FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Projekt do předmětu ISA

Generování NetFlow dat ze zachycené síťové komunikace

14. listopadu 2022 Filip Jahn (xjahnf00)

Obsah

1 Úvod

Cílem projektu bylo naprogramování NetFlow exportéru, který ze zachycených síťových dat ve formátu pcap vytvoří záznamy NetFlow, které odešle na kolektor. NetFlow je protokol, který slouží k monitorování síťového toku pro detailní přehled, co se v naší infrastruktuře děje. Možnost vidět do sítě je klíčové pro její údržbu a bezpečnost.

Spouštění programu umožňuje zadání přepínačů pro zadání pcap souboru, IP adresy kolektoru, příp. UDP port, interval aktivního i neaktivního časovače a posledně velikost flow-cache, tj. kolik flow můžeme mít maximálně uloženo v mapě.

V první kapitole si popíšeme, jaké knihovny byly využity pro realizaci projektu. Uvedeme si seznam globálních proměnných a jejich obhajobu, za jakým účelem použity. V následující kapitole se podíváme, jak probíhá inicializace pcap rozhranní. Zde si také uvedeme, k čemu slouží filterStr a nebo jakým způsobem vytváříme spojení na kolektor. Dále nás čeká rozebrání paketu. Z čeho se skládá a jak z něj získáme veškeré potřebné informace. A tak se samovolně dostaneme do fáze zpracování flows, jak s nimi nakládáme, jak se zpracovávají a k čemu vůbec jsou. Pak nás čeká už pouze exportovat vypršelé flows na kolektor a uzavřít spojení. Samozřejmě nesmí chybět testování, které je popsáno v poslední části dokumentace.

2 Globální prostor

2.1 Knihovní funkce

Zde jsou popsány pouze knihovny, které z mého pohledu stojí za zmínku, leč jsou použity pro práci s pakety, nebo zasíláním. Ostatní knihovny by měly být standardní. Jelikož na Merlinovi překlad funguje a já nic jiného nedoinstalovával, neměla by být potřeba žádné instalace.

Knihovna	Využití
`arpa`	Převody adres na řetězce.
`netinet`	Přetypování paketu, díky tomu získání dat.
`pcap`	Aplikační rozhranní pro odchyt síťové komunikace.
`sys/socket`	Využití primárně funce send, která je upravena pro potřeby zasílání flow.

Tabulka 1: Přehledová tabulka použitých knihoven

2.2 Definovaná makra a globální proměnné

Makra

Název	Délka	Popis
`ETH_HDR`	14	Ethernetová hlavička má vždy 14 bajtů. O tolik se posouváme, chceme-li číst další hlavičku.

Tabulka 2: Makra a jejich využití

Proměnné

Globální proměnné reprezentují počáteční argumenty aplikace, mapu pro ukládání flows a dále pomocné promenné, které se buď využívají ke statistikám, či pro korektní zasílání na kolektor. K naplnění proměnných argumentů zadanými hodnotami využíváme funkci parse_arguments. Absencí předávání argumentů jako parametr napříč funkcemi se tak řešení stává čitelnějším.

- **string:**
  - pcapFile_name_– Zvolený soubor s příponou `pcap`, ze kterého máme číst.
  - netflow_collector_ip_– IP adresa kolektoru, na kterou máme odesílat flows.
  - netflow_collector_port_– Port kolektoru, na který máme odesílat flows.
- **int:**
  - active_timer_– Aktivní časovač, po jehož přetečení se odesílají flows na kolektor.
  - inactive_timer_– Neaktivní časovač, po jehož přetečení se odesílají flows na kolektor.
  - flowcache_size_– Velikost cache, ve které ukládáme flows.
- **ostatní:**
  - flow_map_– Mapa, do které si ukládáme jednotlivé flows sdružené s jejich klíčem. Zde rozumějme:
    - **Flow** – Flow je skupina paketů, které jsou součástí jedné a té samé konverzace mezi dvěma koncovými body. Každou flow definujeme podle jejího klíče. Pakety, které mají stejný klíč se agregují do jedné flow.

