Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

ConnTop – nástroj pro administraci síťové komunikace v Linuxu

Bakalářská práce

Daniel Hryzbil

Školitel: Ing. Jan Fesl, Ph.D.

České Budějovice 2019

HRYZBIL, D., 2019: ConnTop – nástroj pro administraci síťové komunikace v Linuxu. [ConnTop – tool for network communication administration in Linux. Bc. Thesis, in Czech.] – 30 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

The aim of this bachelor thesis is to design and implement a tool for monitoring network traffic in real time. Theoretical part describes possibilities of collecting information about network traffic. Practical part describes design of the resulting tool implemented in C++ programming language. The tool has a terminal user interface and uses the conntrack table inside the Linux kernel to effectively obtain network traffic information. The tool also supports client-server mode.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne ....

Daniel Hryzbil

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Janu Feslovi, Ph.D. za vedení mé bakalářské práce a cenné rady. Dále bych chtěl poděkovat svým rodičům za jejich podporu během mého studia.

Obsah

Kapitola 1

Úvod

Počet zařízení připojených k počítačové síti se neustále zvyšuje. Velký podíl na tom mají různá moderní „smart“ a IoT zařízení, která kromě komunikace s ostatními zařízeními v rámci lokální sítě navíc často komunikují i se vzdálenými servery někde v internetu. Společně s rostoucím počtem připojených zařízení se zvyšují i rychlosti linek. Domácí internetové připojení s rychlostí v řádu stovek Mb/s se pomalu stává běžnou záležitostí. Tyto skutečnosti způsobují neustálé zvyšování nároků nejen na síťovou infrastrukturu jako takovou, ale i na systémy určené k monitorování síťového provozu.

Ve velkých sítích monitorování provozu zpravidla není až takový problém, protože je zde možné využít techniku zrcadlení síťového provozu na výkonný server vyhrazený pouze pro monitorování, případně použít hardwarové sondy přímo určené pro tento účel. Následně je celé řešení zastřešeno speciálním a často komerčním softwarem, který zajišťuje sběr a zpracování dat z monitorovacích serverů a sond zapojených v jednotlivých částech sítě. Velkým problémem je však vysoká cena výsledného řešení a také potřeba vyhrazené infrastruktury. Z těchto důvodů je toto řešení nevhodné pro menší sítě, kde přidání vyhrazeného zařízení pro monitorování síťového provozu není možné z ekonomického nebo jiného důvodu.

V menších sítích je tedy potřeba provádět monitorování přímo na síťových prvcích, přes které prochází provoz, který chceme monitorovat. Typicky se jedná o hraniční router

dané sítě. Ten většinou v menších sítích nedisponuje výkonným hardwarem, a může se stát, že bude značně zatížen zpracováním samotného síťového provozu. To znamená, že monitorování provozu na takovém zařízení musí být co nejefektivnější a spotřebovávat minimum systémových zdrojů. Právě toto je jedna z oblastí, na kterou se zaměřuje aplikace vytvořená v rámci této práce.

Navržená a vytvořená aplikace kromě klasického režimu, kdy je stejně jako v případě už existujících řešení celá aplikace spuštěna jako jeden proces na daném zařízení, podporuje navíc i rozdělení na klientskou a serverovou část. Na daném zařízení je tedy spuštěna pouze serverová část, která zde sbírá informace o aktuálně probíhajícím síťovém provozu. Následně se klientská část spuštěná už na normálním počítači připojí přes síť k danému zařízení se serverovou částí aplikace, a průběžně získává informace o zde probíhajícím provozu. Získané informace o síťovém provozu na daném zařízení jsou poté zobrazeny uživateli. Celá aplikace je navržena tak, že její serverová část obsahuje jen opravdu nezbytné komponenty, a následné zpracování a zobrazení informací o síťovém provozu je plně řešeno v její klientské části.

Vzhledem k tomu, že aplikace slouží k monitorování aktivních síťových spojení a množství dat, která přes ně prochází, tak není potřeba přenášet obsah síťového provozu. Navíc kompletní seznam aktivních síťových spojení se přenáší pouze při startu klientské aplikace nebo při ztrátě synchronizace mezi serverovou a klientskou částí, jinak se přenáší pouze změny. To ve výsledku znamená, že množství přenášených dat mezi oběma částmi aplikace je poměrně nízké, takže nejsou potřeba žádné vyhrazené linky mezi serverovou a klientskou částí a jejich komunikace tak může probíhat v rámci normálního síťového provozu.

Kompletní zdrojový kód aplikace pod licencí GPL3 je k dispozici na serveru GitHub ve veřejném repozitáři na webové adrese `https://github.com/ccomrade/conntop` a také na přiloženém DVD (verze 1.0.0). Součástí je i dokumentace (soubor README) popisující postup sestavení aplikace ze zdrojového kódu a následné zprovoznění.

Kapitola 2

Teoretický rozbor

2.1 Možnosti sběru informací o síťovém provozu

Základní částí každé aplikace pro monitorování síťového provozu je část, která nějakým způsobem sbírá informace o probíhajícím síťovém provozu na daném zařízení. Běžně se využívá zachytávání celého provozu, a to buď ve formě paketů síťové vrstvy, nebo přímo rámců linkové vrstvy. Aplikace následně ze zachycených dat získává potřebné informace. Velkou výhodou takového řešení je fakt, že je možné pracovat s obsahem síťové komunikace. Aplikace určené pro analýzu obsahu síťového provozu tak ani nemají jinou možnost, jak získat potřebná data. Mezi tyto aplikace patří například známý Wireshark [1]. Další výhodou tohoto řešení je existující pcap API [2], které je určené pro zachytávání síťového provozu a je implementované na všech velkých operačních systémech (většinou ve formě knihovny `libpcap`). Aplikace využívající toto rozhraní tak může poměrně snadno fungovat na různých platformách. Kromě pcap API lze také využít přímo rozhraní konkrétního operačního systému a získat tím přístup k dalším dodatečným informacím, které samotné pcap API neposkytuje. Například linuxové jádro pro tento účel podporuje speciální druh soketu `AF_PACKET` [3], přes který lze síťový provoz nejen přijímat, ale i odesílat.

Zachytávání síťového provozu má ale mimo jiné i velkou nevýhodu, kterou je efektivita. S rostoucím množstvím síťového provozu se rychle zvyšují nároky na výpočetní výkon

potřebný pro jeho zpracování. Navíc se zde každý paket musí zpracovat minimálně dvakrát. Jednou v rámci normálního zpracování síťové komunikace, a podruhé v samotné aplikaci, která zachytává síťový provoz. V zařízeních, která slouží jako router, se většina síťového provozu odbaví relativně rychle, protože primární účel těchto zařízení je pouze přeposílání paketů mezi jednotlivými sítěmi (routování). Na takovém zařízení potom aplikace využívající zachytávání provozu může snadno svojí činností spotřebovat více systémových zdrojů, než kolik je potřeba pro samotné routování. To může představovat velký problém při vysokých datových tocích, kdy dané zařízení už nemusí mít dostatek výpočetního výkonu pro činnost aplikace zachytávající síťový provoz. V lepším případě poté budou informace o síťovém provozu nekompletní. V horším případě dojde navíc vlivem přetížení k ovlivnění ostatních funkcí daného zařízení.

Alternativou k neefektivnímu zachytávání síťového provozu je využití existujících informací, které vznikají jako vedlejší efekt standardního zpracování síťové komunikace. Typickým příkladem je tabulka obsahující aktivní TCP sokety, kterou si operační systémy vnitřně udržují, a většinou je možné nějakým způsobem získat její obsah. Například v operačních systémech využívajících linuxové jádro je obsah této tabulky mapován do souborů `/proc/net/tcp` a `/proc/net/tcp6` [4]. Z této tabulky je možné poměrně efektivně získat seznam navázaných TCP spojení, který lze využít pro účely základního monitorování provozu. Stejně jako u všech ostatních řešení sběru informací o síťovém provozu využívajících existující informace, tak i zde změna datového toku u jednotlivých spojení nezvyšuje ani nesnižuje množství výpočetního výkonu potřebného pro samotný sběr informací o síťovém provozu. Na rozdíl od řešení využívajících zachytávání síťového provozu, kde se musí neustále zpracovávat všechny pakety. Změna počtu spojení už zde nějaký výkonnostní dopad má, nicméně i při vysokém počtu síťových spojení by toto řešení mělo být efektivnější než zpracování jednotlivých paketů při zachytávání síťového provozu.

