FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Klasifikace provozu přenášeného pomocí protokolu QUIC
Student:	Andrej Lukačovič
Vedoucí:	Ing. Karel Hynek
Studijní program:	Informatika
Obor / specializace:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Seznamte se s problematikou monitorování síťového provozu na úrovni paketů a tzv. síťových toků (IP Flows) a prozkoumejte specifikaci protokolu QUIC.
Nasbírejte vzorky vybraných typů QUIC provozu (dle dohody s vedoucím práce, např. streamování hudby, videa a procházení webu) v podobě zachycených paketů a rozšířených síťových toků. Zachycené vzorky anotujte a tím vytvořte anotovanou datovou sadu síťového provozu. Proveďte analýzu zachyceného provozu, zaměřte se na charakteristické vlastnosti, které je možné využít pro klasifikaci.
Navrhněte algoritmus pro rozpoznávání typů provozu přenášeného pomocí protokolu QUIC na základě charakteristik chování komunikace.
Dle návrhu vytvořte softwarový prototyp klasifikátoru.
Navržené řešení otestuje a vyhodnoťte přesnost klasifikace a výkonové parametry prototypu (např. potřebné zdroje a rychlost zpracování dat).

Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 23. ledna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Klasifikácia sieťovej komunikácie prenášanej pomocou protokolu QUIC

Andrej Lukačovič

Katedra počítačových systémov
Vedúci práce: Ing. Karel Hynek

13. mája 2021

Poďakovanie

Rád by som sa touto cestou poďakoval vedúcemu práce Ing. Karlovi Hynkovi za obetovaný čas, skvelý prístup, trpezlivosť a v neposlednej rade neutíchajúci entuziazmuz práce. Poďakovanie rovnako patrí celej mojej rodine a priateľom ktorí ma podporujú pri štúdiu na vysokej škole.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 13. mája 2021 . . . . . .

Odkaz na túto prácu

Lukačovič, Andrej. *Klasifikácia sieťovej komunikácie prenášanej pomocou protokolu QUIC*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Táto práca sa zaoberá monitorovaním sieťovej komunikácie prenášanej pomocou protokolu QUIC, následnou analýzou a návrhom z ktorého vzíde model strojového učenia slúžiaci na klasifikáciu jednotlivých tried komunikácie predikujúci s presnosťou 93 %. Teoretická časť opisuje protokol QUIC a jeho fungovanie, rovnako predstavuje monitorovaciu infraštruktúru a použité algoritmy strojového učenia. Proces tvorby dátovej sady je podrobne opísaný v druhej kapitole. Následne je na dátovej sade prevedená analýza ktorá skúma dôležité vlastnosti jednotlivých klasifikačných tried, v tejto časti rovnako skúmame hyperparametre použitých klasifikátorov. Záver práce sa venuje implementácií nástrojov, modelu strojového učenia ktorý bol vyhodnotený za najefektívnejší a metrikám ktoré poukazujú na presnosť všetkých použitých klasifikátorov.

Kľúčové slová QUIC, klasifikácia, počítačová sieť, komunikácie, sieťová bezpečnosť, rozhodovacie stromy

Abstract

This work deals with designing and implementing an algorithm capable of encrypted traffic classification of protocol QUIC into several traffic categories. The theoretical part thoroughly analyzes the specification of the QUIC protocol, its operation, and the architecture of flow-based network monitoring infrastructure. In order to create and also evaluate the algorithm, we have created a labeled dataset of QUIC communication. The dataset is then analyzed for the identification of the essential properties of individual classes. These properties are then used in the feature vector for the Machine Learning algorithm, which achieves an accuracy of more than 93 %. As a result, we implemented a prototype capable of accurate QUIC classification able to process more than 30 000 flows per second.

Keywords QUIC, classification, computer network, communication, network security, decision trees

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Podľa *Google Transparency Report*[1] je dnes viacej ako 90 % sieťovej komunikácie šifrovanej. Hoci šifrovanie prináša značné benefity v otázke súkromia užívateľov, tieto benefity je možné rovnako zneužiť v prospech škodlivých aktivít. v súlade s Európskou Agentúrou pre kybernetickú bezpečnosť [1] môže byť šifrovaná komunikácia závažným bezpečnostným rizikom pretože si s ňou nedokážu poradiť ani moderné monitorovacie nástroje.

Výhod resp. nevýhod šifrovania sú si vedomí rovnako tvorcovia škodlivého softvéru. Podľa spoločnosti Cisco až 70 %[2] malwarových rodín v roku 2020 použilo šifrovanú komunikáciu. Je preto dôležite sa šifrovanej komunikácií venovať a skúmať možnosti jej klasifikácie. V súčasnej dobe je najčastejšie doporučovaným spôsobom obrany voči nežiadúcemu obsahu skrytému za týmto typom komunikácie, dešifrovanie. Avšak nasadenie tohto princípu je limitované veľkosťou monitorovanej siete a taktiež prináša problémy spojené so zásahom do užívateľského súkromia. Jeho nasadenie je teda možné iba v malom počte striktných sieti, medzi ktoré patria napríklad korporácie.

Výzkum monitorovania šifrovanej komunikácie ale ukazuje [2], že je možné využitie tzv. postranných kanálov. Princíp je zameraný predovšetkým na skúmanie štatistických odlišností jednotlivých druhov komunikácie, na ktoré je neskôr aplikovaná technológia strojového učenia. Nástroj, ktorý klasifikuje sieťové toky bežnej komunikácie dokáže značným spôsobom zlepšiť situačné povedomie bezpečnostného experta. Ten vďaka tomu môže lepšie prioritizovať informácie v priebehu riešení bezpečnostných udalostí. V neposlednom rade je taktiež možné nástroj využívať k tvoreniu sieťových štatistík.

Cieľom tejto práce je použiť známe monitorovacie techniky a nástroje (napr. Wireshark[3]) na sledovanie sieťovej komunikácie ktorá je ďalej analyzovaná. Analýzou sa myslí skúmanie vlastností ktoré sú špecifické len pre

[1] `https://transparencyreport.google.com/https/overview`

[2] `https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.html`

[3] `https://www.wireshark.org/`

danú komunikáciu. Následne je na základe tejto analýzy vytvorený model strojového učenia ktorý sa naučí klasifikovať sieťovú komunikáciu vďaka sade trénovacích dát. Model môže byť neskôr použitý na rozpoznávanie a filtrovanie chcenej resp. nechcenej komunikácie . Práca sa zaoberá klasifikáciou komunikácie prenášanej pomocou protokolu QUIC. QUIC je pomerne mladý protokol keďže bol prvý krát predstavený v roku 2013. Radí sa do rodiny protokolov transportnej vrstvy. Jeho použitie neustále narastá. Napriek tomu že najväčším používateľom je spoločnosť Google, ho v dnešnej dobe využíva približne 5,2 %[4] webových stránok.

Štruktúra práce

Kapitola 1 popisuje teoretický základ protokolu QUIC, obsahuje základné informácie o nadviazaní, priebehu spojenia a vysvetlenie dôležitých pojmov. Následne je popísaná teoretická časť monitorovania sieťovej komunikácie a stručne opísané použité algoritmy strojového učenia.

Kapitola 2 sa zaoberá tvorbou dátovej sady. Opisuje spôsoby akými sme dáta označovali. Na konci kapitoly sa nachádzajú štatistiky ktoré obsahujú finálne počty zachytených dát.

Kapitola 3 predstavuje analýzu dátovej sady. Pri analýze dát sa kladie dôraz na hľadanie dôležitých vlastností jednotlivých tried. Ďalej sa venujeme analýze modelov strojového učenia. Každý model obsahuje hyperparametre ktorých voľba ovplyvňuje jeho výsledky, v tejto kapitole si proces hľadania týchto parametrov opíšeme.

Kapitola 4 popisuje implementačné časti našej práce, teda implementáciu automatického nástroja na zachytávanie dátovej sady, implementáciu modulov PHISTS a BSTATS pre exportovanie vlastností tokov a implementáciu modelu strojového učenia ktorý sme vyhodnotili že pracuje s najlepšou presnosťou v závislosti na rýchlosti klasifikácie.

Kapitola 5 posledná kapitola sa zaoberá vyhodnotením použitých algoritmov strojového učenia. Rovnako popisuje výkonnostné vyhodnotenie jednotlivých klasifikátorov.

[4] `https://w3techs.com/technologies/details/ce-quic`

Kapitola 1

Teoretická časť

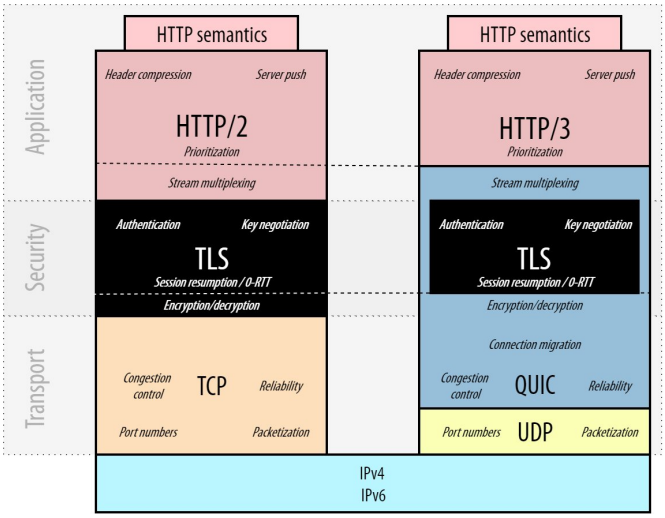
1.1 Protokol QUIC

1.1.1 Základné informácie

QUIC je univerzálny protokol transportnej vrstvy, ako v roku 2012 opísal Jim Roskind zo spoločností Google vo svojom článku [3]. V roku 2013 [4] bol predstavený verejnosti a zároveň Internet Engineering Task Force[5] (IETF). Napriek tomu že by pomenovanie QUIC mohli mnohí považovať za akronym prípadne skratku, nieje tomu tak. Pôvodný návrh [3] J. Roskinda síce pomenúval QUIC ako *Quick UDP Internet Connections*, ale neskoršie dokumentácie spoločnosti IETF nespájajú tento názov s akronymom. V roku 2015 bol podaný návrh [5] na štandardizáciu komunite IETF. V roku 2016 vznikol pod organizáciou IETF výskumný tím a zároveň bol vydaný prvý *Internet-Draft* [6] ktorý sa aktualizuje dodnes. V dnešnej dobe je spájaný prevažne s pojmom HTTP/3. Práve spojením HTTP/2 a QUIC vzniklo nové pomenovanie HTTP/3 [7].

Na Obrázku 1.1 môžeme vidieť porovnanie HTTP/3 ktoré na transportnej vrstve zamestnáva QUIC v spojení s UDP a HTTP/2 ktoré používa TCP. QUIC má niekoľko kľúčových výhod ktoré ho jednoznačne môžu aj naďalej posúvať do popredia pred ostatné transportné protokoly. Tieto výhody si v neskoršej kapitole popíšeme. Najskôr si však musíme vysvetliť niekoľko základných pojmov a konceptov.

[5] `https://www.ietf.org/`



Obr. 1.1: HTTP/2 zásobník v porovnaní s HTTP/3 zásobníkom[6]

1.1.2 Spojenie

Ako najnovší *Internet-Draft* [8] popisuje, spojenie je komunikačný kanál medzi klientom a serverom v ktorom sa pomocou zdieľaných parametrov prenášajú informácie. Každé spojenie vzniká procesom *Handshake* v ktorom si koncové uzly vymenia všetky potrebné počiatočné tajomstvá. Oba koncové uzly používajú IP adresu a port na ktorom komunikujú. Protokol QUIC navyše používa dva unikátne identifikátory spojenia:

1. Identifikátor spojenia koncového uzlu (ISKU)
2. Identifikátor spojenia počiatočného uzlu (ISPU)

Pre tieto parametre spojenia platí:

- ISKU zo strany klienta je rovnaký ako ISPU zo strany serveru a ISKU zo strany serveru je rovnaké ako ISPU zo strany klienta.

Medzi hlavné využitie identifikátorov spojenia patrí odvodzovanie počiatočných tajomstiev (viď Sekcia 1.1.3.3) a migrovanie spojenia.

[6] `https://twitter.com/programmingart/status/1369954365583339520?s=20`

1.1.2.1 Handshake

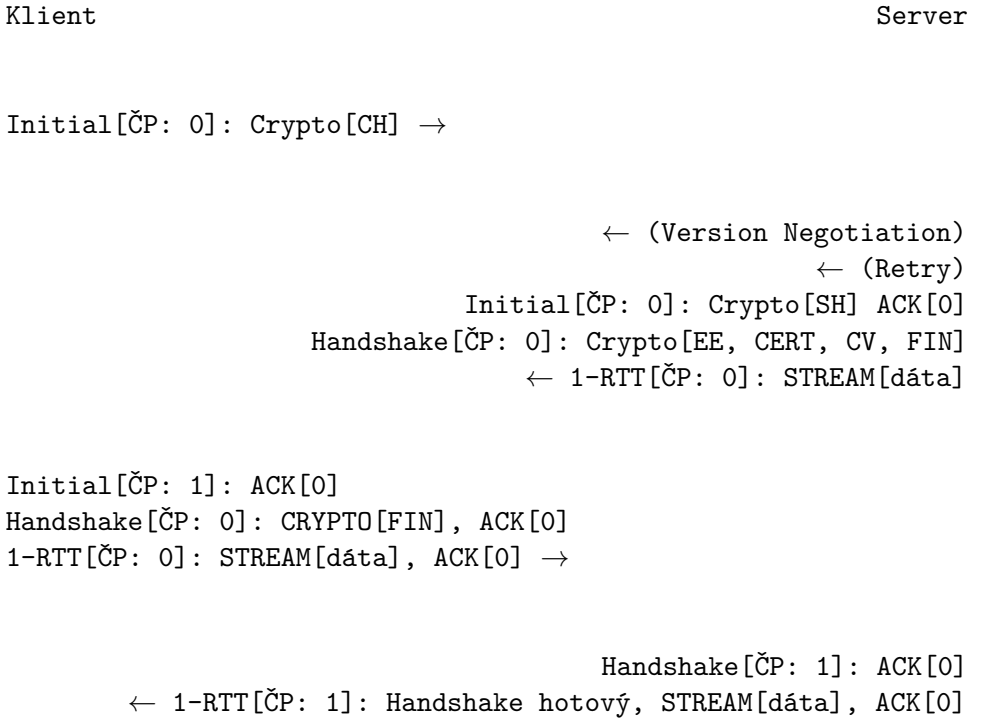
QUIC využíva kombináciu kryptografického a transportného *Handshake* [8] kvôli lepšiemu zabezpečeniu a z dôvodu zníženia réžie na nadviazanie spojenia. Na prenos dôležitých kryptografických tajomstiev počas inicializačného procesu používa CRYPTO rámce, (viď. Sekcia 1.1.4) v ktorých sú uložene informácie z TLS protokolu verzie 1.3. V *Internet Draft* [9] je opísané že protokol TLS poskytuje dva rôzne typy *Handshake*:

- **1-RTT** (Round-trip time): *Handshake* v ktorom môže klient (iniciátor spojenia) poslať dáta až po vykonaní jedného plného cyklu výmeny potrebných počiatočných informácií. Server môže poslať dáta bezprostredne po prijatí prvého *Initial* paketu, respektíve ešte pred tým ako dorazí finálna správa o identite klienta, viď Obrázok 1.2.
- **0-RTT** (Round-trip time): *Handshake* v ktorom klient používa informácie ktoré mu server poskytol v predošlom spojení k tomu aby mohol zaslať dáta hneď pri inicializácií nového spojenia, viď Obrázok 1.4. Týmto spôsobom sa znižuje réžia na prenos, teda latencia, avšak sprístupňuje sa riziko na tzv. *replay attack* [9].

1.1.2.2 1-RTT Handshake

Na Obrázku 1.2 vidíme štruktúru toho ako môže príkladné 1-RTT *Handshake* vyzerať. Počas tohto procesu je možné do UDP datagramu vložiť niekoľko QUIC paketov čo zabezpečuje že celá inicializácia spojenia môže byť vykonaná prenesením len 4 paketov. Každý riadok procesu pozostáva z typu paketu aký sa z koncového uzlu odosiela, čísla paketu, rámca ktorý sa nachádza v odoslanom pakete a krátkej informácie o tom čo rámec obsahuje.

