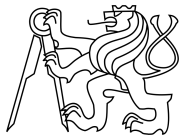
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Generátor síťového provozu na úrovni aplikačních protokolů
Student:	Bc. Jan Karafiát
Vedoucí:	Mgr. Rudolf Bohumil Blažek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Prostudujte technologie pro zachytávání, přehrávání, modelování a generování síťového provozu na různých vrstvách OSI modelu. Vytvořte knihovnu vzorků provozu datové sítě na úrovni vybraných aplikačních protokolů. Navrhněte a implementujte nástroje, které na základě vytvořené knihovny budou generovat provoz s předepsanými charakteristikami a strukturou. Uživatel si tak bude schopen připravit scénář generování síťového provozu a tento následně spustit. Nástroje musí umožňovat měnit IP adresy, MAC adresy v paketech a generovat pakety zvolených charakteristik. Hlavním účelem generátoru je testovat schopnosti nástrojů pro měření a automatickou analýzu provozu datové sítě. Otestujte vytvořený generátor s vybraným analyzátorem síťového provozu. Práce je vypisována ve spolupráci se společností INVEA-TECH.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 4. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Diplomová práce

Generátor síťového provozu na úrovni aplikačních protokolů

Bc. Jan Karafiát

Vedoucí práce: Mgr. Rudolf Bohumil Blažek, Ph.D.

10. května 2016

Poděkování

Zde bych rád poděkoval svému vedoucímu, Dr. Blažkovi, za přátelské vedení této práce, ochotu a poskytnutí cenných rad. Také děkuji své rodině za vytrvalou podporu v průběhu celého studia i při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. května 2016 .....

Odkaz na tuto práci

Karafiát, Jan. *Generátor síťového provozu na úrovni aplikačních protokolů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato práce se zaměřuje na modelování a generování síťového provozu na aplikační úrovni. Zvolený aplikační protokol je HTTP, jehož vybraný behaviorální model je rozšířen o rozlišování MIME typů vestavěných objektů webové stránky. V rámci práce byl vytvořen a otestován generátor HTTP provozu, který umožňuje paralelní obsluhu více klientů a může pracovat ve třech různých režimech: přeposílání dříve zachycených paketů, posílání souborů získaných z pcap souboru a stochastické generování. Testováním nástroje bylo ověřeno spolehlivé generování provozu dle zadaných parametrů modelu. Generovaný provoz byl rozeznán monitorovacími nástroji jako HTTP provoz.

Klíčová slova Webový provoz, behaviorální model webového provozu, generování síťového provozu, generování provozu na aplikační úrovni, analýza pcap souboru, generátor webového provozu.

Abstract

This thesis deals with modeling and generating network traffic at the application level. A selected behavioral model for HTTP traffic is extended with specification of MIME types for objects embedded in web pages. A generator of HTTP traffic has been developed and tested that can service several clients in parallel. It supports generating traffic in three different modes: re-transmitting previously captured packets, resending files extracted from a pcap file, and

stochastic generating. Testing of the developed program confirmed that the generated traffic reliably conforms to the specified model and its parameters. The traffic was recognized by monitoring software as HTTP traffic.

Keywords Web traffic, behavioral web traffic model, network traffic generator, generating traffic in the application layer, pcap file analysis, web traffic generator.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Každodenní internetová komunikace se stala nedílnou součástí dorozumívání dnešní společnosti. S rostoucím počtem připojených lidí a koncových komunikujících zařízení roste důležitost funkčnosti celého tohoto mechanismu. Současně rostou také nároky na efektivitu využívání síťové infrastruktury a na zabezpečení komunikace. Pro uspokojení těchto požadavků vznikají stále nové technologie či protokoly, např. transportní protokol QUIC [1] navržený pro využití šifrovaným webovým provozem. Tím se celý systém stává stále komplexnější a jakýkoli problém mívá významnější dopad, než tomu bylo dříve.

S rozvojem technologií roste také množství útoků, čímž se stává náročnější ochrana přenášených informací. Pro ochranu před síťovými útoky se využívají různé detekční či prevenční mechanismy, např. firewally, systémy detekce průniku (IDS) či systémy prevence průniku (IPS), které fungují na mnoha různých principech, ať už se jedná o využití připravených signatur či detekci statistických anomálií [2]. Jejich správné fungování je zásadní, čímž rostou také nároky na jejich testování. Jako jedna z metod testování těchto zařízení může být využit i synteticky generovaný provoz [3].

Generování syntetického síťového provozu nachází své využití v mnoha oblastech výzkumu, ať už při testování bezpečnosti, např. simulováním provozu ze stanice infikované virem [4] či simulací DDoS útoku [5], kvality poskytovaných služeb, nových protokolů či výkonnosti sítí a síťových zařízení zmíněných výše. Pro lepší pochopení síťového provozu je vhodné jeho významné charakteristiky popsat pomocí modelu. Následné využití modelu při generování syntetického provozu vede ke snadnějšímu srovnání jednotlivých experimentů [6]. Dle cíle experimentu může být síťový provoz modelován pouze na jedné nebo zároveň na několika různých vrstvách OSI modelu.

Problematikou modelování a generování síťového provozu se zabývám v této práci. První kapitola je věnována analýze a shrnutí metod pro modelování a generování síťového provozu. Ve druhé kapitole se zabývám návrhem a zpřesněním vybraného statistického behaviorálního modelu [7], který je ve třetí kapitole využit k implementaci generátoru HTTP provozu. HTTP provoz byl

vybrán jako protokol, který si i přes klesající tendenci stále drží významný podíl zastoupení v síťovém provozu. Poslední kapitola byla věnována testování vytvořeného generátoru. Praktickým cílem této práce je využití vzniklého generátoru jako součásti řešení pro testování síťových zařízení detekujících anomálie a možné hrozby v síťovém provozu.

Kapitola 1

Analýza

1.1 Modelování síťového provozu

Pro simulace změn zatížení infrastruktury jednotlivými protokoly či dopadu navrhovaných úprav v síťové infrastruktuře je důležité mít provoz formálně popsán. Pro tento popis se využívají matematické modely, které slouží jako podklad pro simulaci síťového provozu s využitím síťových generátorů [7]. V této sekci shrnuji dostupné modely, princip jejich tvorby a užití.

Dle [7] je cílem modelování vytvoření vzorce síťového provozu, tzv. modelu, který je generován např. reálnými uživateli v reálné síti podporující stejný počet uživatelů. Výsledný model by měl umožnit návrh parametrů sítě a přesné stanovení parametrů výkonu, podle kterých bude možné generovat syntetický síťový provoz takový, že na dané úrovni rozlišení bude vykazovat stejné charakteristiky jako provoz reálný.

1.1.1 Metody modelování

Pro popis povahy síťového provozu se využívají analytické nebo empirické modely, přičemž volba typu modelu bývá problematická [8].

Analytický model náhodné veličiny je matematický popis jejího pravděpodobnostního rozdělení. Jeho tvorba vychází z odhadů parametrů zvoleného rozdělení tak, aby toto rozdělení co nejlépe odpovídalo vzorovým datům. Rozdělení mívají pouze několik parametrů, díky čemuž jsou výsledné modely snadno interpretovatelné. Jejich velkou výhodou je také možnost snadného porovnání různých datových sad. Problémem naopak je odhadnutí statistického rozdělení a jeho parametrů tak, aby vzorová data rozdělení odpovídala co nejlépe. Statistický přístup bývá pro modelování síťového provozu navíc často využit zjednodušeně [8], což může znamenat např. využití pouze prvních a druhých momentů, odhadování rozdělení tzv. od oka či naprosté ignorování odlehlých hodnot.

Na rozdíl od analytického modelování, empirický přístup je založen pouze na vzorcích reálného provozu a je snadno implementovatelný. Velmi dobře popisuje datovou sadu, ze které byl odvozen, což se analytickému modelu ne vždy musí podařit. Příkladem empirického modelu je Tcplib [9].

Pro modelování síťového provozu se může využít abstrakce k popsání statistických vlastností síťového provozu nezávisle na jeho konkrétních vlastnostech a závislostech. Příkladem může být modelování časování HTTP provozu bez úvahy fungování TCP protokolu, zejména velikosti jeho okénka, která rychlost přenosů HTTP objektů ovlivňuje. Tyto modely se nazývají behaviorální [10]. Závislost HTTP provozu na protokolech transportní vrstvy OSI modelu diskutuji v sekci 1.4.

1.1.2 Modelování provozu na různých úrovních

Modely síťového provozu lze vytvářet na úrovni paketů, toků, aplikací apod. Tyto přístupy se liší modelovanými parametry skutečného provozu [3].

Pro vytvoření modelu síťového provozu na úrovni paketů je možné jednoduše modelovat např. velikosti paketů, čas mezi odchody po sobě jdoucích paketů nebo procentuální zastoupení jednotlivých IP adres. Při využití hluboké analýzy paketů mohou být modelovány i detailnější charakteristiky, jako např. entropie znaků v datovém obsahu, tj. payloadu, paketu.

Při modelování síťového provozu na úrovni aplikací je provoz rozlišován podle komunikačního chování jednotlivých aplikací [11]. Aplikace mohou být děleny podle počtu účastníků komunikace, počtu komunikačních kanálů mezi jednotlivými účastníky komunikace či podle mechanismu výběru portů, na kterých komunikace probíhá. V tomto případě záleží na perspektivě, ze které je daná topologie pozorována. Příkladem může být běžný webový provoz, kde z pohledu klienta převažuje příchozí komunikace obvykle z několika málo destinací, zatímco z pohledu serveru převažuje odchozí komunikace na mnoho destinací. Od této perspektivy je vhodné a obvyklé [11] se oprostit vnějším pohledem na celou síť a rozlišit různé topologie na základě počtu vzájemně komunikujících uzlů, čímž lze dostat dvě základní metody komunikace, a to klient-server a peer-to-peer. Model provozu pak odpovídá zvolené topologii a obsahuje výše uvedené parametry, kterými mohou být rozdělení náhodné volby portů či vyváženost komunikace v určitých směrech topologie sítě [11].

Další možností je generování síťového provozu na úrovni síťových toků dle IPFIX standardu [12] či Cisco formátu NetFlow [13]. V tomto případě se nezohledňuje datový obsah jednotlivých přenášených paketů, ale modeluje se zejména doba trvání spojení, celkový objem přenesených dat a různé agregované atributy přenesených paketů, zejména z hlaviček použitých protokolů. Možnosti modelování síťového provozu pomocí toků vytvářených dle IPFIX standardu detailněji diskutuji v sekci 1.2.3.

1.1.3 Modelování aplikačních protokolů

Struktura složení internetového provozu se neustále a rychle vyvíjí. V roce 2001 bylo provedeno měření [14], ze kterého vyplynula převaha zastoupení transportního protokolu TCP vůči UDP. Nicméně s rostoucí oblibou streamování videa a VoIP telefonie podíl provozu přenesený UDP protokolem vzrostl v porovnání s dříve provedenými studiemi [15]. Přestože byl výzkum proveden před patnácti lety, měření přineslo několik dalších zajímavých poznatků:

- Internetové spojení je obousměrné, nicméně z hlediska objemu přenesených dat obvykle asymetrické. Domnívám se, že tento fakt lze i dnes pozorovat u většiny komunikací klienta se serverem využívajících transportní protokol TCP, kde data převážně proudí jedním směrem, zatímco potvrzení o přijetí (TCP ACK) směrem druhým. Naproti tomu u peer-to-peer komunikací nejspíše bude objem přenesených dat podobný v obou směrech.

- Většina TCP spojení má malý objem přenášených dat a trvá krátkou dobu. Tento poznatek se nejspíše bude lišit na základě konkrétní aplikace, protože např. perzistentní spojení dnes využívané v HTTP provozu funguje na opačném principu.

- Statistický proces popisující příchody paketů není poissonovský, protože pakety mají tendenci se shlukovat.

- Statistický proces popisující vznik relací, tj. inicializaci internetového spojení od klienta k serveru, je poissonovský, protože lidé se obvykle rozhodují nezávisle.

- Velikosti paketů jsou bimodální, kde 50% paketů má maximální možnou velikost, která je omezena pomocí MTU (maximální přenosová jednotka), a 40% paketů je velmi malých. Domnívám se, že toto zjištění přirozeně vyplývá z použití TCP protokolů, kdy jedním směrem proudí velké množství dat rozdělené do co nejmenšího počtu paketů, zatímco druhým směrem chodí převážně krátká potvrzení.

- Provoz paketů je nerovnoměrně rozdělený, kde 10% hostitelských stanic je zodpovědných za 90% veškerého provozu.

Cisco studie [16] z roku 2015 potvrdila rostoucí zastoupení přenosu videa, předpovězené výzkumem [14] z předchozího odstavce. V roce 2014 přenos videa tvořil 35% objemu veškerého internetového provozu iniciovaného uživateli. Video streaming byl z pohledu objemu dat následovaný hraním online her, sdílením souborů a webovým provozem. Zveřejněný odhad vývoje do roku 2019 předpovídá stále rostoucí převahu videa, přičemž poroste i podíl online hraní a webového provozu, zatímco přenos souborů bude stagnovat. Nelze se proto divit, že většina současných studií prezentovaných dále v textu se zaměřuje na

zkoumání a klasifikaci video přenosů, a že vznikají také studie na síťový provoz odpovídající hraní online her. Příkladem takové studie je [17], která vznikla za účelem otestování schopnosti sítě přenášet herní data v reálném čase. Zde nebyly modelovány jednotlivé pakety, ale aplikační protokolové datové jednotky (tzv. APDU) a časy mezi jejich příchody. Analýzou chování uživatelů a tvorbou modelu sítě na doručování video obsahu se zabývala studie [18], kde se výsledný model skládal z unikátního identifikátoru videa, klientské IP adresy, rozdělení časů příchodu požadavků a velikosti obsahu.

Síťový provoz mezi klientem a serverem lze na aplikační úrovni modelovat dvěma způsoby z hlediska počtu klientů [19]:

1. klientský model – individuální modelování provozu generovaného hostitelskou stanicí (klientem),

2. agregační model – modelování agregovaného provozu generovaného skupinou stanic.

Klientský model je schopen zachytit více detailů na aplikační úrovni, nicméně vyšší úroveň detailů implikuje komplexnější model kvůli potřebě více parametrů pro jeho pochopení a konfiguraci. Vyšší úroveň detailů modelu navíc nemusí vždy vést k lepšímu popisu, protože na určitých parametrech nezáleží. Příkladem budiž vliv rozvrhových mechanismů či chování front při detekci a modelování sítí brzdících provoz, kde jsou jednodušší modely lepší volbou [19]. Agregační model je hrubší aproximací reálného provozu, zpravidla však spotřebovává méně výpočetních zdrojů než model klientský. Tyto modely své využití najdou např. při simulacích zátěže serverů, kdy nezáleží na chování jednotlivých klientů, ale pouze na celkovém objemu generovaného provozu.

Aplikační vrstva může být modelována z pohledu chování uživatele nebo fungování aplikace [20]. Přístupy se liší parametry modelu vedoucími ke správnému popsání provozu a chování a vlastností např. webové stránky. Z hlediska uživatele mohou být významnými parametry např. frekvence jeho klikání, rychlost nalezení požadované informace, doba strávená na dané webové stránce či míra okamžitého opuštění stránky. Naproti tomu z hlediska fungování aplikace mohou být důležité např. frekvence výskytu jednotlivých znaků na stránce a entropie stránky, počty objektů vložených ve stránce, určitá specifická formulářová pole či informace obsažené v hlavičkách [7]. Často se oba přístupy kombinují, což konkrétně pro webový provoz popisuji v sekci 1.1.4.

1.1.4 Modelování webového provozu

Pro tuto práci jsem vybral aplikační protokol HTTP používaný během nešifrovaného procházení webu, které podle informací uvedených v odstavci 1.1.3 patří k nejvýznamnějším datovým přenosům probíhajícím na internetu.

Modelování webového provozu lze rozdělit pomocí dvou kritérií: dle počtu zároveň modelovaných klientů (analogicky ke způsobu popsanému v sekci 1.1.3) a podle způsobu rozlišení jednotlivých relací (viz sekce 1.1.4.1).

Bez ohledu na to, zda je model tvořen z pohledu chování uživatele nebo fungování aplikace, anebo pomocí jejich kombinace, existují při modelování aplikační vrstvy zejména dva problémy [21]:

1. Jednotlivé systémové komponenty spolu interagují, servery a webové prohlížeče různých výrobců se chovají odlišným způsobem. Aplikační protokoly (např. HTTP) se kontinuálně rozvíjí a odlišné verze se používají zároveň. I samotná implementace transportních protokolů (např. TCP) se může na různých operačních systémech odlišovat.

2. Vzorce chování uživatelů při používání aplikací se v čase mění. Např. procházení webu se stává více komplexní, používá se, ať už záměrně či náhodně, více webových prohlížečů ve stejný časový okamžik. Navíc lze načítaní webové stránky přerušit využitím tlačítka ve webovém prohlížeči nebo odchodem na jinou stránku.

1.1.4.1 Rozdělení podle odlišení relací

Modely webového provozu mohou být odlišeny na základě toho, zda jsou vytvořeny na základě požadavku na stránku (tzv. page-based) nebo na základě TCP spojení (tzv. connection-based) [22]. Generátory síťového provozu založené na page-based modelech jsou poměrně komplikované, každá náhodná veličina je považována za nezávislou a potenciálně důležité závislosti mezi veličinami nejsou explicitně modelovány. Z toho důvodu autoři prezentují model webového provozu založený na zkoumání TCP spojení. Výsledný model obsahuje jak vlastnosti transportní vrstvy, tak vlastnosti aplikační vrstvy. Modelována je jednak míra navazování spojení, a jednak velikost a časování požadavků a odpovědí v rámci spojení, a to včetně perzistentních. Perzistentní spojení spočívá v opětovném využití navázaného TCP spojení více HTTP spojeními. Tento mechanismus diskutuji v sekci 1.4.1.1.

1.1.4.2 Parametry modelu HTTP provozu

V této části na příkladech vybraných modelů ilustruji postup tvorby modelů a spolu s tím jak obvyklý výběr modelovaných náhodných veličin, tak nalezené odlišnosti mezi nimi. Veškeré termíny označující části HTTP provozu jsou užity v souladu s jeho standardem [23].

Mah a kolektiv [24] vytvořili v roce 1997 empirický model HTTP provozu zachycujícího logicky významné parametry webového provozu. Modelované parametry provozu byly délka požadavku i odpovědi, počet vestavěných objektů, čas mezi stažením po sobě jdoucích objektů, počet po sobě jdoucích dokumentů stažených z jednoho serveru a pravděpodobnost návštěvy serveru

ve smyslu jeho popularity. Statistická rozdělení jednotlivých náhodných veličin, kromě výběru serveru, byla odvozena z jejich empirických distribučních funkcí. Hlavní přenesený objekt a v něm vestavěné objekty (obrázky, css, atd.) v následujícím textu souhrnně označuji jako *HTTP objekty.*

V roce 2005 bylo provedeno měření a analýza HTTP provozu [25]. Pomocí stochastických procesů byly modelovány HTTP relace a s nimi související časování, z jehož hlediska byl HTTP provoz rozdělen do tří částí:

1. ON čas – doba trvání přenosu hlavního HTML objektu spolu se všemi souvisejícími vestavěnými objekty z webového serveru ke klientovi. Za vestavěné objekty v HTML stránce jsou považovány všechny objekty, které slouží buď ke správnému zobrazení a fungování stránky, např. CSS a javascriptové objekty, nebo se zobrazují uživateli v rámci stránky, např. různé obrázkové soubory.

2. Pasivní OFF čas – doba mezi koncem jedné relace a začátkem relace následující. Délka tohoto časového intervalu odpovídá době prohlížení stažené stránky uživatelem.

3. Aktivní OFF čas – doba mezi úspěšnými příchody jednotlivých vestavěných objektů v rámci stejného ON časového intervalu.

Zároveň jsou zde naznačeny dvě možnosti nahlížení na vlastnosti samotného HTTP provozu. První z nich je vlastnost obsahu, kdy webové stránky obsahují vestavěné objekty jako jsou obrázky, zvuky či videa spíše než samotný text či naopak. Druhou je vlastnost struktury webového serveru, kdy např. novinkový server obsahuje mnoho videí a obrázků, zatímco weby výzkumníků obsahují méně objektů s velkými reporty, prezentacemi apod. Z provedeného měření byly odvozeny následující významné parametry HTTP provozu:

- Doba mezi jednotlivými relacemi (pasivní OFF čas), jedná se o čas prohlížení stránky uživatelem. Výsledkem měření bylo zjištění, že příchody na stránky lze modelovat Poissonovým procesem, zatímco čas mezi jednotlivými relaci odpovídá nejlépe exponenciálnímu procesu.

- Aktivní OFF čas je silně ovlivněn webovým prohlížečem. Klasicky jsou požadavky spuštěny sekvenčně, nicméně v případě HTTP/1.1 s pipeliningem (popsáno v sekci 1.4) může klient paralelně otevřít několik HTTP spojení se serverem v rámci jendoho TCP spojení a v takovém případě je tento aktivní OFF čas velmi krátký.

- Velikost HTTP požadavku je obvykle 450B, ale může být větší v okamžiku, kdy uživatel odesílá autentizační informace nebo online formulář.

- Míra úspěšnosti dotazu, kde je za úspěch považován návratový kód 200 OK, byla 88%. Odesílané dotazy na objekty uložené v cachi zde nejsou považovány za úspěšné.

- Počet transakcí, za který může být přibližně považován počet vestavěných objektů ve stránce. Tento parametr lze využít ke zjištění, jak moc přenos hlavního HTML objektu a všech jeho vestavěných objektů zatěžuje webový server.

V roce 2007 byl vyvinut model pro chování uživatele během procházení webu [26]. Podkladem pro tento model byla analýza logů z proxy serverů. Model byl podobně jako předchozí vystavěn na tzv. ON/OFF modelu, kde ON stav odpovídá požadavku na objekt a jeho stažení a OFF stav přísluší neaktivnímu času, tedy čtení stránky uživatelem. Spojení bylo definováno na základě webového požadavku, díky čemuž bylo reflektováno použití více prohlížečů ve stejném okamžiku a uživatelská interakce. Z logů byly anonymizovány URL (Uniform Resource Locator) a referer záznamy, ponechány byly pouze přípony požadovaných souborů z důvodu analýzy složení provozu, pro kterou byla využita standardní statistická rozdělení (lognormální, exponenciální, atd.). Zaznamenáván byl také čas přístupu na stránku a doba do vybavení posledního bajtu odpovědi. Výsledné parametry provozu zohledněné v modelu provozu byly celková velikost HTML objektu, velikost a počet vestavěných objektů, doba parsování, doba mezi přijetím po sobě jdoucích vestavěných objektů, doba čtení a velikost požadavku. V této studii byl také zjištěn rostoucí podíl online streamingu videa v síťovém provozu, což je v souladu s dříve v textu uvedenými studiemi, a kratší doby mezi příchody jednotlivých odpovědí způsobené pravděpodobně užitím webových cachí nebo vyšší dostupností vysokorychlostních sítí.

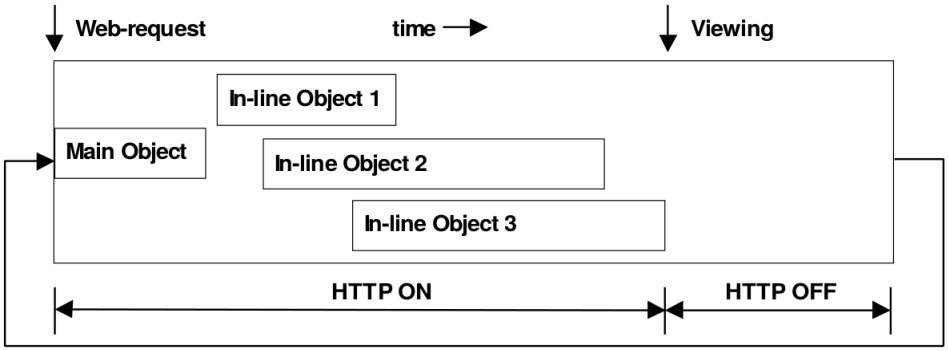
Pro návrh a budoucí implementaci generátoru vycházím z behaviorálního modelu HTTP provozu uvedeného v [7]. Tento model předpokládá postup stažení webové stránky ilustrovaný na obrázku 1.1. Nejdříve je stažen celý hlavní (HTML) objekt, který je zpracován a následně jsou odesílány HTTP požadavky na jednotlivé vestavěné objekty odkazované z hlavního objektu. Tyto objekty jsou přenášeny paralelně. Po ukončení přenosu posledního běžícího stahování vestavěného objektu se začíná měřit doba prohlížení stažené stránky uživatelem.

Modelované náhodné veličiny jsou spolu s jejich statistickými vlastnostmi (střední hodnota, medián, směrodatná odchylka a nejlépe se hodící statistické rozdělení) uvedeny v tabulce 1.1. Rozhodl jsem se zaměřit pouze na přenos necachovaných dat, a proto z původní tabulky vynechávám statistiky rozdělení cachovaných a necachovaných objektů.

Z uvedené tabulky 1.1 stojí za povšimnutí především hodnoty rozdělení pro velikosti vestavěných objektů. Zde je jednak medián více než třikrát menší proti střední hodnotě a hlavně směrodatná odchylka, a potažmo rozptyl, jsou výrazně vyšší, např. v porovnání s poměry hodnot těchto momentů pro velikost hlavních objektů. Z mého pohledu jsou tyto výsledky způsobeny sdružováním různých typů vestavěných objektů do jedné společné kategorie. Tato kategorie tak zahrnuje několikařádkové javascriptové soubory nebo malé obrázkové

Tabulka 1.1: Výchozí HTTP model. (LN=Lognormální, G=Gamma, W=Weibullovo). Převzato z [7].

Náhodná veličina	E	Medián	S.D.	Rozdělení
Velikost požadavku (v B)	360.4	344	106.5	LN
Velikost hlavního objektu (v B)	10710	6094	25032	LN
Velikost vestavěného objektu (v B)	7758	1931	126168	LN
Čas parsování (v s)	0.13	0.06	0.187	G
Počet vestavěných objektů	5.55	2	11.4	G
IAT vestavěných objektů (v s)	0.86	0.17	2.15	G
Čas prohlížení (off time, v s)	39.5	11.7	92.6	W



Obrázek 1.1: Mechanismus stažení webové stránky. Převzato z [7].

soubory, sloužící např. pro vykreslení kulatých rohů v ohraničení jednotlivých částí HTML stránek, ale zároveň také velké fotky.

