VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKACNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

OVLÁDÁNÍ GENERÁTORU SÍŤOVÉHO PROVOZU Z PROSTŘEDÍ OPNET MODELER

CONTROL OF NETWORK TRAFFIC GENERATOR FROM OPNET MODELER ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MILAN BARTL
AUTHOR

VEDOUCÍ PRÁCE doc. Ing. KAROL MOLNÁR, Ph.D.
SUPERVISOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNE

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Milan Bartl ***ID:*** 70204
Ročník: 2 ***Akademický rok:*** 2010/2011

NÁZEV TÉMATU:

Ovládání generátoru síťového provozu z prostředí OPNET Modeler

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s možnostmi pokročilé konfigurace generátoru syntetického síťového provozu IxChariot od firmy IXIA. Zaměřte se především na možnosti tvorby automatizovaných úkolů pomocí skriptovacího jazyka. Podrobně zdokumentujte postup vytváření automatizovaných úloh.
Prostudujte možnost spouštění externích aplikací a způsoby předávání parametrů těmto aplikacím ze simulačního prostředí OPNET Modeler. Vytvořte jednoduchý simulační model v prostředí OPNET Modeler, který umožňuje spustit během simulace automatizovaný úkol z prostředí IxChariot. Podrobně zdokumentujte postup vytváření simulačního modelu a konfiguraci příslušných rozhraní.

DOPORUČENÁ LITERATURA:

[1] IXIA, IxChariot User Guide, Release 6.50, 2008
[2] OPNET Technologies, OPNET Modeler Product Documentation Release 16.0, OPNET Technologies Inc., 2010

Termín zadání: 7.2.2011 ***Termín odevzdání:*** 26.5.2011
Vedoucí práce: doc. Ing. Karol Molnár, Ph.D.

prof. Ing. Kamil Vrba, CSc.
předseda oborové rady

ANOTACE

Tato práce popisuje rozhraní mezi síťovým simulátorem OPNET Modeler a generátorem síťového provozu IxChariot. Cílem je ovládání generátoru pomocí dat ze simulace uvnitř OPNET Modeleru, přesněji generování datového toku s patřičným nastaveným polem DSCP pro identifikaci kvality služeb. Cíle práce lze dosáhnout dvěma cestami: využitím IxChariot TCL API a IxChariot C API. Jsou popsány obě varianty a na závěr je provedena diskuze o možnostech a rozdílech těchto přístupů.

KLÍČOVÁ SLOVA

OPNET Modeler, IxChariot, ESYS, API, TCL, QoS, kosimulace

ABSTRACT

In this work an interface between network simulator OPNET Modeler and network operations generator IxChariot is described. The goal is to control the generator by data from an OPNET Modeler simulation. More accurately the generator is supposed to produce a data flow with desired settings of DSCP field, which is used to support quality of service mechanism. The goal can be achieved by two possible approaches: using IxChariot TCL API or IxChariot C API. Both of these approaches are described and conclusions are made at the end.

KEYWORDS

OPNET Modeler, IxChariot, ESYS, API, TCL, QoS, co-simulation

BARTL, M. *Ovládání generátoru síťového provozu z prostředí OPNET Modeler: diplomová práce*. Brno: FEKT VUT v Brně, 2011. 45 s. 10 s. příloh. Vedoucí práce doc. Ing. Karol Molnár, Ph.D.

Prohlášení

Prohlašuji, že svoji diplomovou práci na téma „Ovládání generátoru síťového provozu z prostředí OPNET Modeler“ jsem vypracoval samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 26. 5. 2011 .....

