VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Síťové aplikace
NetFlow v5 exportér

16. října 2024

Machala Roman (xmacha86)

Obsah

1 Úvod

NetFlow je síťový protokol vyvinut společností CISCO, který slouží k monitorování a analýze síťového provozu. Umožňuje zachytávat jednotlivé pakety, které následně rozděluje do toků na základě např. zdrojové a cílové IP adresy. Tyto data jsou pomocí protokolu UDP z exportéru odeslána do centrálního sběrače (kolektoru), kde mohou být analyzována za účelem zjištění vytížení síte, anomálií nebo například i bezpečnostních incidentů[1].

Pro protokol NetFlow existuje mnoho verzí, nicméně tato práce se zabývá pouze verzí v5, což je jedna z nejpoužívanějších verzí vůbec, dostupná na drtivé většině zařízení různých výrobců. Tato verze byla navržena pro sběr informací síťového provozu pouze na bázi IPv4. Pro podporu IPv6 je nutno vuyžívat vyšších verzí, jako je například NetFlow v9[4].

1.1 IP Tok

Jak již bylo zmíněno výše, protokol NetFlow pracuje s jednotlivými paketami, které rozděluje do **toků**. Každý takový tok je určen unikátní kombinací následujících informací:

- **Zdrojová IP adresa**
- **Cílová IP adresa**
- **Zdrojový PORT**
- **Cílový PORT**
- Typ služby (ToS)
- **Protokol 3 vrstvy**
- Rozhranní

Všechny toky jsou pouze jednosměrné, proto například prohozením adres odesílatele a příjemce nevznikne nový tok, jak je demonstrováno na obrázku 1. V rámci implementace popsané v této práci jsou ovšem za unikátní identifikátory uvažovány pouze **tučně** vyznačené informace. Je tak uvažováno z důvodů stanovení požadavků dle zadání nebo pozdějších upřesnění v rámci diskuzního fóra pro řešení projektu. Více info o rozdílech a limitacích v sekci 1.4.

Tok 1:

Zdrojová IP:	192.168.1.1
Cílová IP:	10.0.0.1
Zdrojový port:	12345
Cílový port:	80
Protokol:	TCP

Tok 2:

Zdrojová IP:	10.0.0.1
Cílová IP:	192.168.1.1
Zdrojový port:	80
Cílový port:	12345
Protokol:	TCP

Obrázek 1: Příklad 2 různých toků

Z obrázku 1 může být taky patrné, že protokol nebere ohled na obsah jednotlivých paket, co se týče rozdělení paket do toků. V rámci protokolu NetFlow v5 se s obsahem paket nijak nemanipuluje, uchovávají se ovšem další informace, které jsou vhodné pro statistické vyobrazení a monitorování síťového provozu.

1.2 Struktura toků/NetFlow v5 datagram

Kromě klíčových informací sloužících k rozdělení jednotlivých paket do toků se ukládají i jiné informace o každém toku. Výčet těchto informací je vyobrazen jako struktura v jazyce C na obrázku 2. V tomto formátu jsou jednotlivé toky pomocí UDP protokolu odeslány na kolektor.

Jelikož je používán protokol UDP pro export datagramů, je možné, že některé datagramy mohou být ztraceny. Z tohoto důvodu novější verze tohoto protokolu, přesněji verze 5, 7 a 8 obsahuje hlavička, dále popsána na obrázku 3, kontrolní číslo toku (**flow control number**), které je rovno kontrolnímu číslu předešlého toku + počet toků v předešlém datagramu. Při přijetí nového datagramu tak kolektor může zkontrolovat, zda došlo ke ztrátě toků[2][3].

```
typedef struct NetFlowv5 {
    uint32_t srcaddr;       /* Source IP address */
    uint32_t dstaddr;       /* Destination IP address */
    uint32_t nexthop;       /* IP address of next hop router */
    uint16_t input;         /* SNMP index of input interface */
    uint16_t output;        /* SNMP index of output interface */
    uint32_t dPkts;         /* Packets in the flow */
    uint32_t dOctets;       /* Total number of Layer 3 bytes */
    uint32_t first;         /* SysUptime at start of flow */
    uint32_t last;          /* SysUptime at the time the last packet */
    uint16_t srcport;       /* TCP/UDP source port number */
    uint16_t dstport;       /* TCP/UDP destination port number */
    uint8_t pad1;           /* Unused (zero) bytes */
    uint8_t tcp_flags;      /* Cumulative OR of TCP flags */
    uint8_t prot;           /* IP protocol type (for example, TCP = 6) */
    uint8_t tos;            /* IP type of service (ToS) */
    uint16_t src_as;        /* Autonomous system number of the source */
    uint16_t dst_as;        /* Autonomous system number of the dest */
    uint8_t src_mask;       /* Source address prefix mask bits */
    uint8_t dst_mask;       /* Destination address prefix mask bits */
    uint16_t pad2;          /* Unused (zero) bytes */
} netflowv5;
```

Obrázek 2: NetFlow v5 tělo datagramu

```
typedef struct NetFlowHeader {
    uint16_t version;           /* NetFlow export format version num */
    uint16_t count;             /* Number of exported flows */
    uint32_t sysUptime;         /* Current time in ms since start */
    uint32_t unix_secs;         /* Current count of seconds since CUT */
    uint32_t unix_nsecs;        /* Residual nanoseconds since CUT */
    uint32_t flow_sequence;     /* Sequence cnt of total flows seen */
    uint8_t engine_type;        /* Type of flow-switching engine */
    uint8_t engine_id;          /* Slot num of flow-switching engine */
    uint16_t sampling_interval; /* Sampling mode + interval (2b - 14b) */
} NetFlowHeader;
```

Obrázek 3: NetFlow v5 hlavička datagramu

1.3 Expirace toků

Jednotlivé toky je nutno v určitém okamžiku uzavřít a připravit je pro odeslání na kolektor. V rámci protokolu NetFlowv5 se řeší 2 typy timeoutů, kdy je tok považován za expirovaný/uzavřený.

1.3.1 Aktivní timeout

Aktivní timeout stanovuje dobu od první přijaté pakety, po které je tok uzavřen, nehledě na tom, zdali přicházejí další pakety. Jednoduchým příkladem může být komunikace mezi bodem **A** a **B**, která trvá 5 minut, tedy (300 sekund). Pokud bude aktivní timeout nastaven na dobu 60 sekund, z této komunikace nám vznikne celkem 5 toků, které budou identické co se týče položek určujících tok, sekce 1.1, ale mohou být různé v počtu paket a celkového počtu bytů v rámci jednotlivých toků.