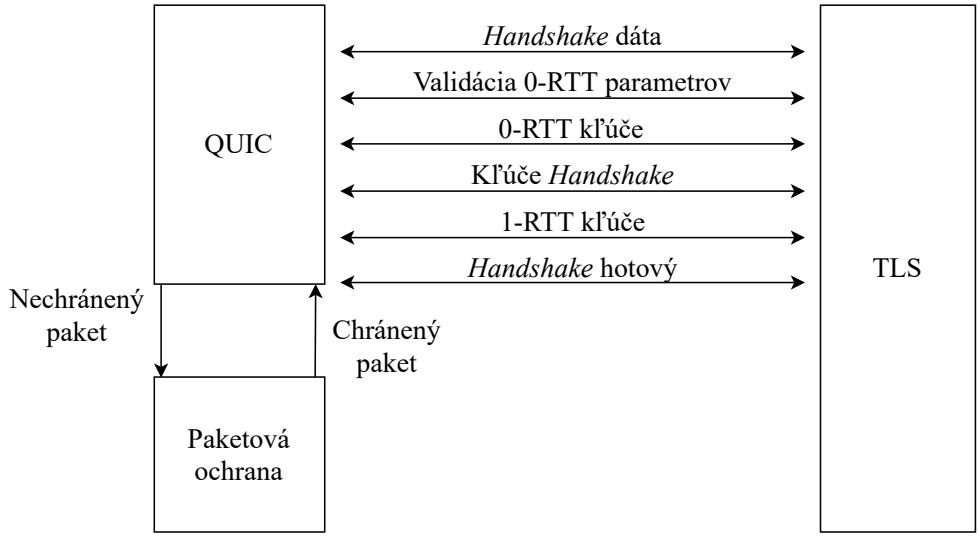
1. V prvom kroku vyšle klient paket typu *Initial*. V zátvorke vidíme číslo paketu, za ktorým nasleduje rámec typu CRYPTO, ten obsahuje správu *Client Hello*. Tento paket rovnako obsahuje Identifikátor spojenia koncového uzlu z ktorého sa odvodzujú počiatočné tajomstvá (viď Sekcia 1.1.3.3).



Obr. 1.2: Príkladné 1-RTT inicializovanie spojenia [8]

2. Server môže odpovedať niekoľkými spôsobmi:

   - *Version Negotiation* - tento spôsob odpovede klientovi naznačuje či server podporuje verziu QUIC protokolu takú akú klient používa. Veľkosť prvého paketu určuje či server odošle paket typu *Version Negotiation* alebo nie.
   - *Retry* - Následne ako server obdrží *Client Hello* správu, môže požiadať o overenie adresy. Táto požiadavka je špeciálny paket typu *Retry*, obsahuje *Token* ktorý server odošle klientovi. Klient následne musí *Token* odoslať v každom *Initial* pakete v danom spojení.
   - Môže pokračovať v nadväzovaní spojenia tým že odošle potrebné údaje a parametre klientovi, ich zoznam nájdeme v RFC 8446 [10]. O tieto informácie sa už stará protokol TLS. Na Obrázku 1.3 vidíme príkladnú komunikáciu. Protokol QUIC dostane informácie od protokolu TLS ktorý sa stará o výmenu kľúčov, certifikátov apod. Následne tieto informácie zabezpečí a odošle. V tomto momente už môže server zasielať aplikačné dáta klientovi, tieto dáta sú uložené v paketoch typu 1-RTT a sú plne zabezpečené.



Obr. 1.3: Komunikácia protokolu TLS s protokolom QUIC [9]

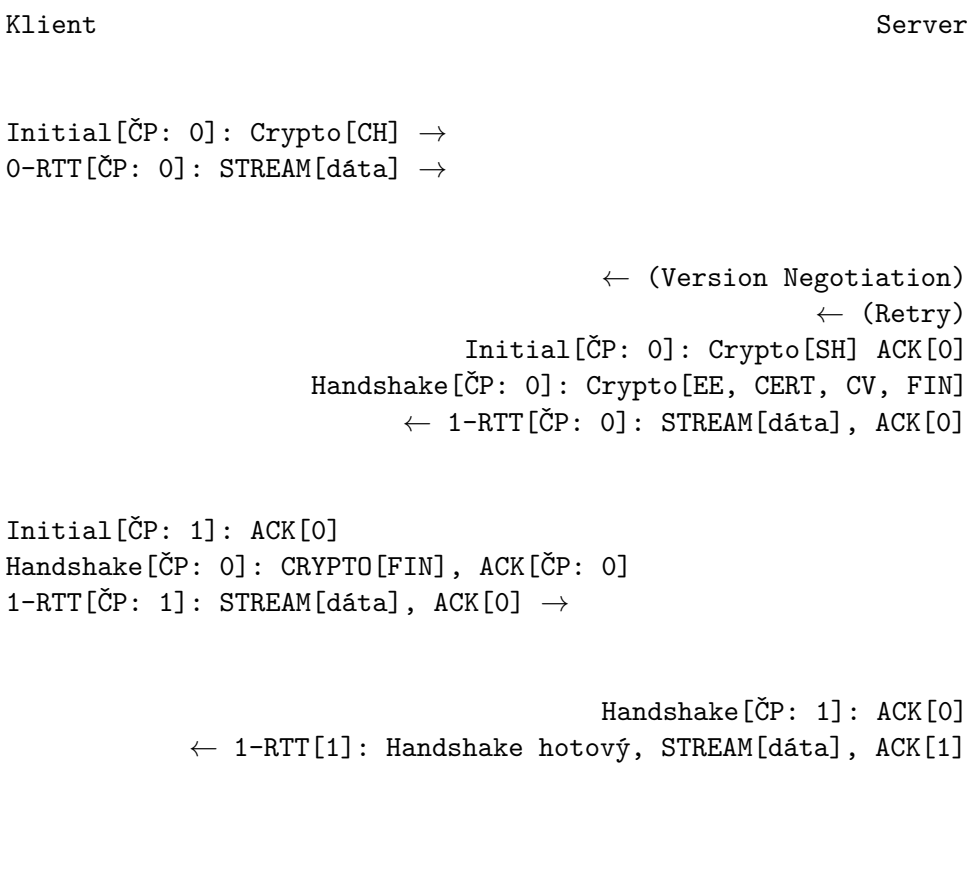
3. Klient môže odpovedať rovnako niekoľkými spôsobmi ktoré závisia od odpovede serveru:

   - V momente ak server nepodporuje verziu ktorú klient použil v prvom *Initial* pakete, môže klient buď zaslať nový *Initial* paket verzie ktorá je podporovaná oboma koncovými uzlami, alebo nadviazanie spojenia ukončí pretože spoločne nepodporujú rovnakú verziu.
   - Ak klient obdrží paket typu *Retry*, je potrebné aby obsiahnutý *Token* v tomto pakete použil v každom nasledujúcom *Initial* pakete daného spojenia. Takže sa od klienta očakáva že odošle znovu *Initial* paket ktorý bude obsahovať *Token* ktorý mu server zaslal.
   - Môže naďalej pokračovať v nadväzovaní spojenia, tým že odošle potrebné informácie späť serveru. V tomto prípade sú to už najmä potvrdzovacie informácie k *Handshake*. Rovnako v tomto štádiu inicializácie spojenia už klient môže zasielať aplikačné dáta ktoré sú uložené v paketoch typu 1-RTT, so silným zabezpečením.

4. V poslednom kroku je *Handshake* už takmer hotový. Server odošle potvrdenie a správu o dokončení. Rovnako môže zaslať aplikačné dáta ako v 3. bode. Po tejto odpovedi je inicializácia dokončená a komunikácia môže voľne prebiehať.

1.1.2.3 0-RTT Handshake

Obrázok 1.4 ukazuje ako môže prebiehať 0-RTT *Handshake*. Najdôležitejší rozdiel oproti 1-RTT, ktorý sa podieľa na zvýšení rýchlosti resp. znížení oneskorenia je že klient môže zaslať dáta bezprostredne po odoslaní *Initial* paketu. Tento fakt sa opiera o parametre ktoré už medzi klientom a serverom boli dohodnuté v predošlom spojení. Dáta ktoré sú odoslané počas inicializácie spojenia klientom sú obsiahnuté v pakete typu 0-RTT. Jednotlivé kroky nadviazania spojenia sú podrobnejšie opísané v Sekcii 1.1.2.2.



Obr. 1.4: Príkladné 0-RTT inicializovanie spojenia [8]

1.1.3 Pakety

Koncové uzly používajúce protokol QUIC komunikujú v spojení pomocou paketov. Pakety sú spracované a následne vložené do UDP datagramov. Majú jednoduchú štruktúru, obsahujú hlavičku a jeden alebo viacej rámcov. *Internet draft* [8] opisuje že pakety sa delia v základe na tie s dlhou a krátkou hlavičkou.

Pakety s dlhou hlavičkou:

- **Initial**: obsahujú prvé správy inicializácie spojenia.
- **0-RTT**: v tomto kontexte sa jedná o typ paketu nie o typ *Handshake*, nesú aplikačné dáta ktoré sú prenášané pri inicializácií spojenia.
- **Retry**: obsahujú *Token* vytvorený serverom.
- **Handshake**: nesú informácie týkajúce sa *Handshake*, teda prevažne zdieľané tajomstvá.
- **Version Negotiation**: používa iba server, obsahujú správy o dohode používanej verzie.

Pakety s krátkou hlavičkou:

- **1-RTT**: v tomto kontexte sa jedná o typ paketu nie o typ *Handshake*, je to jediný typ paketu ktorý obsahuje krátku hlavičku, slúži na prenos dát po tom ako je spojenie nadviazané (čiastočne aj počas nadväzovania viď Obrázok 1.2).

Krátka hlavička je implementovaná najmä kvôli dosiahnutiu čo najmenšej réžie. Teda po inicializácií spojenia a dohode potrebných informácií na prenos dát už nieje potrebne prenášať v hlavičke veľké množstvo informácií. Na Obrázku 1.5 môžeme vidieť príklad krátkej a dlhej hlavičky paketu. Isté časti hlavičiek sú maskované aby sa zachovala integrita a zvýšila bezpečnosť. Spôsob maskovania je opísaný v Sekcii 1.1.3.2.

```
Initial paket {
    Hlavičkový formulár (1) = 1,
    Pevný bit (1) = 1,
    Dlhý typ paketu (2) = 0,
    Rezervované bity (2),                    # Zamaskované
    Dĺžka čísla paketu (2),                  # Zamaskované
    Verzia (32),
    Dĺžka ISKU (8),
    Identifikátor spojenia koncového uzlu (0..160),
    Dĺžka ISPU (8),
    Identifikátor spojenia počiatočného uzlu (0..160),
    Dĺžka token-u (i),
    Token (..),
    Dlžka (i),
    Číslo paketu (8..32),                    # Zamaskované
    Chránené dáta (0..24),                   # Preskočená časť
    Chránené dáta (128),                     # Vzorková časť
    Chránené dáta (..),                      # Zvyšok dát
}

1-RTT paket {
    Hlavičkový formulár (1) = 1,
    Pevný bit (1) = 1,
    Spin bit (1),
    Rezervované bity (2),                    # Zamaskované
    Fáza kľúča (1),                          # Zamaskované
    Dĺžka čísla paketu (2),                  # Zamaskované
    Identifikátor spojenia koncového uzlu (0..160),
    Číslo paketu (8..32),                    # Zamaskované

    Chránené dáta:
    Chránené dáta (0..24),                   # Preskočená časť
    Chránené dáta (128),                     # Vzorková časť
    Chránené dáta (..),                      # Zvyšok dát
}
```

Obr. 1.5: *Initial* paket s dlhou a 1-RTT paket s krátkou hlavičkou [8]

1.1.3.1 Čísla paketov

Ako na Obrázku 1.5 vidíme, hlavičky paketov obsahujú rôzne typy informácií, jednou z najdôležitejších je číslo paketu. Toto číslo sa spája s úrovňou kryptografického zabezpečenia a je použité pri generovaní *Nonce* (viď. Obrázok 1.6). Každý koncový uzol udržuje čísla paketov ktoré sú prijaté a odoslané. Toto číslo sa následne delí do troch číselných priestorov:

1. **Initial**: všetky čísla *Initial* paketov sú v tomto priestore.
2. **Handshake**: v tomto priestore sa nachádzajú čísla všetkých *Handshake* paketov.
3. **Dáta aplikácií**: všetky pakety ktoré obsahujú dáta aplikácií spadajú do tohto priestoru.
4. **Version negotiation** a **Retry** pakety toto číslo neobsahujú.

Typ paketu	**Kategória šifrovacích kľúčov**	**Číselný priestor**
Initial	Počiatočné tajomstvá	Initial
0-RTT	0-RTT	Dáta aplikácií
Handshake	Handshake	Handshake
Retry	Retry	-
Version Negotiation	-	-
1-RTT	1-RTT	Dáta aplikácií

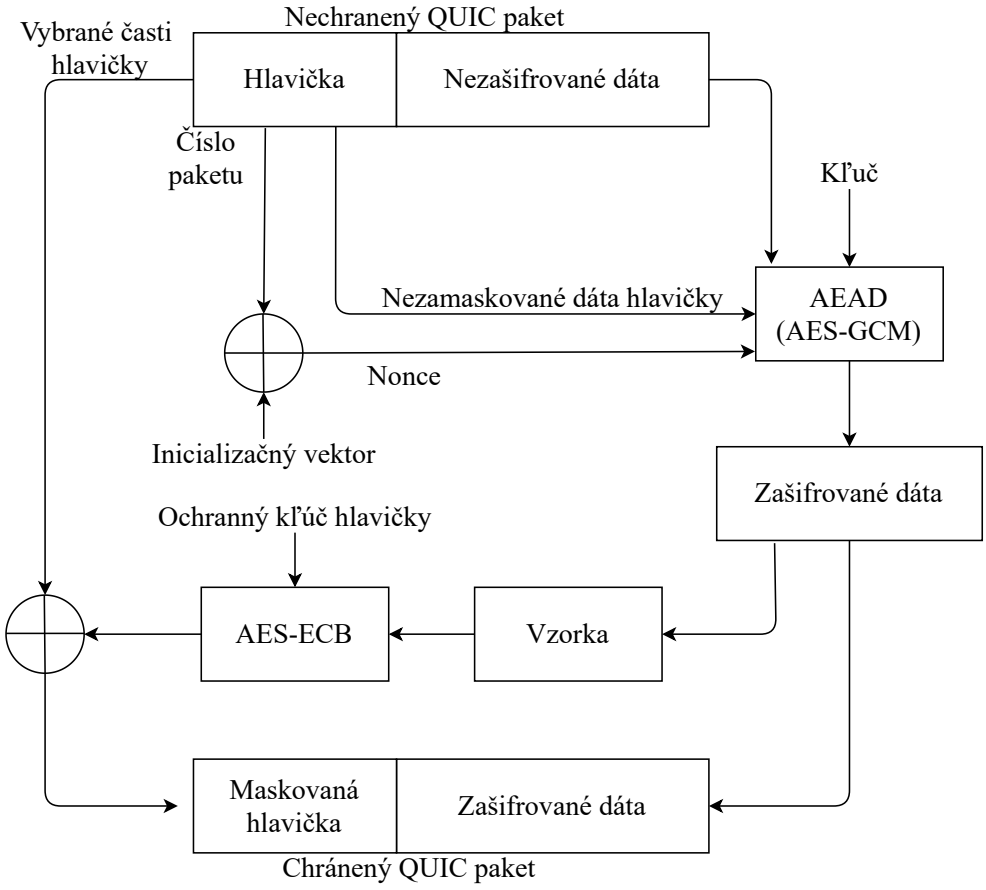
Tabuľka 1.1: Ochrana paketov v súvislosti s číselným priestorom [9]

1.1.3.2 Ochrana paketov

Protokol QUIC sa snaží vytvoriť čo najväčšiu ochranu prenášaných dát a zároveň zabezpečiť integritu [9]. Existuje niekoľko úrovní kryptografickej ochrany. Rozdiely medzi rôznymi úrovňami zabezpečenia spočívajú v tom že sa líši spôsob generovania kryptografických kľúčov. V Tabuľke 1.1 môžeme tieto úrovne vidieť zároveň vidíme prepojenie číselného priestoru paketov s ich zabezpečením. Pakety typu *Handshake*, *0-RTT* a *1-RTT* sú šifrované spoločnými tajomstvami ktoré sú dohodnuté počas inicializácie spojenia. Pakety typu *Initial* a *Retry* sú rovnako zabezpečené, avšak šifrovacie kľúče sú odvodené od viditeľných parametrov, tým pádom je zabezpečenie slabšie. Pakety typu *Version Negotiation* zabezpečené niesu [9].

