VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

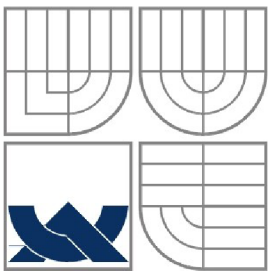
VSTAVANÝ TEST SONDY FLOWMON

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

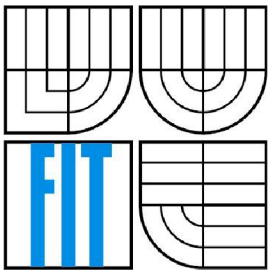
AUTOR PRÁCE
AUTHOR

DANIEL IVANČO

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VESTAVĚNÝ TEST SONDY FLOWMON

SELF TEST OF FLOWMON PROBE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DANIEL IVANČO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN KAŠTIL

BRNO 2009

Abstrakt

Cílem této práce je navrhnout a implementovat autotest sondy FlowMon, monitorující síťový provoz na základě IP toků, která byla vyvinuta během projektu Liberouter. Práce se věnuje teorii testování a kategoriím testů, které nejvíce souvisejí s vyvíjeným autotestem. Zde se také nachází stručný popis monitorování sítí pomocí NetFlow protokolu, spolu s popisem architektury sondy FlowMon. Práce dále obsahuje samotný návrh a řešení autotestu. Součástí řešení jsou dva programové celky. První představuje implementaci generátoru paketů umožňujícího vytvářet všechny potřebné typy paketů a toků používaných druhou částí, která představuje implementaci samotných testů.

Abstract

Aim of this bachelor thesis is to design and implement self test of FlowMon probe. Which is a device monitoring network traffic based on IP flows, developed by Liberouter project team. The thesis includes theories of testing and test categories the self test is related to. There is also a brief description of network monitoring by NetFlow protocol and description of FlowMon probe architecture. Furthermore, the thesis contains the self test design and its description. Final solution consist of two programs. The first one implements packet generator creating all types of required packets and flows used by the second one, which implements the self test itself.

Klíčová slova

černá skřínka, bílá skřínka, šedá skřínka, NetFlow, regresní testování, test zhody, nefunkcionální testování, COMBO, FlowMon

Keywords

black box, white box, gray box, NetFlow, regression testing, conformance testing, nonfunctional testing, COMBO, FlowMon

Citácia

Daniel Ivančo: Vstavaný test sondy FlowMon, bakalárska práca, Brno, FIT VUT v Brne, 2009

Vstavaný test sondy FlowMon

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Jana Kaštila. Ďalšie informácie mi poskytli kolegovia z projektu Liberouter. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……
Daniel Ivančo
10. Máj 2009

Poďakovanie

Chcel by som predovšetkým poďakovať vedúcemu mojej bakalárskej práce Ing. Janovi Kaštilovi za poskytnuté rady a konzultácie, v ktorých sa mi venoval. Ďalej by som chcel poďakovať kolegom z projektu Liberouter a hlavne Ing. Petrovi Špringlovi, ktorí mi tak isto poskytli cenné informácie zúžitkované v tejto bakalárskej práci.

Obsah

1 Úvod

V procese tvorby hardvéru ako aj softvéru je neoddeliteľnou súčasťou ich dôkladné otestovanie. Pretože ani ten najlepší vývojár si nemôže byť istý, že do vytvoreného produktu nezaniesol žiadnu chybu. Bez otestovania novovytvorených produktov sa skôr či neskôr môže objaviť chyba, ktorá môže viesť k rôznym negatívnym dôsledkom ako napríklad k ekonomickým alebo materiálnym stratám.

Jednou s podstatných oblastí informačných technológií, v ktorej sa vyžaduje nepretržitá funkcionalita sú počítačové siete, ktoré sú v dnešnej dobe najpoužívanejším komunikačným prostriedkom medzi ľuďmi. Na to, aby mohli počítačové siete fungovať správne a hlavne efektívne je potrebné ich monitorovanie, ktoré môže odhaliť príčiny rôznych problémov spojených s ich nefunkčnosťou alebo s ich nízkym výkonom.

Spôsobov, ktorými sa dajú počítačové siete monitorovať existuje niekoľko. Táto práca sa venuje konkrétne monitorovacej sonde FlowMon, vzniknutej na projekte Liberouter, ktorý patrí pod výskumný zámer CESNETu.

Sonda FlowMon je hardvérové riešenie, ktoré sa používa vo vysokorýchlostných sietiach, kde bežne dostupné najmä softvérové prostriedky už svojím výkonom nepostačujú. Táto sonda je založená na tzv. COMBO kartách, ktoré sa skladajú z programovateľných hradlových poli (FPGA). [11] Sonda monitoruje siete pomocou tzv. dátových tokov, ktoré sa na vonkajšie kolektory exportujú pomocou protokolov NetFlow a IPFIX.

Cieľom tejto bakalárskej práce je navrhnúť a implementovať test základnej funkčnosti tejto sondy. Úloha testu je overiť jej celkovú funkcionalitu pomocou bežne dostupných softvérových a hardvérových prostriedkov. Pričom pod bežne dostupnými softvérovými prostriedkami sa myslí operačný systém Linux a sieťové nástroje, ktoré sú jeho súčasťou. Pod pojmom bežne dostupné hardvérové prostriedky sa myslí zdroj testovacích paketov, ktoré budú zasielané na sondu a na základe ktorých sa testy budú vyhodnocovať. Tieto pakety budú generované pomocou štandardnej sieťovej karty, prepojenej s COMBO kartou. Nebudú sa používať žiadne iné drahé špecializované hardvérové testovacie nástroje ako napríklad Spirent AX/4000, Spirent TestCenter alebo Agilent N2X, ktoré sú používané aj v projekte Liberouter. Používanie štandardnej sieťovej karty namiesto zmienených hardvérových testerov neumožní uskutočniť isté typy testov, kde sa hlavne vyžaduje veľké množstvo paketov generované za malú jednotku času (napríklad ide o testy priepustnosti). Tak isto do niektorých testov bude potrebné vniesť istú toleranciu vyplývajúcu z nedostatočnej rýchlosti softvéru. Čo je ale pre otestovanie základnej funkčnosti sondy akceptovateľné. Na druhej strane výhoda takéhoto testu spočíva práve v nepotrebnosti špeciálnych hardvérových a softvérových nástrojov, čím sa rozšíri jeho použiteľnosť.

Primárne uplatnenie autotestu je u koncového užívateľa sondy, kde sa môžu odhaliť chyby hlavne v jej hardvérovej časti, ktorá sa mohla poškodiť napríklad prevozom. Pričom nie je potrebné, aby užívateľ vlastnil vyššie zmienené hardvérové testery. Test môže tiež slúžiť ako pomôcka pre vývojárov, ktorí chcú zistiť, či úpravou kódu nespôsobili nefunkčnosť častí, ktoré pôvodne fungovali.

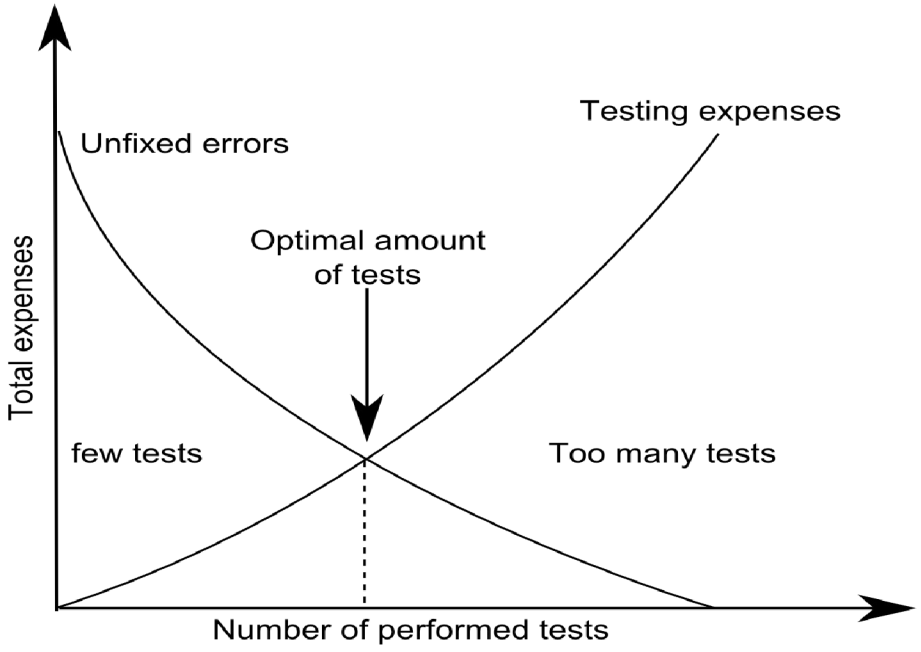
Táto bakalárska práca sa skladá z niekoľkých kapitol. Na začiatku je popísaná teória testovania a testovacie metódy, ktoré najviac súvisia s vyvíjaným autotestom. Ďalej nasleduje stručné vysvetlenie monitorovania sietí na základe NetFlow tokov a popis štruktúry samotnej FlowMon sondy, jej komponent a vlastností, na ktorých sa dané komponenty podieľajú. Nakoniec nasleduje návrh, popis a použitie testov, ktoré sú hlavným cieľom tejto práce.

2 Teória testovania

2.1 Základy testovania

Testovanie je neoddeliteľnou súčasťou procesu vývoja tvorby softvérového ako aj hardvérového produktu. Je veľké riziko predať zákazníkovi slabo alebo vôbec neotestovaný produkt. Takýto produkt môže v konečnom dôsledku vyprodukovať väčšie straty ako zisky. Preto väčšina spoločností vynakladá nemalé prostriedky na vytvorenie kvalitného testerského tímu, ktorý by mal odhaliť čo najviac chýb ešte pred odovzdaním produktu zákazníkovi. Výsledkom dobrého testovania nie je potvrdenie, že daný produkt je bezchybný, naopak, je to objavenie doposiaľ nezistenej chyby. Testovať by sa teda nemalo za účelom potvrdenia správnosti, ale za účelom nájdenia čo najviac chýb. [2] Tak isto každý test musí byť v prípade odhalenia chyby reprodukovateľný. Pretože odhalená chyba, ktorá sa nedá zreprodukovať je len ťažko opraviteľná.

Je potrebné si uvedomiť, že pomocou testovania nie je možné dokázať, že daný produkt je bezchybný. Testovanie slúži len na dokázanie existencie chýb. To znamená, že pri ľubovoľnom počte testov si nemôžeme byť istí 100% bezchybnosťou programu. Čo vedie k myšlienke, že testovanie musí byť dostatočne efektívne. Je potrebné teda nájsť hranicu, kedy prostriedky vynaložené na odhalenie chýb počas vývoja neprevyšujú prostriedky vynaložené na ich opravu v čase, keď sa už produkt používa v praxi (viz Obrázok 2.1). [1]



Obrázok 2.1: Optimálnosť testovania [1]

Pre správne pochopenie účelov testovania si je potrebné vysvetliť tieto základné pojmy: [1]

- Verifikácia systému,
- Validácia systému.

2.1.1 Verifikácia a validácia

Pri testovaní systému treba rozlíšiť, či sa jedná o jeho verifikáciu alebo validáciu.

V prípade verifikácie sa rozumie overenie, či implementácia testovaného systému vyhovuje špecifikácii. Teda či je systém implementovaný správne a nenachádzajú sa v ňom chyby. Verifikáciou systému môže byť napríklad dynamické testovanie či už metódou bielej alebo čiernej skrinky alebo revízia programového kódu. Tieto pojmy sú rozoberané v nasledujúcich kapitolách. Okrem klasického prístupu testovaním existuje ďalšia alternatívna forma verifikácie funkčnosti systému a to pomocou formálnej verifikácie, ktorá dokazuje korektnosť systému pomocou matematických metód. Takáto verifikácia je veľmi náročná a neexistuje záruka, že sa pri jej použití dospeje ku konečnému výsledku.

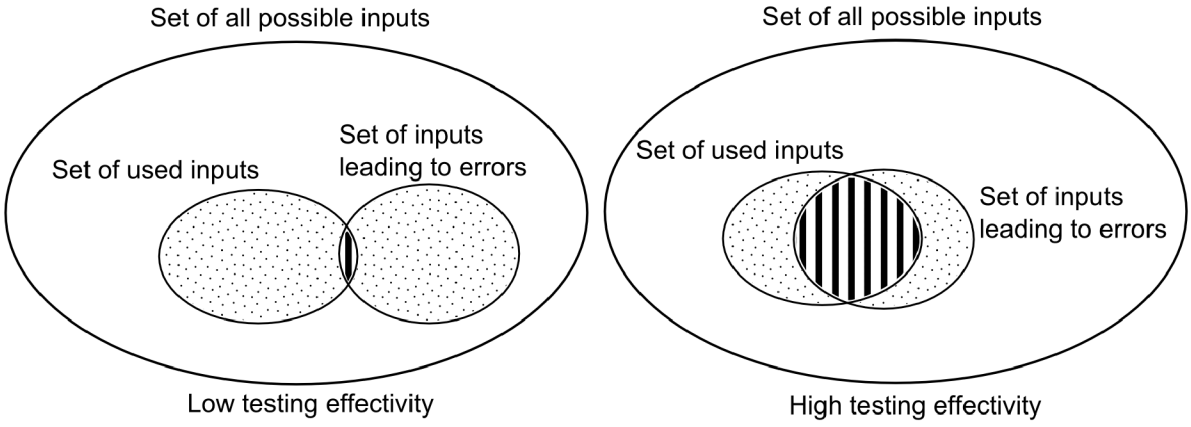
Pod validáciu sa rozumie zistenie, či navrhnutý systém (jeho špecifikácia) vyhovuje požiadavkám zákazníka. Konkrétne sa dá k validácii zaradiť napríklad testovanie špecifikácií, patriace pod statické testovanie čiernej skrinky (viz Kapitola 2.3.3).

Systém môže prejsť verifikáciou a zlyhať vo validácii v prípade, keď je síce dobre implementovaný, ale jeho návrh nezodpovedá reálnemu svetu a požiadavkám zákazníka. Naopak pri úspešnej validácii, keď špecifikácia systému vyhovuje požiadavkám zákazníka, vedie zlyhanie verifikácie taktiež k nepoužiteľnosti produktu, keďže jeho implementácia je chybná. Preto sa pri testovaní kladie dôraz na splnenie obidvoch požiadavok.

2.2 Princíp testovania

Množiny vstupov väčšiny programov sú veľmi veľké, dokonca môžu byť až nekonečné, preto nie je možné otestovať žiadny program úplne. Princíp testovania spočíva vo vyselektovaní istej množiny testovacích vstupov, ktorá bude aplikovaná na testovaný produkt, pričom sa zaznamenajú výstupy vzniknuté z týchto vstupov. Následne sa zaznamenané výstupy porovnajú s očakávanými výsledkami. Ak sa tieto údaje nezhodujú, test odhalil chybu programu. Obecne platí pravidlo, že čím viac chýb sa v priebehu testovania objaví, tým viac neobjavených chýb sa v programe ďalej vyskytuje. [1]

Úlohou testera je nájsť čo najmenšiu množinu testovacích vstupov, ktorá bude viesť k čo najväčšej množine chybných výstupov (viz Obrázok 2.2). Tieto vstupy by sa pri opakovanom testovaní mali časom obmieňať z dôvodu postupne vznikajúcej odolnosti nových verzií programov na ne. Je to spôsobené hlavne navyknutím programátora na konkrétne vstupy testovania. [1]



Obrázok 2.2: Efektivita testovania

2.2.1 Kedy testovať

Forma testovania je do značnej miery ovplyvnená modelom životného cyklu vývoja programu. Exituje niekoľko základných modelov, na ktorých sa zakladá väčšina používaných modelov v praxi: [1]

- Model veľkého tresku,
- Model programuj a opravuj,
- Model vodopádu,
- Špirálový model.