podpis autora

Seznam symbolů a zkratek

API	Application Programmable Interface – rozhraní pro komunikaci aplikace s vnějším prostředím
CSV	Comma-separated Values – datový formát, data jsou oddělena čárkami
DSCP	Differentiated Services Code Point – pole v IP záhlaví pro nastavení identifikátoru kvality služeb
ECN	Explicit Congestion Notification – pole v IP záhlaví pro signalizaci zahlcení sítě
EOF	End of File – znak určující konec souboru
ESYS	External System – rozhraní pro komunikaci simulace v OPNET Modeleru s vnějším prostředím
FTP	File Transfer Protocol – protokol pro přenos souborů
HTML	Hypertext Markup Language – značkovací jazyk pro vytváření webových stránek
HTTP	Hypertext Transfer Protocol – protokol určený pro výměnu hypertextových dokumentů (webových stránek)
IP	Internet Protokol – nespolehlivý nespojovaný síťový protokol
IPTV	Internet Protocol Television – televizní vysílání přes paketové sítě
OM	OPNET Modeler – síťový simulátor
PDF	Portable Document Format – formát souborů pro ukládání dokumentů
QoS	Quality of Service – zajištění dostatečných síťových prostředků pro správný běh síťové aplikace
SDK	Software Development Kit – sada nástrojů pro vývoj aplikací
SNMP	Simple Network Management Protocol – protokol pro dohled a správu sítě
TCL	Tool Command Language – vyšší skriptovací jazyk
TCP	Transmission Control Protocol – spolehlivý spojovaný transportní protokol
TXT	Běžný formát textových souborů
VoIP	Voice over IP – přenos hlasu přes paketové sítě

Obsah

Přílohy

Seznam obrázků

Kapitola 9

1. Úvod

Úkolem této práce je průzkum a realizace propojení síťového simulátoru OPNET Modeler a generátoru síťového provozu IxChariot za účelem propojení simulace s reálným síťovým prostředím.

Na základě dat extrahovaných zevnitř simulace v OPNETu je generován parametrizovaný datový tok. Parametry datového toku jsou nastavovány podle přijatých hodnot. Výstup práce bude využit pro výzkum a vývoj nových funkcí technologie DiffServ. Z tohoto důvodu je důraz kladen především na možnosti nastavení DSCP bitů (pro QoS příznaky) výsledného datového toku.

Cílem této práce je vytvoření middleware vrstvy, která by sloužila jako rozhraní mezi výše zmiňovanými aplikacemi. Vzhledem k charakteru API aplikace IxChariot je možný přístup k vytvoření této aplikace dvojí – s prostřednictvím IxChariot TCL API či C API.

Práce se zabývá vytvořením middleware vrstvy s použitím obou variant API a jejich následným porovnáním a zhodnocením. Výstupem práce jsou dvě sady nástrojů, které reprezentují propojující middleware aplikaci. Pomocí TCL skriptů je realizován middleware komunikující s IxChariot TCl API a komunikaci s C API zajišťují funkce vložené do dynamické knihovny, která slouží k extrakci dat z rozhraní OPNET Modeleru. Na závěru práce jsou diskutovány principy zpracování statistik reálné sítě naměřených aplikací IxChariot.

Ve struktuře tohoto dokumentu se na prvním místě nachází stručný popis obou propojovaných aplikací, síťového simulátoru OPNET Modeler a generátoru síťového provozu IxChariot. V dalších kapitolách je popsán postup vytvoření TCL skriptů pro komunikaci s rozhraním IxChariot TCL API a následně jejich propojení s OPNET Modelerem. Jako další je zpracován postup vytvoření konzolové aplikace v jazyce C, komunikující s IxChariot C API. Předposlední kapitola popisuje začlenění těchto funkcí do architektury kosimulace v OPNET Modeleru. Poslední kapitola pojednává o principu zpracování nashromážděných dat v grafickém uživatelském rozhraní IxChariotu a možnost jejich exportu do obecných datových formátů.