1.3.2 Neaktivní timeout

Neaktivní timeout stanovuje dobu od poslední přijaté pakety, po kterou pokud nedojde další paketa patřící do onoho toku, je tok uzavřen. Příkladem pro tok expirovaný/uzavřený na základě neaktivního timeoutu je komunikace mezi bodem **A** a **B**, která obsahuje 2 pakety, kdy paketa **P1** je zachycena v čase **T1** a paketa **P2** je zachycena v čase **T1 + 30s**. Pokud je neaktivní timeout nastaven na 20 sekund, bude v době **T1 + 20s** tok obsahující pouze paketu **P1** uzavřen. Pro následující paketu **P2** v čase **T1 + 30s** bude vytvořen nový tok, obsahující pouze paketu **P2**.

1.4 Rozdíly v implementaci

V této sekci jsou popsány limitace procesu návrhu a implementace exportéru pro protokol NetFlow v5.

Implementovaný exportér pracuje pouze s informacemi zjistitelnými analýzou zachycených paket z poskytnutého PCAP souboru. Mezi nezjistitelné položky tak patří:

- Adresa routeru dalšího skoku
- SNMP indexy vstupního a výstupního zařízení
- Čísla autonomních systémů zdrojové a cílové sítě
- Počet bitů v maskovacím prefixu zdrojové a cílové adresy
- Typ a ID zařízení zpracovávající toky
- Sampling mód a interval

Takto nezjistitelné informace jsou potom nahrazeny 0, což je standardní postup při analýze paket z PCAP souboru.

Dalším omezením jsou časové značky jednotlivých toků a paket. NetFlowv5 uchovává informace jako je třeba čas zachycení pakety od spuštění exportéru.

$$x = T1 - T0 \tag{1}$$

Kde x představuje čas zachycení pakety od spuštění systému, $T1$ skutečný čas zachycení pakety a $T0$ skutečný čas spuštění exportéru. Jelikož ale pakety byly zachyceny před spuštěním exportéru, musí platit, že:

$$T1 < T0 \tag{2}$$

a z toho důvodu jsou jednotlivé časové značky záporné. Drtivá většina kolektorů ovšem s tímto faktem počítá a záporné hodnoty jsou interpretovány bez problémů.

TCP flagy RST a FIN nejsou brány v potaz a neukončují tak tok, jak je tomu zvykem u některých NetFlow exportérů.

Z důvodu kontroly expirace toků až v momentě, kdy jsou vkládány nové informace do toku, popsáno v sekci 2.5, může nastat situace, že do toku již nemusí přijít nové informace a tok tak bude exportován až v případě kompletního zpracování souboru, tedy pořadí exportovaných toků nemusí odpovídat jejich skutečnému pořadí expirace[1].

[1] Snažil jsem se tomuto více věnovat zkoumáním softflowd a nfcapd nástrojů a ve výsledků se nic nestane, jen nesedí jejich pořadí.

2 Návrh a implementace

Samotná aplikace je rozdělena do několika větších celků, kde každý celek zajišťuje určitou část aplikace. Jednotlivé celky jsou implementovány v samostatných `.c` a `.h` souborech. Jednotlivé celky dohromady potom tvoří výslednou aplikaci pracující jako exportér NetFlow v5 toků.

2.1 Zpracování argumentů

Tento celek se stará o zpracování argumentů příkazové řádky, poskytnuté uživatelem, převodem těchto argumentů do jejich očekávané podoby (pokud je to potřeba), nebo ke kontrole jejich správnosti.

Parametr	**Popis**	**Povinný**
`hostname:port`	IP adresa a port kolektoru	Ano
`pcap_soubor`	PCAP soubor obsahující pakety pro analýzu	Ano
`-a | --active`	Aktivní timeout v sekundách	Ne
`-i | --inactive`	Neaktivní timeout v sekundách	Ne
`-d | --debug`	Povolit ladicí výstup	Ne

Tabulka 1: Parametry aplikace

V tabulce 1 je možno vidět, jaké parametry aplikace podporuje. Na jejich pořadí nezáleží, ovšem povinné parametry musejí být vždy přítomny. Pokud nejsou stanoveny parametry **-a** a **-i**, tedy aktivní a neaktivní timeout, pracuje se s defaultní hodnotou 60 sekund pro oba timeouty.

2.2 Zpracování paket ze souboru PCAP

Pro zpracování jednotlivých paket a jejich manipulace s nimi je prováděna výhradně za pomocí knihovny PCAP. Mezi nejdůležitější funkce používáné pro tento účel jsou `pcap_open_offline()` a `pcap_loop()`, které umožňují otevřít samotný soubor a načítat postupně pakety[7][6].

Jednotlivé pakety josu pak načteny v následujícím formátu:

PCAP hlavička	Ethernetová hlavička	IP hlavička	TCP hlavička	Data

Tabulka 2: Zachycená paketa

V tabulce 2 je paketa barevně odlišena na 2 části, se kterými se dá dále manipulovat.

```
struct pcap_pkthdr *pkthdr;      /* PCAP header */
uint8_t *packet;                 /* Packet */

/* Get ethernet header */
struct ethhdr *ether_header = (struct ethhdr *)packet;

/* Get IP header */
struct iphdr *ip_header = (struct iphdr *)(packet +
        sizeof(struct ethhdr));

/* Get TCP header */
struct tcphdr *tcp_header = (struct tcphdr *)(packet +
        sizeof(struct ethhdr) + (ip_header->ihl * 4));
```

Obrázek 4: Práce s paketami

Na obrázku 4 je vyobrazena následná práce s jednotlivými částmi zachycené pakety a jak se dostat k jedjnotlivým položkám popsaných v tabulce 2. Samotná data nás již nijak nezajímají, jelikož všechny potřebné informace se nachází v jendotlivých hlavičkách[5].

2.3 Tvorba toků

Obrázek 4 nám popisuje jak dostaneme jednotlivé hlavičky ze zachycené pakety. Samotné informace pro tvorbu toku, popsáno v sekci 1.1, jsme již schopni získat poměrně snadno.

Pro každou paketu je vytvořena struktura popsána na obrázku 2 a jsou vyplněny všechny možné informace, které jsou dostupné z jednotlivých hlaviček paket. Následně je tok zkontrolován, zdali již neexistuje a na základě toho, je buďto aktualizován již existující tok nebo uložen nový.