V prípade prvých dvoch modelov, model veľkého tresku a model programuj a opravuj, sa takmer úplne vynecháva plánovanie a návrh programu. Testovanie je tu minimálne a netvorí samostatnú súčasť procesu vývoja, ale je spojené s implementáciou. Posledné dva modely, model vodopádu a špirálový model, sa skladajú z viacerých fáz, návrh a plánovanie tu už majú podstatné miesto. Preto je aj testovanie programu úplne oddelené od implementácie. V modeli vodopádu prebieha kompletné testovanie systému po úplnom dokončení implementácie, zatiaľ čo v špirálovom modeli, je testovanie implementácie iteratívne opakované s každým dokončeným cyklom vývoja programu. [1]

2.3 Typy testov

Testovať konkrétny systém je možné viacerými spôsobmi a stratégiami. Konkrétne sa testy delia do mnohých kategórií, nasleduje výpis najznámejších pohľadov delenia a ich najpopulárnejších zástupcov z vzhľadom k tejto práci.

Z pohľadu dynamiky:

- Statické testovanie,
- Dynamické testovanie.

Z pohľadu úspešnosti testu:

- Testovanie splnením,
- Testovanie zlyhaním.

Z pohľadu úrovne znalostí vnútornej štruktúry testovaného systému:

- Metóda čiernej skrinky (Black Box),
- Metóda bielej skrinky (White box),
- Metóda šedej skrinky (Gray Box).

Z pohľadu účelu testu:

- Regresné testovanie (Regression testing),
- Test zhody (Conformance testing),
- Nefunkcionálne testovanie (Nonfunctional testing).

Z pohľadu vykonávania testov:

- Manuálne,
- Automatizované.

Z pohľadu pokrytia testovaného systému:

- Integračné testovanie (integration testing),
- Testovanie častí (unit testing).

2.3.1 Statické a dynamické testovanie

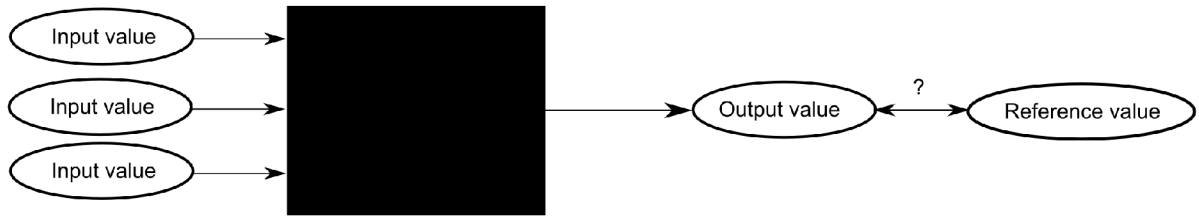
Pri statickom testovaní sa nevyžaduje spustenie testovaného systému. Príkladom môže byť testovanie programovej špecifikácie, ktoré sa používa pri metóde čiernej skrinky. Tento druh testovania sa používa napríklad v prípade, keď ešte nie je hotová implementácia systému poprípade jeho častí alebo keď je potrebné overiť, či má zmysel navrhnutý systém vôbec implementovať. Opakom statického testovania je testovanie dynamické, ktoré testuje správanie systému za jeho behu. [2]

2.3.2 Testovanie zlyhaním a testovanie splnením

Testovanie splnením má za cieľ otestovať, či systém spĺňa základnú funkcionalitu zmienenú v špecifikačnom dokumente, netestujú sa pritom žiadne hraničné podmienky. K systému sa pristupuje opatrnejšie. Zatiaľ čo test zlyhaním cielene hľadá hranice systému, snaží sa odhaliť jeho slabé miesta a dosiahnuť jeho haváriu. V praxi sa uplatňuje postup prvotného testovania splnením a následne, ak je toto testovanie úspešné, sa prechádza k náročnejšiemu testovaniu zlyhaním. [1]

2.3.3 Metóda čiernej skrinky (Black Box)

Metóda čiernej skrinky už podľa názvu napovedá o narábaní s testovaným systémom tak, že sa nevidí do jeho vnútornej štruktúry. Tester sa teda zaujíma, čo daný systém robí, ale nie ako to robí (viz Obrázok 2.3). [1] Táto metóda sa môže používať pri statickom aj dynamickom testovaní.



Obrázok 2.3: Metóda čiernej skrinky

Statické testovanie spočíva v testovaní špecifikácii, nevyžadujúcom beh systému. Testovanie špecifikácie prebieha v dvoch krokoch. Najprv prebehne revízia špecifikácie na vyššej úrovni, kde sa ešte nehľadajú konkrétne chyby, ale zisťuje sa zmyselnosť špecifikácie ako celku. Následne, keď je tester dobre oboznámený so špecifikáciou z pohľadu vyššej úrovne, prichádza na rad testovanie nižšej úrovne, zameranej už na hľadanie jej konkrétnych nedostatkov. V praxi sa metóda čiernej skrinky častejšie používa pri dynamickom testovaní.

Pri dynamickom testovaní metódou čiernej skrinky je pre testera podstatný špecifikačný dokument testovaného systému. [1] Z neho potom vyčíta potrebné informácie o jeho vlastnostiach a vyvodí vstupy, ku ktorým bude očakávať konkrétne výstupy. V prípade, že špecifikačný dokument nie je k dispozícii, je potrebné vychádzať zo samotného správania systému, a za asistencie programátorov, ktorí ho vytvárali a vedia ako sa má systém správať, preštudovať jeho jednotlivé vlastnosti, z ktorých sa odvodí jeho celkové chovanie.

Ako už bolo v Kapitole 2.2 zmienené, najpodstatnejšou úlohou testera je výber tej najvhodnejšej množiny testovacích prípadov. K redukcii tejto množiny sa používa takzvaná metóda rozdelenia ekvivalentných prípadov. [1, 3] Ide o postup, keď sa snažíme pôvodnú nekonečnú množinu všetkých možných vstupov zredukovať na omnoho menšiu, pričom ale musí zostať zachovaná jej efektivita z hľadiska testovania. Použitie tejto metódy spočíva k navrhnutiu tzv.

ekvivalentných množín. Tieto množiny združujú testovacie prípady, ktoré testujú rovnaké vlastnosti systému a teda odhaľujú rovnaké chyby.

Testovanie metódou čiernej skrinky sa delí na dve hlavné oblasti. Prvá je testovanie dát, čo predstavuje všetky možné vstupy, ktoré je možné do systému zadávať. Druhá oblasť je samotné testovanie logiky toku riadenia systému. [1]

2.3.3.1 Testovanie dát

Pri tomto testovaní tvoria jednotlivé triedy ekvivalencie najčastejšie tieto skupiny: hraničné podmienky, subhraničné podmienky, prázdne hodnoty a nesprávne údaje. [1]

Hraničné podmienky predstavujú limitné hodnoty, s ktorými daný systém dokáže ešte pracovať. Testovanie takýchto podmienok spočíva vo výbere hodnôt o pár bodov pod týmito limitmi, kde sa ešte očakáva správna funkčnosť programu v porovnaní s prípadom, keď vstupné hodnoty naopak dané limity o pár bodov prekračujú a systém musí rozpoznať prekročenie tejto hranice. Ideálne je kombinovať testovacie hodnoty z oboch hraničných koncov.

Okrem hraničných podmienok, ktoré sú väčšinou jasné zo špecifikácie existujú aj tzv. subhraničné podmienky, ktoré sú dané implicitne a nemusia byť na prvý pohľad viditeľné. Ide napríklad o vlastnosti implementačného jazyka, platformu na ktorej sa daný program nachádza alebo spôsob akým bol systém implementovaný. Konkrétnym príkladom môžu byť mocniny dvoch, s ktorými väčšina súčasných procesorov pracuje, keďže ide o reprezentáciu binárnej sústavy.

Ďalšou podstatnou triedou ekvivalencie sú prázdne alebo nezadané hodnoty, v prípade keď sa nejaká na vstupe vyžaduje. Pri implementácii sa často zabúda na ošetrenie týchto situácii, čo môže ľahko spôsobiť pád systému.

Poslednou triedou ekvivalencie je privedenie na vstup neplatné údaje. Táto trieda ekvivalencie sa používa pri testovaní zlyhaním, zatiaľ čo sa na doteraz predstavené triedy ekvivalencie používalo testovanie splnením. Väčšinou sa zadávanie neplatných údajov používa až nakoniec, keď všetky predošlé testy prebehli úspešne. Za neplatné údaje sa považujú všetky dáta, ktoré typovo nesúvisia s očakávanými vstupmi. Napríklad, keď je v prípade čísla zadaný znak a podobne.

2.3.3.2 Testovanie toku riadenia systému

Táto časť sa zaoberá testovaním logiky riadenia systému. Logika systému sa dá reprezentovať pomocou stavov, v ktorých sa môže systém nachádzať a prechodov určujúcich, za akých podmienok sa tieto stavy menia. [1]

Keďže už aj pri jednoduchých systémoch, kde je počet stavov menší, vzniká relatívne veľké množstvo ich všetkých možných kombinácií, aj tu sa uplatňujú rôzne metódy na ich redukciu: [1]

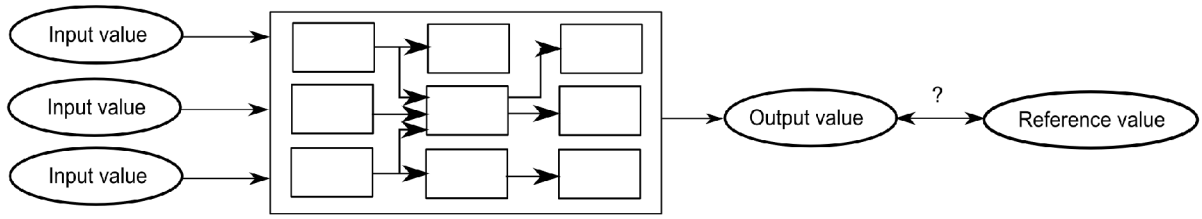
- dostať sa do každého stavu aspoň raz bez ohľadu na to, aké stavové prechody boli použité,
- testovanie takých prechodov, ktoré sa budú pravdepodobne používať čo najčastejšie a preto si vyžadujú najväčšiu pozornosť,
- otestovanie najneobvyklejších prechodov, kde je vysoká pravdepodobnosť, že neboli vyskúšané ani samotnými programátormi,
- otestovanie všetkých chybových stavov vrátane návratu z nich,
- náhodné testovanie stavov.

Po určení stavov, ktoré sa budú testovať, je potrebné následne určiť tzv. stavové premenné definujúce tieto stavy. Tieto premenné poskytujú informáciu, že sa daný systém nachádza v tom ktorom konkrétnom stave. Testy sa vyhodnocujú práve na základe týchto stavových premenných. [1]
Okrem klasických testov splnením, keď sa overuje správny prechod medzi jednotlivými stavmi sa taktiež uplatňuje testovanie zlyhaním. Ide napríklad o testy opakovania, stresu a záťaže. Test opakovaním je založený na neustále sa opakujúcej operácii, kde sa už po niekoľkých opakovaniach môže prejaviť doposiaľ neznáma chyba. Úlohou takéhoto testu je overiť správnu prácu programátora s pamäťou. Pri stresovom testovaní je systém limitovaný prostriedkami, ktoré môže využívať,

napríklad malá pamäť. Je to konkrétny typ testu hraničných podmienok. Opak stresového testovania je záťažové stresovanie, keď systém nie je limitovaný žiadnymi prostriedkami, ale je nútený pracovať čo najdlhšiu dobu s čo najväčším možným množstvom dát. [1, 2]

2.3.4 Metóda bielej skrinky (White Box)

Podobne ako pri metóde čiernej skrinky aj v tejto metóde sa dá aplikovať ako statické tak aj dynamické testovanie. Hlavný rozdiel bielej skrinky oproti čiernej skrinke spočíva v tom, že tentoraz je známa štruktúra testovaného systému, poznáme jeho programový kód a vidíme ako spolu jednotlivé časti spolupracujú (viz Obrázok 2.4). Pri návrhu testov sa zameriavame teda nielen na otázku čo systém robí, ale aj ako to robí. Tieto informácie môžu testerovi pomôcť lepšie navrhnúť triedy ekvivalencie. [1]



Obrázok 2.4: Metóda bielej skrinky

2.3.4.1 Statické testovanie

Pri statickom testovaní bielej skrinky ide hlavne o verifikáciu návrhu a programového kódu. Chyby sa hľadajú bez samotného spustenia programu. Tento proces sa tiež nazýva formálna revízia. [1]

Existujú rôzne nástroje, ktoré do istej miery môžu verifikáciu kódu zautomatizovať, čo môže odbremeniť testerov a poskytnúť im viac času, aby sa zamerali na dôkladnejšie otestovanie najpodstatnejších častí. Veľkou výhodou statického tetovania sú nízke náklady na odstránenie nájdenej chyby. Na druhej strane existujú tímy, ktoré neprisudzujú statickému testovaniu príliš veľkú efektivitu a považujú to za stratu prostriedkov a drahocenného času. [1]

2.3.4.2 Dynamické testovanie

Pri dynamickom testovaní, taktiež známom pod názvom štrukturálne testovanie, testujeme konkrétny programový kód, sledujeme jeho chovanie počas samotného behu systému. Práve znalosť štruktúry systému pri metóde bielej skrinky prispieva k dokonalejšiemu návrhu testov, ktoré môžu byť viac šité na mieru daného systému. Je potrebné nezamieňať si dynamické testovanie bielej skrinky s ladením programu. Kým cieľom testovania je nájdenie chyby, ladenie sa používa ako prostriedok na opravu nájdenej chyby. Medzi dynamické testovanie bielej skrinky môžeme zaradiť nasledujúce spôsoby testovania: [1]

- testovanie jednotlivých funkcií systému,
- testovanie systému ako celku na najvyššej úrovni, pričom testy sú prispôsobené programovému kódu,
- testovanie priamou modifikáciou hodnôt jednotlivých premenných systému,
- testovanie na základe kvantity kódu, ktorý bol vplyvom testu vykonaný a následné modifikácie testov na základe týchto informácií.

Podobne ako pri metóde čiernej skrinky, aj tu sa delí testovanie systému na dátovú a logickú časť. Dáta v tomto prípade reprezentujú premenné a dátové štruktúry programového kódu, vstupy a výstupy z rôznych periférnych zariadení. Logická časť sa zameriava na tok riadenia programu.