    * **Key** – Pětice dat: `Source IP, Destination IP, Source port, Destination Port, Protocol`.
  - time_now_– Čas aktuálně přečteného paketu.
  - time_first_pkt_– Čas prvně přečteného paketu.
  - sending_packets_– Vektor paketů, které mají být odeslány. Řazení probíhá těsně před odesláním.
- **struktura flow:**
  - flow_header_– Netflow hlavička. Vytvořena přesně podle struktury v citaci[2].
  - flow_body_– Netflow tělo. Struktura vytvořena přesně podle struktury v citaci[1].

3 Startovní bod

3.1 Analýza argumentů

Po spuštění programu se vrháme na syntaktickou analýzu zadaných argumentů při spuštění programu ve funkci `parse_arguments`. Celá kontrola je založena na cyklu, který prochází zadané argumenty za pomoci knihovní funkce getopt_long [9]. Využitím této funkce si značně ulehčíme práci. Zároveň zde probíhá sémantická kontrola. Jestliže přijdeme na to, že uživatel zadal nesmyslné požadavky, nastavujeme defaultní hodnotu 0.

Při správně zadaných argumentech přiřazujeme hodnoty náležícím proměnným a pokračujeme inicializací `pcap` rozhraní.

3.2 Vytvoření spojení

K vytvoření spojení mezi exportérem a kolektorem využívám mou upravený soubor `echo-udp-client2`, který nám poskytl pan doc. Matoušek [11]. Do funkce `create_connection` předáváme proměnnou s adresou a portem. Ta už nám pak zajistí, že si vytvoříme client socket a korektně se na něj připojíme. Pokud by nastala nějaká chyba, aplikace vypíše error a ukončí se, leč další práce by byla beze smyslu.

3.3 Inicializace `pcap`

Proniknutí do úvodní problematiky „jak začít“ značně usnadňuje blog, který uvádím v citaci [5]. Inicializace je zde krásně vysvětlena a ukázána na praktických příkladech. Funkce main využívá velkou většinu popsaných funkcí.

3.3.1 Filtrovací pravidla

Pro čtení z pcap souboru využíváme knihovní funkci `pcap_open_offline` [10] z knihovny `pcap`[1]. Pokud otevření proběhne bez problémů, bude dalším úkolem nastavení filtru paketů.

Řetězec `filterStr` udržuje informace o tom, které hlavičky paketů se mají odchytávat, příp. na jakých portech sledovat síťový provoz. Abychom mohli filtr aplikovat v naší problematice, musíme nejprve převést daný řetězec na filtr programu, kterému pcap rozumí. Kompilaci provádíme za použití knihovní funkce `pcap_compile` [6] ze stejné knihovny jako v předchozím případě.

V neposlední řadě nám zbývá filtr vložit do pcap rozhraní. K tomuto účelu je využita třetí funkce, a to `pcap_setfilter` [8]. Pakliže se do tohoto bodu nevyskytly žádné problémy, jsme plně připraveni. Pokud by se však nějaké objevily, bude vyhlášena chybová hláška na `stderr` a program bude ukončen s návratovou hodnotou 2.

3.3.2 Čtení jednotlivých paketů

Nyní přichází čas na procházení pcap souboru a čtení jednotlivých paketů, jenž je vykonáváno ve funkci `pcap_loop`[7]. Tato funkce načítá jeden paket za druhým. S každým načtením paketu se volá funkce, jejíž název je součástí argumentů pcap_loop. V našem případě `process_packet`.

Po zpracování všech paketů dochází k uzavření čtení a uzavření spojení a následuje ukončení aplikace s návratovým kódem 0.

[1] Viz Tabulka 2.1

4 Zpracování paketů

Po načtení paketu si vytvoříme instanci struktury flow. 2.2 Pro zpracovávání jednotlivých paketů se přesouváme do funkce `process_packet`.

Nyní musíme rozhodnout, o který internetový protokol se jedná. Tato informace je nesena v první hlavičce. Jedná se o `ethernetovou hlavičku`. Tato hlavička je vždy 14 bajtů dlouhá[2]. Vytvoříme si proto strukturu: `struct ether_header *ipvNum`, do které uložíme paket přetypovaný na danou strukturu. Tím získáme možnost se odkázat na parametr ve struktuře, kde je k nalezení typ následující hlavičky. V této aplikaci však uvažujeme pouze pakety typu IPv4. Veškeré ostatní internetové protokoly jsou zahazován.