Z toho důvodu jsem se rozhodl model zpřesnit rozdělením vestavěných objektů do čtyř samostatných kategorií – css objekty, javascriptové objekty, obrázky a ostatní vestavěné objekty. Ačkoliv pro skupinu obrázků nejspíše bude možné pozorovat stejný jev jako ve výše popsaném modelu, předpokládám, že zbylé kategorie se budou lišit. Konkrétní hodnoty parametrů modelu závisí na datech, ze kterých je model tvořen, což rozebírám v sekci 2.1.3.

Ze studií popsaných v této sekci lze vidět, že obvykle je interakce mezi klientem a serverem rozdělena na aktivní a neaktivní období. Z hlediska přenášených objektů je modelován čas mezi příchody jednotlivých objektů a jejich velikost. Objekty jsou obvykle rozděleny na hlavní a několik v něm vložených. Druh vestavěných objektů však v modelech rozlišován není, v čemž by šly tyto modely zpřesnit. Možnosti analýzy vestavěných objektů vedoucí k tomuto zpřesnění modelu diskutuji v následující sekci.

1.1.4.3 Možnosti analýzy objektů a zpřesnění vybraného modelu

Jak jsem navrhl v předchozí části, model webového provozu vybraný pro implementaci lze zpřesnit určením jednotlivých druhů objektů vestavěných v hlavním přenášeném objektu. Pro tento účel existuje mnoho přístupů, přičemž mnohdy pracují na naprosto odlišném principu a využití každého přístupu závisí zejména na dostupných datech.

Jedním z postupů, které uvažuji pro budoucí vývoj nástrojů, je klasifikace těchto objektů pomocí vytěžovacích metod [27]. Tento návrh z roku 2012 vychází z předpokladu, že webový prohlížeč slouží k podstatně více účelům než v minulosti. Dříve bylo jeho hlavním využitím prohlížení webových stránek, zatímco dnes se k tomu přidává stahování souborů a hlavně streamování audia a videa. Webový provoz může být klasifikován na základě položky Content-Type v hlavičce HTTP odpovědi. Nicméně streamování multimédií bývá identifikováno typem *application/x-mms-framed*, z čehož nelze rozlišit audio od videa. Multimédia jsou často přehrávána pomocí zásuvných modulů, které využívají namísto HTTP jiný protokol, např. Real-Time Messaging Protocol (RTMP) [28]. Konkrétní obsah tak může být identifikován pomocí jména zásuvného modulu a portu (pro RTMP konkrétně port 1935). Autoři pomocí algoritmu popsaného ve článku klasifikovali veškerý webový provoz na základě MIME typu do pěti tříd: audio (odpovídající signatuře např. audio/aac), soubor (např. application/binary), multimédia (např. application/ogg), video (např. video/mp4) a web (např. application/gif). Analyzované velikosti payloadu objektů v jednotlivých třídách se lišily, čímž se rozdělení objektů do těchto tříd ukázalo být vhodně zvolené.

Výše uvedený způsob, založený na rozlišování hodnot položky Content-Type, je pravděpodobně nejjednodušší. Jako další možný způsob rozšíření vidím využití entropie jednotlivých souborů, případně poměry zastoupení jednotlivých znaků v nich. Texty psané v různých jazycích se obvykle liší svou entropií [29]. Stejný předpoklad by mohlo jít využít i v tomto případě. Různé druhy webových stránek jsou na základě entropie rozlišovány [30]. Domnívám se, že jednotlivé HTTP objekty se entropií mezi sebou budou rovněž lišit, například HTML soubory budou na rozdíl od běžného jazyka mít poměrně velké zastoupení znaků „<“ a „>“, v css a javascriptových souborech by mohly stejně vyčnívat znaky „{“ a „}“. Poslední možností rozšíření modelů je pak vytvoření signatur [31], které jsou následně v souborech vyhledávány a soubory pomocí výsledků těchto kontrol klasifikovány do tříd.

Pro první fázi návrhu a implementace uvažuji metodu rozlišení objektů založenou na hodnotách položky Content-Type hlavičky HTTP odpovědi. Konkrétní návrh využití této metody je popsán v sekci 2.1.1.

1.2 Metody získávání dat

Pro zkoumání charakteristik síťového provozu a tvorbu jeho modelu je nutné analyzovat reálný provoz na dané síti. Analýza se obvykle provádí z nasbíraných vzorků provozu za daný časový úsek. V této sekci prezentuji možnosti sběru takových dat.

Data o provozu a jeho charakteristikách v síti lze získat několika způsoby, které závisí zejména na citlivosti rozlišení, s jakou chceme model vytvářet a na úrovni, na které provoz charakterizujeme. Využít lze logy na jednotlivých síťových zařízeních nebo odposlech (zachytávání, sniffing) provozu na úrovni paketů [24]. Data o webovém provozu lze získat také instalací zásuvného modulu přímo do prohlížeče [32]. Jednotlivým možnostem se detailněji věnuji v následujících podsekcích, navíc zde analyzuji možnost využití síťových toků definovaných standardem IPFIX, který je inspirován a nahrazuje protokol Cisco NetFlow [13].

1.2.1 Sběr dat z logů

Většina síťových zařízení umožňuje uchovávání záznamů o komunikaci, která přes ně prochází, tzv. logy. Tyto logy následně slouží k lepšímu nastavení jednotlivých zařízení, např. z hlediska bezpečnosti či lepšího vyvažování zátěže, nicméně mohou posloužit i jako podklad pro tvorbu modelu provozu dané sítě. Volba zdroje logů závisí na cíli modelování. Pro modelování požadavků na vybraný server se hodí serverové logy. Tyto logy zachytí dotazy všech uživatelů na danou webovou stránku či jiný zdroj, ale neposlouží k zaznamenání aktivity uživatele napříč více stránkami. Naproti tomu klientské logy se pro zachycení procházení uživatele napříč stránkami hodí. Pro analýzu provozu na aplikační úrovni lze také využít sběr logů z (reverse) proxy zařízení, které sice zachycují požadavky až v případě výpadku cache daného uživatele, jejich výhodou ale je zachycování provozu více různých uživatelů ve stejný čas a zároveň na více dotazovaných serverů.

1.2.2 Sběr dat odposlechem a soubory formátu pcap

Odposlech provozu na síťovém médiu (např. datový kabel, optický kabel či vzduch) nebo místě síťové infrastruktury je vhodný v případě, že potřebujeme pracovat přímo s jednotlivými rámci komunikace. Na síťovém médiu lze úspěšně zachytit jak provoz patřící nám, tak provoz mezi jinými dvěma stanicemi. Pro tento druh sběru dat slouží tzv. packet sniffer, tedy softwarové nebo hardwarové zařízení monitorující veškerý síťový provoz [33]. Proces odposlechu paketů se nazývá sniffing a má mnoho využití [34], mezi něž patří logování síťového provozu, podpora při řešení komunikačních problémů, detekce průniků do sítě nebo forenzní analýza bezpečnostních událostí. Metody

odposlechu lze rozlišit např. na IP-based a MAC-based, podle toho, zda je provoz zachycován na základě IP nebo MAC adresy.

Techniky odposlechu se liší podle druhu odposlouchávané sítě, která může být propojena pomoci rozbočovačů, kde pakety jsou broadcastované všem připojeným zařízením nebo switchovaná. Při běžném používání síťové karty je příchozí provoz směřující na jinou adresu zahozen. Na broadcastových sítích stačí uvést síťovou kartu do promiskuitního módu, ve kterém je zachycován veškerý dostupný provoz. Switche mohou umožňovat tzv. zrcadlení portů, což je funkce zajišťující kopírování provozu probíhajícího na určitém portu na jiný port switche. Pro zachycení komunikace probíhající na datovém kabelu lze využít tzv. network tap [35]. Network tap je zařízení se třemi porty, přičemž dva jsou určené pro připojení komunikujících zařízení a na třetí port je jejich komunikace kopírována. Na tomto portu je připojeno zařízení ukládající kopírovanou komunikaci. Network tap může obsahovat ještě další port určený ke správě tohoto zařízení.

Pro sniffing existuje mnoho softwarových nástrojů s většími i menšími možnostmi. Mezi nejznámější patří tcpdump [36] a Wireshark, původně vznikající pod názvem Ethereal [37].

1.2.2.1 Tcpdump

Tcpdump [36] vznikl v 90-tých letech jako UNIXový opensource nástroj ke sběru dat ze síťového rozhraní počítače a jejich následnému zpracování vybraným softwarem či zobrazení v uživatelsky přívětivé formě. Využívá knihovnu libpcap [38] a slouží jako základ většiny dalších obdobných nástrojů včetně např. Wiresharku. Umožňuje omezovat objem zachytávaného provozu pomocí filtrovacích výrazů pro selekci paketů dle zdrojových a cílových IP adres či portů, typu protokolu, apod. [36]. Pro OS Windows je dostupný jako WinDump [39]. Existuje k němu také mnoho rozšíření, např. mmdump [40] rozšiřující ho o parsovací moduly pro multimediální protokoly RTSP a H.323.

Tcpdump ve výchozím nastavení zobrazuje přehledně obsah hlavičky protokolu na řádku pro každý zachycený rámec. Standardně je zobrazena časová známka a jedná-li se o IP datagram, pak je zobrazena zdrojová i cílová adresa rámce, včetně zdrojového i cílového portu. Následně je zobrazen TCP flag, jde-li o TCP protokol, spolu s dalšími příslušnými informacemi, jako např. aktuální velikost okénka TCP protokolu.

Díky využívání libpcap knihovny je možné zachycený provoz ukládat do souborů formátu pcap pro pozdější analýzu. Tento formát byl vytvořen právě pro ukládání zachycených paketů a je pro tento účel nejčastěji používán různými nástroji na odlišných platformách.

1.2.2.2 Wireshark

Wireshark [41] vznikl jako opensource projekt v roce 1997 pod názvem Ethereal a následně byl rozvíjen mnoha přispěvateli. Je dostupný zdarma pro velkou většinu platforem, např. Windows, Linux, OS X, Solaris či FreeBSD. Díky grafickému rozhraní pro manipulaci a analýzu paketů je velmi uživatelsky přívětivý, nicméně je dostupný i pouze pro příkazovou řádku pod názvem TShark. Umožňuje detailní analýzu stovek protokolů, zachycování aktuálního provozu i offline analýzu vzorků, analýzu VoIP paketů či dešifrování mnoha protokolů, jako např. IPsec, ISAKMP či WEP a WPA/WPA2. Podporuje mnoho formátů souborů pro čtení i zápis, např. XML, CSV či pcap.

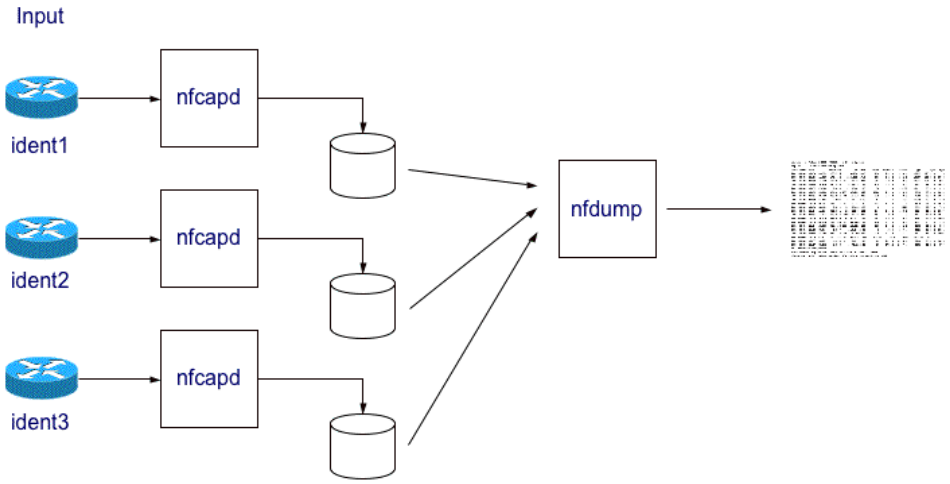
Výhodou Wiresharku oproti Tcpdumpu je větší množství filtrovacích výrazů a vyšší počet podporovaných protokolů [42].

1.2.3 Získávání dat ze síťových toků

Na síťový provoz může být nahlíženo jako na směsi toků procházejících jednotlivými síťovými zařízeními. Minimální údaje pro identifikaci toku jsou zdrojové a cílové IP adresy a porty a transportní protokol. Síťové toky poskytují informace např. o době trvání popisované komunikace, o objemu přenesených paketů či dat a také o počtech jednotlivých návěstí TCP protokolu (SYN, ACK, apod.). Toky se dají zpětně rekonstruovat z paketů, nicméně ukládání paketů pro tento účel je kapacitně náročné a zbytečné. Ve velkých sítích často z mnoha důvodů postačí místo odposlouchávání všech paketů mít informace pouze o těchto tocích, hodí se např. pro detekci bezpečnostních problémů či monitorování využití síťové infrastruktury. Aby se tyto informace daly jednoduše získat, je vhodné mít standardizované prostředky pro tuto práci napříč celým spektrem síťových technologií. Tímto standardem se v roce 2013 stal protokol IPFIX [12].

IPFIX (IP Flow Information Export) je protokol definující přístup k informacím o IP toku, jehož základem se stal proprietární NetFlow [13] protokol od společnosti Cisco. IPFIX standardizuje formát IP toku a mechanismus přenosu jeho záznamu ze síťového zařízení do kolektoru, ve kterém jsou jednotlivé údaje uchovány.

Pozorované pakety v určitém časovém intervalu jsou do toků vybírány na základě vyhodnocení svých atributů. Pakety se stejnými atributy jsou přiřazeny stejnému toku. Atributy paketů pro rozlišení jednotlivých toků z IP hlavičky jsou zdrojová IP adresa, cílová IP adresa a typ protokolu (TCP, UDP). Z hlavičky protokolu transportní vrstvy se používají zdrojový a cílový port TCP nebo UDP protokolu. Dále se pro určení příslušnosti paketu do toku využívají charakteristiky samotných paketů (např. počet MPLS tagů) a chování paketů při průchodu síťovou infrastrukturou (např. výstupní rozhraní). IPFIX rovněž poskytuje agregované informace o dané komunikaci, např. o době trvání komunikace či o počtu přenesených bajtů komunikace.



Obrázek 1.2: Schéma fungování NfDumpu popsaného v části 1.2.3. Převzato z [44].

Záznam toku je standardně [43] ukončen po pěti minutách a pro pokračující komunikaci je vytvořen tok nový. Tento mechanismus byl vytvořen kvůli snížení zpoždění získání údajů o dané komunikaci pro toky trvající dlouhou dobu, protože v okamžiku ukončení toku se agregují zjišťované statistiky.

Příkladem opensource projektu postaveného za účelem sběru a analýzy síťových toků je nfdump [44] skládající se z několika modulů. `Nfcapd` je software zajišťující sběr dat o tocích ze sondy, která zpracovává pakety, a jejich uložení do souborů pro následující zpracování. Umí pracovat s daty ve formátu Netflow verze 5, 7 a 9. O zpracování uložených souborů se stará `nfdump`, který tyto soubory čte a na základě zjištěných údajů tvoří statistiky toků dle definovaných požadavků. Pro odfiltrování nepotřebných záznamů z uložených souborů lze použít `nfprofile`, pro přehrání sesbíraných dat `nfreplay`. V projektu jsou přítomny ještě utility `nfclean.pl` a `ft2nfdump` sloužící k odstranění dat, resp. jejich konverzi. Základní princip fungování NfDumpu je zobrazen na obrázku 1.2. Možným rozšířením tohoto projektu je NfSen [45], který poskytuje nejen webové grafické rozhraní usnadňující práci s nasbíranými daty, ale také je poměrně snadno rozšiřitelný o různorodé zásuvné moduly.

Na linuxových strojích může sběr dat a jejich odeslání (ve formátu NetFlow) specifikovanému kolektoru zajistit `Fprobe` [46]. Toto řešení je založeno na libpcap [38] knihovně. Odnož Fprobe nazvaná `Fprobe-ulog` je flexibilní framework sloužící pro náročné logování paketů, např. na firewallech.

Příkladem komerčního bezpečnostního řešení postaveného na IPFIX standardu je produkt Flowmon [47] české společnosti Flowmon Technologies. Sběr dat ze sítě až do rychlosti 100 Gb/s zajišťuje Flowmon sonda, což je pasivní síťové zařízení sbírající informace zejména z druhé, třetí a čtvrté vrstvy OSI

modelu, nicméně částečnou viditelnost poskytuje i do aplikační vrstvy. Data jsou následně odeslána Flowmon kolektoru, který je dále zpracovává. Flowmon kolektor může být rozšířen specializovanými moduly pro pokročilou analýzu datových toků. Zpracování aplikační vrstvy nebylo doposud standardizováno, nicméně se jím zabývají autoři návrhu [48]. Částečnou analýzu několika aplikačních protokolů poskytuje také řešení AppFlow [49] společnosti Citrix.

1.2.4 Rozpoznání aplikací ze zachyceného síťového provozu

Po zachycení paketů síťového provozu je nutné získat ze záznamů relevantní informace. Budu-li uvažovat, že cílem je zpřesnění obecného modelu webového provozu prezentovaného v 1.1.4, jedná se o identifikaci webového provozu v nasbíraných paketových stopách a správné rozlišení jednotlivých objektů přenášených v rámci webového provozu. Zároveň zde zmíním určité problémy a nedostatky jednotlivých metod [31], kterými jsou rozpoznání na základě čísel portů, využití signatur, rozpoznání dle chování uživatele a hloubková analýza payloadu přenášených paketů.

Pro vytvoření modelu aplikace je nutné v nasbíraném síťovém provozu jednotlivé aplikace správně rozlišit. V historii populární a přímočaré řešení spoléhající na cílové porty dnes již bohužel není pro identifikaci stoprocentně účinné především z důvodu stále rostoucího podílu streamovaného videa a P2P provozu. Stejně tak není možné rozlišit různé aplikace využívající stejné číslo portu, tedy např. text, hlas, streaming či přenos souboru mohou být všechny provedeny přes HTTP. Navíc mnoho aplikací využívá porty vyšší než 1024, více paralelních přenosů i přes více transportních protokolů zároveň. Aplikace nemusí být registrovány u IANA nebo mohou využívat dynamické přidělování cílového portu. I přes tyto nevýhody je tato metoda stále velmi populární a efektivní, spoléhá na ni částečně také např. Wireshark.

Možnostmi těchto identifikací se zabývá [11], přičemž cílem je rozdělení veškerého internetového provozu do několika skupin, což by usnadnilo následné zpracování. První možností je komunikační chování aplikací, podle kterého člení aplikační protokoly do skupin podle počtu relací (jedna či více), portu (fixní nebo dynamický) a počtu účastníků komunikace (dva, nebo tři a více). Druhá navrhovaná možnost je založena na hledání závislostí v tocích generovaných stejnými aplikacemi a jejich shlukování. Poměrně zajímavým zjištěním tohoto výzkumu bylo, že pozorovaný počet UDP spojení byl větší než TCP spojení, ale zároveň přes něj bylo přeneseno mnohem méně paketů, což bude nejspíše způsobeno velkým množstvím síťových aplikací, které pro svoji činnost využívají malé UDP pakety.

Pro již zmíněné P2P spojení se nabízí využít přímo hloubkovou analýzu payloadu [50]. U těchto typů spojení obvykle existují dva komunikační kanály, kontrolní a datový. Kontrolní kanál slouží k domluvě na spojení pro datový kanál, která zahrnuje také cílový port. Podobným způsobem funguje také FTP protokol. Tato technika vyžaduje znalost cílového portu pro kontrolní kanál

a selže např. v případě šifrovaného provozu. V případě nutnosti odlišit P2P provoz od zbytku provozu byla navržena metoda TCP-Big [51] detekující tento provoz na základě přenášeného objemu dat v jednotlivých tocích provozu.

Další metoda je založená na známých signaturách, které mohou být kontrolovány proti přenášenému payloadu bez ohledu na použitý protokol. Nevýhodou této metody je poměrně velká výpočetní náročnost jak při definici signatur, tak při jejich hledání v nasbíraném provozu. I přes tuto náročnost je princip signatur využíván např. pro detekce anomalit vyskytujících se v síťovém provozu. Tím se zabývají systémy detekce průniku (Intrusion Detection System, IDS), které se dělí dle principu fungování právě na ty založené na signaturách (např. Snort) spoléhající na známé signatury a na ty založených na detekci anomalit (např. SPADE, NIDES) spoléhající např. na rozdělení IP adres a portů [52]. Výhoda detekce na základě anomalit je odpadnutí nutnosti psaní pravidel a přirozená detekce nových útoků, nicméně neumí říct nic o chování jednotlivých útoků. Pomocí v článku prezentované metody autoři rozdělili anomálie do pěti kategorií – chování uživatele, zneužití zranitelností, anomálie odpovědi jako reakce na útok, chyby v útoku a skrývání.

Identifikace aplikace na základě jejího používání uživatelem vyžaduje předem vytvořený uživatelský profil, kde se následně sledují odchylky vybraných parametrů zkoumaného provozu od referenčních profilů [31]. Celý postup spočívá v extrakci dat paketu (např. zjištění času příchodu, portů, adres, protokolů, velikost paketů, počet audio a video klíčových slov, payload), generování toku a jeho následné analýzy. Generovaný tok se zde skládá z celkového počtu vyslaných a přijatých paketů a bytů, doby trvání, průměrné velikosti přijatého paketu a poměru přijatých a vyslaných paketů.

V této práci se zaměřuji na modelování a generování HTTP provozu. Vzhledem k tomu, že většina webů je přístupná přes cílový port 80, vybral jsem si pro rozpoznání tohoto provozu metodu založenou na portech i přes vědomí výše uvedených problémů. Zde se může jednat zejména o video vestavěné ve stránce, které tímto způsobem zůstane nedetekováno a ovlivní tak výsledné parametry modelu.

Běžný postup v empirickém modelování webového provozu je rekonstrukce chování koncového uživatele z přístupových logů. Pro správné určení parametrů modelu je nutné dobře určit hranice stránek, na což jsou v zásadě dva postupy [53]. První spočívá v analýze času mezi jednotlivými odchozími požadavky. Pokud je tento čas kratší než předem stanovený práh, je požadavek považován za automaticky generovaný prohlížečem, v opačném případě za požadavek generovaný uživatelem. Druhý přístup je využití položky Content-Type objektu. Tento přístup jednoduše považuje každý HTML objekt za požadavek generovaný uživatelem a zbytek za vestavěné objekty. První přístup má nevýhodu v chybném rozpoznání např. Ajaxových požadavků, které jsou obvykle spouštěny s určitým zpožděním a byly by tak klasifikovány jako samostatný požadavek. Druhý přístup zase selže v případě rámců obsažených ve stránce, kdy jsou klasifikovány jako samostatné požadavky. Ve článku [53]

byl prezentován nový přístup nazvaný StreamStructure skládající se ze tří částí – shlukování streamů, detekování hlavních objektů a identifikace výchozí stránky. Pro detekci závislostí při shlukování streamů využívají položku Referer v hlavičce HTTP dotazu.

1.3 Generování síťového provozu

Vytvořené modely síťového provozu popsané v sekci 1.1 mohou sloužit jako podklad pro generátory syntetického provozu. Principy generování takového provozu, požadavky na generátory a popisem několika vybraných generátorů se zabývám v této sekci.

1.3.1 Požadavky na generátory

Generátory síťového provozu injektují do sítě syntetický provoz vytvořený na základě modelu chování uživatele nebo aplikace. Generování provozu na úrovni paketů nelze snadno implementovat pro aplikace využívající transportní protokol TCP kvůli kontrole zahlcení probíhající mezi komunikujícími uzly [22]. Časování příchodu paketů je touto kontrolou ovlivněno kvůli potvrzování přijatých dat a nemůže být modelováno pomocí generátoru bez této kontroly. Z toho plyne, že velkým problémem v síťových simulací je věrohodné generování aplikačně závislého, ale síťově nezávislého provozu korespondujícího s modelem chování uživatele nebo aplikace.

Syntetický síťový provoz lze generovat pomocí hardwarových a softwarových generátorů [54]. Hardwarové generátory jsou obvykle přesnější, s lepším výkonem a tudíž obecně dražší, vyvíjeny jsou zpravidla korporacemi, přičemž své generátory vyvíjí např. společnosti Agilent či Candelatech. Softwarové generátory jsou naopak méně přesné, levnější a obecně bývají vyvíjeny výzkumnými skupinami různých velikostí či univerzitami. Bývají snadněji nasaditelné, snadno reprodukují scénáře generování, umožňují jednoduše rozšířit zdrojový kód pro specifické účely či přidat podporu nových operačních systémů a hardwarových platforem. Přesnost jednotlivého generátoru lze odvodit z popisu jednotlivých charakteristik, např. konfidenčních intervalů (např. pro bit rate), které nemívají softwarové generátory detailně popsané z důvodu závislosti jejich metrologických vlastností na užitém hardwaru či operačním systému, díky čemuž některé jejich vlastnosti zůstávají nejisté.

Generátory lze také rozlišit podle způsobu získání a vytvoření generovaného provozu [3]. Prvním způsobem je přesná replikace obsahu a časování provozu dříve nachytaného v reálném prostředí, tzv. trace-based přístup. Druhý přístup spoléhá na statistické modely, na kterých jsou založeny průběh i procesy generující jednotlivé pakety. Tento přístup se nazývá model-based. Pro generování realistických síťových průběhů je vhodné oba přístupy kombinovat.

Generátory síťového provozu lze rozlišit na základě úrovně detailů, se kterou je provoz generován, do čtyř kategorií [54]:

- Generátory na aplikační úrovni. Generátory spadající do této kategorie emulují chování konkrétní síťové aplikace ve smyslu jí produkovaného síťového provozu. Možným zástupcem je např. Surge [55].
- Generátory na úrovni síťového toku. Tyto generátory se používají v případě potřeby replikace realistického provozu na úrovni toku, např. pro odpovídající počet přenesených paketů či bajtů. Možným zástupcem této kategorie je Harpoon [56].
- Generátory na úrovni paketů. Generátory z této kategorie jsou založeny na velikosti paketů a časem mezi odchody dvou po sobě jdoucích paketů. Hodnoty uvedených proměnných obvykle nastavuje uživatel generátoru, typicky volbou statistického rozdělení každé proměnné.
- Víceúrovňové generátory. Generátory beroucí v úvahu spolupráci mezi více vrstvami protokolového stacku. Možným zástupcem je Swing [57].

Detailnější popis vybraných generátorů spolu s jejich vlastnostmi uvádím v sekci 1.3.2.