2.3.5 Metóda Šedej skrinky (Gray Box)

Táto metóda je akýmsi zlúčením predošlých dvoch metód čiernej a bielej skrinky. Stratégia testovania je založená na vedomostiach vnútornej štruktúry testovaného systému, aj keď tieto vedomosti nemusia byť také detailné ako pri metóde bielej skrinky. K samotnému testovaniu sa pristupuje metódou čiernej skrinky. Celá myšlienka spočíva v tom, že ak má tester isté znalosti o vnútorných častiach, z ktorých sa systém skladá, môže lepšie navrhnúť jeho testovanie zvonka. Treba si uvedomiť, že od bielej skrinky sa táto metóda líši v tom, že tester stále pristupuje k systému zvonka ako pri metóde čiernej skrinky, nesnaží sa pokryť všetky jeho vnútorné časti detailne. [2]

2.3.6 Manuálne a automatizované testovanie

Nie vždy je spôsob testovania formou manuálneho zadávania vstupov do testovacieho systému najefektívnejší, čo môže byť napríklad v prípade regresných testov (viz Kapitola 2.3.8) zbytočne opakujúca sa práca. Preto sa namiesto zmieneného manuálneho testovania často používa možnosť automatických testov, keď je proces testovania zautomatizovaný použitím softvéru a nevyžaduje sa testerova prítomnosť v priebehu testu. [1] Jeho úlohou je po ukončení testovania už len zanalyzovať výsledky a zhodnotiť úspešnosť, poprípade neúspešnosť testu a jeho dôsledky.

2.3.7 Testovanie častí a integračné testovanie

Pri testovaní častí sa testujú jednotlivé časti systému osobitne, teda nezávisle na ostatných. Naopak pri integračnom testovaní sa testuje funkčnosť týchto častí spoločne. Existujú dva spôsoby integračného testovania. V prvom spôsobe sa po otestovaní jednotlivých častí, postupne integrujú a testujú vo väčších celkoch, až kým sa netestuje celý systém ako celok. Výhoda tohoto postupu spočíva v tom, že ak je zistená chyba až po spojení viacerých modulov, pravdepodobne bude problém v ich komunikácii, keďže moduly boli predtým úspešne otestované jednotlivo. Týmto spôsobom sa znižuje riziko prekrývania viacerých chýb súčasne, čo zjednodušuje ich lokalizáciu. Okrem zmieneného postupu zdola nahor sa dá v prípade, keď je už systém kompletne naimplementovaný postupovať aj v opačnom poradí, ide teda o testovanie zhora nadol. [4]

2.3.8 Regresné testovanie

Tento typ testovania sa používa na ubezpečenie, že časti systému, ktoré v minulosti fungovali správne, fungujú tak aj naďalej. Používa sa to najmä pri implementačných zmenách, či už kvôli opravám chýb alebo pridávaním nových vlastností. [2] Pretože podľa skúseností často nastáva situácia, že sa programátori schyľujú k tým istým chybám, ktoré boli už v minulosti opravené. [1] Keďže ide o testy, ktoré sa spúšťajú relatívne často (pri každej implementačnej zmene) býva zvykom tieto testy automatizovať.

2.3.9 Test zhody

Test zhody určuje, či sa testovaný systém zhoduje s požadovanými vlastnosťami alebo štandardmi. V prípade potvrdenia splnenia štandardov, býva systém testovaný samotnou externou organizáciou, ktorá po úspešnom otestovaní certifikuje daný produkt. Príkladom testu zhody môže byť testovanie prekladačov, kde sa zisťuje, či sú splnené štandardy programovacieho jazyka, ktorý prekladač podporuje. [15]

2.3.10 Nefunkcionálne testovanie

Ide o testy, ktorých úlohou nie je overiť funkcionalitu systému, ale jeho reakciu na nezmyselné a nepredvídané vstupné dáta. [2] Medzi najznámejšie nefunkcionálne testy patrí napríklad záťažové testovanie, pri ktorom sa sleduje ako je systém schopný vysporiadať sa s veľkým množstvom vstupných dát. Ďalší príklad predstavuje bezpečnostné testovanie, overujúce mieru bezpečnosti dát a zraniteľnosť systému.

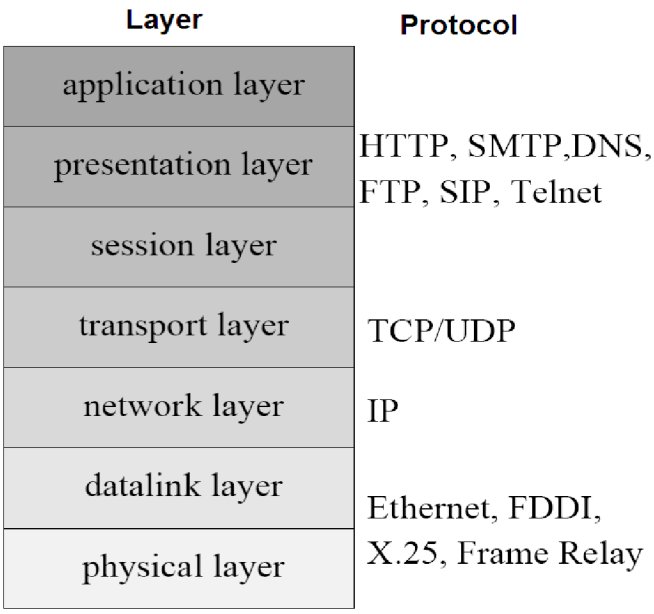
2.4 Kategorizácia testu sondy FlowMon

Test, ktorý je vytváraný v rámci tejto bakalárskej práce môžeme kategorizovať do viacerých vyššie zmienených skupín. V prvom prípade ide o testovanie šedej skrinky, pretože sa v teste budú používať metódy čiernej skrinky, kde sa k sonde pristupuje zvonka ako k celku, pričom je známa štruktúra a vlastnosti jej vnútorných častí. Ďalej pôjde o test dynamický, keďže sa testuje správanie sondy za jej behu. Tento test bude tak isto automatizovaný prostredníctvom shell skriptu, ktorý jednotlivé testy zastrešuje, netreba ich preto spúšťať manuálne. Po ukončení testovania je vygenerovaná správa o úspešnosti alebo neúspešnosti jednotlivých testov. Z pohľadu úspešnosti testu pôjde väčšinou o testovanie splnením. Tak isto sa budú vyskytovať niektoré testy overujúce reakciu na chybne zadané vstupy presahujúce nastavené limity, čo patrí medzi testovanie zlyhaním. Test by sa z hľadiska využitia dal zaradiť k regresným testom, ktoré sa môžu opakovane spúšťať pri opravách kódu. Tak isto sa môže zaradiť k testu zhody, kde sa testuje zhoda vlastností sondy s vlastnosťami, ktoré sú popísané v špecifikácií. Z hľadiska pokrytia testu pôjde o integračné testovanie, kde sa postupne testuje spolupráca jednotlivých častí celku.

3 Sonda FlowMon

3.1 Monitorovanie sietí pomocou NetFlow

Všetky informácie, ktoré sa prenášajú prostredníctvom počítačových sietí sú zapúzdrené do tzv. paketov. Jednotlivé pakety obsahujú informácie patriace do rôznych sieťových protokolov, ktoré predstavujú pravidlá založené na štandardoch, určujúce formát dát, komunikáciu a dátový prenos medzi dvoma komunikujúcimi bodmi. Jednotlivé protokoly sa podľa kontextu komunikácie medzi sebou spájajú do vrstiev. [5] Popis architektúry jednotlivých vrstiev a protokolov, ktoré do nich patria sú popísané v referenčnom OSI modeli (viz Obrázok 3.1). [6, 7]



Obrázok 3.1: OSI model [6]

Dôležitou úlohou pri spravovaní sietí je vedieť sieťovú prevádzku monitorovať, či už kvôli zaisteniu jej efektívnej činnosti, detekcie neželaných vonkajších útokov alebo k presnému meraniu jej parametrov napríklad za účelom spoplatnenia poskytovaných služieb.

Jeden z možných spôsobov monitorovania sietí je agregácia paketov do tzv. IP tokov, ktoré sú definované protokolom NetFlow vyvinutým spoločnosťou Cisco Systems.

Jeden tok reprezentuje spojenie medzi dvoma komunikujúcimi bodmi. Konkrétne reprezentuje agregáciu najmä TCP, UDP a ICMP paketov (ide o pakety patriace pod IP protokol sieťovej vrstvy referenčného OSI modelu) na základe 7 kľúčových položiek: [8]

- zdrojová a cieľová IP adresa,
- zdrojový a cieľový port,
- protokol transportnej vrstvy,
- rozhranie na ktorom boli pakety zachytené,
- Type of Service (ToS) z IP hlavičky.

NetFlow záznam môže v závislosti od verzie obsahovať rôzne informácie o vzniknutom toku. Najrozšírenejšie sú NetFlow verzie 5 a 9. Verzia 9 je rozšírením verzie 5 a môže obsahovať viacero

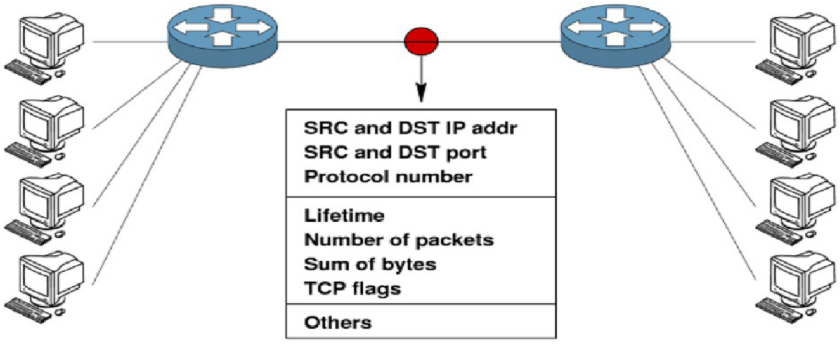
typov položiek, pričom ich konkrétna podoba je určená šablónou. Verzia 5 obsahuje okrem vyššie zmienených 7 položiek aj nasledujúce informácie: [8]

- verzia NetFlow záznamu,
- číslo sekvencie,
- TCP flags v prípade TCP paketu,
- počet a súčet veľkostí paketov vytvárajúcich tok,
- časové značky reprezentujúce trvanie toku,
- routovacie informácie transportnej vrstvy.

To že NetFlow nie je oficiálnym internetovým štandardom má v blízkej budúcnosti vyriešiť tzv. IPFIX (IP Flow Information eXport) odvodený z implementácie NetFlow verzie 9. [8]

3.2 Charakteristika sondy FlowMon

FlowMon sonda patrí medzi pasívne monitorovacie zariadenie sietí. Monitorovanie prebieha na základe protokolu NetFlow, pričom je sonda pre ostatné sieťové zariadenia neviditeľná (viz Obrázok 3.2). [9]



Obrázok 3.2: FlowMon sonda v sieti [9]

Sonda je postavená na PC architektúre. Konkrétne sa skladá z hardvérovej a softvérovej časti, pričom softvérová časť je implementovaná pod operačným systémom Linux. Jej úlohou je zapúzdrovať komunikáciu s hardvérovou časťou tvorenou COMBO kartou (viz Obrázok 3.3). [10] Ide o PCI kartu zloženú z programovateľných hradlových polí FPGA. [11] COMBO karta poskytuje dostatočnú hardvérovú akceleráciu, potrebnú pre monitorovanie vysokorýchlostných sietí (rádovo niekoľko GB/s), v ktorých už konkurenčné riešenia svojim výkonom nepostačujú.

Obrázok 3.3: COMBO karta [10]

Sonda je schopná zbierať údaje o sieťových tokoch a exportovať ich na externé kolektory vo formátoch NetFlow verzie 5, 9 a IPFIX. [9]

3.3 Vlastnosti sondy

Pre prispôsobenie sondy konkrétnym požiadavkám administrátora a parametrom siete, ktorú je potrebné monitorovať, existuje možnosť nastavenia jej viacerých vlastností:

- **Vstupné vzorkovanie paketov**
  Táto vlastnosť umožňuje nastaviť pomer paketov, ktoré budú spracovávané sondou a tých ktoré budú zahodené. Pomocou softvérových nástrojov sa dajú nastaviť dva hlavné typy vzorkovania a to je konštantné a náhodné. Pri konštantnom vzorkovaní je zahodený každý n-tý paket v poradí zodpovedajúci nastavenému pomeru, kým pri náhodnom vzorkovaní sa o zahodení konkrétneho paketu rozhoduje na základe pravdepodobnosti, odvodenej z aktuálneho pomeru vzorkovania. Vzorkovanie prijatých paketov sa používa na ochranu sondy v prípade veľkej sieťovej prevádzky, keď už jej výkon nie je úplne postačujúci.
- **Filtrovanie paketov na základe ich veľkostí**
  Umožňuje nastaviť horný a dolný veľkostný limit paketov, ktoré majú byť prijaté
- **Vzorkovanie sample and hold**
  Toto vzorkovanie sa narozdiel od klasického vzorkovania líši v tom, že ak spracovávaný paket patrí do jedného z už zaznamenaných tokov, vzorkovanie sa na ňom neuplatňuje a nedochádza k jeho prípadnému zahodeniu. Tak isto je potrebné nastaviť hodnotu treshold, udávajúcu pri akom zaplnení pamäte s tokmi sa má vzorkovanie realizovať.
- **Aktívny a neaktívny timeout**
  Aktívny timeout predstavuje maximálny časový interval v sekundách, počas ktorého môžu byť prijímané pakety patriace do jedného toku. Pri jeho prekročení je daný tok zo sondy exportovaný. Neaktívny timeout naopak určuje maximálny čas, počas ktorého môže byť ľubovoľný tok neaktívny (nie sú prijímané žiadne pakety, ktoré sú súčasťou daného toku) pred tým ako bude exportovaný do softvérovej časti. Oba typy timeoutov pracujú na základe časových značiek (timestamps), ktoré sú prijatým paketom prideľované IBUF jednotkou (viz Kapitola 3.5).

3.4 Softvérová časť

Softvérová časť sprostredkúva komunikáciu medzi aplikáciami, ktoré so sondou pracujú a COMBO kartou, v ktorej je implementovaná hardvérová časť sondy.

Na najnižšej úrovni operačného systému sa nachádzajú ovládače, ktoré sprostredkúvajú komunikáciu s COMBO kartou na tej najzákladnejšej úrovni. Ďalej sú to softvérové nástroje zapúzdrujúce nahrávanie firmvéru (designu) do karty, vyčítavanie informácii zo sondy a umožňujúce jej nastavenia. Nakoniec sú to exportéry, ktoré exportujú zachytené toky na zvolené kolektory podporujúce už zmienené NetFlow protokoly. [12] Z pohľadu autotestu medzi najpodstatnejšie patria tieto nástroje:

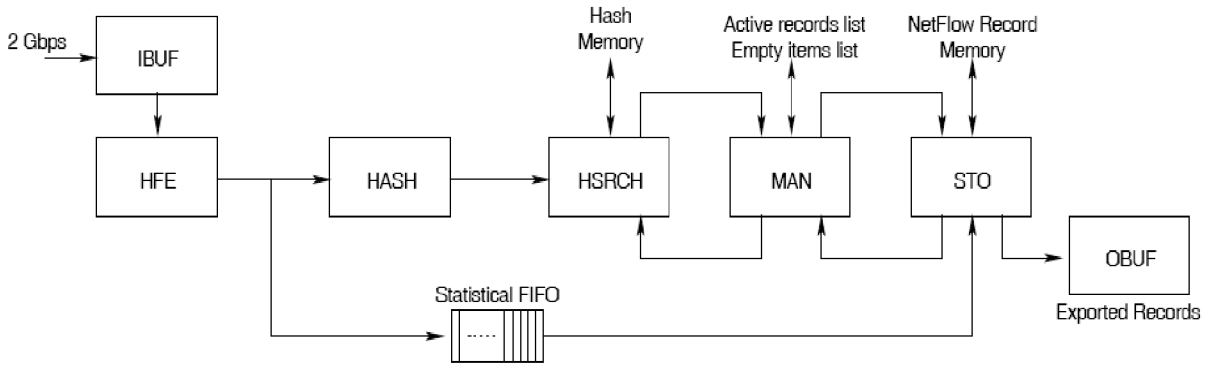
- `flowmon`
  Skript zastrešuje celý proces nahrania (nabootovania) firmvéru FlowMon sondy do COMBO karty.
- `sw_obuf_ctl`
  Softvérový nástroj zaisťujúci vyčítavanie tokov zo softvérového buffera.