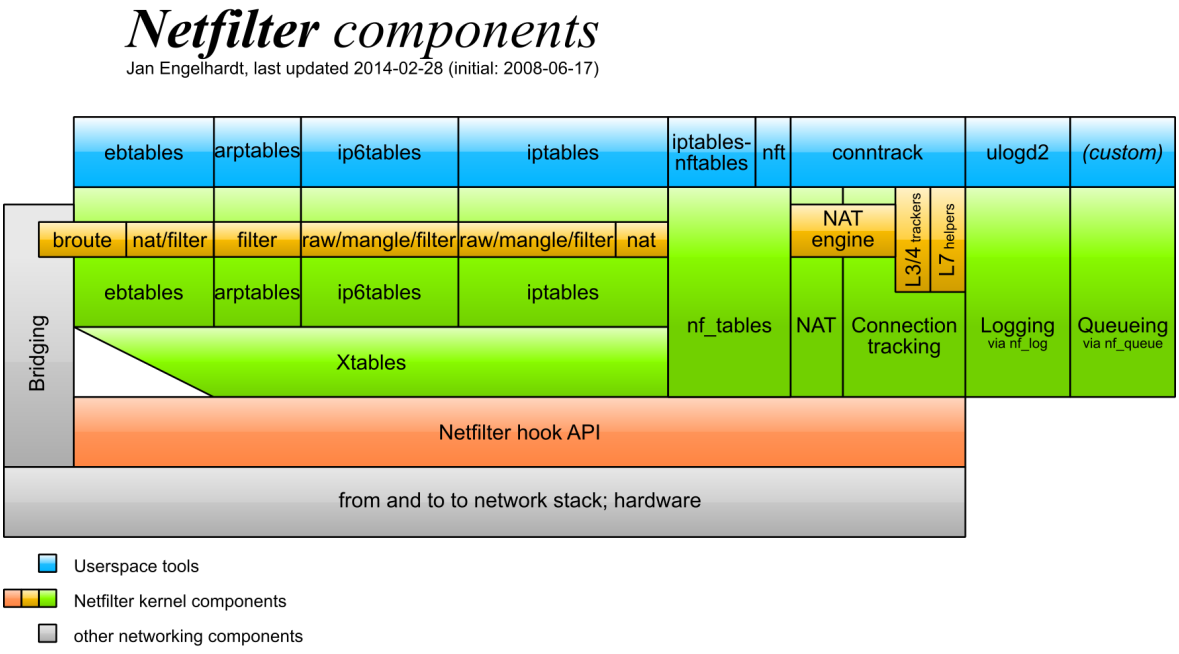
Stejně jako zachytávání síťového provozu tak i tato metoda má své nevýhody. Předně je to množství informací, které lze tímto způsobem získat. Zmíněná TCP tabulka například vůbec neobsahuje informaci o množství přenesených dat přes jednotlivá spojení.

Dále v této tabulce z principu nejsou zahrnutá routovaná spojení, což je poměrně zásadní problém, pokud chceme provádět monitorování provozu na routeru. Navíc jsou zde jen TCP spojení. Podobné tabulky sice existují i pro další protokoly, ale pokud je daný protokol nespojový (jako například protokol UDP), tak jeho tabulka neobsahuje jednotlivá spojení, protože zde nedochází k vytvoření nového soketu při navázání komunikace.

Řešení, které nemá výše popsané nevýhody při zachování vysoké efektivity, je využití conntrack tabulky popsané dále.

2.2 Conntrack tabulka

Tato tabulka se používá pro sledování síťových spojení (connection tracking), tj. přiřazování přijatých a odeslaných paketů k jednotlivým spojením, které je nutné pro implementaci stavového firewallu a překladu adres (NAT). Je součástí frameworku Netfilter [5], který se nachází uvnitř linuxového jádra.



Obrázek 2.1: Struktura jaderného frameworku Netfilter [6]

Obsahem této tabulky jsou tedy veškerá síťová spojení na daném zařízení, a to včetně těch routovaných. Navíc jsou zde ve formě spojení zahrnuty i protokoly, které jsou nespojové, jako například protokoly UDP a ICMP. Volitelně je zde také pro každé spojení dostupná informace o množství přenesených dat ve formě počtu přijatých a odeslaných paketů a bajtů. Tato funkce je ve výchozím stavu vypnuta, ale je možné ji zapnout pomocí parametru `acct` jaderného modulu `nf_conntrack` [7]. Nastavení tohoto parametru je vhodné provést při startu systému, protože při aktivaci za běhu bude množství přenesených dat počítáno pouze u nově vytvořených spojení.

Pro získání obsahu conntrack tabulky existují dvě základní možnosti. První možností je soubor `/proc/net/nf_conntrack`, do kterého je obsah conntrack tabulky mapován. Pro čtení obsahu tohoto souboru je vyžadováno oprávnění uživatele root. Druhou efektivnější možností je získání obsahu tabulky přes Netlink soket [8]. Tento speciální druh soketů se používá v linuxových systémech převážně pro komunikaci mezi jádrem operačního systému a procesy v uživatelském prostoru. Je možné využít jej také pro komunikaci mezi jednotlivými procesy. Kromě získání obsahu conntrack tabulky je možné touto cestou také měnit její obsah. Dále je možné aktivovat automatické zasílání změn v tabulce (vytvoření, odstranění a změna stavu spojení). To umožňuje ještě efektivnější sběr informací o síťovém provozu. Stejně jako u zmíněného souboru, tak i zde je vyžadováno oprávnění uživatele root. Navíc je zde také možnost udělit potřebná oprávnění pomocí `CAP_NET_ADMIN` capability [9].

Struktura dat posílaných přes Netlink sokety při komunikaci s conntrack tabulkou a dalšími komponentami jaderného frameworku Netfilter není záměrně nikde přesně definována. Místo toho jsou k dispozici knihovny, které implementují přímo API pro práci s jednotlivými komponentami. Díky tomu se vývojáři aplikací využívající tyto komponenty nemusí příliš zabývat samotnou komunikací s jádrem operačního systému a mohou se soustředit na vývoj své aplikace. Také to umožňuje vývojářům jádra provádět případné změny v těchto komponentách, protože následně stačí upravit jen zmíněné knihovny bez potřeby upravovat každou aplikaci, která je používá. Pro práci s conntrack tabulkou je určena knihovna `libnetfilter_conntrack` [10].

Využití conntrack tabulky pro sběr informací o síťovém provozu se zdá být velmi výhodné, nicméně i zde lze nalézt nějaké nevýhody. Předně je to problém aktivace této tabulky. K její aktivaci totiž dochází jen tehdy, pokud je obsah této tabulky vyžadován jinou komponentou uvnitř linuxového jádra. Například pokud uživatel aktivuje zmíněný stavový firewall nebo překlad adres (NAT), tak zároveň dojde i k aktivaci této tabulky. To ale znamená, že pokud tyto komponenty nejsou aktivní, tak není aktivní ani samotná tabulka a nelze tedy získat její obsah pro účely monitorování síťového provozu.

Ve starších verzích linuxového jádra je možné conntrack tabulku aktivovat i ručně, a to zavedením jaderných modulů `nf_conntrack_ipv4` a `nf_conntrack_ipv6`. Od verze linuxového jádra 4.14 při tomto postupu ale nedojde k samotnému zahájení sledování spojení [11], takže conntrack tabulka zůstává prázdná. Od verze 4.19 jsou také zmíněné jaderné moduly sloučeny [12] do hlavního modulu `nf_conntrack`. Pro aktivaci conntrack tabulky v moderních linuxových systémech je tedy potřeba aktivovat alespoň minimální stavový firewall. Tato skutečnost do jisté míry znevýhodňuje aplikace pro monitorování síťového provozu využívající obsah conntrack tabulky. Naštěstí v nadcházející verzi linuxového jádra 5.1 bude přidán nový parametr `enable_hooks` [13] jaderného modulu `nf_conntrack`, díky kterému bude opět možné conntrack tabulku aktivovat ručně pouhým zavedením tohoto modulu.

Při ruční aktivaci conntrack tabulky pouze pro účely monitorování síťového provozu také vyvstává otázka, zda se tento postup ještě stále nějak významně odlišuje od neefektivního zachytávání síťového provozu přímo v aplikaci. Samotná conntrack tabulka se ale nachází přímo v jádře operačního systému, takže z principu bude vždy efektivnější než zachytávání a následné zpracování síťové komunikace v aplikaci. Navíc aplikace pro monitorování síťového provozu využívající conntrack tabulku nemusí být spuštěna po celou dobu běhu systému pro zajištění získání kompletních informací o síťovém provozu, na rozdíl od aplikace využívající zachytávání síťového provozu. V případě využití conntrack tabulky totiž stačí zajistit aktivaci jen této tabulky při startu systému místo samotné aplikace.

Jako další nevýhodu využití conntrack tabulky pro sběr informací o síťovém provozu lze také považovat skutečnost, že ve výše popsané formě je tato tabulka přítomna pouze v operačních systémech založených na linuxovém jádře. Některé další operační systémy pravděpodobně obsahují podobnou tabulku využitou například také pro implementaci stavového firewallu, ale možnosti přístupu k jejímu obsahu budou nejspíše zcela odlišné, případně nebude vůbec možné získat její obsah.