Pro dobrou použitelnost by vzniklý generátor měl mít následující vlastnosti [3]:

- vhodným způsobem zachycovat komplexitu reálného provozu v různých scénářích,
- měřit specifické vlastnosti provozu pro účely konkrétního experimentu,
- měřit indikátory výkonu na požadované vrstvě OSI modelu,
- umožňovat ovlivnění semínka generujících náhodných procesů kvůli možnosti zopakování experimentálního provozu.

Zároveň je nutné, aby hypotézy o generovaném provozu byly ověřitelné [4], tj. aby je bylo možné potvrdit či vyvrátit. Pro experimentální vyhodnocení generátoru je mít možnost experiment opakovat a nezávisle tak zkoumat vliv jednotlivých parametrů jejich dílčími změnami. Autoři se zabývají právě tvorbou opakovatelných a ovladatelných experimentů k vyhodnocení síťových útoků na klientské straně a obranou proti nim v izolovaných sítích.

Existují síťové aplikace, např. VoIP, u kterých závisí na časových rozestupech příchodu jednotlivých paketů. Většina dnes používaných síťových generátorů běží uživatelském paměťovém prostoru a spoléhá na kernelovou vrstvu, čímž dochází k tomu, že generované pakety přicházejí ve shlucích, které mohou být pro analýzu síťového protokolu a chování nevhodné [58]. Shluky může ovlivnit mnoho faktorů, jako např. přesnost časovačů, technika rozvrhování

procesů či zvolená implementace, např. softwarová implementace v aplikační vrstvě je nejvíce vzdálená od fyzické vrstvy, a tak velmi závislá na mezivrstvách, které tak ovlivňují generování.

1.3.2 Vybrané generátory síťového provozu

V této sekci stručně popisuji několik vybraných, především softwarových, generátorů síťového provozu.

1.3.2.1 Swing

Swing [59], [57] je software, který zachycuje síťový provoz a následně tento provoz reprodukuje na základě automaticky extrahovaných pravděpodobnostních rozdělení chování uživatele, aplikace a sítě. Cílem generátoru je realisticky reflektovat následující charakteristiky původního provozu:

- časy mezi příchody dvou po sobě příchozích paketů,
- shlukování paketů napříč rozsahem časových škál,
- rozdělení velikosti paketů,
- časy mezi příchody jednotlivých toků a rozdělení jejich délek,
- rozdělení cílových IP adres a portů.

Generátor je citlivý na vstup uživatele, přičemž umožňuje měnit charakteristiky provozu jako např. propustnost linky, poměr provozu jednotlivých aplikací sdílejících linku či změny v provozu jednotlivých aplikací. Pro generování a korektní ověřování provozu byl vytvořen strukturální model zahrnující následující:

- Uživatelé. Tímto se určují charakteristiky v používání různorodých aplikací, je zde zahrnuto např. přemýšlení uživatele mezi návštěvou dvou webových stránek. Bez tohoto modelu by nebylo možné ověřit dopad změn v síti na chování koncového uživatele.
- Relace. Tímto je určena síťová aktivita vedoucí např. ke stažení stránky nebo souboru, může se skládat z více spojení do více destinací.
- Spojení. Relace může obsahovat více spojení, v úvahu se bere destinace, velikost požadavku a korespondující odpovědi, doba čekání na odpověď.
- Charakteristika sítě zahrnující její ztrátovost, kapacitu a latenci.

1.3.2.2 Brute

Brute [60] je síťový generátor běžící na linuxových operačních systémech. Zahrnuje množství modulů, psaných v programovacím jazyku C, implementujících běžné profily síťového provozu, např. konstantní bit rate či Poissonovy procesy. Zároveň je možné rozšířit ho o libovolné moduly pro generování upravených vzorků provozu.

Podstatnou částí návrhu tohoto generátoru bylo rozhodování, zda bude implementován v uživatelském nebo kernelovém paměťovém prostoru. Implementace v uživatelském prostoru má výhodu snadné přenositelnosti a ladění, zatímco výhodou v kernelové části je požadavek na chování v reálném čase, protože zde jsou požadavky oproštěny od zpoždění způsobených mechanismy systémových volání. Experimentálně ověřený rozdíl zpoždění obou implementací byl zanedbatelný, tudíž byla zvolena snadnější implementace v uživatelském adresním prostoru.

1.3.2.3 Harpoon

Harpoon [56] je generátor na úrovni toků pro testování routerů a síťových segmentů. Cílem bylo umožnit generování snadno nastavitelného provozu odpovídajícího reálnému provozu v místě sítě. Model byl vytvořen z následujících sedmi parametrů provozu: velikost souboru, doba mezi požadavky na dva soubory, doba trvání relace, čas mezi požadavky na jednotlivé relace (inter-session request time), IP rozsah, uživatelský aktivní (ON) čas a počet aktivních uživatelů. Umožňuje nastavit provoz tak, aby přesně odpovídal nějaké aplikaci nebo aby byl kompletně aplikačně nezávislý. Použitý model se skládá ze tří úrovní:

- Úroveň souborů. Zahrnuje velikost přenášených souborů a čas mezi požadavky na přenos souboru.
- Úroveň relace. Parametry jsou prostorové rozdělení IP adres, čas mezi dvěma relacemi a délka relace.
- Úroveň uživatele. Zahrnuje počet aktivních uživatelů a tzv. ON čas uživatele.

1.3.2.4 Geist

Geist [61] je generátor síťového provozu vytvořen k zátěžovému testování webových serverů, z toho důvodu se na rozdíl od většiny ostatních tento generátor zaměřuje na charakteristiky agregovaného provozu přicházejícího na server. Poskytuje velké množství parametrů, které dovolují simulovat provoz od statických webových stránek po elektronické obchody. Agregovaný provoz není nijak spojen s konkrétním uživatelem, což způsobuje problémy v okamžiku

potřeby modelování uživatelské úrovně, kdy např. při obchodování v elektronických obchodech musí jednotlivé transakce následovat ve správném pořadí. Geist tento problém řeší virtuálními uživatelskými relacemi.

1.3.2.5 Tcpreplay

Tcpreplay [62] je software vytvořený pro UNIXové operační systémy umožňující užití nasbíraného síťového provozu ve formátu libpcap k otestování různorodých síťových zařízení. Replikovaný provoz může být odesílán s modifikovanými hlavičkami na linkové, síťové a transportní vrstvě OSI modelu.

Celá sada Tcpreplay se skládá z dílčích softwarových nástrojů sloužících k různým úkolům: `tcpreplay` slouží k odesílání paketů, `tcprewrite` modifikuje nachytané pakety, `tcpprep` označuje jednotlivé pakety značkou zdroje (klient nebo server). Každý ze zmíněných modulů poskytuje mnoho možností nastavení, např. změnu IP a MAC adres na náhodně zvolené hodnoty či odesílání paketů v cyklech.

1.3.2.6 D-ITG

Distributed Internet Traffic Generator (D-ITG) [3] je platformně nezávislý distribuovaný generátor. Principielně vychází z ITG (Internet Traffic Generator) [63], což je generátor spustitelný na FreeBSD a linuxových platformách mající dvě základní komponenty – One-way-delay meter a Round-trip-time meter, přičemž lze modelovat časy mezi odchody jednotlivých paketů a zároveň jejich velikost. Z hlediska zjednodušení kódu zde je pro každý spouštěný tok vytvářen nový proces potomka.

Tento software obsahuje mnoho nastavitelných parametrů jako jsou doba trvání, počáteční zpoždění, celkový počet odeslaných paketů či kilobajtů. Podporuje IPv4 i IPv6, transportní protokoly TCP, UDP, IMCP, DCCP, SCTP, nastavitelné zdrojové i cílové IP adresy i porty. Čas mezi odchody paketů a velikosti paketů mohou být určeny zvoleným statistickým rozdělením. Možná rozdělení stejně jako další vlastnosti tohoto generátoru lze nalézt v odkazované literatuře. Software je tvořen čtyřmi základními komponentami:

- ITGSend zajišťuje generování a odesílání síťového provozu.
- ITGRecv zajišťuje příjem síťového provozu.
- ITGLog zodpovídá za příjem a ukládání logovacích informací získaných z předchozích dvou komponent.
- ITGDec zajišťuje parsování logových souborů do lidsky čitelné podoby a pro tvorbu reportů.

ITGSend komponenta může pracovat ve třech různých režimech. Single-flow je režim využívaný pro generování toku paketů s jednou instancí dle parametrů zadaných na příkazové řádce. Multi-flow režim je určen pro generování

provozu na základě informací poskytnutých v konfiguračním souboru a to bez ohledu na to, zda je cílová destinace jedna nebo více. Daemon režim spustí démona naslouchajícího na vybraném portu a poskytujícího API (Application Program Interface) pro ovládání generování.

Jak jsem uvedl výše, generování toků může být řízeno statistickými rozděleními. Ač se v odkazované literatuře může čtenář dočíst, že velikosti generovaných paketů odpovídají zvolenému rozdělení, nemusí to být vždy pravda. Zaprvé nejsou generovány přímo pakety, ale payload celého toku. Celková velikost jednotlivých paketů je tedy vždy zvětšena o konstantí hodnotu odpovídající hlavičkám protokolů na nižších vrstvách. Při experimentech, kdy jsem generoval payload dle exponenciálního rozdělení se zvoleným parametrem λ, nachytané pakety tomuto rozdělení neodpovídaly. Při hledání příčiny jsem zjistil, že vzhledem ke generování síťových toků, tedy provozu na transportní vrstvě OSI modelu, dochází na nižších vrstvách k jejich fragmentaci, čímž dochází k velkému nárůstu počtu paketů s maximální velikosti MTU oproti hodnotám očekávaným dle statistického rozdělení. Celkové rozdělení velikostí payloadů toků ale předpokládanému rozdělení odpovídalo velmi přesně, stejně jako časy mezi odchody jednotlivých paketů.

Objem generovaného payloadu lze specifikovat počtem paketů, celkovou velikostí nebo dobou trvání. Při specifikování všech vyhrává nejvíce omezující volba, nicméně žádná z nich mi při experimentech neposkytla požadovanou přesnost nastavení. Volba celkové velikosti payloadu je umožněna pouze v kilobajtech, přičemž zadaná hodnota je převedena na celočíselnou oříznutím části za desetinnou čárkou. Jednotlivé generované pakety pak mají velikosti payloadů dané statistickým rozdělením, ve výchozím nastavením konstantní hodnota 512 B. Kontrola podmínky dosažení vygenerované velikosti dat se provádí vždy až po odeslání vygenerovaného paketu, což v praxi znamená, že při nastavení konstantní velikosti na 500 B a požadavku na vygenerování provozu o velikost 1 kB (1024 B) dojde k vygenerování 1500 B.

Tento problém jsem se pokusil obejít volbou počtu paketů zadané velikosti, nicméně tento způsob vzhledem k omezením daných velikostí MTU nižších vrstev a pokusu o minimální fragmentaci volbou maximálního možného payloadu v paketech nebyl vždy úspěšný. Při jednoduché aplikaci tohoto způsobu občas docházelo k nesoudělnosti celkové velikosti payloadu s velikostí payloadu ve fragmentovaných paketech a jejich počtem. Tento problém by šel pravděpodobně obejít důkladnějšími výpočty a odpovídajícím nastavením vstupních parametrů, což by ale zase zdržovalo rychlost generování.

Pokus o generování celkového objemu na základě statistických údajů o počtu generovaných paketů a velikosti payloadu za jednotku času a následné nastavení doby trvání pro samotné generování se ukázal jako neefektivní zejména kvůli náhodným zpožděním na síti.

1.3.2.7 TG

TG [64] je generátor jednosměrného síťového provozu, vytvořený pro unixové platformy. Podporuje transportní protokoly TCP a UDP (i multicastové vysílání), provoz je popsán na základě rozdělení času mezi odchody paketů a délek paketů. Tyto parametry provozu jsou určeny zvoleným statistickým rozdělením, kterým může být např. konstantní, rovnoměrné či exponenciální. Informace o provozu jsou logované do binárního souboru, který může následně být zpracován softwarem *dcat*.

Stejně jako D-ITG popsaný v sekci 1.3.2.6, i TG generuje payload a požadovaná velikost je následně zvětšena o velikosti hlaviček protokolů nižších vrstev. Na rozdíl od D-ITG využívá pro veškerou komunikaci (navazování spojení i samotný generovaný provoz) pouze jeden kanál, což podstatně zhoršuje možnosti filtrování provozu kvůli ověřování generovaných dat. Při generování s využitím TCP protokolu dochází ke sdružování více paketů dohromady. Příkladem je generování paketů o konstantní velikosti 576 B, ale vysílány jsou i pakety s velikostí payloadu 1152 B nebo 1728 B, přičemž druhé zmíněné jsou sítí následně fragmentovány a obsahují pouze ACK flag, zatímco ostatní pakety obsahují ještě PSH flag. Naproti tomu nezvyklou, a pro mě nepochopitelnou, vlastností je, že při využití UDP protokolu a požadavku na konstantní velikost payloadu se reálně generovaný payload vždy odchýlil od požadované hodnoty. Nejdříve jsem se domníval, že je velikost požadované velikosti payloadu zaokrouhlována na násobky 5 B směrem k vyšší hodnotě, pro požadavek 576 B byly generovány pakety s velikostí payloadu 580 B, pro požadavek 581 B s velikostí 585 B. Nicméně tuto domněnku jsem musel zamítnou po vyzkoušení požadavku 222 B, pro který byly generovány pakety s velikostí 226 B. Při generování pomocí protokolu TCP ani při využití exponenciálního rozdělení jsem tento problém nezaznamenal.

Další zajímavou nepříjemnou vlastností je doba spouštění samotného softwaru. Při nastavování experimentu je nutné zadat po jaké době od spuštění se má experiment spustit. Při požadavku na krátký čas však často dochází k tomu, že po spuštění programu je požadovaný čas již v minulosti a program se tak ihned ukončí.

1.3.3 Simulátory

1.3.3.1 NCTUns

NCTUns [65] je rozšiřitelný síťový simulátor a emulátor mnoha protokolů užívaný v klasických i bezdrátových IP sítích. Na rozdíl od konkurenčních produktů nespoléhá na abstrakci protokolů, díky čemuž je velmi přesný a tím i užitečný. Přestože byl vyvinutý akademickou sférou, následně se stal komerčním produktem pod názvem Estinet 9.0 a dnes je nemožné vyzkoušet ho zdarma. Umožňoval tvorbu topologie sítě v grafickém prostředí, monitoring výkonností či animace toku paketů. Podporuje distribuované emulace velkých

sítí na více fyzických strojích řízené jedním centrálním, využívá přímo s linuxovým TCP/IP protokolovým stackem k dosažení simulace v reálném čase. Zároveň je možné přímo v simulovaném uzlu spustit reálnou aplikaci zdrojového provozu stejně jako monitorovací software, např. tcpdump či traceroute.

1.3.3.2 ns-2

NS-2 simulátor [66] je díky své implementaci v C++ snadno přenositelný a pokrývá řadu síťových protokolů a aplikací. Jednotlivé experimenty mohou být snadno připraveny a modifikovány díky využití Tcl a Otcl programování, kde lze snadno definovat i topologii experimentální sítě a parametry síťových linek, což je např. ztrátovost.

1.4 HTTP protokol

Při modelování HTTP provozu existují problémy plynoucí z jeho fungování [6]. Jednotlivé vlastnosti a problémy a možná řešení popisuji v této sekci.

1.4.1 Vlastnosti sítě a transportní protokol

HTTP protokol standardně využívá transportní protokol TCP, který se pro tento účel příliš nehodí. Přístup na web obvykle znamená přenos většího množství souborů, které obsahují text či obrázky. Tomuto faktu nasvědčují také hodnoty parametrů modelu diskutovaného v sekci 1.1.4.2, kde je velmi malá střední hodnota velikostí vestavěných souborů vůči velkému rozptylu. Problémem zde je režie způsobená navazováním nového TCP spojení pro přenos malého objektu, které využije více paketů na svou správu (vytvoření a uzavření komunikačního kanálu) než na přenesení požadovaného objektu.

Kromě režie při navazování spojení je problémem TCP také pomalý start přenosu a určování velikosti okénka pro přenos segmentů. Algoritmus pomalého startu spočívá v postupném růstu velikosti okénka, které je inicializováno na jeden segment a postupně roste při každém přijetí potvrzení (TCP ACK). Nicméně v okamžiku chyby kanálu, např. z důvodu zahlcení, je velikost okénka ihned snížena na polovinu. Pro přesné modelování HTTP provozu by tak bylo vhodné znát počty odeslaných TCP segmentů mezi přijatými potvrzeními. Na to se zaměřili např. autoři článku [6], kteří předpokládají exponenciální růst počtu odeslaných segmentů.

Další možnou nevýhodou spoléhání se na TCP protokol je také fakt, že přenos HTTP objektů může být kdykoliv ukončen. Pro správné modelování velikosti objektů je také problematické vypořádání se se ztracenými, zopakovanými či přehozenými pakety a to vše ideálně bez nahlížení do payloadu, což vede k vytvoření mechanismu rozlišení dat a metadat.

Jevem ztěžujícím analýzu jednotlivých spojení je také fragmentace jednotlivých zpráv způsobená vlastnostmi sítě. Vzhledem k tomu, že HTTP hlavička

i odpověď mohou být téměř náhodně dlouhé, způsobí fragmentace jejich možné rozdělení přes několik paketů a následně problematické skládání.

Aby se tyto problémy eliminovaly, vytvořily se různé návrhy a opatření, mezi něž patří perzistentní TCP spojení, transakční TCP spojení či využití jiných transportních protokolů. Jednotlivé možnosti jsou popsány v následujících podsekcích.

1.4.1.1 Perzistentní spojení

Perzistentní spojení spočívá v principu vytvoření jednoho TCP spojení mezi komunikujícími stranami, které je následně využito více HTTP relacemi. Toto spojení může být uzavřeno jednou z komunikujících stran nebo automaticky při expiraci časového intervalu neaktivity v komunikačním kanále. Tento mechanismus vede k šetření zdrojů na straně serveru i sítě. Silným se stává zejména v okamžiku využití pipeliningu, ve kterém zároveň probíhá několik přenosů bez čekání na dokončení předchozích.

Tento princip komunikace značně ztěžuje analýzu většiny modelovaných atributů HTTP provozu, ať už se jedná o počty přenášených souborů, rozestupy mezi jejich přenosy či jejich velikosti. Naopak předpoklad nezávislosti jednotlivých požadavků modelování usnadňuje. Možnostmi získání atributů z modelu se zabývám v části návrhu 2.1.3.

1.4.1.2 Transakční TCP spojení

Transakční TCP spojení spočívá v cachování informací identifikující zdrojové zařízení, čímž dochází k obcházení třícestného navázání TCP spojení a následnému vyhnutí se pomalému startu. Informace o navázání spojení jsou při prvním spojení uloženy do cache a při příštích přístupech načteny a použity, čímž je bez zbytečného čekání využita i použitelná velikost okénka. Z hlediska HTTP modelování ovlivňuje tento typ spojení časové intervaly mezi jednotlivými přenosy, nicméně samotnou detekci přenosů nijak neovlivňuje.

1.4.1.3 Využití UDP protokolu

Pro HTTP přenosy lze použít také protokoly spoléhající na UDP, příkladem takového protokolu je ARDP (Asynchronous Reliable Delivery Protocol) [67], jehož cílem je poskytnutí spolehlivé, ale odlehčené komunikace mezi klientem a serverem. Funkčnost spočívá v nahrazení třícestného navázání spojení zvolením náhodného identifikátoru nicméně stále trpí pomalým startem kvůli kontrole zahlcení.

Příkladem transportního protokolu zaměřeného přímo na užití ve webovém provozu je protokol QUIC (Quick UDP Internet Connections) [1]. Protokol obsahuje vestavěné šifrování s využitím TLS, tudíž se nepoužívá pro HTTP, ale HTTPS. Výhodou jsou např. nižší latence při navazování spojení, ovládání zahlcení, multiplexing a další.

1.4.2 Obsah HTTP dat

HTTP hlavička obsahuje položky, např. Content-Length či Referer, které by zdánlivě měly usnadnit analýzu HTTP provozu poskytnutím konkrétních údajů o přenášeném objektu. Díky uvedeným položkám by měla jít snadno určit velikost přenášených objektů, resp. zdroj, ze kterého byl přenos vyvolán. Faktické informace použité v této sekci čerpám z [68].

1.4.2.1 Velikost přenášeného objektu

Většina HTTP objektů je přenášena v komprimované formě, do které je objekt převeden na straně serveru ještě před zahájením přenosu, např. nástrojem gzip. Největší vliv má komprese u rozsáhlých textových objektů, zatímco třeba pro gif obrázky nepřináší prakticky žádný efekt a zbytečně zatěžuje server. Položka HTTP hlavičky Content-Length je nepovinná, proto v hlavičce informace o velikosti přenášených dat často chybí. Navíc při využití komprese tato položka nesděluje velikost samotného objektu, ale pouze velikost jeho komprimované verze.

Položka Content-Length je nepovinná zejména z důvodu přenášení dynamicky generovaných dat. V případě, že by byla povinná a odpověď měla zahrnovat např. dlouhou HTML tabulku s daty získanými SQL dotazem z databáze, musel by být výsledek dotazu uložen do bufferu, spočítána jeho velikost a teprve poté by mohl být zahájen přenos směrem ke klientovi. Praxe je však taková, že jak jsou data získávána, tak jsou po kouskách (tzv. chunks) odesílána ke klientovi a konec přenosu je signalizován odesláním kousku s nulovou velikostí datové části.

Poslední vlastností ovlivňující velikost odpovědi je webový klient, který odeslal daný požadavek. Zejména dříve nebylo výjimkou, že pro různé prohlížeče byla serverem vracena různá data.

1.4.2.2 HTTP Referer

Informace, na základě navštívení jaké stránky byl stažen který objekt, se nejjednodušeji získá z položky hlavičky HTTP požadavku Referer. Ta je sice také nepovinná, nicméně všechny moderní webové prohlížeče ji poskytují a v hlavičce odesílají.

1.4.2.3 Data v GET požadavku

Velikost jednotlivých GET požadavků se liší nejen délkou požadovaného URL, ale také daty, která jsou pomocí těchto požadavků odesílána na server. Tato data musí být buď zakódována do URL požadavku nebo do cookie. Tedy přesto, že přes GET požadavek nemůže být např. nahrán soubor na server, velikost těchto požadavků se může výrazně lišit.

Kapitola 2

Návrh

V této práci se zaměřuji na HTTP protokol, na kterém se dají zkoumat následující charakteristiky:

- Z hlediska struktury webu se lze zaměřit na počty jednotlivých vestavěných objektů v hlavní stránce, jejich velikost, entropii jejich obsahu nebo pořadí jejich odkazování ve zdrojovém kódu HTML stránky. Dále lze zkoumat rozdělení jednotlivých metod požadavků HTTP protokolu, jako jsou GET, POST, HEAD a další.

- Z hlediska chování uživatele lze zkoumat rychlost klikání na zobrazené odkazy v závislosti na jejich rozmístění a označení ve stránce, dobu prohlížení jednotlivých stránek nebo rychlost scrollování při hledání určité informace.

- Z hlediska chování síťové infrastruktury se lze zaměřit na počty cachovaných souborů, na časy, po které trvá přenos souborů, časy mezi jejich příchody nebo doba jejich zpracování, např. od přijetí do provedení odpovídající reakce, což v tomto případě může být zobrazení obrázku v HTML stránce.

Tato kapitola je rozdělena na část návrhu modelu a na část návrhu generování síťového provozu.

Pro implementaci zvažuji programovací jazyky C++ a Python verze 3. Velkou výhodou C++ je rychlejší běh výsledného programu a nižší paměťová náročnost. Naopak zdrojový kód zapsaný v Pythonu je přehlednější a kratší [69]. Python navíc disponuje množstvím rozšiřujících modulů knihoven, jako např. NumPy [70] či SciPy [71]. Z těchto důvodů navrhuji pro implementaci zpracování vstupních dat k tvorbě modelu, např. z pcap souborů či IPFIX toků, jazyk C++, pro zajištění zbytku funkcionality Python.

Tabulka 2.1: Mapování obsahu Content-Type položek na typ objektů v modelu

Typ objektu	Obsah Content-Type položky
Hlavní	text/html
CSS	text/css
Javascript	text/javascript
Obrázky	image/png, image/jpeg, image/gif
Ostatní	Všechny neuvedené pro CSS, javascriptové a obrázkové typy objektů

2.1 Model

2.1.1 Návrh modelu

Přenášené HTTP objekty jsem rozdělil na hlavní objekty a čtyři kategorie objektů vestavěných. Typy objektů definuji na základě obsahu Content-Type položek v hlavičce HTTP odpovědi a tyto definice jsou uvedeny v tabulce 2.1.

Nevýhodou navržených definic je fakt, že při využití html tagu `<iframe>` v rámci html stránky pro zobrazení jiné html stránky, dojde k započítání vnořené stránky jako nového hlavního objektu. Abych se této situaci vyhnul, bylo by nutné definovat jednotlivé vestavěné objekty na základě jejich odkazování přímo z těla html stránky. To by znamenalo nutnost znát přenášený obsah, který je mnohdy přenášen v komprimované podobě a jeho zpětné dekomprimování by program zdržovalo. V případě využití IPFIX toků pouze se znalostí časování jednotlivých toků by byl vnořený html objekt správně detekován jako vestavěný. Tato metoda by naopak selhala v okamžiku, kdy uživatel v krátkém časovém intervalu začne stahovat více html stránek.

Pro jednotlivé typy objektů budu v souladu s výchozím modelem sledovat jejich velikosti a pro vestavěné objekty také jejich počty. Za velikost objektu, ať už hlavního nebo vestavěného, považuji velikost payloadu aplikační vrstvy, tedy délku datové části TCP segmentu po odečtení délky hlavičky HTTP protokolu. Není výjimečné, že tyto objekty jsou velké, proto je nutné počítat s rozdělením objektu přes několik paketů, které navíc mohou být proloženy pakety patřícími k přenosu nesouvisejících dat. Za počet vestavěných objektů daného typu považuji všechny objekty tohoto typu, které byly prohlížečem automaticky staženy na základě odkazu obsaženého ve zdrojovém kódu hlavního objektu. Pro detekci takového stažení využívám hodnotu položky Referer v HTTP hlavičce požadavku, ač tato metoda není stoprocentní a selže např. v okamžiku, kdy uživatel otevře vestavěný obrázek na samostatnou stránku. Parametry provozu obsažené v navrhovaném modelu jsou spolu s definicemi uvedeny v tabulce 2.2.

Tabulka 2.2: Parametry provozu navrhovaného HTTP modelu.