- `flowmonctl`
  Softvérový nástroj umožňujúci modifikovanie nastavení rôznych vlastností sondy ako je nastavenie timeoutov, vzorkovania a podobne.
- `ibufctl`
  Nástroj umožňujúci prácu s jednotkou IBUF. Umožňuje napríklad vyčítanie paketov prijatých touto jednotkou alebo nastavenie intervalu veľkostí prijímaných paketov.
- `flowmonlkm`
  Tento nástroj umožňuje nahranie modulov ovládača FlowMon do jadra operačného systému (kernel).

3.5 Hardvérová časť

Hardvérová časť sondy sa skladá z týchto kľúčových jednotiek (viz Obrázok 3.4): [13]

- **Input Buffer (IBUF),**
- **Header Field Extractor (HFE),**
- **Hash unit (HASH),**
- **Hash search (HSRCH),**
- **Manager (MAN),**
- **Storage (STO),**
- **Statistical FIFO,**
- **Output Buffer (OBUF).**



Obrázok 3.4:Architektúra sondy FlowMon [13]

Všetky pakety, prijímané na sonde sú na začiatku spracované **IBUF** jednotkou. Je to vstupný buffer, ktorý overuje kontrolný súčet (Frame Check Sequence nachádzajúci sa na datalinkovej vrstve OSI modelu) prijatých rámcov, ich veľkosť a pridelí každému paketu tzv. časovú známku (čo je 32 bitový čítač pracujúci s rozlíšením 640 ns [10], reprezentujúci okamih detekcie paketu danou jednotkou). Tak isto je tu implementované vstupné vzorkovanie paketov.

Pakety spracované **IBUF** jednotkou, sú následne predané do **HFE** jednotky. Ide o procesor s RISC (Reduced Instruction Set Computer) architektúrou. Úlohou procesora je rozparsovať IP a TCP, UDP, ICMP hlavičky paketov a použiť získané položky na vytvorenie unikátneho kľúča určujúceho tok, do ktorého spracovávaný paket patrí. Kľúč pozostáva zo zdrojovej a cieľovej IP adresy, zdrojového a cieľového portu a protokolu transportnej vrstvy. Vytvorený kľúč spolu s parametrami ako priradené časové známky, dĺžka paketu a typ služby (ToS) tvorí informáciu o danom pakete zaslanú do **Statistical FIFO** jednotky. Zároveň sa vyššie zmienený kľúč zasiela do **HASH** jednotky, kde sa z neho vytvoria dva typy hashovacích hodnôt, z ktorých jedna určuje adresu položky v zázname tokov a druhá určuje, či položka na danej adrese naozaj zodpovedá kľúčom predstavujúci hľadaný tok. Následne je pomocou týchto dvoch hodnôt daná položka vyhľadávaná v jednotke **HSRCH**. Vyhľadávanie môže skončiť viacerými možnými výsledkami, ktoré rozhodnú, či má byť obnovený už existujúci záznam o toku alebo má byť vytvorený nový záznam, poprípade má byť v dôsledku plnej pamäte vybraný záznam, ktorý sa exportuje zo sondy. [13] Tak isto je v tejto jednotke implementované vzorkovanie typu sample and hold.

Okrem modifikácie záznamov tokov na základe príkazov z **HSRCH** jednotky sa **Manager (MAN)** jednotka tiež stará o udržovanie stavu aktívneho a neaktívneho timeoutu. [13]

Jednotlivé dáta pochádzajúce zo zásobníka **Statistical FIFO** ukladá v pamäti jednotka **STO (Storage)**, riadená príkazmi jednotky **HSRCH** prostredníctvom **MAN**, ktorý prekladá adresné priestory medzi **STO** a **HSRCH**. Tak isto tu prebieha kontrola aktívneho timeoutu. Exportované toky nakoniec putujú do jednotky **OBUF**, ktorá tvorí rozhranie medzi softvérovou a hardvérovou časťou sondy. [13]

4 Autotest sondy FlowMon

4.1 Motivácia

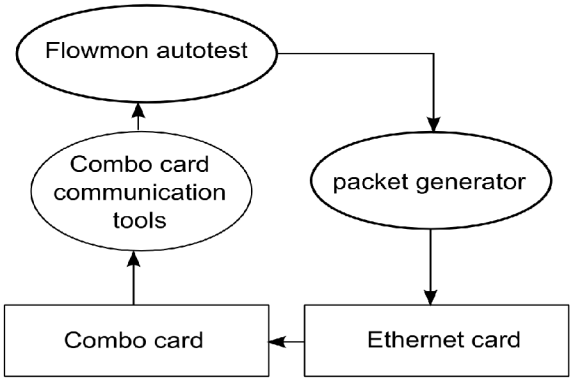
Pre potvrdenie správnej funkčnosti sondy slúži v rámci projektu Liberouter vyčlenený testerský tím, ktorého úlohou je odhaľovať implementačné chyby a zároveň uskutočňovať nefunkcionálne testovanie ako napríklad testy priepustnosti potvrdzujúce schopnosť sondy spracovávať extrémne množstvá paketov na sieti, crash testy snažiace sa dosiahnuť pád sondy a podobne. Tieto typy testov sú testerským tímom zvládnuté na uspokojivej úrovni.

Sonda FlowMon je ale často používaná rôznymi partnermi, ktorí s projektom spolupracujú, kde sa nasadzuje v reálnych sietiach. Sonda je teda používaná mimo prostredia, kde sa vyvíja, pričom sa môže ľahko stať, že sa hardvérová časť transportom karty do destinácie nasadenia poškodí. Preto je potrebné overiť funkcionalitu sondy na mieste jej použitia. Tu ale chýbajú sofistikované hardvérové testery využívané testerskou skupinou v prostredí projektu Liberouter. Preto je nutné vystačiť si s bežne dostupnými hardvérovými zariadeniami, ktoré dokážu so sondou komunikovať, čo sú najmä klasické sieťové karty nachádzajúce sa väčšinou v každom serveri. Tak isto je nutné v prípade detekovanej chyby poskytnúť čo najviac údajov pomáhajúcich pri jej lokalizácii a reprodukcii, pre urýchlenie jej opravy. Celý proces testovania by mal z časového hľadiska trvať čo najkratšiu dobu, aby boli jeho výsledky čo najskôr použiteľné. Práve na tieto účely vznikol autotest sondy FlowMon implementovaný v rámci tejto bakalárskej práce. Autotest môže byť tiež používaný samotnými vývojármi, ktorí môžu ľahko skontrolovať základnú funkčnosť sondy, bez výraznejšieho zaťaženia testerského tímu.

4.2 Princíp testovania

Väčšina testov je založená na zasielaní paketov s rôznymi parametrami a v rôznych počtoch zo sieťovej karty do sondy. Na generáciu paketov sa využíva generátor, ktorý je tak isto implementovaný v rámci tejto bakalárskej práce. Vygenerované pakety sú pomocou pcap (packet capture) rozhrania uložené do súboru a následne pomocou nástroja `tcpreplay` zaslané na sondu. Tieto nástroje boli vybrané hlavne preto, že sú štandardnou súčasťou operačného systému Linux, v ktorom je sonda používaná. Výstupy sondy, ktoré sú získané pomocou softvérových nástrojov vyvinutých na projekte (viz Kapitola 3.4), sa následne porovnávajú s očakávanými výsledkami.

Pre potreby autotestu je teda nutné mať prepojené dve rozhrania a to rozhranie COMBO karty s ľubovoľným rozhraním Ethernetovej sieťovej karty. Obidve komponenty sa musia nachádzať na tom istom serveri (viz Obrázok 4.1).



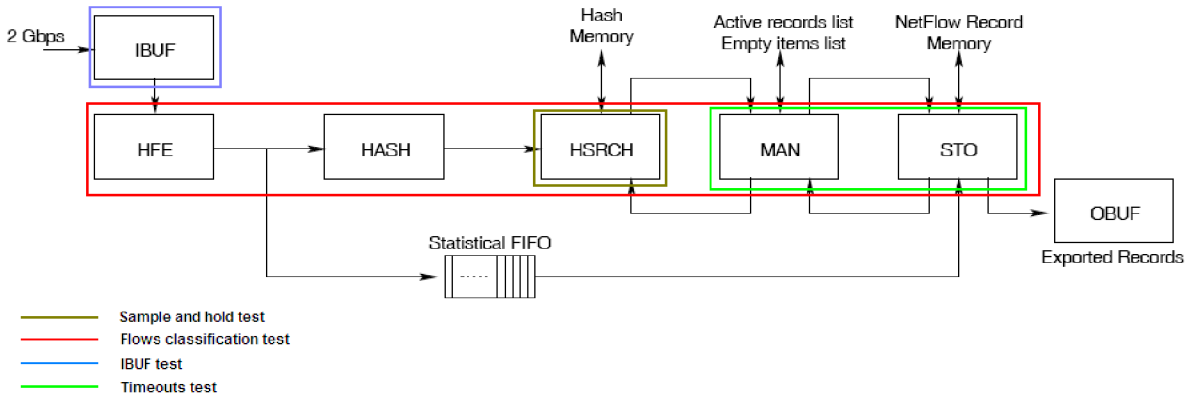
Obrázok 4.1: Prepojenie rozhraní komponent pre potreby autotestu

Je potrebné brať ohľad na parametre ako je najmä rýchlosť generácie paketov, rýchlosť ich zasielania na sondu a presnosť s akou dokážu byť odosielané, keďže sonda FlowMon je určená na monitorovanie vysokorýchlostných sietí (rádovo niekoľko Gb/s) a štandardné sieťové karty a softvérové nástroje nedokážu v takýchto rýchlostiach operovať.

4.3 Návrh testovaných častí sondy

V autoteste sondy FlowMon je snaha otestovať čo najširšiu škálu jej hardvérových vlastnosti nastaviteľných pomocou dostupných softvérových prostriedkov.

Jednotlivé vlastnosti sondy sú testované približne v poradí v akom sa aplikujú pri spracovávaní prijatých rámcov hardvérom (viz Obrázok 4.2). Testovanie správnosti softvéru exportujúceho zachytené toky na kolektory (konkrétne ide o formáty NetFlow v5, v9 a IPFIX) nie je súčasťou autotestu. Pretože ide o testovanie softvérovej časti, ktorú ma za úlohu dôkladne otestovať testerský tím a pri vzniku funkčnej verzie nie je možné negatívne ovplyvniť jej funkcionalitu napríklad pri transporte sondy k zákazníkovi, kde má autotest primárne použitie.



Obrázok 4.2: Architektúra sondy FlowMon rozdelená podľa testovaných vlastností

Ako prvá je testovaná **IBUF** jednotka a jej schopnosť prijať správny počet paketov pri rôznom nastavení vzorkovania a filtrovať ich podľa veľkostí. Testovanie správneho kontrolného súčtu paketov nebolo do testu zahrnuté, keďže použité nástroje na ich generáciu nemajú možnosť tento parameter modifikovať.

Ďalej je testovaný export detekovaných tokov na základe **aktívneho a neaktívneho timetoutu**. Korektný export tokov sa overuje prostredníctvom detekovaného počtu tokov a tak isto aj pomocou ich časových známok, kde sa absolútna hodnota rozdielu začiatočnej a koncovej známky každého toku musí približovať hodnotám testovaných timeoutov. Na základe rozlíšenia pridelených časových známok (viz Kapitola 3.5) sa prevádzajú nastavené hodnoty timeoutov v sekundách do 32 bitových hodnôt porovnateľných s týmito známkami. Tiež je ošetrená situácia pretečenia čítačov. V tomto prípade sa počíta podľa vzťahu 4.1, kde R je rozdiel časových známok, K je hodnota koncovej časovej známky a S je hodnota začiatočnej časovej známky toku.

$$R = K + (2^{32} - S)$$

(vzťah 4.1)

Pri porovnávaní časových známok je nutné zaviesť istú toleranciu kvôli už vyššie zmieneným nepresnostiam použitých komponent pri testovaní. Na funkcionalite tejto vlastnosti sa najvýznamnejšie podieľajú jednotky MAN a STO.

Ďalší dodatkový test sa zameriava na správnu funkčnosť vzorkovania typu **sample and hold**, ktorý je implementovaný v jednotke HSRCH. Vzorkovanie je konkrétne testované tak, že sa na sondu pošle viacero tokov, pričom pakety tvoriace jeden tok nasledujú hneď za sebou. Pri konkrétnom nastavení pomeru vzorkovania je počet paketov rovnajúci sa hodnote menovateľa nastaveného pomeru (ktorý bude vždy menší ako celkový počet paketov v jednom toku) preskočený, až kým nenasleduje paket, z ktorého sa vytvorí záznam o novom toku. Do tohoto toku sú automaticky pridelené všetky ostatné pakety doň patriace. Rovnaký postup sa uplatní aj na všetky ostatné toky. Dokopy má každý prijatý tok pozostávať z počtu paketov rovnajúcemu sa vzťahu 4.2, kde P je výsledný počet paketov, S je počet všetkých zaslaných paketov na sondu patriacich do daného toku a R je hodnota menovateľa nastaveného pomeru vzorkovania.

$$P = S - R$$

(vzťah 4.2)

Pri nastavení náhodného typu vzorkovania je nutné opäť zaviesť adekvátnu toleranciu výsledkov.

Posledná a zároveň najkľúčovejšia vlastnosť sondy FlowMon je **klasifikácia paketov do tokov**. Táto vlastnosť zahrňuje spoluprácu väčšiny hardvérových jednotiek, z ktorých sa sonda skladá (viz Obrázok 4.2). Test pozostáva zo zasielania paketov s rôznymi parametrami a následnej kontroly týchto položiek, ktoré sa na definícii tokov podieľali:

1. počet a súčet veľkostí paketov patriacich do toku,
2. zdrojová a cieľová IP adresa,
3. zdrojový a cieľový port,
4. protokol transportnej vrstvy.

Vynechaná je kontrola rozhrania COMBO karty, na ktorom bol tok zachytený, keďže sa v jednom okamihu dá testovať len jedno rozhranie. V situácií, keď je detekovaná strata paketov je tiež vynechaná kontrola počtu paketov tvoriacich jeden tok a súčtu ich veľkostí.

4.4 Generátor paketov

Ako už bolo zmienené v úvode, namiesto využívania hardvérových generátorov paketov sa v testoch používa softvérový generátor implementovaný v rámci tejto bakalárskej práce. Konkrétne je použitý jazyk `Python`, využívajúc rozšírenie `Scapy` (viac v [14]). Dôvod uprednostnenia implementácie vlastného generátora pred použitím už existujúceho je hlavne ten, že v uskutočnenom prieskume nebolo nájdené žiadne existujúce riešenie, ktoré by zahŕňalo všetky vlastnosti potrebné pre test sondy.