Na Obrázku 1.6 vidíme proces ktorý podstupuje takmer každý paket. Nezašifrovaný paket je najskôr rozdelený na hlavičku a dáta. Následne je použitý šifrovací algoritmus, v tomto prípade *Advanced Encryption Standard with Galois/Counter Mode* (AES-GCM). Na šifrovanie potrebujeme niekoľko vstupných parametrov:

- Z hlavičky sa použijú príslušné nezamaskované atribúty.
- Nonce vznikne ako XOR Čísla paketu a Inicializačného vektoru.
- Kľuč je vygenerovaný na základe kryptografickej úrovne do ktorej daný typ paketu spadá.



Obr. 1.6: Authenticated Encryption with Associated Data [11]

Po zašifrovaní sa časť týchto dát odoberie ako vzorka, tá slúži ako vstup pre ďalší šifrovací algoritmus, v tomto prípade *Advanced Encryption Standard with Electronic Codebook* (AES-ECB). Kľúč hlavičky je odvodený od šifrovacieho kľúča danej kryptografickej úrovne a inicializačného vektoru. Na výstup tohto algoritmu je následné použitá operácia XOR spolu s vybranými atribútmi hlavičky, napríklad čísla paketu. Týmto spôsobom sa časť hlavičky zamaskuje.

1.1.3.3 Špeciálne prípady ochrany paketov

Na pakety typu *Initial* je rovnako použitý šifrovací algoritmus *Authenticated Encryption with Associated Data* z Obrázku 1.6, avšak tento proces musí byť mierne odlišný keďže sa server a klient zatiaľ nedohodli na šifrovacích kľúčoch.

- Počiatočné tajomstvo je odvodené od tzv. soli (konštanta ktorá je definovaná v protokole) a identifikátoru spojenia koncového uzlu (ISKU). Z neho sú ďalej vygenerované počiatočné tajomstvá serveru a klienta z ktorých sú následne odvodené kľúče a inicializačný vektor.

Na Obrázku 1.7 môžeme vidieť pseudokód toho ako sa v prípade *Initial* paketov odvodzujú počiatočné tajomstvá.

```
počiatočná_soľ = 0x38762cf7f55934b34d179ae6a4c80cadccbb7f0a
počiatočné_tajomstvo = HKDF-Extract(počiatočná_soľ,
                                    ISKU)


počiatočné_tajomstvo_k = HKDF-Expand-Label(počiatočné_tajomstvo,
                                           "client in", "",
                                           Hash.length)
počiatočné_tajomstvo_s = HKDF-Expand-Label(počiatočné_tajomstvo,
                                           "server in", "",
                                           Hash.length)
```

Obr. 1.7: Inicializačné tajomstvá

- Počiatočná soľ je vstupná konštanta.
- Identifikátor spojenia koncového uzlu je náhodné číslo zvolené klientom.
- „client in“ respektíve „server in“ sú konštantné reťazce.
- Hash.length je dĺžka *hash* funckie (v tomto prípade sa používa SHA-256) a teda hodnota je 256.

Podľa [12] použité funkcie sú:

- *HMAC-based Key Derivation Function-Extract*
- *HMAC-based Key Derivation Function-Expand-Label*

1.1.4 Rámce

Rámce sú entity ktoré obsahujú samotné dáta. Tieto dáta nemusia prislúchať len dátam aplikácií, môžu to byť napríklad informácie potrebné na šifrovanie správ, rôzne stavové vlastnosti spojenia a podobne. Rámce sú prenášané v paketoch, ich veľkosť musí byť taká aby odpovedali veľkosti paketu. Teda nieje možné aby jeden rámec bol rozdelený a následne poslaný v dvoch paketoch. Zároveň v jednom pakete môže byť prenesených aj viacej rámcov.

Na Obrázku 1.8 vidíme klasickú štruktúru rámca QUIC protokolu. Medzi atribúty patrí:

- *Frame Type*: hovorí nám o aký rámec sa jedná. Príkladom je Tabuľka 1.2.
- *Type-Dependant Fields*: tento atribút je závislý od toho aký typ rámca je práve použitý.

```
Rámec {
    Frame Type              (),
    Type-Dependant Fields   (...),
}
```

Obr. 1.8: Všeobecná štruktúra rámca

V Tabuľke 1.2 vidíme typy rámcov ktoré protokol QUIC používa, podľa *Internet Draft* [8] pre ich využitie platí:

- **PADDING:** bez dôležitejšieho využitia, používa sa len na manipuláciu veľkosti paketov, respektíve na ich zväčšenie.
- **PING:** slúži na zistenie dostupnosti komunikujúcej strany.
- **ACK:** rámce typu ACK obsahujú potvrdenia o prijatí a spracovaní paketov.
- **RESET_STREAM:** náhle ukončenie odosielania *stream* dát.
- **STOP_SENDING:** koncový uzol vyšle STOP_SENDING aby informoval druhú komunikujúcu stranu že prijaté dáta budu zahodené.

- **CRYPTO:** obsahuje informácie o kryptografickom *Handshake*.
- **NEW_TOKEN:** vyšle server klientovi v momente keď vyžaduje aby klient používal v inicializačných paketoch nový *Token*.
- **STREAM:** obsahuje stream dáta aplikácií.
- **MAX_DATA:** obsahuje informáciu o maximálnom počte dát ktoré môžu byť poslané v celom spojení.
- **MAX_STREAM_DATA:** obsahuje informáciu o maximálnom počte dát ktoré môžu byť prenesené v jednom *stream*.
- **MAX_STREAMS:** informujú komunikujúci uzol o maximálnom počte *streams* ktoré môžu byť otvorené.
- **DATA_BLOCKED:** obsahuje informáciu o tom že odosielateľ nedokázal odoslať dáta kvôli obmedzeniu spojenia (typicky obmedzenie maximálneho počtu dát).
- **STREAM_DATA_BLOCKED:** obdobné ako predošle, akurát sa jedná o obmedzenie na úrovni *stream* dát.
- **STREAMS_BLOCKED:** obdobné ako predošlé, akurát sa jedná o obmedzenie na úrovni počtu otvorených *streams*.
- **NEW_CONNECTION_ID:** obsahuje informáciu o novom identifikátore spojenia pre komunikujúcu druhú stranu.
- **RETIRE_CONNECTION_ID:** informácia že komunikujúci uzol už naďalej nebude používať identifikačné číslo spojenia ktoré bolo pridelené druhou stranou.
- **PATH_CHALLENGE:** používa sa pri migrácií spojenia, testuje dosiahnuteľnosť druhej strany.
- **PATH_RESPONSE:** odpoveď na PATH_CHALLENGE
- **CONNECTION_CLOSE:** informácia o uzavretí resp. skončení spojenia.
- **HANDSHAKE_DONE:** prázdny rámec bez obsahu, indikuje potvrdenie zo strany serveru o podaní rúk.

Hodnota typu	Typ rámca
0x00	PADDING
0x01	PING
0x02-0x03	ACK
0x04	RESET_STREAM
0x05	STOP_SENDING
0x06	CRYPTO
0x07	NEW_TOKEN
0x08-0x0f	STREAM
0x10	MAX_DATA
0x11	MAX_STREAM_DATA
0x12-0x13	MAX_STREAMS
0x14	DATA_BLOCKED
0x15	STREAM_DATA_BLOCKED
0x16-0x17	STREAMS_BLOCKED
0x18	NEW_CONNECTION_ID
0x19	RETIRE_CONNECTION_ID
0x1a	PATH_CHALLENGE
0x1b	PATH_RESPONSE
0x1c-0x1d	CONNECTION_CLOSE
0x1e	HANDSHAKE_DONE

Tabuľka 1.2: Typy rámcov

1.1.5 Stream

Stream je usporiadaná postupnosť bajtov dát ktoré chceme preniesť medzi koncovými uzlami. Jedná sa najmä o dáta aplikácií. V základe sa delí na dve kategórie jednosmerný a obojsmerný. Jednosmerný *stream* slúži na prenos dát v jednom smere a teda od iniciátora ku koncovému uzlu. Obojsmerný umožňuje prenos dát v oboch smeroch. *Stream* je prenášaný v rámcoch. Typický príklad STREAM rámcu môžeme vidieť na Obrázku 1.9. Prvým atribútom je *stream type*, táto hodnota nieje jednotná pretože hodnoty posledných 3 bitov sa môžu meniť a síce definujú:

- Prvý bit sprava alebo inak *FIN bit* nám definuje či momentálny rámec ukončuje *stream*, ak áno finálna dĺžka je spočítaná ako súčet premennej *Length* a *Offset*.
- Druhý bit sprava alebo inak *LEN bit* nám definuje či je prítomná premenná *Length*.

- Tretí bit sprava a síce posledný premenný, inak nazývaný aj *OFF bit* nám definuje či je prítomná hodnota premennej *Offset*. Ak je tento bit nastavený na 0 a teda hodnota *Offset* nieje prítomná znamená to že tento STREAM rámec obsahuje prvé bajty daného *stream*.

```
STREAM rámec {
    Type        () = 0x08..0x0f,
    Stream ID   (),
    [Offset     ()],
    [Length     ()],
    Stream data (..),
}
```

Obr. 1.9: Stream rámec

- *Stream ID*: tento atribút obsahuje unikátny identifikátor. Slúži na rozlíšenie a zároveň priradenie prijatých *stream* dát. Jeho využitie prináša výhodu *Stream multiplexing* (viď Sekcia 1.1.6).
- *Offset*: hodnota premennej *Offset* nám indikuje istú polohu. Keď si predstavíme *stream* ako postupnosť 0 a 1, je zrejmé že táto postupnosť môže byť príliš dlhá a teda sa nezmestí do jediného rámca. Preto musíme niesť informáciu o indexe na ktorom momentálne prenášané dáta začínajú. Táto informácia je uložená ako celé číslo v premenne *Offset*, môže nadobúdať len kladné hodnoty a 0.
- *Length*: je dĺžka dát prenášaných v momentálnom STREAM rámci.

1.1.6 Výhody protokolu QUIC

Z vlastností opísaných v tejto kapitole vzchádza niekoľko výhod ktorými protokol QUIC disponuje [8]:

- **Migrácia spojenia**: Vďaka unikátnym identifikátorom spojenia je možné rýchle obnovenie komunikácie napríklad pri situácií *NAT rebinding* [8].
- **TLS pomocou QUIC**: Komunikácia protokolu TLS prostredníctvom protokolu QUIC, zvyšuje bezpečnosť a dokáže znížiť latenciu nadviazania spojenia.
- **Stream multiplexing**: každý *stream* obsahuje svoj unikátny identifikátor, pomocou ktorého je spracovaný na strane serveru/klienta. Teda, ak nastane chyba v jednom *stream*, tak ostatné môžu bez problémov pokračovať.

- **Kontrola preťaženia**: QUIC umožňuje nastaviť limity prenesených dát a tým kontrolovať preťaženie v jednotlivých spojeniach. Tento princíp môže slúžiť ako ochrana v momente ak server (alebo útočník) odosiela dáta rýchlejšie ako ich klient stíha spracovať.
- **0-RTT**: vďaka 0-RTT *Handshake* je výrazne znížená doba nadväzovania spojenia.
- **Šifrovanie a maskovanie**: Vďaka hlavičkovej ochrane a pseudo-ochrane *Initial* paketov sa zvyšuje zložitosť odpočúvania. Aj keď šifrovací algoritmus používa viditeľné parametre na odvodenie kľúčov, dešifrovanie stojí nemalý výkon v porovnaní s hodnotou informácie ktorú nám ponúkne.

1.2 Monitorovanie

S rastúcou úrovňou zabezpečenia sieťových protokolov z jednej strany prichádza vyššia úroveň bezpečnosti, no na strane druhej je zložitejšie sieťovú komunikáciu monitorovať a tým zabrániť prípadným útokom. Medzi dva najzákladnejšie prístupy monitorovania sieťovej komunikácie patrí monitorovanie na úrovni takzvaných sieťových tokov a hĺbková analýza paketov.

1.2.1 Hĺbková analýza paketov

Hĺbková analýza paketov sa od metódy monitorovania na základe sieťových tokov líši najmä tým že skúma priamo obsah paketov. Pri monitorovaní na úrovni tokov sa skúma iba obsah IP hlavičky, podľa článku [13] hĺbková analýza môže skúmať aj hlavičky vyšších vrstiev. Medzi jej výhod a nevýhody patria:

- Skúma dáta paketov tým pádom nám dokáže podať lepšie informácie o obsahu paketov a teda lepšie výsledky monitorovania.
- Náročná na výkon a teda sa znižuje rýchlosť spracovania.
- Protokol QUIC používa silné kryptografické šifrovanie a teda nieje možné túto metódu aplikovať.
- Ak šifrovacie kľúče už poznáme, je veľmi náročné ich udržať aktuálne.
- Skúmaním obsahu paketov môže zasahovať do súkromia užívateľov.

Tento postup je pri protokole QUIC takmer nepoužiteľný a to hneď z niekoľkých dôvodov. QUIC využíva silný šifrovací algoritmus a používa maskovanie hlavičky. V prípade *Initial* paketov je možné sledovať parametre a pokúsiť sa o dešifrovanie. Napriek tomu jediná informácia ktorá by mohla byť užitočná z dešifrovaného *Client Hello* je SNI [14], ktoré aj tak nemusí byť v prípade protokolu QUIC úplne prínosné, pretože pre rôzne služby môže nadobúdať rovnaké hodnoty, čo by znamenalo chybné označenie pre klasifikáciu.

1.2.2 Monitorovanie pomocou sieťových tokov

Monitorovanie pomocou sieťových tokov sa stáva čím ďalej tým viac využívané najmä vo vysoko rýchlostných sieťach s veľkým počtom prenesených dát. Táto metóda pracuje s monitorovaním na úrovni IP tokov, teda exportuje informácie z IP hlavičiek. Ako je spomenuté v RFC 7011 [15], „*Tok je sada paketov prechádzajúcich cez pozorovací bod na sieti počas určitého časového intervalu. Všetky pakety prislúchajúce do konkrétneho toku majú súbor spoločných vlastností*“. Medzi tieto vlastnosti patria parametre hlavičky, ako napríklad IP adresy, použité porty, protokol a iné.

Medzi základné výhody tohto princípu patrí:

- Aj napriek nízkej práci s obsahom paketov, toky dokážu predať veľké množstvo informácií.
- Niesu náročné na spracovanie.
- Značne nižšia výkonová náročnosť oproti hĺbkovej analýze paketov.
- Nižšia výkonová náročnosť implikuje vyššiu rýchlosť exportovania dát.
- Toky nepredstavujú veľkú hrozbu z hľadiska zásahu do súkromia užívateľa, tým že nesledujú obsah paketov.

1.2.2.1 Protokol IPFIX

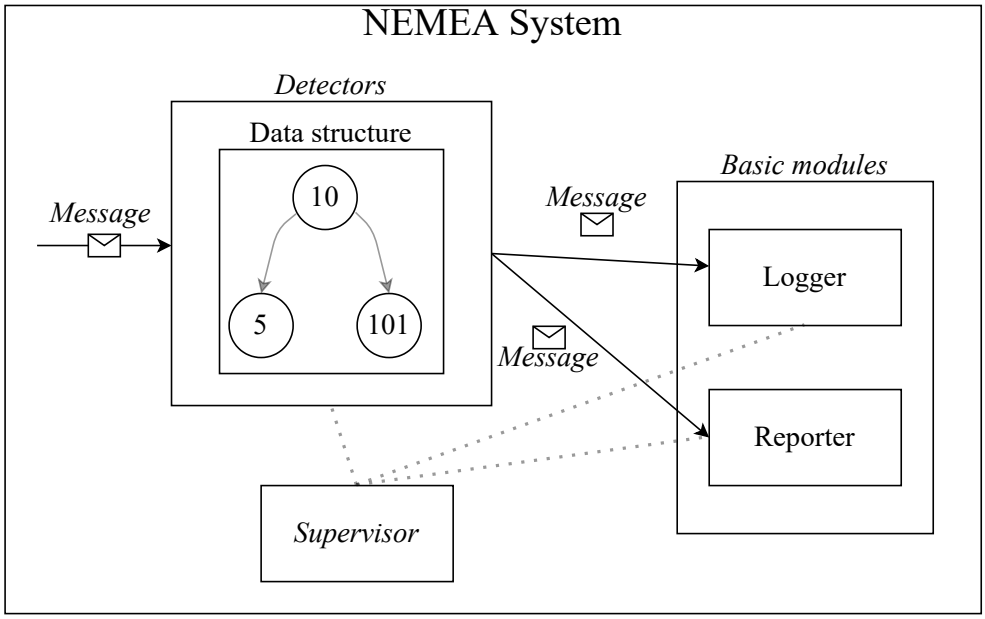
Jedným z protokolov ktorý sa zaoberá exportovaním IP tokov je protokol *IP Flow Information Export* (IPFIX) [15]. Jeho oficiálny štandard [16] bol vydaný v roku 2008 komunitou *Internet Engineering Task Force*. Exportované dáta tokov môžu obsahovať veľké množstvo informácií ako vidíme v [17].

