Sem vložte zadání Vaší práce.

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

Diplomová práce

Analyzátor protokolu CAT-TP

Bc. Zdeněk Pešek

Vedoucí práce: Ing. Jiří Dostál

4. května 2015

Poděkování

Chtěl by poděkovat své rodině za podporu po celou dobu mého studia. Dále také vedoucímu práce Ing. Jiřímu Dostálovi za cenné rady a pomoc při tvorbě 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 4. května 2015 .....

Odkaz na tuto práci

Pešek, Zdeněk. *Analyzátor protokolu CAT-TP.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce se zabývá transportním protokolem CAT-TP, který je používán v oblasti mobilních sítí. V textu práce lze naleznout základní informace o tomto protokolu a popis implementace praktikté části. Ta je tvořena rozšířením síťového analyzátoru o podporu protokolu CAT-TP a simulačním prostředím. Zmíněné rozšíření slouží k detailní analýze zachycených dat protokolu CAT-TP.

Klíčová slova CAT-TP, Wireshark, disektor, Scapy

Abstract

This thesis deals with a CAT-TP communication protocol, which is used in the field of mobile networks. Text consists of a basic information about this protocol and description of the practical part. This implementation part contains plugin for the network analyzer and simulation environment. The plugin allows a detailed analysis of the captured CAT-TP data.

Keywords CAT-TP, Wireshark, dissector, Scapy

Obsah

Seznam obrázků

Úvod

Počítačovými sítěmi v dnešní době prochází obrovské množství dat. Objem přenesených dat se ale i nadále zvětšuje[3], což znesnadňuje jejich interpretaci a dedukci širších souvislostí. Velmi snadno můžeme velké množství proudících dat zaznamenat. Mají ale zaznamenaná data nějakou hodnotu? Odpověď samozřejmě není jednoznačná, ale v oblasti síťové komunikace dává většinou pravou hodnotu datům až jejich interpretace a vyvození souvislostí.

Jedním ze základních principů síťové architektury je její rozdělení do vrstev. Konkrétní počet a definice záleží na specifikaci (především ISO/OSI[4] nebo také zjednodušený model čtyř vrstev[5]), ale princip zůstává stejný. Zatím co u dat se v průběhu vývoje technologií stále zvětšuje jejich objem přenesený po síti, u vrstev dochází k odlišnému vývoji. S rozmachem nových technologií se objevují nároky, které žádný z existujících protokolů nesplňuje a je třeba implementovat nový. Díky zmíněnému principu vrstvení je možné vytvářet velmi komplexní síťové architektury, které může být následně složité dekomponovat zpět do původních vrstev. Právě seznámení se a dekompozice síťového provozu, který obsahuje vrstvu protokolu vzniklou na základě potřeby naplnit nové požadavky, je náplní této práce.

Kapitola 1

Motivace a cíle práce

V období, kdy jsem si vybíral téma mé závěrečné práce se na půdě naší fakulty konal Den spolupráce s průmyslem. Jedním z důvodů, proč jsem akci navštívil, byl i příslib zajímavého zadání, které nalezne po dokončení i další uplatnění v praxi. Se zástupci jedné z firem jsem navázal kontakt a díky vstřícnému jednání byla rychle na světě první verze zadání této práce.

1.1 Motivace

Každý vývojář se snaží pracovat během procesu psaní kódu co nejefektivněji. Velkou měrou k efektivnímu vývoji přispívá prostředí pro vývoj a testování. Hlavní motivací této práce je zjednodušit a urychlit vyvíjení zdrojového kódu vývojářům, kteří pracují s protokolem CAT-TP na denní bázi. Výsledné řešení, které splňuje požadované nároky, by mělo mít nezanedbatelný pozitivní vliv na produktivitu programátorů pracujících ve zmíněném odvětví.

1.2 Cíle

Na úvod by měla tato práce poskytnout čtenáři ucelený přehled protokolu CAT-TP. Míra detailu bude stanovena tak, aby mohl poučený odborník, který zatím nemá znalosti o tomto konkrétním protokolu, získat dostatečné znalosti k porozumění základních principů fungování tohoto protokolu.

Poté bude zhodnocen soubor dostupných síťových analyzátorů dle předem stanovených kritérií. U vybraného analyzátoru, který ve výběru zvítězil, bude zhodnoceno z jakých důvodů se tak stalo a to zejména s přihlédnutím ke stanoveným kritériím.

Čtenář bude seznámen s vybraným síťovým analyzátorem. Nejprve v obecné rovině a následně i detailně v míře potřebné pro porozumění nutným úpravám, které je nutné provést pro rozšíření podpory o nový protokol.

Následovat bude pojednání o krocích, které bylo nutné provést při implementaci rozšíření. Přidanou hodnotou této části pro budoucí čtenáře bude upozornění na problematická místa implementace, která mohou být pro nepoučeného programátora obtížná pro implementaci. V neposlední řadě se bude tato část práce zabývat také popisem pokročilých vlastností analyzátoru, které rozšíření bude využívat.

Implementované řešení bude otestováno v připraveném simulátoru, který bude popsán v samostatné kapitole. Pro ilustraci funkce naprogramovaného rozšíření i simulátoru budou přiloženy snímky obrazovky.

Kapitola 2

Protokol CAT-TP

Protokol je specifikován normou ETSI TS 102 127[1], která definuje jeho základní funkcionalitu, stavy, strukturu paketů a další. Naopak konkrétní implementace, API a definice vyšších vrstev(zejména bezpečnostní funkcionalita) již není součástí tohoto dokumentu. Kromě tohoto zdroje jsem také čerpal informace z interní prezentace[6] společnosti Gemalto, s.r.o., která ji využívá pro výuku základů protokolu BIP[7] a souvisejícího CAT-TP.

2.1 Představení protokolu

SIM karta, která je přítomna v každém běžném mobilním telefonu může v určitých případech vyžadovat přístup k datům z externího zdroje, který je umístěn v rámci infrastruktury poskytovatele mobilních služeb. V letech minulých se k těmto účelům využíval SMS kanál(tzv. OTA over SMS), který byl omezen velikostí jedné SMS 140 bajtů[8]. Doba ale pokročila a přenášená data nabývají na objemu, pro který již není SMS kanál dostatečně široký. Řešením je použití transportního protokolu CAT-TP, který podporuje výrazně vyšší propustnost a využívá jiného kanálu(tzv. OTA over IP) přes GPRS nebo 3G připojení, který zvýšil omezení na 1472 bajtů pro jednu zprávu.

Stěžejní scénáře použití jsou:

- Stahování aplikací(apletů) na SIM
- Aktualizace souborů na SIM
- Poskytnutí konektivity pro aplikace na SIM kartě
- Obecně vzdálená správa obsahu SIM karty

2.1.1 Segmentace

Vzhledem k omezené velikosti PDU bufferu na straně SIM karty bylo nutné zavést mechanismus pro umožnění zasílání zpráv, které ho velikostí přesahují.

V takové situaci je nezbytné využít rozdělení dat na více částí a zaslat je příjemci segmentované.

V hlavičce protokolu můžeme naleznout jeden bit, který signalizuje segmentaci. Zapsání hodnoty 1 na tuto pozici v hlavičce znamená paket se segmentací. Logicky 0 charakterizuje paket bez segmentace. Pozor si ale musíme dát na jednu výjimku. Pokud se jedná o paket, který ukončuje řadu segmentovaných zpráv, je tento bit nastavený také na 0.

2.1.2 Pozitivní potvrzování

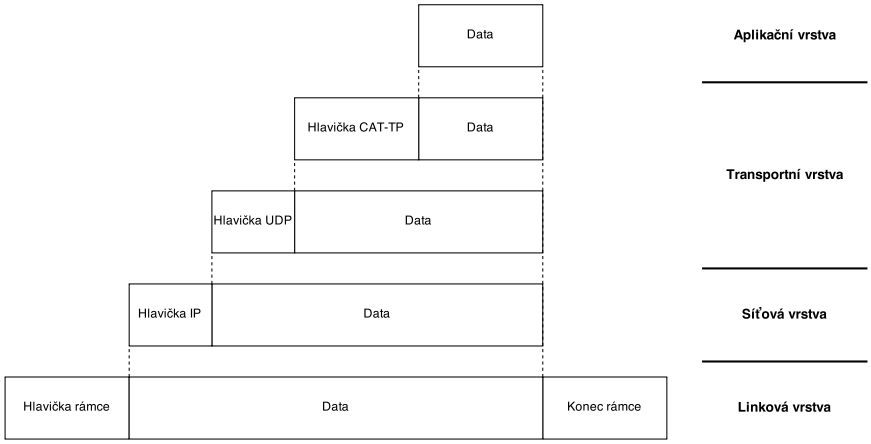
Námi popisovaný transportní protokol podporuje zasílání dat po spolehlivém, ale i nespolehlivém kanále, který může způsobit ztrátu jednoho nebo i více paketů. Pro překonání chybových stavů, které jsou vyvolané zmíněnou chybou na lince, je protokol CAT-TP vybaven pozitivním potvrzováním. To znamená, že všechny pakety, které vyžadují potvrzení (pakety typu NUL, SYN a ACK s daty) inkrementují sekvenční číslo a odesílatel čeká na potvrzení jejich přijetí protistranou. Standardně je tomu tak učiněno odesláním ACK paketu s příslušně nastaveným polem potvrzovacího čísla. Pokud však používáme zasílání s velikostí okna větší než jedna, může nastat situace, kdy příjemce obdrží pakety mimo pořadí vlivem zpoždění nebo chyby na komunikačním kanálu. Na takový případ reaguje zasláním paketu EACK, který je blíže popsán později.

2.1.3 Timeout a znovuzasílání

Po odeslání paketu, který vyžaduje potvrzení od protistrany, je inicializován časovač na předem definovanou hodnotu a začne jeho odpočet. Ten je zastaven v případě přijetí příslušného potvrzovacího paketu. Pokud do vypršení časového limitu potvrzení není přijato, dojde k pokusu o znovuzaslání a časovač je restartován. Počet pokusů o doručení paketu je omezen konstantou, která je nastavitelná administrátorem. Pokud je tato hodnota překročena, dojde k ukončení komunikace zasláním paketu RST.

2.1.4 Variabilní velikost okna

Ve většině případů se pro komunikaci protokolem CAT-TP používá velikost okna 1. To znamená, že každý odeslaný datový segment musí být potvrzen před odesláním segmentu s následujícím sekvenčním číslem. Je tu ale i možnost zasílat více paketů najednou bez čekání na potvrzení. Velikost okna je pro tyto případy typicky rovna podílu maximální velikosti SDU a PDU. Pokročilou vlastností tohoto protokolu je možnost měnit velikost okna v průběhu komunikace a to oběma směry.



Obrázek 2.1: Zapouzdření protokolu

2.2 Zapouzdření

Přiložený obrázek2.1 ilustruje typické zapouzdření CAT-TP protokolu v průběhu komunikace. CAT-TP vrstva může být zapouzdřena do dvou různých protokolů a to v závislosti na místě, ve kterém se paket aktuálně nachází v průběhu komunikace. Při cestě paketu sítí od vzdáleného serveru až po mobilní telefon je využíván standardní UDP/IP protokol, ve kterém je zapouzdřen CAT-TP protokol. Pro podrobnější informace o protokolu IP doporučuji prostudovat RFC specifikaci protokolu IP[9] případně UDP[10] Typickým UDP portem, který je spjatý s CAT-TP je *6000*. Pro následnou komunikaci mobilního telefonu se SIM kartou se využívá protokolu BIP. Ten ale není pro potřeby této práce relevantní. Případné další informace o BIP je možné získat z technické specifikace[7].

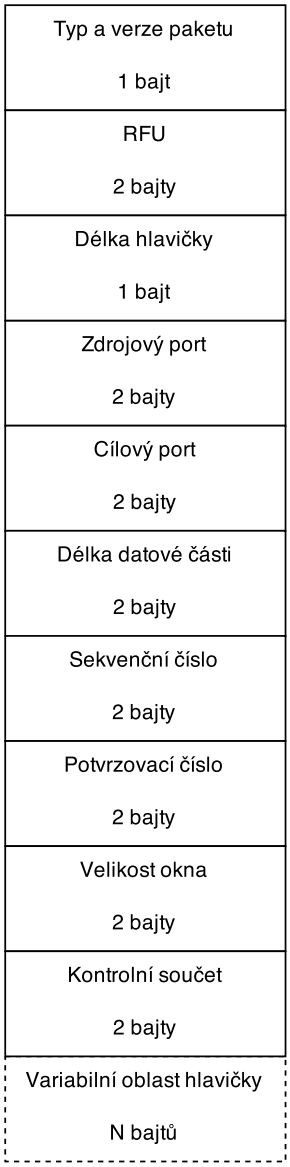
Lze se také setkat se zapouzdřením CAT-TP do protokolu TCP[11], ale jedná se o velmi neobvyklý jev.