Výhodou použitia vlastného generátora oproti napríklad napevno vytvorenými súbormi paketov vo formáte pcap je možnosť generovať pri každom novom teste pakety s rôznymi parametrami, čím sa zvyšuje pravdepodobnosť odhalenia chyby pri opakovaní testov.

4.4.1 Vlastnosti generátora

Generátor generuje pakety viacerých protokolov. Konkrétne sú to protokoly TCP, UDP transportnej vrstvy a ICMP patriacich pod IP protokol sieťovej vrstvy referenčného OSI modelu. Okrem IP protokolu je tiež možné generovať pakety obsahujúce ARP protokol.

Najpodstatnejšia vlastnosť generátora je generovanie sieťových tokov. Pomocou parametrov sa dá konkrétne určiť, aký má byť počet generovaných tokov a koľko paketov má každý tok obsahovať. Toky sú tvorené základnou päticou tvorenou týmito parametrami paketov:

1. cieľová IP adresa,
2. zdrojová IP adresa,
3. cieľový port,
4. zdrojový port,
5. číslo protokolu transportnej vrstvy referenčného OSI modelu.

Požadované vlastnosti paketov, ktoré sa majú na tvorbe tokov podieľať sa určujú pomocou parametra, ktorého argumentom je 32 bitové číslo, kde každý bit reprezentuje boolovskú hodnotu prítomnosti alebo neprítomnosti práve jedného parametra vo vyššie zmienenom poradí začínajúc od najmenej významného bitu. Pre parametre paketov, ktoré majú tvoriť toky sa budú náhodne generovať ich rôzne hodnoty, pričom pri parametroch, ktoré nemajú byť pri tvorbe tokov významné budú ich hodnoty rovnaké.

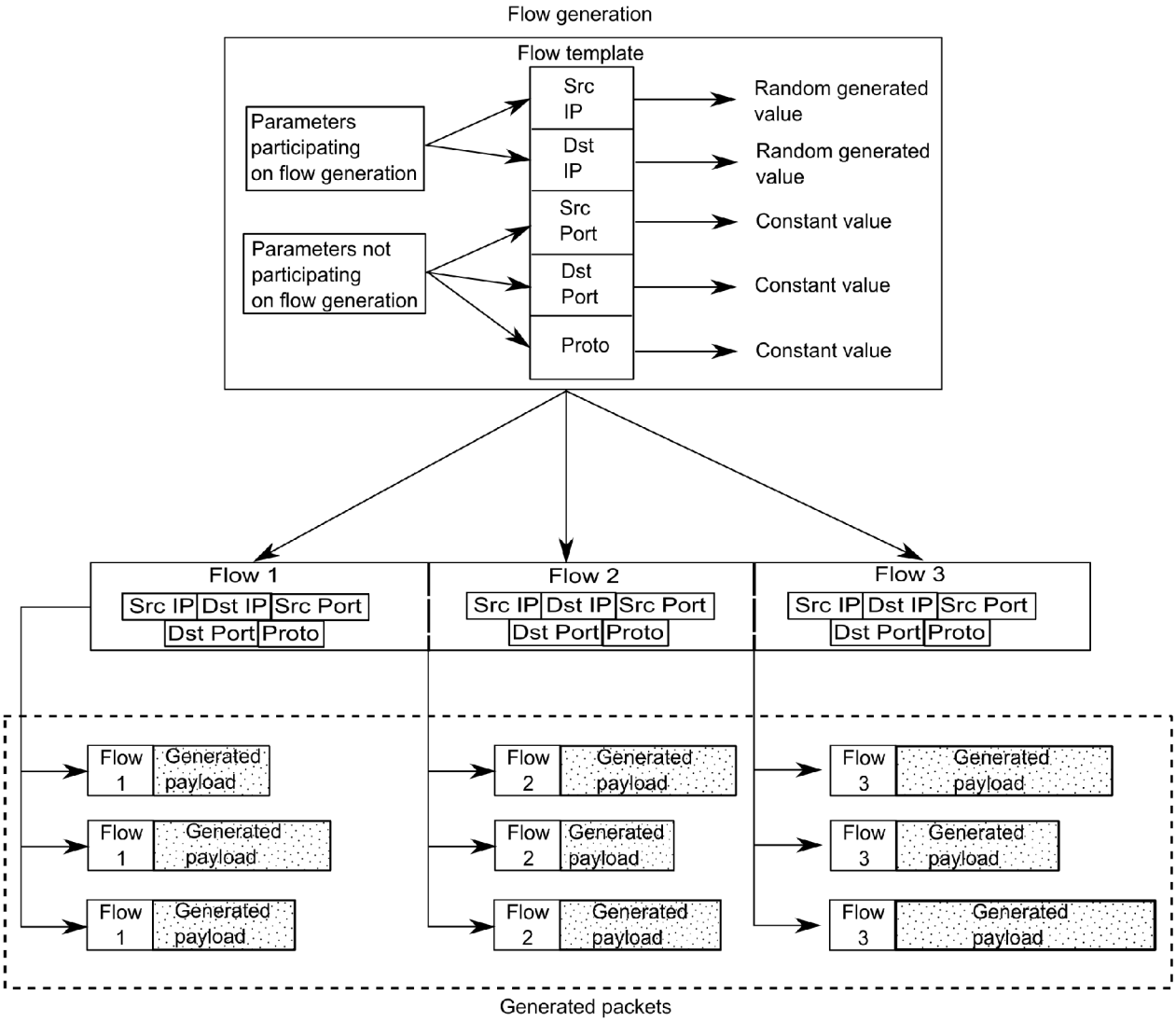
Sekvencia náhodných hodnôt použitých generátorom je určená semienkom, ktoré je implicitne vytvárané zo systémového času. Explicitne môže byť nastavené pomocou parametra pri spustení generátora.

Keďže je generácia hodnôt úplne náhodná, môže nastať situácia, keď budú dva rôzne toky tvorené rovnakou hodnotou. Aby sa tak nestalo, na konci generovania všetkých tokov sa vznik takýchto duplicít overuje, v prípade jej detekcie sa tieto duplicitné hodnoty nahradzujú novými. Tak isto sú ošetrené situácie, keď je požadovaný väčší počet tokov, ako je počet všetkých možných kombinácií parametrov, z ktorých majú byť toky tvorené. Ide napríklad o situáciu keď sa majú toky skladať z hodnôt zdrojového a cieľového portu, kde je zrejmé, že väčší počet tokov ako 2^{32} nemôže vzniknúť.

Generátor funguje tak, že sa najprv vygenerujú toky, ktoré sa následne duplikujú do počtu požadovaných paketov (viz Obrázok 4.3). Celkový počet paketov musí byť deliteľný počtom vytvorených tokov, v opačnom prípade sa pre jednotlivé toky vygeneruje rôzne množstvo paketov. Tak isto sa môže stať, že nebude sedieť celkový počet vygenerovaných paketov s tým, ktorý bol požadovaný parametrom pri spustení. Preto je ošetrené, aby takáto situácia nenastala.

Po úspešnom vygenerovaní všetkých paketov sa podľa potreby pred alebo za ne môžu dodatočne doplniť ďalšie toky slúžiace hlavne na prednaplnenie alebo vyprázdnenie flow cache (pamäť obsahujúca záznamy tokov). Nakoniec sa všetky vygenerované pakety uložia do pcap súboru a pomocou nástroja `tcpreplay` sa čo najrýchlejšie zašlú na požadovaný sieťový interface. Ak nástroj `tcpreplay` nie je k dispozícii, na odoslanie paketov sa použije vnútorná funkcia nástroja `Scapy`.

Generátor tak isto poskytuje možnosť uloženia informácií o vygenerovaných tokoch vo formáte, zhodným s formátom softvérového nástroja `sw_obuf_ctl` používaného na vyčítavanie tokov zo sondy.



Obrázok 4.3: Generácia paketov

Ďalšou funkciou generátora je možnosť určiť veľkosti generovaných paketov. Implicitne sa pakety generujú v náhodných veľkostiach spadajúcich do rozsahu `64B-1520B`. Rozsah veľkosti paketov sa nastavuje definovaním horného a dolného limitu v bajtoch, ktorý nesmie prekročiť vyššie zmienený implicitne nastavený rozsah.

4.5 Váha jednotlivých testov

Keďže sa celý autotest sondy skladá z viacerých testov rozdelených do hlavných troch kategórii (viz Kapitola 4.6), je potrebné určiť ich dôležitosť. Pri neúspešnosti testu treba rozhodnúť, ktoré ďalšie testy môžu byť touto neúspešnosťou ovplyvnené a či má zmysel ďalej pokračovať v testovaní (viz Kapitola 4.5.3).

Testy sa z hľadiska váhy delia do dvoch hlavných kategórií a to na kritické a nekritické.

4.5.1 Nekritické testy

Prvú kategóriu tvoria testy, ktorých prípadná neúspešnosť nemusí nutne viesť ku koncu celého autotestu ako celku a väčšina zostávajúcich testov môže naďalej prebehnúť. Medzi nekritické testy patria:

- Prijatý počet všetkých paketov (viz Kapitola 4.6.2.2),
- Test zahadzovania dlhých a krátkych paketov (viz Kapitola 4.6.2.2),
- Test vzorkovania (viz Kapitola 4.6.2.2),
- Test rýchlosti používanej sieťovej karty a minimálnych timeoutov (viz Kapitola 4.6.2.3),
- Test aktívneho timeoutu (viz Kapitola 4.6.2.3),
- Test neaktívneho timeoutu (viz Kapitola 4.6.2.3),
- Test kombinácie aktívneho s neaktívnym timeoutom (viz Kapitola 4.6.2.3),
- Test vzorkovania typu sample and hold (viz Kapitola 4.6.2.4).

4.5.2 Kritické testy

Druhá kategória je kategória kritických testov. Tieto testy overujú kľúčové prvky sondy, na ktorých sa zakladajú ostatné testy. V prípade, že neuspeje test spadajúci do tejto kategórie, je nutné ukončiť celý autotest sondy. Medzi kritické testy patria:

- Testovanie softvérovej častí (viz Kapitola 4.6.1),
- Overenie funkčnosti COMBO karty (viz Kapitola 4.6.2.1),
- Funkčnosť spojenia COMBO karty so sieťovou kartou (viz Kapitola 4.6.2.2),
- Test správnej klasifikácie paketov do tokov (viz Kapitola 4.6.3).

4.5.3 Závislosti testov

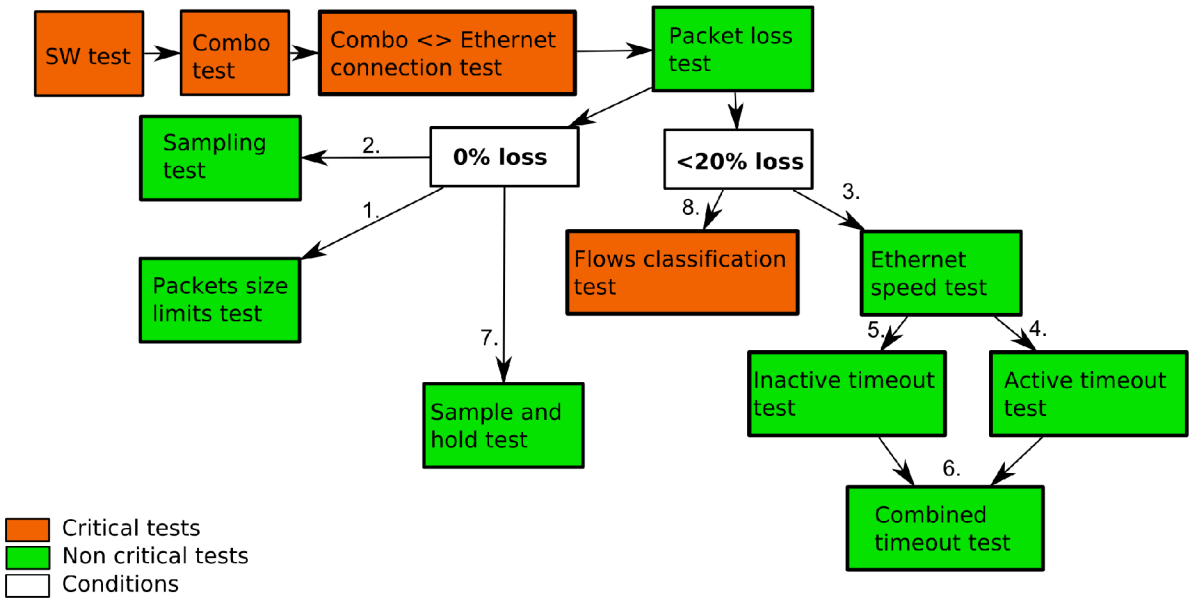
V nasledujúcej kapitole je detailnejšie rozobraté, prečo boli jednotlivé testy zaradené do tej ktorej váhovej kategórie (viz Obrázok 4.4).

Testy ako testovanie softvérových častí, test COMBO karty a jej spojenia so sieťovou kartou, sú testy, ktorých úspešnosť je nevyhnutná pre ďalšie pokračovanie v testovaní. Pretože je zrejmé, že bez existencií softvérových nástrojov a knižníc ako aj bez úspešného nahrania designu do karty nie je možné uskutočniť žiadne nasledujúce testy. To isté platí aj v prípade, nefunkčnosti spojenia sieťovej karty s COMBO kartou, keďže to znemožňuje zasielanie paketov do sondy, na ktorých je celé testovanie postavené. Napriek tomu, že na úspešnosti klasifikácie paketov do tokov niektoré testy nezávisia, bol aj tento test zaradený medzi kritické testy, hlavne z toho dôvodu, že je to test kľúčovej vlastnosti sondy.

Medzi prvý nekritický test môžeme zaradiť test počtu prijatých paketov. Tento test nemusí byť nutne úspešný. Existuje povolená tolerancia straty odoslaných paketov rovnajúca sa dvadsiatim percentám. Pri jej dodržaní je ešte stále možné vykonať niektoré testy. Sú to testy, ktorých výsledky nezávisia na presnom počte prijatých paketov, pretože sa testuje vlastnosť, ktorá ich istým spôsobom agreguje. Medzi tieto testy patrí test rýchlosti sieťovej karty, na ktorého splnenie postačuje prekročenie minimálnej hranice počtu paketov. Ďalej sú to testy timeoutov, ktoré sa zameriavajú na počet vytvorených tokov a nie na počet paketov, ktorými sú toky tvorené. Podobné zohľadnenie v tomto prípade existuje aj pri teste samotnej klasifikácie paketov do tokov. Na druhej strane pre testovanie vzorkovania a veľkostí paketov musí byť strata paketov nulová, keďže ich úspešnosť sa

zakladá na presných počtoch prijatých paketov, kde už aj ich malá strata môže výrazne negatívne ovplyvniť výsledky.