I přes výše popsané nevýhody je využití conntrack tabulky stále nejlepším dostupným řešením z pohledu efektivity a množství informací o síťovém provozu, které lze tímto způsobem získat. Z tohoto důvodu je tato tabulka využívána aplikací vytvořenou v rámci této práce. Obsah tabulky je v aplikaci získáván přímo z jádra operačního systému pomocí zmíněné knihovny `libnetfilter_conntrack` pro zajištění maximální efektivity.

2.3 Další zdroje informací

Samotné informace získané ze síťového provozu zpravidla nestačí a pro praktické použití je potřeba získat další informace. Především jde o síťové adresy a porty, které v číselné podobě většinou nemají moc velkou vypovídající hodnotu z pohledu uživatele aplikace pro monitorování síťového provozu.

Některé IP adresy v internetu mají přiřazené doménové jméno, které lze získat pomocí reverzního dotazu v systému DNS. Toto jméno někdy obsahuje mimo jiné i stručný popis funkce a umístění zařízení s danou IP adresou. Pro získání tohoto jména je možné použít funkci getnameinfo [14], která je dostupná ve většině operačních systémů. Stejným způsobem lze získat i názvy síťových služeb přiřazených k některým portům. Většinou se jedná o tzv. „dobře známé“ (well-known) porty v rozsahu $0 - 1023$ a také tzv. „registrované“ (registered) porty v rozsahu $1024 - 49152$ [15].

Dalším zdrojem užitečných informací o síťových adresách jsou různé GeoIP databáze. Tyto databáze slouží primárně k získání informace o geografické poloze jednotlivých IP adres, ale často obsahují i další informace. Pro aplikaci vytvořenou v rámci této práce

byla zvolena volně dostupná databáze GeoLite2 [16] od firmy MaxMind. Tato databáze obsahuje kromě informací o geografické poloze také čísla autonomních systémů (AS), do kterých jednotlivé adresy patří, společně s názvy organizací, které tyto autonomní systémy spravují. Firma MaxMind poskytuje také několik placených databází, které obsahují ještě podrobnější informace. Velkou výhodou těchto databází je možnost jejich stažení ve formě souboru, takže je možné tyto databáze používat lokálně a provádět v nich rychlé a efektivní vyhledávání. Na rozdíl od většiny ostatních řešení, která poskytují pouze webové API pro přístup ke svým databázím. Takové řešení je značně neefektivní (obzvláště při větším množství dotazů) a také snižuje robustnost výsledné aplikace, protože přidává závislost na externí službě. Navíc zde dochází k vytváření síťového provozu, což pro aplikaci zabývající se monitorováním síťového provozu není úplně ideální. Právě možnost stažení celé databáze byl hlavní důvod pro výběr zmíněné GeoIP databáze. Menší nevýhodou stažených GeoIP databází je potřeba provádět jejich aktualizaci, protože informace v nich obsažené se mohou měnit, nicméně použitá databáze se velikostně pohybuje řádově v jednotkách až desítkách MB, takže její občasná aktualizace by neměla představovat velký problém.

Existují ještě další zdroje informací, jako například WHOIS databáze, nicméně výsledná aplikace zatím využívá jen výše zmíněné zdroje informací.

2.4 Existující řešení

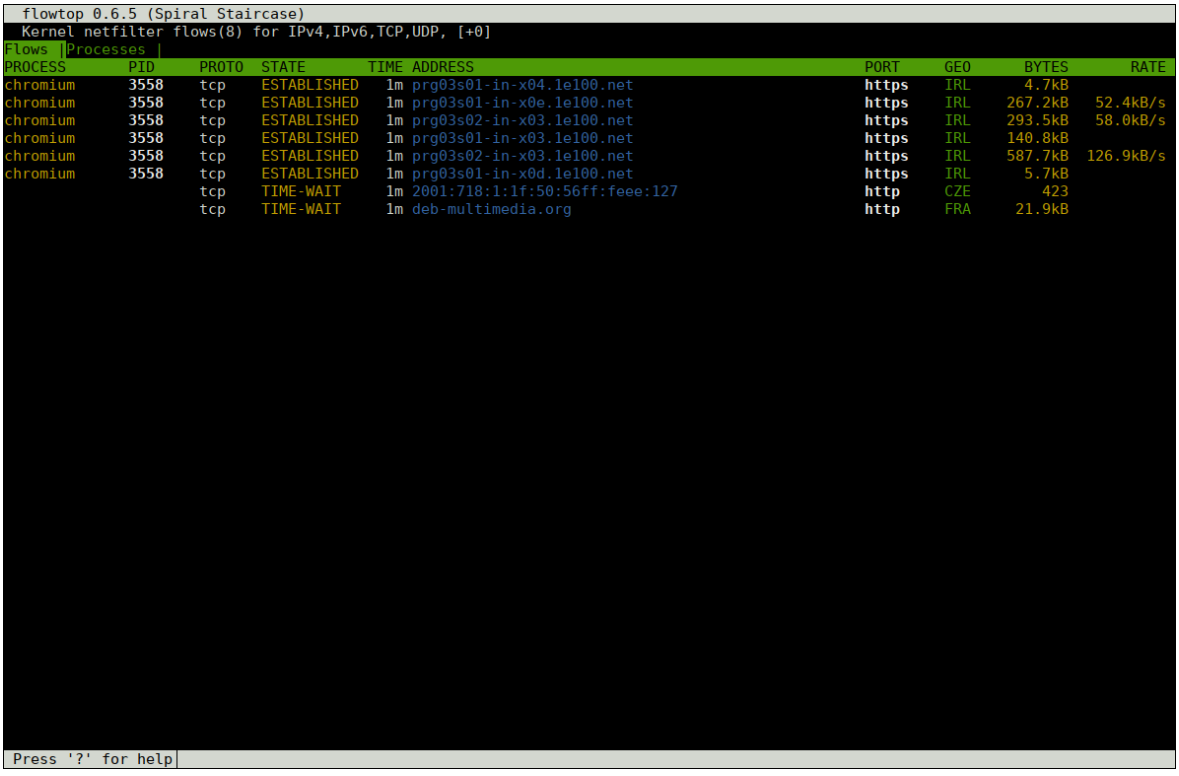
V této podkapitole jsou popsány nalezené existující nástroje, které využívají zmíněnou conntrack tabulku pro efektivní získání informací o síťovém provozu.

2.4.1 Nástroj flowtop

Nástroj flowtop z balíku netsniff-ng [17] je určen pro monitorování aktivních síťových spojení v reálném čase a má interaktivní terminálové uživatelské rozhraní. Mezi zobrazovanými informacemi jsou všechny důležité informace poskytované samotnou conntrack tabulkou. Kromě základních protokolů UDP a TCP jsou zde podporovány také

další protokoly, jako například ICMP a SCTP. Nástroj dále automaticky získává doménová jména IP adres, názvy služeb přiřazené některým portům, název procesu a PID u lokálních spojení a také geografickou polohu IP adres. Pro získání této polohy je využívána starší volně dostupná databáze GeoLite od firmy MaxMind. Tato databáze je ale označena jako zastaralá a z oficiálních webových stránek už ani není možné tuto databázi stáhnout [18]. Existující databáze tohoto typu už tedy pravděpodobně neobsahují aktuální informace. Vývojářům je doporučován přechod na novou databázi GeoLite2 zmíněnou výše.

Během úvodního testování ještě před samotným vývojem aplikace vytvořené v rámci této práce vykazoval tento nástroj značnou nestabilitu a docházelo k jeho častým pádům. Nyní se zdá, že tyto problémy byly již odstraněny a aktuálně nejnovější verze tohoto nástroje 0.6.5 nevykazuje žádné problémy se stabilitou a jedná se tedy o celkem dobře použitelný nástroj.