1.2.2.2 NEMEA

NEMEA (*Network Measurements Analysis*) [18] je detekčný systém používaný na analýzu sieťovej komunikácie pomocou IP tokov. Celý systém sa skladá z niekoľkých modulov ktoré sú určené na spracovávanie alebo vyhodnocovanie dát. Na Obrázku 1.10 vidíme jednoduchú štruktúru toho ako NEMEA systém vyzerá.

- *Detectors*: sú prevažne moduly ktoré sa zaoberajú detekciou sieťovej komunikácie, snažia sa rozpoznávať rôzne typy chýb/útokov.
- *Basic modules*: sú základné moduly ktoré sa zaoberajú spracovaním dát, jedným z nich je aj modul IPFIXProbe.
- *Supervisor*: stará sa o plynulý chod modulov podľa zadanej konfigurácie.

- *Message*: sú rôzne typy dát či už dáta v podobe sieťových tokov pripravené na analýzu pomocou *Detectors*, alebo dáta potrebné k spracovaniu pomocou *Basic modules*.



Obr. 1.10: NEMEA systém [19]

1.2.2.3 IPFIXProbe

IPFIXProbe[7] je jedným z modulov detekčného systému Nemea. Jeho úlohou je exportovať sieťové toky obsahujúce vybrané informácie z PCAP súborov alebo sieťového rozhrania. V našej práci pracujeme s dátami ktoré sú exportované v nasledujúcej podobe:

1. **Základné informácie**

   Exportované toky s týmito parametrami obsahujú základné informácie, v Tabuľke 1.3 môžeme vidieť časť z nich. V skutočnosti exportované dáta obsahujú ešte pár parametrov naviac, avšak tie neboli použité v tejto práci.

[7] `https://github.com/CESNET/ipfixprobe`

Parameter Toku	Popis
DST_IP	IP adresa cieľového uzlu
SRC_IP	IP adresa počiatočného uzlu
BYTES	Počet bajtov prenesených v smere klient → server
BYTES_REV	Počet bajtov prenesených v smere server → klient
TIME_FIRST	Prvá časová značka toku
TIME_LAST	Posledná časová značka toku
PACKETS	Počet paketov prenesených v smere klient → server
PACKETS_REV	Počet paketov prenesených v smere server → klient
DST_PORT	Číslo portu cieľového uzlu
SRC_PORT	Číslo portu počiatočného uzlu

Tabuľka 1.3: Základné parametre IP tokov

2. **Paketové histogramy**

   Histogramy obsahujú informácie o tom ako sú veľkostne rozdelené pakety a medzipaketové časy v tokoch. Tieto parametre si môžeme predstaviť ako 8 dimenzionálny vektor, kde každá dimenzia prislúchala jednej paketovej resp. časovej skupine. Skupiny sú rozdelené nasledovne:

Skupina	Veľkosť	Medzipaketový čas
0	0 - 15 B	0 - 15 ms
1	16 - 31 B	16 - 31 ms
2	32 - 63 B	32 - 63 ms
3	64 - 127 B	64 - 127 ms
4	128 - 255 B	128 - 255 ms
5	256 - 511 B	256 - 511 ms
6	512 - 1023 B	512 - 1023 ms
7	1024 B a viac	1024 ms a viac

Tabuľka 1.4: Veľkostné skupiny kategórie Histogramy

Parameter Toku	Popis
D_PHISTS_IPT	Histogram medzipaketových časov v smere server → klient
D_PHISTS_SIZES	Histogram paketových veľkostí v smere server → klient
S_PHISTS_IPT	Histogram medzipaketových časov v smere klient → server
D_PHISTS_SIZE	Histogram paketových veľkostí v smere klient → server

Tabuľka 1.5: Exportované Histogramy

3. **Burst**

   Burst je časť toku v ktorej sa prenieslo určité množstvo paketov v konkrétnom smere tak aby medzi paketmi neboli príliš veľké medzipaketové časy. Každý *burst* je tvorený aspoň 3 paketmi a zároveň medzi paketmi nesmú byť medzipaketové časy dlhšie ako 1000ms. Maximálny počet *bursts* pre jeden tok v oboch smeroch je 15. V Tabuľke 1.6 vidíme informácie ktoré *bursts* obsahujú.

Parameter Toku	Popis
SBI_BRST_PACKETS	Počet paketov prenesených v i-tom *burst*, v smere klient → server
SBI_BRST_BYTES	Počet bajtov prenesených v i-tom *burst*, v smere klient → server
SBI_BRST_TIME_START	Počiatočný čas i-tom *burst* v smere klient → server
SBI_BRST_TIME_STOP	Koncový čas i-tom *burst* v smere klient → server
DBI_BRST_PACKETS	Počet paketov prenesených v i-tom *burst*, v smere server → klient
DBI_BRST_BYTES	Počet bajtov prenesených v i-tom *burst*, v smere server → klient
DBI_BRST_TIME_START	Počiatočný čas i-tom *burst* v smere server → klient
DBI_BRST_TIME_STOP	Koncový čas i-tom *burst* v smere server → klient

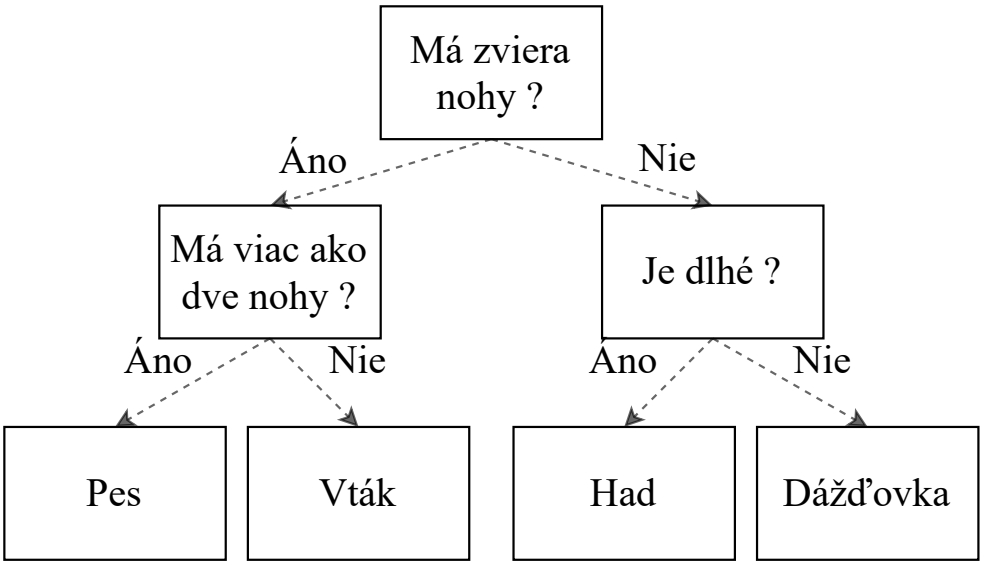
Tabuľka 1.6: Exportované bursts dáta

1.3 Použité algoritmy stojového učenia

Finálnou časťou našej práce je vytvoriť klasifikátor ktorého úlohou bude rozpoznávať jednotlivé triedy komunikácie prenášané pomocou protokolu QUIC. V tejto sekcii si v krátkosti opíšeme nami použité algoritmy strojového učenia.

1.3.1 Decision Tree

Podľa článku [20] je *Decision Tree* alebo Rozhodovací strom algoritmus strojového učenia ktorý je založený na jednoduchej stromovej štruktúre. Skladá sa z koreňa, niekoľkých rozhodovacích uzlov a listov. Algoritmus začína od koreňa, pomocou rozhodovacích uzlov postupuje smerom k listom, kde nakoniec vstupným dátam priradí štítky. V našej práci je takáto stromová štruktúra vytvorená z vlastností ktoré skúmame v 3.1. Na Obrázku 1.11 môžeme vidieť jednoduchý model.



Obr. 1.11: Decision Tree

1.3.2 Random Forest

Podľa práce [21] je *Random Forest* štruktúra zložená z niekoľkých *Decision Tree*. Tento prístup prináša určitú náhodnosť ktorá dokáže zlepšiť presnosť klasifikácie. Konečné označovanie nieje určené len jedným *Decision Tree* ale pozostáva z najviac zastúpenej predikcie.

1.3.3 Extremly Randomized Trees

Algoritmus *Extremely Randomized Trees* pracuje obdobne ako *Random Forest*, až na dva hlavné rozdiely ktoré vychádzajú z práce [22] P. Geurtsa:

- **Vyberanie vzoriek s náhradou**: Podľa [23] sa v *Random Forest* algoritme každý *Decision Tree* učí na podmnožine vzoriek ktoré sú vybrané s náhradou z celkovej množiny vzoriek vstupných dát algoritmu. Vybraná podmnožina má ale rovnaký počet vzoriek ako celková množina vstupných dát, to znamená že niektoré vzorky sa môžu opakovať. V prípade *Extremly Randomized Trees* sa každý *Desicion Tree* učí na celej množine vstupných dát algoritmu.
- **Rozdelenie rozhodovacích uzlov**: Algoritmus *Random Forest* rozdeľuje každý rozhodovací uzol tak, že vyberie najlepšie rozdelenie z množiny možných rozdelení, avšak ako je v [22] spomenuté *Extremly Randomized Trees* vyberie náhodné rozdelenie.

	Random Forest	***Extremly Randomized Tress***
Počet *Decision Trees*	Mnoho	Mnoho
Počet vlastností na rozdelenie rozhodovacieho uzlu	Náhodná podmnožina vlastností	Náhodná podmnožina vlastností
Vyberanie vzoriek s náhradou	Áno	Nie
Ako je rozhodovací uzol rozdelený	Najlepšie rozdelenie	Náhodne

Tabuľka 1.7: Porovnanie vlastností *Random Forest* a *Extremly Randomized Trees*

1.3.4 Adaptive Boosting

Adaptive Boosting alebo *AdaBoost* je algoritmus ktorý sa snaží zlepšiť učiaci proces iného, slabšieho algoritmu [21]. V našej práci sme *Adaptive Boosting* používali s algoritmami *Random Forest* a *Extremly Randomized Trees*. *AdaBoost* si udržuje váhy jednotlivých vzoriek na základe toho či sú správne klasifikované. Ak je vzorka nesprávne klasifikovaná jej váha sa zvýši, naopak ak je správne klasifikovaná jej váha sa znižuje. Následne vzorky ktorých váha je vysoká, teda tie ktoré boli nesprávne klasifikované sú použité v ďalšom cykle učenia.

Kapitola 2

Tvorba dátovej sady

Prvým krokom tvorby sieťového klasifikátoru je zachytiť dátovú sadu ne ktorej bude vykonaná analýza a následne sa pomocou nej bude náš klasifikátor učiť a neskôr aj rozpoznávať triedy komunikácie.

2.1 Kategórie sieťovej komunikácie

Protokol QUIC sa dostáva do popredia veľmi rýchlo, práve vďaka výhodám ktoré ponúka. Avšak v súčasnej dobe medzi najväčšieho používateľa tohto protokolu patrí spoločnosť Google, keďže z prevažnej časti ona stojí za jeho zrodom. Práve preto značnú časť našich zvolených tried tvoria práve služby ktoré ponúka, zároveň sme si nie vedomí že by iné služby používali tento protokol. Zoznam tried ktoré klasifikujeme je nasledovný:

- **Video**: služba ktorá obsahuje videá a hudbu, tejto kategórií prislúcha služba Youtube.
- **Hudba**: služba veľmi podobná predchádzajúcej, ale zameraná na hudbu a videoklipy, použitá služba Youtube-Music.
- **Mapy**: služba ktorá obsahuje mapy, zástupcom tejto kategórie je Google-Maps.
- **Nahrávanie súborov**: online úložisko Google Drive., táto trieda je zameraná na nahrávanie dát **do** tohto úložiska.
- **Sťahovanie súborov**: rovnaká služba ako predošlá, trieda sa sústredí na sťahovanie dát **z** online úložiska Google Drive.

- **Prehliadanie webu**: trieda ktorá obsahuje bežné prezeranie webových stránok. Zoznam webových stránok používajúcich protokol QUIC sme použili z [24].
- **Ostatné**: táto trieda obsahuje dáta ktoré sú prenášané popri sledovaní videa, teda rôzne štatistiky, náhľady videí, *watch-time* atp.

2.2 Zachytávanie dát

Na zachytávanie dát sme použili nástroj na analýzu sieťovej komunikácie Wireshark[8], resp. jeho konzolový ekvivalent TShark[9]. Dáta z týchto nástrojov sme exportovali v podobe PCAP súborov, tieto boli následne použité ako vstup do NEMEA modulu IPFIXProbe odkiaľ boli exportované v podobe IP tokov. Dáta sme zachytávali na 2 virtuálnych strojoch s operačným systémom Microsoft Windows 10 Pro, s využitím internetového prehliadača Google Chrome v ktorom je v súčasnej dobe použitie protokolu QUIC povolené predvolené. Počas procesu zachytávania sme vytvorili nástroj ktorý umožňuje automatizovať samotný proces a tvorbu PCAP súborov (viď Sekcia 4.1).

2.3 Označovanie dát

Po zachytení dát a následnom využití *NEMEA* modulu, ktorý z PCAP súborov exportoval sieťové toky, sme museli toky označiť štítkom tak aby nám bolo zrejmé pri analýze ktorý tok patrí do ktorej triedy a zároveň, aby sme označené analyzované dáta mohli predať klasifikátoru. Pri označovaní dát sme skúmali niekoľko metód.

2.3.1 Označovanie pomocou SNI

Server Name Indication (SNI) je reťazec ktorý označuje meno serveru s ktorým sa klient snaží nadviazať spojenie. Je obsiahnutý v *Client Hello* správach protokolu TLS. Vieme že, protokol QUIC, využíva protokol TLS na vykonanie kryptografického *Handshake*. V *Initial* paketoch protokolu QUIC sú obsiahnuté CRYPTO rámce ktoré TLS správy prenášajú. Analýzou *Initial* paketov by sme sa mohli k SNI dopracovať a teda každý *Initial* paket by sme vedeli priradiť ku konkrétnemu webovému serveru. Problém nastáva však v dvoch veciach:

- Protokol QUIC používa zabezpečenie *Initial* paketov čo znamená že na to aby sme informáciu o SNI dostali, museli by sme sa snažiť tieto počiatočné pakety dešifrovať. Navzdory tomu že kľúče použité na šifrovanie *Initial* paketov sú odvodené od známych parametrov spojenia,

[8] `https://www.wireshark.org/`
[9] `https://www.wireshark.org/docs/man-pages/tshark.html`

dešifrovanie by bolo výkonnostne náročné v porovnaní s hodnotou tejto informácie.

- Pomenovania jednotlivých webových serverov v prípade spoločnosti Google a ich služieb niesu vôbec jednoznačné respektíve by sa nám mohlo stať že dve triedy by boli spojené s rovnakým SNI, čo by viedlo k nesprávnemu označeniu.

2.3.2 Označovanie pomocou ASN

Autonomous System Number (ASN) je 16-bitové číslo ktoré identifikuje autonómnu časť internetu. Ako Gao L. vo svojej práci [25] vysvetľuje, Internet je rozdelený na veľké množstvo administratívnych domén, ktoré obsahujú jeden alebo viacej autonómnych systémov. Príklady administratívnych domén môžu byť školský kampus, podniková sieť, ale aj sieť rozsiahleho *Internet Service Provider*. Každý takýto autonómny systém má svoj identifikátor, teda *Autonomous System Number*, ktorý je následne využívaný pomocou *Border Gateway Protocol*. Spoločnosť Google nieje výnimkou primárne používa autonómny systém s identifikátorom 15169[10].