2.4 Struktura pro ukládání toků

Jakožto struktura uchovávající jednotlivé záznamy o všech tocích je zvolena mírně upravená hashovací tabulka. Na základě potřebných informací tvořící unikátní kombinaci toku, sekce 1.1, je vypočítán hash udávající index umístění toku v tabulce. Postup výpočtu hashe je možno vidět na obrázku 5.

```
int hash_function(netflowv5 *flow){
    uint64_t hash = flow->srcaddr;
    hash ^= flow->dstaddr;
    hash ^= (flow->srcport << 16);
    hash ^= (flow->dstport << 16);
    hash ^= (flow->prot << 16);

    return hash % MAX_FLOW_LENGTH;
}
```

Obrázek 5: Výpočet hashe

2.5 Kontrola expirace toků

Před vložením jednotlivých toků do tabulky je třeba zkontrolovat zdali neexistuje již takový tok v tabulce a pokud ano, jestli může být tok aktualizován, aby neporušil některý z timeoutů, popsaných v sekci 1.3.

Logika vkládání a kontroly je popsána v pseudokódu na obrázku 6.

```
1
2               /* Creates new flow based on provided packet */
3               flow = create_new_flow()
4
5               /* If same flow already exists, finds it */
6               orig_flow = get_flow(flow)
7
8               /* if it exists */
9               if orig_flow:
10                  /**
11                  * Checks wheter by updating original flow either active or inactive
12                  * timeout is exceeded
13                  */
14                  if check_active(orig_flow, flow) or check_inactive(orig_flow, flow):
15                      /* Closes flow and removes it from table */
16                      handle_flow(orig_flow)
17                      /* Inserts new flow into table */
18                      insert_flow(flow)
19                  else:
20                      /* Updates flow if no timeoutes are exceeded */
21                      update_flow(orig_flow, flow)
22              else:
23                  /* if no flow is found then insert as a new flow */
24                  insert_flow(flow)
```

Obrázek 6: Vkládání a kontrola expirace

Základní princip spočívá v tom, že při načtení pakety je vytvořen nový tok, následně se vyhledá v tabulce, zdali již takový tok neexistuje. Pokud ne, je nový tok vložen do tabulky. Pokud ano, zkontroluje se, pokud aktualizací již existujícího toku dojde k překročení nějakého z timeoutů, je původní tok uzavřen a uchován ve struktuře pro export toků, více v sekci 2.6, a nově vytvořený tok je vložen do tabulky. Pokud k překročení timeoutu nedojde, je původní tok aktualizován nově vytvořeným.

2.6 Export toků

Všechny uzavřené toky jsou uchovávány ve struktuře popsané na obrázku 7. Úkolem této struktury je uchovávat jednotlivé toky, považované za uzavřené na jednom místě, dokud nedojde k jejich exportu na kolektor. Aby byl počet exportovaných toků využit na maximum, dochází k jejich exportu, pokud je jejich počet roven 30, nebo pokud se zpracovaly všechny pakety ze souboru.

```
1               typedef struct ProcessedFlows{
2                   int count;                              /* Current num of flows */
3                   int total_count;                        /* Num of flows */
4                   netflowv5 *flows[MAX_NUMBER_FLOWS];     /* Flows */
5               }ProcessedFlows;
```

Obrázek 7: Struktura pro uzavřené toky

Položka **count** uchovává počet aktuálně uzavřených toků a při její hodnotě rovno 30 dojde k exportu. **Count** je pak resetován a inkrementován s dalším přibývajícím tokem. Položka **total_count** potom slouží k uchování celkového počtu všech toků, pro naplnění kontrolního počtu toků v hlavičce NetFlowv5 protokolu, viz. obrázek 3

Samotnou logiku pro vytvoření datagramu pro export toků lze pak vidět na obrázku 8.

```
NetFlowHeader header;
/* Fill header info */
...
/* Create socket and check for succes */
int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
...
/* Fill colector address */
struct sockaddr_in collector_addr;
...
/* Calculate UDP size and allocate memory */
size_t packet_size = sizeof(header) +
        sizeof(struct NetFlowv5) * set.count;
uint8_t *buffer = malloc(packet_size);

/* Copy header data to buffer */
memcpy(buffer, &header, sizeof(header));
/* Copy all flows into buffer */
for(int i = 0; i < set.count; i++){
    convert_flow_to_network_order(set.flows[i]);
    memcpy(buffer + sizeof(header) +
            (i * sizeof(struct NetFlowv5)), set.flows[i],
                sizeof(struct NetFlowv5));
}
/* Send datagram to collector */
ssize_t sent_bytes = sendto(sock, buffer, packet_size, 0,
        (struct sockaddr*)&collector_addr,
        sizeof(struct sockaddr_in));
...
```

Obrázek 8: Tvorba datagramu

3 Testování aplikace

Pro zajištění správného chodu aplikace, byla jednak v průběhu implementace řádně zkoumána řadou nástrojů, ale taktéž důkladně testována. Většina těchto nástrojů je považována za referenční, proto je testování uzpůsobeno porovnávání výsledků implementovaného exportéru a referečních nástrojů. Více o těchto nástrojích v sekci 3.2. Testy jsou plně automatizované a jsou dostupné ve složce `/tests`. Samotné testování je pak možno spustit následovně:

sudo ./tests.sh

kde je potřeba se nacházet ve složce `/tests`.

3.1 Návrh testů

Samotné testy jsou rozvrženy do několika skupin, kde každá skupina se snaží otestovat jinou část programu.

- Nejjednoduší testy se zabývají zpracováním 3-5 paket, kde všechny pakety patří do stejného toku. Cílem je tak ověřit schopnost programu zpracovat primitivní možství paket a samotnou schopnost vytvoření toku a následné agregace příslušných paket do oneho toku. Jsou zpracovávány zachycené komunikace obsahující pouze TCP protokol.
- Pokročilější testy jsou uzpůsobeny pro kontrolu schopnosti programu zpracovávat větší množství paket, cca 150, patřících do různých toků. Důklad je tedy kladen na správnou tvorbu toků, agregaci nových paket do toků a správnou manipulaci s pamětí.
- Náročnější testy potom kombinují vělké množství paket a různé protokoly. Cílem je opět správně zapracovat pakety do příslušných toků, správná manipulace s pamětí a zpracování pouze TCP paket.

- Samostatnou skupinu potom tvoří testy pro kontrolu správné implentace jednotlivých timeoutů, viz sekce 1.3. V těchto testech je primární zaměření na korektní tvorbu toků v závislosti na jednotlivých timeoutech. Pracuje se s menším množstvím paket, aby bylo zřetelné množství jednotlivých toků, nicméně je předpokládáno, že je možno zpracovávat libovolné množství paket[2].