2. OPNET Modeler

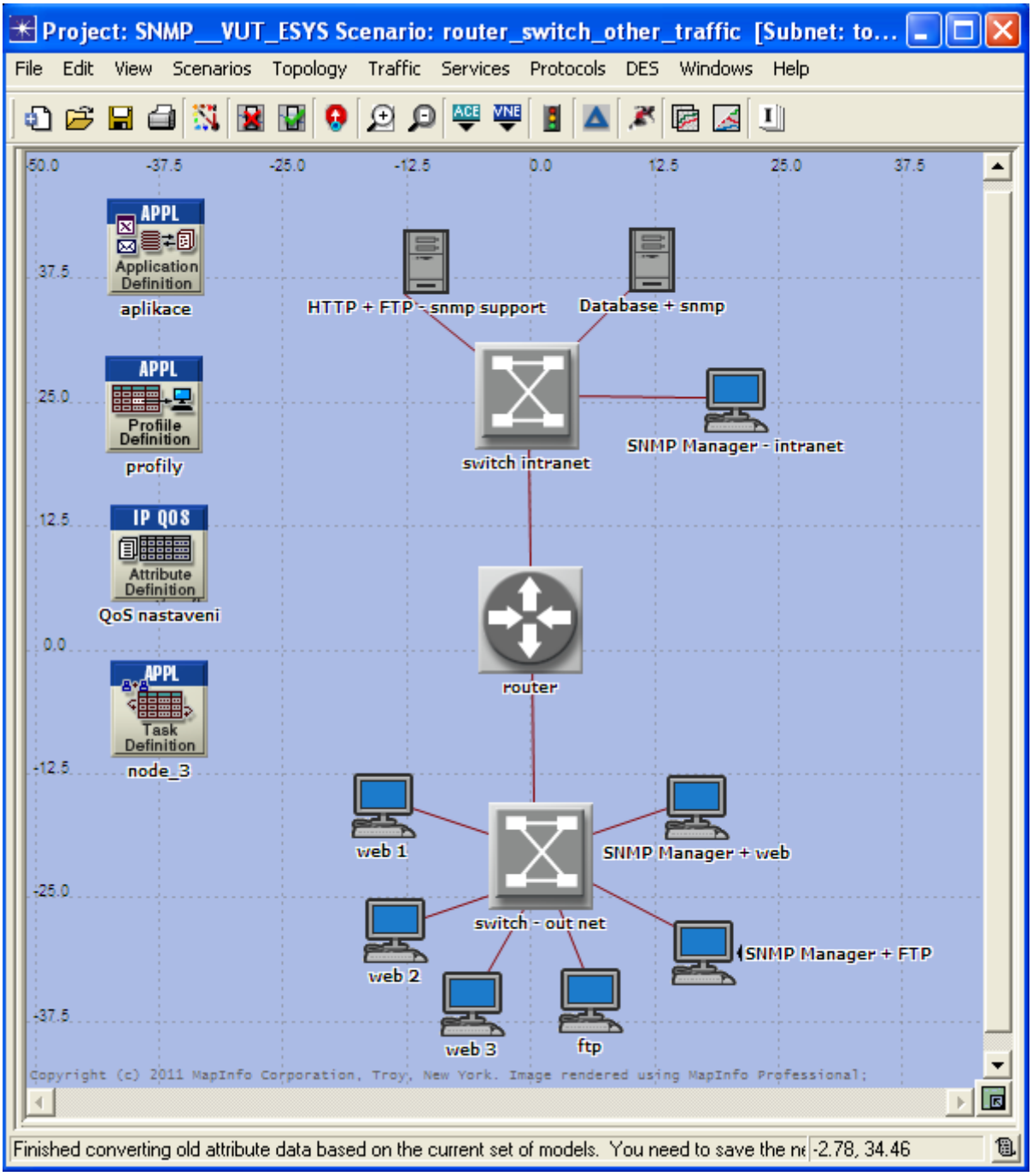
OPNET Modeler (OM) je světově nejrozšířenějším softwarovým nástrojem pro simulaci počítačových sítí všech rozsahů vytvořeným společností OPNET Technologies Inc. Na rozdíl od simulátoru NS-2, který se svojí rozšířeností umístil na druhé příčce, není OM freeware aplikací.

Jedná se o hybridní simulátor, založený na kombinaci analytického modelování a diskrétních událostí. Disponuje přehledným grafickým rozhraním s podporou tzv. „Drag and Drop“ modelování.

Je charakteristický svojí robustností a všestranností. Pomocí hierarchické struktury objektů má uživatel možnost nastavit síťovým prvkům velice rozsáhlou škálu parametrů a definovat chování simulovaných aplikací na čtyřech úrovních abstrakce. Hierarchie modelů je následovná:

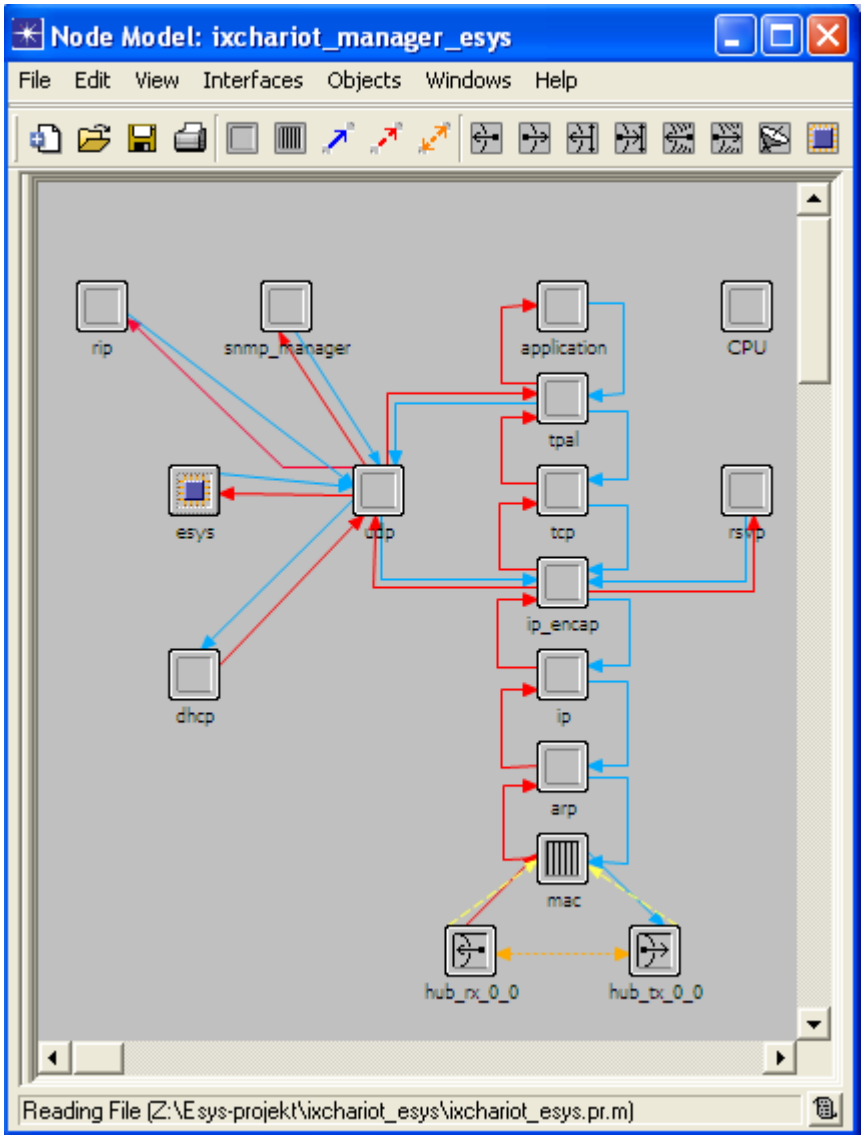
- model objektů (Object model),
- model uzlu (Node model),
- model procesu (Process model),
- zdrojový kód.

V modelu objektů uživatel umisťuje na mapu síťové prvky (uzly), vytváří mezi nimi spojení, přiřazuje prvkům síťové aplikace a definuje základní chování těchto aplikací. Pro tuto definici slouží speciální uzly *Application Config*, *Profile Config* a *Task Config* (viz obr. 2.1). Všem uzlům v modelu objektů lze nastavit atributy kliknutím pravého tlačítka myši a zvolením položky *Edit attributes* z menu.



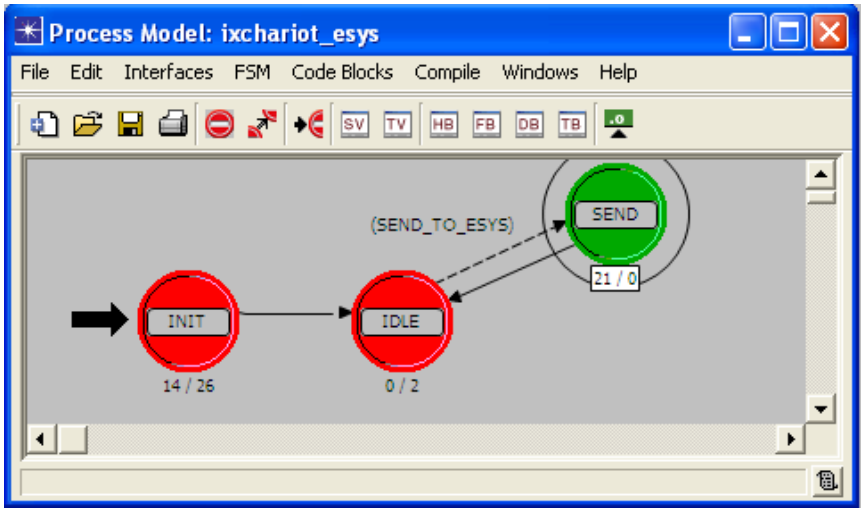
Obr. 2.1 Model objektů

Dvojklikem na libovolný uzel se otevře nové okno zobrazující vnitřní strukturu uzlu (model uzlu). Nachází se zde jednotlivé procesy, které reprezentují funkce protokolů různých vrstev modelu ISO/OSI, řídící procesy atd. Procesy jsou propojeny vazbami, pomocí kterých si mohou posílat informace. Ukázka modelu uzlu se nachází na obr. 2.2.