- Nevhodnosť využitia tohto princípu spočíva v tom že rôzne služby spoločnosti Google spadajú do jedného autonómneho systému, teda dve rôzne triedy komunikácie by boli označené rovnakým číslom, čo by viedlo k nesprávnemu označeniu.

2.3.3 Označovanie pomocou oktetov IP adresy

Jedným z atribútov obsiahnutý v exportovaných tokoch je IP adresa klienta resp. serveru. IP adresa je číselné označenie ktoré je priradené každému zariadeniu na sieti. Rozoznávame 2 druhy adries:

- *Internet Protocol version 4*: používa 32-bitové číselné označenie. Skladá sa zo 4 oktetov.
- *Internet Protocol version 6*: pomerne nový prístup, využíva číselné označenie veľkosti 128-bitov.

[10] `https://peering.google.com/#/options/peering`

V našej práci sme používali iba označovanie adries verzie *Internet Protocol version 4*. Postupovali sme nasledovne:

1. V prvom kroku sme využili nástroj internetového prehliadača *Chrome DevTools Network*[11]. Tento nástroj ponúka informácie o tom aké dáta z akých zdrojov sa do prehliadača stiahli a zároveň ich jednoduchý náhľad. Všimli sme si podobnosť zdrojových IP adries, teda adries serverov z ktorých sa sťahovali dáta. Platí že IP adresy z ktorých sa sťahovala hudba alebo video sa líšili prevažne v poslednom oktete IP adries. To isté platí aj pre štatistické dáta, náhľady videí apod. ktoré sa sťahovali počas prehrávania.
2. Pomocou prvého bodu nám vznikol zoznam číselných označení ktorého ukážku vidíme na Obrázku 2.1, nejednalo sa o celé IP adresy, ale iba o prvé tri oktety, teda 24 bitov z IP adresy.
3. Využitím tohto zoznamu sme mohli označiť toky tak aby sme ich rozdelili na toky ktoré obsahujú stiahnuté dáta prislúchajúce prehrávaniu **Videa** či **Hudby** a na toky ktoré patria do skupiny **Ostatné**.

```
"74.125.11"   = Video
"173.194.187" = Video
"172.217.23"  = Ostatné
"217.119.123" = Music
```

Obr. 2.1: Ukážka zoznamu známych oktetov

2.3.4 Označovanie pomocou čísla portu

Na triedy **Sťahovanie/Nahrávanie súborov** sme použili proces označovania dátovej sady pomocou čísla portu. Z exportovaných tokov stačilo nájsť taký tok ktorý je špecifický a následne použiť číslo portu klienta na označenie ostatných tokov.

- Pre **Nahrávanie súborov** to znamená že sme museli nájsť tok v ktorom je prenesené veľké množstvo dát v smere od klienta k serveru a zároveň v histogramoch prevažovalo zastúpenie veľkých paketov v tomto smere. Tento tok obsahuje číslo portu klienta a číslo portu serveru. Číslo portu na strane serveru sa nemení, teda je konštantne 443. Číslo portu na strane klienta je premenlivé, avšak odosielanie dát prebiehalo na tom istom porte, tým pádom sme podla jedného toku mohli označiť ostatné. Toky, ktoré toto číslo portu neobsahovali sme zaradili do triedy **Ostatné**.

[11] `https://developer.chrome.com/docs/devtools/`

- **Sťahovanie súborov**: pri tejto triede sme postupovali rovnako ako pri **Nahrávaní súborov** avšak špecifický tok musel obsahovať veľké množstvo dát prenesených v smere od serveru ku klientovi a rovnako v tomto smere muselo prevažovať zastúpenie veľkých paketov. Následne sme podľa čísla portu klientskej strany označili ostatné toky. Toky ktoré toto číslo neobsahovali boli zaradené do triedy **Ostatné**.

2.3.5 Zhrnutie označovania dátovej sady

Finálny proces označovania dátovej sady vyzerá nasledovne:

- **Video a Hudba**: tieto triedy sme označili spôsobom ktorý je opísaný v Sekcii 2.3.3, teda pomocou prvých troch oktetov IP adresy serveru, čím vznikli aj toky patriace do triedy **Ostatné**.
- **Mapy a Prezeranie webu**: tieto triedy sme brali ako celok, takže všetky zachytené dáta z danej skupiny sme označili ako **Mapy** resp. **Prezeranie webu**.
- **Sťahovanie/Nahrávanie súborov**: na označenie týchto tried sme využili spôsob označenia pomocou čísla portu klientskej strany (viď Sekcia 2.3.4), čím vznikli aj toky prislúchajúce do triedy **Ostatné**.
- **Ostatné**: táto trieda vznikla popri označovaní predošlých tried.

2.4 Štatistiky dátovej sady

	Množstvo zachytených dát	Počet tokov
Video	19,9 GB	4291
Hudba	6,47 GB	11 861
Mapy	7,72 GB	4575
Sťahovanie súborov	23,43 GB	744
Nahrávanie súborov	25,79 GB	2067
Prezeranie webu	5,58 GB	7683
Spolu	88,89 GB	31 221

Tabuľka 2.1: Štatistiky zachytenej dátovej sady

Štatistika v Tabuľke 2.1 obsahuje čísla surovej dátovej sady. Tá bola odfiltrovaná od tokov ktoré v oboch smeroch nepreniesli aspoň 15 paketov, následne sme toky označili (čím vznikla trieda **Ostatné**). Finálna dátová sada ktorú sme analyzovali a z ktorej sme vytvorili validačnú a trénovaciu podmnožinu vyzerá nasledovne:

	Počet tokov
Video	999
Hudba	1864
Mapy	1639
Sťahovanie súborov	281
Nahrávanie súborov	322
Prezeranie webu	3478
Ostatné	3136
Spolu	11 719

Tabuľka 2.2: Odfiltrovaná dátová sada

2.4.1 Rozdelenie dátovej sady

Z každej triedy z Tabuľky 2.2 sme náhodne vybrali 1000 tokov. Tie sme rozdelili v pomere 20 % pre validačnú a 80 % pre trénovaciu. To znamená že z celkového počtu 1000 tokov pre každú triedu komunikácie je náhodne vybraných 200 tokov vo validačnéj dátovej sade a 800 v trénovacej. Pre dátové sady platí:

- **Trénovacia dátová sada**: tieto dáta sú použité v časti vyberania najlepších hyperparametrov klasifikátoru (viď Sekcia 3.2.1) a v procese učenia vybraného klasifikátoru (viď Sekcia 4.3.1).

- **Validačná dátová sada**: pre túto dátovú sadu platí že je použitá **IBA** v procese testovania resp. validácie klasifikátoru (viď Sekcia 4.3.2).

Pre jednotlivé dátové sady platí že pomery zastúpenia tried komunikácie sú balancované, to znamená že v oboch dátových sadách je rovnaký počet vzoriek z každej triedy. Tento prístup nemusí odzrkadľovať chovanie na Internete v reálnom čase, avšak snažili sme sa predísť chybe *Overfitting* a zároveň nedokážeme odhadnúť v akých pomeroch sú v reálnom čase triedy v komunikácií zastúpené.

- Napriek veľkému množstvu zachytených dát triedy **Sťahovanie súborov** a **Nahrávanie súborov** nieje počet tokov týchto tried po exportovaní z IPFIXProbe rovnaký ako počet tokov ostatných tried. Preto ďalej v práci používame metódu `SMOTE`[12] ktorá nám pomáha vygenerovať toky týchto tried tak aby rozloženie dátovej sady bolo rovnomerné. Táto metóda je použitá **IBA** na trénovaciu dátovú sadu aby sme žiadnym spôsobom neovplyvnili testovanie klasifikátoru.
- Pre triedu **Nahrávanie súborov** platí že dátovú sadu sme rozdelili nasledovne, 200 tokov patrí do validačnej dátovej sady. 101 tokov prislúcha do trénovacéj dátovej sady, tie boli následne rozšírené pomocou metódy `SMOTE`.
- Pre triedu **Sťahovanie súborov** platí že dátová sadu bola rozdelená v počtoch 200 tokov do validačnej dátovej sady. 81 tokov do trénovacej dátovej sady, tieto boli následne rozšírené pomocou metódy `SMOTE`.

[12] `https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html`

Kapitola 3

Analýza a návrh

3.1 Analýza kategórií komunikácie

Po zachytení dátovej sady môžeme začať s analýzou tokov jednotlivých tried. Analýza predstavuje proces kedy sa snažíme pre jednotlivé toky každej triedy nájsť také vlastnosti ktoré sú pre ne špecifické. Tieto vlastnosti neskôr použijeme ako vstup do klasifikátoru ktorému predáme aj označenie jednotlivých vstupov, na nich sa klasifikátor naučí toky rozpoznávať a zaradovať ich do príslušných tried. V Tabuľke 3.6 môžeme vidieť rozdelenie vlastností do troch skupín ktoré prislúchajú rozdeleniu dát exportovaných tokov zo Sekcie 1.2.2.3.

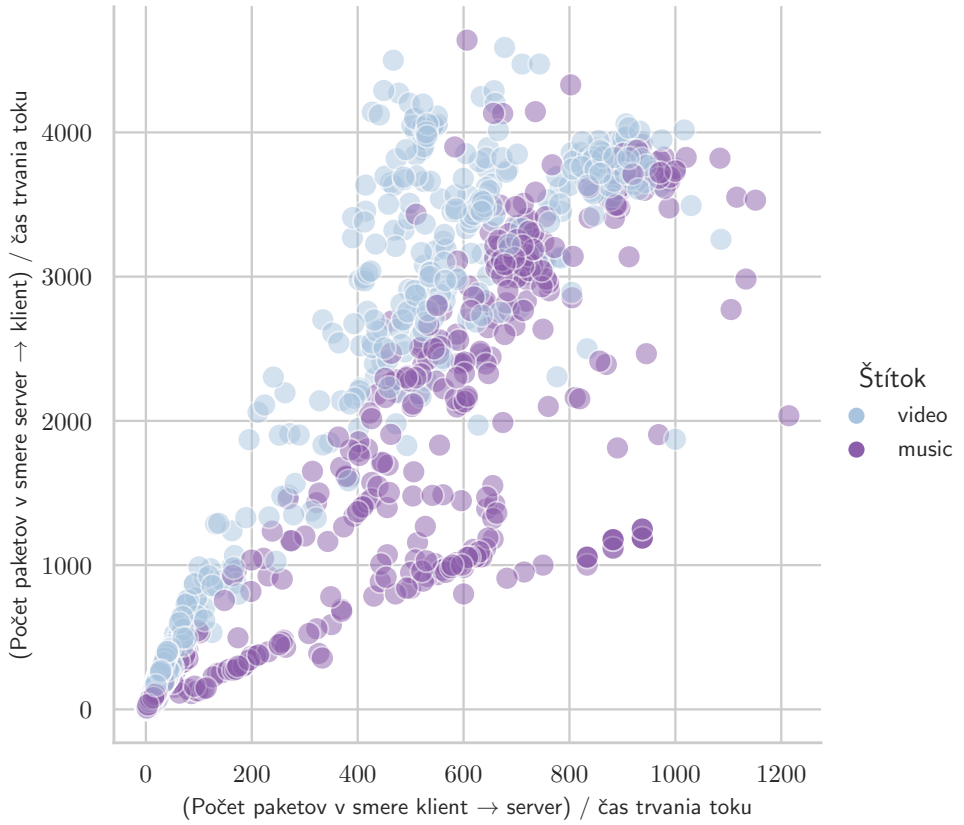
Základné	Paketová priepustnosť Bajtová priepustnosť Paketové pomery
Histogramy	Absolútne počty paketov v paketových skupinách Priepustnosti zlúčených paketových skupín Podiely paketových skupín Priepustnosť nezlúčených paketových skupín
Bursts	Minimum Maximum Medián Smerodajná odchýlka Priemer Rozptyl Kvantily

3.1.1 Základné vlastnosti

Táto skupina sa zaoberá najmä základnými vlastnosťami tokov. Patria sem:

1. **Paketová priepustnosť** - opisuje počet paketov prenesený za jednotku času, teda podiel počtu paketov k dĺžke trvania toku. Na obrázku 3.1 môžeme vidieť príklad ktorý nám pomáha rozlíšiť kategóriu **Hudba** od kategórie **Video**. Samozrejme sme rozlišovali smery paketov, tým pádom nám vznikli tri podkategórie. V smere od klienta k serveru, v smere od serveru ku klientovi a oba smery dohromady.

2. **Bajtová priepustnosť** - opisuje rovnakú charakteristiku ako paketová priepustnosť s tým rozdielom že sa zaoberajú prenesenými bajtami nie paketmi.



Obr. 3.1: Paketová priepustnosť

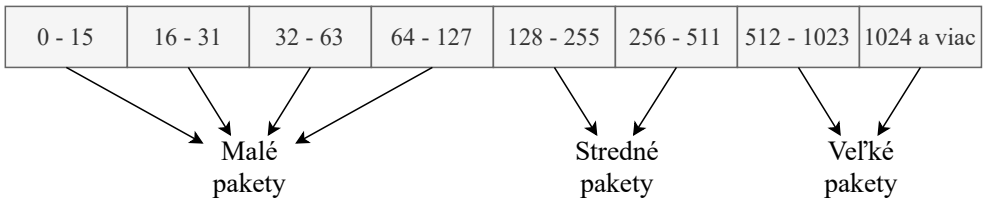
3.1.2 Pokročilé vlastnosti

Pokročilé vlastnosti môžeme rozdeliť do dvoch podskupín a to **Histogramy** a **Bursts**. V **Histogramoch** sme sledovali vlastnosti ktoré úzko súvisia s rozloženiami paketov naprieč paketovými skupinami. Venovali sme sa najmä normalizovaným podielom, teda percentám. V kategórií **Bursts** sa venujeme rozptylu, kvanilom apod. Čo je to *burst* sme si opísali v Sekcii 1.2.2.1.

3.1.2.1 Vlastnosti histogramov

Exportované dáta ktoré skúmame v tejto časti majú štruktúru ktorú môžeme vidieť v Sekcii 1.2.2.1. K ich parametrom sme pristupovali nasledovne:

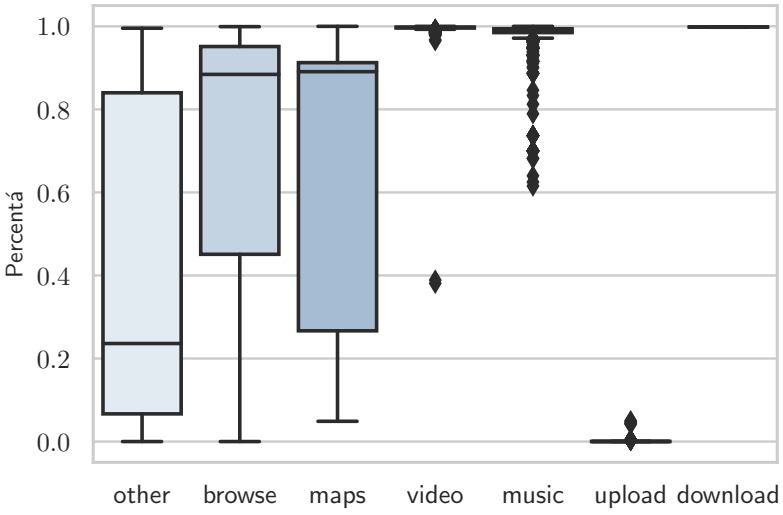
1. Sledovali sme absolútne počty v paketových skupinách pre jednotlivé toky, avšak 8-dimenzionálne vektory sme redukovali na 3-dimenzionálne, respektíve z ôsmich veľkostných skupín sme spravili tri. Názornú redukciu môžeme vidieť na Obrázku 3.2. Dôvodom takéhoto rozloženia je fakt že prvé dve skupiny veľkostí sú zastúpené len zriedka, preto sme sa zároveň rozhodli zlúčiť až prvé 4 veľkostné skupiny do jednej.



