3a4e8082c`.

[14] Tornado web server - tornado 6.1 documentation. URL: `https://www.tornadoweb.org/en/stable/`.

[15] Eve Porcello Alex Banks. *Learning React: Functional Web Development with React and Redux*. O'Reilly Media, 1 edition, 2017. URL: `http://gen.lib.rus.ec/book/index.php?md5=5e57cc0c2e767e239cfc89944d8f3158`.

[16] Artemij Fedosejev. *React.js Essentials: A fast-paced guide to designing and building scalable and maintainable web apps with React.js*. Packt Publishing, 2015. URL: `http://gen.lib.rus.ec/book/index.php?md5=04d7e6f328d5ed5213d7c1634862e12e`.

[17] Josh Powell Michael Mikowski. *Single page web applications: JavaScript end-to-end*. Manning Publications, 2013.

[18] Vangos Pterneas. *Getting Started with HTML5 WebSocket Programming*. Packt Publishing, 2013. URL: `http://gen.lib.rus.ec/book/index.php?md5=61ae9fb30be4d938982c60b4bf6c3ec0`.

[19] Varun Chopra. *WebSocket Essentials: Building Apps with HTML5 WebSockets*. Packt Publishing - ebooks Account, 2015. URL: `http://gen.lib.rus.ec/book/index.php?md5=19a83d6d9a1da07f3b986123f824bda2`.

[20] Peter Moskovits Vanessa Wang, Frank Salim. *The Definitive Guide to HTML5 WebSocket*. Apress, 1 edition, 2013. URL: `http://gen.lib.rus.ec/book/index.php?md5=afa79dc63045822f2aacfd0c41cf61d2`.

[21] Mark Elman, Julia;Lavin. *Lightweight Django*. O'Reilly, 2015. URL: http://gen.lib.rus.ec/book/index.php?md5=8AA82FF797CE3504BC5DE4CC5DE521D0.

[22] Documentation. URL: https://kafka.apache.org/documentation/.

[23] Welcome to apache zookeeper™. URL: https://zookeeper.apache.org/.

Seznam symbolů, veličin a zkratek

AJAX	Asynchronous JavaScript And XML
APON	ATM Passive Optical Network
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
BIP	Bit Interleaved Parity
BER	Bit Error Ratio
BPON	Broadband Passive Optical Network
CRC	Cyclic Redundand Check
CRUD	Create, Read, Update, Delete
CSS	CascadingStyle Sheets
DOM	Document Object Model
FEC	Forward Error Correction
EPON	Ethernet Passive Optical Network
GEM	GPON Encapsulation Mode
GPON	Gigabit Passive Optical Networks
GPS	Global Positioning System
GTC	GPON Transmission Convergence Layer
GUI	Graphical User Interface
ITU	International Telecomunication Unit
HEC	Header Error Control
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
MPA	Multi-Page Applications
OLT	Optical Line Termination

OMCI	ONU Management and Control Interface
ONT	Optical Network Termination
ONU	Optical Network Unit
P2MP	Point-To-Multipoint
PCBd	Physical Control Block downstream
PLI	Payload Length Indicator
PLend	Payload Length downstream
PLOAMd	Physical Layer OAM Operations, Administrations and Maintenance downstream
PON	Passive Optical Network
Psync	Physical Synchronization
PTI	Payload Type Indicator
SEO	Search Engine Optimization
SDU	Service Data Unit
SPA	Single Page Application
SSL	Secure Sockets Layer
TDM	Time Division Multiplex
T-CONT	Transmission Container
UE	User Experience
UI	User Interface
URL	Uniform Resource Locator
WDM	Wavelength Division Multiplexing
WSGI	Web Server Gateway Interface
WWW	World Wide Web

Seznam příloh

A Příklady návrhu dat

Výpis A.1: Návrh struktury dat pro zobrazení aktivních a neaktivních ONU

```
[
  {
    "onu_id": 20,
    "log": [
      {
        "state": 1,
        "time_stamp": 1621246629
      },
      {
        "state": 0,
        "time_stamp": 1921246789
      }
    ]
  },
  {
    "onu_id": 100,
    "log": [
      {
        "state": 0,
        "time_stamp": 1627848448
      }
    ]
  }
]
```