Obr. 2.2 Model uzlu

Po otevření objektu procesu dvojklikem je zobrazeno okno jako na obr. 2.3, obsahující model procesu. Každý proces (protokol, aplikace) prochází během simulace určitými stavy. Tyto stavy jsou zde vyobrazeny včetně přechodů mezi nimi. Stavy procesu se dělí na dvě kategorie: vynucené a nevynucené. Nevynucené stavy jsou označeny červenou barvou a proces v nich setrvává tak dlouho, dokud v něm není vyvoláno přerušení. Zelená barva značí vynucené stavy, ze kterých proces po vykonání všech instrukcí přechází automaticky. Přechody mezi stavy jsou značeny šipkami, začínajícími a končícími v jednom ze stavů. Přechody lze definovat bez podmínek, a v takovém případě jsou vykonány při příchodu prvního přerušení. Dalším typem jsou podmíněné přechody, které mají definovány podmínky, které musí při příchodu přerušení nastat, aby byly vyvolány. Tyto podmínky se definují pomocí makra *#define* v hlavičkovém bloku procesu (*Header Block*). Pokud se vyskytne případ, kdy je v procesu vyvoláno přerušení a existuje více než jeden nebo žádný definovaný přechod, vyvolá proces tzv. „zotavitelnou chybu“ (*Recoverable Error*). Tato chyba varuje uživatele o možné chybě či nedefinovaném chování, nicméně nepřeruší běh simulace.

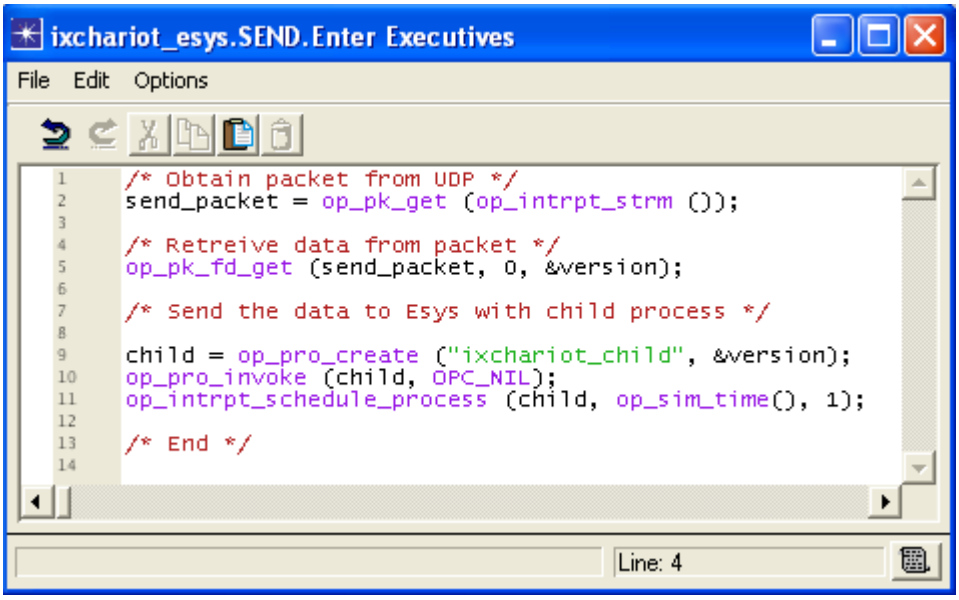


Obr. 2.3 Model procesu

Přestože OM není aplikace s volně dostupným zdrojovým kódem (tzv. open source), parametry modelů lze měnit až na úrovni zdrojového kódu (obr. 2.4). Tento kód je rozdělen do bloků jednotlivých stavů a lze jej vyvolat dvojklikem na požadovaný stav v modelu procesů. Každý stav je definován dvěma bloky kódu: vstupní a výstupní. Vstupní blok je proveden při vstupu procesu do daného stavu, výstupní před přechodem do stavu dalšího. Každý proces má přiděleno několik dalších bloků, které jsou sdílené mezi všemi stavy v procesu. Jsou to:

- hlavičkový blok – deklarace struktur, makra *#define* a *#include*,
- blok stavových proměnných – proměnné, které jsou uchovávány i při neaktivitě procesu,
- blok dočasných proměnných – jsou uchovávány pouze během aktivity procesu,
- funkční blok – pro deklaraci vlastních funkcí,
- ukončovací blok – kód, který bude proveden při ukončení simulace (např. dealokace určitých globálních prostředků).

Kód podporuje syntaxi jazyka C (nikoli však C++) s instrukční sadou rozšířenou o funkce vytvořené vývojáři OM. OM disponuje vlastními funkcemi pro alokaci/dealokaci paměti, ukončení simulace apod. Seznam a popis funkcí instrukční sady se nachází v dokumentaci [4].



Obr. 2.4 Zdrojový kód

Z globálního hlediska je OM robustní simulační nástroj, jehož mohutnost je zároveň i jeho slabinou. Dokumentace k produktu zabírá přes pět set megabytů paměti a místy je nalezení informací velmi složité a nejednoznačné.

Podrobnější popis tvorby simulací v OM lze nalézt v [3] a [6].

3. IxChariot

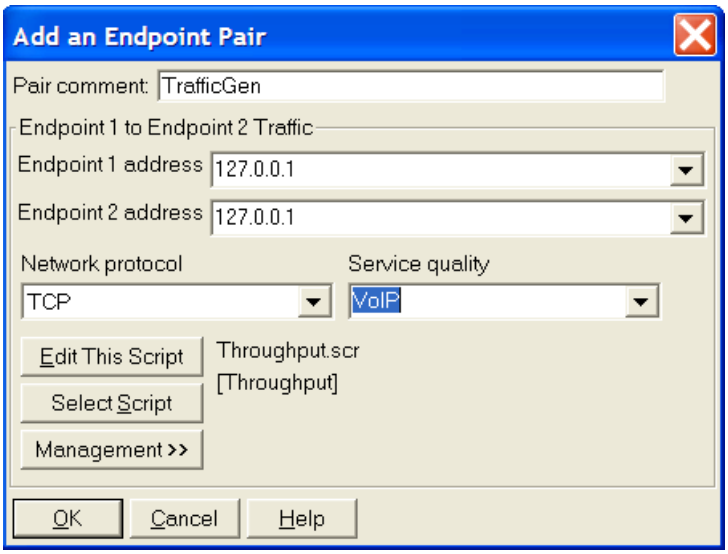
IxChariot je testovací software pro simulaci chování reálných aplikací. Umožňuje predikci chování sítě, síťových zařízení i simulované aplikace v reálných podmínkách. [1]

Výše zmíněného cíle dosahuje generováním konfigurovatelného datového toku, který posílá do fyzické sítě. Ke konfiguraci a ovládání datového toku slouží IxChariot Console, což je řídící aplikace disponující grafickým rozhraním, umožňující vytváření a spouštění simulací, vyhodnocování jejich výsledků, uživatelskou konfiguraci datových toků atd.

Samotné generování datového toku je úkolem aplikací zvaných Performance Endpoint (koncový bod). Tyto koncové body na základě informací přijatých od IxChariot Console generují datový tok a posílají pakety do sítě, sbírají statistiky a ty následně odesílají zpět řídící aplikaci. Výhodou této architektury je možnost umístění endpointů na řadu různých koncových stanic v síti. Pro vytvoření komunikace jsou vždy třeba minimálně dva endpointy kontrolované řídící aplikací.