Obrázek 2.2: Ukázka nástroje flowtop

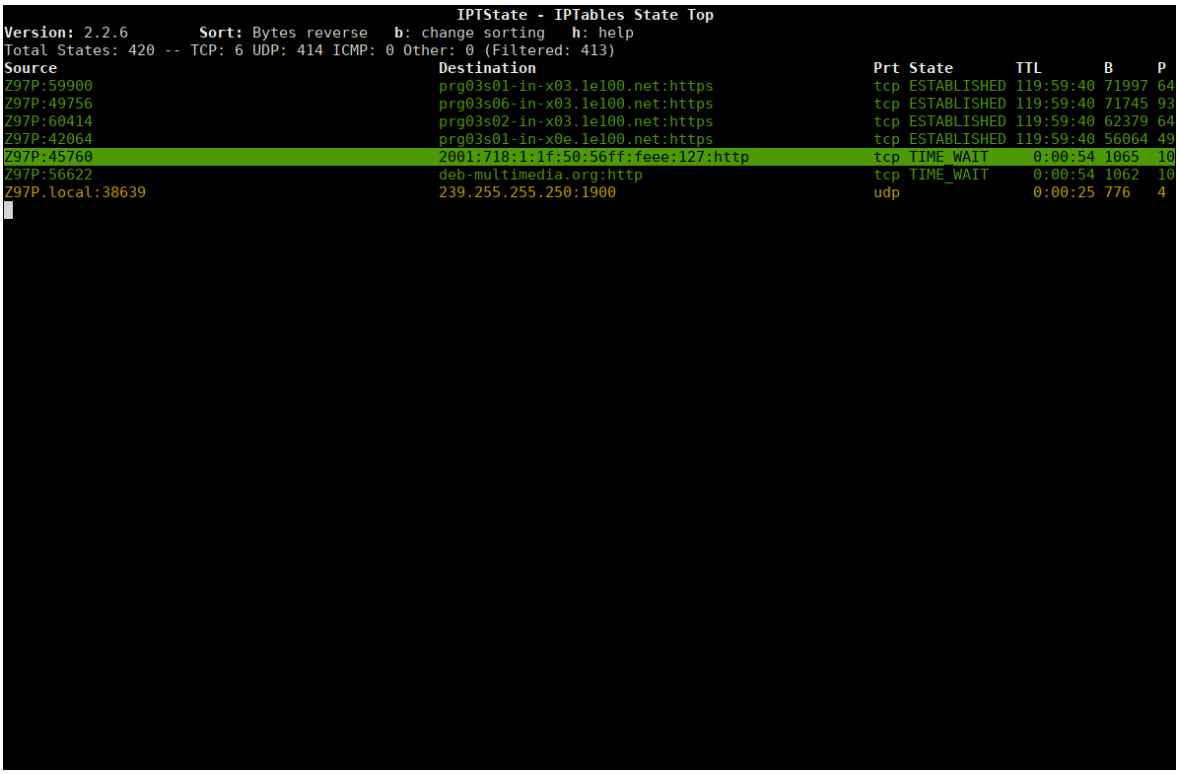
2.4.2 Nástroj iptstate

Stejně jako předchozí nástroj, tak i tento nástroj [19] je určen pro monitorování aktivních síťových spojení v reálném čase a má také interaktivní terminálové uživatelské rozhraní. Využívány jsou zde opět všechny důležité informace poskytované samotnou conntrack tabulkou. Na rozdíl od předchozího nástroje jsou zde ale jako dodatečné informace získávána pouze doménová jména IP adres a názvy služeb přiřazené některým portům. Naopak výhodou tohoto nástroje je možnost základního filtrování zobrazených síťových spojení a také možnost řazení seznamu síťových spojení.

Při testování tohoto nástroje bylo objeveno několik nedostatků plynoucích z nepříliš robustního návrhu tohoto nástroje. Například při stisku jakékoliv klávesy dojde zároveň i k obnovení zobrazovaných údajů, takže například při rychlejším pohybu kurzoru v seznamu síťových spojení dochází k rychlým změnám některých údajů. Nástroj také z nějakého důvodu vytváří velké množství DNS dotazů a tím i značné množství UDP spojení. Tato spojení jsou navíc ve výchozím stavu záměrně odfiltrována ze zobrazeného seznamu síťových spojení.

2.4.3 Nástroj conntrack

Tento nástroj z balíku conntrack-tools [20] ovládaný z příkazové řádky je určen pro správu obsahu conntrack tabulky. Nejedná se tedy o interaktivní aplikaci pro monitorování síťového provozu. Kromě jednorázového získání obsahu conntrack tabulky s podporou filtrování je možné pomocí tohoto nástroje provádět také úpravu a odstranění jednotlivých spojení v tabulce a dokonce i vytvářet nová spojení. Nástroj dále umožňuje odstranit veškerá spojení v tabulce pomocí jediného příkazu a také v reálném čase zobrazovat události conntrack tabulky. Primárně lze tento nástroj využít pro rychlé zobrazení obsahu conntrack tabulky a také pro různé experimenty.



Obrázek 2.3: Ukázka nástroje iptstate

Kapitola 3

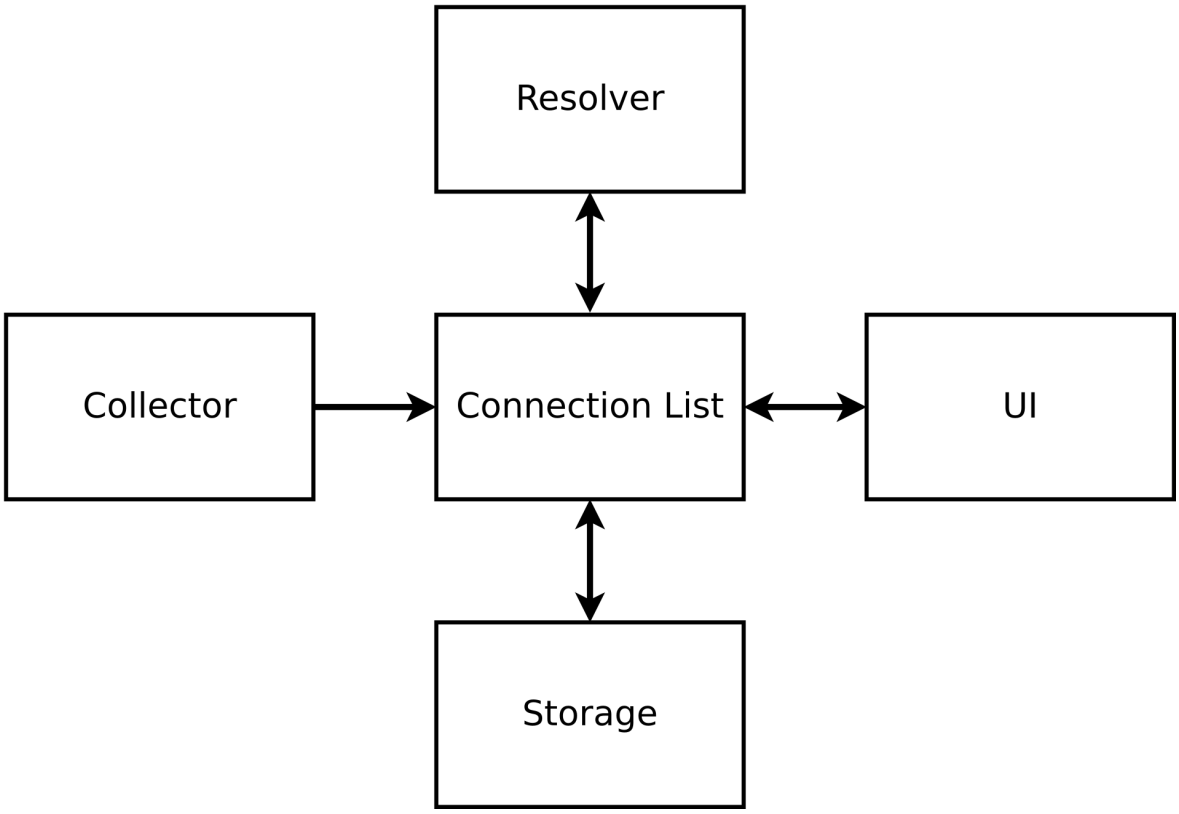
Návrh

3.1 Název aplikace

Pro výslednou aplikaci byl zvolen název „conntop“. Podle dostupných informací z internetových vyhledávačů neexistuje žádný projekt s tímto názvem, takže by nemělo dojít ke kolizi názvu s již existující aplikací. Samotný název tvoří spojení slov „connection“ a „top“. První představuje zaměření aplikace na monitorování síťových spojení. Druhé odkazuje na původní interaktivní nástroj top [21] s terminálovým uživatelským rozhraním, který v různých unixových systémech slouží k zobrazení seznamu běžících procesů a základních informací o systému, jako je využití procesoru a operační paměti. Všechny zobrazené informace jsou periodicky obnovovány. Právě tato vlastnost je důvodem, proč se název tohoto nástroje používá jako základ názvu některých později vytvořených nástrojů, které fungují na podobném principu.