4.1 IPv4

Jediným přijímaným protokolem je tedy `IPv4`. Využíváme zde strukturu `struct iphdr *ipHeader`, která je blíže popsaná zde v dokumentaci [12]. Abychom ale mohli přijatý paket přetypovat na tuto strukturu, musíme se prvně posunout o ethernetovou hlavičku. Tak učiníme bitovým posuvem o 14 bajtů. Tím se dostaneme na `IP hlavičku`, která poskytuje parametr `protocol`, díky němuž můžeme určit, jaký protokol transportní vrstvy je přítomen. Potřebná data ukládáme do proměnné `ipHeader`.

Následně si ukládáme čas prvního paketu do globální proměnné.[3] Dále si ukládáme zjistitelné parametry z ipHeader do flow struktury. Zde můžeme tedy vyčíst prototyp, ToS, zdrojovou a cílovou IP adresu a velikost celého paketu.

Přepínač na jednotlivé protokoly pak rozhoduje na základě ipHeader -> protocol. Zde máme výběr ze tří možností: ICMP, TCP nebo UDP. Jednotlivé protokoly se zpracovávají v příslušných funkcích: `icmp_v4`, `udp_v4`, `tcp_v4`.

Myšlenka všech funkcí je velice podobná, avšak se liší v odlišném přetypování na dané struktury protokolů a následným výběrem hodnot z parametrů sPort, dPort.

Před vstupem do jednotlivých funkcí voláme funkci `check_timers`, o které si řekneme později v sekci 6.1.

ICMP, TCP a UDP

Jednou z prvních věcí, kterou děláme ve všech funkcích, je vytvoření klíče 2.2. Pokud se jedná o TCP, nebo UDP protokol, vytáhneme si ještě data o zdrojovém a cílovém portu. ICMP má porty nastavené na 0. Pak již pokračujeme ve funkcích stejně.[4].

V mapě `flow_map_` 2.2 se podíváme, zdali daný klíč již existuje. 5.2

- Pokud ano, aktualizujeme jednotlivé prvky jeho flow.
- Pokud ne, zakládáme novou flow a přidáme ji do mapy.

[2] Připomenutí: Tabulka 2.1

[3] Pro připomenutí sekce 2.2.

[4] Rozdělení do daných funkcí je pouze kvůli přehlednosti kódu. Ačkoli se jisté pasáže opakují, lépe jsem se pak v kódu vyznal.

5 Zpracování flow

5.1 Vytvoření nové flow

Pro vytvoření nové flow užíváme funkci `create_flow`, jejíž parametr je odkaz na novou flow, která se má přidat do mapy.

Při jakékoliv manipulaci s flows jsou nejdůležitější časové údaje, od kterých se vše odvíjí. V naší aplikaci neběží reálný čas, jako je tomu při skutečném netflow v reálných systémech. My čteme již někdy v minulosti načtená data z .pcap souboru, tudíž se veškeré časy určují čtenými pakety.

Jako aktuální čas aplikace uvažujeme tedy vyčtený čas právě načteného paketu. Do hlavičky `flow.sysUpTime` ukládáme čas běhu systému, což rozumíme jako rozdíl mezi časem nyní s časem prvního načteného paketu. Dále si ukládáme časy `flow.First` a `flow.Last`, které prozatím přebírají čas z flow.sysUpTime. Do `flow.unix_secs` ukládáme sekundy ze struktury time_now_.secs a obdobně pro `flow.unix_nsecs`, tady ale pozor! Zde musíme vynásobit 1000, protože ve struktuře timeval máme mikrosekundy.

Ostatní informace byly již vyplněny ve funkci process_packet viz 4, nebo jsou nastaveny na výchozí hodnotu 0. Pokud se jedná o TCP, nastavujeme ještě flags do `flow.flags`.

Následně novou flow ukládáme s jejím klíčem do mapy `flow_map_`.

5.2 Aktualizace flow

Pro aktualizování flow se přesouváme do funkce `update_flow_record` se dvěma vstupními parametry. Prvním je ukazatel na existující záznam a druhý na novou flow, jíž chceme stávající aktualizovat.