2.3 Struktura paketů

Pro potřeby této práce definujme, že jednotlivá pole jsou ukládána metodou Big-endian tj. na nejnižší místo v paměti je zapisován nejvýznamnější bit.

2.3.1 Společná hlavička

Všechny validní pakety protokolu CAT-TP musí obsahovat společnou část, kterou nazýváme hlavičkou. Ta je umístěna na samém počátku paketu a nese



Obrázek 2.2: Hlavička protokolu CAT-TP

Obrázek 2.3: Příznaky v prvním bajtu hlavičky

Parameters⇨ PDU types ⇩	SYN	ACK	EACK	RST	NUL	SEG	Seq Nb	Ack Nb	Optional Header Area	Data
SYN PDU	= 1	= 0	= 0	= 0	= 0	= 0	SND_INITIAL_SEQ_NB	Insignificant	Max PDU Size Max SDU Size Identification	No Data
SYN/ACK PDU	= 1	= 1	= 0	= 0	= 0	= 0	SND_INITIAL_SEQ_NB	RCV_INITIAL_SEQ_NB	Max PDU Size Max SDU Size Identification	No Data
ACK PDU	= 0	= 1	= 0	= 0	= 0	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	None	No data
ACK PDU + un-segmented data	= 0	= 1	= 0	= 0	= 0	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	None	SDU
ACK PDU + segmented data	= 0	= 1	= 0	= 0	= 0	= 1 = 0 if last segmented PDU	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	None	SDU Segment
ACK/EACK PDU	= 0	= 1	= 1	= 0	= 0	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	RCV_OUT_OF_SEQ_PDU_SEQ_NBs	No data
ACK/EACK PDU + un-segmented data	= 0	= 1	= 1	= 0	= 0	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	RCV_OUT_OF_SEQ_PDU_SEQ_NBs	SDU
RST PDU	= 0	= 0	= 0	= 1	= 0	= 0	SND_NEXT_SEQ_NB	Insignificant	None	No data
RST/ACK PDU	= 0	= 1	= 0	= 1	= 0	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	None	No data
NUL/ACK PDU	= 0	= 1	= 0	= 0	= 1	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	None	No data
NUL/ACK/EACK PDU	= 0	= 1	= 1	= 0	= 1	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	RCV_OUT_OF_SEQ_PDU_SEQ_NBs	No data

Obrázek 2.4: Typy paketů, převzato z[1]

základní informace, bez kterých by nebylo možné získat kontext tohoto paketu v rámci celé komunikace.

2.3.1.1 Flagy a verze

První bajt paketu je založen na principu flagování tj. musíme ho porovnat s maskou, aby jsme získali informaci, kterou nese. Rozvržení flagů v rámci prvního bajtu hlavičky ilustruje přiložený obrázek2.3. V případě námi analyzovaného protokolu můžeme tento bajt rozdělit na dvě části. První část, která je vymezena dvěma bity, nese informaci o verzi protokolu CAT-TP a je podstatná jen pro pakety typu SYN během inicializace spojení. V současném standardu je definována jako 0 a novější verze zatím není publikována. V budoucnosti se ale tato skutečnost samozřejmě může změnit. Vždy se použije nižší verze z obou získaných hodnot od komunikujících stran a je tedy vyžadováno zachovat kompatibilitu se všemi staršími verzemi.

Druhá část tohoto pole deklaruje typ celého paketu. Je potřeba mít na paměti, že je povolené vytvářet i kombinace typů paketů. Přiložená tabulka2.4 poskytuje přehled možných kombinací, které odpovídají standardu.

2.3.1.2 RFU

Jak bylo zmíněno, protokol CAT-TP se nyní nachází v první verzi (nese označení 0). Pro případné budoucí potřeby je možné zvýšit číslo verze. Pokud bude potřeba, je možné využít i 2 bajty paměťového místa, které jsou nyní dle specifikace nevyužité (Reserved for future use) a jejich hodnota by měla vždy být inicializovaná na 0.

2.3.1.3 Délka hlavičky

Toto pole udává délku hlavičky příslušného paketu. Její délka je důležitá pro oddělení hlavičky a datové části, kterou tento paket může nést. Základní velikost hlavičky je 18 bajtů. Tato hodnota může být i větší a to v několika případech, kdy je využita variabilní sekce.

2.3.1.4 Porty

Stejně jako v běžně užívaných transportních protokolech TCP/UDP i v případě CAT-TP je potřeba definovat koncové uzly, které spolu komunikují. Dále je tato hodnota nezbytná pro úplnou identifikaci komunikující strany. Jednoznačná identifikace se skládá z adres nižších vrstev (typicky IP adresa a UDP port) a v poslední fázi i tohoto CAT-TP portu. Kombinace identifikátoru obou komunikujících stran slouží k rozpoznání jednotlivých relací (sessions).

2.3.1.5 Délka dat

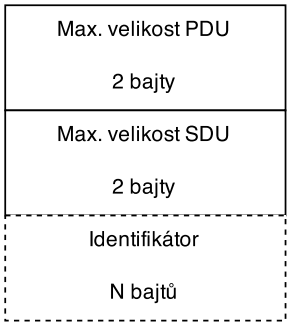
Pakety, které nesou data, mají v tomto poli zapsanou jejich délku (bez hlavičky). Tato hodnota je poté použita k jejich extrakci z paketu a dalšímu použití ve vyšších vrstvách sítě, popřípadě až v aplikacích, které protokolu využily pro přenos dat.

2.3.1.6 Sekvenční číslo

Toto číslo je inicializováno na začátku komunikace na libovolnou hodnotu, kterou lze uložit do dostupných dvou bajtů. Po překročení maximální hodnoty se čísluje opět od nuly a je tedy cyklické. V průběhu komunikace se postupně inkrementuje a to, pokud je vyžadováno potvrzení přijetí právě odesílaného paketu.

2.3.1.7 Potvrzovací číslo

Potvrzovací číslo udává sekvenční číslo paketu, který byl v sekvenci obdržen jako poslední. Pro toto pole platí převážná část předchozího odstavce, ale na rozdíl od sekvenčního čísla je pevně spojeno s nastavením příznaku ACK. Po úspěšném otevření komunikačního kanálu nesou tento příznak všechny pakety až do fáze ukončování spojení.



Obrázek 2.5: Pole paketu typu SYN

2.3.1.8 Velikost okna

Protokol umožňuje variabilně měnit velikost okna pro odesílané pakety. Toto nastavení je přenášeno právě v této části hlavičky a specifikuje počet paketů, které je možné odeslat bez čekání na jednotlivá potvrzení přijetí. Posledním sekvenčním číslem, které je možné odesílat je poslední přijaté potvrzovací číslo + velikost okna. Pokud strana, která přijímá data obdrží v rámci okna pakety mimo pořadí, odesílá o tom informaci odesílateli paketem typu EACK. Specifikací je povolená i hodnota velikosti okna 0. To ale znamená okamžité pozastavení spojení a čekání na opětovné navýšení hodnoty na větší než nulovou pomocí paketu typu NUL. K takové situaci může dojít, pokud má příjemce limitované zdroje (zejména paměť) a není již nadále schopen přijmout další data.

2.3.1.9 Kontrolní součet

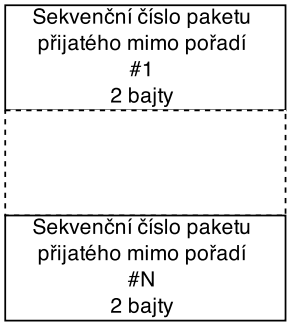
Toto pole obsahuje kontrolní součet, který slouží k detekci poškozených paketů a je počítán z dat obsažených v hlavičce protokolu bez datové části. Samotný algoritmus je definován v RFC[11] a délka jeho výstupu jsou 2 bajty. Pole kontrolního součtu je součástí dat hlavičky, ze kterých se samotná hodnota počítá. Dle specifikace je po dobu jejího počítání toto pole nastaveno na 0.

2.3.1.10 Variabilní sekce hlavičky

Tato sekce hlavičky nemá pevně stanovenou délku a může být přítomna jen u paketů typu SYN, EACK a RST.

2.3.2 SYN

Tento paket, který je charakteristický pro fázi zahájení komunikace, udává počáteční sekvenční číslo, které je následně dle pravidel inkrementováno. Dále nese inicializační údaje koncového uzlu, které jsou umístěny ve variabilní sekci



Obrázek 2.6: Pole paketu typu EACK

hlavičky a jsou její nedělitelnou součástí. To v důsledku znamená zahrnutí délky této části do velikosti hlavičky a také nastavení pole délky dat na nulu.

Údaje koncového uzlu2.5 zahrnují dvě povinná a jedno volitelné pole. Vyžadována je maximální délka PDU, které je uzel schopný přijmout. Tato délka zahrnuje celý paket včetně CAT-TP hlavičky a případných dat, které nese. Druhým povinným polem je maximální délka SDU, což je celková délka přenášených dat po případném sestavení segmentů. Posledním polem je nepovinný identifikátor, který může určovat identitu uzlu. Specifikaci a nároky na toto pole ponechává standard na konkrétní implementaci. Jedinou povinnou částí pro toto pole je stanovení délky identifikátoru. Pokud je cílem odesílatele jej nespecifikovat, nastaví délku identifikátoru na 0.

V průběhu otevírání spojení jsou alokování buffery na na obou stranách. Na obou stranách je to zejména SDU (Service Data Unit) buffer, který uchovává odesílaná popř. přijímaná data. Na straně serveru nemá pevně stanovený limit, ale na straně SIM karty je zpravidla limitován. Na straně karty je to dále PDU (Protocol Data Unit) buffer, který určuje maximální velikost CAT-TP zprávy, kterou je možné přijmout.

2.3.3 ACK

Tento příznak paketu je nastaven v převážném množství paketů v rámci komunikace protokolu CAT-TP. Je často využíván k přenosu dat. Taková situace je signalizována nenulovou hodnotou v poli délky dat.

2.3.4 EACK

Pakety s tímto příznakem signalizují přijetí paketů se sekvenčním číslem, které bezprostředně nenavazuje na poslední úspěšně přijaté. Takové pakety označujeme jako přijeté mimo pořadí. Dle specifikace je tento příznak vždy kombinován s ACK, což umožňuje uvést i sekvenční číslo paketu, který byt posledním obdrženým v sekvenci. Ve variabilní části paketu se nacházejí sekvenční čísla

Tabulka 2.1: Přehled odůvodnění konce spojení

Kód (hex)	Důvod ukončení spojení
00	Normální ukončení
01	Navázání spojení se nezdařilo
02	Dočasně je nemožné navázat spojení
03	Požadovaný port není dostupný
04	Byl přijat neočekávaný paket
05	Byl překročen maximální počet pokusů o obnovení spojení
06	Nepodporovaná verze protokolu
07 a dále	Rezervováno pro budoucí použití

paketů, které byly přijaty mimo pořadí. Poznamenejme, že počet těchto dvoubajtových čísel není v hlavičce specifikován a je třeba jej odvodit od délky celé hlavičky.

2.3.5 RST

Teto příznak paketu signalizuje uzavření spojení. Důvodem může být regulérní uzavření spojení po úspěšném přenesení dat, ale i chyba v průběhu komunikace. Pro specifikaci důvodu ukončení spojení je ve variabilní části hlavičky vyhrazeno jednobajtové pole. Hodnoty, které jsou dle specifikace validní lze naleznout v tabulce2.1. Paket tohoto typu zásadně nenese data.

2.3.6 NUL

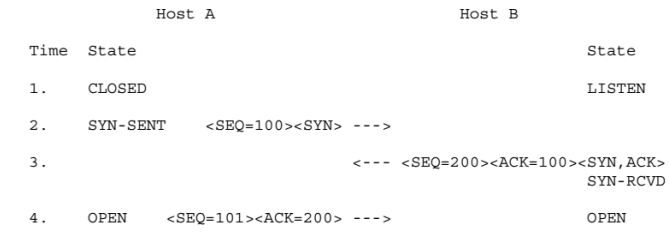
V průběhu komunikace mohou z různých důvodů vznikat prodlevy a následně je třeba detekovat zda využívané spojení s protistranou je stále aktivní a připravené pro přenos dat. K tomuto účelu slouží pakety s příznakem NUL. Můžeme je zjednodušeně považovat za PING druhé strany komunikace. Pokud je protistrana stále aktivní, reaguje zasláním paketu s příznakem ACK.