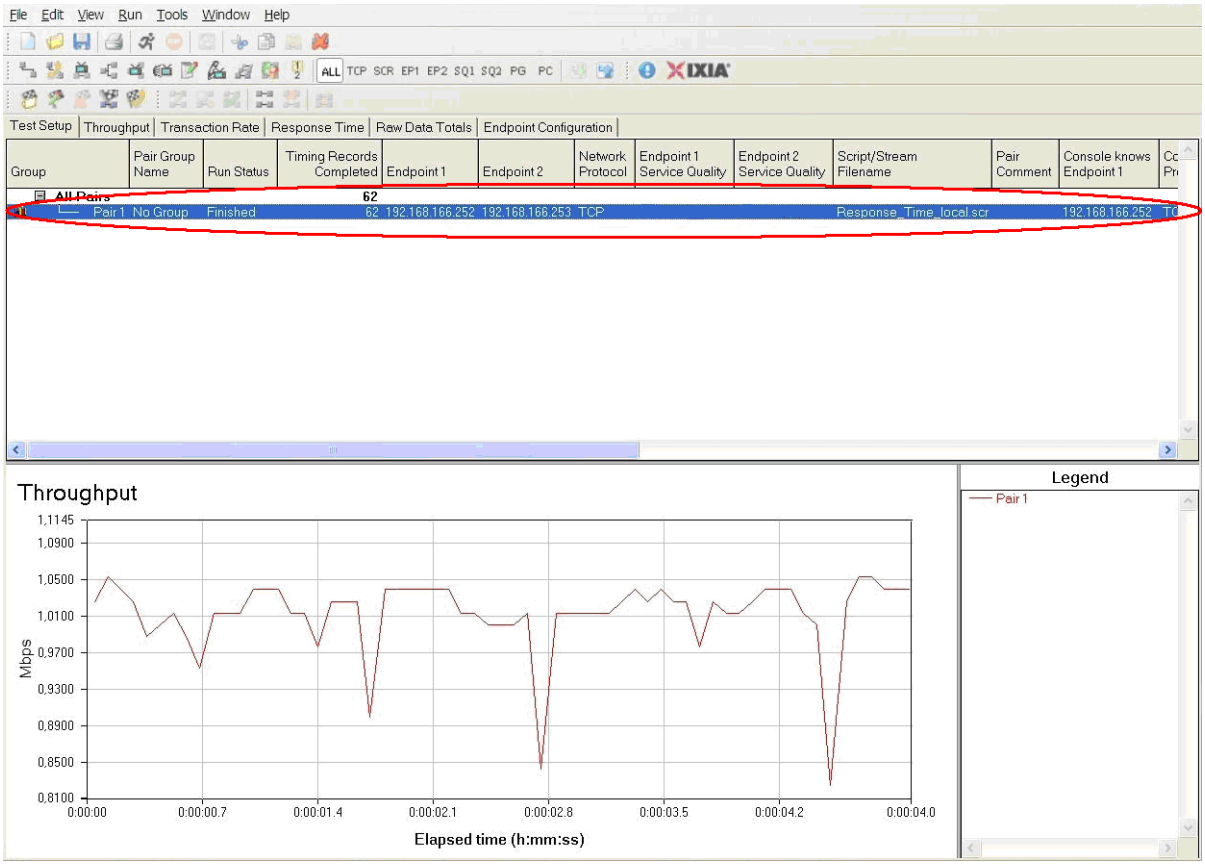
3.1. Komunikační páry

Generovaný datový tok vždy proudí mezi dvěma konkrétními endpointy. Dvěma takto „spojeným“ endpointům se říká komunikační pár. Jeden endpoint může náležet k více komunikačním párům. Stiskem tlačítka  v grafickém rozhraní se zobrazí základní konfigurace komunikačního páru (obr. 3.1).



Obr. 3.1 Základní konfigurace komunikačního páru

Nastavují se IP adresy obou endpointů, použitý transportní protokol, kvalita služeb (QoS) a komunikační skript, který bude popsán dále. Po jeho vytvoření je komunikační pár včetně svého nastavení zobrazen v prostřední části okna (viz obr. 3.2).



Obr. 3.2 Hlavní okno grafického rozhraní IxChariotu s vytvořeným komunikačním párem