3.2 Referenční programy a jejich použití

V této sekci jsou popsány jednotlivé nástroje použity pro účely ladění programu, testování anebo pro zkoumání jejich chování. U jednotlivých nástrojů je taky popsáno, jakým způsobem jsou nástroje spouštěny a k čmeu byly přesně použity.

3.2.1 Wireshark

Program Wireshark je použit především pro zachycení komunikace pro samotné testy a pro zkoumání detailů implementace nástroje Softflowd. Komunikace byla zachycena na počítači s kabelovým internetovým připojením na rozhraní `eth0`.

3.2.2 Softflowd

Nástroj softflowd slouží jako referenční exportér. Byl využit jednak pro zkoumání detailů implementace, ale i pro generování referenčních výsledků, kterých by měl implementovaný exportér dosáhnout.

Nástroj byl spouštěn následovně:

softflowd -r FILE -n localhost:1010 -v 5 -d[3]

3.2.3 Tcpdump

Tento nástroj byl především použit pro filtraci paket pro jednotlivé testy. Některé ze zachycených komunikací pro účely testování obsahují protokoly i jiné, než TCP. Z tohoto důvodu jsou testovací pakety před zpracováním nástrojem softflowd přefiltrovány, aby obsahovaly pouze pakety s protokolem TCP. Tato filtrace je ovšem použita pouze před použitím referenčního nástroje, který neumožňuje, například přepínačem, zpracovávat pouze určité druhy protokolů. Implementovaný exportér zpracovává pouze TCP, není tedy nutná filtrace před jeho použitím. Nástroj byl spouštěn následovně:

tcpdump -r FILE -w FILTERED_FILE -p tcp[4]

3.2.4 Nfcapd

Tento nástroj byl použit jako kolektor pro jednak referenční exportér, ale i pro implementovaný. Jeho úkolem bylo zachycení exportovaných toků a vyobrazení jednotlivých statistik, jejichž shoda pak znamenala úspěch daného testu. Nástroj byl spouštěn následovně:

nfcapd -l OUTPUT_DIR -p 1010[5]

3.2.5 Nfdump

Nfdump slouží pro vyobrazení jednotlivých statisktik, jež jsou výstupem kolektoru. Jedná se tak o nástroj zobrazující statistiky zpracovaných exportovaných toků. Tyto statistiky jsou následně klíčové pro ověření správnosti implementovaného exportéru vůči referenčnímu. Nástroj byl spouštěn následovně:

nfdump -r FILE[6]

[2] Tyto testy jsou jediné, jež nejsou automatizované.

[3] Export paketů ze souboru FILE na kolektor na adrese localhost:1010, verze NetFlowv5

[4] Filtrace tcp paket ze souboru FILE, zápis do souboru FILTERED_FILE

[5] Výstup nástroje uložen do složky OUTPUT_DIR, poslouchá na portu 1010

[6] Zobrazí statistiky daného souboru

3.2.6 Valgrind

Nástroj valgrind byl využit pro kontrolu práce s pamětí, její alokace a především její správné uvolnění. Každý test kromě kontroly požadovaných statistik kontroluje i správu paměti, která je jednou z mnoha podmínek pro splnění daného testu.

3.3 Podmínky testování

Samotné testování funguje na tom principu, že je spuštěn kolektor a jsou na něj odeslány toky z referenčního a implementovaného exportéru. Klíčové vlastnosti, které jsou kontrolovány jsou následující:

- Správná manipulace s pamětí
- Celkový počet toků
- Celkový počet bytů
- Celkový počet paket
- Celkový počet zpracovaných toků
- Počet přeskočených bloků
- Přečteno bytů
- Časová okna toků

Položky jako průměrný počet paket za sekundu atd. jsou taktéž kontrolovány, ale v případě neshody jsou pouze vypsaný, nejsou brány jako důvod k označení testu za selhaný, protože tyto informace vyžadují vysokou časovou přesnost a jsou závislé na celkové době zpracování toků, což může být ovlivněno specifickou implementací. Příklad, jak vypadají kontrolované položky je na obrázku 9.

```
Date first seen          Event  XEvent Proto      Src IP Addr:Port          Dst IP Addr:Port     X-Src IP Addr:Port       X-Dst IP Addr:Port   In Byte Out Byte
2024-10-10 14:18:01.269 INVALID  Ignore TCP       192.168.0.28:50189 ->   142.251.36.138:443           0.0.0.0:0     ->         0.0.0.0:0           41        0
2024-10-10 14:18:07.185 INVALID  Ignore TCP       192.168.0.28:50049 ->  142.250.102.188:5228          0.0.0.0:0     ->         0.0.0.0:0           41        0
Summary: total flows: 2, total bytes: 82, total packets: 2, avg bps: 110, avg pps: 0, avg bpp: 41
Time window: 2024-10-10 14:18:01 - 2024-10-10 14:18:07
Total flows processed: 2, Blocks skipped: 0, Bytes read: 288
Sys: 0.002s flows/second: 892.9      Wall: 0.000s flows/second: 10989.0
```

Obrázek 9: Příklad výstupu

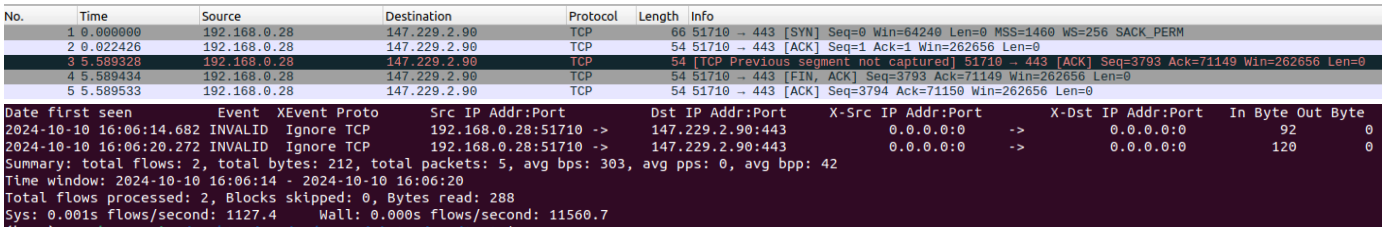
3.4 Provedení testování

Samotné testování proběhlo v pořádku. Pro jednotlivé testy zaměřující se na klíčové vlastnosti programu, viz. sekce 3.1, jsou všechny testy průchozí v souladu s podmínkami v sekci 3.3. Tyto testy jsou automatizované a je tak možno si je snadno zreplikovat, viz úvod sekce 3. Testování timeoutů probíhalo ručně, jelikož nebyla možnost specificky nastavit tyto hodnoty u referenčního exportéru, tudíž se testy nedaly zautomatizovat. Následující část popisuje toto testování a zobrazuje jejich vstupní data a jejich výsledky[7].