Parametr provozu	Definice
Velikost požadavku	Ve výchozím návrhu se zaměřuji pouze na HTTP požadavky provedené metodou GET. Za velikost požadavku považuji součet velikostí payloadu všech segmentů TCP přenášejících daný požadavek. Je nutné počítat s možností přenosu požadavku rozděleného přes více paktů.
Velikost hlavního objektu	Velikost hlavního objektu v souladu s definicí uvedenou v 2.1.1.
Velikost css souboru	Velikost css objektu v souladu s definicí uvedenou v 2.1.1.
Velikost javascriptového souboru	Velikost javascriptového objektu v souladu s definicí uvedenou v 2.1.1.
Velikost obrázkového souboru	Velikost obrázkového objektu v souladu s definicí uvedenou v 2.1.1.
Velikost ostatních souborů	Velikost ostatních objektů v souladu s definicí uvedenou v 2.1.1.
Počet css objektů	Počet stažených css objektů, které měly v položce Referer HTTP hlavičky uvedeno URL daného hlavního objektu.
Počet javascriptových objektů	Počet stažených javascriptových objektů, které měly v položce Referer HTTP hlavičky uvedeno URL daného hlavního objektu.
Počet obrázkových objektů	Počet stažených obrázkových objektů, které měly v položce Referer HTTP hlavičky uvedeno URL daného hlavního objektu.
Počet ostatních objektů	Počet stažených ostatních objektů, které měly v položce Referer HTTP hlavičky uvedeno URL daného hlavního objektu.
Čas parsování	Délka časového intervalu od ukončení přenosu hlavního objektu do zahájení přenosu prvního vestavěného objektu.
IAT vestavěných objektů	Délka časového intervalu mezi začátkem přenosu dvou po sobě jdoucích vestavěných objektů.
Čas prohlížení	Délka časového intervalu mezi ukončením přenosu vestavěného objektu, jehož přenos skončí jako poslední, hlavního objektu i a odeslání HTTP požadavku na hlavní objekt $i+1$.

Pro každý modelovaný parametr provozu je nutné zjistit jeho statistické parametry jako jsou střední hodnota, směrodatná odchylka, nejlépe se hodící

statistické rozdělení a případně medián. Tyto údaje lze získat různými způsoby podle toho, jaká data jsou dostupná, což diskutuji v sekci 2.1.3. V okamžiku, kdy znám uvedené statistické údaje, mohu spočítat parametry jednotlivých statistických rozdělení, ze kterých jsou v průběhu generování HTTP provozu generovány konkrétní náhodné hodnoty. Podle výchozího modelu v tabulce 1.1 budou implementována Lognormální, Weibullovo a Gamma rozdělení. Výpočty parametrů těchto rozdělení jsou popsány v sekci 2.1.2.

Takto navržený model je možné dále zpřesňovat. Jako první se nabízí vytvoření individuálních kategorií pro další typy vestavěných objektů, např. na poslední dobou stále častější video objekty. Další nabízející se možností je rozlišení obrázkových souborů na jednotlivé kategorie. Toto lze při znalosti Content-Type signatur udělat přímo, nicméně v okamžiku, kdy není vhodné či nutné zasahovat do zdrojového kódu, lze využít EM algoritmus popsaný v sekci 2.1.3.3.

2.1.2 Výpočty parametrů statistických rozdělení

V této sekci prezentuji postup výpočtu parametrů jednotlivých statistických rozdělení při znalosti středních hodnot a rozptylů dat. Definice jednotlivých rozdělení jsou v souladu s [72]. Z takto definovaných rozdělení s parametry spočítanými níže uvedeným způsobem jsou následně při běhu programu generovány jednotlivé náhodné hodnoty.

2.1.2.1 Gamma rozdělení

Pro střední hodnotu a rozptyl Gamma rozdělení $X \sim Gamma(\alpha, \beta)$ platí:

$$EX = \alpha \cdot \beta, \qquad VarX = \alpha \cdot \beta^2.$$

Z těchto vzorců lze snadno vyjádřit parametry rozdělení jako:

$$\alpha = \frac{(EX)^2}{VarX} \qquad \text{a} \qquad \beta = \frac{VarX}{EX}.$$

2.1.2.2 Lognormální rozdělení

Pro střední hodnotu a rozptyl Lognormálního rozdělení $X \sim lnN(\mu, \sigma^2)$ platí:

$$EX = \mathrm{e}^{\mu+\sigma^2/2}, \qquad VarX = (\mathrm{e}^{\sigma^2} - 1) \cdot \mathrm{e}^{2\mu+\sigma^2}.$$

Z těchto vzorců lze snadno vyjádřit parametry rozdělení jako:

$$\sigma = \sqrt[2]{\ln \frac{VarX + (EX)^2}{(EX)^2}} \qquad \text{a} \qquad \mu = \ln(EX) - \frac{\sigma^2}{2}.$$

2.1.2.3 Weibullovo rozdělení

Pro střední hodnotu a rozptyl Weibullova rozdělení $X \sim Weibull(\alpha, \beta)$ platí:

$$EX = \beta \cdot \Gamma(1 + \frac{1}{\alpha}), \qquad VarX = \beta^2 \cdot \Gamma(1 + \frac{2}{\alpha}) - (EX)^2.$$

Z rovnice pro střední hodnotu lze vyjádřit parametr β jako $\beta = \frac{EX}{\Gamma(1+\frac{1}{\alpha})}$ a po jeho dosazení do rovnice pro rozptyl numericky řešit rovnici

$$varX - (EX)^2 \cdot \frac{\Gamma(1 + \frac{2}{\alpha})}{(\Gamma(1 + \frac{1}{\alpha}))^2} + (EX)^2 = 0,$$

čímž lze získat parametr α a po jeho zpětném dosazení do původní rovnice střední hodnoty dopočítat i parametr β.

2.1.3 Získání dat pro tvorbu modelu

Ve výsledném programu předpokládám dvě možnosti automatizovaného získání dat pro tvorbu modelu. První z nich je extrakce dat z pcap souboru, druhou je extrakce dat ze souborů v IPFIX formátu. Obě možnosti rozebírám v této sekci, v její poslední části se zabývám využitím EM algoritmu pro zpřesnění vybraného modelu.

2.1.3.1 Extrakce dat z pcap souboru

Při získání pcap souboru se všemi pakety sledované komunikace je tvorba modelu nejsnazší díky množství obsažených detailních informací. Vzhledem k častému využívání perzistentního spojení mezi komunikujícími stranami nelze statistiky získat pouze na základě rozlišení jednotlivých TCP spojení, ale je nutné zkoumat přímo aplikační vrstvu. Ta všechny informace potřebné pro tvorbu navrženého modelu obsahuje.

Výsledný software bude procházet poskytnutý pcap soubor paket po paketu a extrahovat z něj potřebné údaje. V této fázi prvního návrhu a implementace jsem se rozhodl zaměřit pouze na GET požadavky. GET požadavky mohou obsahovat dlouhá URL způsobené množstvím přenášených parametrů a také dlouhé řetězce identifikující cookie. Zejména tyto dva parametry mohou způsobit, že požadavek bude fragmentován a přenášen několika pakety. Za velikost požadavku je tedy považován součet velikostí payloadu všech segmentů transportní vrstvy přenášející daný požadavek. První segment je identifikován pomocí začátku hlavičky obsahující GET, poslední pak pomocí řetězce `\r\n\r\n` v payloadu segmentu.

Po detekci HTTP požadavku bude v analyzátoru pcap souboru vytvořena struktura představující dané komunikační spojení. Tato struktura bude otevřena od detekce HTTP požadavku po ukončení přenosu související odpovědi, bude udržovat časy prvního a posledního příchozího paketu obsahujícího odpověď na předchozí požadavek a zároveň velikost payloadu aplikační vrstvy,

URL adresu požadavku či URL adresu položky Referer v HTTP hlavičce. Čas počátku HTTP odpovědi je stanoven jako čas zachycení paketu obsahujícího HTTP hlavičku odpovědi přicházející na stejný port, ze kterého byl dříve odeslán HTTP požadavek. Čas konce HTTP odpovědi je stanoven jako čas zachycení posledního paketu v daném aplikačním spojení, který uzavře tuto programovou strukturu. Paket uzavírající strukturu je takový, který směřuje na cílový soket, ze kterého byl odeslán požadavek, a je zachycen před detekcí následujícího požadavku odeslaného ze stejného cílového soketu. Za tento paket je považován také paket obsahující segment, který uzavírá TCP spojení (obsahuje FIN návěstí) a zároveň velikost jeho datové části není nulová.

Po uzavření všech struktur spojení se struktury rozliší na ty obsahující hlavní a na ty obsahující vestavěné objekty. Následně se struktury s vestavěnými objekty přiřadí k těm s hlavními podle následujícího klíče:

- hodnota proměnné označující položku Referer vestavěného objektu se shoduje s URL adresou hlavního objektu,
- čas začátku přenosu HTTP odpovědi vestavěného objektu následuje po čase začátku přenosu hlavního objektu,
- čas začátku přenosu HTTP odpovědi vestavěného objektu předchází požadavku na další hlavní objekt odeslaný ze stejného soketu, ze kterého byl odeslán nyní zkoumaný hlavní objekt.

Jednotlivé struktury s hlavními objekty jsou vzestupně seřazeny dle časů počátku přenosu HTTP odpovědi. Analogicky, struktury s vestavěnými objekty příslušející stejnému hlavnímu objektu jsou vzestupně seřazeny dle času počátku přenosu HTTP odpovědi a následně jsou spočítány hodnoty časových parametrů provozu následujícím způsobem:

- Čas parsování jako délka časového intervalu mezi časem ukončení přenosu hlavního objektu a časem zahájení přenosu prvního vestavěného objektu příslušejícího danému hlavnímu objektu. Pokud je tento čas menší než nula, což může být důsledkem chování prohlížeče, který odesílá požadavky na vestavěné objekty ještě před přijetím kompletního hlavního objektu, je tato hodnota stanovena na nulu.
- Čas mezi příchody vestavěných objektů jako délka časového intervalu mezi časy zahájení přenosů dvou po sobě jdoucích vestavěných objektů.
- Čas prohlížení stránky jako délka časového intervalu mezi časem ukončení přenosu posledního vestavěného objektu a času odchozího HTTP dotazu příslušejícího následujícímu HTTP hlavnímu objektu. Pokud je tato hodnota menší než nula, což může být způsobené tím, že uživatel odešle požadavky na několik hlavních objektů v krátkém časovém intervalu a až následně je začne číst, je tato hodnota stanovena na nulu.

Jednotlivé údaje o počtech vestavěných objektů jsou pak získány spočítáním prvků obsažených v daných strukturách. Výhodami tohoto zpracování je, že i několik HTTP spojení v rámci jednoho perzistentního TCP spojení je správně detekováno a určení velikostí jednotlivých objektů nezávisí na obsahu položky Content-Length v HTTP hlavičce odpovědi. Naopak nevýhodou je, že i kliknutí na obrázek vestavěný ve stránce, které vede např. na jeho načtení v plné velikosti, je tímto způsobem započítáno jako další vestavěný objekt. Chybné získání parametrů by způsobil také přenos HTTP objektů pomocí pipelinování jednotlivých přenosů.

Při práci s pcap souborem se může vyskytnout problém, kdy se některé pakety nepodařilo vůbec zachytit nebo se paket zachytil či uložil poškozen. Tento problém pomáhá eliminovat využití IPFIX toků popsané v bezprostředně následující části.

2.1.3.2 Extrakce dat z IPFIX toků

Jak jsem uvedl v sekci 1.2.3 popisující standard IPFIX, lze tato data považovat za záznam statistik z transportní vrstvy. Vzhledem k perzistentnímu spojení využívaném pro přenos HTTP provozu se domnívám, že z takových dat nelze o složení HTTP provozu téměř nic zjistit. Předpokládám-li, že bych se zaměřoval na toky z portu a na port 80 pro vybraný server, diagram toků vytvořený podle principu použitého na obrázku 1.1 by mohl vypadat několika způsoby:

- Veškerý HTTP provoz by probíhal sekvenčně v rámci jednoho TCP spojení, vše by tedy vypadalo jako dva toky, jeden každým směrem. Z těchto informací je možné sestavit pouze model celkového objemu přenesených dat v rámci jednoho spojení.
- Veškerý HTTP provoz by byl rozdělen do několika paralelních toků, nicméně stále s využitím perzistentního spojení. V takovém případě by šla modelovat alespoň transportní vrstva takového spojení, konkrétně okamžiky navazování komunikace a objem dat přenesený v rámci jednoho perzistentního spojení.
- Perzistentní spojení by nebylo využito, čímž by každý vytvořený tok popisoval jeden přenášený objekt. Časový diagram vytvořený z těchto toků by pak mohl vypadat velmi podobně jako model na obrázku 1.1. Z těchto informací by šlo s určitou chybou, způsobenou např. variabilní délkou HTTP hlaviček, sestavit výchozí model. V principu by šlo o detekci jednoho toku, za nímž následuje sekvence dalších toků vytvořená v krátkém časovém intervalu. Tato množina toků by označovala přenos jednoho hlavního objektu spolu se všemi jeho vestavěnými objekty.

Z výše uvedených poznámek plyne, že pro modelování provozu na aplikační vrstvě by šlo využít IPFIX toky mezi určitými porty. Nicméně pro získání

stejně detailní úrovně navrhovaného modelu jako při extrakci dat z pcap souborů, tedy rozlišení typů jednotlivých vestavěných souborů, lze buď rozšířit šablonu IPFIX toků o informace z aplikační vrstvy nebo využít EM algoritmus na hledání směsí statistických rozdělení. Z hlediska informací z aplikační vrstvy by se jednalo zejména o informace z HTTP hlavičky jako např. Content-Type pro určení typu vestavěného objektu či Content-Length pro určení jeho velikosti. Z toho důvodu by jednotlivé toky musely poskytovat informace o přenesených bajtech aplikační vrstvy po detekování a odečtení délky aplikační hlavičky, v opačném případě by nešlo vytvořit model přenesených souborů, ale pouze model přeneseného objemu provozu na aplikační vrstvě. EM algoritmus popsaný v sekci 2.1.3.3 by v tomto případě šlo využít pro rozpoznání přenosů jednotlivých typů vestavěných objektů v rámci všech vytvořených toků příslušejících přenosům vestavěných objektů.

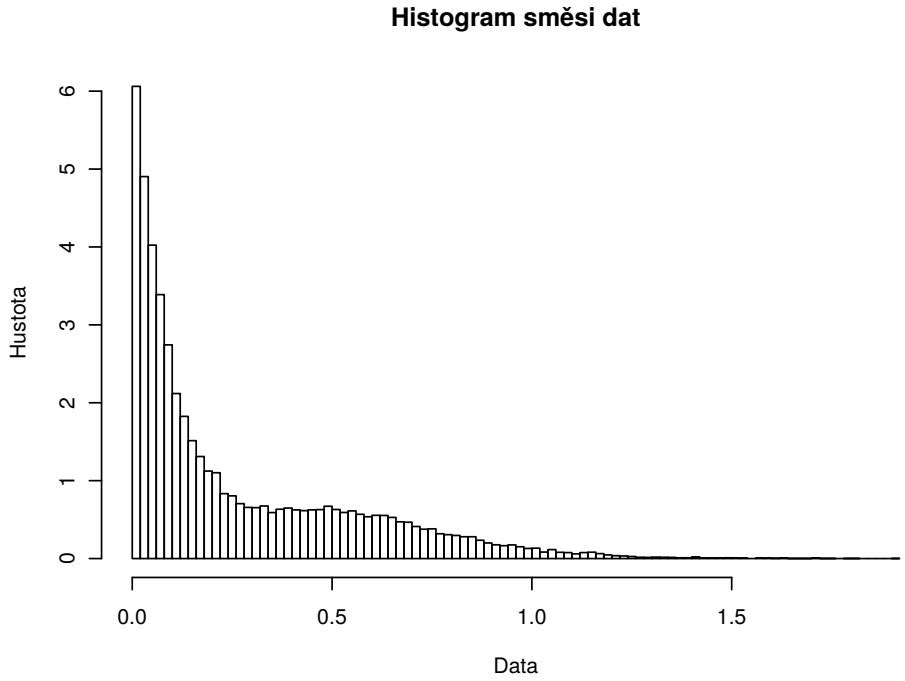
2.1.3.3 EM algoritmus

V případě získání množiny dat, která s velkou pravděpodobností vznikla smícháním nebo nerozlišováním menších množin, lze pro určení těchto původních množin využít EM algoritmus, popsán např. pro geometrická rozdělení v [73]. Algoritmus spočívá v odhadování poměrů zastoupení jednotlivých množin ve výsledné a zároveň odhaduje pravděpodobnostní funkci, resp. hustotu pravděpodobnosti, rozdělení každé podmnožiny. Algoritmus funguje iterativně, přičemž zpřesňování odhadů funguje na základě odhadu a následné maximalizace věrohodnostní funkce příslušející danému rozdělení.

V navrhovaném modelu byly velikosti jednotlivých objektů rozděleny podle Lognormálního rozdělení, jejich počty pak podle Gamma rozdělení. V okamžiku, kdy nelze ze získaných dat určit konkrétnější vlastnosti objektů, např. MIME typ vestavěného objektu, může využití tohoto algoritmu přinést řešení. Vhodnost jeho využití však závisí na konkrétních datech. Pokud histogram dat, příslušejících např. počtu obrázkových vestavěných objektů, vypadá obdobně jako na obrázku 2.1 a z dříve analyzovaných dat pro tuto náhodnou veličinu předpokládám Gamma rozdělení, lze zejména z téměř konstantní výšky sloupců mezi hodnotami 0.25 a 0.5 na ose x usuzovat, že tato data jsou získána smícháním několika dílčích rozdělení. Tato konkrétní směs byla pořízena smícháním dvou Gamma rozdělení popsaných v tabulce 2.3.

Pakliže vím, že se jedná o vestavěné obrázkové objekty, lze odhadovat, že v rozdělení mohou být smíchané malé *.gif* obrázky, fotografie ve formátu *.jpeg*, nekomprimované *.png* obrázky či malé ikonky formátu *.ico*. Podle toho se následně můžu rozhodnout, směs kolika rozdělení budu hledat. Pro směs na obr. 2.1 jsem algoritmus spustil pro hledání dvou komponent, s výsledkem na obr. 2.2 a v tabulce 2.4, poté tří komponent s výsledkem na obr. 2.3 a v tabulce 2.5.

V uvedeném případě nelze z grafů ani konkrétních hodnot s jistotou říct, směs kolika rozdělení byla analyzována. Využití tohoto algoritmu pro zpřesnění



Obrázek 2.1: Směs Gamma rozdělení vytvořená dle parametrů v tabulce 2.3.

Tabulka 2.3: Vytvořená směs Gamma rozdělení zobrazená na obr. 2.1.

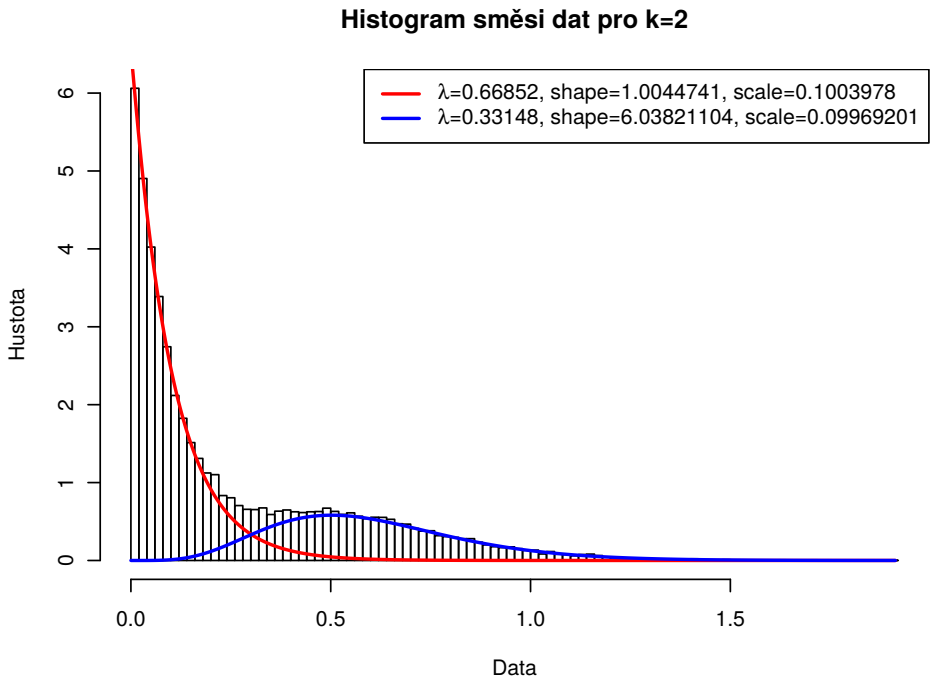
Velikost množiny	Shape	Scale
20000	1	0.1
10000	6	0.1

Tabulka 2.4: Výsledky hledání dvou komponent ve směsi Gamma rozdělení. P-hodnota pro Kolmogorov-Smirnův test je 0.9286.

λ	Shape	Scale
0.66852	1.0044741	0.1003978
0.33148	6.03821104	0.09969201

Tabulka 2.5: Výsledky hledání tří komponent ve směsi Gamma rozdělení. P-hodnota pro Kolmogorov-Smirnův test je 0.9833.

λ	Shape	Scale
0.1749158	0.98564520	0.03319391
0.4908072	1.46429107	0.08397079
0.3342770	6.02471339	0.09974719



Obrázek 2.2: Výsledek hledání dvou komponent ve směsi Gamma rozdělení.

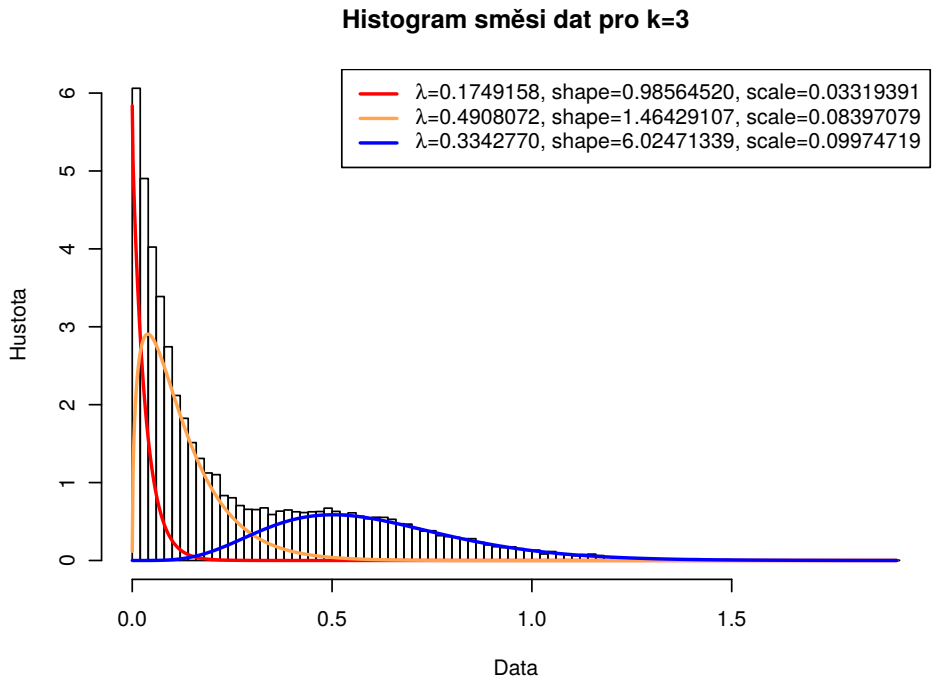
vytvořeného modelu je tedy silně ovlivněno interpretací analytika, což zároveň ztěžuje plně automatizované využití této metody. V tomto případě byla směs vytvořena ze dvou dílčích množin, nicméně tři nalezená rozdělení proložila výslednou směs lépe.

2.2 Generování

Cílem navrhovaného softwaru je generování aplikačního HTTP provozu mezi serverem a několika klienty. Jednotlivé navrhované úrovně, fáze a vlastnosti generování jsou popsány v této sekci. Generování síťového provozu lze provádět buď v reálném čase, nebo nejdříve simulaci kompletně připravit a následně spustit. První možnost využívají režimy posílání souborů a stochastického generování, jejichž návrh je popsán v sekci 2.2.3.2, resp. 2.2.3.3, na principu přípravy scénáře a jeho následného spuštění funguje režim přeposílání paketů navržený v sekci 2.2.3.1.

2.2.1 Sdílená sekvence

Při tvorbě podobných experimentů, kdy je požadována komunikace jednoho serveru s několika nezávisle se chovajícími klienty, je užitečné, když jsou ná-



Obrázek 2.3: Výsledek hledání tří komponent ve směsi Gamma rozdělení.

hodně generované sekvence mezi klienty sdíleny. V případě, kdy by každý klient fungoval na základě privátní náhodné sekvence závislé pouze na vstupním semínku generování, mohlo by dojít k následujícím nechtěným situacím.

- Generátor náhodné sekvence každého z N klientů je inicializován stejným semínkem. Toto vede k paralelnímu generování N stejných sekvencí hodnot a tím naprosto zjevně závislému chování jednotlivých klientů.
- Generátor náhodné sekvence každého z N klientů je inicializován náhodnou hodnotou. Pak nezávislost generovaných sekvencí závisí na délce takové sekvence a stále existuje nezanedbatelná pravděpodobnost, že generované hodnoty v sekvencích jednotlivých klientů budou závislé.

Řešením uvedených problémů s generováním náhodných sekvencí je vytvoření jedné náhodné sekvence, která je sdílena všemi klienty. Z toho důvodu jsem se rozhodl pro umístění generátoru na server, který bude získané hodnoty distribuovat mezi všechny připojené klienty. Vzhledem k tomu, že generátor bude napsán v jazyku Python a obsluha jednotlivých klientů bude zajištěna pomocí sólo vláken běžících v rámci procesu, ověřil jsem sdílení náhodné sekvence mezi spuštěná vlákna následujícím kódem.