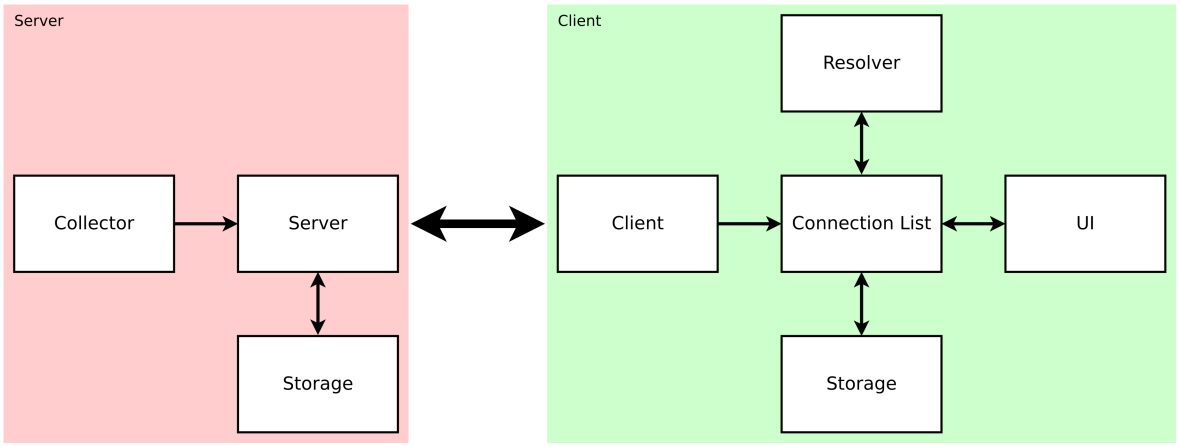
3.2 Blokové schéma aplikace

Hlavní komponentou aplikace je Connection List. Zde se udržuje seznam aktivních síťových spojení, dále nazývaný jako seznam spojení, který je zobrazen uživateli přes komponentu uživatelského rozhraní (UI). Také jsou zde zpracovány veškeré požadavky uživatele na provedení určité operace se seznamem spojení (například změna řazení).



Obrázek 3.1: Blokové schéma aplikace

Všechna aktivní síťová spojení jsou společně s ostatními daty uložena v komponentě Storage a samotný seznam spojení obsahuje jen odkazy na zde uložená síťová spojení. Seznam spojení také může obsahovat jen malou aktuálně viditelnou část síťových spojení podle možností uživatelského rozhraní. Veškeré změny dat uložených v komponentě Storage se provádí pouze přes komponentu Connection List, aby bylo případně možné zároveň aktualizovat i seznam spojení a data zobrazená v uživatelském rozhraní. Sběr informací o síťovém provozu zajišťuje komponenta Collector. Odtud se následně tyto informace v pravidelných intervalech předávají do komponenty Connection List, kde dochází k aktualizaci obsahu komponenty Storage a také k případné aktualizaci vlastního seznamu spojení. Společně s tím se všechny nové adresy a porty předávají do komponenty Resolver, kde se získávají dodatečné informace. Získané informace jsou poté předány zpět komponentě Connection List, která zajistí jejich uložení v komponentě Storage a také případnou aktualizaci seznamu spojení a následně i dat zobrazených v uživatelském rozhraní.



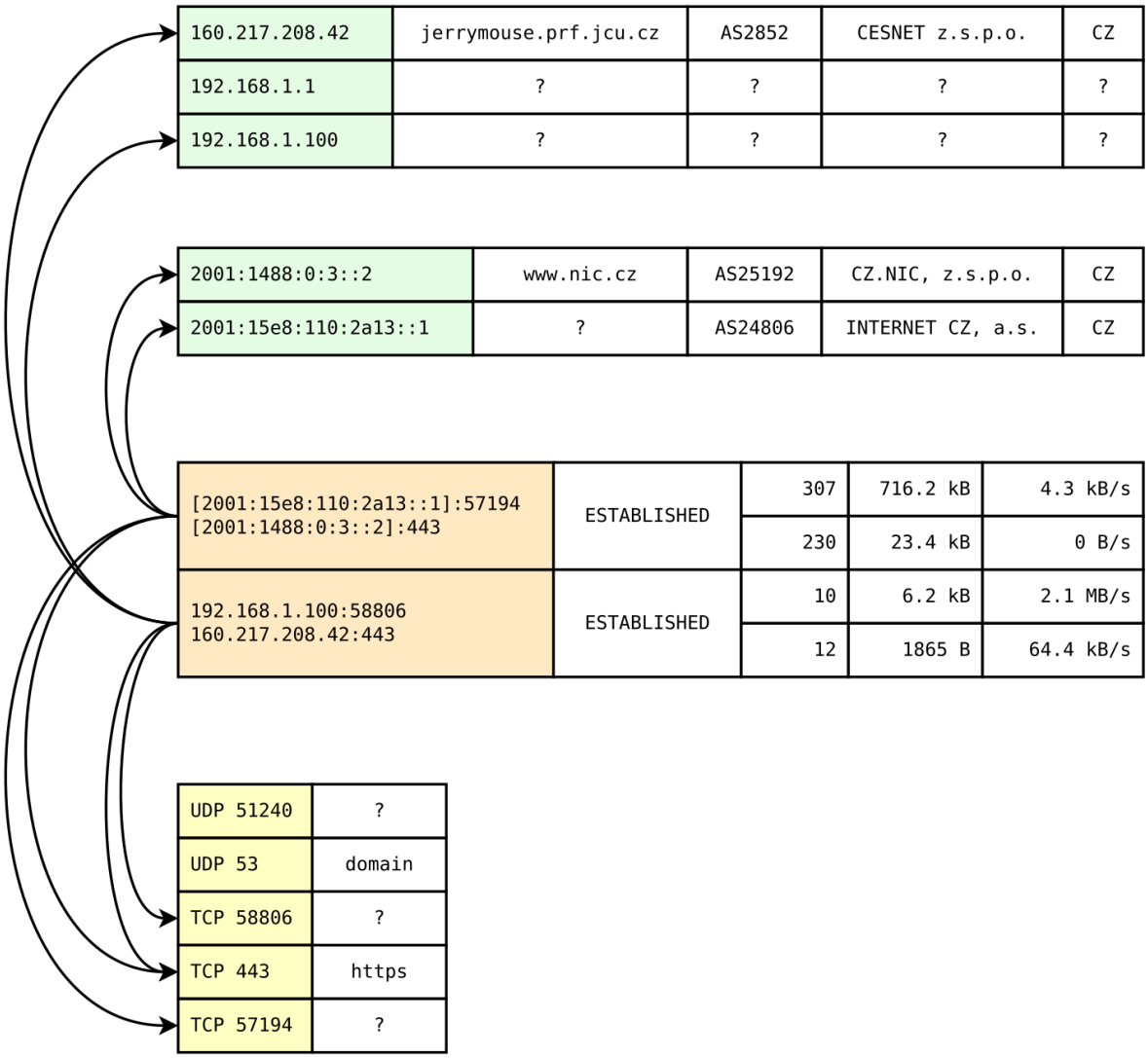
Obrázek 3.2: Blokové schéma aplikace v režimu klient-server

Při provozu aplikace v režimu klient-server se na klientské straně místo komponenty Collector použije komponenta Client. Vše ostatní zde zůstává stejné. Komponenta Client funguje z pohledu ostatních komponent aplikace stejně jako komponenta Collector, jen místo sběru informací o síťovém provozu ze systému zajišťuje komunikaci se serverovou částí, ze které následně získává tyto informace. Samotný sběr informací o síťovém provozu pomocí komponenty Collector nyní probíhá v serverové části aplikace. Získané informace jsou zde následně předány komponentě Server, která zajišťuje jejich uložení v komponentě Storage. Komponenta Server dále zajišťuje komunikaci s klientskou částí, které jsou získané informace poskytovány. Důležitým rysem celého návrhu je skutečnost, že serverová část aplikace obsahuje jen nezbytně nutné komponenty a získávají se zde pouze informace, které nelze efektivně získat v klientské části.

3.3 Důležité datové struktury

Pro efektivní zpracování všech získaných informací je potřeba dobrý návrh způsobu jejich uložení v rámci aplikace. Ukázka výsledného řešení je na obrázku 3.3, který znázorňuje obsah komponenty Storage zmíněné výše. Všechny získané informace jsou uloženy do celkem 4 oddělených asociativních datových struktur. Jsou to IPv4 adresy, IPv6 adresy, samotná síťová spojení a síťové porty. První barevně zvýrazněná položka ve všech těchto strukturách představuje klíč, který jednoznačně identifikuje danou položku, a

odkazuje na její hodnotu, která obsahuje získané informace o dané položce. V případě struktury obsahující síťová spojení představuje klíč jen odkazy na položky uložené v ostatních strukturách. Jedná se o zdrojovou a cílovou adresu a také na zdrojový a cílový port. Položky nejsou ve svých strukturách uloženy v žádném definovaném pořadí. To umožňuje jako jednotlivé datové struktury použít rychlé hašovací tabulky. Díky tomu je výsledná aplikace schopná efektivně fungovat i při vysokém počtu aktivních síťových spojení.