[7] Kolektor v následujících testech byl spouštěn v souladu s popisem v sekci 3.2.4

3.4.1 Testování aktivního timeoutu

../p2nprobe localhost:1010 timeouts/z_timeout_test_0.pcap -a 1

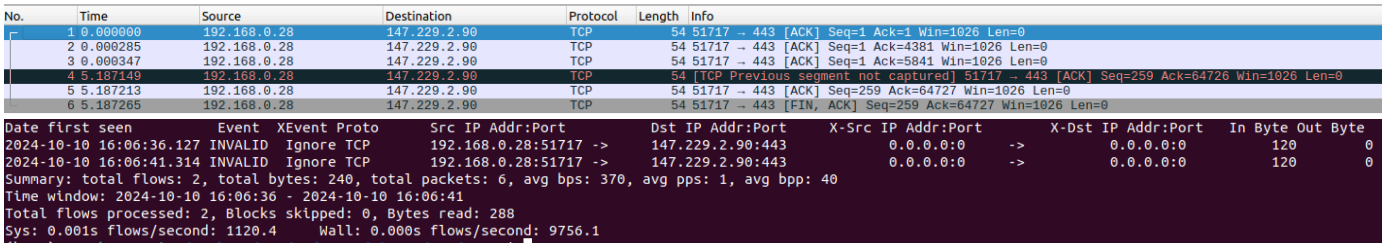


```
No.  Time      Source          Destination     Protocol  Length  Info
     1 0.000000  192.168.0.28    147.229.2.90    TCP         66 51710 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
     2 0.022426  192.168.0.28    147.229.2.90    TCP         54 51710 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
     3 5.589328  192.168.0.28    147.229.2.90    TCP         54 [TCP Previous segment not captured] 51710 → 443 [ACK] Seq=3793 Ack=71149 Win=262656 Len=0
     4 5.589434  192.168.0.28    147.229.2.90    TCP         54 51710 → 443 [FIN, ACK] Seq=3793 Ack=71149 Win=262656 Len=0
     5 5.589533  192.168.0.28    147.229.2.90    TCP         54 51710 → 443 [ACK] Seq=3794 Ack=71150 Win=262656 Len=0

Date first seen          Event  XEvent Proto      Src IP Addr:Port          Dst IP Addr:Port     X-Src IP Addr:Port        X-Dst IP Addr:Port   In Byte Out Byte
2024-10-10 16:06:14.682 INVALID  Ignore TCP       192.168.0.28:51710 ->     147.229.2.90:443          0.0.0.0:0     ->          0.0.0.0:0          92        0
2024-10-10 16:06:20.272 INVALID  Ignore TCP       192.168.0.28:51710 ->     147.229.2.90:443          0.0.0.0:0     ->          0.0.0.0:0         120        0
Summary: total flows: 2, total bytes: 212, total packets: 5, avg bps: 303, avg pps: 0, avg bpp: 42
Time window: 2024-10-10 16:06:14 - 2024-10-10 16:06:20
Total flows processed: 2, Blocks skipped: 0, Bytes read: 288
Sys: 0.001s flows/second: 1127.4     Wall: 0.000s flows/second: 11560.7
```

Obrázek 10: Test 1

../p2nprobe localhost:1010 timeouts/z_timeout_test_0.pcap -a 5

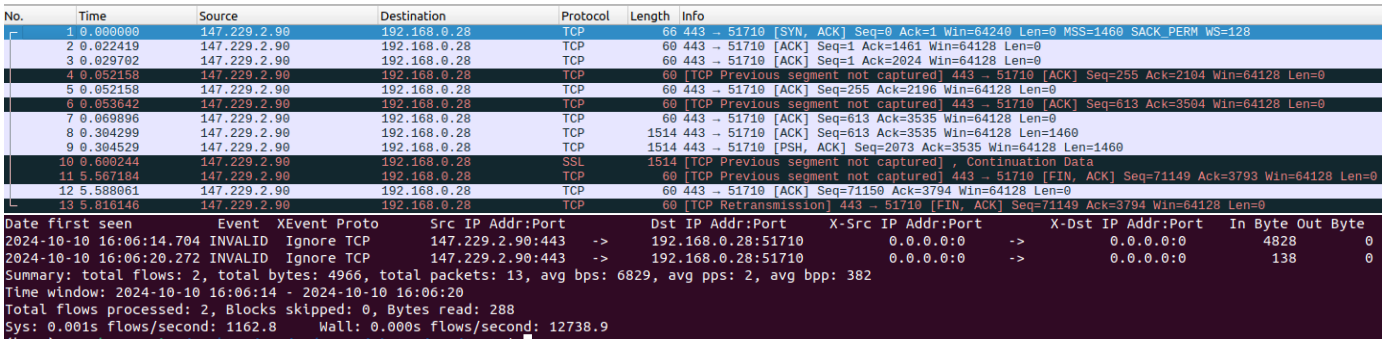


```
No.  Time      Source          Destination     Protocol  Length  Info
     1 0.000000  192.168.0.28    147.229.2.90    TCP         54 51717 → 443 [ACK] Seq=1 Ack=1 Win=1026 Len=0
     2 0.000285  192.168.0.28    147.229.2.90    TCP         54 51717 → 443 [ACK] Seq=1 Ack=4381 Win=1026 Len=0
     3 0.000347  192.168.0.28    147.229.2.90    TCP         54 51717 → 443 [ACK] Seq=1 Ack=5841 Win=1026 Len=0
     4 5.187149  192.168.0.28    147.229.2.90    TCP         54 [TCP Previous segment not captured] 51717 → 443 [ACK] Seq=259 Ack=64726 Win=1026 Len=0
     5 5.187213  192.168.0.28    147.229.2.90    TCP         54 51717 → 443 [ACK] Seq=259 Ack=64727 Win=1026 Len=0
     6 5.187265  192.168.0.28    147.229.2.90    TCP         54 51717 → 443 [FIN, ACK] Seq=259 Ack=64727 Win=1026 Len=0

Date first seen          Event  XEvent Proto      Src IP Addr:Port          Dst IP Addr:Port     X-Src IP Addr:Port        X-Dst IP Addr:Port   In Byte Out Byte
2024-10-10 16:06:36.127 INVALID  Ignore TCP       192.168.0.28:51717 ->     147.229.2.90:443          0.0.0.0:0     ->          0.0.0.0:0         120        0
2024-10-10 16:06:41.314 INVALID  Ignore TCP       192.168.0.28:51717 ->     147.229.2.90:443          0.0.0.0:0     ->          0.0.0.0:0         120        0
Summary: total flows: 2, total bytes: 240, total packets: 6, avg bps: 370, avg pps: 1, avg bpp: 40
Time window: 2024-10-10 16:06:36 - 2024-10-10 16:06:41
Total flows processed: 2, Blocks skipped: 0, Bytes read: 288
Sys: 0.001s flows/second: 1120.4     Wall: 0.000s flows/second: 9756.1
```