Test rýchlosti sieťovej karty nezaraďujeme medzi kritický, pretože je len orientačný a nasledujúce testy timeoutov môžu byť úspešné aj pri jeho zlyhaní. Samotné zlyhanie testu timeoutov a testu zahadzovania dlhých a krátkych paketov tak isto nepatrí medzi kritické, keďže v ďalších testoch sú v tomto prípade použité posledné úspešne otestované hodnoty. Ďalšie nekritické testovanie je vzorkovanie IBUF jednotky a vzorkovanie typu sample and hold, ktoré nemajú výraznejší vplyv na ostatné vlastnosti sondy. Pre uskutočnenie týchto testov je ale potrebná nulová strata paketov.



Obrázok 4.4: Závislosti testov

4.6 Popis jednotlivých testov

Celý test je zastrešený shell skriptom, z ktorého sa volajú jednotlivé podtesty. V prípade neúspešného testu môže nastať viacero situácií závislých od nastavených parametrov pri spustení skriptu alebo od typu testu (viz Kapitola 4.5).

Ak ide o nekritický test, informácia o jeho zlyhaní sa zaznamená a v testovaní sa pokračuje ďalej. V prípade, že zlyhá kritický test a pomocou parametra pri spustení nie je nastavené ignorovanie prvého zlyhaného testu, celkové testovanie sa ukončí. Tak isto je možné zvoliť niekoľkonásobné opakovanie zlyhaného testu (implicitne sú nastavené 3) a až po všetkých neúspešných pokusoch ho označiť za definitívne neúspešný. Táto možnosť slúži na odtienenie jednorázových zlyhaní spôsobených hlavne kvôli zriedkavému prekročeniu nastavených tolerancií (napríklad pri náhodnom vzorkovaní), kde je vysoká pravdepodobnosť, že pri opakovaní testu budú tieto limity splnené.

Kvôli subhraničným podmienkam (viz Kapitola 2.3.3.1) sú hodnoty veľkostí, počtu paketov a tokov zasielaných na sondu násobkami čísla 8, pretože sa dá predpokladať, že veľkostí používaných pamätí sú tiež násobkami tohoto čísla, hlavne kvôli použitej hardvérovej platforme pracujúcej v binárnej sústave.

Pri väčšine neúspešných testoch sa uloží informácia o nájdenej chybe a súbor paketov (vo formáte pcap), ktoré danú chybu spôsobili, aby ju bolo možné kedykoľvek zreprodukovať. Tieto informácie sú ukladané do vopred vytvoreného chybového adresára obsahujúcom podadresáre patriace jednotlivým zlyhaným testom. Aktuálny stav prebehnutých testov sa vypisuje na štandardný výstup, pričom sa zároveň zaznamenáva do súboru.

Jednotlivé testy sa delia na tri hlavné časti:

1. Prvá časť ma za úlohu overiť existenciu všetkých potrebných softvérových nástrojov, knižníc a súborov (napríklad konfiguračných súborov, súborov obsahujúcich firmvérovú časť sondy atď.) pre ovládanie a vyčítavanie informácii zo sondy. Po úspešnom overení softvérových časti, sa budú tieto nástroje používať počas nasledujúcich testov.

2. Druhá časť overí samotnú funkcionalitu hardvérovej časti sondy. Jednotlivé testy sú rozdelené podľa vlastností sondy, ktoré sa testujú.

3. Konečne tretia a zároveň posledná časť je zameraná na overenie správnej klasifikácie paketov do tokov.

4.6.1 Testovanie softvérovej časti

Úlohou testu je na základe konfiguračného súboru overiť, či existujú súbory obsahujúce potrebný firmvér, ktorý sa bude nahrávať do sondy. Ďalej binárne súbory, ktoré budú používané pri ovládaní sondy a testovaní a niektoré nimi používané knižnice. Konkrétne ide o linuxové nástroje z oblasti sietí ako `nfcapd`, `nfdump` a `tcpreplay`. Potom je to interpret jazyka `Python`, v ktorom je implementovaný generátor paketov a softvérové nástroje projektu Liberouter (`ibufctl`, `flowmonlkm` `flowmonctl`, `sw_obuf_ctl` a `flowmon`). Pri kontrole interpreta `Python` sa tiež overuje, či ide minimálne o verziu 2.5, ktorú vyžaduje rozšírenie `Scapy`, zároveň je ale v prípade verzie 3.x a vyššej užívateľ upozornený že môže dôjsť k nekompatibilite. Z knižníc sa zisťuje existencia súborov `libpcap`, `libcommlbr`, `libcombo` a `libcsflow`.

Vstup testu:

Cesta ku konfiguračnému súboru, obsahujúcemu inicializačné premenné.

Výstup testu:

V prípade že bol test neúspešný, do adresára s chybami testov sa uloží informácia o chýbajúcej komponente pripadne výpis skriptu, ktorý zlyhal. V testovaní sa ďalej nepokračuje.

4.6.2 Testovanie vlastností sondy

4.6.2.1 Overenie funkčnosti COMBO karty

Tento test vyvinutý softvérovým tímom Liberouter nie je priamou súčasťou autotestu sondy FlowMon. Jeho úlohou je otestovať hlavné komponenty COMBO karty, do ktorej sa bude nahrávať design sondy. Hlavnou časťou testu je otestovanie komunikácie s kartou, ďalej sú to testy SSRAM a SDRAM pamätí.

Ak sa nezistí žiadna chyba, na záver testu sa nahrá potrebný firmvér do COMBO karty, čo slúži ako posledná fáza testu, keďže tento skript vyžaduje existenciu väčšiny súborov, ktorých prítomnosť bola testovaná. Ak všetko prebehne bezchybne, test bude úspešný.

Vstup testu:

Spustí sa skript testujúci vyššie zmienené časti (cesta k nemu je definovaná parametrom pri spustení autotestu), následne sa spustí skript `flowmon`, ktorý nahrá design sondy do COMBO karty.

Výstup testu:

V prípade, že je test neúspešný, v adresári obsahujúcom informácie o chybách sa vytvori podadresár, kde sa uloží výpis používaných skriptov. V ďalšom testovaní sa nepokračuje.

4.6.2.2 Overenie funkčnosti IBUF jednotky

Testy overujúce správnu funkčnosť IBUF jednotky sa budú skladať z niekoľkých podtestov, pričom každý z nich bude na tvorbu paketov využívať program paket generátor.

Funkčnosť spojenia COMBO karty so sieťovou kartou

Vygeneruje sa 128 paketov (konkrétne parametre paketov budú generované náhodne), ktoré sa zašlú na zvolené sieťové rozhranie a bude sa sledovať, či prijaté pakety na IBUF jednotke vykazujú nenulovú hodnotu. Tento test má za úlohu overiť správne prepojenie sieťovej karty s COMBO kartou. Počet generovaných paketov bol určený na základe toho, že sa pri tomto teste má detekovať iba základná komunikácia sieťových jednotiek, na čo by mali postačovať desiatky zaslaných paketov. Tak isto generácia veľkého množstva paketov pomocou použitého paket generátora (rádovo tisíce) je časovo zbytočne zdĺhavá.

Test je neúspešný, ak nie sú na IBUF jednotke detekované žiadne pakety. V tomto prípade sa test snaží overiť, či sa užívateľ nepomýlil pri určovaní, ktoré Ethernetové rozhranie je spojené s ktorým rozhraním COMBO karty. Preto sa zašle rovnaký počet paketov postupne na všetky dostupné Ethernetové rozhrania a sleduje sa, či neboli tieto pakety prijaté IBUF jednotkou. Tento postup sa aplikuje pre všetky rozhrania COMBO karty. Ak sa pakety posielajú a detekujú na iných rozhraniach aké sú aktuálne nastavené, test sa ukončí s chybovým hlásením, ktoré navrhuje užívateľovi spustiť test s nastavením týchto rozhraní.

Vstup testu:

128 paketov, ktorých všetky parametre budú generované náhodne (veľkosť paketov nesmie prekročiť nastavený horný a dolný veľkostný limit na jednotke IBUF).

Výstup testu:

V prípade, že je test neúspešný, v ďalšom testovaní sa nepokračuje.

Prijatý počet všetkých paketov

Paket generátor bude generovať rôzne počty paketov a bude sa sledovať, či sú presne rovnaké počty zachytené aj na IBUF jednotke.

Vstup testu:

Test sa bude púšťať presne trikrát, konkrétne pre počty paketov 128, 1024 a 8192. Postupne zvyšujúce sa počty paketov majú za úlohu odhaliť ich prípadnú stratu, kde sa pri vyššom počte zvyšuje pravdepodobnosť stratovosti. Najvyšší počet je určený vzhľadom na výkonnostné možnosti softvérového generátora paketov. Všetky parametre paketov budú generované náhodne (veľkosť paketov nesmie prekročiť nastavený horný a dolný veľkostný limit na IBUF jednotke).

Výstup testu:

V prípade, že je test neúspešný sa zistí, koľko percent odoslaných paketov bolo stratených. Ak táto hodnota neprekročí hranicu 20% testovanie pokračuje ďalej s tým, že sa uskutočnia len niektoré testy (viz Kapitola 4.5.3), v opačnom prípade sa test ukončí. Hraničná hodnota 20% bola

určená experimentálne, keď je ešte testovanie niektorých nasledujúcich testov pri tejto strate dostatočne spoľahlivé.

Test zahadzovania dlhých a krátkych paketov

Tento test sa skladá z dvoch podtestov. V prvom podteste sa najprv v jednotlivých cykloch nastaví postupne limit IBUF jednotky tak, aby boli prijímané pakety presne v nasledujúcich veľkostiach *128B*, *512B*, *1024B* a *1520B*. Tieto čísla boli zvolené, tak aby rovnomerne pokryli nastaviteľné veľkostné rozmedzie paketov na IBUF jednotke. Z časového hľadiska trvania testu boli zvolené práve 4 hodnoty. V jednotlivých nastaveniach sa posielajú pakety v troch fázach, z ktorých má tretina celkového počtu odoslaných paketov veľkosť rovnajúcu sa aktuálnej veľkostnej medze, ďalšia tretina má veľkosť o *1B* menšiu a posledná tretina paketov má veľkosť o *1B* väčšiu ako je povolená veľkosť paketov.

V druhom podteste sa veľkostné limity postupne nastavia na nasledujúce veľkostné intervaly: *96B-384B*, *384B-512B*, *512B-768B*, *768B-896B*, *896B-1280B*, *1280B-1520B*. Tieto nastavenia majú námatkovo detekovať, či v prípade chyby v zahadzovaní paketov pri istých dĺžkach, existuje veľkostné pásmo, ktoré funguje korektne. Z časového hľadiska trvania testu bolo pásmo rozdelené na 6 častí približne rovnakej veľkosti. Následne sa parametrami *896-1152B*, *768-1280B*, *512-1408B*, *96-1520B* overí najväčšie možné nastaviteľné veľkostné rozmedzie prijímaných paketov. Tieto hodnoty postupne rozširujú možný veľkostný interval, smerom k hornému aj dolnému limitu, pričom sa začína v jeho strede. Rozširovanie nastáva v štyroch fázach, ktoré boli experimentálne určené vzhľadom na efektívnosť testu. Pakety sa generujú znova v troch dávkach, kde prvá tretina z celkového počtu odoslaných paketov bude patriť do nastaveného veľkostného limitu, ďalšia tretina sa bude skladať z paketov patriacich pod dolnú veľkostnú hranicu a posledná tretina bude obsahovať pakety s veľkosťami prekračujúcimi nastavenú hornú hranicu.

V obidvoch podtestoch sa sleduje, či bola prijatá presne jedna tretina paketov, ktorá spĺňala nastavené kritéria a ostatné dve tretiny paketov musia byť zahodené (ide o test zlyhaním).

Vstup testu:

14×(768) (prvý aj druhý podtest sa púšťa pre každé nastavenie povolenej veľkostnej medze paketov zvlášť teda dokopy 14krát, kde sa pri každom spustení posiela 256 paketov v každej fáze, čo je dokopy 768) paketov s vyššie zmienenými veľkosťami, ktorých všetky ostatné parametre sú generované náhodne. Počet paketov na jednu fázu bol určený vzhľadom na rýchlosť generácie v použitom paket generátore tak, aby bol ich počet pre korektné výsledky testov už postačujúci a pritom nebola doba testu generáciou paketov neúmerne predĺžená.

Výstup testu:

V prípade neúspešnosti testu, sa nastaví naposledy úspešne otestovaný limit veľkostí paketov a v teste sa ďalej pokračuje. Zároveň sa odteraz budú generovať len pakety, ktoré do tejto veľkosti spadajú. Ak nie je známe žiadne predošlé úspešné nastavenie limitov pre veľkosť paketov, nastaví sa najväčšie možné veľkostné rozmedzie, kde je najväčšia pravdepodobnosť, že sa pakety potrebné pre nasledujúce testy nebudú zahadzovať. Tak isto sa do adresára s informáciami o chybných testoch uložia pakety, ktoré boli pri tomto teste použité. Ďalej sa uloží informácia o nastavenom limite pre maximálnu a minimálnu veľkosť paketov a údaje o počte a veľkostí jednotlivých vygenerovaných paketov.

Test vzorkovania

Tento test sa tak isto skladá z dvoch podtestov. Počet posielaných paketov bol oproti predošlému testu zdvojnásobený práve kvôli náhodnému vzorkovaniu, kde je snaha sa čo najviac

priblížiť ideálnym pravdepodobnostným hodnotám, pri zachovaní relatívne stále krátkej doby generácie paketov.

V prvom podteste sa vygeneruje 512 paketov s náhodnými parametrami, pričom sa na jednotke IBUF nastavia hodnoty konštantného vzorkovania na požadované pomery a bude sa sledovať počet všetkých prijatých, navzorkovaných a zahodených paketov na jednotke (viz Obrázok 4.5). Keďže ide o konštantné vzorkovanie pomer nezahodených paketov k zahodeným sa musí presne rovnať nastavenému pomeru vzorkovania.

Druhý podtest pozostáva s rovnakých postupov ako prvý s tým rozdielom, že sa bude používať náhodné vzorkovanie. Pri tomto vzorkovaní sa pri overovaní pomeru prijatých paketov k zahodeným toleruje nepresnosť 5% zo všetkých odoslaných paketov (táto hodnota bola určená experimentálne vzhľadom na celkové množstvo odosielaných paketov).

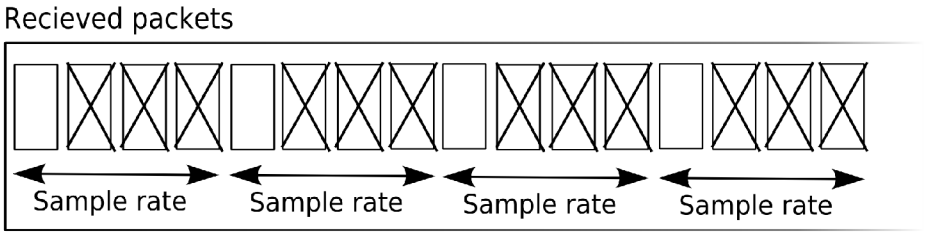
Z časového hľadiska testu boli určené práve 4 nastavenia pomerov vzorkovania. Vyššie zmienené podtesty sa teda opakujú 4krát, konkrétne pre tieto hodnoty vzorkovania: `1:2`, `1:4`, `1:16`, `1:32`. Posledná hodnota pomeru vzorkovania bola vybraná s ohľadom na náhodné vzorkovanie, keď test ešte dostatočne často produkuje výsledky spadajúce do použitej 5% tolerancie.