2.4 Typický scénář komunikace

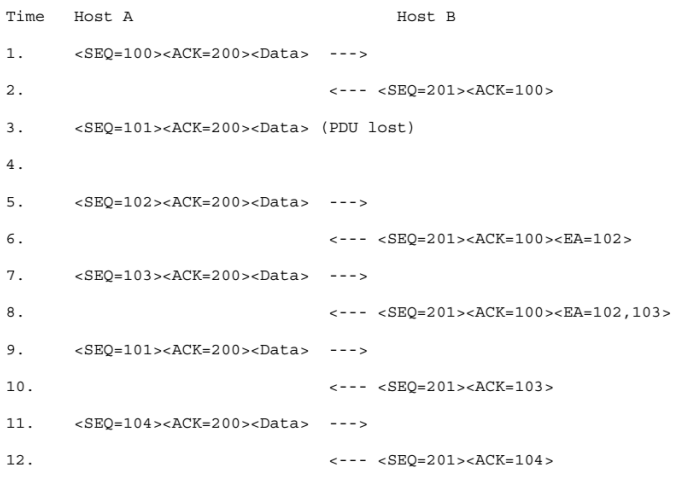
Nyní, když známe základní stavební kameny protokolu, se na něj podíváme z vyšší úrovně a představíme si kompletní příklad komunikace.

2.4.1 Navázání spojení

Před navázáním spojení se typicky jedna strana(server) nachází ve stavu naslouchání na daném portu a čeká na první paket od protistrany (klienta). Klient iniciuje tzv. třícestné navázání spojení (3-way handshake) zasláním paketu s příznakem SYN a nezbytnými náležitostmi. Server následně odpovídá a přijímá parametry klienta paketem typu SYN-ACK se svými parametry v



Obrázek 2.7: Diagram zahájení komunikace, převzato z[1]



Obrázek 2.8: Diagram přenosu dat, převzato z[1]

hlavičce. Poslední fází při otevírání spojení je odpověď klienta serveru typu ACK na znamení přijetí zaslaných parametrů od serveru. Úspěšným přijetím posledního paketu serverem je spojení navázané a je možné po nově vybudovaném kanálu začít posílat data.

2.4.2 Přenos dat

Datový přenos dodržuje principy popsané v předchozích částech práce a to zejména pozitivní potvrzování. Na obrázku2.8 můžeme vidět ukázku komunikace, která demonstruje chování protokolu v případě nepřijetí jednoho z paketů v rámci phyblivého okna okna. Paket odeslaný v čase 3 nedorazil k protistraně a to vede k přijetí dvou paketů (102 a 103) mimo pořadí. Dotčený Host B o této situaci informuje protistranu zasláním paketů typu EACK. To má za následek opětovné zaslání paketu se sekvenčním číslem 101, který je na druhý pokus úspěšně přijatý Hostem B a ten tedy opět začne potvrzovat pakety typu ACK, což značí přijímání paketů v sekvenci.

```
3.  CLOSED  <SEQ=101><RST> --->                    (abort)
4.  CLOSED                                         CLOSED
```

Obrázek 2.9: Diagram ukončení komunikace, převzato z[1]

2.4.3 Ukončení spojení

Ukončení komunikace je velmi jednoduché. Pouze jedno úspěšné přijetí paketu RST stačí k ukončení spojení a uvolnění alokovaných zdrojů na obou stranách, což dokládá obrázek2.9.

Kapitola 3

Volba analyzátoru

3.1 Úvod

Zadání této práce pochází z potřeby zákazníka získat přehledné a strukturované informace o zachyceném nebo právě probíhajícím síťovém provozu specializovaného protokolu. Existuje řada přístupů, které tento problém řeší a to s různou mírou efektivity, časové náročnosti, robustnosti, kvality grafického zpracování atd. Na jedné straně stojí možnost začít tvořit jednoúčelovou aplikaci šitou na míru daným požadavkům. Na straně druhé by bylo standardně možné použít již existující řešení, které by mohlo znamenat i vynaložení finančních prostředků v případě zvolení komerční aplikace. Po setkání se zákazníkem a prodiskutování očekávaných vlastností výsledného řešení bylo rozhodnuto o zvolení varianty, která se nachází mezi těmito mezními možnostmi. Došli jsme k závěru, že nejvhodnějším postupem je rozšíření funkcionality již existujícího softwaru.

3.2 Definice

Paketovým analyzátorem budeme pro potřeby této práce považovat softwarové řešení, které ulehčuje uživateli práci s daty, které aktuálně procházejí sítí nebo byly již dříve zaznamenány a nyní jsou zpětně použity k provedení analýzy.

3.3 Přehlednost prezentace

Pojem ulehčení práce může mít v tomto kontextu vícero podob. V první řadě se jedná o přehlednou reprezentaci datových paketů. Velké množství binárních dat, které se mihotají na obrazovce jistě přehledné není a proto je potřeba zamyslet se na lidsky čitelnou podobu, která nemusí být nutně v grafické podobě. Mnohé nástroje příkazové řádky dokazují, že i v tomto prostředí je možné předat uživateli velkou informační hodnotu. Je ale třeba poznamenat, že po-

Tabulka 3.1: Porovnání síťových analyzátorů

Název	Win	Linux	Solaris	Licence	GUI	Pluginy
Kismet[12]	✓	✓	✗	✓	✗	✓
Microsoft N. Monitor[13]	✓	✗	✗	✗	✓	✗
ngrep[14]	✓	✓	✓	✓	✗	✗
tcpdump[15]	✓	✓	✓	✓	✗	✗
Wireshark[2]	✓	✓	✓	✓	✓	✓

užívání těchto nástrojů je především doménou zkušenějších administrátorů. Pro méně zkušené uživatele je jistě příjemnější setkání s grafickou nástavbou než strohou příkazovou řádkou. Vzhledem k množství zachycených paketů je jedním ze základních požadavků propracovaná funkce filtrování. V té nejzákladnější variantě je často nabízena možnost specifikovat určitý bajt paketu, který odpovídá předepsanému pravidlu. Takové pojetí filtrování, ale naráží na několik překážek. V první řadě je to princip enkapsulace - hierarchie jednotlivých vrstev paketu. Filtr, který je zmíněného typu, přestane být po změně jedné vrstvy na nižší úrovni funkční. Nejen, že ve výsledcích takového filtrování s největší pravděpodobností nenalezneme požadované pakety, ale dokonce můžeme mylně považovat falešně vyfiltrované pakety za relevantní. Taková situace ilustruje důležitost filtrování po vrstvách dle jejich specifických atributů.

3.4 Požadavky

Při výběru paketového analyzátoru, který nám poslouží jako základ pro rozšíření funkčnosti, jsme vázáni několika požadavky zákazníka. Výsledné řešení by mělo být použitelné na co největším počtu platforem (minimálně na OS Linux,Windows a Solaris). Druhým zásadním požadavkem je vhodná licence zvoleného softwaru, která nám umožní upravit jej a šířit bez porušení zákona. Třetí požadavek nám ukládá zabývat se samotnou možností rozšiřitelnosti o požadovanou funkcionalitu, tedy zda je taková úprava v možnostech zvolené aplikace. Posledním hlediskem je uživatelská přívětivost. Předpokládá se, že řešení bude ke své práci používat i uživatel bez znalosti obsluhy z příkazové řádky.

3.5 Vyhodnocení

Po zvážení uvedených nároků3.1 se ukázal být nejvhodnější volbou nástroj Wireshark, který nyní zhodnotíme z pohledu uvedených kritérií.

Již první omezení značně zúžilo výběr vhodného kandidáta, neboť podpora pro tři zmíněné operační systémy se vyskytuje u minima dostupných nástrojů. Wireshark, jako jeden z mála, tento požadavek splňuje. Dále je distribuován

pod licencí GNU GPL, která nebrání potřebným úpravám a zveřejnění výsledné aplikace. Ba právě naopak, volná dostupnost vytvořených zdrojových kódů je její podmínkou. Architektura aplikace dle dostupné dokumentace a struktury zdrojového kódu bez pochyb umožňuje provedení plánovaných úprav a nebude pro realizaci této práce překážkou. Až poslední omezení však staví nástroj Wireshark před ostatní možné kandidáty. Právě uživatelským komfortem se tato aplikace vyznačuje a odlišuje od všech ostatních. Nabízí přehlednou prezentaci zapouzdření paketů až po tu nejnižší vrstvu (pokud je podporovaná) a zároveň nepožaduje po uživateli žádné přehnané znalosti.

Ačkoliv nabízí i použití z příkazové řádky, jedná se především o grafickou nástavbu, diky které se tento nástroj proslavil a slaví úspěchy i po více než patnácti letech od první verze. Díky této dlouhé historii si získal důvěru i ve výpočetních stanicích mnoha profesionálů uvnitř firem, pro které by měl být výsledek práce určen. Výsledné řešení tedy nebude znamenat potřebu přecházet na nový a neotestovaný produkt, ale rozšíření schopností již používaného nástroje.

Kapitola 4

Wireshark

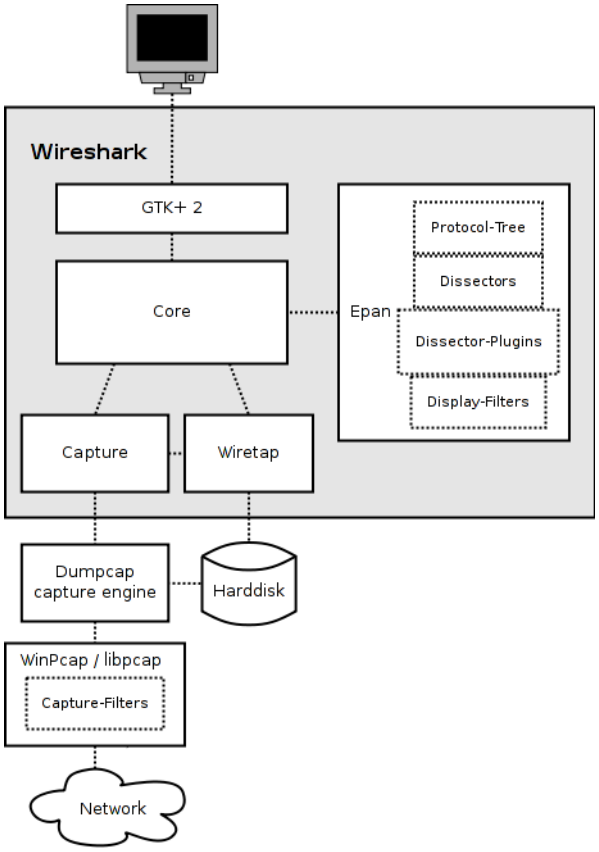
Primárním zdrojem informací pro tuto kapitolu byla oficiální dokumentace[2] programu a to uživatelská i vývojářská.

4.1 Úvod

V roce 1997, ve kterém pracoval tvůrce Wiresharku pro malého poskytovatele internetového připojení v USA, byly již dostupné komerční protokolové analyzátory. Ty si však malá firma nemohla finančně dovolit. Druhým zásadní problémem byla omezená nabídka podporovaných platforem, která nezahrnovala Linux ani Solaris, které byly v dotyčné firmě používané. Z těchto důvodů dospěl Gerald Combs k názoru, že bude nejlepší vytvořit si vlastní nástroj, který bude tyto nároky splňovat. Jeho snažení dosáhlo prvních hmatatelných úspěchů v roce 1998, kdy představil světu první verzi svého nástroje pod názvem Ethereal. První verze postrádala mnohé pokročilé funkce placených alternativ, ale udala směr budoucího směřování v podobě zdrama dostuplného a multiplatformního síťového analyzátoru. Projekt se setkal s velmi pozitivní odezvou a díky rozsáhlé podpoře komunity vývojářů a jiné odborné veřejnosti mohl pokračovat v dalším rozvoji. Později byl z právních důvodů změněn název na současný - Wireshark. Gerald Combs se i v současnosti podílí na udržování kódu produktu a vydávání nových verzí nástroje. Aktuální kód Wiresharku je výsledkem spolupráce více než šestiset vývojářu z celého světa.

4.2 Struktura

Strukturu Wiresharku lze popsat následujícím diagramem4.1, který ilustruje rozdělení do několika funkčních bloků.



Obrázek 4.1: Architektura nástroje Wireshark, převzato z[2]

4.2.0.1 Jádro

Základní částí je jádro programu, které propojuje ostatní funkční moduly, umožňuje jejich komunikaci a spolupráci. Z dalších funkčních modulů nejprve jmenujme knihovnu Wiretap, která zajišťuje čtení a zápis záznamů síťového provozu na disk. Podporována je celá řada formátů. Dále se na diagramu nachází modul pro záznam síťového provozu z dostupných síťových zařízení. Neodmyslitelnou součástí struktury je modul grafického rozhraní, který zajišťuje interakci s uživatelem.