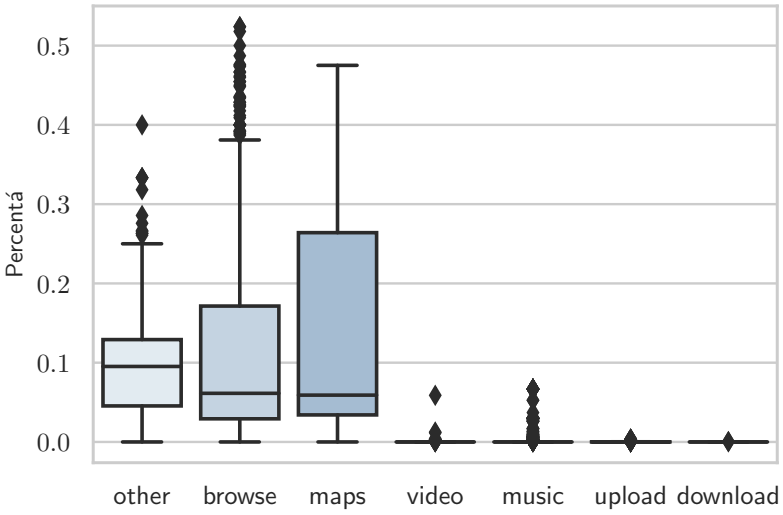
Obr. 3.2: Zlúčenie histogramových skupín

2. Pre jednotlivé skupiny z bodu 1. sme vypočítali priepustnosť. Tú sme získali ako (počet paketov v danej skupine) / (dĺžka trvania spojenia), môžeme hovoriť že toto číslo vyjadruje počet prenesených paketov z danej veľkostnej skupiny za jednotku času. Pre danú vlastnosť sme rozlišovali aj smer, teda v smere od klienta k serveru, v smere od serveru ku klientovi a súčet oboch smerov.

3. Pre zlúčené paketové skupiny z bodu 1. sme ďalej vypočítali percentuálne hodnoty v závislosti na smere. Na obrázku 3.3 vidíme ako sú rozložené percentuálne zastúpenia veľkých paketov naprieč triedami v smere od serveru ku klientovi. Na obrázku 3.4 môžeme na druhej strane vidieť zastúpenie stredných paketov v smere od klienta k serveru.



Obr. 3.3: Podiely veľkých paketov v smere server → klient



Obr. 3.4: Podiely stredných paketov v smere klient → server

4. Poslednou časťou sú priepustnosti nezlúčených paketových skupín. Teda na náš 8-dimenzionálny vektor sme použili výpočty obdobne ako v bode 2.

Na obrázku 3.3 a 3.4 vidíme podstatné rozdiely medzi dvomi podskupinami tried:

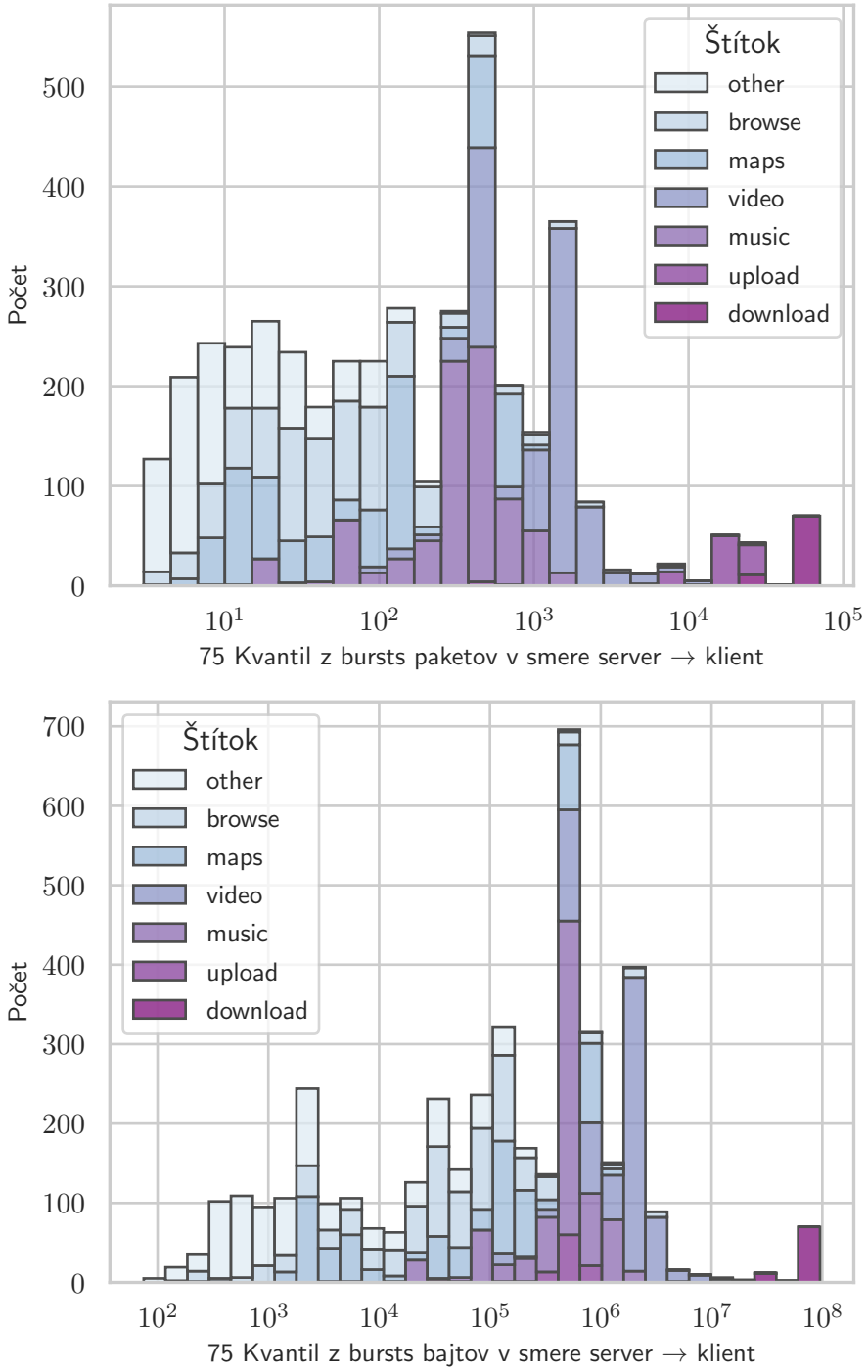
- **Video, Hudba, Nahrávanie súborov, Sťahovanie súborov**
- **Mapy, Prezeranie webu, Ostatné**

3.1.2.2 Vlastnosti bursts

Exportované dáta z kategórie **Bursts** z modulu IPFIXProbe ktorých podobu vidíme v Sekcii 1.2.2.3 si môžeme predstaviť ako vektory ktorých dimenzie sú maximálne 15. Pre každú dimenziu platí že predstavuje jeden *burst*. Skúmali sme nasledovné vlastnosti dimenzií týchto vektorov:

- **Priemer**
- **Smerodajnú odchýlku**
- **Rozptyl**
- **Minimum**
- **Maximum**
- **Medián**
- **25** a **75 Kvantil**

Na Obrázku 3.5 môžeme vidieť 75 kvantil pre počty paketov a počty bajtov jednotlivých *bursts* v smere zo serveru ku klientovi.



Obr. 3.5: Kvantily z vlastností bursts

3.1.3 Výber najlepších vlastností

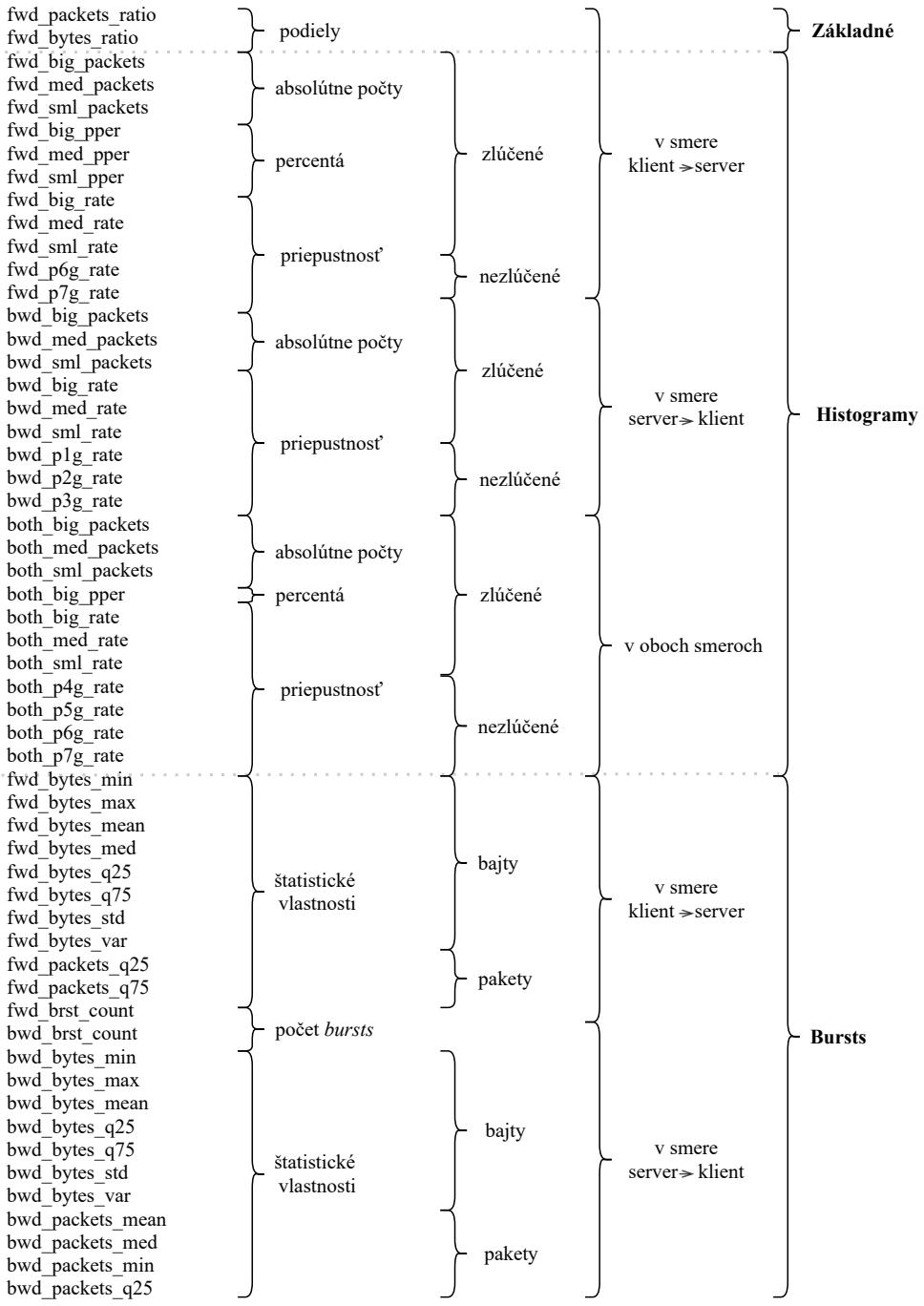
Poslednou časťou analýzy vlastností tokov jednotlivých tried je výber vlastností ktoré najviac vyhovujú pre našu klasifikáciu. Výber vlastností sa skladá z dvoch krokov:

- **Malý rozptyl**: tento krok pozostáva z odstránenia vlastností s nízkym rozptylom, pretože tie neposkytujú dostatočnú jednoznačnosť pri rozhodovaní. Na to sme použili triedu `VarianceThreshold` [13] implementovanú v knižnici *scikit-learn*.
- ***RFECV***: algoritmus *Recursive Feature Elimination with Cross-Validation*[14] sme použili na vytvorenie poradia jednotlivých vlastností. Základom algoritmu je *Decision Tree* bez špecifikovaných *hyperparametrov*.

V Tabuľke 3.6 sa nachádza konečný rozpis použitých vlastností. Ich celkový počet je 56. Najväčšie zastúpenie má kategória **Histogramy**, za ňou nasledujú **Bursts** a nakoniec kategória **Základné**.

[13] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html`

[14] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html`



Obr. 3.6: Finálny výčet vlastností

3.2 Analýza algoritmov strojového učenia

Algoritmy strojového učenia spomenuté v Sekcii 1.3 používajú hyperparametre. Hyperparametre algoritmu sú také parametre, ktoré sú zvolené pred spustením procesu učenia daného algoritmu. Jednoduchým príkladom takéhoto parametru môže byť maximálna hĺbka a počet jednotlivých *Decision Trees* v *Random Forest* algoritme. V Tabuľke 3.2 vidíme ktoré hyperparametre sme skúmali pre použité klasifikátori.

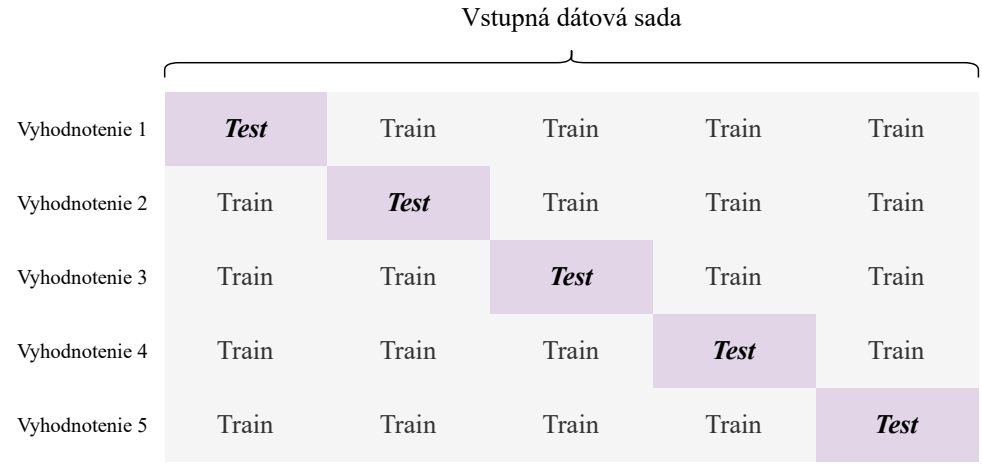
	Hyperparametre
Random Forest*, *Extremly Randomized Trees	*n_estimators*, *max_features*, *max_depth*, *min_samples_split*, *min_samples_leaf*, *bootstrap*, *criterion*
Adaptive Boosting	*n_estimators, algorithm, learning_rate*

- ***n_estimators***: parameter určujúci počet *Decision Trees* v danom algoritme. V kontexte *Adaptive Boosting* tento parameter znamená maximálny počet na ktorom je prevedený zlepšený proces učenia.
- ***max_features***: znamená maximálny počet skúmaných vlastností ktoré sú brané do úvahy v hľadaní najlepšieho rozdelenia rozhodovacieho uzlu.
- ***max_depth***: maximálna hĺbka každého *Decision Tree*.
- ***min_samples_split***: minimálne množstvo vzoriek potrebné na rozdelenie rozhodovacieho uzlu.
- ***min_samples_leaf***: minimálne množstvo vzoriek ktoré sa musia nachádzať v liste *Decision Tree*.
- ***bootstrap***: určuje či je využitý princíp **Vyberania vzoriek s náhradou** ak nie, na každý *Decision Tree* je použitá celá množina vstupných dát.
- ***criterion***: funkcia určujúca kvalitu rozdelenia rozhodovacieho uzlu.
- ***algorithm***: určuje algoritmus ktorý je použitý na zlepšený proces učenia.
- ***learning_rate***: váha aplikovaná na každý klasifikátor v každej iterácií zlepšeného učenia. Vyššia váha implikuje vyšší príspevok klasifikátoru.

3.2.1 Výber hyperparametrov klasifikátoru

Na výber najlepších hyperparametrov klasifikátoru sme použili metódu `GridSearchCV`[15] ktorá je obsiahnutá v knižnici *scikit-learn*. Jedným z hlavných parametrov tejto metódy je `param_grid`, ktorý očakáva na vstupe slovník s inicializovanými hyperparametrami. Na Obrázku 3.8 môžeme vidieť ukážku procesu výberu najlepších hyperparametrov pre *Random Forest*. Po tom ako `GridSearchCV` vyhodnotí ktoré hyperparametre sú tie najlepšie, použijeme metódu `cross_val_predict`[16], tá je založená na procedúre *Cross-Validation*, pomocou ktorej otestujeme presnosť klasifikátoru na trénovacích dátach.