Výpis A.2: Návrh struktury dat pro zobrazení unikátních PLOAM zpráv pro konkrétní ONU

```
[
  {
    "onu_id": 20,
    "message": [
      {
        "id": 12,
        "name": "PLOAM_EMPTY_MESSAGE",
        "bin": "b\\x14",
        "count": 23
      },
      {
        "id": 18,
        "name": "BER_Interval",
        "bin": "b\\x18",
        "count": 1000
      }
    ]
  },
    {
    "onu_id": 200,
    "message": [
      {
        "id": 12,
        "name": "PLOAM_EMPTY_MESSAGE",
        "bin": "b\\x14",
        "count": 2000
      },
      {
        "id": 18,
        "name": "BER_Interval",
        "bin": "b\\x18",
        "count": 50
      }
    ]
  }
]
```

Výpis A.3: Struktura dat pro zobrazení záhlaví rámce GPON

```
{
  "Psync": 3064672736,
  "Identification": {
    "FEC": 1,
    "Reserved": 0,
    "SuperframeCounter": 231644900
  },
  "PLOAMdownstream": {
    "ONUid": 255,
    "MessageID": 11,
    "Data": "AAAAAAAAAAAAAA==",
    "CRC": 158
  },
  "BIP": 66,
  "Plend": [
    {
      "Blen": 6,
      "Alen": 0,
      "CRC": 245
    },
    {
      "Blen": 6,
      "Alen": 0,
      "CRC": 245
    }
  ],
  "Bwmap": [
    {
      "AllocID": 0,
      "Flags": 0,
      "StartTime": 0,
      "StopTime": 79,
      "CRC": 234
    },
    {
      "AllocID": 1024,
      "Flags": 0,
      "StartTime": 80,
      "StopTime": 95,
      "CRC": 251
    }
  ]
}
```

B Příklady dat Kafka spotřebitel

Výpis B.1: Příklad Kafka spotřebitel téma GPONFrames


```
{
  "Psync": 3064672736,
  "Identification": {
    "FEC": 1,
    "Reserved": 0,
    "SuperframeCounter": 231644900
  },
  "PLOAMdownstream": {
    "ONUid": 255,
    "MessageID": 11,
    "Data": "AAAAAAAAAAAAAA==",
    "CRC": 158
  },
  "BIP": 66,
  "Plend": [
    {
      "Blen": 6,
      "Alen": 0,
      "CRC": 245
    },
    {
      "Blen": 6,
      "Alen": 0,
      "CRC": 245
    }
  ],
  "Bwmap": [
    {
      "AllocID": 0,
      "Flags": 0,
      "StartTime": 0,
      "StopTime": 79,
      "CRC": 234
    },
    {
      "AllocID": 1024,
      "Flags": 0,
      "StartTime": 80,
      "StopTime": 95,
      "CRC": 251
    }
  ]
}
```

Výpis B.2: Příklad Kafka spotřebitel téma ConnectedOnus

```
{
  "active": [{
    "onu_id": 255,
    "date": "2021-05-11␣18:37:55.994931"
  }, {
    "onu_id": 0,
    "date": "2021-05-11␣18:38:00.638313"
  }, {
    "onu_id": 1,
    "date": "2021-05-11␣18:38:00.673316"
  }, {
    "onu_id": 2,
    "date": "2021-05-11␣18:38:01.064806"
  }],
  "deactivated": []
}
```

Výpis B.3: Příklad Kafka spotřebitel téma UniquePloamMessages

```
{
  "255": {
    "11": {
      "onu_id": 255,
      "ploam_message_id": 11,
      "ploam_message_name": "PLOAM_EMPTY_MESSAGE",
      "ploam_message_id_bin": "PLOAM_EMPTY_MESSAGE",
      "counter": 18
    }
  },
  "235": {
    "12": {
      "onu_id": 235,
      "ploam_message_id": 12,
      "ploam_message_name": "PLOAM_EMPTY_MESSAGE",
      "ploam_message_id_bin": "PLOAM_EMPTY_MESSAGE",
      "counter": 15
    }
  }
}
```

C Příklad webového soketu na straně serveru

Výpis C.1: Příklad webového soketu na straně serveru

```
1  class WSHandler(websocket.WebSocketHandler):
2
3      def open(self):
4          print('Connection␣established')
5
6      def on_message(self, message):
7          print('Message␣received:␣␣%s' % message)
8
9      def send_data(self):
10         print("Sending␣data")
11         self.write_message("Sending␣data")
12
13     def on_close(self):
14         print('Connection␣closed')
15
16     def check_origin(self, origin):
17         return True
```