Vstup testu:

`4×2×512` paketov, ktorých všetky ostatné parametre budú generované náhodne (veľkosť paketov nesmie prekročiť úspešne otestovaný horný a dolný limit na IBUF jednotke).

Výstup testu:

Po ukončení testu sa nastaví hodnota vzorkovania na pomer `1:1`. V prípade neúspešnosti testu sa táto skutočnosť zaznamená, do adresára s chybami sa uložia pakety, ktoré boli použité. Tiež sa uložia informácie o počte odoslaných paketov a nastavenej hodnote vzorkovania. V prípade, že ide o test náhodného vzorkovania, uloží sa aj informácia o tolerancii použitej pri porovnávaní počtu prijatých paketov v percentách. V ďalšom testovaní sa následne pokračuje.



Obrázok 4.5: Vzorkovanie IBUF jednotky

4.6.2.3 Overenie timeoutov

Tento test je zameraný na overovanie funkčnosti aktívneho a neaktívneho timeoutu pri exportovaní tokov z hardvérovej do softvérovej časti.

Test rýchlosti používanej sieťovej karty a minimálnych timeoutov

Pred samotným testovaním timeoutov sa najprv spustí test, ktorého úlohou je zistiť, či používaná sieťová karta vysiela pakety do COMBO karty s dostatočnou rýchlosťou. Konkrétne je potrebné, aby karta dokázala odoslať 512 vygenerovaných paketov za menej ako 0.5 sekundy (počet paketov je určený z rovnakých dôvodov ako v predošlých testoch, tak isto je zohľadnená prípadná povolená stratovosť paketov) s najväčšou možnou veľkosťou, čo predstavuje dolnú hranicu všetkých možných vytvorených paketov za jednotku času danou sieťovou kartou. Čo zahŕňa dostatočnú rezervu pre ostatné testy, ktoré budú vykonávané v intervaloch v násobkoch sekúnd. Ďalšia súčasť testu je overenie funkčnosti nastavenia minimálneho aktívneho a neaktívneho timeoutu. Konkrétne sú to hodnoty aktívneho timeoutu 1 sekunda a neaktívneho 0 sekúnd. Toto nastavenie bude použité

v nasledujúcich testoch, v prípadoch keď je potrebné okamžite exportovať všetky aktuálne toky uložené na sonde. Test pozostáva z nastavenia vyššie zmienených hodnôt timeoutov a zaslania 512 paketov tvoriacich jeden tok. Tým, že je nastavená hodnota neaktívneho timeoutu na 0 sekúnd, má byť každý paket zaradený do samostatného toku, napriek tomu že majú všetky pakety rovnaké parametre. Úspešnosť testu závisí na počte detekovaných tokov sondou, ktorý sa musí rovnať počtu zaslaných paketov.

Vstup testu:

512 paketov, ktorých všetky ostatné parametre budú generované náhodne (veľkosť paketov nesmie prekročiť úspešne otestovaný horný a dolný limit na IBUF jednotke) v prípade testu rýchlosti a 512 paketov tvoriacich jeden tok v prípade testu minimálnych timeoutov.

Výstup testu:

Ak bude hociktorá časť testu neúspešná, v testovaní timeoutov sa bude napriek tomu pokračovať, ale bude vypísané upozornenie, že v nasledujúcich testoch timeoutov je možná neúspešnosť. Tak isto sa v prípade zlyhania testu rýchlosti uloží do adresára s chybnými testami informácia o detekovanom počte paketov za jednotku času a o minimálnom množstve, ktoré bolo pre úspešnosť testu potrebné.

Test aktívneho timeoutu

Neaktívny timeout sa nastaví na maximálnu možnú hodnotu, čo je 60 sekúnd, aby neovplyvňoval výsledky aktívneho timeoutu. Generátor generuje 4 toky, kde pre každý tok vygeneruje 4 pakety. Počet tokov a paketov patriacich do jedného dátového toku bol určený vzhľadom na možnosti generátora a sieťovej karty, keď je potrebné opakovane zasielať tieto toky dostatočne rýchlo, aby nebola prekročená hranica neaktívneho timeoutu. A zároveň aby sa pri ukončení zasielania tokov hodnoty ich časových známok čo najviac blížili ideálnym, tým že sa toky pomerne často obnovujú. Tieto pakety sa budú opakovane posielať do FlowMon sondy po dobu o sekundu menšiu ako je súčasná hodnota aktívneho timeoutu. Následne sa nastaví dočasne hodnota neaktívneho timeoutu na 0 sekúnd, čo exportuje aktuálne zachytené toky sondou. Priebežne sa skontroluje, či boli na sonde zachytené práve 4 toky (viz Obrázok 4.6). Potom sa rovnaké pakety budú odosielať na sondu o sekundu dlhšie ako je hodnota testovaného aktívneho timeoutu a skontroluje sa, či boli na sonde zachytené práve 2×4 toky (viz Obrázok 4.7). Dokopy by malo preto vzniknúť 12 tokov. Ak v ktoromkoľvek prípade pri priebežnom overovaní správneho počtu vytvorených tokov nastane chyba, test sa okamžite neukončí, ale chybu zaznamená a testovanie aktuálne nastaveného timeoutu dokončí, aby odhalil prípadne ďalšie chyby. Okrem overovania počtu vytvorených tokov sa súčasne overuje rozdiel ich časových známok. Tento rozdiel sa nesmie v sekundách líšiť od hodnoty práve testovaného aktívneho timeoutu o viac ako 15% (tolerancia bola určená experimentálne vzhľadom na možnosti používaných prostriedkov). Test je považovaný za neúspešný ak buď zlyhá detekcia počtu paketov alebo budú chybné hodnoty časových známok tokov.

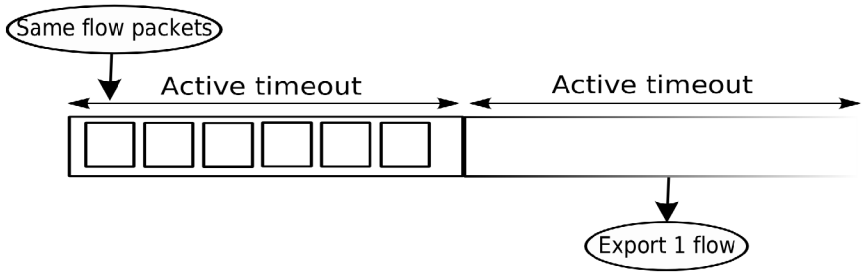
Tento test prebehne dokopy trikrát, konkrétne pre hodnoty aktívneho timeoutu 15, 30 a 60 sekúnd. Keďže, maximálna hodnota aktívneho timeoutu môže nadobúdať až 1200 sekúnd, z časového hľadiska trvania testu boli vybrané práve tri hodnoty, spadajúce do jednej minúty.

Vstup testu:

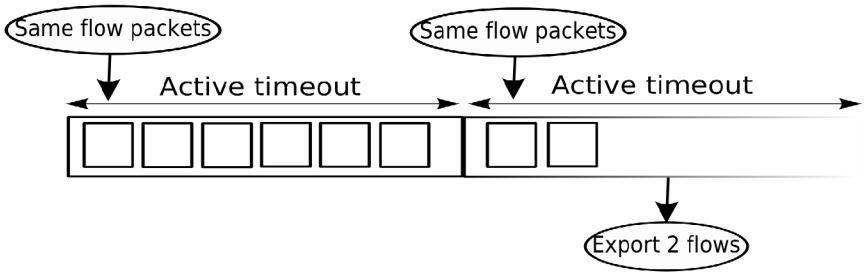
$3 \times$((počet paketov odoslaných za čas nastaveného aktívneho timeoutu − 1 sekunda) + (počet paketov odoslaných za čas nastaveného aktívneho timeoutu + 1 sekunda)) paketov pre rôzne nastavenia aktívneho timeoutu na vybrané hodnoty (konkrétne sú to 15, 30 a 60 sekúnd). Všetky vygenerované pakety patria do štyroch dátových tokov.

Výstup testu:

V prípade neúspešnosti testu sa nastaví aktívny aj neaktívny timeout na hodnoty naposledy úspešne otestované. V prípade, že nie je známa žiadna úspešne otestovaná hodnota, nastavia sa implicitné hodnoty 10 sekúnd pre aktívny a 3 sekundy pre neaktívny timeout. Tieto hodnoty boli určené experimentálne a v prípade funkčnosti týchto nastavení by mali postačovať pre nasledujúce testy. Ďalej sa do adresára s informáciami o chybných testoch uložia pakety, ktoré boli pri teste použité spolu s informáciami o hodnotách timeoutov, pri ktorých test zlyhal. V ďalších testoch sa následne pokračuje ďalej.



Obrázok 4.6: Neprekročený aktívny timeout



Obrázok 4.7: Prekročený aktívny timeout

Test neaktívneho timeoutu

Najprv sa aktívny timeout nastaví na maximálnu možnú hodnotu 1200 sekúnd, aby neovplyvňoval výsledky neaktívneho timeoutu. Potom generátor postupne generuje 4 toky, kde každý obsahuje 4 pakety (z rovnakých dôvodov ako to bolo pri aktívnom timeoute). Tieto pakety sa pošlú do FlowMon sondy, potom sa posielanie zastaví na dobu aktuálneho neaktívneho timeoutu zníženú o 2 sekundy a opätovne sa pošlú tie isté 4 toky na sondu, ktoré sa vďaka malému počtu posielajú extrémne krátko. Všetky zaslané pakety musia vytvoriť práve 4 dátové toky, pretože pauza v zasielaní neprekročila hodnotu neaktívneho timeoutu (viz Obrázok 4.8). Počet vytvorených tokov sa dočasným nastavením neaktívneho timeoutu na 0 sekúnd exportuje a priebežne skontroluje. Následne sa pošlú ďalšie 4 toky, počká sa o sekundu dlhšie ako je hodnota neaktívneho timeoutu a posielanie sa zopakuje (viz Obrázok 4.9). Takto zaslané pakety musia vytvoriť 2×4 dátových tokov.

Overovanie vzniknutých tokov prebieha rovnakým spôsobom ako pri teste aktívneho timeoutu.

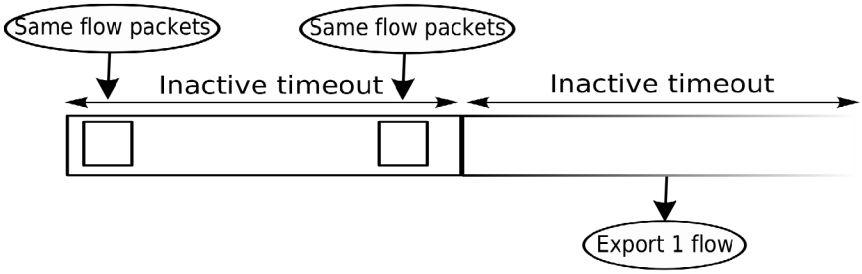
Tento test prebehne dokopy trikrát, konkrétne pre hodnoty neaktívneho timeoutu 10, 20 a 30 sekúnd (hodnoty sú určené na základe rovnakých kritérií ako pri aktívnom timeoute).

Vstup testu:

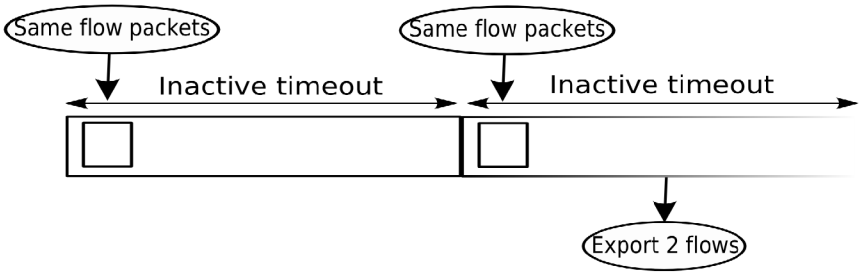
$3 \times 4 \times 4 \times 4$ paketov pre rôzne nastavenia aktívneho timeoutu na vybrané hodnoty (konkrétne je to 10, 20 a 30 sekúnd). Vygenerované pakety tvoria 4 dátové toky.

Výstup testu:

V prípade neúspešnosti testu sa uplatňuje rovnaký postup ako pri teste aktívneho timeoutu.



Obrázok 4.8: Neprekročený neaktívny timeout



Obrázok 4.9: Prekročený neaktívny timeout

Test kombinácie aktívneho s neaktívnym timeoutom

Tento test overí funkčnosť aktívneho a neaktívneho timeoutu súčasne. Je to kombinácia oboch predošlých postupov (testu aktívneho a neaktívneho timeoutu), tak isto sa skombinujú aj nastavené hodnoty aktívneho a neaktívneho timeoutu. Vo výsledku sa nakoniec spolu s časovými známkami overuje vznik 24 samostatných tokov, 12 vznikne pri teste aktívneho a 12 pri teste neaktívneho timeoutu.

4.6.2.4 Test vzorkovania typu sample and hold

Úlohou testu je overiť správne vzorkovanie paketov typu sample and hold. Test sa skladá z dvoch podtestov overujúcich dva typy sample and hold: konštantného a náhodného. Na začiatku každého testu sa najprv vyprázdni pamäť s tokmi od prípadných paketov z predošlých testov a aktivácia vzorkovania sa nastaví tak, aby sa začalo aplikovať už pri minimálnom zaplnení tejto pamäte (nastavenie treshold hodnoty na 0).

V prvom podteste testujúcom konštantný typ vzorkovania sa najprv vygeneruje potrebný počet paketov. Využíva sa tu možnosť generátora pridať pred tieto pakety dva doplnkové toky slúžiace na prednaplnenie pamäte, čo aktivuje vzorkovanie.

Vygenerované pakety sa pošlú do sondy, kde sa sleduje počet vygenerovaných tokov a počet paketov, ktoré dané toky tvoria, pričom sa neberú do úvahy prvé dva toky, ktoré slúžili na prednaplnenie pamäte. Vo všetkých ostatných tokoch sa musí počet paketov, ktorými sú tvorené rovnať rozdielu počtu všetkých zaslaných paketov patriacich do daného toku a hodnote menovateľa aktuálne nastaveného pomeru vzorkovania (viz Vzťah 4.2).

V druhom podteste sa aplikuje rovnaký postup ako v prvom. Pakety sa ale negenerujú odznova, ale využívajú sa tie z prvého podtestu, kvôli ušetreniu času potrebného na ich generovanie. Keďže ide o náhodné vzorkovanie, hodnota prijatého počtu paketov sa počíta z priemeru viacerých pokusov a pri overovaní správneho počtu paketov tvoriacich detekované toky je zavedená experimentálne určená tolerancia kolísania hodnôt na `5%`.

Vstup testu:

Každé spustenie testu zahŕňa zaslanie dokopy piatich tokov (na tvorbe jednotlivých tokov sa podieľajú všetky vlastností paketov), kde sa výsledky vyhodnocujú na základe posledných troch. Prvé dva toky, slúžia len na prednaplnenie pamäte sondy. Celý test skladajúci sa zo zmienených dvoch podtestov je spustený dokopy trikrát a to konkrétne pre tieto počty paketov na jeden tok:

1. `2880`,
2. `3840`,
3. `4800`.

Pre každú z vyššie uvedených počtov paketov sú nastavené tieto vzorkovacie hodnoty:

1. `1:32`,
2. `1:48`,
3. `1:128`.