S časy je to zde obdobné. Do lokální proměnné `sysUpTime` si vypočteme za použití funkce `timersub`[5] [4] čas běhu systému.

Poté vkládáme tuto informaci v milisekundách[6] do `flow.sysUpTime` a `flow.Last`. Přičteme informaci o velikosti nového paketu a zvýšíme počítadlo paketů v hlavičce flow. Pokud se jedná o TCP pakety, zde využíváme komulativní OR operaci pro korektní zápis nových flags.

[5] Při výpočtech času používáme vždy tuto funkci.

[6] Pro výpočet milisekund jsem si vytvořil funkci `getMillis`, která přijímá argument struktury timeval, která obsahuje jednak sekundy od roku 1970, tzv. epoch time, a druhak mikrosekundy, které jsou doplňkem k sekundám do daného času.

6 Odesílání flow

6.1 Kontrola časovačů

Jak již bylo zmíněno dříve, časová razítka jsou jedním z nejdůležitějších údajů. Na základě vypršených časovačů odesíláme flows na kolektor. Jejich vypršení obsluhuje funkce `check_timers`, která pro každý záznam v mapě vypočte aktivní a neaktivní časovač. Pokud některý z nich přetekl, vypočteme si pro danou flow její čas, kdy přetekla.

Tím, že víme, který časovač přetekl, můžeme do flow.SysUpTime přičíst daný vypršený časovač. Tím řekneme v jaký moment časovač přetekl a tedy v jakém čase budeme odesílat danou flow. Vkládáme ji do vektoru pro odeslání `sending_packets_`.

Pokud se jedná o TCP flow, kontrolujeme ještě její flags, jestli náhodou komunikace již neskončila. Když je totiž nastaven příznak `0x01`, víme že byla komunikace ukončena a my tak záznam můžeme rovnou odeslat na kolektor. Zde pak nastavujeme časy podle nýnějšího paketu.

6.2 Zasílání na kolektor

Po projití celé mapy se zanořujeme do funkce `send_flows`, která má zprvu za úkol seřadit vektor s flows, které mají být zaslány, od nejmladšího po nejstarší. Nejstarší proto, že odesíláme flows z vektoru zezadu. K řazení využívám funkci `sort` [3].

Tuto funkcionalitu jsem implementoval pro co největší přiblížení se reálnému exportéru NetFlow. Uveďme si následující příklad, který nám demonstruje problematiku, kdybychom řazení neřešili, a jak jsem se to rozhodl řešit já:

Algoritmus 1: Příklad problematiky (ne)řazení dle časů

1 `flow1 { t = 10, t = 20 }`
2 `flow2 { t = 9, t = 18 }`
3 a kupř. `inactive_timer = 5`
4 Přijde nový paket v čase `t = 26`
5 V takovém případě ve svém řešení odesílám prvně `flow2`, a teprve pak `flow1` a to s časy `{flow2 t=23}, {flow1 t=25}`
6 **Kdybychom řazení a vypočtení exportu neřešili, pak bychom zkrátka zasílali flows postupně, jak leží v mapě, následně bychom na kolektoru dostali `{flow1 t = 26}, {flow2 t = 26}`... Což by neodpovídalo reálnému NetFlow (dle mého názoru) a proto jsem udělal takovýto algoritmus.**

Po seřazení si vytahujeme jednotlivé flows z vektoru a mažeme je z tabulky. Zde inkrementujeme hodnotu `flow.flow_sequence`, která nám udává, kolik flows jsme již na kolektor odeslali. Flow odesíláme upravenou funkcí `send_data` [11], která ho zašle na kolektor.

7 Testování

Testování proběhlo porovnáním výstupu mého programu s referenčním programem `softflowd`.