- ***Cross-Validation***: je procedúra ktorá umožňuje vyhodnotiť presnosť modelu klasifikátoru na limitovanom množstve dát. Má jediný parameter k, ktorý hovorí na koľko rovnakých podmnožín sa má vstupná dátová sada rozdeliť. Po tom ako dáta rozdelíme na k-rovnakých podmnožín, je vybraná jedna ktorá slúži ako testovacia dátová sada. Na ostatných (teda k-1) podmnožinách dátovej sady sa klasifikátor učí. Po tom ako sa klasifikátor naučí je otestovaná jeho presnosť na vybranej testovacej podmnožine dátovej sady. Tento proces sa opakuje s tým že je vybraná iná testovacia podmnožina. Na Obrázku 3.7 vidíme príklad *Cross-Validation* s parametrom k=5.



Obr. 3.7: k-fold Cross-Validation, k = 5

[15] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

[16] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_predict.html`

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE

# ------------------------------------------
n_estimators = [50,100,150,200]
# ------------------------------------------
max_features = [10,30,50]
max_features.extend(['auto', 'sqrt' , 'log2'])
# ------------------------------------------
max_depth = [10,30,50]
max_depth.append(None)
# ------------------------------------------
min_samples_split = [3,6,9]
# ------------------------------------------
min_samples_leaf = [3,6,9]
# ------------------------------------------
bootstrap = [True]
# ------------------------------------------
criterion = ["gini","entropy"]
# ------------------------------------------
grid_rfc = {'n_estimators': n_estimators,
            'max_features': max_features,
            'max_depth': max_depth,
            'min_samples_split': min_samples_split,
            'min_samples_leaf': min_samples_leaf,
            'bootstrap': bootstrap,
            'criterion': criterion}
rfc_clf = RandomForestClassifier()
smote = SMOTE()
pipeline = Pipeline([('smote', smote), ('classifier', rfc_clf)])

rscv = GridSearchCV(pipeline, param_grid=grid_rfc,
                    cv = 3, n_jobs=8)
rscv.fit(features_train, labels_train)
pipeline.set_params(**rscv.best_params_)
CVpredict = cross_val_predict(pipeline, features_train,
                              labels_train, cv=5)
```

Obr. 3.8: Hľadanie hyperparametrov pre *Random Forest*

Rovnakým prístupom sme postupovali pre *Extremly Randomized Trees*. Ďalej, sme potrebovali nájsť najlepšie hyperparametre pre *Adaptive Boosting* algoritmus, ktorý zlepšuje proces učenia iného algoritmu. Na Obrázku 3.9 môžeme vidieť postup pre *Adaptive Boosting* s *Extremly Randomized Trees*. V tomto momente sme už poznali najlepšie hyperparametre pre *Extremly Randomized Trees*, preto v kóde inicializujeme triedu `ExtraTreesClassifier` s parametrom `**best_hyperparameters` ktorý obsahuje slovník najlepších hyperparametrov pre *Extremly Randomized Trees*. Rovnako na vyhodnotenie použijeme metódu `cross_val_predict` ktorej princíp sme si opísali v Sekcii 3.2.1.

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier

# ----------------------------------------
n_estimators = [x for x in range(50,201,50)]
# ----------------------------------------
algorithm = ["SAMME","SAMME.R"]
# ----------------------------------------
learning_rate = [0.5,1,1.5]
# ----------------------------------------
grid_ada = {'n_estimators': n_estimators,
            'algorithm': algorithm,
            'learning_rate': learning_rate}

etc_clf = ExtraTreesClassifier(**best_hyperparameters)
ada_boost = AdaBoostClassifier(base_estimator=etc_clf)
smote = SMOTE()

pipeline = Pipeline([('smote', smote), ('classifier', ada_boost)])
rscv = GridSearchCV(pipeline, param_grid=grid_ada,
                    cv = 3, n_jobs=8)
rscv.fit(features_train, labels_train)
pipeline.set_params(**rscv.best_params_)
CVpredict = cross_val_predict(pipeline, features_train,
                              labels_train, cv=5)
```

Obr. 3.9: Hľadanie hyperparametrov pre *Adaptive Boosting* so základom *Extremly Randomized Trees*

3.2.2 Vyhodnotenie hyperparametrov

Pre najlepšie hyperparametere algoritmov strojového učenia *Random Forest*, *Extremly Randomized Trees* a *Adaptive Boosting* použitý na oboch predchádzajúcich, platia nasledujúce hodnoty:

Hyperparametre	**Hodnoty**	**Najlepšie**
n_estimators	[50, 100, 150, 200, 250, 300]	200
max_features	[10, 30, 50, 'auto', 'sqrt', 'log2']	'auto'
max_depth	[10, 30, 50, None]	50
min_samples_split	[3, 6, 9]	6
min_samples_leaf	[3, 6, 9]	3
bootstrap	True	True
criterion	['gini', 'entropy']	'gini'

Tabuľka 3.1: Finálne hodnoty pre *Random Forest*

Hyperparametre	**Hodnoty**	**Najlepšie**
n_estimators	[50, 100, 150, 200, 250, 300]	250
max_features	[10, 30, 40, 50, 'auto', 'sqrt', 'log2']	40
max_depth	[10, 30, 50, None]	50
min_samples_split	[3, 6, 9]	6
min_samples_leaf	[3, 6, 9]	3
bootstrap	False	False
criterion	['gini', 'entropy']	'gini'

Tabuľka 3.2: Finálne hodnoty pre *Extremly Randomized Trees*

Hyperparametre	**Hodnoty**	**Najlepšie**
algorithm	['SAMME', 'SAMME.R']	SAMME
learning_rate	[0.5, 1, 1.5]	1
n_estimators	[50, 100, 150, 200]	100

Tabuľka 3.3: *Random Forest* s využitím *Adaptive Boosting*

Hyperparametre	**Hodnoty**	**Najlepšie**
algorithm	['SAMME', 'SAMME.R']	SAMME
learning_rate	[0.5, 1, 1.5]	0.5
n_estimators	[50, 100, 150, 200]	50

Tabuľka 3.4: *Extremly Randomized Trees* s využitím *Adaptive Boosting*

Po vyhodnotení klasifikátorov s najlepšími hyperparametrami a s použitím metódy `cross_val_predict`, nám najvyššia presnosť v závislosti na rýchlosti klasifikácie vyšla v prípade *Random Forest* bez použitia *Adaptive Boosting*. Ďalej v práci sa budeme zaoberať len týmto modelom, vyhodnotenie metrík a rýchlostí predikcie jednotlivých klasifikátorov môžeme nájsť v Sekcii 5.

Kapitola 4

Implementácia

4.1 Nástroj na automatické zachytávanie dát

```
def wire_cap(event, capture_type):
    # cmd command vary based on traffic type
    # below is simple example
    cmd_command = 'tshark -a files:50 -b filesize:100000
    # create subprocess for Tshark command
    p1 = subprocess.Popen(cmd_command, shell=True,
                          stderr=subprocess.PIPE)
    # wait till Tshark captured enough network traffic
    p1.wait()
    print("--> Terminating")
```

Obr. 4.1: Inicializácia *subprocess* pre nástroj TShark

Počas zachytávania dátovej sady vznikol softvérový prototyp v jazyku Python ktorý tento proces uľahčil. Využíva knižnicu *selenium*[17] ktorá implementuje funkcionlitu potrebnú na jednoduché manipulovanie s webovým prehliadačom Google Chrome. Proces automatizácie prebieha v niekoľkých krokoch:

1. V prvom kroku je potrebné aby si užívateľ zvolil akú triedu komunikácie sa chystá zachytávať.

2. V momente keď si užívateľ zvolil triedu (napríklad **Video**) je inicializované nové vlákno ktorému je priradená funkcia `wire_cap` z Obrázku 4.1. V nej sa vytvorí nový *subprocess* v ktorom pobeží konzolový príkaz TShark. Po dosiahnutí obmedzenia zachytávania (typicky to môže byť

[17] `https://selenium-python.readthedocs.io/`

počet zachytených dát, tento údaj je špecifikovaný v TShark príkaze) sa *subprocess* ukončí a tento fakt oznámi hlavnému čakajúcemu vláknu.

3. Po spustení nového vlákna z bodu 2. sa v hlavnom vlákne otvorí služba Youtube s náhodným videom ktoré vyberieme z nami vytvoreného zoznamu. Video je prehrávané až do momentu pokiaľ vlákno z bodu 2. neoznámi hlavnému vláknu že *subprocess* zachytávania dát skončil.

4. V poslednom kroku sa Chrome Driver zavrie a celý proces môže prebiehať znovu, prípadne s inou triedou.

```
def merged_fields(row):
    # initialize dictionary based on features
    stats = {x: 0 for x in hists_fields_merged}
    # merge Histogram fields
    fwd_big_packets = sum(row["s_phists_sizes"][-2:])
    fwd_medium_packets = sum(row["s_phists_sizes"][4:6])
    fwd_small_packets = sum(row["s_phists_sizes"][0:4])
    # other way
    bwd_big_packets = sum(row["d_phists_sizes"][-2:])
    bwd_medium_packets = sum(row["d_phists_sizes"][4:6])
    bwd_small_packets = sum(row["d_phists_sizes"][0:4])

    stats["fwd_big_packets"] = fwd_big_packets
    stats["fwd_sml_packets"] = fwd_small_packets
    stats["fwd_med_packets"] = fwd_medium_packets

    stats["fwd_big_pper"] = fwd_big_packets / row["packets"]
    stats["fwd_sml_pper"] = fwd_small_packets / row["packets"]
    stats["fwd_med_pper"] = fwd_medium_packets / row["packets"]

    stats["both_big_packets"] = bwd_big_packets +
                                fwd_big_packets
    stats["both_sml_packets"] = bwd_small_packets +
                                fwd_small_packets
    stats["both_med_packets"] = bwd_medium_packets +
                                fwd_medium_packets
    return stats
```

Obr. 4.2: Generická štruktúra výpočtu vlastností

4.2 Moduly exportujúce vlastnosti

Diplomová práca [26] Daniela Uhříčka obsahuje nástroj *Feature Exploration Toolkit* ktorý sa zaoberá skúmaním vlastností tokov. My sme v našej práci do tohto nástroja vytvorili dva moduly ktoré spracovávajú vlastnosti z kategórie **Histogramy** a **Bursts**. Moduly sú implementované v jazyku Python. Na Obrázku 4.2 vidíme ukážku funkcie ktorá počíta vlastnosti zlúčených paketových skupín z kategórie **Histogramy**.

4.3 Implementácia najlepšieho klasifikátoru

V tejto časti si popíšeme implementáciu klasifikátoru s najvyššou presnosťou. Pre klasifikáciu používame vlastnosti ktorých proces hľadania a výčet nájdeme v Sekcii 3.1.3. Najlepšie hyperparametre klasifkátoru sme našli v Sekcii 3.2.1.

4.3.1 Proces učenia

Na Obrázku 4.3 môžeme vidieť implementáciu *Random Forest* v procese učenia s najlepšími hyperparametrami.

```
from imblearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE

# initialize classifier with best hyperparameters
classifier = RandomForestClassifier(n_estimators=200,
                                    max_features='auto',
                                    max_depth=50,
                                    min_samples_split=6,
                                    min_samples_leaf=3,
                                    bootstrap=True,
                                    citerion='gini')
# initialize SMOTE and Pipeline
smote_class = SMOTE()
pipeline = Pipeline([('smote', smote_class),
                     ('classifier', classifier)])
# fit training dataset
pipeline.fit(features_train,labels_train)
```

Obr. 4.3: Proces učenia *Random Forest*

1. V prvom kroku inicializujeme model `Random Forest` s najlepšími hyperparametrami.

2. Následne musíme inicializovať triedu `SMOTE` a `Pipeline`. Trieda `SMOTE` slúži k vyrovnaniu počtov tokov v dátovej sade. Trieda `Pipeline` pomáha správnemu fungovaniu triedy `SMOTE` s použitím klasifkátoru.

3. Metóda `pipeline.fit(features_train,labels_train)` spustí učiaci proces klasifikátoru.

   - `features_train`: sú vlastnosti na ktorých sa klasifikátor učí.
   - `labels_train`: sú označenia tried jednotlivých vlastností tokov pre trénovaciu dátovú sadu `features_train`.

4.3.2 Proces validácie

Klasifikátor ktorý sme použili v procese učenia v predchádzajúcej Sekcii 4.3.1, môžeme použiť následne na validovanie resp. testovanie.

1. Validácia pozostáva z jediného kroku, zavoláme funkciu `predict` pomocou ktorej klasifikátor predpovedá označenia vstupných dát. Parameter `features_validate` je validačná dátová sada pomocou ktorej môžeme vyhodnotiť presnosť s akou klasifikátor pracuje.

```
# predict labels based on trained features
predicted_labels = pipeline.predict(features_validate)
```

Obr. 4.4: Proces predikcie *Random Forest*

Keďže poznáme reálne a predpovedané označenia validačnej dátovej sady môžeme sledovať ako presne klasifikátor pracuje. Tomuto vyhodnoteniu sa venuje Kapitola 5.

Kapitola 5

Vyhodnotenie

5.1 Výkonnostné vyhodnotenie

Výkonnostné parametre sme merali na softvérovom prototype v ktorom sú implementované všetky použité algoritmy strojového učenia. Na vyhodnotenie týchto parametrov sme použili počítač s nasledujúcimi komponentami:

- *Processor AMD Ryzen 5 3600 6-Core Processor, 3593 MHz, 6 Core(s), 12 Logical Processor(s)*
- *32 GB 3200MHz RAM*
- *KC2500 NVMe PCIe SSD 3500 MB/s read, 2500 MB/s write*
- *Microsoft Windows 10 Education N*

V Tabuľke 5.1 vidíme namerané hodnoty ktoré prislúchajú **iba predikcii** zo Sekcie 4.3.2, teda bez ostatných operácií ktoré ku klasifikácií prislúchajú. Následne v Tabuľke 5.2 môžeme pozorovať trvanie počas ktorého sa pre vstupné dáta **vypočítali potrebné vlastnosti** zo Sekcie 3.1.3 na ktorých bola ďalej použitá **predikcia**.

	14 000 tokov	**# tokov za sekundu**
Random Forest	∼316 milisekúnd	∼44 303 tokov
Extremly Randomized Trees	∼408 milisekúnd	∼34 313 tokov
Adaptive Boosting s RF	∼27 sekúnd	∼518 tokov
Adaptive Boosting s ERT	∼4 sekundy	∼3 500 tokov

Tabuľka 5.1: Vyhodnotenie rýchlosti procesu predikcie modelov

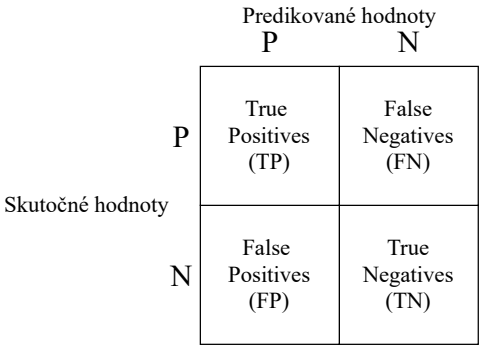
	14000 tokov	**# tokov za sekundu**
Random Forest	∼24 sekúnd	∼583 tokov
Extremly Randomized Trees	∼24 sekúnd	∼583 tokov
Adaptive Boosting s RF	∼52 sekúnd	∼269 tokov
Adaptive Boosting s ERT	∼28 sekúnd	∼500 tokov

Tabuľka 5.2: Vyhodnotenie rýchlosti výpočtu vlastností a procesu predikcie modelov

5.2 Vyhodnotenie metrík

Vyhodnotenie metrík prebiehalo rovnako na všetkých použitých algoritmoch strojového učenia. Na porovnanie klasifikátorov sme použili 4 rôzne metriky, na ich vysvetlenie musíme zadefinovať ukážku z Obrázku 5.1 teda ako S. Raschka vo svojej práci [27] opísal:

- **P**: je označenie kategórie komunikácie.
- **N**: sú všetky ostatné označenia.
- ***True Positives***: je počet tokov ktorým je predikované označenie P a ich reálne označenie je P.
- ***False Negatives***: je počet tokov ktorým nebolo predikované označenie P ale ich skutočné označenie je P.
- ***False Positives***: je počet tokov ktorým bolo predikované označenie P ale ich skutočné označenie nieje P.
- ***True Negatives***: je počet tokov ktorým nebolo predikované označenie P a zároveň ich skutočné označenie nieje P.