Obrázek 11: Test 2

../p2nprobe localhost:1010 timeouts/z_timeout_test_0.pcap -a 4



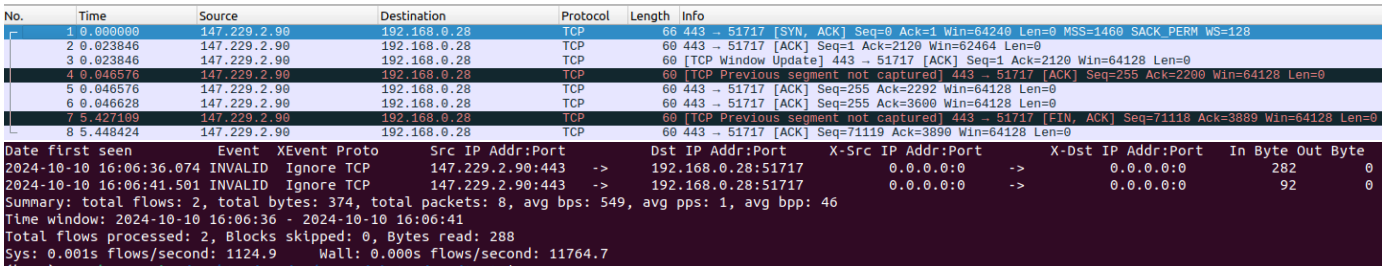
```
No.  Time      Source          Destination     Protocol  Length  Info
     1 0.000000  147.229.2.90    192.168.0.28    TCP         66 443 → 51710 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
     2 0.022419  147.229.2.90    192.168.0.28    TCP         60 443 → 51710 [ACK] Seq=1 Ack=1461 Win=64128 Len=0
     3 0.029702  147.229.2.90    192.168.0.28    TCP         60 443 → 51710 [ACK] Seq=1 Ack=2024 Win=64128 Len=0
     4 0.052158  147.229.2.90    192.168.0.28    TCP         60 [TCP Previous segment not captured] 443 → 51710 [ACK] Seq=255 Ack=2104 Win=64128 Len=0
     5 0.052158  147.229.2.90    192.168.0.28    TCP         60 443 → 51710 [ACK] Seq=255 Ack=2196 Win=64128 Len=0
     6 0.053642  147.229.2.90    192.168.0.28    TCP         60 [TCP Previous segment not captured] 443 → 51710 [ACK] Seq=613 Ack=3504 Win=64128 Len=0
     7 0.069896  147.229.2.90    192.168.0.28    TCP         60 443 → 51710 [ACK] Seq=613 Ack=3535 Win=64128 Len=0
     8 0.304299  147.229.2.90    192.168.0.28    TCP       1514 443 → 51710 [ACK] Seq=613 Ack=3535 Win=64128 Len=1460
     9 0.304529  147.229.2.90    192.168.0.28    TCP       1514 443 → 51710 [PSH, ACK] Seq=2073 Ack=3535 Win=64128 Len=1460
    10 0.600244  147.229.2.90    192.168.0.28    SSL       1514 [TCP Previous segment not captured] , Continuation Data
    11 5.567184  147.229.2.90    192.168.0.28    TCP         60 [TCP Previous segment not captured] 443 → 51710 [FIN, ACK] Seq=71149 Ack=3793 Win=64128 Len=0
    12 5.588061  147.229.2.90    192.168.0.28    TCP         60 443 → 51710 [ACK] Seq=71150 Ack=3794 Win=64128 Len=0
    13 5.816146  147.229.2.90    192.168.0.28    TCP         60 [TCP Retransmission] 443 → 51710 [FIN, ACK] Seq=71149 Ack=3794 Win=64128 Len=0

Date first seen          Event  XEvent Proto      Src IP Addr:Port          Dst IP Addr:Port     X-Src IP Addr:Port        X-Dst IP Addr:Port   In Byte Out Byte
2024-10-10 16:06:14.704 INVALID  Ignore TCP       147.229.2.90:443    ->    192.168.0.28:51710        0.0.0.0:0     ->          0.0.0.0:0        4828        0
2024-10-10 16:06:20.272 INVALID  Ignore TCP       147.229.2.90:443    ->    192.168.0.28:51710        0.0.0.0:0     ->          0.0.0.0:0         138        0
Summary: total flows: 2, total bytes: 4966, total packets: 13, avg bps: 6829, avg pps: 2, avg bpp: 382
Time window: 2024-10-10 16:06:14 - 2024-10-10 16:06:20
Total flows processed: 2, Blocks skipped: 0, Bytes read: 288
Sys: 0.001s flows/second: 1162.8     Wall: 0.000s flows/second: 12738.9
```