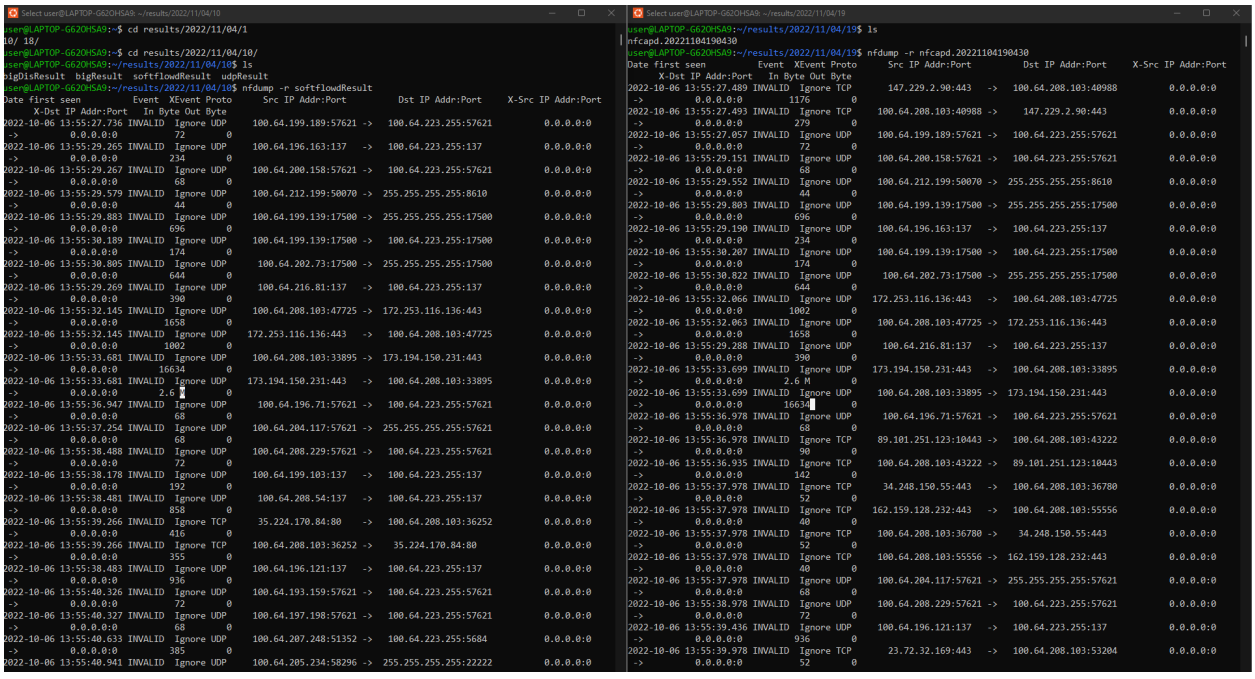
Na levé straně vidíme výstup aplikace softflowd, která byla spuštěna příkazem: `sudo softflowd -v 5 -n 127.0.0.1:2055 -r big.pcap`. Na pravé straně pak vidíme výstup mého NetFlow exportéru, který také odeslal data na kolektor ležící na ip adrese 127.0.0.1 (localhost) s portem 2055. Výpis exportu, jak lze vyčíst z obrázku, pak za použití referenčního řešení `nfdump -r {nazev_souboru}`.

Pokud nahlédneme pořádně, zjistíme, že řádky přesně nesedí. Je tomu proto, že aplikace softflowd odesílá UPD pakety ihned, avšak já čekám na vypršení časovačů.

Ohledně testování na Merlinovi... Tam boužel práva na otevírání portů nemáme a nenapadla mě možnost, jak otestovat aplikaci jako takovou. Testování funkčnosti však proběhlo na platformě Ubuntu WSL pro Windows a Ubuntu pro VM, takže projekt považuji za otestovaný na platformě Linux.

Abych si byl jist, že mé řešení dává smysl, poprosil jsem kolegu z univerzity, aby mi poslal svůj výstup z pcap souboru, který jsem mu zaslal. Dostali jsme stejné výsledky, což značí minimálně stejné pochopení problematiky.