Listing 2.1: Zdrojový kód pro ověření sdílení náhodné sekvence mezi jednotlivá výpočetní vlákna

```
#! /usr/bin/python3

import random
import threading
import time

lockGen = threading.Lock()
lockWr = threading.Lock()
g_Res = []
g_MaxPerThread = 10
g_Threads = 4

def getVal():
  return random.uniform(1, 100)

def threadFun(threadNr, c):
  global g_Res
  tloc = threading.local()

  for i in range(1,c):
    lockGen.acquire()
    tloc.val = random.uniform(1, 100)
    lockGen.release()

    lockWr.acquire()
    g_Res.append([threadNr, tloc.val])
    lockWr.release()
    print ("Thread: " + str(threadNr) + ", " + str(tloc.
        val))

def test(threadsNr, tnow):
  random.seed(2)
  global g_Res
  g_Res = []
  threadsArr = []
  c = g_MaxPerThread

  if (tnow == 1):
    c = c * threadsNr - threadsNr + 1
  for i in range(0,tnow):
    t = threading.Thread(target=threadFun, args=[i, c])
```

```
    threadsArr.append(t)

  for t in threadsArr:
    t.start()

  for t in threadsArr:
    t.join()

def printRes(res):
  for item in res:
    print (str(item[0]) + ", " + str(item[1]))

test(g_Threads, 1)
res1 = g_Res
print ("-----------------------------------------")
test(g_Threads, g_Threads)
res2 = g_Res
print ("-----------------------------------------")
printRes(res1)
print ("-----------------------------------------")
printRes(res2)
```

Z výpisu získaného spuštěním výše uvedeného programu plyne, že výstupní sekvence získaná jedním vláknem se při spuštění generátoru s totožným semínkem shoduje se sekvencí získanou několika vlákny. Porovnání jsem prováděl z obsahu jednotlivých listů `res1` a `res2`. Při porovnání výpisů produkovaných přímo jednotlivými vlákny bylo pořadí některých položek na výstupu přehozené, což bylo nejspíše způsobené náhodnou volbou pořadí jednotlivý vláken při zápisu na standardní programový výstup.

2.2.2 Komunikační kanály

Při umístění sdílené sekvence na server a požadavku na obousměrně generovanou komunikaci mezi klienty a serverem, tedy odpověď serveru je odeslána na základě příchozího požadavku od klienta, je nutné vytvořit speciální komunikační kanál, kterým bude server přikazovat jednotlivým klientům odeslání požadavku na určitý objekt. Tento kanál budu označovat jako *signální*. Pro samotnou generovanou komunikaci simulující provoz na aplikační úrovni se mezi serverem a klienty budou používat jiné kanály, které budu dále v textu označovat jako *datové*.

2.2.3 Režimy generování

Navrhované režimy generování jsou tři, přičemž se liší možnostmi nastavení. Jedná se o režim přeposílání paketů nalezených v pcap souboru, o režim posí-

lání souborů nalezených v pcap souboru a o režim stochastického generování syntetického provozu.

2.2.3.1 Přeposílání paketů

Přeposílání paketů funguje na principu výběru paketů z původního pcap souboru a jejich odeslání na určené síťové rozhraní. Nepočítám zde s možností obousměrné komunikace, nicméně naopak zde chci umožnit přepisování zdrojových i cílových MAC a IP adres na libovolné hodnoty volené uživatelem. Zároveň v tomto režimu nebude probíhat generování v reálném čase, ale jednotlivé pakety se nejdříve extrahují, upraví se jejich parametry a až následně se spustí jejich odesílání na síťové rozhraní. Tento postup navrhuji kvůli předpokládaným zpožděním způsobených vyhledáváním a úpravami jednotlivých paketů v původním pcap souboru. Celkový počet vybraných hlavních objektů, čímž je implicitně určen i počet opakování cyklů, ve kterých se jednotlivé objekty vyberou a upraví jejich časování, bude volen uživatelem jako vstupní parametr programu.

Tento režim pracuje se seznamy dostupných hlavních a vestavěných objektů, přičemž pro každý objekt se pomocí zdrojové a cílové IP adresy, zdrojového a cílového portu a času začátku a konce přenosu přesně určí pakety, které přenos daného objektu reprezentují v původním pcap souboru. Každý objekt v seznamu je definován pomocí parametrů uvedených v tabulce 2.6. Jednotlivé vestavěné objekty v tomto režimu nejsou rozlišeny z hlediska svého typu. Jako úložiště pro seznamy objektů spolu s jejich parametry se využijí soubory v csv formátu.

V tomto režimu je respektován počet vestavěných objektů příslušejících vybranému hlavnímu objektu a v okamžiku, kdy je vybrán řádek souboru hlavních objektů, je tedy určen i počet vestavěných objektů, které budou spolu s ním odeslány. Volbou uživatele je pouze to, zda posílané vestavěné objekty budou stejné, které byly s hlavním objektem posílány původně a zachyceny do pcap souboru, nebo zda se náhodně vyberou ze seznamu dostupných vestavěných objektů. V prvním případě zůstává zachováno i časování rozestupů odesílání jednotlivých objektů, ve druhém případě je časování upraveno dle hodnot náhodně vygenerovaných dle vytvořeného modelu.

2.2.3.2 Posílání souborů

V režimu posílání souborů jsou nejdříve extrahovány HTTP objekty z pcap souboru a poté jsou posílány serverem jednotlivým klientům jako obsah payloadu aplikační vrstvy paketů. Generovaná komunikace mezi serverem a klienty probíhá oboustranně s využitím datových kanálů popsaných v sekci 2.2.2, přičemž zdrojové a cílové IP i MAC adresy uvedené v posílaných paketech odpovídají adresám síťových rozhraní, ze kterých komunikace probíhá.

Tabulka 2.6: Hodnoty na řádku csv souboru definujícího jednotlivé objekty pcap souboru.

Jméno parametru	Význam
id	Pořadové číslo objektu vyjadřující jako kolikátý byl daný objekt zpracováván
clientAddr	IP adresa klienta (uzlu, který odeslal HTTP požadavek)
clientPort	port klienta
serverAddr	IP adresa serveru (uzlu, který přijal HTTP požadavek)
serverPort	port serveru
serverRespStartTime	časová značka prvního paketu příslušejícího odpovědi serveru (tj. přenosu daného objektu)
serverRespEndTime	časová značka posledního paketu příslušejícího odpovědi serveru (tj. přenosu daného objektu)
var	V souboru hlavních objektů: počet vestavěných objektů příslušejících tomuto hlavnímu objektu. V souboru vestavěných objektů: identifikátor hlavního objektu, ke kterému vestavěný objekt přísluší.

Navrhovaný model je zde využit s výjimkou velikostí jednotlivých objektů. Objekty extrahované z pcap souboru jsou rozděleny do adresářů dle svých typů (hlavní, obrázky, atd.) a při generování je z určené složky náhodně vybrán jeden soubor. Tento soubor je odeslán ke klientovi celý, tedy celková velikost payloadu aplikační vrstvy při odesílání vybraného souboru je rovna velikosti tohoto souboru.

Na serveru běží pro každého klienta nezávislé vlákno, které obsluhuje komunikaci mezi klientem a serverem. Každé takové vlákno od řídícího vlákna procesu získává náhodně generované hodnoty podle jednotlivých rozdělení. Tyto hodnoty určují časování mezi odesílanými objekty a počty jednotlivých typů vestavěných objektů. Jednotlivé HTTP objekty jsou generovány v následujícím pořadí:

1. HTML objekt,
2. žádný až několik css objektů,
3. žádný až několik javascriptových objektů,
4. žádný až několik obrázků,
5. žádný až několik ostatních vestavěných objektů.

Toto pořadí jsem určil na základě analýzy zdrojových kódů několika náhodně vybraných webový stránek, ve kterých jsou zpravidla v hlavičce od-

kazovány css soubory spolu s javascriptovými soubory, v těle stránky pak javascriptové kódy, obrázky a ostatní objekty. Pro další rozšíření by stála za zvážení implementace náhodného pořadí odesílání jednotlivých souborů či zahrnutí tohoto pořadí do modelu.

2.2.3.3 Stochastické generování

Stochastické generování se z velké části shoduje s návrhem generování provozu na úrovni posílání souborů popsaným v předchozí sekci 2.2.3.2, přestože původním záměrem pro tuto část bylo využití generátorů D-ITG popsaného v sekci 1.3.2.6. Několik jeho nedostatků toto využití však znemožnilo, což je detailněji popsáno v bezprostředně následujícím textu, za kterým následuje popis navrženého způsobu generování v tomto režimu.

- Vzhledem ke své struktuře, kde jsou jednoznačně odděleny části vysílače (ITGSend) a přijímače (ITGRecv), D-ITG neumožňoval posílat obousměrný síťový provoz. Tento problém znemožnil posílat požadavky z různých portů klienta na jeden port serveru a odpovědi opačným směrem.

- Možnosti rozlišení nastavení objemu generovaného provozu jsou omezeny na celé kilobajty. Poté, co jsem program D-ITG modifikoval přidáním možnosti zvolit požadovaný objem provozu na úrovni bajtů, objevil se problém s volbou velikosti payloadu v jednotlivých paketech. Program totiž při generování payloadu paketů (ať už konstantní nebo náhodné velikosti) nekontroluje, zda velikost posledního vygenerovaného payloadu již v součtu nepřesáhla celkový požadovaný objem dat. Tím velmi často dochází k výraznému překročení maximálního vygenerovaného objemu dat, které se mi nepodařilo zcela eliminovat ani kombinací parametrů počtu paketů a velikosti payloadu zadaných při spouštění programu.

- Komunikace mezi ITGSend a ITGRecv čas od času z neznámých důvodů selhala, nicméně oba procesy zůstaly běžet. Z mého generátoru byly oba volány pomocí příkazu `subprocess.Popen`, pro který jsem nenašel přímočarý způsob, jak detekovat standardní nebo chybový výstup podprocesu před jeho ukončením. Tím docházelo k problému detekce této chyby a praktické nepoužitelnosti tohoto generátoru pro účely práce.

Jedinou návrhovou změnou tohoto režimu oproti návrhu posílání souborů (2.2.3.2) je určení celkové velikosti posílaného aplikačního payloadu a možnost volby obsahu tohoto payloadu. Payload aplikační vrstvy paketů posílaných na síťové médium v tomto režimu může být buď zcela náhodně generovaný, nebo čtený ze souboru určeného typu. Čtení ze souboru určeného typu představuje např. situaci, kdy při generování přenosu javascriptového objektu je aplikační payload čten z javascriptového souboru, který byl extrahován z poskytnutého pcap souboru.

Postup volby souboru a jeho poslání se od předchozího režimu liší. V režimu posílání souborů se nejdříve vybral daný soubor, který byl následně odeslán na síť. Zde je z daného statistického rozdělení vygenerována hodnota odpovídající velikosti daného typu souboru, což znamená, že na síť bude odeslán payload právě této velikosti bez ohledu na to, kterým z výše uvedených způsobů je vytvářen.

Při generování náhodného payloadu nejsou z pcap souboru extrahovány soubory. Ve zbytku navrhovaného postupu generování se tento režim shoduje s režimem posílání souborů.

Kapitola 3

Realizace

Výsledný software se skládá ze dvou samostatných částí, které spolu komunikují. První částí je zpracování pcap souboru, přičemž výstupem této části jsou, v závislosti na vstupních parametrech, soubory s množinami hodnot sloužících k výpočtu parametrů modelu navrženého v sekci 2.1.1 nebo soubory se seznamy nalezených hlavních a vestavěných objektů využitých v režimu přeposílání paketů. Druhou částí je generátor, který v závislosti na zvoleném operačním režimu využívá množiny dat získaných z první části k tvorbě modelu a jeho následnému využití pro tvorbu a odesílání generovaného provozu. Využití IPFIX toků v této fázi implementováno nebylo.

Část zpracování pcap souboru byla implementována v C++, generátor provozu v Pythonu.

V této kapitole popisuji postup a řešené obtíže v průběhu implementace. Uživatelské ovládání obou částí softwaru je popsáno v uživatelské příručce v příloze B, architektura aplikací v dokumentaci na přiloženém médiu.

3.1 Zpracování pcap souboru

Statistické informace o síťovém provozu se získávají z dat zachycených v pcap souborech. Pro zpracování pcap souborů jsem implementoval PcapAnalyzer, přičemž jeho výstupem jsou v závislosti na parametrech zadaných uživatelem:

- konkrétní hodnoty zachyceného síťového provozu, např. velikosti jednotlivých HTTP požadavků, zapisovány na standardní programový výstup nebo do souborů,

- dva csv soubory (jeden pro hlavní a jeden pro vestavěné objekty) obsahující záznamy pro identifikaci objektů obsažených v pcap souboru spolu s informacemi pro získání příslušných paketů.

3.1.1 Zjišťované parametry provozu

Zjišťované hodnoty odpovídají modelovaným parametrům provozu uvedeným v návrhu, jedná se o:

- velikosti HTTP požadavků,
- velikosti HTTP hlavních objektů,
- velikosti javascriptových, css, obrázkových a ostatních HTTP vestavěných objektů,
- počty javascriptových, css, obrázkových a ostatních HTTP vestavěných objektů obsažených v hlavním objektu,
- dobu parsování,
- dobu mezi počátky příchodů jednotlivých HTTP vestavěných objektů a
- dobu prohlížení stránky.

Definice jednotlivých parametrů jsou uvedeny v tabulce 2.2. Z pohledu implementace na úrovni paketů je nutné specifikovat, co se rozumí začátkem a koncem přenosu souboru, které jsou užity v definicích pro parametry časů. Začátek přenosu souboru definuji jako časovou značku zachycení prvního rámce příslušejícího přenosu daného objektu. Konec přenosu souboru definuji jako časovou značku zachycení posledního rámce příslušejícího přenosu daného objektu. Tyto časové značky jsou v pcap souboru uvedeny pod označením `frame.time_epoch`.

3.1.2 Popis implementace a zjištění hodnot pro výpočet parametrů

Tento software sekvenčně prochází celý pcap soubor a analyzovaný síťový provoz rozlišuje na dvě kategorie, na provoz směrem k serveru a provoz směrem od serveru. Toto rozlišení se provádí, i přes nevýhody uvedené v sekci 1.2.4, na základě komunikujících portů. Provoz směrem k serveru je charakterizován cílovým portem 80, opačný provoz pak má port 80 jako zdrojový. Zároveň zde díky tomuto mechanismu není rozlišeno více případných klientů. Analyzátor tedy dobře poslouží, pokud chceme získat data o surfování jednoho uživatele na více webových stránek, ale pro soubor obsahující provoz více klientů poskytuje zkreslené výsledky.

3.1.2.1 Použité knihovny a struktury

Implementace zpracování pcap souboru spoléhá na externí knihovnu Libtins [74], což je multiplatformní C++ knihovna vytvořená pro snadnější zachytávání a modifikaci paketů. Svými možnostmi lze tuto knihovnu přirovnat např. ke knihovně Scapy [75] v Pythonu.

Hlavní třídou celého programu je třída CPcapReader, která se stará o čtení pcap souboru, zpracování jednotlivých záznamů a s tím souvisejícími manipulacemi s třídami CTcpStreamInfo, např. jejich uzavírání či řazení dle implementovaných požadavků. Třída CTcpStreamInfo slouží k uchování informací o HTTP požadavku a jemu příslušející HTTP odpovědi, k čemuž využívá třídy CHttpRequest a CHttpResponse. Získané hodnoty sloužící jako podklad pro model zpracovává a vypisuje třída CResults. Úplná dokumentace vytvořených tříd a funkcí je na přiloženém médiu.

3.1.2.2 Provoz směrem od klienta

V provozu směrem k serveru, na port 80, se vyskytují HTTP požadavky a potvrzení přijatých rámců v rámci komunikace mezi klientem a serverem. Z hlediska tvorby modelu jsou potvrzovací rámce nepodstatné, a tudíž se ve zpracování ignorují. Zpracovávají se pouze HTTP požadavky, z nichž je v modelu zahrnuta jejich velikost. Velikost požadavků je určena jako součet velikostí payloadu segmentů transportní vrstvy, které přenáší daný požadavek. V této fázi implementace jsou zpracovány pouze požadavky provedené metodou GET. Požadavky provedené pomocí jiné metody jsou ignorovány.

HTTP požadavek může žádat o HTTP hlavní nebo vestavěný objekt. To však nelze s jistotou určit již z dotazu, protože soubory v URL dotazu mnohdy nemají žádnou příponu a jejich typ je určen na základě obsahu Content-Type položky v hlavičce HTTP odpovědi. Z toho důvodu je z těla požadavku kromě požadovaného URL extrahován také obsah položky Referer pro zjištění případné příslušnosti vestavěného objektu k hlavnímu objektu. Ke zkoumání této příslušnosti dojde, když je objekt v HTTP odpovědi určen jako vestavěný.

3.1.2.3 Provoz směrem od serveru

Detekcí HTTP požadavku je vytvořena instance třídy TcpStreamInfo zpracovávající komunikaci zahájenou odesláním daného požadavku. Všechny tyto instance jsou otevřené až do detekce dalšího HTTP požadavku, který odchází ze stejného portu. Otevřenost instance třídy TcpStreamInfo znamená, že všechny zpracovávané komunikační rámce mezi odpovídajícími sokety jsou do tohoto komunikačního proudu zařazeny. Vzhledem k ignorovaným potvrzením zasílaných TCP protokolem se jedná pouze o rámce obsahující odpověď na daný požadavek.

HTTP odpověď je detekována na základě HTTP hlavičky, konkrétně na základě přítomnosti textového řetězce *HTTP/1.1* v payloadu segmentu transportní vrstvy. Následně je zjištěn kód HTTP odpovědi, přičemž zpracovávány jsou pouze odpovědi s kódem *200 OK*, které mají vliv na přenášený síťový provoz z hlediska modelovaných parametrů provozu. Přenášené objekty se kvůli své velikosti často nevejdou do jednoho segmentu. Určování velikosti jednotlivých objektů tedy probíhá na základě sčítání velikost payloadu aplikační

vrstvy segmentů příslušejících do jednoho komunikačního proudu. Do této velikosti se nezapočítává velikost HTTP hlavičky odpovědi, která je obsažena v prvním segmentu odpovědi.

Typy objektů jsou určeny podle obsahu položky Content-Type v HTTP hlavičce odpovědi. Tato položka obsahuje MIME typ objektu, přičemž nalezené signatury jsou na typy vestavěných objektů mapovány podle tabulky 2.1.

Po přečtení celého pcap souboru dojde k závěrečnému zpracování programových struktur. Nejdříve jsou struktury hlavních objektů uspořádány podle času svého začátku. Následně se určí příslušnost jednotlivých vestavěných objektů k hlavním objektům. Pokud je požadován zápis hodnot pro tvorbu modelu, ať už na standardní programový výstup nebo do souborů, spočítají se nakonec hodnoty požadovaných časových rozestupů mezi jednotlivými objekty.

Jak jsem uvedl v sekci návrhu, u času prohlížení stránky zjišťovaného uvedeným způsobem se může stát, že délka intervalu je záporná. Toto je způsobeno tím, že HTTP požadavek na hlavní objekt je odeslán ještě před kompletním načtením všech vestavěných objektů příslušejících předchozímu hlavnímu objektu. V takovém případě pak program pracuje s nulovou délkou časového intervalu prohlížení stránky. Dalším možným problémem je, že v závislosti na prohlížeči se mohou vestavěné objekty začít stahovat ještě před ukončením stahování hlavního objektu, což dle použité definice způsobí zápornou délku časového intervalu pro dobu parsování. V takovém případě se rovněž pracuje s nulovou délkou tohoto intervalu.

3.1.2.4 Výstup programu

V závislosti na zadaných vstupních parametrech se liší výstup tohoto programu. Pokud jsou požadovány hodnoty pro výpočet parametrů modelu, jsou tyto hodnoty spolu s pevně definovaným jménem parametru provozu vypsány na standardní programový výstup. Hodnoty velikostí jednotlivých objektů jsou uváděny v bajtech, časy v mikrosekundách. Příklad takového výpisu pro časy parsování v mikrosekundách je zobrazen v ukázce níže. Při této volbě jsou zároveň data zapsána do souborů, nazvanými jmény modelovaných parametrů provozu, pro další zpracování generátorem.

Listing 3.1: "Ukázka výpisu pcapAnalyzeru. Jedná se o hodnoty sloužící k výpočtu parametrů modelu pro čas parsování."

```
parsTimeArr=[150453, 121139, 452605, 69065, 91490,
    412842, 411984, 246408, 70071, 118376, 67411, 97561,
    87241, 64386, 42507, 196397, 159104, 50752, 81521,
    106883, 63937, 131917, 110911, 133016, 54295, 159318,
     144789, 352273166, 412196, 216957, 166671, 252855,
    101305, 289626, 78744, 169662, 145814, 298369,
    16778623, 142393, 133693, 173439, 209570, 262813];
```

Druhou možností je vytvoření seznamů hlavních a vestavěných objektů. V takovém případě jsou vytvořeny dva csv soubory obsahující daný typ objektů (hlavní nebo vestavěné) vzestupně seřazené podle času začátku jejich přenosu. Jména a výstupní adresáře pro tyto soubory jsou zadány jako vstupní parametr při spouštění programu.

3.2 Generování síťového provozu

Vyvinutý nástroj umožňuje vytvářet a odesílat síťový provoz ve třech různých režimech. Prvním z nich je přeposílání paketů poskytnutých v pcap souboru, druhým je posílání HTTP objektů extrahovaných z poskytnutého pcap souboru a třetím režimem je stochastické generování. Všechny tři režimy do určité míry využívají statistický model vytvořený na základě výstupu programu zpracovávající pcap soubor popsaného v sekci 3.1.

Veškerou funkcionalitu kolem výpočtu parametrů modelu, generování a přijímání síťového provozu zajišťuje skript `generator.py`. Tento skript nemá implementovány žádné třídy, spuštěn může být v operačním režimu klienta nebo serveru, stejně tak je při spouštění definován režim generování. Instrukce pro práci s tímto skriptem jsou sepsány v uživatelské příručce B. Veškeré dočasné soubory jsou ukládány do adresáře `/tmp/netgen`, zachování těchto souborů i po ukončení běhu programu lze ovlivnit volbou příslušného parametru zadaného při spouštění programu. Při určité kombinaci vstupních parametrů může být vyžadováno spuštění s administrátorskými právy, např. když má webový server pracovat na portu 80.

3.2.1 Tvorba modelu

Data pro tvorbu modelu jsou zajištěna voláním podprocesu spouštějícího PcapAnalyzer a zpracováním dat z ním vytvořených souborů. Tuto funkcionalitu zajišťuje funkce `getModelRawData()`.

Data jsou ze souborů načtena, z nich jsou spočítány jejich střední hodnoty a rozptyly a z nich pak parametry příslušných statistických rozdělení. V současné implementaci je pro každou modelovanou náhodnou veličinu pevně určeno, jaké statistické rozdělení jí přísluší. Pro další rozšíření se nabízí výpočet několika rozdělení a následné vybrání toho nejlépe se hodícího. Parametry jednotlivých rozdělení jsou počítány podle postupů uvedených v sekci 2.1.2.

3.2.2 Režim přeposílání paketů

Prvním z režimů generování je režim přeposílání paketů, který umožňuje přeposílání paketů čtených ze vstupního pcap souboru. Přeposílané pakety jsou takové, které přenášely nyní vybraný objekt. Přeposílají se vždy hlavní objekty a k nim příslušející počet vestavěných objektů. V závislosti na hodnotě

příslušného přepínače při spouštění programu mohou být přeposílané vestavěné objekty buď náhodně vybrány, nebo původně patřit vybranému hlavnímu objektu. V tomto režimu se neprovádí generování v reálném čase, ale všechny pakety jsou nejdříve extrahovány z pcap souboru, modifikovány dle zadaných požadavků a následně sloučeny do jednoho souboru, který je následně posílán na vybrané síťové rozhraní. Jednotlivé fáze jsou detailněji popsány v následujících podsekcích.

Tento režim jako jediný kromě dat pro tvorbu modelu požaduje při zpracování pcap souboru vytvoření seznamů v něm obsažených objektů. Toto je zajištěno voláním programu zpracovávajícího pcap soubor s parametry uvedenými v následujícím výpisu.

Listing 3.2: Funkce sloužící pro získání dat pro tvorbu modelu spolu se seznamy objektů.

```
cmd = subprocess.Popen(["pcapAnalyzer", "-i", g_args['
    inputPcapPath'], "-f", "-m", "-om",
    mainObjectsFileName,"-oe", inlineObjectsFileName],
    stdout=subprocess.PIPE, stderr=subprocess.PIPE)
```

3.2.2.1 Generování a odesílání provozu

Generování v reálném čase se v tomto režimu neprovádí kvůli velkému počtu volání externích programů, které je časově náročné. Patrně by bylo možné zpracování vybraných paketů zrychlit originální implementací potřebných funkcí, nicméně vzhledem k dostupnosti hotových nástrojů mi to přišlo zbytečné. Nevýhodou je bohužel nutnost instalace těchto externích nástrojů.

Csv soubor obsahující seznam hlavních objektů označuji v následujícím textu jako *soubor hlavních objektů*, csv soubor obsahující seznam vestavěných objektů označuji jako *soubor vestavěných objektů*. Příprava výsledného pcap souboru, který bude přehrán na síťové rozhraní, probíhá v těchto krocích:

1. Ze souboru hlavních objektů je náhodně vybrán řádek určující jeden hlavní objekt.

2. Na základě IP adres, portů a časových značek ve vybraném řádku jsou ze vstupního pcap souboru vybrány pakety HTTP odpovědi obsahující tento objekt. Pak jsou uloženy do nového pcap souboru, který obsahuje pouze pakety tohoto objektu. Tento soubor dále označuji jako *pcap soubor hlavního objektu*.

3. V závislosti na tom, zda se jedná o první nebo několikátý vybraný hlavní objekt, je u paketů pcap souboru hlavního objektu upraveno časování s ohledem na generovanou dobu prohlížení stránky uživatelem. Pravidla pro úpravu časování jsou podrobně popsána v sekci 3.2.2.2. Časování se

upravuje pomocí volání podprogramu `editcap` s příslušnými parametry. V programu tuto úlohu zajišťuje funkce `editPcapTiming`.

4. Z řádku vybraného v kroku 1 je extrahována hodnota počtu vestavěných objektů, které tento hlavní objekt původně obsahoval. Tato hodnota určuje množství řádků, které se vyberou ze souboru vestavěných objektů.

5. Způsob výběru řádků v souboru vestavěných objektů je závislý na zvolené možnosti příslušnosti vestavěných objektů k hlavnímu objektu.

   Pokud byla zvolena možnost přeposílat hlavní objekt spolu se svými původními vestavěnými objekty, jsou tyto objekty hledány v souboru vestavěných objektů a následně extrahovány z výchozího pcap souboru. Pakety každého vestavěného objektu jsou extrahovány do individuálního pcap souboru. Pokud hlavní objekt není vybrán jako první z hlavních objektů, je v nově vzniklém pcap souboru upraveno časování s ohledem na generovanou dobu prohlížení předchozí stránky uživatelem. Tato doba prohlížení je náhodně vygenerovaná podle modelu.