Obrázek 3.3: Ukázka způsobu uložení informací o síťovém provozu

3.4 Komunikační protokol v režimu klient-server

Po delší úvaze byl jako základ komunikačního protokolu zvolen známý datový formát JSON [22]. Důvodem pro zvolení tohoto datového formátu je právě jeho rozšířenost a poměrně snadná práce s daty v tomto formátu. Právě rozšířenost a standardizace tohoto formátu výrazně zjednodušuje případnou implementaci zde navrženého komunikačního protokolu i do dalších aplikací, které nějakým způsobem využívají nebo získávají informace o síťovém provozu. Formát JSON vnitřně rozlišuje několik různých datových typů (řetězec, číslo, boolean, null, objekt, pole) a má navíc poměrně nízkou režii z pohledu množství přenášených dat. Hlavní nevýhodou použití formátu JSON je potřeba externí knihovny pro práci s tímto formátem a také z pohledu bezpečnosti složitější parsování dat v tomto formátu při přenosu přes síť, nicméně tyto nevýhody jsou převáženy zmíněnými výhodami.

Samotný komunikační protokol je založen na zasílání několika typů zpráv. Jednotlivé zprávy protokolu tvoří vždy jediný validní JSON objekt. Data přenášená tímto protokolem jsou vždy obsažena uvnitř objektu zprávy. Pro přenos jednotlivých zpráv je vyžadován transportní protokol, který zaručuje spolehlivý přenos (například TCP).

Primární účelem navrženého protokolu je přenos dat poskytovaných serverem na klienta. Protokol umožňuje přenos libovolných dat serializovaných do formátu JSON. Protokol naopak neumožňuje klientské straně provádět jakékoliv změny na straně serveru, takže server je schopný bez problému obsluhovat více klientů zároveň.

Komunikace mezi klientem a serverem probíhá zhruba následovně. Klient nejprve naváže spojení se serverem. Pokud je vše v pořádku, dojde na serveru k vytvoření „sezení“ pro daného klienta. Nyní je klient připojen k serveru a může si vyžádat zaslání dat, která potřebuje. Server v pravidelných intervalech posílá najednou všem připojeným klientům krátkou zprávu indikující začátek nového cyklu. V aktuální implementaci je tento interval vždy 1 vteřina. Tato zpráva se používá pro synchronizování stavu klientů a serveru, dále pro detekci ztráty spojení a také pro udržení otevřeného spojení v době, kdy klient nežádá žádná data. Ihned po odeslání této zprávy server začíná se zasíláním dat, která si jednotliví klienti vyžádali. Po dokončení přenosu dat si klient opět

vyžádá potřebná data a následně čeká do začátku dalšího cyklu. Klient si může vyžádat buď kompletní data, nebo jen změny od posledního cyklu, případně nemusí žádat žádná data. Pokud se přenos nestihne dokončit do začátku následujícího cyklu, server dokončí přenos aktuálně odesílaného typu dat a poté ihned pošle opožděnou zprávu indikující začátek nového cyklu. Tím je klient informován o ztrátě synchronizace dat a vyžádá si zaslání kompletních dat.

Podrobnosti o fungování komunikačního protokolu a přesnou strukturu zasílaných zpráv je možné vyčíst ze zdrojového kódu výsledné aplikace.

Při návrhu protokolu bylo počítáno i s možností, že v budoucích verzích aplikace bude protokol změněn, takže server posílá v úvodní zprávě číslo s verzí protokolu, které klient následně při navazování spojení porovná s verzí svého protokolu. Pokud jsou verze rozdílné, tak klient ukončí navazování spojení a informuje uživatele o chybě.

Protokol neobsahuje podporu pro autentizaci klientů ani šifrování přenosu. Přidání těchto funkcí by v některých případech bylo spíše na škodu z důvodu složitější konfigurace aplikace, potřeby dalších knihoven a také zvýšení složitosti zdrojového kódu. Výsledná aplikace je určena primárně pro správce systémů a sítí, takže autentizaci klientů a šifrování přenosu je možné efektivně řešit například pomocí VPN nebo SSH tunelů. Základní autentizaci je rovněž možné provést přímo na úrovni sítě pomocí firewallu. Přenášené informace o síťovém provozu mohou být považovány za citlivé, takže pokud komunikace mezi klientskou a serverovou částí aplikace prochází přes nějakou nedůvěryhodnou (veřejnou) síť, tak je vhodné zajistit šifrování přenosu například pomocí zmíněného VPN nebo SSH tunelu.

3.5 Uživatelské rozhraní

Aplikace obsahuje interaktivní terminálové (textové) uživatelské rozhraní. Velkou výhodou tohoto typu uživatelského rozhraní je možnost efektivního vzdáleného přístupu. Je tedy možné k aplikaci s tímto typem uživatelského rozhraní spuštěné na vzdáleném počítači bez problému přistupovat z lokálního počítače například přes SSH. Výsledný

datový tok mezi lokálním a vzdáleným počítačem je navíc velmi nízký, protože se přenášejí pouze znaky reprezentující obsah terminálu a vstupy z klávesnice. Naopak velkou nevýhodou je samotná skutečnost, že se jedná o textové uživatelské rozhraní, takže je zpravidla možné zobrazovat informace pouze ve znakové podobě, což u některých aplikací může představovat zásadní problém.

Design uživatelského rozhraní je inspirován existujícím nástrojem htop [23] určeným pro zobrazení a správu procesů běžících v systému. Hlavní okno aplikace obsahuje seznam aktivních síťových spojení a je vodorovně rozděleno na 3 části. První část obsahuje různé stavové informace, jako například celkové množství síťových spojení nebo stav některých nastavení seznamu síťových spojení. Druhou část, která zabírá nejvíce místa, tvoří samotný seznam aktivních síťových spojení, kde na prvním řádku je vždy záhlaví tohoto seznamu tvořené názvy důležitých sloupců. Poslední třetí část tvoří jediný řádek nacházející se na dolním okraji okna aplikace. Tento řádek známý z různých aplikací s interaktivním terminálovým uživatelským rozhraním obsahuje popis aktuální funkcionality přiřazené klávesám F1 − F10. Aplikace dále obsahuje několik dialogových oken. Jedná se o okna s nastavením seznamu síťových spojení a okno s podrobnými informacemi o vybraném síťovém spojení. Také je zde okno s nápovědou, které je možné vyvolat klávesou F1. Nápověda obsahuje popis kláves, kterými se aplikace ovládá. Primárně se aplikace ovládá kurzorovými šipkami. Jednotlivé prvky uživatelského rozhraní jsou barevně odlišeny pro větší přehlednost. Použité barevné schéma je rovněž inspirováno nástrojem htop.

Kapitola 4

Implementace

4.1 Vývojové nástroje

Zdrojový kód aplikace je napsán v jazyce C++, konkrétně ve standardu C++14. Pro přeložení zdrojového kódu do výsledné aplikace je tedy vyžadován kompilátor s podporou tohoto standardu. Důvodem pro nepoužití aktuálně nejnovějšího standardu C++17 je skutečnost, že tento standard je podporován pouze v nejnovějších verzích kompilátorů, které zatím nemusejí být na všech systémech dostupné.

Jako nástroj pro řízení překladu zdrojového kódu (build system) byl zvolen CMake [24]. Jedná se o moderní nástroj a zároveň jednoduchý skriptovací jazyk určený primárně pro projekty využívající jazyk C++, ve kterém je také tento nástroj implementován. Samotné řízení překladu zdrojového kódu probíhá pomocí sady skriptů, které jsou součástí zdrojového kódu aplikace. Všechny tyto skripty v jazyce CMake byly během vývoje aplikace vytvořeny ručně s důrazem na přehlednost a robustnost. Před vlastním překladem zdrojového kódu nástroj CMake na základě těchto skriptů vygeneruje soubory, které jsou následně použity k řízení dalších nástrojů dostupných na dané platformě, které zajišťují samotný překlad zdrojového kódu aplikace. V případě linuxových systémů se většinou jedná o nástroj make. Vlastní překlad zdrojového kódu tedy probíhá přímo za použití nástrojů daného systému.

Vývoj aplikace probíhal na počítači s operačním systémem Debian GNU/Linux ve verzi sid s desktopovým prostředím Xfce. Téměř celý zdrojový kód aplikace byl vytvořen v textovém editoru gedit. Zbývající menší části byly vytvořeny v IDE Qt Creator, které bylo využito primárně během ladění aplikace za použití grafické nadstavby debuggeru GDB integrované v tomto IDE. Během vývoje byl zdrojový kód aplikace překládán pomocí kompilátoru GCC. Následně byl překlad zdrojového kódu testován také za použití kompilátoru Clang.