4.2.0.2 EPAN

Zřejmě nejzásadnější částí diagramu je modul zvaný EPAN, což je zkratka pro "Ethereal Packet ANalyzer". Jak již název napovídá, jedná se o část programů, která se stará o analýzu paketů, a to z obou zmíněných zdrojů - souboru na disku nebo "živých" dat ze síťového rozhraní. EPAN je je natolik komplexní součástí Wiresharku, že dává smysl nezastavit se na této úrovni detailu, ale

pokračovat v analýze o úroveň dál. EPAN se se dále skládá z protokolového stromu, který udává cestu po disektorech. které má Wireshark k dispozici a odpovídají zapouzdření vrstev daného protokolu. Za touto částí již operují jednotlivé disektory, které se postarají o svou část dat v paketu ("payload") a předají níže ležící vrstvu dalšímu disektoru s ohledem na protokolový strom. Tyto disektory mohou být s programem zkompilovány a stávají se tak jeho nedílnou součástí nebo lze využít druhé možnosti, tj. zkompilování samostatné knihovny a přidat je k Wiresharku ve formě rozšíření ("plugin"). Poslední částí jsou filtry pro zobrazování paketů se specifickými charakteristikami.

4.2.0.3 Dumpcap a libpcap

Mimo samotný Wireshark můžeme na uvedeném diagramu naleznout další dvě nepostradatelné části. První se nazývá Dumpcap, což je modul, který jako jedný běží s právy administrátora ("elevated privs"). Jeho úkolem je logika zachytávání paketů. Poslední částí struktury, kterou jsme zatím nepopsali je knihovna[15], která bezprostředně komunikuje se síťovým rozhraním při zachytávání síťového provozu. Její instalace je součástí instalačního balíku Wiresharku a je nezbytní pro jeho plnou funkcionalitu. Název knihovny se liší dle používané platformy. Na linuxových operačních systémech se jedná o libpcap, v prípadě systému Windows je to WinPcap. Tento modul také obsahuje podporu pro filtrování. Jedná se však o filtrování na té nejnižší úrovni, které může eliminovat značné množství paketů a udržet tak průchodnost při vysoké úrovni toku dat do síťového zařízení. Pokročilejší filtrování dat na úrovni jednotlivých vrstev se odehrává v modulu EPAN, který již byl zmíněn.

4.3 Disekce paketů

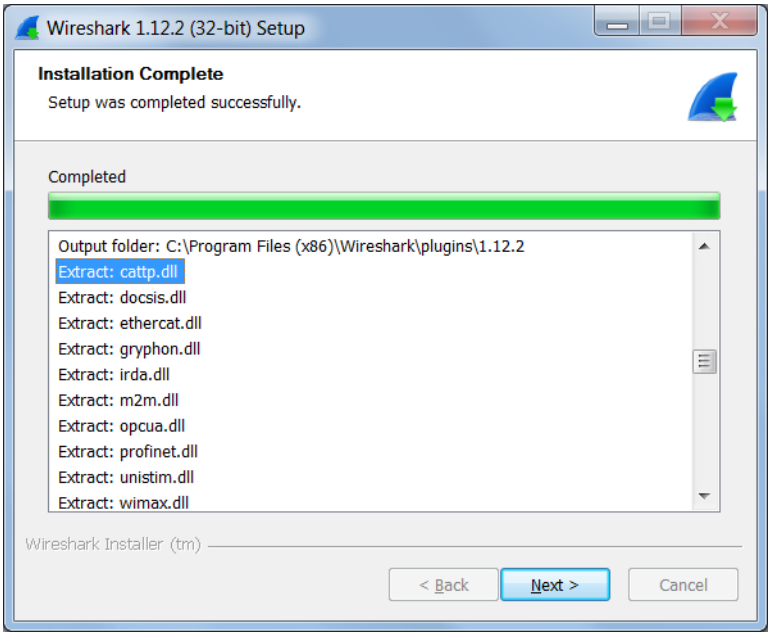
Disekce paketů je proces dekódování části paketu, která náleží určitému protokolu, disektorem, který je určen k tomu určen. Volání disektorů je rekurzivního charakteru. Každý zanalyzuje svojí část paketu a dokud zbývají data, která ještě analýze podrobena nebyla, předá tuto část dalšímu disektoru.

Kapitola 5

Rozšíření funkčnosti síťového analyzátoru

Rozšířit analyzátor Wireshark o podporu nového protokolu lze dvěma způsoby. Disektory, které slouží k analýze často používaných protokolů, prošly testovací fází a jsou tedy stabilní, jsou umístěny ve složce *epan/dissectors* a při kompilaci jsou zařazeny přímo do programu. Rozšíření, která procházejí vývojem jsou umístěny ve složce *plugins* a program je při startu nahrává z předem definované složky. Nevýhodou je mírné snížení výkonnosti oproti zařazení přímo do programu, ale to je kompenzováno výrazně rychlejší kompilací, kdy není potřeba kompilovat celý projekt, ale jen konkrétní rozšíření. Naopak výhodou je snadná přenositelnost, která umožňuje získat požadovanou funkcionalitu pouhým překopírováním knihovny do správného adresáře s pluginy Wiresharku.

Pluginy jsou, stejně jako celý Wireshark, psány v jazyce C s knihovnou GLib[16]. Tato knmihovna nabízí některé pokročilé funkce (pokročilé datové typy, kontejnery, ...), které jsou nezbytné pro implementaci pokročilejších vlastností disektoru a zásadním způsobem ulehčují implementaci. Program využívá systém CMake[17] pro kompilaci. Pro uživatele systému Windows je možné zkompilovat kód i na zde a následně připravit instalační soubor, který má identické parametry s instalátorem na oficiálních stránkách Wiresharku. Takto zkompilovaný instalátor nabízí budoucím uživatelům implementovaných funkcí značný komfort. Nemusejí řešit jakékoliv závislosti (nástroj WinPcap je také součástí instalace) a ihned po úspěšné instalaci mohou začít využívat nově implementovanou funkcionalitu. Přiložený obrázek5.1 ilustruje přítomnost rozšíření v instalátoru, který právě úspěšně dokončil kopírování požadovaných souborů.



Obrázek 5.1: Ukázka instalačního dialogu v prostředí systému Windows

5.1 Registrace pluginu

Prvotní povinností pro disektor je registrovat své vstupní body do Wiresharku pro budoucí použití. Hlavní funkcí, která je v našem případě registrována, je heuristika pro rozpoznání protokolu CAT-TP. Alternativně je možné registrovat přímo disektor, který je pevně vázaný na protokol a port. Pokud je v komunikaci příslušná kombinace protokolu a portu detekována, je předána datová část paketu k analýze disektoru. Naopak změní-li se pouze charakteristický port a datová část nese tytéž validní informace, příslušný disektor není zavolán a data skončí buď neinterpretovaná nebo v horším případě interpretovaná nesprávným rozšířením s chybovým výstupem. Právě z těchto důvodů je v dokumentaci doporučeno tento způsob nepoužívat a upřednostnit registraci heuristické funkce.

Kromě zmíněné heuristické funkce je při registraci ještě získán ukazatel pro předání dat nesených námi analyzovaným protoklem dále. Více informací lze naleznout v následující sekci. Poslední důležitou součástní registrace je přidání záznamu do menu se statistikami, který povede na tabulku s daty generovanými naším disektorem.

5.2 Rozpoznání protokolu

Nyní se zaměříme na specifika spojená s implementací heuristické funkce. Ta má za úlohu v co nejkratším čase, ale s co největší jistotou určit, zda poskytnutá část paketu splňuje daná kritéria pro identifikaci protokolu. Inspiraci je vhodné hledat u již implementovaných heuristik z hojně používaných a odladěných protokolů.

Ve finální verzi je paket posuzován z hlediska tří kritérií. Nejprve je zkontrolována minimální velikost paketu, který ještě může být validním pro CAT-TP. V tomto konkrétním protokolu je vyžadována alespoň délka společné části hlavičky, která je rovna 18 bajtům. K získání potřebné hodnoty slouží funkce *tvb_captured_length*, která nám vrací délku reálně dostupných dat pro disekci. Následně, když víme, že data dodaná pro disekci mají délku rovnou nebo větší společné části hlavičky, můžeme zkusit sečíst udávanou délku hlavičky a případných dat. Takto získanou hodnotu následně provnáme s udávanou celkovou délkou zaznamenaných dat, která je dostupná díky funkci *tvb_reported_length*. Pokud se zmíněné dvě hodnoty rovnají, můžeme přikročit k poslední fázi kontroly, která spočívá ve validaci typu paketu. Pokud hodnota získaná z hlavičky patří do množiny povolených hodnot typů paketů, jsou tato data předána k disekci našemu rozšíření a funkce vrací boolean hosnotu *true* na znamení, že daný paket úspěšně validovala. Pokud některý ze zmíněných testů neskončí úspěchem, vrací heuristická funkce boolean hodnotu *false*, která je signálem pro vyzkoušení jiné heuristické funkce a příslušného disektoru.

Při pročítání existujících heuristických funkcí mě zarazilo použití odlišných funkcí pro získání délky dat pro analýzu a tak jsem se obrátil na dokumentaci. Zde jsem po krátkém hledání nalezl odpověď. Prvně zmíněná funkce *tvb_captured_length* vrací délku dat, která byla skutečně zaznamenaná a můžeme k nim přistupovat. Toto je hlavní kritérium pro první bod kontroly, protože pokud nemáme dostupná data, alespoň o délce společné části hlavičky, postrádá disekce smysl. Naopak funkce *tvb_reported_length* vrací délku odvozenou z nižších vrstev paketu a nemusí tak odpovídat reálně dostupným datům. Toto je požadované chování pro druhou část kontroly dostupný dat, protože nám pro potřeby disekce CAT-TP nevadí, že datová část paketu částečně nebo dokonce zcela chybí.

5.2.1 Inicializace

Před samotným procesem disekce je potřeba inicializovat struktury, které budeme v jejím průběhu používat. Tou nejpodstatnější částí je specifikace jednotlivých polí ve vrstvě protokolu, která obsahuje zejména tyto vlastnosti:

- Název pole - zobrazováno v GUI uživateli
- Zkratka názvu - klíčové slovo pro filtrování

- Datový typ
- Bitová maska
- Ukazatel na datovou strukturu s překladem hodnoty na řetězec znaků

Název pole je důležitý pro grafické rozhraní, neboť hodnota v něm uložená je v GUI zobrazena uživateli po otevření podrobností protokolu. Zkratka je zčásti předepsána konvencí a to výhradně jako *<název protokolu>.<zkratka názvu pole>*. Tato hodnota je významná pro funkci filtrování paketů (např. *cattp.seq == 2*). Datový typ udává jaké množství bajtů od počátku předaného ukazatele patří tomuto poli. Hned první pole protokolu CAT-TP je třeba deklarovat mírně odlišným způsobem než je běžné, protože hned první bajt hlavičky nese informace o dvou různých polích. Tento případ řeší specifikace masky, díky ktré je možné pro jedno pole získat jen určitý počet bitů a zbytek ignorovat. Poslední podstatná položka specifikace není nezbytnou součástí této striktury, ale výrazně zpřehledňuje poskytovaný výstup v GUI. Pokud dodáme pro získané datové hodnoty datovou strukturu s překladem, uživatel uvidí v GUI místo ní definovaný řetězec znaků a může tak získat lepší informační hodnotu. Této vlastnosti bylo využito pro interpretaci typu paketu.

Po inicializaci všech potřebných polí pro disekci přichází na řadu registrace funkcí, struktur a zařízení pro pokročilé vlastnosti našeho disektoru. Těm se ale budeme věnovat později v samostatných částech práce.

5.3 Disekce

Mnohokrát zmíněná disekce je stěžejní částí všech rozšíření nástroje Wireshark. Jedná se o proces procházení paketu po jednotlivých polích, jejich zobrazování v GUI, odvozování stavů relace, detekce chyb a obecně obohacování obsahu GUI o informace specifické pro daný protokol.

5.3.1 Průběh

Volání disektoru probíhá ve dvou vlnách. Nejprve je při načtení síťové komunikace zavolán pro každý její paket. Toto první zavolání lze rozeznat dvěma způsoby především parametr funkce disektoru *tree* je neinicializován a symbolizuje tak, že v GUI ještě nebyl vybrán konkrétní paket, pro který si žádá GUI podrobnosti. Druhým znakem je nastavení příznaku *visited* na boolean hodnotu *false*.

Při tomto prvním volání je nutné zaměřit se na sledování stavu relace a následné přidělování chybových příznaků pro pakety, které porušují pravidla dané specifikací. Pouze a jen při úvodním načítání dat jsou sekvenčně načteny všechny pakety zachycené komunikace, což je nezbytné pro úspěšnou detekci chyb a analýzu komunikace obecně.

Při druhém volání disektoru jsou již připravená data o stavu relace, počtu sledovaných chyb a k dalšímu volání funkcí pro jejich výpočet nedochází. Účelem druhého volání je tedy předat tyto informace přehlednou formou uživateli v podobě dostupných statistik pod příslušnou položkou v menu a obarvení chybných paketů. Dále je na rozdíl od prvního volání funkce již inicializovaná struktura *tree*, což nás vybízí k jejímu naplnění jednotlivými položkami protokolu.