   Pokud mají být vestavěné objekty náhodně vybrané, jsou náhodně vybírané řádky ze souboru vestavěných objektů. Následně je upraveno časování všech vestavěných objektů s ohledem na dobu parsování mezi časem posledního paketu hlavního objektu a časem prvního paketu prvního vestavěného objektu a dobou mezi příchody každých dvou po sobě jdoucích vestavěných objektů odpovídající časovým značkám jejich prvních paketů. Tyto hodnoty jsou náhodně vygenerované z modelu.

6. Po výběru všech vestavěných objektů je z modelu vygenerována doba prohlížení stránky uživatelem.

7. V závislosti na tom, kolikátý hlavní objekt byl generován se buď pokračuje dalším krokem, nebo se vše opakuje od kroku 1. Počet vybíraných hlavních objektů a tím daný počet opakování celého cyklu od začátku po tento bod je dán zadanou hodnotou u příslušného přepínače při spouštění tohoto programu.

8. Po výběru všech posílaných objektů jsou dočasné pcap soubory sjednoceny voláním podprogramu `mergecap` do jednoho.

9. Soubor vzniklý v předchozím kroku je předán podprogramu `tcpprep`, čímž je vytvořen cache soubor identifikující jednotlivé komunikující strany buď jako server, nebo jako klient.

10. Cache soubor vzniklý v předchozím kroku slouží jako podklad pro přepsání adres v dočasném pcap souboru, které jsou přepisovány s požadavky uvedenými u příslušných parametrů při spouštění programu. Adresy jsou přepsány voláním podprogramu `tcprewrite`, jehož výstupem je finální pcap soubor.

11. Finální pcap soubor z předchozího kroku je přehrán na zvolené síťové rozhraní voláním podprogramu `tcpreplay`.

Přestože zde jsou přeposílány pakety na základě jejich příslušnosti k určitému objektu HTTP provozu, není zaručeno, že se složením payloadu aplikační vrstvy z jednotlivých paketů získá původní objekt. Toto je způsobeno určením vybíraných paketů jako všech paketů, které byly zachyceny v určitém časovém intervalu mezi sledovaným zdrojem a cílem. V okamžiku, kdy byly některé pakety odeslány vícekrát, např. vlivem chování TCP protokolu, budou i zde tyto opakované pakety přítomny.

3.2.2.2 Modifikace časů

Podstatnou částí tohoto režimu generování je úprava časování jednotlivých paketů vybíraných objektů. Pro hlavní objekty se řídí těmito pravidly:

- Pokud je vybíraný hlavní objekt první vybíraný, jeho časování se žádným způsobem neupravuje.
- Pokud je vybíraný hlavní objekt několikátý vybíraný, časování jeho paketů se upravuje tak, aby jeho první paket následoval za posledním paketem ze všech paketů příslušejících vestavěným objektům předchozího hlavního objektu o vygenerovanou hodnotu odpovídající času prohlížení stránky uživatelem.

Pro vestavěné objekty se tato úprava liší dle způsobu jejich výběru. Pokud jsou vestavěné objekty vybrány na základě původní příslušnosti hlavnímu objektu, jednotlivé časy se modifikují o stejnou hodnotu jako časy hlavního objektu. Pokud však jsou vestavěné objekty vybrány ze seznamu dostupných objektů nezávisle na vybraném hlavním objektu, celý proces se řídí následujícími pravidly:

- Časy paketů prvního vestavěného objektu jsou upraveny tak, aby rozdíl mezi časem posledního paketu hlavního objektu a času prvního paketu prvního vestavěného objektu byl roven hodnotě vygenerované z rozdělení modelované náhodné veličiny pro *čas parsování*.
- Časy paketů každého následujícího vestavěného objektu jsou upraveny tak, aby rozdíl mezi časem prvního paketu aktuálně vybíraného vestavěného objektu a časem prvního paketu předchozího vestavěného objektu byl roven hodnotě vygenerované ze statistického rozdělení modelované náhodné veličiny pro *IAT vestavěných objektů*.

3.2.2.3 Příjem generovaného provozu

Vzhledem ke způsobu generování posílaných objektů se na straně příjemce žádným způsobem neověřuje, zda tam generovaný provoz skutečně dorazil. Také

nejsou přepisovány porty v síťovém provozu ze vstupního pcap souboru. Pro příjem generovaného provozu by to znamenalo naslouchat na všech portech, na které provoz směřuje. Požadavky na soubory zde odesílány nejsou a klient tedy žádný síťový provoz neprodukuje.

V tomto režimu tedy není nutné na straně klienta spouštět žádný přijímač, nicméně možné to je, což v praxi znamená spuštění podprocesu `nc`, který poslouchá na přiděleném síťovém rozhraní a zahazuje veškerý přijatý provoz přesměrováním do `/dev/null`. Tato možnost se dá využít, pokud původně zachycený provoz směřuje na jeden klientský port.

3.2.3 Režim posílání souborů

Generování síťového provozu v tomto režimu se skládá z několika po sobě jdoucích kroků, kterými jsou tvorba statistického modelu a extrakce objektů ze vstupního pcap souboru, vytvoření signálního kanálu pro synchronizaci komunikace mezi klientem a serverem, vytvoření soketu představujícího komunikační rozhraní webového serveru a následné generování komunikace mezi klientem a serverem. Jednotlivé kroky, kromě tvorby statistického modelu popsané v sekci 3.2.1, jsou popsány v následujícím textu.

3.2.3.1 Využití vláken

Jak jsem uvedl v návrhu, tento režim umí generovat provoz mezi serverem a libovolným množstvím připojených klientů, kteří využívají sdílenou náhodnou sekvenci. Pro paralelní obsluhu mnoha možných příchozích požadavků jsem využil mechanismu vláken jak na straně jednotlivých klientů, tak na straně serveru. Ověření sdílení náhodné sekvence mezi jednotlivá výpočetní vlákna je popsáno v sekci 2.2.1.

Na straně serveru lze rozlišit tři typy vláken, pro větší přehlednost v následujícím textu je budu označovat jako typy S1, S2 a S3 s následujícími definicemi:

1. Typ S1 je hlavní programové vlákno serveru, které zodpovídá za naslouchání na uživatelem zvoleném portu a příjem HTTP spojení generovaných klienty. Toto vlákno je v programu pouze jedno.

2. Typ S2 je vlákno, které zodpovídá za získávání náhodných hodnot z modelu a za signální komunikaci s klientem. Toto vlákno je jedno pro každého klienta.

3. Typ S3 je vlákno, které zodpovídá za vlastní generování a odesílání provozu klientovi. Toto vlákno je tvořeno pro každý odesílaný objekt každému klientovi a po odeslání objektu je zničeno.

Každému vláknu S2 je vytvořena fronta čekajících spojení `queue.Queue()`, která příslušný klient otevřel, zámek `threading.Lock()` zajišťující výlučnou

manipulaci s prvky fronty a událost `threading.Event()` synchronizující uspávání a probouzení daného vlákna při čekání na příchozí požadavek.

Aby všichni klienti mohli zasílat požadavky na stejný port webového serveru, musí obsluhu webového serveru zajišťovat pouze jedno vlákno. V mé implementaci má tuto úlohu hlavní programové vlákno (typ S1), které poslouchá na daném soketu. Po přijetí požadavku určí IP adresu odesílatele a na jejím základě přiřadí obsluhu požadavku vybranému vláknu typu S2. Omezením tohoto mechanismu je, že z jedné IP adresy se může připojit nejvýše jeden klient. Přiřazení obsluhy probíhá následovně:

1. hlavní vlákno (S1) vybere jedno z vláken (S2) a zajistí si výlučný přístup do jeho fronty požadavků,
2. hlavní vlákno (S1) umístí otevřené spojení do vybrané fronty,
3. hlavní vlákno (S1) odemkne přístup do fronty a probudí vlákno (S2) obsluhující klienta, které požadavek odebere z fronty a obslouží ho.

Z pohledu vlákna (S2) obsluhujícího komunikaci s jedním klientem probíhá výše popsaný proces následovně:

1. vlákno (S2) odešle signálním kanálem klientovi příkaz k odeslání požadavku na specifikovaný objekt a port,
2. vlákno (S2) se uspí a čeká na probuzení vláknem (S1),
3. po svém probuzení si vlákno (S2) zajistí výlučný přístup do fronty, ze které odebere příchozí spojení a předá ho nově vytvořenému vláknu (S3), které tímto spojením odešle data ke klientovi.

Na straně klienta rozlišuji dva typy vláken, které označuji jako K1 a K2 s následujícími definicemi:

1. Typ K1 je hlavní programové vlákno, které naslouchá na portu určeném pro signální kanál a zpracovává příchozí instrukce.
2. Typ K2 je vlákno vytvořené pro vygenerování a odeslání HTTP požadavku a následnou komunikaci se serverem, tj. zejména přijetí generovaného objektu.

3.2.3.2 Extrakce souborů

Pro přeposílání objektů reálně poslaných na síti je nutné získat tyto objekty z poskytnutého pcap souboru. To je v programu vyřešeno voláním podprogramu `tcpflow` s parametry uvedenými v přiloženém výpise, který v poskytnutém pcap souboru vyhledá objekty přenášené přes HTTP protokol, detekuje jejich typ a vytvoří z nich samostatné soubory, jejichž typ je identifikován

příponou. Takto vytvořené soubory jsou následně přesunuty do složek sdružujících vždy jeden typ souboru, tedy html, css, javascripty, obrázky a ostatní vestavěné objekty. Toto přesunutí se provádí kvůli rychlejšímu výběru souboru daného typu při generování.

Listing 3.3: TCPflow příkaz pro extrakci objektů z pcap souboru

```
tcpflow -e http -r cestaKeVstupnimuPcapSouboru -o
    cestaVystupnihoAdresare
```

3.2.3.3 Signální kanál

Pro řízení chování klientů a jejich synchronizaci se serverem se využívá signální kanál. Ke každému klientovi je vytvořen individuální signální kanál a komunikace v něm probíhá pouze ve směru od serveru. Tento kanál je vytvořen před samotným zahájením generování následujícím způsobem:

- Klient po svém spuštění naslouchá na soketu zadaném parametrem při spouštění programu a přijímá první příchozí TCP spojení.
- Server po svém spuštění vytvoří pro každého klienta síťový soket obsluhovaný vláknem S2, kterým se připojí k rozhraní, na kterém naslouchá klient. Toto spojení trvá po celou dobu běhu generátoru.

Signální kanál má několik účelů:

- Server ho před odesláním každého generovaného objektu využívá k tomu, aby přikázal klientovi vytvořit požadavek na daný objekt a tento požadavek zaslat na port, na kterém poslouchá webový server.
- Server pomocí něj sděluje klientovi délku časového intervalu čekání před odesláním dalšího objektu. Toto pomáhá k vyšší přehlednosti o generovaném provozu přímo za běhu, protože jednotlivé údaje jsou vypisovány na standardní výstup odděleně pro každého klienta.
- Server pomocí něj sděluje klientovi informace o počtech a typech generovaných souborů. Klient tyto informace tiskne na standardní výstup, čímž se komunikace stává pro uživatele přehlednější.
- Uzavření signálního kanálu ze strany serveru je pro klienta signál, že server ukončil generování a klient se tak může rovněž ukončit.

Signální zpráva se skládá z parametrů oddělených čárkami. První parametr určuje, o jaký druh zprávy se jedná. Druhy zpráv jsou dva:

1. Komunikační zpráva přikazuje klientovi odeslat HTTP požadavek na daný objekt. Parametry této zprávy jsou:

- typ objektu, na který má být odeslán požadavek,
- požadovaná velikost požadavku,
- port serveru, se kterým má být spojení navázáno a kam má být požadavek odeslán.

2. Informační zpráva má dva typy. V prvním informuje klienta o době čekání před odesláním dalšího objektu. Ve druhém informuje klienta o počtu souborů daného typu, které budou generovány a o pořadí aktuálně generovaného souboru. Parametrem této zprávy je přímo text, který bude vypsán na standardní programový výstup klienta.

3.2.3.4 HTTP komunikace

Na straně serveru je vytvořen soket představující rozhraní webového serveru. Server (konkrétně vlákno S1) na tomto rozhraní přijímá požadavky odeslané jednotlivými klienty. Mechanismus zpracování takto přijatých požadavků je uveden v odstavci 3.2.3.1. Tímto způsobem je zajištěna podobnost s průběhem reálného webového provozu pouze s tím rozdílem, že zde je pro každou dvojici požadavek – odpověď vytvořeno nové spojení. Nevyužívá se zde tedy perzistentní spojení, což může být námět pro rozvoj v budoucnu.

Odesílané požadavky jsou tvořeny HTTP hlavičkou požadavku prováděného metodou GET. Tělo této hlavičky je tvořeno postupným vkládáním položek hlavičky tak, aby nedošlo k překročení velikosti určené náhodně vygenerovanou hodnotou dle modelu. Vkládané položky mají fixní obsah s dvěma výjimkami, které obsahují hodnotu zvolené proměnné. První proměnnou jsou jména požadovaných souborů. Tato jména jsou náhodně generované alfanumerické řetězce délky deset až dvacet znaků, přičemž zakončeny jsou příponou typu objektu, který bude následně odeslán v odpovědi. Druhou proměnnou je cookie. Ta obsahuje náhodně generované alfanumerické řetězce takové délky, aby celková délka požadavku po vložení všech možných položek odpovídala náhodně vygenerované délce z rozdělení modelované náhodné veličiny pro *velikost požadavku*.

Před odesílané objekty ze strany serveru jsou vsazeny HTTP hlavičky odpovědi. Tyto hlavičky mají fixní podobu se dvěma proměnnými. První proměnnou je MIME typ zasílaného objektu, druhým pak jeho velikost. Přijaté objekty se na straně klienta žádným způsobem nezpracovávají. Jako možné rozšíření pro další rozvoj se nabízí např. tvorba souborů z přijatých objektů.

Velikosti payloadu jednotlivých segmentů jsou maximální (daná parametrem od uživatele) po celou dobu, pouze poslední segment obsahuje zbytek dat, která doposud nebyla odeslána. Program neověřuje, že uživatelem zadaná hodnota se do paketů skutečně vejde, a že na použitém médiu nedojde k fragmentaci. Zároveň prozatím ignoruji fakt, že většina objektů reálného provozu se posílá v komprimované podobě, zatímco generátor pouze otevírá

vybraný soubor pro binární čtení a výsledek tohoto čtení je odesílán jako aplikační payload. V obou těchto případech je také prostor pro další zpřesnění v následujícím rozvoji.

3.2.3.5 Generování provozu

Metody navazování komunikace a princip jejího fungování jsem popsal v předchozích částech textu, nyní popíšu samotný způsob generování požadavků a metodu obsluhy komunikujících vláken. Jednotlivé HTTP objekty jsou generovány v následujícím pořadí:

1. HTML objekt,
2. žádný až několik css objektů,
3. žádný až několik javascriptových objektů,
4. žádný až několik obrázků,
5. žádný až několik ostatních vestavěných objektů.

Zatímco hlavní objekt je vždy odesílán právě jeden, počty jednotlivých vestavěných objektů jsou generované ze statistického rozdělení příslušejícího konkrétnímu typu vestavěných objektů dle modelu. Generování komunikace serveru s jedním klientem probíhá v následujících krocích:

1. Server se rozhodne odeslat hlavní (HTML) soubor a náhodně jeden vybere ze složky dostupných HTML objektů.
2. Odešle signálním kanálem (komunikační) zprávu klientovi, ve které přikazuje klientovi zeptat se na HTML objekt.
3. Klient na základě informace ze signální zprávy vytváří vlákno (K2), které vytváří spojení a odesílá požadavek na rozhraní představující webový server, následně vyčkává přijetí souboru.
4. Po přijetí spojení na straně webového serveru (vláknem S1) je spojení předáno příslušnému vláknu (S2) obsluhujícímu daného klienta pomocí mechanismu popsaného v sekci 3.2.3.1.
5. Serverové vlákno (S2) obsluhující daného klienta vytvoří nové vlákno (S3) speciálně pro přenos určeného souboru a to pak odesílá daný soubor ke klientovi.
6. Po dokončení přenosu souboru je komunikační spojení mezi webovým serverem a klientem uzavřeno a na straně serveru je vygenerována hodnota představující čas parsování, na kterou je celé vlákno (S2) obsluhující daného klienta uspáno.

7. Po probuzení vlákna (S2) je vygenerována hodnota představující počet vestavěných souborů první skupiny, které mají být odeslány.

8. Z příslušné složky vestavěných souborů je jeden náhodně vybrán a následně probíhá komunikace mezi klientem a serverem stejným způsobem jako při posílání HTML souboru od kroku 1 až do kroku 4.

9. Ihned po vytvoření vlákna (S3) odesílajícího soubor a jeho spuštění je vygenerována hodnota z rozdělení modelované náhodné veličiny pro *IAT vestavěných objektů* a na tuto dobu je dané vlákno S2 uspáno. Toto se neaplikuje, pokud je odesílán poslední vestavěný objekt příslušející stejnému hlavnímu objektu.

10. Po probuzení se pokračuje stejným způsobem pro všechny další vestavěné objekty stejné skupiny objektů a následně všemi dalšími skupinami, dokud nejsou všechny vestavěné soubory odeslány.

11. Po odeslání posledního vestavěného souboru se čeká na dokončení činnosti všech vláken.

12. Po dokončení činnosti všech vláken je vygenerována hodnota z rozdělení náhodné veličiny pro *čas prohlížení*. Po tuto dobu je celé vlákno obsluhující daného klienta uspáno a po probuzení pokračuje od začátku dalším HTML objektem.

Výše uvedený postup běží paralelně pro všechna vlákna (S2) obsluhující jednotlivé klienty do doby, než je rozhodnuto o ukončení činnosti generátoru. Metody ukončování činnosti jsou popsány v sekci 3.2.3.6.

3.2.3.6 Ukončování programu

Možnosti ukončení programu jsou dvě:

1. Při spouštění generátoru je nutné zadat jako vstupní parametr časový limit (v počtu vteřin), po jehož uplynutí se generátor začne vypínat. Zároveň je při začátku generování uložen aktuální čas a proti této hodnotě je v daných místech kódu porovnáván aktuální čas na podmínku ukončení.

2. Program může být kdykoliv v průběhu generování požádán o ukončení pomocí zaslání signálu SIGINT stiskem kláves Ctrl+C. Po přijetí tohoto signálu je v programu nastaven interní příznak, aby se program začal ukončovat.

Obě výše uvedené podmínky jsou ověřovány ve funkci `continueRunning()` uvedené níže, která určuje, zda se má pokračovat v generování nebo program ukončit. Tato funkce je volána vlákny S2 před zahájením odesílání vybraného

objektu, tj. před spuštěním vlákna S3. Pokud podmínka není splněna, objekt se již neodesílá a pouze se čeká na doběhnutí již spuštěných vláken S3. Teprve po jejich ukončení končí také vlákno S2. Hlavní vlákno S1 čeká na dokončení běhu všech S2 vláken a teprve pak se ukončuje. V závislosti na vstupních parametrech může před ukončením ještě provádět určité výpočetní úkony, jako např. odstranění dočasných souborů.

Při implementaci se vyskytl drobný problém s ukončováním hlavního vlákna S1, které naslouchá příchozím požadavkům. V okamžiku, kdy se vlákna S2 ukončí, uzavřou se jejich signální kanály, na základě čehož se ukončí také příslušný klient. Po ukončení všech klientů již žádné požadavky na server nepřijdou, a proto bylo nutné nastavit nejvyšší dobu, po jejímž uplynutí přestane vlákno S1 na požadavky čekat a zkontroluje, zda celý program není ve fázi ukončování. Tato hodnota je opětovně nastavena po přijetí každého příchozího požadavku. Původně jsem ji nastavil jako rozdíl požadované doby běhu programu a doby, po kterou již program běžel. Toto však způsobovalo zbytečné čekání při vynuceném ukončení ze strany uživatele. Nakonec jsem jako přijatelnou dobu zvolil deset vteřin, po jejichž uplynutí se provede kontrola, zda se program má ukončit. Pokud tato podmínka není splněna, vlákno dále naslouchá na rozhraní a po deseti vteřinách uvedený postup opakuje.

Listing 3.4: Funkce ověřující, zda se má pokračovat v běhu programu, nebo zda se má program ukončit

```
def continueRunning():
  if (time.time() - g_startTime < g_args['runTime']) and
      g_StartTermination != True:
    return True
  return False
```

3.2.4 Režim stochastického generování

Stochastické generování se z velké části shoduje s generováním provozu na úrovni přeposílání souborů popsaným v sekci 3.2.3. Přestože původním záměrem pro tuto část bylo využití generátorů D-ITG, několik jeho nedostatků toto využití však znemožnilo, což je detailněji popsáno v sekci 2.2.3.3.

Payload odesílaného provozu v tomto režimu může být generován náhodně nebo čten z náhodně vybraného souboru stejného typu, jakého typu je generovaný a posílaný objekt. Možnost načítat do payloadu reálný obsah zvyšuje věrohodnost síťového provozu. Drobnou nevýhodou je, že zatímco v reálném provozu se většina objektů posílá v komprimované podobě, soubory zde použité jsou odesílány v podobě nekomprimované. V okamžiku, kdy by se model rozšířil o náhodnou veličinu modelující počet souborů přenesených komprimovaně a program by se rozšířil o komprimaci náhodně vybraného souboru, byl by to další posun v generování co nejvěrohodnějšího provozu. Při náhodném generování payloadu jsou tato data kvůli rychlosti čtena z `/dev/urandom`.

Další odlišností od generování v režimu posílání celých souborů je fakt, že bez ohledu na to, zda je využito čtení ze souboru nebo generování náhodného payloadu, velikost odeslaného payloadu závisí pouze na náhodně vygenerované hodnotě z rozdělení příslušné modelované náhodné veličiny. Tedy zatímco předchozí režim vždy odeslal celý soubor, zde může být z každého souboru odeslána pouze část a stejně tak může být přečten a odeslán několikrát. Soubor není při čtení ukládán do proměnné, a proto je při několikanásobném odeslání několikanásobně čten.

Při generování náhodného payloadu nejsou z pcap souboru extrahovány reálně poslané soubory, které by v tomto případě zůstaly nevyužity. Ve zbytku postupu generování se tento režim shoduje s režimem posílání souborů.

3.2.5 Uživatelské vlastnosti vyvinutého nástroje

V této sekci popisuji několik dalších vlastností vytvořeného softwaru, které zlepšují použití vyvinutého nástroje uživatelem.

3.2.5.1 Konfigurační soubor

Model vytvořený generátorem je možné uložit do konfiguračního souboru a později z něj model opět načíst a použít ho pro generování náhodných hodnot bez nutnosti zpracovávat pcap soubor. Tento způsob načtení modelu je však v současné verzi možný pouze pro stochastické generování.

Konfigurační soubor pro každou modelovanou náhodnou veličinu zaznamenává její označení, rozdělení a parametry tohoto rozdělení. Jednotlivé údaje jsou na každém řádku odděleny čárkami, jedná se tedy o csv soubor.

3.2.5.2 Mazání dočasných souborů

Volbou přepínače při spouštění programu je možné ovlivnit, zda jsou soubory vytvořené při běhu programu po jeho skončení ponechány nebo odstraněny.

3.2.5.3 Opakovatelnost experimentu

Pro možnost zopakovaní konkrétního experimentu program umožňuje zadání semínka použitého náhodného generátoru. Toto semínko může být specifikováno jako vstupní parametr při spouštění programu. V případě, že semínko není uživatelem specifikováno, je místo něj použit čas spuštění programu.

3.2.5.4 Simulace generování provozu

V případě, kdy je nutné vygenerovat v krátkém čase velké množství hodnot podle modelu, lze tuto volbu zajistit zadáním příslušného parametru při spouštění programu. Hodnoty generované při běhu programu navíc mohou být zapisovány do souboru. Spojení těchto dvou možností dobře poslouží při ověřování modelu.

Kapitola 4

Testování

Původním záměrem bylo otestovat vyvinutý nástroj pomocí síťového zařízení umožňujícího automatizovanou analýzu webového provozu, žádné takové zařízení se však nepodařilo zajistit. Testování funkčnosti jsem proto provedl pomocí mnou vyvinutých nástrojů a nástroje Wireshark postupem uvedeným v následující sekci.

4.1 Získání dat pro tvorbu testovacího modelu

Nejdříve jsem pomocí Wiresharku několik minut zachytával webový provoz probíhající mezi mým počítačem a mnou navštěvovanými webovými servery. Vzhledem k tomu, že cílem bylo v co nejkratším čase získat rozumné množství dat pro tvorbu modelu, neodpovídají např. parametry veličiny modelující dobu prohlížení stránek běžnému chování uživatele. Tímto způsobem jsem zachytil zhruba 50 MB dat. Takto získaná data nemají obecnou vypovídající hodnotu o HTTP provozu, nicméně jejich využití pro otestování vytvořeného softwaru to nebrání.

4.2 Tvorba modelu a stochastické generování

V této části popisuji postup vytvoření modelu a jeho otestování využitím generování v režimu stochastického generování.

4.2.1 Postup testování

Generátor v režimu serveru z dat získaných podle popisu v části 4.1 spočítal hodnoty parametrů modelovaných veličin a poté podle nich řídil generovanou komunikaci se stranou klienta. Server byl umístěn na fyzickém stroji, klient byl umístěn ve virtuálním stroji spuštěném ve Virtualboxu s přístupem k síti v režimu Bridge. Přestože server i klient byly umístěny na jednom fyzickém stroji, komunikace probíhala přes router. Simulace byla spuštěna po dobu

jedné hodiny, zasílaný aplikační payload byl generován náhodně a celá komunikace byla opět zachycována Wiresharkem. Zároveň byly náhodně generované hodnoty zapisovány programem do souborů.

Po ukončení generování jsem pro jednotlivé modelované veličiny vytvořil histogramy rozdělení jednak hodnot zapsaných do souborů během generování a také hodnot získaných pomocí PcapAnalyzeru z Wiresharkem zachyceného generovaného provozu. Do histogramů jsem také zakreslil hustotu pravděpodobnosti rozdělení náhodných veličin.

Výše uvedeným postupem jsem ověřil následující:

- Generované náhodné hodnoty odpovídají modelu.
- Velikosti odesílaných dat a počty jednotlivých odesílaných objektů a časy čekání odpovídají generovaným hodnotám.
- PcapAnalyzer správně analyzuje data zachycená v pcap souborech.