- Ukázka výstupu

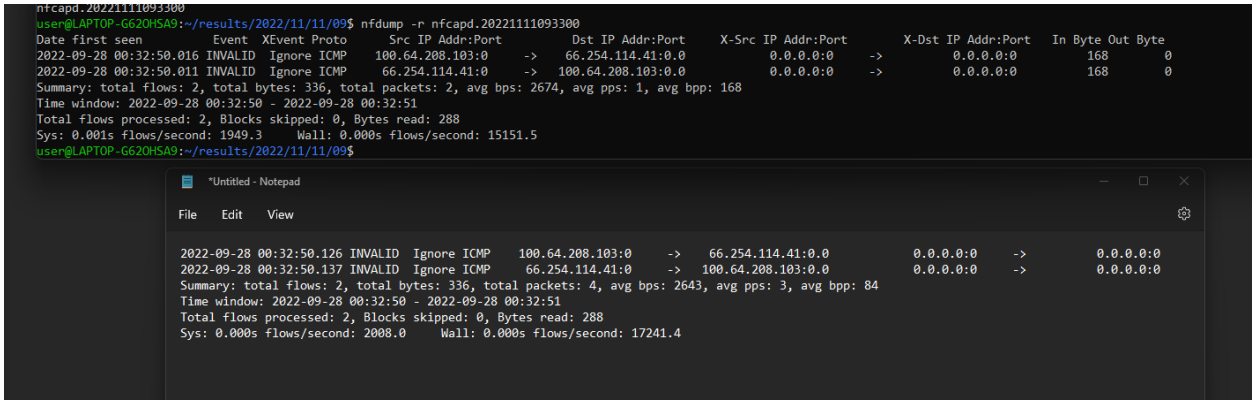


Obrázek 1: Výstup mého Netflow s porovnáním aplikace softflowd

- Kompilace

Obrázek 2: Překlad na Merlinovi

- Srovnání s kolegou



Obrázek 3: Srovnání výstupů s jedním z kolegů

Reference

[1] CISCO.COM. *NetFlow Export Datagram Format Body* [online]. Poslední změna 2022 [cit. 2. listopadu 2022]. Dostupné na: <https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.html#wp1006186>.

[2] CISCO.COM. *NetFlow Export Datagram Format Header* [online]. Poslední změna 2022 [cit. 2. listopadu 2022]. Dostupné na: <https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.html#wp1006108>.

[3] CPLUSPLUS.COM. *Manuál k používání funkce sort* [online]. Poslední změna 30.10.2022 [cit. 4. listopadu 2022]. Dostupné na: <https://cplusplus.com/reference/algorithm/sort/>.

[4] HTTPS://LINUX.DIE.NET. *Manuál k používání funkce timersub* [online]. Poslední změna 4.11.2022 [cit. 4. listopadu 2022]. Dostupné na: <https://linux.die.net/man/3/timersub>.

[5] LINUX.DIE.NET. *Blog o úvodní inicializaci* [online]. Poslední změna 1. listopadu 2020 [cit. 25. dubna 2021]. Dostupné na: <https://www.tcpdump.org/pcap.html>.

[6] LINUX.DIE.NET. *Dokumentace pcap_compile* [online]. Poslední změna 1. listopadu 2020 [cit. 25. dubna 2021]. Dostupné na: <https://linux.die.net/man/3/pcap_compile>.

[7] LINUX.DIE.NET. *Dokumentace pcap_loop* [online]. Poslední změna 1. listopadu 2020 [cit. 25. dubna 2021]. Dostupné na: <https://linux.die.net/man/3/pcap_loop>.

[8] LINUX.DIE.NET. *Dokumentace pcap_setfilter* [online]. Poslední změna 1. listopadu 2020 [cit. 25. dubna 2021]. Dostupné na: <https://linux.die.net/man/3/pcap_setfilter>.

[9] LINUX.DIE.NET. *Getopt_long* [online]. Poslední změna 1. listopadu 2020 [cit. 25. dubna 2021]. Dostupné na: <https://linux.die.net/man/3/getopt_long>.

[10] LINUX.DIE.NET. *Pcap_open_offline* [online]. Poslední změna 1. listopadu 2020 [cit. 25. dubna 2021]. Dostupné na: <https://docs.oracle.com/cd/E88353_01/html/E37845/pcap-open-offline-3pcap.html>.

[11] MOODLE.VUT.CZ. *Síťové aplikace a správa sítí* [online]. Poslední změna 4.11.2022 [cit. 4. listopadu 2022]. Dostupné na: <https://moodle.vut.cz/course/view.php?id=231021>.

[12] SITES.UCLOUVAIN.BE. *Dokumentace ip hlavičky* [online]. Poslední změna 1. listopadu 2020 [cit. 25. dubna 2021]. Dostupné na: <https://sites.uclouvain.be/SystInfo/usr/include/netinet/ip.h.html>.