5.3.2 Stav relace

Pro pokročilou detekci chyb bylo nezbytné implementovat detekci stavu relace. Ta je uchovávaná v datovém konteineru typu hash mapa, který je poskytován knihovnou GLib. Pozornost bylo třeba věnovat tvorbě klíče, který musí být unikátní pro každou relaci, ale zároveň identický pro oba směry komunikace. Finální řešení spočítá za pomoci nástrojů ze zmíněné knihovny hash z IP adresy, UDP portu a CAT-TP portu. Oba hashe jsou následně uloženy do klíče.

Hodnota uložená v konteineru obsahuje pro všechny relace následující položky, které jsou definovány pro klienta a server zvlášť:

- Hash identifikátor iniciátora spojení
- Aktuální sekvenční číslo
- Aktuální potvrzovací číslo
- Příznak určující, zda poslední odedslaný paket vyžaduje potvrzení protistranou

5.3.3 Chyby

Na základě uloženého stavu relace je možné detekovat pakety, které porušují pravidla stanovené technickou specifikací. Ve finální verzi rozšíření jsou detekovány následující chyby:

- Duplicitní pakety
- Paket nepříslušející aktuálním stavu relace
- Paket, který nepřísluší aktuálně otevřeným relacím
- Odeslání paketu nekorektním účastníkem komunikace
- Porušení pravidel pro stanovení sekvenčního čísla
- Nekorektní potvrzovací číslo

5.4 Obarvování paketů

Chyby je samozřejmě nezbytné nejen detekovat, ale i přehledně zobrazit uživateli. Wireshark k tomu to účelu nabízí nástroje ze sekce expert_info. Zde jsou definovány následující stupně závažnosti sdělení:

- Chat(šedá) - informace o běžné události
- Note(modrá) - upozornění (běžně se vyskytující chyba)
- Warn(žlutá) - varování (neobvyklá chyba)
- Error(červená) - závažná chyba

Společně se závažností paketu, která je spojena s obarvením paketu na danou barvu, je vyžadována i bližší specifikace nalezeného problému. Opět je na výběr z předdefinovaných možností a to konkrétně těchto:

- Checksum - neplatný kontrolní součet
- Sequence - chyba sekvenčního čísla nebo znovuzaslání paketu
- Response Code - neplatný kód odpovědi
- Undecoded - nelze úspěšně dokončit disekci
- Reassemble - neúspěch při pokusu o spojení segmentů
- Protocol - porušení specifikací protokolu
- Malformed - neplatný paket
- Debug - výstup pro ladění (nesmí se dostat do produkce)

Pro lepší představu je přiložen obrázek5.2, který ilustruje zobrazení paketu. U tohoto paketu byly detekovány dvě chyby a to neplatný kontrolní součet a také výskyt mimo otevřenou relaci.

5.5 Statistiky

Druhým způsobem, kterým můžeme uživateli poskytnout informace o detekovaných chybách je využít možností programu Wireshark pro práci se statistikami.

Prvním krokem, který je třeba provést je zavedení tzv. TAP interface. Díky této definici je možné volat externí funkci během disekce paketů a předat jí vývojářem definovanou strukturu s podrobnostmi o probíhajícím dekódování. V našem případě se jedná záznam specifikující chyby daného paketu.

```
⊞ Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
⊞ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
⊞ User Datagram Protocol, Src Port: 40000 (40000), Dst Port: 6000 (6000)
⊟ Card Application Toolkit Transport Protocol, Src Port: 0, Dst Port: 0, Seq: 0, Ack: 0
    0110 00.. = Packet Type: EACK (24)
    .... ..00 = Protocol Version: 0
    Header Length: 22
    Source Port: 0
    Destination Port: 0
    Data Length: 0
    Sequence Number: 0
    Acknowledgement Number: 0
    Window Size: 0
    Checksum: 0x0000 [incorrect]
  ⊞ EACK Sequence Numbers
  ⊞ [Expert Info (Warn/Protocol): Packet without session]
  ⊞ [Expert Info (Note/Checksum): Incorrect checksum]
```

Obrázek 5.2: Ukázka detekce chyb

CAT-TP with filter:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
⊟ Detected Errors	16				0,0078	100%	0,0900	0,000
Incorrect checksum	11				0,0054	68,75%	0,0700	0,000
Invalid ACK number	3				0,0015	18,75%	0,0200	0,000
Invalid SEQ number	1				0,0005	6,25%	0,0100	2,050
Duplicate packet	1				0,0005	6,25%	0,0100	2,028
Total Packets	11				0,0054	100%	0,0700	0,000

Copy Save As Close

Obrázek 5.3: Ukázka tabulky se statistikami detekce chyb

V rámci funkce, která obsluhuje námi zavedený TAP interface, probíhá inkrementace statistických čítačů. Na rozdíl od barvení paketů není potřeba deklarovat nic jiného než název čítače pro inkrementaci a stromovou strukturu zobrazení v tabulce.

Tabulka5.3 shrnující statistiky protokolu CAT-TP byla rozdělena do dvou částí. V první můžeme vidět souhrnný počet nalezených chyb, který je možno rozbalit a podívat na toto číslo detailněji. Po rozbalení se objeví statistiky pro jednotlivé detekované chyby. Pod touto položkou se nachází celkový počet analyzovaných CAT-TP paketů.

5.6 Poznámky k implementaci

Vzhledem k tomu, že implementace pokročilých funkcí disektoru zvýšily počet řádek kódu k jednou tisíci, měl jsem v úmyslu rozdělit zdrojový soubor

na několik samostatných částí. To by vedlo ke zvýšení přehlednosti a snadnější orientaci při potřebě zásahu do kódu. Zjistil jsem však, že standardem je pouze rozdělení na hlavičku a soubor s implementací disektoru. Ani velmi rozsáhlé rozšíření pro protokol TCP nevybočuje z této zvyklosti a soubor s implementací má v současnosti téměř šest tisíc řádků. Nezbylo mi tedy než respektovat daná pravidla a nerozdělovat implementaci a ponechat rozdělení na jeden hlavičkový a jeden zdrojový soubor. Pokud se bude funkcionalita protokolů v současných souborech nadále rozšiřovat, přeroste jednoho dne velikost zdrojových souborů únosnou míru a doporučoval bych proto zvážit změnu zvyklostí.

V průběhu implementace jsem nenarazil na požadavek, který nemohl být splněn z důvodu chybějící podpory v platformě pro vývoj nástroje Wireshark. Ale musím zmínit, že cestu za plně fungujícím rozšířením mi komplikovala zastaralá a nekompletní dokumentace, která je vhodná pro první kroky v programování v tomto specifickém prostředí, ale nedostačuje pro implementaci pokročilejších funkcí. Protože byly stanoveny požadavky na implementaci nadstandardních rysů pluginu, bylo nutné procházet zdrojový kód pro nalezení komentářů autorů a specifikace argumentů potřebných funkcí. To mělo zásadní vliv na rychlost vývoje a rozhodně by se projekt Wireshark měl do budoucna zabývat nápravou této situace.

Kapitola 6

Testovací prostředí

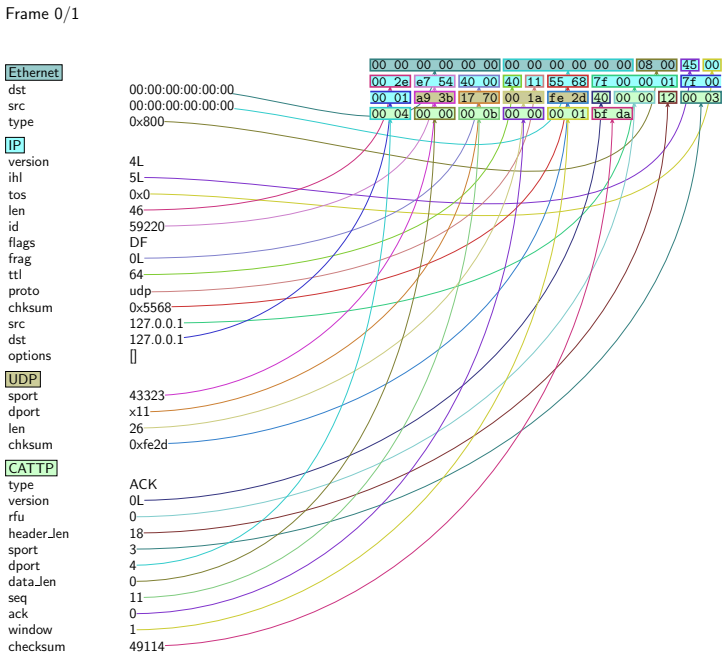
Nejlepším způsobem otestování pro vytvořené rozšíření síťového analyzátoru by bylo nasazení do reálného provozu a verifikovat na něm funkčnost dodaného řešení. Bohužel tato varianta není z technických a právních důvodů možná. Protokol je používán pro předávání citlivých informací uživatele mobilního telefonu, což znamená, že umožnit přístup k těmto informacím třetí osobě je nepřípustné. Na druhou stranu je potřeba ověřit chování disektoru, alespoň v simulovaném prostředí, které se bude z pohledu síťové analýzy blížit tomu reálnému. Implementace simulátoru znamená ještě jeden přínos. Analyzátor nahlíží na síťový provoz z jiného pohledu než komunikující strany. Implementace klienta a serveru tak nabízí nahlédnutí na vnitřní mechanismy CAT-TP protokolu z jiného úhlu a prohloubit tak doposud získané znalosti tohoto protokolu v oblasti konstrukce jednotlivých paketů, přechodů mezi stavy komunikace a reakcí na jednotlivé události.

6.1 Volba platformy

Vzhledem k požadavku multiplatformnosti analyzátoru, bylo nutné zvolit adekvátní platformu i pro simulátor síťového provozu. Díky cvičením předmětu MI-SIB[18] byl výběr poměrně rychlý. V rámci tohoto předmětu byl využíván nástroj Scapy[19], který slouží k vytváření, zachytávání a manipulaci paketů. Jedná se o knihovnu programovacího jazyka Python, který umožňuje jednoduchou přenositelnost na požadované platformy. Licence GPLv2[20] je také vyhovující.

6.1.1 Scapy

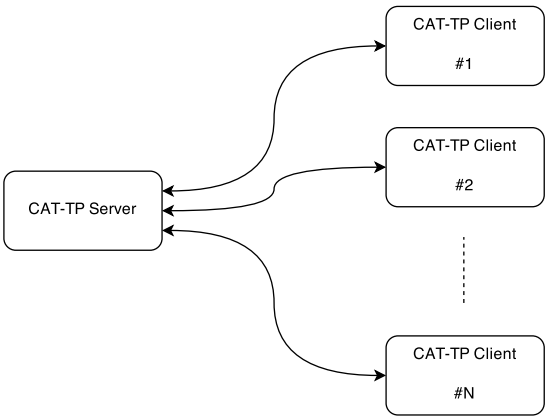
Specializovaných nástrojů pro tvorbu, manipulaci a výměnu paketů je celá řada. Nástroj Scapy je ale jedninečný svou univerzálností. Z příkazové řádky umožňuje provádět následující operace:



Obrázek 6.1: Grafická reprezentace paketu z nástroje Scapy

- Definice vlastního protokolu
- Využití definic pro tvorbu a analýzu paketů
- Libovolné zapouzdřování protokolů
- Generování grafického výstupu6.1 dekódovaného paketu

Pro potřeby této práce bylo využito jen zlomku jeho schopností, ale i na potřebné malé podmnožině bylo více než patrné dodržování konceptu jednoduchosti a také přehlednosti. Jednoduchost můžeme demonstrovat na příkladu,



Obrázek 6.2: Diagram architektury

jak složité je získat základní podporu pro disekci nového protokolu. V předchozích kapitolách byl popsán způsob, jakým je možné dosáhnout tohoto cíle v prostředí programu Wireshark. Můžeme tvrdit, že se v tomto případě nejednalo o triviální množství práce. Naopak definice nového protokolu ve Scapy byla řádově jednodušší. Specifikace se v zásadě skládá jen z definice datového typu a názvu pole. Pro větší uživatelský komfort byla, stejně jako u disektoru ve Wiresharku, doplněna mapa pro překlad typu paketu na řetězec znaků.

Jediná výtka se týká možností odesílání a přijímání paketů. Tuto funkcionalitu Scapy nabízí, ale bohužel neposkytuje dostatečné možnosti konfigurace. Toto zjištění v důsledku znamenalo omezit použití tohoto nástroje pouze na disekci a paketovou manipulaci, kterou zvládá výborně. Pro funkce odesílání a přijímání paketů bylo použito standardních modulů jazyka Python a to na straně klienta i serveru.

Pro automatické dekódování paketů po jednotlivých vrstvách používá Scapy pevně definované charakteristické porty pro každý protokol. V porovnání s heuristickou funkcí v programu Wireshark se jedná o kompromisní řešení. Lze jej ale s přihlédnutím ke zmíněnému přístupu zjednodušování pochopit.