Obrázek 12: Test 3

3.4.2 Testování inaktivního timeoutu

../p2nprobe localhost:1010 timeouts/z_timeout_test_0.pcap -i 4

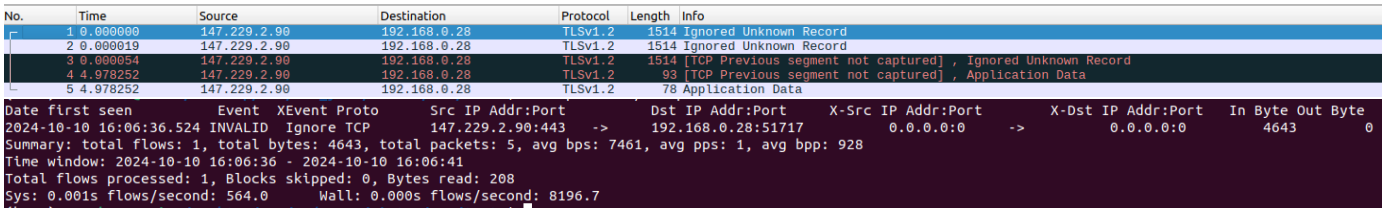


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	147.229.2.90	192.168.0.28	TCP	66	443 → 51717 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
2	0.023846	147.229.2.90	192.168.0.28	TCP	60	443 → 51717 [ACK] Seq=1 Ack=2120 Win=62464 Len=0
3	0.023846	147.229.2.90	192.168.0.28	TCP	60	[TCP Window Update] 443 → 51717 [ACK] Seq=1 Ack=2120 Win=64128 Len=0
4	0.046576	147.229.2.90	192.168.0.28	TCP	60	[TCP Previous segment not captured] 443 → 51717 [ACK] Seq=255 Ack=2200 Win=64128 Len=0
5	0.046576	147.229.2.90	192.168.0.28	TCP	60	443 → 51717 [ACK] Seq=255 Ack=2292 Win=64128 Len=0
6	0.046628	147.229.2.90	192.168.0.28	TCP	60	443 → 51717 [ACK] Seq=255 Ack=3600 Win=64128 Len=0
7	5.427109	147.229.2.90	192.168.0.28	TCP	60	[TCP Previous segment not captured] 443 → 51717 [FIN, ACK] Seq=71118 Ack=3889 Win=64128 Len=0
8	5.448424	147.229.2.90	192.168.0.28	TCP	60	443 → 51717 [ACK] Seq=71119 Ack=3890 Win=64128 Len=0

```
Date first seen          Event  XEvent Proto      Src IP Addr:Port          Dst IP Addr:Port    X-Src IP Addr:Port      X-Dst IP Addr:Port   In Byte Out Byte
2024-10-10 16:06:36.074 INVALID  Ignore TCP       147.229.2.90:443    ->    192.168.0.28:51717         0.0.0.0:0    ->         0.0.0.0:0          282        0
2024-10-10 16:06:41.501 INVALID  Ignore TCP       147.229.2.90:443    ->    192.168.0.28:51717         0.0.0.0:0    ->         0.0.0.0:0           92        0
Summary: total flows: 2, total bytes: 374, total packets: 8, avg bps: 549, avg pps: 1, avg bpp: 46
Time window: 2024-10-10 16:06:36 - 2024-10-10 16:06:41
Total flows processed: 2, Blocks skipped: 0, Bytes read: 288
Sys: 0.001s flows/second: 1124.9     Wall: 0.000s flows/second: 11764.7
```

Obrázek 13: Test 4

../p2nprobe localhost:1010 timeouts/z_timeout_test_0.pcap -i 5

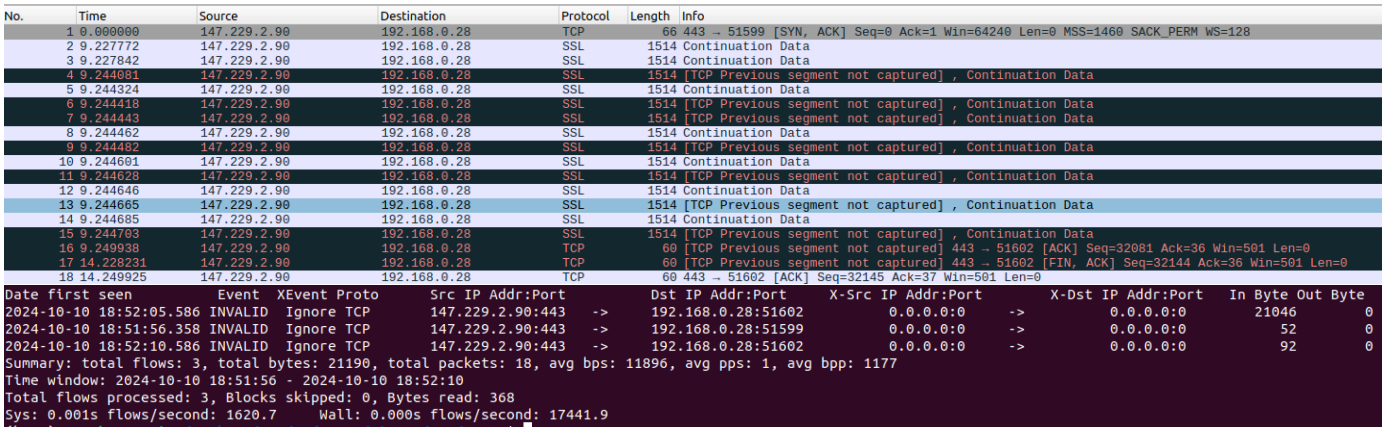


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	147.229.2.90	192.168.0.28	TLSv1.2	1514	Ignored Unknown Record
2	0.000019	147.229.2.90	192.168.0.28	TLSv1.2	1514	Ignored Unknown Record
3	0.000054	147.229.2.90	192.168.0.28	TLSv1.2	1514	[TCP Previous segment not captured] , Ignored Unknown Record
4	4.978252	147.229.2.90	192.168.0.28	TLSv1.2	93	[TCP Previous segment not captured] , Application Data
5	4.978252	147.229.2.90	192.168.0.28	TLSv1.2	78	Application Data

```
Date first seen          Event  XEvent Proto      Src IP Addr:Port          Dst IP Addr:Port    X-Src IP Addr:Port      X-Dst IP Addr:Port   In Byte Out Byte
2024-10-10 16:06:36.524 INVALID  Ignore TCP       147.229.2.90:443    ->    192.168.0.28:51717         0.0.0.0:0    ->         0.0.0.0:0         4643        0
Summary: total flows: 1, total bytes: 4643, total packets: 5, avg bps: 7461, avg pps: 1, avg bpp: 928
Time window: 2024-10-10 16:06:36 - 2024-10-10 16:06:41
Total flows processed: 1, Blocks skipped: 0, Bytes read: 208
Sys: 0.001s flows/second: 564.0      Wall: 0.000s flows/second: 8196.7
```