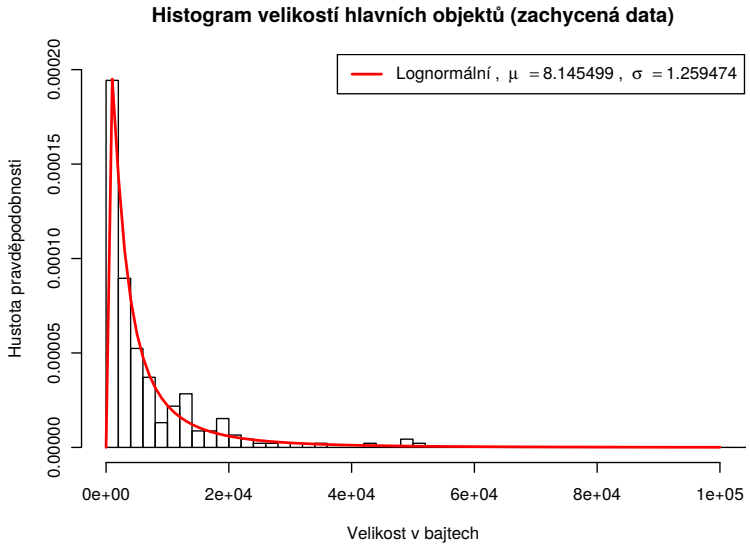
4.2.2 Analýza výsledků testování

Zjištěné výsledky lze analyzovat ze dvou hledisek:

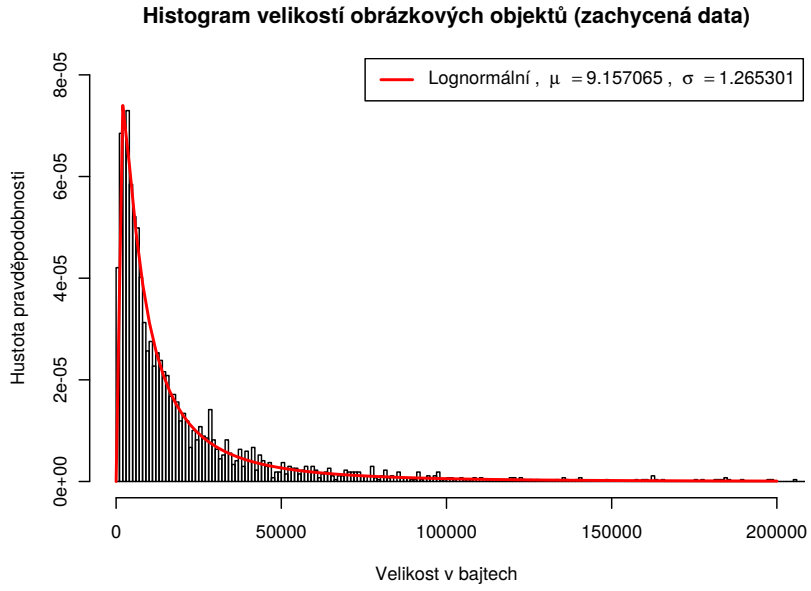
1. z pohledu shody generovaných hodnot a hodnot předpokládaných z modelu (a to porovnáním histogramu s předpokládanou hustotou pravděpodobnosti),
2. z pohledu shody generovaných hodnot a velikostí a počtů odesílaných objektů a použitých časů čekání (a to porovnáním histogramů generovaných a zachycených dat).

Na základě analýzy výsledků získaných postupem popsaným v sekci 4.2.1 lze říci, že rozdělení velikostí generovaných objektů poměrně přesně odpovídá velikostem daným modelem. Histogramy generovaných velikostí a velikostí zachycených objektů vypadají totožně a tvar histogramu se shoduje s grafem hustoty pravděpodobnosti. Dobrou shodu je možné pozorovat také pro čas prohlížení a velikost požadavků. Na ukázku proto přikládám pouze několik vybraných histogramů velikostí vytvořených ze zachycených dat, konkrétně pro velikosti hlavních objektů (obr. 4.1), obrázkových objektů (obr. 4.2) a javascriptových objektů (obr. 4.3). Ostatní vytvořené histogramy lze nalézt na přiloženém médiu.

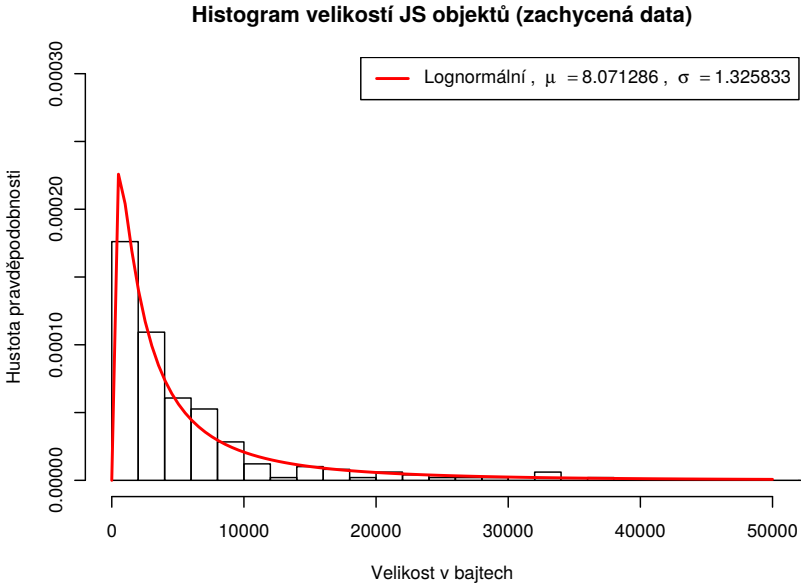
Rozdíly lze pozorovat pro počty odesílaných vestavěných objektů. Tyto rozdíly jsou způsobené implementací. Zatímco hodnoty jsou generovány ze spojitého rozdělení, objekt může být vždy odeslán celý nebo vůbec. Z toho důvodu jsou vygenerované hodnoty vždy zaokrouhleny k nejbližší celočíselné hodnotě a tento počet vestavěných objektů je následně odeslán. Způsobený rozdíl v počtu odeslaných objektů proti generovaným hodnotám lze vidět např.



Obrázek 4.1: Histogram velikostí generovaných hlavních objektů (analýza zachyceného provozu).



Obrázek 4.2: Histogram velikostí generovaných vestavěných obrázkových objektů (analýza zachyceného provozu).



Obrázek 4.3: Histogram velikostí generovaných vestavěných javascriptových objektů (analýza zachyceného provozu).

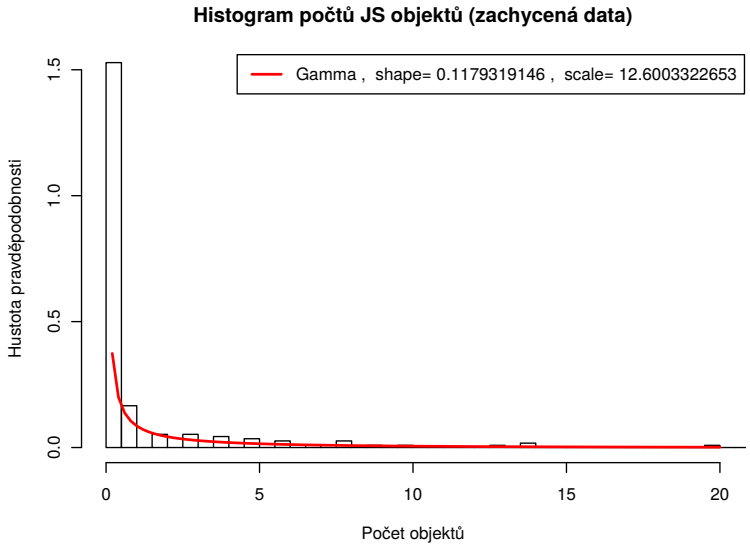
na obrázcích 4.4 a 4.5. Vhodnější úpravou vygenerovaných hodnot na celočíselné, např. postupným načítáním desetinných zbytků, by šlo patrně docílit lepší shody mezi generovaným a odeslaným počtem objektů.

Histogramy na obrázcích 4.6 a 4.7 pro časy mezi odchody jednotlivých vestavěných objektů se rovněž liší. Rozdíl je způsoben režií na manipulaci s vlákny, která se při krátkých generovaných rozestupech nestíhají spouštět dostatečně rychle.

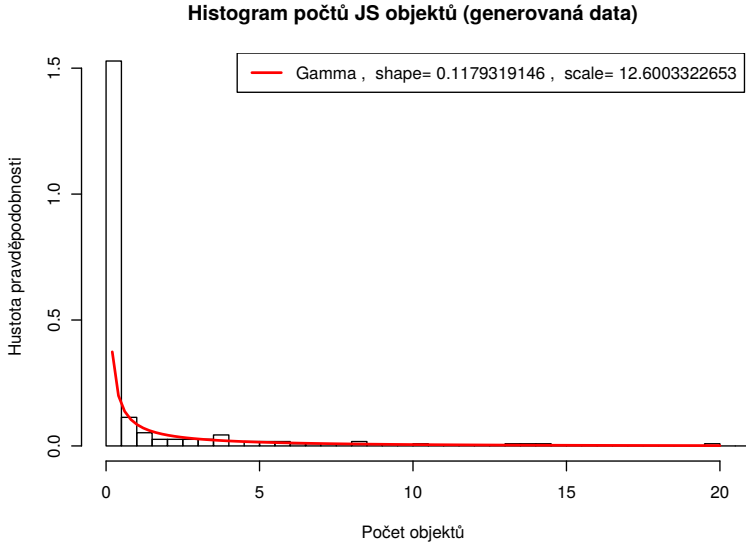
Protože v provozu vygenerovaném a odeslaném během šedesáti minut nevzniklo dostatečné množství dat, aby byla shoda mezi histogramy generovaných dat a hustotami rozdělení náhodných veličin modelu evidentní, provedl jsem ještě jedno ověření. Toto ověření spočívalo pouze v generování náhodných hodnot dle daných rozdělení bez toho, aby byl provoz generován a odesílán. Tuto simulaci jsem spustil po dobu třiceti vteřin, což stačilo k zisku dostatečného množství dat. Zpřesnění je vidět např. na histogramech pro čas prohlížení (obr. 4.8) nebo pro čas parsování (obr. 4.9).

4.3 Posílání souborů

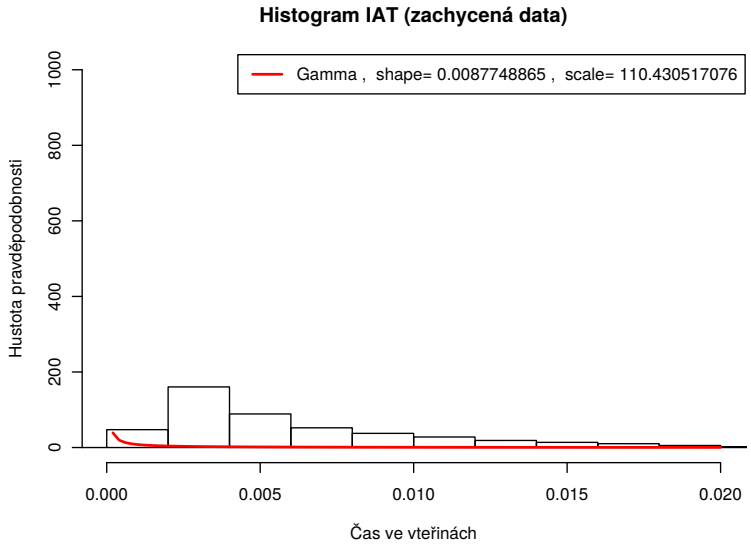
Zde jsem ověřil počty odesílaných souborů a časové rozestupy mezi jejich odesíláním, generování testovacího provozu probíhalo po dobu deseti minut. Tvorba modelu a generování náhodných hodnot probíhá stejným způsobem



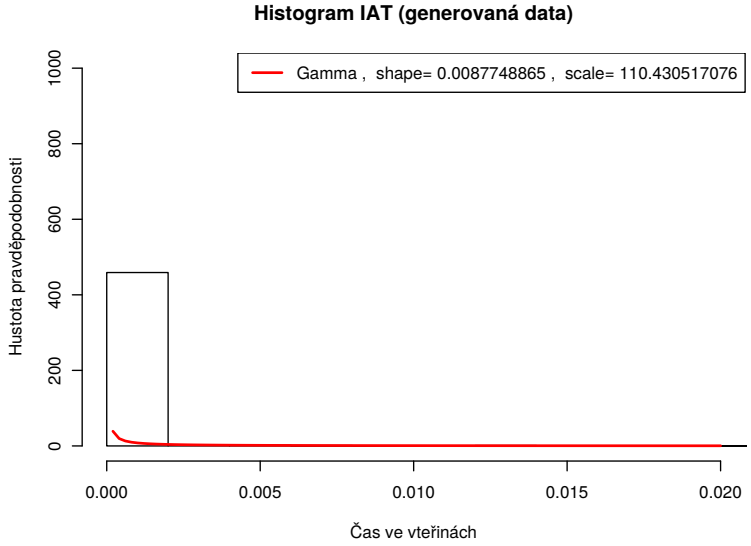
Obrázek 4.4: Histogram počtů zachycených javascriptových objektů.



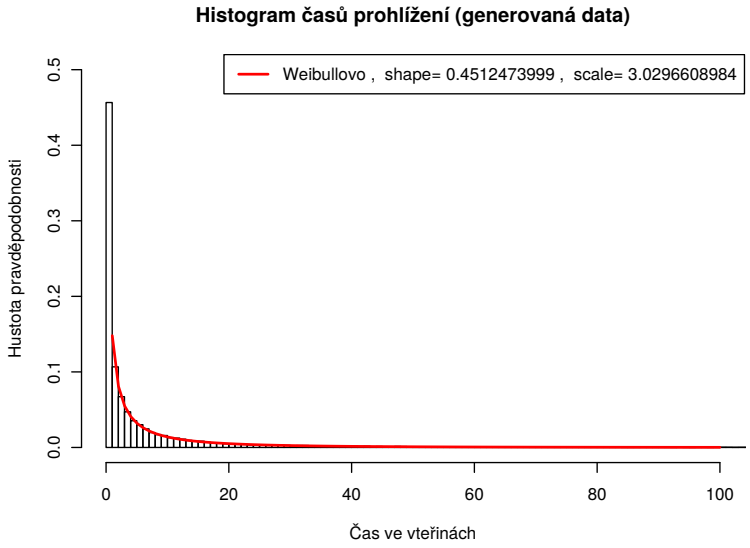
Obrázek 4.5: Histogram generovaných počtů javascriptových objektů.



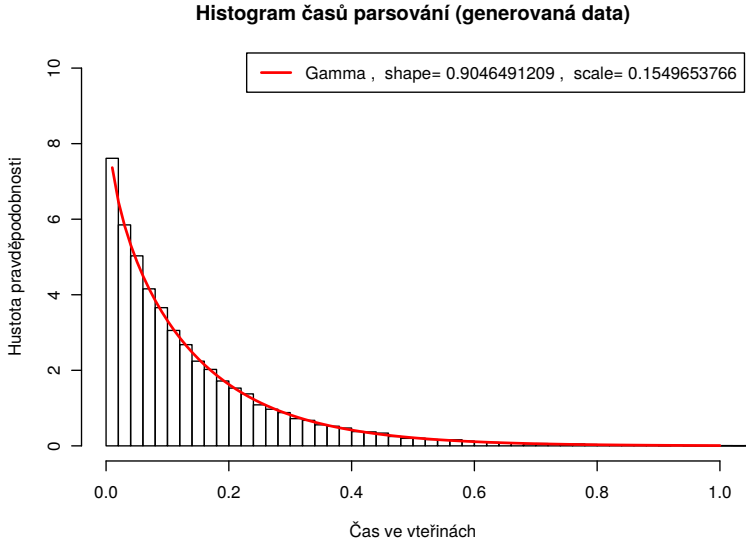
Obrázek 4.6: Histogram IAT zachycených vestavěných objektů.



Obrázek 4.7: Histogram generovaných IAT vestavěných objektů.



Obrázek 4.8: Histogram časů prohlížení, data získána simulací generování bez odesílání provozu.



Obrázek 4.9: Histogram časů parsování, data získána simulací generování bez odesílání provozu.

jako v předchozím režimu, proto není překvapující dobrá shoda vybraných histogramů s předpokládaným rozdělením daným modelem. Protože jsou výsledné histogramy velmi podobné těm prezentovaným v předchozí části, nepřikládám je sem a ponechávám je pouze na přiloženém médiu.

Díky použitým hlavičkám HTTP protokolu odesílaným spolu s daty je generovaný provoz při zachycení Wiresharkem rozpoznán jako HTTP provoz. Mnoho posílaných objektů je při zachycení Wiresharkem rozpoznáno jako skutečné HTTP objekty, přičemž jsou automaticky zobrazeny jejich vlastnosti, např. hlavičky PNG obrázků obsahující výšku či šířku obrázku. Bohužel jsem také našel některé objekty, které jsou při zobrazení Wiresharkem označeny jako `[Malformed Packet]`. Dle dokumentace [76] to nejspíše znamená, že se zaslaná data nepodařilo správně složit nebo paket není v očekávaném formátu. Protože se tento problém objevuje téměř výhradně u velkých JPEG obrázků, domnívám se, že uvedené označení je paketu přiděleno na základě neúspěšného složení zachycených dat. To může být způsobeno nesouladem mezi informacemi v HTTP hlavičce a daty v payloadu, nicméně jednoznačnou příčinu jsem neidentifikoval.

Podstatným zjištěním této a předchozí části testování je fakt, že zachycený provoz je automaticky rozpoznán jako HTTP, čímž je splněn cíl generování provozu na úrovni vybraného aplikačního protokolu.

4.4 Přeposílání paketů

Ověřování funkčnosti tohoto režimu jsem provedl dvakrát, přičemž jednou jsem ručně analyzoval úpravu časování, podruhé už pouze správnost odeslání vytvořeného souboru. Oba způsoby jsou popsány v této sekci.

4.4.1 Test s ověřením generovaného časování

Testování tohoto režimu jsem provedl v několika krocích. Prvním krokem bylo ověření, že časování ve výsledném pcap souboru odpovídá náhodným hodnotám generovaným dle modelu. Protože nejsou do souboru vkládány HTTP požadavky, nelze ho zpracovat pomocí PcapAnalyzeru. Ověření jsem provedl spuštěním generátoru s požadavkem na vytvoření pcap souboru obsahujícího dva hlavní objekty. První náhodně vybraný hlavní objekt měl dva vestavěné objekty, druhý neměl žádné. Na těchto čtyřech objektech jsem mohl ověřit všechna generovaná časování – čas parsování, čas mezi odchody vestavěných objektů a čas prohlížení stránky uživatelem.

Rozdíly časování paketů obsažených v pcap souboru velmi přesně odpovídaly vygenerovaným hodnotám, konkrétní hodnoty časů a spočtené rozdíly mezi nimi jsou uvedeny v tabulce 4.1.

Vzniklý soubor byl poté přehrán pomocí Tcpreplay, provoz zachycen do nového pcap souboru a časování v obou pcap souborech porovnáno. Časy posílání jednotlivých objektů se mírně odlišovaly od pátého desetinného místa,

Tabulka 4.1: Časování ve vytvořeném testovacím pcap souboru porovnané s generovanými hodnotami. t_1 = čas prvního paketu v sekundách, t_l = čas posledního paketu v sekundách, $diff_{real}$ = spočítaný příslušný rozdíl v sekundách, $diff_{gen}$ = generovaný rozdíl v sekundách, typ času – pars = čas parsování, IAT = čas mezi příchody prvních paketů vestavěných objektů, view = čas prohlížení stránky uživatelem.

Identifikace objektu	$t_1[s]$	$t_l[s]$	$diff_{real}[s]$	$diff_{gen}[s]$	typ
1. hlavní objekt	0.000000	0.051669	0.084713	0.0847136	pars
1. vestavěný objekt	0.136382	0.136382	0.000000	10^{-21}	IAT
2. vestavěný objekt	0.136382	0.136382	0.008418	0.0084178	view
2. hlavní objekt	0.144800	0.144800	–	–	–

což je nejspíše způsobeno rychlostí čtení a manipulace s pakety nástrojem Tcpreplay. Zachycené objekty jsou správně identifikované jako HTTP objekty a je možné zjistit jejich informace stejně jako v předchozím režimu.

4.4.2 Test bez ověření generovaného časování

Pro ověření odesílání jsem nechal vygenerovat soubor obsahující pět náhodně vybraných hlavních objektů, přičemž některé z nich obsahovaly velké množství vestavěných objektů. Vytvořený pcap soubor obsahující tyto objekty byl odeslán z určeného síťového rozhraní, kde byl provoz zachycen. Cílem bylo srovnání shody časů v generovaném a zachyceném provozu. Oba soubory jsou na přiloženém médiu.

Časování odpovídá rozdílům popsaným v předchozí části, kdy se jednotlivé časy liší na méně významných desetinných pozicích. Podstatný rozdíl jsem zaznamenal mezi 248. a 249. zachyceným rámcem. Oba rámce patří ke stejnému objektu, nicméně je mezi jejich zachycením více než desetisekundový rozdíl. V původně vytvořeném pcap souboru je rozdíl zhruba milisekunda. Skokový nárůst času zachycení je také mezi 569. a 570. rámcem (23 sekund oproti očekávanému rozdílu v řádu mikrosekund) a mezi 726. a 727. rámcem (11 sekund oproti očekávanému rozdílu v řádu mikrosekund). Domnívám se, že tento rozdíl je dán velikostí síťové vyrovnávací paměti (bufferu) alokované operačním systémem [62]. Pro správné chování generátoru v tomto režimu při posílání většího množství paketů je nutné velikost této paměti nastavit, generátor tuto funkcionalitu nezajišťuje.

Závěr

V této práci jsem se zaměřil na modelování síťového provozu na aplikační úrovni. Během analýzy existujících přístupů jsem vybral behaviorální model pro webový provoz [7]. Pro zpřesnění vybraného modelu jsem navrhl rozšíření pomocí rozlišení typů vestavěných objektů ve webové stránce. Toto rozšíření je aplikovatelné i na jiné podobné modely popsané v sekci 1.1.4. Pro rozlišení těchto typů objektů jsem navrhl dva postupy. První postup spočívá ve využití signatur MIME typů jednotlivých objektů, druhý vychází z využití EM algoritmu pro modelování směsí statistických rozdělení. Postup spoléhající na MIME typy objektů jsem využil pro implementované rozšíření vybraného modelu webového provozu. Rozšířený model se stal základem pro vytvořený generátor HTTP provozu. Konkrétní parametry modelu jsou spočítány přímo generátorem implementovaným v této práci na základě vstupního pcap souboru. Generátor může pracovat v režimu přeposílání dříve zachycených paketů, v režimu posílání celých souborů vytvořených ze zachycených paketů a v režimu stochastického generování, přičemž míra využití statistického modelu se pro každý režim liší.

Součástí této práce je vytvořený generátor síťového provozu na aplikační úrovni. Vybraný aplikační protokol je HTTP, pro který je možné modelovat strukturu jeho složení, charakteristiky jeho časování a chování surfujícího uživatele. Zdůvodnění této volby je popsáno v sekci 1.1.4. Návrh generátoru provozu je popsán v kapitole 2. Dobrou vlastností vyvinutého generátoru je paralelní obsluha více klientů, díky čemuž lze simulovat komunikaci více klientů s webovým serverem. Při této komunikaci je náhodná sekvence sdílena mezi jednotlivými klienty, což jsem zajistil centralizovaným řízením komunikace serverem.

V kapitole 4 bylo testováno odesílání provozu v jednotlivých režimech a statistická přesnost vůči použitému modelu. Toto testování dopadlo velmi dobře pro velikosti a počty odesílaných objektů, nicméně odhalilo slabinu v podobě režie na spouštění a komunikaci výpočetních vláken. Ta způsobila odchylky v rozdělení časů mezi odchody jednotlivých vestavěných objektů. Díky využití

hlaviček HTTP protokolu vkládaných do generovaného provozu je tento provoz skutečně rozpoznáván jako HTTP. V režimu posílání dříve zachycených souborů a monitorování generovaného provozu např. pomocí Wiresharku, jsou tyto soubory Wiresharkem rozpoznány.

I přes dobrou funkčnost výsledného nástroje se nabízí velký prostor pro další rozvoj. Současná implementace závisí na poskytnutém modelu s parametry, které mohou být zadány uživatelem, nebo je generátor sám dovede odhadnout z napozorovaných dat v pcap souboru. V budoucnu by bylo vhodné implementovat automatické zvolení (odhad) statistického rozdělení pro jednotlivé náhodné veličiny modelu, protože v současné podobě jsou rozdělení jednotlivým modelovaným veličinám pevně přiřazena a odhadovány jsou pouze hodnoty parametrů rozdělení. To je však komplikované a nebylo cílem této práce. Vzhledem k růstu využívání IPFIX toků uvažuji vyvinutý nástroj rozšířit i o analýzu těchto toků a jejich využití pro automatizované odhadování konkrétních hodnot parametrů modelu. Možnosti získání těchto hodnot z IPFIX toků jsou popsány v sekci 2.1.3.2.