4.2 Architektura aplikace

Aplikace je řízena událostmi (event-driven programming). Klíčovou komponentou aplikace je proto třída `EventSystem`, která zajišťuje vytváření a zpracování událostí. Při startu aplikace hlavní vlákno nejprve inicializuje všechny potřebné komponenty aplikace a následně vstoupí do smyčky událostí uvnitř třídy `EventSystem`. Zde postupně zpracovává jednotlivé události čekající ve frontě. Pokud je tato fronta prázdná, tak hlavní vlákno čeká na další příchozí událost. Naopak při ukončení aplikace hlavní vlákno nejprve opustí smyčku událostí, poté provede ukončení všech aktivních komponent a následně celé aplikace.

Zpracování události představuje spuštění všech zpětných volání (callbacks) zaregistrovaných pro daný typ události. Tato zpětná volání jsou spuštěna vždy z hlavního vlákna. To výrazně zjednodušuje návrh jednotlivých komponent aplikace, protože většina z nich nemusí podporovat konkurenční přístup z více vláken. Ze stejného důvodu se také v kódu zpětných volání nesmí nacházet žádné blokující operace, protože by došlo k zablokování činnosti celé aplikace. Blokující volání je proto potřeba provádět v mimo hlavní vlákno a následně výsledek předat do hlavního vlákna pomocí událostí.

Pro tento účel je v aplikaci použito několik dalších vláken. Jsou to vlákna Signal, Resolver a Poll. Vlákno Signal čeká na signály [25] od operačního systému a následně generuje potřebné události. Vlákno Resolver zajišťuje sběr dodatečných informací o síťových adresách a portech. Poslední nejdůležitější vlákno Poll uvnitř třídy `PollSystem`

umožňuje asynchronní provádění operací, které jsou za normálních okolností blokující. Toto vlákno většinu času spí uvnitř systémového volání poll [26], kde čeká, až bude možné provést požadovanou činnost s některým z objektů zaregistrovaných v třídě `PollSystem`. Pokud je možné některou z požadovaných činností provést, tak toto vlákno rychle vytvoří událost informující o této skutečnosti. Následně je v hlavním vlákně spuštěno zpětné volání, kde je požadovaná činnost provedena bez potřeby jakéhokoliv blokování. Původně byla třída `PollSystem` určena pouze pro implementaci asynchronních síťových soketů použitých pro komunikaci mezi klientskou a serverovou částí aplikace. Při použití asynchronních soketů například server nemusí vytvářet nové vlákno pro každého připojeného klienta, protože je možné obsluhovat všechny klienty z jediného v tomto případě hlavního vlákna. Systémové zdroje jsou tedy využívány mnohem efektivněji. V průběhu vývoje se ukázalo, že pomocí třídy `PollSystem` lze elegantně řešit i další blokující operace. Například zpracování vstupu z klávesnice v komponentě uživatelského rozhraní nebo komunikaci s conntrack tabulkou uvnitř jádra operačního systému pomocí Netlink soketů. Aplikace si tedy za všech okolností vystačí jen s hlavním vláknem a zmíněnými třemi pomocnými vlákny.

4.3 Organizace zdrojového kódu

Při návrhu a vývoji jednotlivých komponent aplikace byl kladen důraz na modulárnost výsledného řešení a také na případnou možnost využití jednotlivých komponent i v dalších aplikacích. Komponenta Collector pro sběr informací o síťovém provozu je oddělena od zbytku zdrojového kódu pro usnadnění možného přidání další implementace této komponenty využívající jiné řešení pro sběr informací o síťovém provozu. Stejným způsobem je oddělena i komponenta uživatelského rozhraní. Veškerý kód závislý na dané platformě je rovněž oddělen od hlavního zdrojového kódu. To umožňuje snadné přidání podpory dalších operačních systémů. Aplikace je tedy navržena jako multiplatformní software. Navíc i přes skutečnost, že je aplikace určena pro použití pouze na operačních systémech založených na linuxovém jádře, tak během návrhu kódu závislého na platformě bylo preferováno použití funkcí definovaných ve standardu POSIX, který

implementuje velká část existujících unixových operačních systémů. To ve výsledku znamená, že většinu kódu závislého na platformě by mělo být možné bez problému přeložit a spustit i na dalších unixových operačních systémech.

Kapitola 5

Testování

Jednotlivé komponenty výsledné aplikace byly testovány průběžně během vývoje. Po dokončení vývoje bylo následně provedeno důkladné ruční testování všech funkcí aplikace. Kromě testování na poměrně výkonném počítači použitém pro vývoj byla aplikace testována také na malém počítači s procesorem Intel Celeron N3150, který slouží jako domácí síťové uložiště a zároveň i jako brána do internetu. Toto zařízení tedy zajišťuje funkci hraničního routeru pro malou domácí síť a je ideálním místem pro monitorování síťového provozu. Aplikace funguje na tomto zařízení bez jakýchkoliv problémů a navíc spotřebovává minimum systémových prostředků.

Během vývoje a při závěrečném testování byla rovněž použita sada nástrojů Valgrind [27], která obsahuje různé nástroje pro ladění a profilování aplikací. Hlavním nástrojem je memcheck, který umožňuje detekovat různé chyby při práci s pamětí. Mezi tyto chyby patří například únik paměti (memory leak) nebo přístup do nepřidělené části paměti. Během profilování aplikace byly využity nástroje callgrind a massif z této sady. Nástroj callgrind je call-graph profiler a umožňuje odhalit části programu, ve kterých se spotřebovává nejvíce procesorového času. Nástroj massif je heap profiler a slouží k zobrazení množství paměti alokované jednotlivými komponentami aplikace.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a vytvořit aplikaci pro monitorování síťového provozu. Tento cíl se podařilo splnit a výsledná aplikace je plně funkční.

Aplikace poskytuje obdobnou funkcionalitu jako stávající řešení při zachování vysoké efektivity. Navíc je zde možnost provozovat aplikaci v režimu klient-server. Hlavním cílem ale bylo vytvořit robustní základ, na kterém bude možné vystavět univerzální nástroj s velkým množstvím funkcí, který umožní snadné a efektivní monitorování síťového provozu prakticky kdekoliv. V budoucnu je tedy očekáván další vývoj aplikace.

Zdrojový kód výsledné aplikace je poměrně rozsáhlý. Během vývoje bylo značné množství úsilí a času věnováno refaktorování pro zajištění právě vysoké robustnosti a čistoty výsledného zdrojového kódu.

Literatura

[1] Wireshark. *Wireshark · Go Deep.* [online]. [cit. 2019-04-16]. Dostupné z: https://www.wireshark.org/

[2] Manpage of PCAP. *TCPDUMP/LIBPCAP public repository* [online]. [cit. 2019-04-16]. Dostupné z: https://www.tcpdump.org/manpages/pcap.3pcap.html

[3] packet(7) - Linux manual page. *Michael Kerrisk - man7.org* [online]. [cit. 2019-04-16]. Dostupné z: http://man7.org/linux/man-pages/man7/packet.7.html

[4] proc_net_tcp. *The Linux Kernel Archives* [online]. [cit. 2019-04-16]. Dostupné z: https://www.kernel.org/doc/Documentation/networking/proc_net_tcp.txt

[5] The netfilter.org project. *netfilter/iptables project homepage* [online]. [cit. 2019-04-16]. Dostupné z: https://netfilter.org/

[6] ENGELHARDT, Jan. *j.eng's site* [online]. [cit. 2019-04-16]. Dostupné z: http://inai.de/images/nf-components.svg

[7] nf_conntrack-sysctl. *The Linux Kernel Archives* [online]. [cit. 2019-04-16]. Dostupné z: https://www.kernel.org/doc/Documentation/networking/nf_conntrack-sysctl.txt

[8] netlink(7) - Linux manual page. *Michael Kerrisk - man7.org* [online]. [cit. 2019-04-16]. Dostupné z: http://man7.org/linux/man-pages/man7/netlink.7.html

[9] capabilities(7) - Linux manual page. *Michael Kerrisk - man7.org* [online]. [cit. 2019-04-16]. Dostupné z: http://man7.org/linux/man-pages/man7/capabilities.7.html

[10] The netfilter.org "libnetfilter_conntrack"project. *netfilter/iptables project homepage* [online]. [cit. 2019-04-16]. Dostupné z: https://netfilter.org/projects/libnetfilter_conntrack/index.html

[11] netfilter: conntrack: do not enable connection tracking unless needed · torvalds/linux@4d3a57f · GitHub. *The world's leading software development platform · GitHub* [online]. [cit. 2019-04-16]. Dostupné z: https://github.com/torvalds/linux/commit/4d3a57f23dec59f0a2362e63540b2d01b37afe0a

[12] netfilter: conntrack: remove l3proto abstraction · torvalds/linux@a0ae256 · GitHub. *The world's leading software development platform · GitHub* [online]. [cit. 2019-04-16]. Dostupné z: https://github.com/torvalds/linux/commit/a0ae2562c6c4b2721d9fddba63b7286c13517d9f