Obr. 5.1: *Binary Confusion Matrix* [27]

- ***Accuracy***: ACC = $\frac{TP+TN}{TP+TN+FP+FN}$
- ***Precision***: PRE = $\frac{TP}{TP+FP}$
- ***Recall***: REC = $\frac{TP}{TP+FN}$
- ***F1-Score***: F1 = $2 * \frac{PRE*REC}{PRE+REC}$

Na Obrázku 5.3 sú výsledky z vyhodnotenia ktoré bolo prevedené pomocou metódy `cross_val_predict` na trénovacej dátovej sade. V tejto implementovaný *Random Forest* dosiahol najnižšie výsledky.

	Accuracy	***F1-score***	***Precision***	***Recall***
Random Forest	90,76 %	90,79 %	90,88 %	90,76 %
Extremly Randomized Trees	91,34 %	91,36 %	91,44 %	91,34 %
Adaptive Boosting s RF	91,19 %	91,23 %	91,32 %	91,19 %
Adaptive Boosting s ERT	91,62 %	91,65 %	91,74 %	91.62%

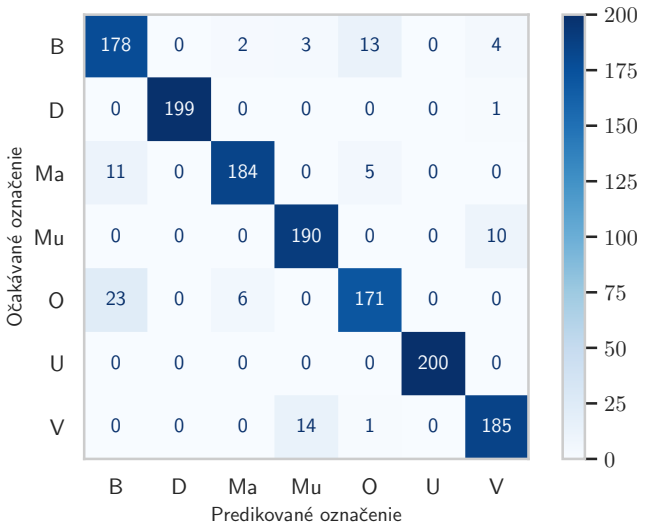
Tabuľka 5.3: Výsledky `cross-val-predict`

	Accuracy	***F1-score***	***Precision***	***Recall***
Random Forest	93,35 %	93,37 %	93,44 %	93,35 %
Extremly Randomized Trees	93,28 %	93,29 %	93,33 %	93,28 %
Adaptive Boosting s RF	93,57 %	93,58 %	93,60 %	93,57 %
Adaptive Boosting s ERT	93,42 %	93,43 %	93,49 %	93,42 %

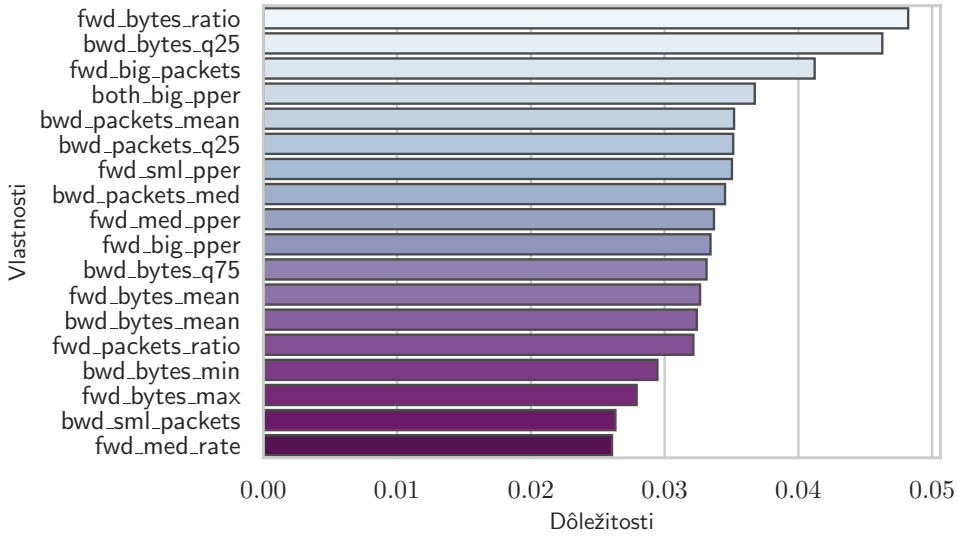
Tabuľka 5.4: Výsledky procesu validácie

Na druhej strane v testovacom vyhodnotení ktoré bolo prevedenom na validačnej dátovej sade dosiahol *Random Forest* vyššie výsledky ktoré sú porovnateľné s ostatnými klasifikátormi. V Sekcii 5.1 navyše vidíme že klasifikátor ktorý používa *Random Forest* pracuje najrýchlejšie preto sme sa rozhodli pre implementáciu práve tohto algoritmu. *Adaptive Boosting* síce zlepšuje o malé množstvo presnosť tohto algoritmu, avšak s jeho použitím je klasifikácia značne pomalšia čo neumožňuje jej použitie v reálnom čase. Ďalšiu dôležitú vec ktorú si môžeme všimnúť je že všetky algoritmy pracujú s vysokou presnosťou, to môže znamenať že analýza z ktorej vzišli hlavné klasifikačné vlastnosti (viď Sekcia 3.1.3) bola prevedená správne.

Na Obrázku 5.2 sa nachádza *confusion matrix* ktorá poukazuje na to koľko označení klasifikátor predikoval správne resp. nesprávne. *Confusion matrix* prislúcha validačnému procesu ktorého postup sme opísali v Sekcii 4.3.2. Ďalej na Obrázku 5.3 môžeme vidieť dôležitosti prvých 15 vlastností (z celkového počtu 56) ktoré obsahuje model zo Sekcie 4.3.2.



Obr. 5.2: *Confusion Matrix*, platí: B = Prezeranie webu, D = Sťahovanie súborov, Ma = Mapy, Mu = Hudba, O = Ostatné, U = Nahrávanie súborov, V = Video



Obr. 5.3: Dôležitosti prvých 15 vlastností použitých v *Random Forest*

Záver

Cieľom tejto práce bolo monitorovať sieťovú komunikáciu a následne vytvoriť klasifikátor založený na modely strojového učenia ktorý bude pracovať na základe analyzovaných vlastností tokov. Klasifikácia dát prenášaných po sieti dokáže značne zlepšiť bezpečnosť a povedomie o prenášaných dátach, či už z pohľadu filtrovania nevhodného obsahu , alebo naopak filtrovania iba toho vhodného.

Výstupom práce je softvérový prototyp obsahujúci klasifikátor založený na skúmaných algoritmoch strojového učenia, ktorý vie s vysokou presnosťou a rýchlosťou klasifikovať sieťovú komunikáciu, tento dokáže byť neskôr implementovaný ako *Detector* modul do systému NEMEA čím zúži objem neznámych dát prenášaných po sieti. Počas vypracovania bol rovnako vytvorený nástroj na automatizáciu procesu zachytávania dátovej sady ktorý môže byť aplikovaný nielen na protokol QUIC ale aj na iné. Ďalej vznikli dva moduly PHISTS a BSTATS slúžiace na skúmanie kľúčových vlastností tokov monitorovanej sieťovej komunikácie, ktorých funkcionalita dokážu byť plne integrované do nástroja Feature Exploration Toolkit a tým prispieť v skúmaní vlastností sieťových dát prenášaných pomocou iných protokolov.

Výsledné experimenty ukázali že skúmané modely strojového učenia *Random Forest*, *Extremly Randomized Trees* a *Adaptive Boosting* ktorý bol použitý na predošlých dvoch, pracujú s takmer rovnakou presnosťou, teda hodnoty výsledných metrík majú nízky rozptyl. Následne boli skúmané časové závislosti ktorých výsledky sa líšia omnoho viac. Suverénne najmenej časovo závislý klasifikátor používa *Random Forest*, na základe ktorého vznikol softvérový prototyp.

V rámci budúcich práci je potrebné zväčšiť dátovú sadu ktorá bude obsahovať väčšie množstvo IP tokov kategórie sťahovanie a odosielanie súborov. Rovnako je žiadúce preskúmať vlastnosti tokov spojené s medzipaketovými intervalmi a časy medzi jednotlivými *bursts*. Napriek tomu že tieto vlastnosti boli implementované v moduloch ktoré skúmajú charakteristiky tokov, sa nimi táto práca nezaoberá z dôvodu úzkej súvislosti s rýchlosťou pripojenia

a odozvy spojenia (tieto parametre sa môžu líšiť naprieč používateľmi preto je obtiažne ich simulovať na jednom monitorovacom stroji). V neposlednom rade je potrebné dátovú sadu rozšíriť o ďalšie kategórie sieťovej komunikácie ktoré zvýšia kontrolu nad sieťovou komunikáciou.

Literatúra

[1] Paraskevi, D.; Jan, F.; Nicolas, M.; aj.: Encrypted Traffic Analysis, Use Cases & Security Challenges. Technická zpráva, European Union Angency for Cybersecurity, 2019, [cit. 2021-04-25]. Dostupné z: https://www.enisa.europa.eu/publications/encrypted-traffic-analysis

[2] Velan, P.; Čermák, M.; Čeleda, P.; aj.: A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, ročník 25, č. 5, 2015: s. 355–374, doi:10.1002/nem.1901, [cit. 2021-04-25]. Dostupné z: https://doi.org/10.1002/nem.1901

[3] Roskind, J.: MULTIPLEXED STREAM TRANSPORT OVER UDP. Internet-draft, April 2012, [cit. 2021-04-25]. Dostupné z: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit

[4] Roskind, J.: Experimenting with QUIC. Online, June 2013, [cit. 2021-04-25]. Dostupné z: https://blog.chromium.org/2013/06/experimenting-with-quic.html

[5] Simon, S. D.: Google Will Propose QUIC As IETF Standard. Online, April 2015, [cit. 2021-04-25]. Dostupné z: https://www.infoq.com/news/2015/04/google-quic-ietf-standard/

[6] Iyengar, J.; Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-00, IETF Secretariat, November 2016, [cit. 2021-04-25]. Dostupné z: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-00

[7] Cimpanu, C.: HTTP-over-QUIC to be renamed HTTP/3. Online, November 2018, [cit. 2021-04-25]. Dostupné z: https://www.zdnet.com/article/http-over-quic-to-be-renamed-http3/

[8] Iyengar, J.; Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-34, IETF Secretariat, 2021, [cit. 2021-04-25]. Dostupné z: http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-34.txt

[9] Thomson, M.; Turner, S.: Using TLS to Secure QUIC. Internet-Draft draft-ietf-quic-tls-34, IETF Secretariat, 2021, [cit. 2021-04-25]. Dostupné z: http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-34.txt

[10] Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, RFC Editor, August 2018, [cit. 2021-04-25]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc8446

[11] Gagliardi, E.; Levillain, O.: Analysis of QUIC Session Establishment and Its Implementations. *Information Security Theory and Practice Lecture Notes in Computer Science*, Dec 2019: str. 169–184, [cit. 2021-04-25]. Dostupné z: https://hal.archives-ouvertes.fr/hal-02468596/document

[12] Krawczyk, H.; Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, RFC Editor, May 2010, [cit. 2021-04-25]. Dostupné z: http://www.rfc-editor.org/rfc/rfc5869.txt

[13] Yang, F.: The tale of deep packet inspection in China: Mind the gap. In *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, 2015, s. 348–351, doi:10.1109/ICoICT.2015.7231449, [cit. 2021-04-25].

[14] Yamauchi, H.; Nakao, A.; Oguchi, M.; aj.: Service Identification Based on SNI Analysis. In *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, 2020, s. 1–6, doi:10.1109/CCNC46108.2020.9045315, [cit. 2021-04-25].

[15] Claise, B.; Trammell, B.; Aitken, P.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. STD 77, RFC Editor, September 2013, [cit. 2021-04-25]. Dostupné z: http://www.rfc-editor.org/rfc/rfc7011.txt

[16] Claise, B.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, RFC Editor, January 2008, [cit. 2021-04-25]. Dostupné z: http://www.rfc-editor.org/rfc/rfc5101.txt

[17] IANA: IP Flow Information Export (IPFIX) Entities. Online, May 2007, [cit. 2021-04-25]. Dostupné z: https://www.iana.org/assignments/ipfix/ipfix.xml

[18] Cejka, T.; Bartos, V.; Svepes, M.; aj.: NEMEA: A Framework for Network Traffic Analysis. In *12th International Conference on Network and Service Management (CNSM 2016)*, 2016, doi:10.1109/CNSM.2016.7818417, [cit. 2021-04-25]. Dostupné z: `http://dx.doi.org/10.1109/CNSM.2016.7818417`

[19] Cejka, T.; Bartos, V.; Svepes, M.; aj.: NEMEA: A framework for network traffic analysis. In *2016 12th International Conference on Network and Service Management (CNSM)*, 2016, s. 195–201, doi:10.1109/CNSM.2016.7818417, [cit. 2021-04-25].

[20] Al Hamad, M.; Zeki, A. M.: Accuracy vs. Cost in Decision Trees: A Survey. In *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, 2018, s. 1–4, doi:10.1109/3ICT.2018.8855780, [cit. 2021-04-25].

[21] Montalbo, F. J. P.; Festijo, E. D.: Comparative Analysis of Ensemble Learning Methods in Classifying Network Intrusions. In *2019 IEEE 9th International Conference on System Engineering and Technology (ICSET)*, 2019, s. 431–436, doi:10.1109/ICSEngT.2019.8906310, [cit. 2021-04-25].

[22] Geurts, P.; Ernst, D.; Wehenkel, L.: Extremely randomized trees. *Machine Learning*, ročník 63, č. 1, 2006: str. 3–42, doi:10.1007/s10994-006-6226-1, [cit. 2021-04-25]. Dostupné z: `https://link.springer.com/content/pdf/10.1007/s10994-006-6226-1.pdf`

[23] R., A.: Resampling Methods — A Simple Introduction to The Bootstrap Method. May 2020, [cit. 2021-04-25]. Dostupné z: `https://arifromadhan19.medium.com/resampling-methods-a-simple-introduction-to-the-bootstrap-method-3a36d076852f`

[24] Rogers, A.; Brewer, G.: Websites using QUIC. Online, [cit. 2021-04-25]. Dostupné z: `https://trends.builtwith.com/websitelist/QUIC`

[25] Gao, L.: On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking*, ročník 9, č. 6, 2001: s. 733–745, doi:10.1109/90.974527, [cit. 2021-04-25].

[26] Uhříček, D.: Detection of IoT Malware in Computer Networks. Master thesis, Czech Technical University in Prague, Faculty of Information Technology, 2021, [cit. 2021-04-25].

[27] Raschka, S.: An Overview of General Performance Metrics of Binary Classifier Systems. 2014, [cit. 2021-04-25]. Dostupné z: `https://arxiv.org/pdf/1410.5330.pdf`

Dodatok **A**

Zoznam použitých skratiek

IETF Internet Engineering Task Force

UDP User Datagram Protocol

HTTP Hyper Transfer Text Protocol

TCP Transmission Control Protocol

TLS Transport Layer Security

ISKU Identifikátor spojenia koncového uzlu

ISPU Identifikátor spojenia počiatočného uzlu

RTT Round Trip Time

RFC Request For Comments

AES-GCM Advanced Encryption Standard with Galois/Counter Mode

AEAD Authenticated Encryption with Associated Data

AES-ECB Advanced Encryption Standard with Electronic Codebook

HMAC Keyed-Hashing for Message Authentication

NAT Network Address Translation

IP Internet Protocol

SNI Server Name Indication

IPFIX Internet Protocol Flow Information Export

NEMEA Network Measurements Analysis

PCAP Packet Capture

ASN Autonomous System Number

Dodatok **B**

Obsah priloženého CD

- `ReadMe.txt` ..... obsahuje stručný popis obsahu CD
- `src`
  - `moduly` ..... zdrojové kódy modulov BSTATS a PHISTS
  - `dataset_tool` ..... zdrojový kód nástroju na tvorbu dátovej sady
  - `prototyp` ..... zdrojové kódy prototypu klasifikátoru
  - `ostatne` ..... ostaté použité zdrojové kódy
- `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF
- `dátové_sady` ..... validačná a trénovacia dátová sada