6.2 Architektura

Přiložený diagram6.2 znázorňuje architekturu implementovaného simulátoru. Ten se skládá z jednoho serveru a jednoho či více klientů. Server podporuje vícevláknové zpracování. Příchod více požadavků od klientů najednou tak nepředstavuje pro jejich obsloužení významnější zpoždění. Klientská část je tvořena jedním vláknem, které je spuštěno pro každou sekci klienta v konfiguraci (viz. manuál v příloze).

6.3 Implementace

Zdrojový kód je rozdělen na tři části. Samozřejmě s dělí na server a klienta. Třetí soubor obsahuje sdílené definice protokolu pro nástroj Scapy a také pomocné funkce pro validaci uživatelsky definované konfigurace.

Vzhledem k tomu, že CAT-TP je transportním protokolem, simulátor je založen na následujícím scénáři. Klient nejprve iniciuje připojení k serveru. po úspěšném navázání spojení začíná server přenášet předem definovaný soubor klientovi. Po dokončení přenosu je spojení ukončeno.

6.3.1 Konfigurace

Kvůli množství požadovaných parametrů pro spouštění serveru i klienta bylo před konfigurací pomocí argumentů upřednostněno řešení využívající externího zdroje dat. Spolehlivým formátem pro tyto účely je XML. Soubor může být díky této volbě sdílený (obsahuje konfiguraci pro server i klienta) nebo můžeme parametry definovat zvlášť.

6.3.2 Server

Již bylo zmíněno, že server je vícevláknový. Standardní UDP server, který je dostupný v Pythonu je však jen jednovláknový. Cíleného stavu bylo dosaženo použitím tzv. mixinu. Jedná se o obecný princip, který v objektově orientovaných jazycích umožňuje vytvořit třídu, která sdružuje metody a následně i funkcionalitu z několika předků. V tomto konkrétním případě je sdružen do jedné třídy standardní UDP server a speciální vícevláknový mixin. Výsledný UDP server má identické parametry s normálním a nevyžaduje jakoukoliv dodatečnou konfiguraci.

Vícevláknová implementace serveru na jedné straně přináší užitek z vyššího výkonu a připravenosti na příjem mnoha požadavků v krátkém časovém úseku. Současně ale přináší povinnost zabývat se přístupem do sdílené paměti, je-li přítomna. V našem případě je nezbytné udržovat sdílený seznam otevřených relací, aby bylo možné navázat na poslední odeslaný paket dle specifikací protokolu a odeslat adekvátní datový segment. Začal jsem se tedy zabývat, jak sdílenou paměť zabezpečit. Protože jsem na naší fakultě prošel kurzem vícevláknového programování, nejprve jsem pomyslel na zámek typu mutex. Po prostudování dokumentace jsem však zjistil, že toto řešení nebude potřebné. Dle oficiální dokumentace[1] jsou všechny základní datové typy v Pythonu bezpečné pro sdílené použití více vlákny.

Seznam otevřených relací je implementován pomocí datového typu slovník. Tento kontejner se skládá z klíče a hodnoty. Je optimalizován pro čtení. Pomocí klíče lze získat příslušnou hodnotu, ale ne naopak. Tyto vlastnosti jsou slučitelné se zamýšleným použitím. Je ale nutné zabývat se tvorbou vhodné

[1] `https://docs.python.org/2/glossary.html#term-global-interpreter-lock`

hodnoty klíče. Ta musí být jednoznačným identifikátorem protistrany a unikátní pro každou otevřenou relaci. Její výpočet musí být velmi rychlý, aby nezpomaloval vícevláknové odbavování požadavků klientů. Tyto vlastnosti jsou typické pro hashovací funkce a právě jedna z nich se při tvorbě klíče používá. Nejprve je zkonstruován řetězec znaků obsahující zdrojovou IP adresu, UDP port a CAT-TP port. Ten je následně zaslán, jako parametr hashovací funkci MD5[21].

6.3.3 Simulování chyb

Pro věrnější simulaci síťového provozu byla do simulátoru implementována možnost ignorovat přijatý paket. Tato situace simuluje chybu při přenosu paketu a jeho ztracení během cesty sítí. Na straně klienta i serveru byly implementovány mechanismy protokolu, které mají za úkol se s tímto problémem vyrovnat a vrátit spojení do stabilního stavu.

Četnost výskytu těchto simulovaných chyb je řízena dvěma hodnotami v konfiguraci. Jednou u nich je minimální rozestup mezi chybami. Tato hodnota umožňuje dát protokolu příležitost pro překonání nestandardního stavu po ztrátě paketu a návratu do normálu. Pokud je hodnota nastavena na 0, není možné vyloučit úplné zastavení komunikace a zrušení spojení s příslušným chybovým kódem.

Simulování chyb je také závislé na druhé hodnotě, kterou je parametr exponenciálního rozdělení lambda, jehož základní charakteristiky budou shrnuty v následující části práce.

6.3.4 Exponenciální rozdělení

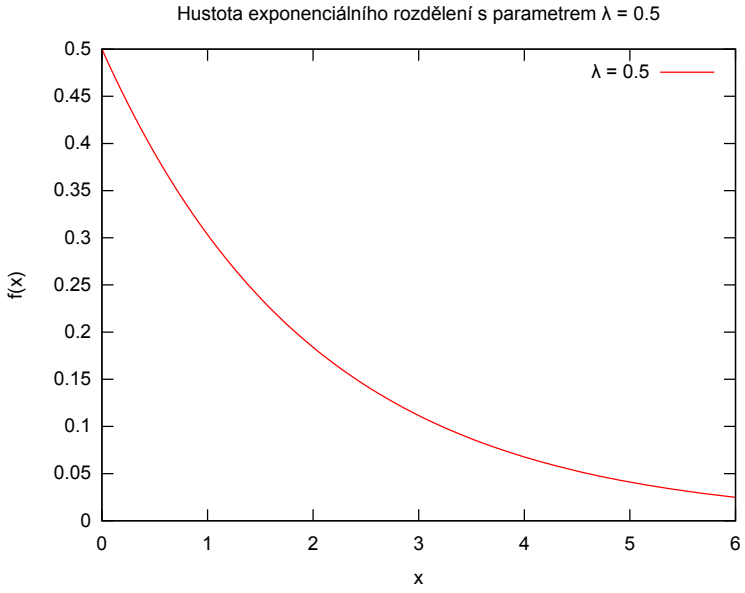
Toto rozdělení spojité náhodné veličiny charakterizuje délku intervalu mezi dvěma poissonovskými jevy. Náhodná veličina X má exponenciální rozdělení (s parametrem $\lambda > 0$), pokud platí následující vztah6.1.

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{pokud } x \geq 0 \\ 0, & \text{pokud } x < 0 \end{cases} \tag{6.1}$$

Parametr λ udává intenzitu výskytu daného jevu. Můžeme se setkat i s parametrem uvedeným ve formě střední hodnoty, která je definována následovně6.2.

$$EX = \frac{1}{\lambda} \tag{6.2}$$

Toto rozdělení bývá často vysvětlováno na příkladu příchodu zákazníků do obchodu. Tento jev naplňuje svými parametry kritéria poissonovského rozdělení a můžeme tedy použít exponenciální rozdělení k modelování intervalů mezi příchody jednotlivých zákazníků. Budiž příkladem, že pozorováním bylo



Obrázek 6.3: Graf hustoty exponenciálního rozdělení

zjištěno, že za danou časovou jednotku vejdou do obchodu průměrně dva zákazníci. Díky vztahu pro výpočet střední hodnoty můžeme z tohoto údaje odvodit6.3 parametr λ.

$$\lambda = \frac{1}{EX} = \frac{1}{2} = 0.5 \tag{6.3}$$

Na zmíněný příklad jsem si vzpomenul při řešení otázky, jakým způsobem generovat náhodné chyby na lince. Místo analogie se zákazníkem jsem si představil chyby při komunikaci a zvolil právě toto rozdělení pro generování rozestupů mezi simulacemi chyb. Pro lepší ilustraci vlastností zvoleného rozdělení pro standardně nastavený parametr $\lambda = 0.5$ přikládám graf hustoty6.1.

6.4 Simulace

Před začátkem simulace je vhodné připravit si dvě okna terminálu. Jedno pro server a druhé pro klienta. Pokud budeme chtít komunikaci v reálném čase i sledovat, je také nutné otevřít program Wireshark a zapnout naslouchání na příslušné síťovém zařízení. Je-li naším záměrem jen zaznamenat připravovanou simulaci pro pozdější analýzu rozšířením, nemusíme nutně použít program Wireshark, ale i libovolný jiný nástroj umožňující záznam síťové komunikace.

6.4.1 Scénáře testování

Při testování rozšíření jsem prošel několika scénáři testování. Na počátku vývoje jsem volil nejjednodušší konfiguraci jednoho klienta a jednoho serveru. Velikost souboru, který měl být pomocí protokolu CAT-TP přenesen byla rovněž minimální, tak aby se vešla do jediného nesegmentovaného paketu. Na této konfiguraci byly odstraněny některé chyby, které bylo při statické analýze kódu obtížné najít.

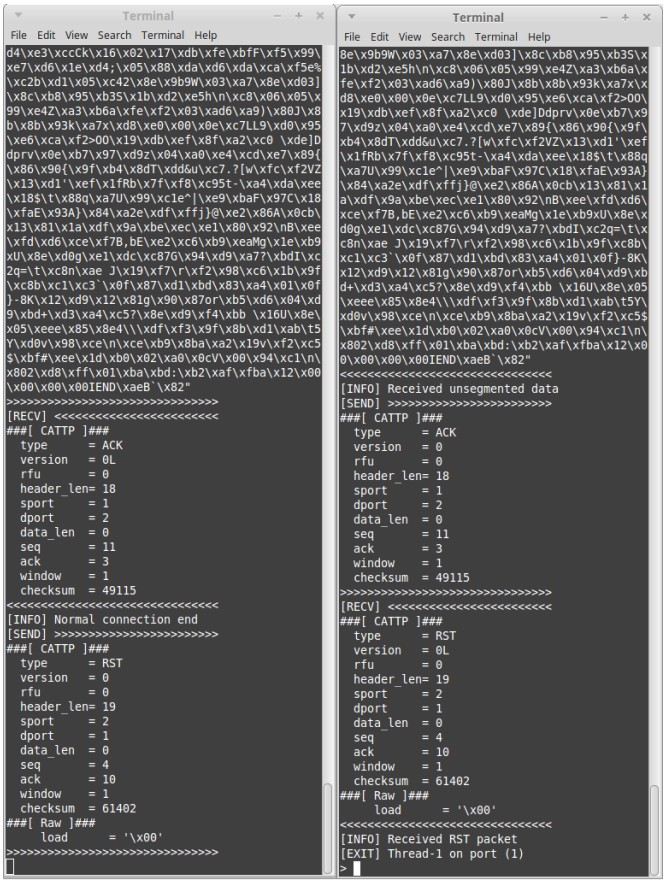
Od tohoto základního scénáře jsem se posunul dále. Prvním parametrem, který jsem začal měnit byla velikost zasílaného souboru. Změna tohoto parametru simulátoru zapojila další část implementace, která se stará o rozdělení souboru do segmentů na straně serveru a naopak slučování segmentů pro rekonstrukci původních dat na straně klienta.

V poslední fázi testování jsem zvyšoval počet klientů, kteří se v jeden okamžik snaží komunikovat se serverem. Tento krok na rozdíl od předchozích neodhalil jedinou chybu. Tuto skutečnost je možné přikládat na straně serveru mechanismu mixinu, který potvrzuje, že cesta od fungujícího jednovláknového serveru k vícevláknovému je velmi přímočará. Na straně klienta jsou sice jednotlivá vlákna spouštěna jednoho společného předka, ale ihned po vytvoření vlákna se stávají samostatnými a výskyt problému byl tedy velmi nepravděpodobný.

6.4.2 Ukázka výstupů

První přiložený obrázek6.4 v této sekci nabízí pohled na výstup skriptů klienta a serveru v průběhu simulace. V levém okně terminálu běží server a v pravém okně právě ukončil jeden z klientů komunikační relaci. Pakety proudící oběma směry jsou díky nástroji Scapy zobrazeny v čitelné podobě. Průvodním jevem vícevláknové simulace bylo mísení výstupů z jednotlivých vláken. Tuto nepříjemnost jsem řešil definicí nové funkce, která vytiskne řetězec ze svého argumentu a následně zavolá *sys.stdout.flush()*. Tato funkce přinutí ihned vytisknout řetězec do terminálu a minimalizuje tak jev mísení výstupů. Z obrázku je patrné, že toto řešení problém napravilo a výstup je tak dobře čitelný.