D Příklady konfiguračních souborů serveru

Výpis D.1: Globální konfigurace serveru

```
@dataclass()
class Config:
    server: ServerConfig = ServerConfig()
    server_static: StaticConfig = StaticConfig()
    kafka: KafkaConfig = KafkaConfig()
```

Výpis D.2: Dílčí konfigurace Kafky

```
@dataclass()
class KafkaConfig:
    PORT: int = config('KAFKA_PORT', cast=int)
    IP: str = "localhost"
    FULL_ADRESS = f"{IP}:{str(PORT)}"
    # Kafka topic definition
    GPON_FRAMES = 'GPONFrames'
    CONNECTED_ONUS = 'ConnectedOnus'
    UNIQUE_PLOM_MESSAGES = 'UniquePloamMessages'
```

Výpis D.3: Dílčí konfigurace Tornado serveru

```
@dataclass()
class ServerConfig:
    """
    Basic server configuration
    """
    NAME: str = "tornado-react-kafka"
    DEBUG: bool = config('DEBUG', default=False, cast=bool)
    PORT: int = config('SERVER_PORT', cast=int)
    SECRET_KEY: str = config('SECRET_KEY')
    XSRF_COOKIES: bool = config('XSRF_COOKIES', default=True,
        cast=bool)
```

E Dokumentace API

```
příklad hlavičky dotazu:

Accept-Encoding: gzip, deflate, br
Accept-Language: cs-CZ,cs;q=0.9,en;q=0.8,sk;q=0.7
Cache-Control: no-cache
Connection: Upgrade
Cookie: csrftoken=DvU75YCV9; tabstyle=raw-tab
Host: 127.0.0.1:8080
Origin: http://127.0.0.1:8080
Pragma: no-cache
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits
Sec-WebSocket-Key: LU82FAOZdsKMaVE0P2GFHQ==
Sec-WebSocket-Version: 13
Upgrade: websocket

příklad úspěšné odpovědi: status kód: 101
příklad chybné odpovědi: status kód: 404

příklad vrácených dat viz příloha B Příklad dat Kafka spotřebitel
-------------------------------------------------------------------------------
příklad dotazu URL: ws://127.0.0.1:8080/connected-onus
příklad úspěšné odpovědi:
{
  "active": [{
    "onu_id": 255,
    "date": "2021-05-11 18:37:55.994931"
  },],
  "deactivated": []
}
-------------------------------------------------------------------------------
příklad dotazu URL: ws://127.0.0.1:8080/unique-ploam-mess
příklad úspěšné odpovědi: viz příloha B
-------------------------------------------------------------------------------
příklad dotazu URL: ws://127.0.0.1:8080/gpon-frames
příklad úspěšné odpovědi: viz příloha B
```

Výpis E.1: Ukázka dokumentace REST API pro ověření uživatele

```
Dokumentace REST API, vrající token a uživatele

URL: /login

Method: POST

URL Params : None
Data Params: {"username": admin, "password":password}

Success Response:
    Code: 200 OK
    Content: { "token":"token fkedjfkje", "user":"admin" }

Error Response:
    Code: 404 NOT FOUND
    Content: { "user not found" }
```

F Implementace WebSocket endpointů

Výpis F.1: Ukázka WebSocket - ConnectedOnu

```
class ConnectedOnuWebSocket(WebSocketHandler):
    """
    Specific handler for connected ONU
    """
    config: Config = Config()

    c = Consumer(group_id="connected", )

    tp = TopicPartition(config.kafka.CONNECTED_ONUS, 0)
    c.consumer.assign([tp])
    c.consumer.seek_to_end(tp)
    lastOffset = c.consumer.position(tp)
    c.consumer.seek_to_beginning(tp)

    @gen.coroutine
    def send_data(self):

        for message in self.c.consumer:
            if message.offset == self.lastOffset - 1:
                break
            if self.stop:
                break

            yield gen.sleep(self.config.server.TIME_DELAY)
            self.write_message(json.dumps(message.value))

        self.c.consumer.close()

        self.close()

    def on_close(self):
        self.c.consumer.close()
```