Literatura

[1] Wilk, A.; Iyengar, J.; Swett, I.; aj.: QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. 2016, [cit. 25.4.2016]. Dostupné z: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02

[2] Ertoz, L.; Eilertson, E.; Lazarevic, A.; aj.: Minds-minnesota intrusion detection system. *Next generation data mining*, 2004: s. 199–218, [cit. 25.4.2016]. Dostupné z: https://www.researchgate.net/profile/Vipin_Kumar26/publication/2878372_MINDS_-_Minnesota_Intrusion_Detection_System/links/00b4951675a87be3e0000000.pdf

[3] Botta, A.; Dainotti, A.; Pescapé, A.: A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, ročník 56, č. 15, 2012: s. 3531–3547, [cit. 25.4.2016]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S1389128612000928

[4] Wright, C. V.; Connelly, C.; Braje, T.; aj.: Generating client workloads and high-fidelity network traffic for controllable, repeatable experiments in computer security. In *Recent advances in intrusion detection*, Springer, 2010, s. 218–237, [cit. 25.4.2016]. Dostupné z: http://link.springer.com/chapter/10.1007/978-3-642-15512-3_12

[5] Sommers, J.; Yegneswaran, V.; Barford, P.: A framework for malicious workload generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, ACM, 2004, s. 82–87, [cit. 25.4.2016]. Dostupné z: http://dl.acm.org/citation.cfm?id=1028799

[6] Heidemann, J.; Obraczka, K.; Touch, J.: Modeling the performance of HTTP over several transport protocols. *Networking, IEEE/ACM Transactions*, ročník 5, č. 5, 1997: s. 616–630, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=649564

[7] Choi, H.-K.; Limb, J. O.: A behavioral model of Web traffic. In *Seventh International Conference on Network Protocols, 1999. (ICNP '99) Proceedings*, 1999, s. 327–334, [cit. 25.4.2016]. Dostupné z: http://www.cc.gatech.edu/computing/Networking/people/Phd/hkchoi/paper/model.pdf

[8] Paxson, V.: Empirically-derived analytic models of wide-area TCP connections: Extended report. Technická zpráva, Lawrence Berkeley Laboratory, 1993, [cit. 25.4.2016]. Dostupné z: http://www.cise.ufl.edu/class/lbsctp/papers/vern_paxson_paper.pdf

[9] Danzig, P. B.; Jamin, S.: tcplib: A library of internetwork traffic characteristics. 1991, [cit. 25.4.2016]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.5210

[10] Casilari, E.; Reyes-Lecuona, A.; Gonzalez, F. J.; aj.: Characterisation of web traffic. In *Global Telecommunications Conference, 2001. GLOBECOM'01.*, ročník 3, IEEE, 2001, s. 1862–1866, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=965897

[11] Kim, M.-S.; Won, Y. J.; Hong, J. W.-K.: Application-level traffic monitoring and an analysis on IP networks. *ETRI journal*, ročník 27, č. 1, 2005: s. 22–42, [cit. 25.4.2016]. Dostupné z: http://etrij.etri.re.kr/etrij/journal/article/article.do?volume=27&issue=&page=22

[12] B. Claise, P. A., B. Trammell: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. 2013, [cit. 29.4.2016]. Dostupné z: https://tools.ietf.org/html/rfc7011

[13] Claise, B.: Cisco systems NetFlow services export version 9. 2004, [cit. 25.4.2016]. Dostupné z: http://tools.ietf.org/html/rfc3954.html

[14] Williamson, C.: Internet traffic measurement. *Internet Computing, IEEE*, ročník 5, č. 6, 2001: s. 70–74, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=968834

[15] Cáceres, R.; Danzig, P. B.; Jamin, S.; aj.: Characteristics of wide-area TCP/IP conversations. ACM, 1991, s. 101–112, [cit. 25.4.2016]. Dostupné z: http://dl.acm.org/citation.cfm?id=116003

[16] CISCO: Cisco Visual Networking Index: Forecast and Methodology, 2014-2019. Technická zpráva, CISCO, 2015, [cit. 25.4.2016]. Dostupné z: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf

[17] Suznjevic, M.; Stupar, I.; Matijasevic, M.: A model and software architecture for MMORPG traffic generation based on player behavior. *Multimedia systems*, ročník 19, č. 3, 2013: s. 231–253, [cit. 25.4.2016]. Dostupné z: http://link.springer.com/article/10.1007/s00530-012-0269-x

[18] Zink, M.; Suh, K.; Gu, Y.; aj.: Characteristics of YouTube network traffic at a campus network-measurements, models, and implications. *Computer Networks*, ročník 53, č. 4, 2009: s. 501–514, [cit. 25.4.2016]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S1389128608003423

[19] Cardoso, K. V.; de Rezende, J. F.: Design and Use of an aggregated HTTP Traffic Model. [cit. 25.4.2016]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.6814&rep=rep1&type=pdf

[20] Pries, R.; Magyari, Z.; Tran-Gia, P.: An HTTP web traffic model based on the top one million visited web pages. In *2012 8th EURO-NGI Conference on Next Generation Internet (NGI)*, 2012, s. 133–139.

[21] Dainotti, A.; Pescapé, A.; Ventre, G.: A packet-level characterization of network traffic. In *Computer-Aided Modeling, Analysis and Design of Communication Links and Networks, 2006 11th International Workshop*, IEEE, 2006, s. 38–45, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1649716

[22] Cao, J.; Cleveland, W. S.; Gao, Y.; aj.: Stochastic models for generating synthetic HTTP source traffic. *contract*, ročník 30602, č. 02-C, 2004: str. 0093, [cit. 25.4.2016]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.156.9421&rep=rep1&type=pdf

[23] Mogul, J. C.; Gettys, J.; Frystyk, H.; aj.: Hypertext Transfer Protocol – HTTP/1.1. 1997, [cit. 25.4.2016]. Dostupné z: https://tools.ietf.org/html/rfc2068

[24] Mah, B.; aj.: An empirical model of HTTP network traffic. In *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, ročník 2, IEEE, 1997, s. 592–600, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=644510

[25] Bhole, Y.; Popescu, A.: Measurement and analysis of http traffic. *Journal of Network and Systems Management*, ročník 13, č. 4, 2005: s. 357–371, [cit. 25.4.2016]. Dostupné z: http://link.springer.com/article/10.1007/s10922-005-9000-y

[26] Lee, J. J.; Gupta, M.: A new traffic model for current user web browsing behavior. *Intel corporation*, 2007, [cit. 25.4.2016]. Dostupné z: http://blogs.intel.com/wp-content/mt-content/com/research/HTTP%20Traffic%20Model_v1%201%20white%f20paper.pdf

[27] Bujlow, T.; Riaz, T.; Pedersen, J. M.: Classification of HTTP traffic based on C5. 0 Machine Learning Algorithm. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, IEEE, 2012, s. 882–887, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6249413

[28] Parmar, H.; Thornbutgh, M.: Adobe's Real Time Messaging Protocol. 2012, [cit. 25.4.2016]. Dostupné z: http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf

[29] Berger, A. L.; Pietra, V. J. D.; Pietra, S. A. D.: A maximum entropy approach to natural language processing. *Computational linguistics*, ročník 22, č. 1, 1996: s. 39–71, [cit. 25.4.2016]. Dostupné z: http://dl.acm.org/citation.cfm?id=234289

[30] Kao, H. Y.; Lin, S. H.; Ho, J. M.; aj.: Mining web informative structures and contents based on entropy analysis. *Knowledge and Data Engineering, IEEE Transactions*, ročník 16, č. 1, 2004: s. 41–55, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1264821

[31] Kaoprakhon, S.; Visoottiviseth, V.: Classification of audio and video traffic over HTTP protocol. In *Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on*, IEEE, 2009, s. 1534–1539, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5341026

[32] Feldmann, A.; Greenberg, A.; Lund, C.; aj.: Continuous online extraction of HTTP traces from packet traces. In *Proc. W3C Web Characterization Group Workshop*, Citeseer, 1998, [cit. 25.4.2016]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.2164&rep=rep1&type=pdf

[33] Spangler, R.: Packet sniffer detection with antisniff. *University of Wisconsin, Whitewater. Department of Computer and Network Administration*, 2003, [cit. 25.4.2016]. Dostupné z: http://ma-dn-project-soa.googlecode.com/svn-history/r38/trunk/DOC/Snort/snifferdetection.pdf

[34] Ansari, S.; Rajeev, S. G.; Chandrashekar, H. S.: Packet sniffing: a brief introduction. *Potentials, IEEE*, ročník 21, č. 5, 2002: s. 17–19,

[cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1166620

[35] McCanne, S.; Jacobson, V.: The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference*, USENIX Association, 1993, s. 2–2, [cit. 25.4.2016]. Dostupné z: http://dl.acm.org/citation.cfm?id=1267305

[36] Tcpdump.org: *Tcpdump & Libpcap*. [software]. [cit. 29.4.2016]. Dostupné z: http://www.tcpdump.org/

[37] Fuentes, F.; Kar, D. C.: Ethereal vs. Tcpdump: a comparative study on packet sniffing tools for educational purpose. *Journal of Computing Sciences in Colleges*, ročník 20, č. 4, 2005: s. 169–176, [cit. 25.4.2016]. Dostupné z: http://dl.acm.org/citation.cfm?id=1047873

[38] *pcap - Packet Capture library*. [software]. [cit. 29.4.2016]. Dostupné z: http://manpages.ubuntu.com/manpages/xenial/en/man3/pcap.3pcap.html

[39] WinDump. [software]. [cit. 9.5.2016]. Dostupné z: https://www.winpcap.org/windump/

[40] Cáceres, R.; Sreenan, C. J.; Van Der Merwe, J. E.: mmdump–a tool for monitoring multimedia usage on the internet. *ACM Computer Communication Review.*, 2000, [cit. 25.4.2016]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.2232&rep=rep1&type=pdf

[41] Wireshark. 2016, [software]. [cit. 29.4.2016]. Dostupné z: https://www.wireshark.org/

[42] Gandhi, C.; Suri, G.; Golyan, R. P.; aj.: Packet Sniffer–A Comparative Study. [cit. 9.5.2016]. Dostupné z: http://www.ijcncs.org/published/volume2/issue5/p6_2-5.pdf

[43] Wagner, A.; Trammell, B.; Claise, B.: Flow Aggregation for the IP Flow Information Export (IPFIX) Protocol. 2013, [cit. 9.5.2016]. Dostupné z: https://tools.ietf.org/html/rfc7015

[44] NfDump. 2014, [software]. [cit. 29.4.2016]. Dostupné z: http://nfdump.sourceforge.net/

[45] NfSen. 2011, [software]. [cit. 4.5.2016]. Dostupné z: http://nfsen.sourceforge.net/

[46] *fprobe – a NetFlow probe*. [software]. [cit. 29.4.2016]. Dostupné z: http://manpages.ubuntu.com/manpages/wily/man8/fprobe.8.html

[47] Flowmon Networks: Flowmon. 2016, [software]. [cit. 29.4.2016]. Dostupné z: https://www.flowmon.com/cs/products/flowmon

[48] Fan, P.: Requirements for Application Layer Information Export in IP Flow Information Export (IPFIX). 2013, [cit. 20.4.2016]. Dostupné z: https://tools.ietf.org/html/draft-fan-ipfix-content-info-req-01

[49] Citrix Systems, Inc.: AppFlow. 2015, [software]. [cit. 29.4.2016]. Dostupné z: http://docs.citrix.com/en-us/netscaler/10-1/ns-system-wrapper-10-con/ns-ag-appflow-intro-wrapper-con.html

[50] Takeshita, K.; Kurosawa, T.; Tsujino, M.; aj.: Evaluation of HTTP video classification method using flow group information. In *Telecommunications Network Strategy and Planning Symposium (NETWORKS), 2010 14th International*, IEEE, 2010, s. 1–6, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5624929

[51] Gerber, A.; Houle, J.; Nguyen, H.; aj.: P2p the gorilla in the cable. *Proc of National Cable and Telecommunications Association (NCTA), National Show*, 2003: s. 8–11, [cit. 25.4.2016]. Dostupné z: http://web2.research.att.com/export/sites/att_labs/techdocs/TD_5KEP8A.pdf?Big%20Data

[52] Mahoney, M. V.: Network traffic anomaly detection based on packet bytes. In *Proceedings of the 2003 ACM symposium on Applied computing*, ACM, 2003, s. 346–350, [cit. 25.4.2016]. Dostupné z: http://dl.acm.org/citation.cfm?id=952601

[53] Ihm, S.; Pai, V. S.: Towards understanding modern web traffic. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, ACM, 2011, s. 295–312, [cit. 25.4.2016]. Dostupné z: http://dl.acm.org/citation.cfm?id=2068845

[54] Botta, A.; Dainotti, A.; Pescapé, A.: Do you trust your software-based traffic generator? *Communications Magazine*, ročník 48, č. 9, 2010: s. 158–165, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5560600

[55] Barford, P.; Crovella, M.: An architecture for a WWW workload generator. In *Wide Web Consortium Workshop on Workload Characterization*, Citeseer, 1997, [cit. 25.4.2016]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.1522&rep=rep1&type=pdf

[56] Sommers, J.; Kim, H.; Barford, P.: Harpoon: a flow-level traffic generator for router and network tests. In *ACM SIGMETRICS Performance*

Evaluation Review, ročník 32, ACM, 2004, s. 392–392, [cit. 25.4.2016]. Dostupné z: http://dl.acm.org/citation.cfm?id=1005733

[57] Vishwanath, K. V.; Vahdat, A.: Swing: Realistic and responsive network traffic generation. In *Networking, IEEE/ACM Transactions on*, ročník 17, 2009, s. 712–725, [cit. 25.4.2016]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4914755

[58] Papadopoulos, V.: *Experimental Assessment of Benchmark-oriented Network Traffic Generators*. Dizertační práce, Universidad Carlos III de Madrid, Španělsko, 2013, [cit. 25.4.2016]. Dostupné z: http://eprints.networks.imdea.org/545/

[59] Vishwanath, K. V.; Vahdat, A.: Realistic and responsive network traffic generation. In *ACM SIGCOMM Computer Communication Review*, ročník 36, 2006, s. 111–122, [cit. 25.4.2016]. Dostupné z: http://dl.acm.org/citation.cfm?id=1159928

[60] Bonelli, N.; Giordano, S.; Procissi, G.; aj.: BRUTE: A high performance and extensible traffic generator. 2005, s. 839–845, [cit. 25.4.2016]. Dostupné z: http://homepages.abdn.ac.uk/r.secchi/pages/siteroot/papers/TMA-019_Secchi.pdf

[61] Kant, K.; Tewary, V.; Iyer, R.: An internet traffic generator for server architecture evaluation. In *Proc. CAECW*, 2001, [cit. 25.4.2016]. Dostupné z: http://iacoma.cs.uiuc.edu/caecw01/geist.pdf

[62] *Tcpreplay*. [software]. [cit. 4.5.2016]. Dostupné z: http://tcpreplay.synfin.net/wiki/tcpreplay

[63] Avallone, S.; Pescape, A.; Ventre, G.: Analysis and experimentation of internet traffic generator. In *NEW2AN*, 2004, s. 70–75, [cit. 25.4.2016]. Dostupné z: http://traffic.comics.unina.it/software/ITG/D-ITGpublications/New2an-ITG.pdf

[64] McKenney, P. E.; Lee, D. Y.; Denny, B. A.: *Traffic generator software release notes*. SRI International and USC/ISI Postel Center for Experimental Networking, 2002, [cit. 25.4.2016]. Dostupné z: http://128.9.160.29/tg/tg2002.ps

[65] Wang, S. Y.; Huang, Y. M.: NCTUns distributed network emulator. In *Internet Journal*, ročník 4, 2012, s. 61–94, [cit. 25.4.2016]. Dostupné z: https://nsl23.cs.nctu.edu.tw/NSL/files/NovaDisEmu.pdf

[66] *ns-2: The Network Simulator*. [software]. [cit. 4.5.2016]. Dostupné z: http://www.isi.edu/nsnam/ns/

[67] Salehi, N.; Obraczka, K.; Neuman, C.: The performance of a reliable, request-response transport protocol. IEEE, 1999, str. 102, [cit. 25.4.2016]. Dostupné z: http://www.computer.org/csdl/proceedings/iscc/1999/0250/00/02500102.pdf

[68] Neumetrix Limited: *HTTP Gallery*. 2016, [online]. [cit. 4.5.2016]. Dostupné z: https://www.httpwatch.com/httpgallery/

[69] Python 3 vs C++ g++ (64-bit Ubuntu quad core) | Computer Language Benchmarks Game. [cit. 25.4.2016]. Dostupné z: http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gpp

[70] *NumPy*. 2016, [software]. [cit. 29.4.2016]. Dostupné z: http://www.numpy.org/

[71] *SciPy*. 2016, [software]. [cit. 29.4.2016]. Dostupné z: https://www.scipy.org/

[72] Blažek, R. B.: *Random number generator for the command line*. České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2012, [cit. 29.4.2016]. Dostupné z: http://users.fit.cvut.cz/~blazerud/miesib/Binaries-Linux/rg0.4.4beta/rgDocs.pdf

[73] Blažek, R. B.: *The EM algorithm for geometric mixtures*. České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2015.

[74] Fontanini, M.: *libtins: C++ packet sniffing and crafting library*. 2016, [software]. [cit. 29.4.2016]. Dostupné z: http://libtins.github.io/

[75] Biondi, P.: Scapy. 2007, [software]. [cit. 25.4.2016]. Dostupné z: http://www.secdev.org/projects/scapy/

[76] Lamping, U.; Sharpe, R.; Warnicke, E.: *Wireshark User's Guide: Wireshark Messages*. 2014, [online]. [cit. 4.5.2016]. Dostupné z: https://www.wireshark.org/docs/wsug_html_chunked/AppMessages.html

Příloha **A**

Seznam použitých zkratek

API Application Program Interface

APDU Application Protocol Data Unit, Datová jednotka aplikačního protokolu

COM Component Object Model

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

IP Internet Protocol

MAC Media Access Control

MTU Maximum Transmission Unit

OSI Open Systems Interconnection

P2P Peer-to-peer

RREs Request-Response-Exchanges

RTMP Real-Time Messaging Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

URL Uniform Resource Locator

Příloha B

Uživatelská příručka

B.1 Zpracování pcap souboru

B.1.1 Instalace

Zpracování pcap souboru zajišťuje PcapAnalyzer. Zdrojový kód je napsán v jazyku C++, verze C++11, s využitím knihovny STL. Prerekvizitou pro jeho úspěšné zkompilování je nainstalovaná knihovna Libtins [74]. Spustitelný binární soubor se vytvoří příkazem `make compile`, dokumentace příkazem `make doc`, případně společně příkazem `make all`. Příkazem `make install` je spustitelný soubor zkopírován do `/usr/local/bin`, kde ho následně hledá generátor.

B.1.2 Použití

Vstupní přepínače ke spuštění programu jsou uvedeny v tabulce B.1.

B.2 Generování

B.2.1 Příprava ke spuštění

Zdrojový kód generátoru byl napsán v jazyku Python verze 3. Pro spuštění programu stačí mít právo spuštění na daný skript.

Generátor v předstihu nekontroluje, zda jsou splněny všechny programové závislosti, nicméně v případě nenalezení požadovaného programu se ukončí. Pro běh programu ve všech režimech je nutné mít nainstalované programy uvedené v seznamu níže, nicméně některé programy nejsou využity každým režimem.

- Editcap, nutný pro režim přeposílání paketů.
- Mergecap, nutný pro režim přeposílání paketů.

Tabulka B.1: Vstupní přepínače PcapAnalyzeru

Přepínač	Význam
`-h`, `--help`	Vytiskne nápovědu na standardní výstup.
`-d <cestaKAdresari>`	Cesta k adresáři, do kterého budou umístěny soubory s hodnotami pro výpočet parametrů modelu.
`-f`	Když je přítomen, budou vytvořeny soubory se seznamy hlavních a vestavěných objektů. Cesty souborů je nutné specifikovat přepínači `-om` a `-oe`.
`-i <cestaKPcapSouboru>`	Cesta k pcap souboru, který má být analyzován.
`-m`	Když je přítomen, budou na standardní výstup vytisknuty seznamy hodnot pro výpočty jednotlivých parametrů modelu.
`-oe <cestaKSouboru>`	Cesta k souboru, do kterého bude zapsán seznam vestavěných objektů. Má význam pouze při přítomnosti přepínače `-f`.
`-om <cestaKSouboru>`	Cesta k souboru, do kterého bude zapsán seznam hlavních objektů. Má význam pouze při přítomnosti přepínače `-f`.

- Netcat (nc), vhodný v případě, že je spuštěn klient pro režim přeposílání paketů. Přijímání provozu na straně klienta v tomto režimu však není nutné.
- PcapAnalyzer, nutný pro vytvoření modelu.
- Tcpflow, nutný pro režim posílání souborů a případně režim stochastického generování, pokud má být payload čten z reálných souborů.
- Tcpprep, nutný pro režim přeposílání paketů.
- Tcpreplay, nutný pro režim přeposílání paketů.
- Tcprewrite, nutný pro režim přeposílání paketů.
- Tshark, nutný pro režim přeposílání paketů.

B.2.2 Vstupní parametry

Vstupní přepínače pro generátor síťového provozu jsou uvedeny v tabulce B.2.

Tabulka B.2: Vstupní přepínače generátoru síťového provozu.

Parametr	Význam
`-h`, `--help`	Vytiskne nápovědu na standardní výstup.
`-b <modChovani>`	Chování programu, klient nebo server.
`-c <IP:port,IP:port,..>`	Adresy a porty klientů, dvojice `IP:port` oddělené čárkami. Na těchto portech bude spuštěn signální kanál.
`-d`	Pokud je přítomen, náhodně generované hodnoty budou uloženy do příslušných souborů.
`-e`	Pokud je přítomen, budou v režimu přeposílání paketů s hlavním objektem přeposílány vestavěné objekty jemu původně příslušející.
`-f <cestaKSouboru>`	Pcap soubor, na jehož základě bude vytvořen model.
`-g`	Pokud je přítomen, bude spuštěno pouze náhodné generování hodnot dle modelu, bez generování síťového provozu.
`-i <rozhrani>`	Síťové rozhraní, na které v režimu přeposílání paketů budou pakety vkládány.
`-k`	Pokud je přítomen, nebudou po skončení programu odstraněny dočasné soubory.
`-m <modGenerovani>`	Režim generování: přeposílání paketů, posílání souborů nebo stochastické generování.
`-n <zvolenaHodnota>`	Počet vybraných hlavních objektů, pokud je zvolen režim přeposílání paketů. Počet sekund udávajících čas generování, pokud je zvolen jiný než režim přeposílání paketů.
`-o <cestaKSouboru>`	Cesta k souboru, do kterého bude zapsán vytvořený model.
`-p <zvolenaHodnota>`	Maximální velikost payloadu aplikační vrstvy v jednom paketu.
`-q <cestaKSouboru>`	Cesta k souboru s modelem, pokud má být načten připravený model ze souboru.
`-r`	Pokud je přítomen, bude při stochastickém generování čten payload z připravených souborů.
`-s <zvolenaHodnota>`	Semínko náhodného generátoru.

`-t <zapouzdreni>`	Zapouzdření paketů použité v režimu přeposílání paketů.
`-w <IP:port>`	IP adresa a port webového serveru ve formátu `IP:port`.
`-cmac <MACadresa>`	MAC adresa, na kterou bude v režimu přeposílání paketů přepsána cílová MAC adresa.
`-smac <MACadresa>`	MAC adresa, na kterou bude v režimu přeposílání paketů přepsána zdrojová MAC adresa.

B.2.3 Konfigurace spouštění

B.2.3.1 Režim přeposílání paketů

V tomto režimu je nutné poskytnout programu pcap soubor se zachycenými pakety, které jsou následně extrahovány a přepsány dle zadaných požadavků. Přepisovány jsou IP adresy všech serverů na zvolenou adresu serveru a IP adresy všech klientů na zvolenou IP adresu klienta. Pro přepsání MAC adres je nutné zadat nové adresy klienta i serveru, v opačném případě k přepsání nedojde.

Porty přepisovány nejsou, nicméně kvůli jednotnému vstupnímu formátu je nutné zadat je stejně jako v ostatních režimech generování.

Možné nastavení vstupních parametrů generátoru je na následujícím výpise. Stranu klienta, která by provoz přijímala, v tomto režimu spouštět není nutné.

Listing B.1: Možné nastavení vstupních parametrů pro spuštění generátoru v režimu přeposílání paketů

```
sudo ./../../src/impl/generator/generator.py -b server -
    m packets -t ether -i enp37s0 -w 192.168.20.30:80 -c
     192.168.1.41:5987 -smac 00:12:13:14:15:16 -cmac
     00:22:33:44:55:66 -f ../real-traffic-for-model-
     creation.pcap -k -n 5 -d ./out/
```

B.2.3.2 Režim posílání souborů

Stejně jako v předchozím režimu je i zde nutné vždy poskytnout pcap soubor s dříve zachycenými pakety. Všechny objekty nalezené v tomto souboru jsou extrahovány a následně využity pro přeposílání. V tomto režimu je generována oboustranná komunikace, tudíž je nutné spustit skript jak na straně serveru, tak i na straně klienta. Vzhledem k mechanismu tvorby signálního kanálu je doporučeno nejdříve spustit stranu klienta. Možná nastavení vstupních parametrů skriptu pro server i klienta jsou na následujících výpisech.

Listing B.2: Možné nastavení vstupních parametrů pro spuštění generátoru na straně klienta v režimu posílání souborů

```
sudo ./generator.py -b client -m files -c
    192.168.1.41:11111
```

Listing B.3: Možné nastavení vstupních parametrů pro spuštění generátoru na straně serveru v režimu posílání souborů

```
sudo ./../../src/impl/generator/generator.py -b server -
    m files -w 192.168.1.107:80 -c 192.168.1.41:11111 -f
    ../real-traffic-for-model-creation.pcap -o
    createdModel.csv -d ./out/ -n 600
```

B.2.3.3 Režim stochastického generování

Tento režim je jako jediný možné spustit i bez vstupního pcap souboru, který je nahrazen souborem s již vytvořeným modelem, který je pouze načten do vnitřních programových struktur. Ostatní možnosti spouštění jsou shodné s režimem posílání souborů. Možné konfigurace vstupních parametrů pro spuštění serveru jsou uvedeny v následujících výpisech, nastavení parametrů pro spuštění klienta je shodné jako v režimu posílání souborů.

Listing B.4: Možné nastavení vstupních parametrů pro spuštění generátoru na straně serveru v režimu stochastického generování se vstupním pcap souborem

```
sudo ./../../src/impl/generator/generator.py -b server -
    m stochastic -w 192.168.1.107:80 -c
    192.168.1.41:11111 -f ../real-traffic-for-model-
    creation.pcap -o createdModel.csv -d ./out/ -n 3600
```

Listing B.5: Možné nastavení vstupních parametrů pro spuštění generátoru na straně serveru v režimu stochastického generování bez vstupního pcap souboru

```
sudo ./../../src/impl/generator/generator.py -b server -
    m stochastic -w 192.168.1.107:80 -c
    192.168.1.41:11111 -q ./out/createdModel.csv -d ./
    out2/ -n 3600
```

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `doc` ..... dokumentace zdrojových kódů
  - `generator` ..... dokumentace zdrojových kódů generátoru
  - `pcapAnalyzer` ..... dokumentace zdrojových kódů pcapAnalyzeru
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `generator` ..... generátor síťového provozu
    - `pcapAnalyzer` ..... analyzátor pcap souboru
    - `r` ..... výpočetní skripty v jazyku R
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `test` ..... testovací skripty a výstupy
  - `generovana-data-bez-posilani` ..... testy k ověření modelu
  - `rezim-posilani-souboru` .... testy k ověření režimu posílání souborů
  - `rezim-preposilani-paketu` testy k ověření režimu přeposílání paketů
  - `rezim-stochasticke-generovani` ..... testy k ověření režimu stochastického generování
- `text` ..... text práce
  - `DP_Karafiat_Jan_2016.pdf` ..... text práce ve formátu PDF