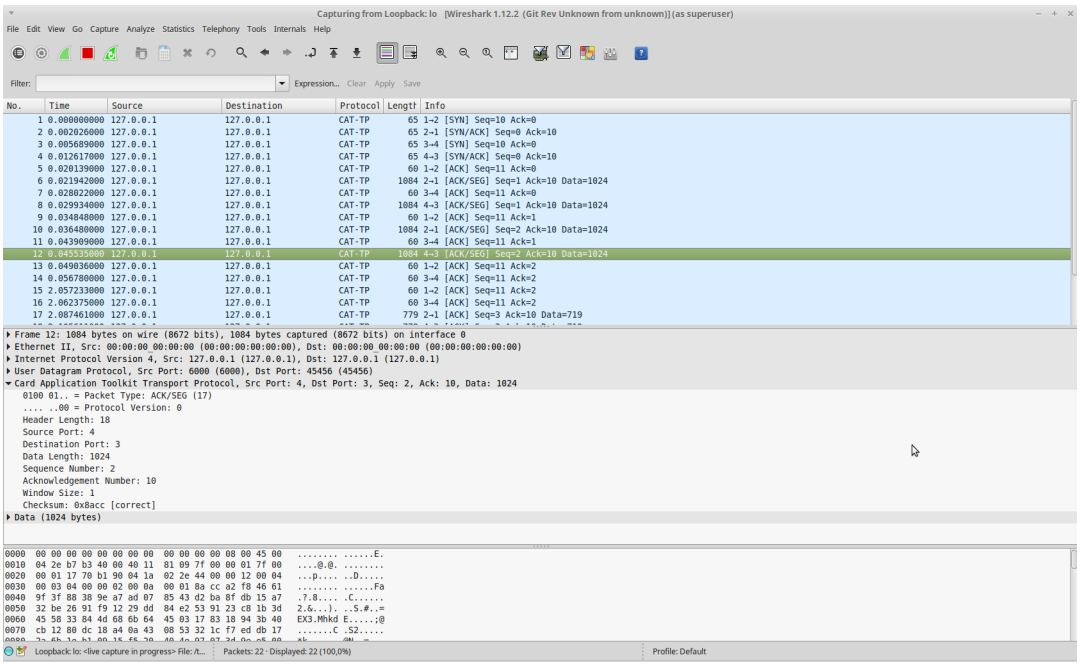
Druhý obrázek6.5 je pohledem na program Wireshark, který právě zaznamenává komunikaci serveru a dvou klientů. Jak se dá očekávat pro možnost záznamu ze síťového zařízení je nutné tento program spustit s právy administrátora(root), což je viditelné v názvu otevřeného okna analyzátoru. Mezi pakety s pořadovým číslem 14 a 15 si můžeme všimnout prodlevy dvou vteřin. Tento jev je způsobený simulovanou chybou na lince. V jejím důsledku nebyl odeslán jeden paket a spojení tak na jedné komunikující straně timeoutovalo. Časovač byl dle výpisu nastaven na ony dvě vteřiny. Následně byl proveden pokus o zotavení z tohoto chybového stavu opětovným odesláním ztraceného paketu. Na tento pokus reagovala protistrana pozitivně a spojení se opět vrátilo do normálního stavu s tím, že přenos dat mohl pokračovat.



Obrázek 6.4: Výstup simulátoru v oknech terminálu

6.4.3 Zhodnocení testování

Testování rozšíření analyzátoru splnilo svůj účel a pomohlo odhalit několik chyb v algoritmu simulátoru. Změny nastaly i ve formátu výpisů, které vedly ke zpřehlednění s snadnějšímu ověření správnosti fungování. Pozorování během disekce paketů protokolu CAT-TP v reálném čase neodhalilo výkonnostní nedostatky řešení. Pakety v okně programu přibývaly dle očekávání. Protože není v současné době volně dostupný vzorek komunikace s protokolem CAT-TP, poslouží implementovaný simulátor i jako zdroj pro vzorky komunikace do přílohy této práce. Tyto přiložené ukázky mohou sloužit jednak k ověření funkčnosti po instalaci rozšíření uživatelem, ale budou také volně k dispozici všem, kteří budou tento protokol analyzovat v budoucnu.



Obrázek 6.5: Naslouchání programu Wireshark při probíhající simulaci

Závěr

Vypracování této práce proběhlo v následujících fázích dle schváleného zadání. Nejprve proběhlo seznámení s technickou specifikací, která poskytuje informace o struktuře, mechanismech a dalších aspektech protokolu CAT-TP. Následně proběhl výběr síťového analyzátoru, který nejlépe odpovídá zadaným požadavkům(viz. zadání práce). Na tuto fázi bylo navázáno analýzou vnitřní struktury vybraného analyzátoru s přihlédnutím k budoucí implementaci vedoucí k jeho rozšíření. Praktická část práce těžila ze získaných znalostí a jejím výstupem je v první řadě funkční rozšíření pro analýzu protokolu CAT-TP. Funkčnost byla ověřena jeho použitím při zachytávání komunikace implementovaného simulátoru. Všechny zmíněné kroky byly úspěšně dokončeny a naplňují tak stanovené cíle této práce.

Hlavním kritériem pro ověření implementace rozšíření síťového analyzátoru by logicky měla být úspěšná validace v reálném provozu. Protože je však v tomto případě technicky a právně nemožné provést takové testování, bylo nutné spokojit se s ověřením řešení na simulované komunikaci. Tato aktivita se skládala ze statické analýzy zaznamenané komunikace protokolu CAT-TP, ale následně také z analyzování živého provozu z nově implementovaného simulátoru. V konečně fázi mi pomohlo toto testování odhalit chybu v implementaci výpočtu kontrolního součtu, která by jinak byla velmi těžko odhalitelná. Díky procesu testování tedy do výsledného řešení nepronikla chyba, která sice nebyla fatálního charakteru, ale způsobovala zobrazení nekorektní hodnoty.

Implementovaný analyzátor protokolu, který je hlavním výstupem této práce, nyní nalezne využití na pracovních stanicích vývojářů a analytiků, kteří se denně setkávají s tímto protokolem a poskytnutý nástroj pro ně bude prostředkem pro zefektivnění práce. Druhý výstup, který je možné ihned použít jsou implementované definice a struktury tohoto protokolu pro nástroj Scapy. Následně je možné díky Scapy jednoduše vytvářet syntetické pakety, které by jinak bylo nutné obtížně konstruovat. Stejně tak je možné použít jej v omezené míře pro základní disekci paketů CAT-TP protokolu.

Možná budoucí práce, která by navazovala na tuto, by spočívala v rozší-

ření detekovaných chyb. Současná implementace detekuje převážnou většinu možných problémů a porušení pravidel komunikace definovaných technickou specifikací. Na druhou stranu ale poskytuje prostor pro další vývoj. Dále je možné usilovat o zařazení do standardního distribučního kanálu programu Wireshark. To by znamenalo zařazení podpory protokolu CAT-TP do všech budoucích verzí a nebylo by tak nutné distribuovat a instalovat rozšíření nad rámec základní instalace.

Díky této práci jsem získal nové znalosti a zkušenosti z této specifické oblasti programování. Dále jsem měl možnost uplatnit teoretické znalosti, které jsem získal v magisterské etapě vzdělávání. Bylo velmi příjemné zjistit, že na otázky, které se v průběhu implementování objevovaly, jsem častokrát našel odpověď díky znalostem získaným během studia. Zejména bych zmínil problematiku simulování chyb na lince a volby správných parametrů exponenciálního rozdělení. Přestože byla implementace ze zmíněných důvodů občas těžkopádná, po celou dobu zpracovávání mě nepřestalo zadané téma bavit a hnát dopředu za co nejlepším výsledkem.

Literatura

[1] European Telecommunications Standards Institute: *Smart Cards, Transport protocol for CAT applications, Stage 2, Release 6*. 2009, [ETSI TS 102 127 R6].

[2] Wireshark · Documentation. Dostupné z: `https://www.wireshark.org/docs/`

[3] SINTEF: *Big Data, for better or worse: 90% of world's data generated over last two years. [online]*. [ScienceDaily, 22 May 2013]. Dostupné z: `www.sciencedaily.com/releases/2013/05/130522085217.htm`

[4] International Telecommunication Union: *Reference Model of Open Systems Interconnection for CCITT Applications*. 1988.

[5] Internet Engineering Task Force: *RFC 1122 Requirements for Internet Hosts - Communication Layers*. October 1989.

[6] Gemalto, s.r.o.: *Understanding BIP - course presentation*. [Gemalto internal].

[7] European Telecommunications Standards Institute: *Smart Cards, Card Application Toolkit (CAT), Release 12*. 2014, [ETSI TS 102 223].

[8] Mayes, K.; Markantonakis, K.: *Smart Cards, Tokens, Security and Applications*. První vydání, ISBN 0387721975, 9780387721972.

[9] Internet Engineering Task Force: *RFC 791, Internet Protocol - DARPA Inernet Programm, Protocol Specification*. September 1981.

[10] Internet Engineering Task Force: *RFC 768, User Datagram Protocol, Protocol Specification*. August 1980.

[11] Internet Engineering Task Force: *RFC 793, Transmission Control Protocol*. September 1981.

[12] Kismet - network detector, sniffer, and intrusion detection system. Dostupné z: https://www.kismetwireless.net/

[13] Microsoft Network Monitor. Dostupné z: https://support.microsoft.com/cs-cz/kb/933741/en-us

[14] ngrep - network grep. Dostupné z: http://ngrep.sourceforge.net/

[15] tcpdump/libpcap public repository. Dostupné z: http://www.tcpdump.org/

[16] GLib. Dostupné z: https://developer.gnome.org/glib/

[17] CMake. Dostupné z: http://www.cmake.org/

[18] Blažek, R.: Síťová bezpečnost - magisterský předmět. Dostupné z: https://edux.fit.cvut.cz/courses/MI-SIB/start

[19] Biondi, P.: Scapy: interactive packet manipulation tool. Dostupné z: http://www.secdev.org/projects/scapy

[20] GNU General Public License v2. Dostupné z: https://www.gnu.org/licenses/gpl-2.0.html

[21] Rivest, R. L.: *The MD5 Message-Digest Algorithm*. April 1992.

Příloha **A**

Seznam použitých zkratek

3G Third Generation (mobile communication system)

API Application Programming Interface

BIP Bearer Independent Protocol

CAT-TP Card Application Toolkit Transport Protocol

EPAN Ethereal Packet ANalyzer

ETSI European Telecommunications Standards Institute

GPL General public license

GPRS General packet radio service

GUI Graphical user interface

IP Internet protocol

ISO International organization for standardization

OSI Open systems interconnection

OTA Over the air

PDU Protocol data unit

RFC Request for comments

SDU Service data unit

SIM Subscriber identity module

SMS Short message service

TAP Network tap

TCP Transmission control protocol

TS Technical specification

UDP User datagram protocol

XML Extensible markup language

Příloha B

Obsah přiloženého CD

- `analyzer` ..... adresář s implementací analyzátoru
  - `src` ..... adresář se zdrojovými kódem
  - `unix_lib` ..... adresář s kinhovnou pro Unix
  - `win_installer` ..... adresář s instalátorem pro Windows
  - `win_ilb` ..... adresář s kinhovnou pro Windows
  - `Installation_Guide.pdf` ..... instalační příručka
  - `User_Manual.pdf` ..... uživatelský manuál
- `simulator` ..... adresář s implementací simulátoru
  - `src` ..... adresář s implementací simulátoru
  - `Simulator_Manual.pdf` ..... intalační a uživateský manuál
- `text` ..... text práce
  - `DP_Pesek_Zdenek_2015.pdf` ..... text práce ve formátu PDF
- `traffic_samples` ..... adresář se vzorky komunikace
  - `simulation_with_errors.pcap` ..... komunikace s chybami
  - `simulation_with_retries.pcap` ..... komunikace ze simulátoru
  - `single_eack_packet.pcap` ..... jeden paket typu EACK

PŘÍLOHA C

CAT-TP Wireshark Plugin - Installation Guide

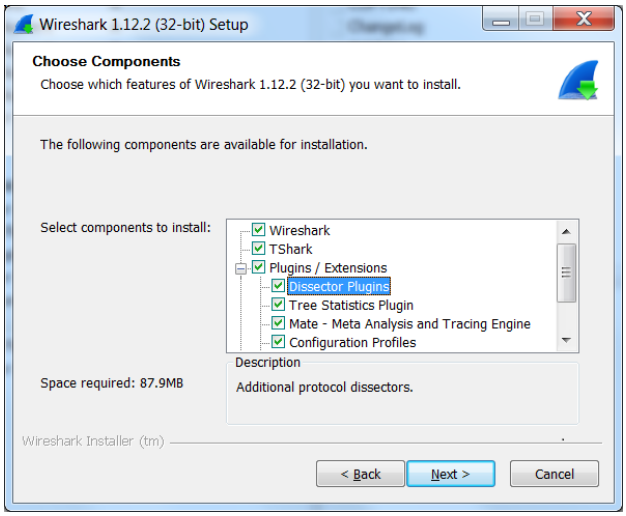
1) Requirements

System requirements of this plugin are identical with the Wireshark requirements specified on the official website.