Obrázek 14: Test 5

../p2nprobe localhost:1010 timeouts/z_timeout_test_0.pcap -i 2



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	147.229.2.90	192.168.0.28	TCP	66	443 → 51599 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
2	9.227772	147.229.2.90	192.168.0.28	SSL	1514	Continuation Data
3	9.227842	147.229.2.90	192.168.0.28	SSL	1514	Continuation Data
4	9.244081	147.229.2.90	192.168.0.28	SSL	1514	[TCP Previous segment not captured] , Continuation Data
5	9.244324	147.229.2.90	192.168.0.28	SSL	1514	Continuation Data
6	9.244418	147.229.2.90	192.168.0.28	SSL	1514	[TCP Previous segment not captured] , Continuation Data
7	9.244443	147.229.2.90	192.168.0.28	SSL	1514	[TCP Previous segment not captured] , Continuation Data
8	9.244462	147.229.2.90	192.168.0.28	SSL	1514	Continuation Data
9	9.244482	147.229.2.90	192.168.0.28	SSL	1514	[TCP Previous segment not captured] , Continuation Data
10	9.244601	147.229.2.90	192.168.0.28	SSL	1514	Continuation Data
11	9.244628	147.229.2.90	192.168.0.28	SSL	1514	[TCP Previous segment not captured] , Continuation Data
12	9.244646	147.229.2.90	192.168.0.28	SSL	1514	Continuation Data
13	9.244665	147.229.2.90	192.168.0.28	SSL	1514	[TCP Previous segment not captured] , Continuation Data
14	9.244685	147.229.2.90	192.168.0.28	SSL	1514	Continuation Data
15	9.244703	147.229.2.90	192.168.0.28	SSL	1514	[TCP Previous segment not captured] , Continuation Data
16	9.249938	147.229.2.90	192.168.0.28	TCP	60	[TCP Previous segment not captured] 443 → 51602 [ACK] Seq=32081 Ack=36 Win=501 Len=0
17	14.228231	147.229.2.90	192.168.0.28	TCP	60	[TCP Previous segment not captured] 443 → 51602 [FIN, ACK] Seq=32144 Ack=36 Win=501 Len=0
18	14.249925	147.229.2.90	192.168.0.28	TCP	60	443 → 51602 [ACK] Seq=32145 Ack=37 Win=501 Len=0

```
Date first seen          Event  XEvent Proto      Src IP Addr:Port          Dst IP Addr:Port    X-Src IP Addr:Port      X-Dst IP Addr:Port   In Byte Out Byte
2024-10-10 18:52:05.586 INVALID  Ignore TCP       147.229.2.90:443    ->    192.168.0.28:51602         0.0.0.0:0    ->         0.0.0.0:0        21046        0
2024-10-10 18:51:56.358 INVALID  Ignore TCP       147.229.2.90:443    ->    192.168.0.28:51599         0.0.0.0:0    ->         0.0.0.0:0           52        0
2024-10-10 18:52:10.586 INVALID  Ignore TCP       147.229.2.90:443    ->    192.168.0.28:51602         0.0.0.0:0    ->         0.0.0.0:0           92        0
Summary: total flows: 3, total bytes: 21190, total packets: 18, avg bps: 11896, avg pps: 1, avg bpp: 1177
Time window: 2024-10-10 18:51:56 - 2024-10-10 18:52:10
Total flows processed: 3, Blocks skipped: 0, Bytes read: 368
Sys: 0.001s flows/second: 1620.7     Wall: 0.000s flows/second: 17441.9
```

Obrázek 15: Test 6

3.5 Závěr testování

Z jednotlivých testů je možno vidět, že aktivní a inaktivní timeout označují toky za expirované správně. Provedení testování ve větším množství je poměrně náročné z důvodu nemožnosti specifického nastavení těchto timeoutů u referenčního exportéru. Nicméně, z těchto testů je odvozeno, že zpracování většího množství nemá na chod implementovaného exportéru žádný vliv.

Všechny přiložené testy pokrývají základní/očekávanou funkčnost exportéru NetFlowv5 a všemi těmito testy aplikace prochází. Všechny tyto testy je možno zreplikovat, buďto spuštěním přiloženého skriptu nebo ručním spuštěním pro otestování jednotlivých timeoutů. Výsledky jednotlivých testů jsou uloženy a je možno si je zpětně procházet a v případě neúspěšného testu tak určit pravděpodobnost neúspěchu.

Z testování aplikace tedy vyplývá, že aplikace je řádně otestována na její požadované/očekávané vlastnosti.

4 Návod na použití

V tabulce 1 jsou vyobrazeny všechny parametry, které se dají použít při spuštění aplikace. Povinné paramtery musí být vždy přítomné. Před spouštěním exportéru je vhodné ujistit se, že je zapnut kolektor na příslušné adrese, že jednotlivé soubory existují a že uživatel spouštějící aplikaci má příslušná práva.

Spouštění aplikace:

./p2nprobe address:port file [switches]

kde `switches` jsou:

- -a value, Aktivní timeout v sekundách
- -i value, Neaktivní timeout v sekundách
- -d, Debugovací mód

4.1 Příklady spuštění

```
./p2nprobe localhost:1010 file.pcap
```

Spustí program pro zpracování paket ze souboru file.pcap a odešle toky na kolektor na adrese localhost:1010.

```
./p2nprobe localhost:1010 file.pcap -a 10
```

Aktivní timeout je specifikován na dobu 10s.

```
./p2nprobe localhost:1010 file.pcap -i 10
```

Inaktivní timeout je specifikován na dobu 10s.

```
./p2nprobe localhost:1010 file.pcap -a 10 -i 5
```

Aktivní timeout je specifikován na dobu 10s a inaktivní na dobu 5s.

```
./p2nprobe localhost:1010 file.pcap -d
```

Aktivován debugovací mód pro výpis aktuálního stavu programu. Určen primárně pro ladění programu.

5 Závěr

Aplikace `p2nprobe`, je určena pro monitorování a statistické vyobrazení vytížení sítě společně s NetFlowv5 kolektorem. Zpracovává zachycenou komuniakci ze souboru `.pcap`, agreguje jednotlivé pakety do toků a následně jednotlivé toky odesílá na adresu kolektoru v síti.

Reference

[1] CISCO. *Cisco IOS Flexible NetFlow*. 2006. Dostupné z: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/flexible-netflow/product_data_sheet0900aecd804b590b.html. Navštíveno 24.9.2024.

[2] CISCO. *NetFlow Export Datagram Format*. 2007. Dostupné z: https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.html. Navštíveno 24.9.2024.

[3] IBM. *NetFlow V5 formats*. 2022. Dostupné z: https://www.ibm.com/docs/en/npi/1.3.1?topic=versions-netflow-v5-formats. Navštíveno 24.9.2024.

[4] MANAGEENGINE. *What is a NetFlow Collector?* 2002. Dostupné z: https://www.manageengine.com/products/netflow/what-is-netflow.html?nfa-index-flowtypes. Navštíveno 24.9.2024.

[5] PCAP. *Pcap - Man Page*. 1999. Dostupné z: https://www.tcpdump.org/manpages/pcap.3pcap.html. Navštíveno: 01.10.2024.

[6] PCAP. *Pcap_loop - Man Page*. 1999. Dostupné z: https://www.tcpdump.org/manpages/pcap_loop.3pcap.html. Navštíveno: 01.10.2024.

[7] PCAP. *Pcap_open_offline - Man Page*. 1999. Dostupné z: https://www.tcpdump.org/manpages/pcap_open_offline.3pcap.html. Navštíveno: 01.10.2024.