Jednotlivé počty paketov boli určené z rovnakých dôvodov ako v predošlých testoch, kde sa hlavný ohľad berie na rýchlosť ich generácie a celkovú časovú náročnosť testu. Cieľ bol experimentálne nájsť čo najvyššiu hranicu počtu paketov, ktorá je z časového hľadiska generácie ešte únosná. K týmto hodnotám boli následne určené hodnoty jednotlivých pomerov vzorkovania. Kde bolo experimentálne zistené, že vybraná horná hranica `1:128` (kedy sa ideálne preskakuje až 128 paketov z toku) produkuje pri náhodnom vzorkovaní výsledky, ktoré ešte dostatočne často spadajú do povolenej tolerancie hodnôt.

Výstup testu:

Po ukončení testu sa nastaví hodnota vzorkovania sample and hold na `1:1`. V prípade neúspešnosti sa v testovaní pokračuje ďalej.

4.6.3 Test správnej klasifikácie paketov do tokov

Úlohou tohoto testu je overenie správnej klasifikácie jednotlivých paketov do výsledných dátových tokov na základe nasledujúcich parametroch: počet a súčet veľkostí paketov patriacich do toku, zdrojová a cieľová IP adresa, zdrojový a cieľový port a IP protokol.

Generovať sa budú postupne rôzne množstvá paketov, ktoré budú tvoriť rôzne počty dátových tokov. Jednotlivé dátové toky budú tvorené na základe rôznych kombinácii meniacich sa vyššie zmienených vlastností posielaných paketov. Rôzne zvolenými parametrami generátora sa bude určovať ich množstvo, počet dátových tokov, ktoré majú vzniknúť a parametre paketov, ktoré sa majú na tvorbe týchto tokov podieľať. Konkrétne hodnoty parametrov paketov budú generované náhodne. Test bude spustený 4krát s nasledujúcimi množstvami paketov:

- 512 paketov tvoriacich 128 tokov,
- 1024 paketov tvoriacich 256 tokov,
- 2048 paketov tvoriacich 256 tokov,
- 3840 paketov tvoriacich 384 tokov.

Tieto počty paketov a tokov boli opäť zvolené vzhľadom k výkonnostným možnostiam generátora paketov. Hodnoty sa postupne mierne zvyšujú, aby sa zvýšila pravdepodobnosť odhalenia chyby. Je nutné, aby jednotlivé toky obsahovali vždy viac ako jeden paket, práve kvôli povolenej tolerancii straty určitého množstva paketov po sieti. Pri každom spustení testu sa pred samotným zasielaním paketov najprv vyprázdni pamäť s tokmi (ide o exportovanie všetkých tokov nachádzajúcich sa v COMBO karte), aby nedošlo k zmiešaniu prípadných zostatkových paketov z predošlých testov s novo vygenerovanými.

Kombinácie vlastností paketov podieľajúcich sa na tvorbe tokov boli vybrané tak, aby pokrývali čo najväčší počet prípadov všetkých možných kombinácií. Konkrétne sa delia do nasledujúcich skupín:

- Prvá časť dátových tokov bude tvorená vždy zmenou len jednej vlastnosti paketu, dokopy sa bude teda posielať 5 dávok. V jednotlivých dávkach sa budú na klasifikácii jednotlivých dátových tokov postupne samostatne podieľať jednotlivé parametre paketov: cieľová a zdrojová IP adresa, cieľový a zdrojový port a IP protokol. V prípade, keď sa budú toky tvoriť len z IP protokolu, je vytvorená výnimka a toky sa budú vytvárať vždy práve 3. Keďže to je maximálna hodnota, ktorá môže kombináciou protokolov TCP, UDP a ICMP vzniknúť.
- Druhá časť dátových tokov bude tvorená práve dvomi zmenami parametrov vysielaných paketov, budú sa dokopy posielať 3 dávky. Na klasifikácii týchto tokov sa postupne budú podieľať nasledujúce dvojice parametrov: zdrojová a cieľová IP adresa, zdrojový a cieľový port a nakoniec zdrojová IP adresa kombinovaná s IP protokolom paketu.
- Tretia časť tokov bude tvorená 4 dávkami paketov, kde sa na klasifikácii dátových tokov budú podieľať tieto trojice parametrov :
  - zdrojová, cieľová IP, zdrojový port,
  - zdrojová, cieľová IP, cieľový port,
  - zdrojový, cieľový port, zdrojová IP,
  - zdrojový, cieľový port, cieľová IP.
- Štvrtá a posledná časť tokov bude tvorená jednou dávkou paketov, kde sa nakoniec na klasifikácii tokov budú podieľať všetky zmienené parametre.

Vstup testu:

$(512+1024+2048+3840)\times(5+3+4+1)$ paketov s parametrami tvoriacimi vyššie zmienené 4 skupiny tokov.

Výstup testu:

V prípade neúspešnosti tohoto testu sa uložia pakety, ktoré boli pri teste použité. Ďalej sa uloží informácia o tokoch, ktoré boli na sondu vyslané a ktoré boli sondou detekované a tiež informácia o tom v čom sa tieto toky líšia. Taktiež sa zaznamená údaj, ktoré parametre paketov sa na tvorbe tokov podieľali. V ďalšom testovaní sa už nepokračuje.

4.7 Použitie výsledkov autotestu

Autotest sondy FlowMon je riešený tak, aby bol pre neznalého užívateľa čo najjednoduchšie pochopiteľný a zvládnuteľný. Úlohou užívateľa je autotest spustiť s jemu vyhovujúcimi parametrami (ako počet opakovaní zlyhaného testu, ignorovanie prvého zlyhaného testu, určenie rozhraní COMBO a sieťovej karty, ktoré sú prepojené, názvy jednotlivých výstupných súborov a podobne). Po ukončení autotestu užívateľ pomocou výpisu výsledkov jednotlivých testov v prehľadnom formáte (viz Príloha B) zistí, či všetky testy prebehli úspešne. V prípade, že bola zistená chyba, sa v adresári, kde sa autotest nachádza, vytvorí nový podadresár obsahujúci detailnejšie informácie (vrátane paketov, ktoré chyby spôsobili) týkajúce sa jednotlivých zlyhaných testov. Tento adresár je vrátane súboru obsahujúcom výsledky testov potrebné príslušným archivovacím programom zabaliť a poslať vývojovému tímu projektu Liberouter.

Keďže sa jednotlivé podtesty zameriavajú na viacero podstatných častí sondy FlowMon ako z pohľadu jej hardvérovej štruktúry tak aj z hľadiska jej jednotlivých vlastností, autotestom vykonané testy pomôžu vývojárom zúžiť oblasť, ktorej sa nájdená chyba môže týkať, čo pomôže urýchliť detekciu jej príčiny a navrhnutie jej riešenia.

5 Záver

Úlohou tejto bakalárskej práce bolo navrhnúť a implementovať autotest základnej funkčnosti sondy FlowMon vytvorenej v rámci projektu Liberouter.

Pre návrh jednotlivých testov bolo potrebné preštudovať teóriu testovania, vlastnosti a architektúru sondy a nástroje z oblasti sietí poskytované operačným systémom Linux, ktoré mohli byť vzhľadom na cieľ autotestu použité. Tak isto bolo potrebné určiť nástroj, v ktorom by bola manipulácia s paketmi používanými pri testoch z hľadiska programátora čo najpraktickejšia. Na tento účel bol zvolený skriptovací jazyk `Python` s rozšírením `Scapy`.

Po navrhnutí testov, boli následne nezávisle implementované dve hlavné časti autotestu. Jednou boli skripty tvoriace samotný autotest a druhou bol generátor paketov používaný jednotlivými testami. Komunikácia a nastavenie sondy prebiehali pomocou softvérových nástrojov vyvinutých na projekte Liberouter. Konkrétne sa používali nástroje `flowmon`, `sw_obuf_ctl`, `flowmonctl`, `ibufctl` a `flowmonlkm`. Pri implementácii autotestu sa kládol dôraz na jednoduchosť formátu výstupu, aby bola pre užívateľa ľahko rozoznateľná prípadná úspešnosť alebo neúspešnosť testov.

Počas implementácie programu bolo nutné uskutočňovať množstvo pokusov, slúžiacich na určenie čo najideálnejších parametrov jednotlivých testov, ako napríklad počty odosielaných paketov, testované hodnoty vzorkovania, timeoutov a podobne. Cieľom bolo určiť tieto parametre tak, aby bol celkový autotest čo najefektívnejší a zároveň čo najmenej časovo náročný.

Autotest vytvorený v tejto bakalárskej práci slúži na otestovanie sondy verzie FlowMon, v budúcnosti sa môže rozšíriť jeho použitie aj na ďalšiu verziu sondy Flexible FlowMon, ktorá je v súčastnosti vyvíjaná na projekte Liberouter. Tak isto je možné rozšíriť vlastnosť autotestu na opakované pretestovanie sondy s využitím paketov z pôvodne zlyhaných testov.

Literatúra

[1] PATTON, Ron. *Testování softwaru*. Libor Pácl; David Krásenský. 1. vyd. [s.l.] : Computer Press, 2000. 307 s. ISBN 80-7226-636-5.

[2] *Software testing* [online]. 13:57, 5 December 2001 , 11 May 2009 [cit. 2009-05-11]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Software_testing>.

[3] *Equivalence partitioning* [online]. 16:41, 8 April 2005 , 8 May 2009 [cit. 2009-05-10]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Equivalence_partitioning>.

[4] *Integration testing* [online]. 07:37, 3 November 2003 , 7 May 2009 [cit. 2009-05-09]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Integration_testing>.

[5] *Protocol (computing)* [online]. 16:04, 2 May 2004 , 12 May 2009 [cit. 2009-05-12]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Network_protocol>.

[6] MATOUŠEK, Petr. *Sít'ové aplikace a správa sítí : Architektura sítí, adresování a konfigurace* [online]. [2008] [cit. 2009-03-01]. Dostupný z WWW: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ISA-IT/lectures/isa1-architektura.pdf>.

[7] *OSI model* [online]. 01:52, 17 December 2001 , 12 May 2009 [cit. 2009-05-12]. Dostupný z WWW: <http://en.wikipedia.org/wiki/OSI_model>.

[8] *Netflow* [online]. 21:28, 24 November 2004 , 17 April 2009 [cit. 2009-04-20]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Netflow>.

[9] *FlowMon Probe Handbook* [online]. Version 1.5.0. CESNET, z.s.p.o., 2006, 2007, 2008, 2009, 18 Feb 2009 [cit. 2009-03-25]. Dostupný z WWW: <http://www.liberouter.org/flowmon/handbook.html>.

[10] ČELEDA, Pavel, et al. *FlowMon Probe* [online]. CESNET z. s. p. o, 2006-2009 , 21.12.2006 [cit. 2009-03-15]. Dostupný z WWW: <http://www.cesnet.cz/doc/techzpravy/2006/flowmon-probe/>.

[11] *Liberouter* [online]. 2002-2006 [cit. 2009-03-04]. Dostupný z WWW: <http://www.liberouter.org/>.

[12] ŠPRINGL, Petr. *Návrh a implementace programového vybavení pro ovládání a konfiguraci sondy NetFlow*. [s.l.], 2006. 45 s. , popis ovládania programov, pamäťové médium - CD. Vysoké Učení Technické v Brně. Vedoucí bakalářské práce Ing. Tomaš Martínek.

[13] ŽÁDNÍK, Martin. *Design of Network Monitoring Adapter*. [s.l.], 2005. 31 s. Brno University of Technology. Vedoucí bakalářské práce Ing. Tomáš Pečenka.

[14] *Scapy v2.0.1 documentation* [online]. 2.0.1. Philippe Biondi and the Scapy community, Copyright 2008, 2009 , April 17, 2009 [cit. 2009-05-05]. Dostupný z WWW: <http://www.secdev.org/projects/scapy/doc/>.

[15] *Conformance testing* [online]. 16:42, 10 June 2002 , 10 April 2009 [cit. 2009-05-10]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Conformance_testing>.

Zoznam príloh

Príloha A - Popis ovládania programov

V tejto prílohe sa nachádza popis jednotlivých programov vytvorených v rámci tejto bakalárskej práce. Konkrétne ide o výpis ich parametrov spolu s ich detailnou charakteristikou.

A.1 generator

Program slúžiaci na generovanie paketov, pre potreby jednotlivých testov.

Parametre

-h, --help

zobrazenie obrazovky s výpisom parametrov a ich popisom

-p „output_packet_file“

určenie názvu súboru do ktorého sa majú ukladať vygenerované pakety

-i „output_flow_file“

určenie predpony názvu súborov obsahujúcich informácie o vygenerovaných tokoch

-s „seed“

definícia hodnoty semienka určujúceho náhodnosť generácie

-P „packet_count“

určenie počtu paketov, ktoré sa majú vygenerovať

-F „flow_count“

určenie počtu tokov, ktoré sa majú vygenerovať

-r „boolean“

boolovská hodnota určujúca, či sa majú generovať pakety patriace pod IP protokol

-I „interface“

určenie rozhrania použitej sieťovej karty

-S „min_size:max_size“

nastavenie veľkostného intervalu generovaných paketov

-d „definition“

určenie tých parametrov paketov, ktoré sa majú zúčastňovať na tvorbe tokov (0. bit = zdrojová IP, 1. bit = cieľová IP, 2. bit = zdrojový port, 3. bit = cieľový port, 4. bit = protokol)

-c „boolean“

boolovská hodnota dovoľujúca generovanie dodatočných 32 paketov na vyprázdnenie pamätí

-n „boolean“

boolovská hodnota určujúca, či sa majú generované pakety zároveň odoslať na vybrané rozhranie alebo len uložiť do súboru

-b „number“

určenie počtu paketov patriacich do dvoch tokov, ktoré majú byť pridané pred vygenerované pakety. Tieto pakety slúžia na prednaplnenie pamäte.

A.2 autotest

Program zastrešujúci všetky testy patriace do autotestu sondy Flowmon.

Parametre

-h, --help

zobrazenie obrazovky s výpisom parametrov a ich popisom

-e „error_dir“

názov adresára obsahujúceho podrobnejšie informácie o chybných testoch

-i „ethernet_iface“

ethernetové rozhranie použité pri generovaní paketov

-I „combo_iface“

číslo rozhrania COMBO karty, na ktoré budú pakety zasielané

-D „boolean“

povolenie zobrazenia ladiacich výpisov

-r „result_file“

názov súboru obsahujúceho výsledky jednotlivých testov

-f „boolean“

nastavenie ignorovania prvého zlyhaného testu

-l „number“

nastavenie počtu opakovania zlyhaného testu

-w „path“

nastavenie cesty ku skriptu spúšťajúceho hardvérový test

Príloha B - Ukážka štandardného výstupu autotestu sondy FlowMon

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
TEST RUNNING : SW_check
 SW_check PASSED
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
TEST RUNNING : HW_test
 HW_test FAILED
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
running test again

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
TEST RUNNING : HW_test
 HW_test PASSED
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
loading design …

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
TEST RUNNING : combo_iface_check
 combo_iface_check PASSED
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
TEST RUNNING : ibuf_test_correct_packets_count
generating 128 packets
generating 1024 packets
generating 8192 packets
 ibuf_test_correct_packets_count PASSED
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
TEST RUNNING : ibuf_test_correct_packets_size
generating packets with size 128
generating packets with size 512
generating packets with size 1024
generating packets with size 1520
 ibuf_test_correct_packets_size PASSED
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```