- Common (Linux and Windows)
  - CPU: 32-bit x86 or 64-bit AMD64/x86-64 processor
  - RAM: at least 200 MB
  - Disk space: at least 75 MB
  - Screen: at least 1024×768 resolution with at least 16 bit color
- Microsoft Windows
  - Versions: Win8, Win7, Vista
  - Server Versions: Server 2012, Server 2008 R2, Server 2008, Server 2003
- Linux
  - Wireshark is supported by majority of the currently available distributions

2) Installation on Microsoft Windows

The simplest way to fully working Wireshark dissection of the CAT-TP traffic is to use the prepared installation package *Wireshark-win32-1.12.2.exe*. This executable file was compiled with the new dissector source files, so the plugin is installed together with the main program and WinPcap. There is no need to do any extra steps. You can start with dissecting the files or live traffic out of the box. Keep in mind, that updating the program can break the plugin compatibility.



NOTE: Make sure, that the option *Plugins/Extensions - Dissector Plugins* is checked on the *Choose Components* dialog during the installation process.

There is also a second option. If you have the correct Wireshark version (1.12.2 32bit) already installed on your computer, there is a *cattp.dll* library in the resources. To enable the CAT-TP dissection copy this library to the plugins directory inside the Wireshark installation folder. Typical location for Win7 64bit OS is following:

C:\Program Files (x86)\Wireshark\plugins

3) Installation on Linux

Dependencies, which are required to be installed, are distribution specific and therefore it is not possible to present a universal guide for installation. In my case (Linux Mint 17.1 Rebecca), I had to install the following packages to successfully compile and install Wireshark.

autoconf bison flex libtool libgtk2.0-dev libpcap-dev libc-ares-dev libsmi2-dev libgnutls-dev libgcrypt11-dev libkrb5-dev libcap2-bin libgeoip-dev libortp-dev libportaudio-dev

After these prerequisites are successfully installed, go to the root directory of the unpacked Wireshark sources. Now, we can execute the traditional set of commands, which will configure the build environment, execute the build and install the compiled binaries with corresponding libraries.

```
./autogen.sh
./configure
make
sudo make install
```

The user can now run the program by executing the *wireshark* command inside the terminal.

Alternatively the user can place a precompiled library *cattp.so* to the plugins folder of the local Wireshark installation, if the environment and Wireshark version is compatible with Linux Mint 17.1 Rebecca and Wireshark 1.12.2. Typical *plugins* directory location is following:

/etc/wireshark/plugins

4) Links

https://www.wireshark.org/docs/wsug_html_chunked/ChIntroPlatforms.html

http://binarynature.blogspot.cz/2011/05/compile-and-install-wireshark-on-fedora.html

Příloha D

CAT-TP Wireshark Plugin - User Manual

Assuming you have successfully installed the Wireshark tool with the supplied CAT-TP plugin, we can now have a look at the features this plugin provides.

1) Summary window

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	CAT-TP	65	1→2 [SYN] Seq=10 Ack=0
2	0.005865	127.0.0.1	127.0.0.1	CAT-TP	65	2→1 [SYN/ACK] Seq=0 Ack=10
3	0.016835	127.0.0.1	127.0.0.1	CAT-TP	65	3→4 [SYN] Seq=10 Ack=0
4	0.024272	127.0.0.1	127.0.0.1	CAT-TP	65	4→3 [SYN/ACK] Seq=0 Ack=10
5	0.030848	127.0.0.1	127.0.0.1	CAT-TP	60	1→2 [ACK] Seq=11 Ack=0
6	0.033058	127.0.0.1	127.0.0.1	CAT-TP	1084	2→1 [ACK/SEG] Seq=1 Ack=10 Data=1024
7	0.036142	127.0.0.1	127.0.0.1	CAT-TP	60	3→4 [ACK] Seq=11 Ack=0
8	0.038952	127.0.0.1	127.0.0.1	CAT-TP	1084	4→3 [ACK/SEG] Seq=1 Ack=10 Data=1024
9	0.043421	127.0.0.1	127.0.0.1	CAT-TP	60	1→2 [ACK] Seq=11 Ack=1
10	0.046741	127.0.0.1	127.0.0.1	CAT-TP	1084	2→1 [ACK/SEG] Seq=2 Ack=10 Data=1024
11	0.052297	127.0.0.1	127.0.0.1	CAT-TP	60	3→4 [ACK] Seq=11 Ack=1
12	0.058681	127.0.0.1	127.0.0.1	CAT-TP	1084	4→3 [ACK/SEG] Seq=2 Ack=10 Data=1024
13	0.062934	127.0.0.1	127.0.0.1	CAT-TP	60	1→2 [ACK] Seq=11 Ack=2
14	0.064869	127.0.0.1	127.0.0.1	CAT-TP	60	3→4 [ACK] Seq=11 Ack=2
15	2.068419	127.0.0.1	127.0.0.1	CAT-TP	60	1→2 [ACK] Seq=11 Ack=2
16	2.075324	127.0.0.1	127.0.0.1	CAT-TP	60	3→4 [ACK] Seq=11 Ack=2
17	2.080337	127.0.0.1	127.0.0.1	CAT-TP	779	2→1 [ACK] Seq=3 Ack=10 Data=719
18	2.093993	127.0.0.1	127.0.0.1	CAT-TP	60	1→2 [ACK] Seq=11 Ack=3
19	2.107823	127.0.0.1	127.0.0.1	CAT-TP	779	4→3 [ACK] Seq=3 Ack=10 Data=719

After opening a CAT-TP traffic sample, the main UI component will be displayed. The summary window should provide a basic information about the captured packet. If the packet was recognized to be a CAT-TP, the protocol column is changed accordingly. In the Info column there is a information about a source and destination port of the CAT-TP protocol, packet type, sequence number and acknowledgement number. Also, if the packet has a data payload, length of this data segment is displayed there.

2) Packet detail

```
⊞ Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
⊞ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
⊞ User Datagram Protocol, Src Port: 40000 (40000), Dst Port: 6000 (6000)
⊟ Card Application Toolkit Transport Protocol, Src Port: 0, Dst Port: 0, Seq: 0, Ack: 0
    0110 00.. = Packet Type: EACK (24)
    .... ..00 = Protocol Version: 0
    Header Length: 22
    Source Port: 0
    Destination Port: 0
    Data Length: 0
    Sequence Number: 0
    Acknowledgement Number: 0
    Window Size: 0
    Checksum: 0x0000 [incorrect]
  ⊟ EACK Sequence Numbers
      EACK Sequence Number: 1
      EACK Sequence Number: 2
  ⊞ [Expert Info (Warn/Protocol): Packet without session]
  ⊞ [Expert Info (Note/Checksum): Incorrect checksum]
```

After selecting a single packet in the summary window, user has a chance to inspect this packet in detail. All parts of the header (both common and variable part) are dissected and displayed in a separate rows. There is one exception in case we are viewing a EACK packet.

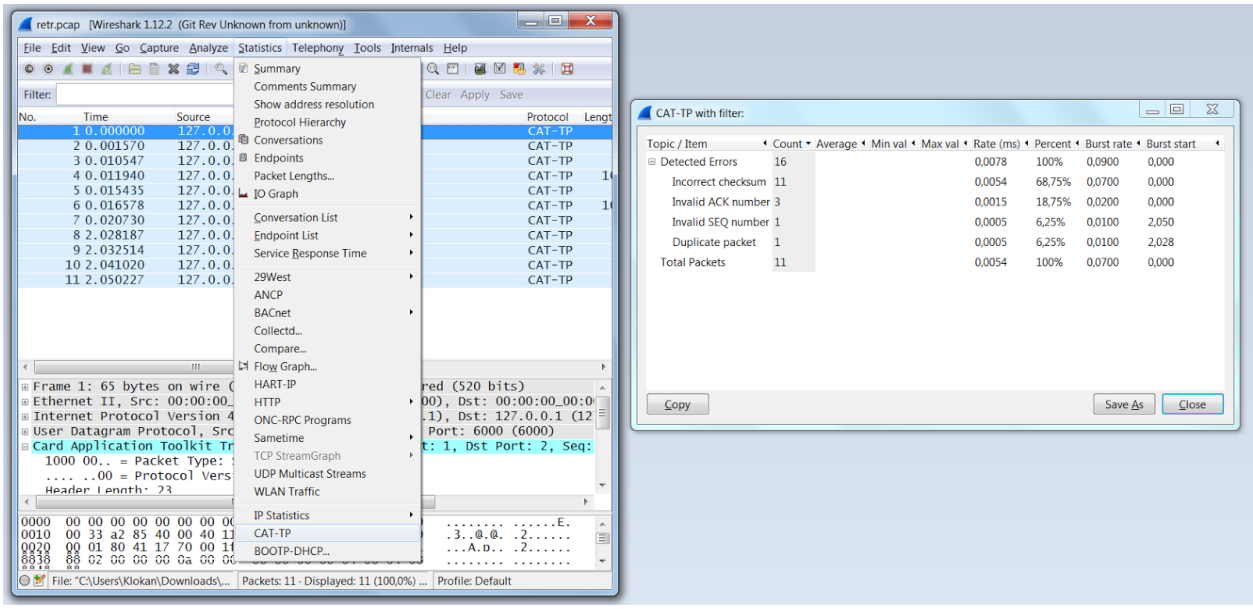
In this case we have to unroll the subtree of this section to see all the sequence numbers inside.

On the picture we can see a yellow color highlighting and two error records in the bottom. Both these UI features are implemented to clearly show detected errors to the user and ease the orientation.

Implemented plugin can detect the following errors:

- Duplicate packet
- Invalid state for packet
- Packet without session
- Invalid sender
- Invalid ACK number
- Invalid SEQ number
- Incorrect checksum

3) Statistics



There is a possibility to view the overall error statistics of the analyzed traffic. To view them, go to the statistics menu and click on the CAT-TP button. On the next window, you can specify a filter to see only some specific records. If you want to see all of them, just continue to the main statistics window (located on the right in the picture). There are two root items. One is there to show the total number of packets, which were dissected by CAT-TP plugin. The second root element represents a subtree with detected errors. List of the errors is same as in the previous section.

Příloha E

CAT-TP Simulator Manual

1) Requirements

- Linux operating system
  - Tested on: Linux Mint 17.1 Rebecca
  - Kernel version: 3.13.0-37-generic
  - Architecture: i686

- Python 2
  - Package name: python
  - Supported versions: 2.5+
  - Tested on version: 2.7.6
- Scapy
  - Package name: python-scapy
  - Supported versions: 2.2.0+
  - Tested on version: 2.2.0

2) Configuration

Both client and server are accepting one argument, which is a XML configuration file. This configuration has to have a root element called *settings* and the following sections placed under this root. A new client thread is executed for each *client* section inside the configuration file.

- *client* element [repeatable]
  - *sport* and *dport* - Source and destionation CAT-TP port
  - *pdusize* - Maximal PDU size
  - *sdusize* - Maximal SDU size
  - *initseq* - Initial sequence number
  - *outputfile* - Path to the output file
  - *iface* - Network interface on which the server is listening
- server element
  - *pdusize* - Maximal PDU size
  - *sdusize* - Maximal SDU size
  - *inputfile* - Path of the input file
  - *iface* - Network interface on which the server should be listening
  - *errormin* - Minimal nr. of packets between error simulations
  - *errorlambda* - Lambda param. for the Exponential distribution (error simulation)

3) Executing the simulator

Default version is Python 2
Open two terminal tabs/windows, one for server and one for client. If your environment uses the Python 2 as a default version after calling *python* command, use the following commands to execute the simulation:

server.py <pathToSettings>
client.py <pathToSettings>

Default version is not Python 2
Execute the following commands if your default version is not Python 2:

python2 server.py <pathToSettings>
python2 client.py <pathToSettings>

4) Stopping the simulator

The client threads and client main process are stopped automatically once the communication with server is completed (both successful and error terminated). The **server** process is running in a infinite loop and can be stopped by pressing **Ctrl+C** combination on a keyboard.

5) Sample configuration

```
<?xml version="1.0"?>
<settings>
	<server>
		<pdusize>4096</pdusize>
		<sdusize>4096</sdusize>
		<inputfile>image.png</inputfile>
		<iface>localhost</iface>
		<errormin>2</errormin>
		<errorlambda>0.5</errorlambda>
	</server>
	<client>
		<sport>1</sport>
		<dport>2</dport>
		<pdusize>1024</pdusize>
		<sdusize>4096</sdusize>
		<initseq>10</initseq>
		<outputfile>image-Client1.png</outputfile>
		<iface>localhost</iface>
```

```
        </client>
        <client>
                <sport>3</sport>
                <dport>4</dport>
                <pdusize>1024</pdusize>
                <sdusize>4096</sdusize>
                <initseq>10</initseq>
                <outputfile>image-Client2.png</outputfile>
                <iface>localhost</iface>
        </client>
</settings>
```