[13] netfilter: nf_conntrack: provide modparam to always register conntrack hooks · torvalds/linux@ba3fbe6 · GitHub. *The world's leading software development platform · GitHub* [online]. [cit. 2019-04-16]. Dostupné z: https://github.com/torvalds/linux/commit/ba3fbe663635ae7b33a2d972c5d2def036258e42

[14] getnameinfo(3) - Linux manual page. *Michael Kerrisk - man7.org* [online]. [cit. 2019-04-16]. Dostupné z: http://man7.org/linux/man-pages/man3/getnameinfo.3.html

[15] RFC 6335 - Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. *IETF Tools* [online]. [cit. 2019-04-16]. Dostupné z: https://tools.ietf.org/html/rfc6335

[16] GeoLite2 Free Downloadable Databases. *MaxMind Developer Site* [online]. [cit. 2019-04-16]. Dostupné z: https://dev.maxmind.com/geoip/geoip2/geolite2/

[17] netsniff-ng toolkit. *netsniff-ng toolkit* [online]. [cit. 2019-04-16]. Dostupné z: http://netsniff-ng.org/

[18] GeoLite Legacy Discontinuation Information. *Support Center | MaxMind* [online]. [cit. 2019-04-16]. Dostupné z: https://support.maxmind.com/geolite-legacy-discontinuation-notice/

[19] IP Tables State. *Phil's Technical Pages* [online]. [cit. 2019-04-16]. Dostupné z: https://www.phildev.net/iptstate/

[20] The netfilter.org "conntrack-tools"project. *netfilter/iptables project homepage* [online]. [cit. 2019-04-16]. Dostupné z: https://netfilter.org/projects/conntrack-tools/index.html

[21] top (software). *Wikipedia, the free encyclopedia* [online]. [cit. 2019-04-16]. Dostupné z: https://en.wikipedia.org/wiki/Top_(software)

[22] JSON. *JSON* [online]. [cit. 2019-04-16]. Dostupné z: https://json.org/

[23] htop - an interactive process viewer for Unix. *hisham.hm* [online]. [cit. 2019-04-16]. Dostupné z: https://hisham.hm/htop/

[24] CMake. *CMake* [online]. [cit. 2019-04-16]. Dostupné z: https://cmake.org/

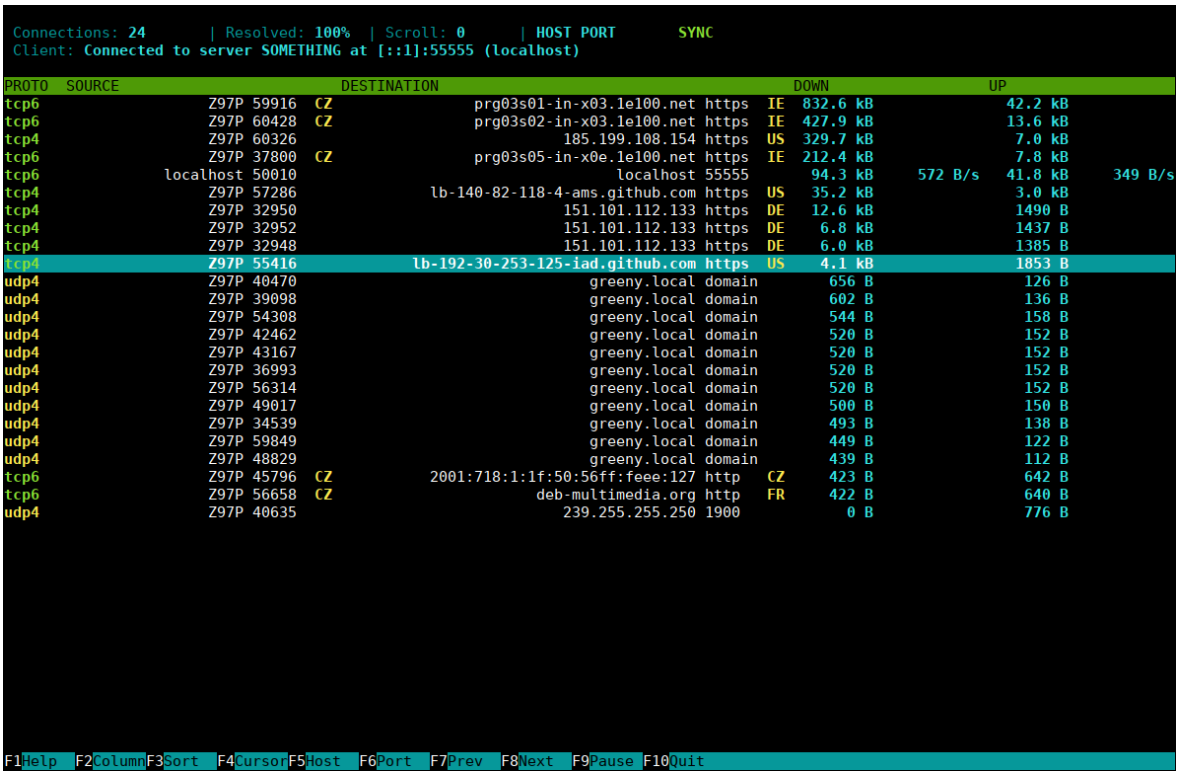
[25] signal(7) - Linux manual page. *Michael Kerrisk - man7.org* [online]. [cit. 2019-04-16]. Dostupné z: http://man7.org/linux/man-pages/man7/signal.7.html

[26] poll(7) - Linux manual page. *Michael Kerrisk - man7.org* [online]. [cit. 2019-04-16]. Dostupné z: http://man7.org/linux/man-pages/man2/poll.2.html

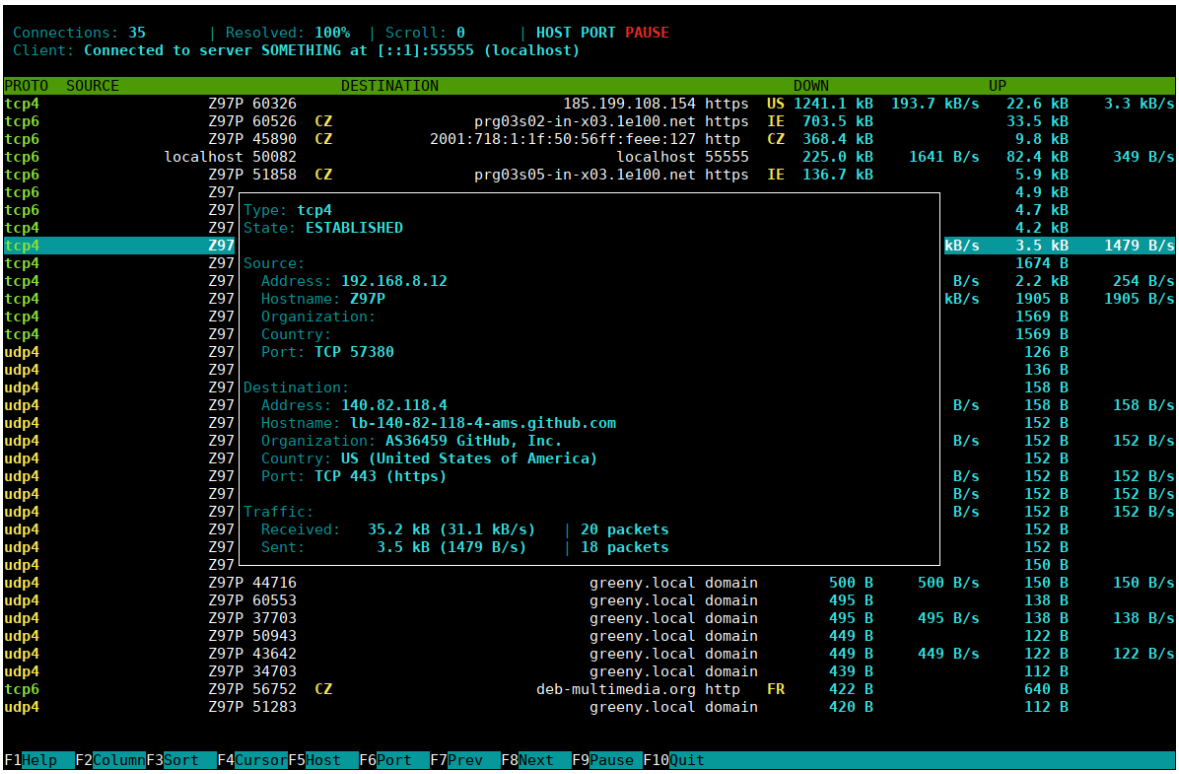
[27] Valgrind. *Valgrind Home* [online]. [cit. 2019-04-16]. Dostupné z: http://valgrind.org/

Příloha A

Ukázka výsledné aplikace



Obrázek A.1: Aplikace v režimu klient-server



Obrázek A.2: Zobrazení podrobných informací o síťovém spojení