Detailnější konfigurace datového toku (než nastavení transportního protokolu a QoS) je prováděna pomocí tzv. komunikačního skriptu. Jedná se o speciální skript popisující požadované chování aplikací endpointů. Lze definovat např. velikost paketů, interval mezi jejich odesíláním, prodlevu mezi začátkem simulace a startem vysílání atp. IxChariot obsahuje několik skriptů představujících základní druhy provozu jako FTP přenos dat, HTTP spojení, IPTV a další. Tyto skripty jsou modifikovatelné dle uživatelských potřeb. Skript se vždy skládá ze dvou částí; každá z nich ovládá jeden endpoint v páru (obr. 3.3).

```
1  SLEEP
2    time = initial_delay (0)
3  CONNECT_INITIATE
4    port = source_port (AUTO)
5    send_buffer = DEFAULT
6    receive_buffer = DEFAULT
7  LOOP
8    count = number_of_timing_records (100)
9    START_TIMER
10   LOOP
11     count = transactions_per_record (1)
12     SEND
13       size = file_size (100000)
14       buffer = send_buffer_size (DEFAULT)
15       type = send_datatype (NOCOMPRESS)
16       rate = send_data_rate (UNLIMITED)
17     CONFIRM_REQUEST
18     INCREMENT_TRANSACTION
19   END_LOOP
20   END_TIMER
21   SLEEP
22     time = transaction_delay (0)
23 END_LOOP
24 DISCONNECT
25   type = close_type (Reset)
```

```
CONNECT_ACCEPT
  port = destination_port (AUTO)
  send_buffer = DEFAULT
  receive_buffer = DEFAULT
LOOP
  count = number_of_timing_records (100)

  LOOP
    count = transactions_per_record (1)
    RECEIVE
      size = file_size (100000)
      buffer = receive_buffer_size (DEFAULT)


    CONFIRM_ACKNOWLEDGE

  END_LOOP


END_LOOP
DISCONNECT
  type = close_type (Reset)
```

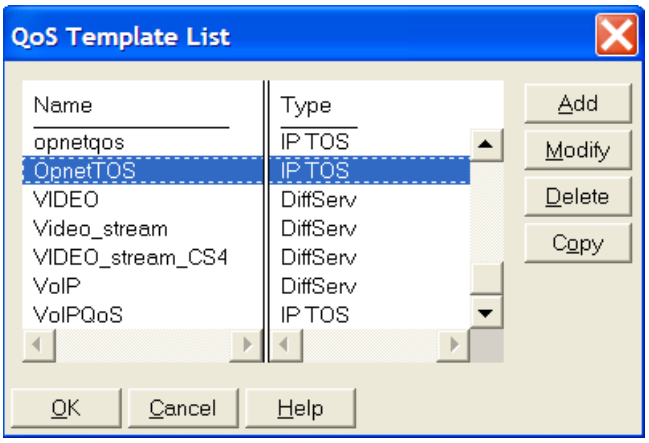
Obr. 3.3 Příklad komplementárních skriptů na dvou koncových bodech

Skripty v IxChariotu jsou dvojího typu: streaming a non-streaming. Skripty kategorie streaming emulují provoz síťových aplikací (streamování videa, zvuku). Non-streaming kategorie slouží pro simulaci aplikací komunikujících obousměrně (např. aplikace pro práci s databázemi, produkující sérii požadavků a odpovědí). [1]

Pár koncových bodů se může také nacházet na jednom stroji (viz obr 3.1 – nastavení IP adres na 127.0.0.1). V tomto případě proudí datový tok pouze zařízením síťové karty zpět do aplikace druhého endpointu.

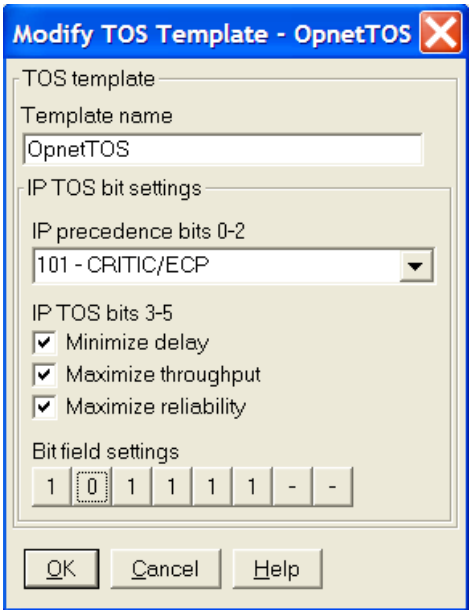
3.2. Šabony QoS

IxChariot podporuje nastavování QoS pomocí tzv. šablon. Každá šablona má přidělen unikátní název a obsahuje informace o bitovém nastavení. Šablony jsou ukládány do textového souboru v adresáři s instalací IxChariotu. Jeden řádek souboru odpovídá jedné šabloně. Při nastavování QoS je soubor se šablonami prohledán a uživateli jsou nabídnuty dostupné šablony (viz obr. 3.4).



Obr. 3.4 Okno seznamu QoS šablon

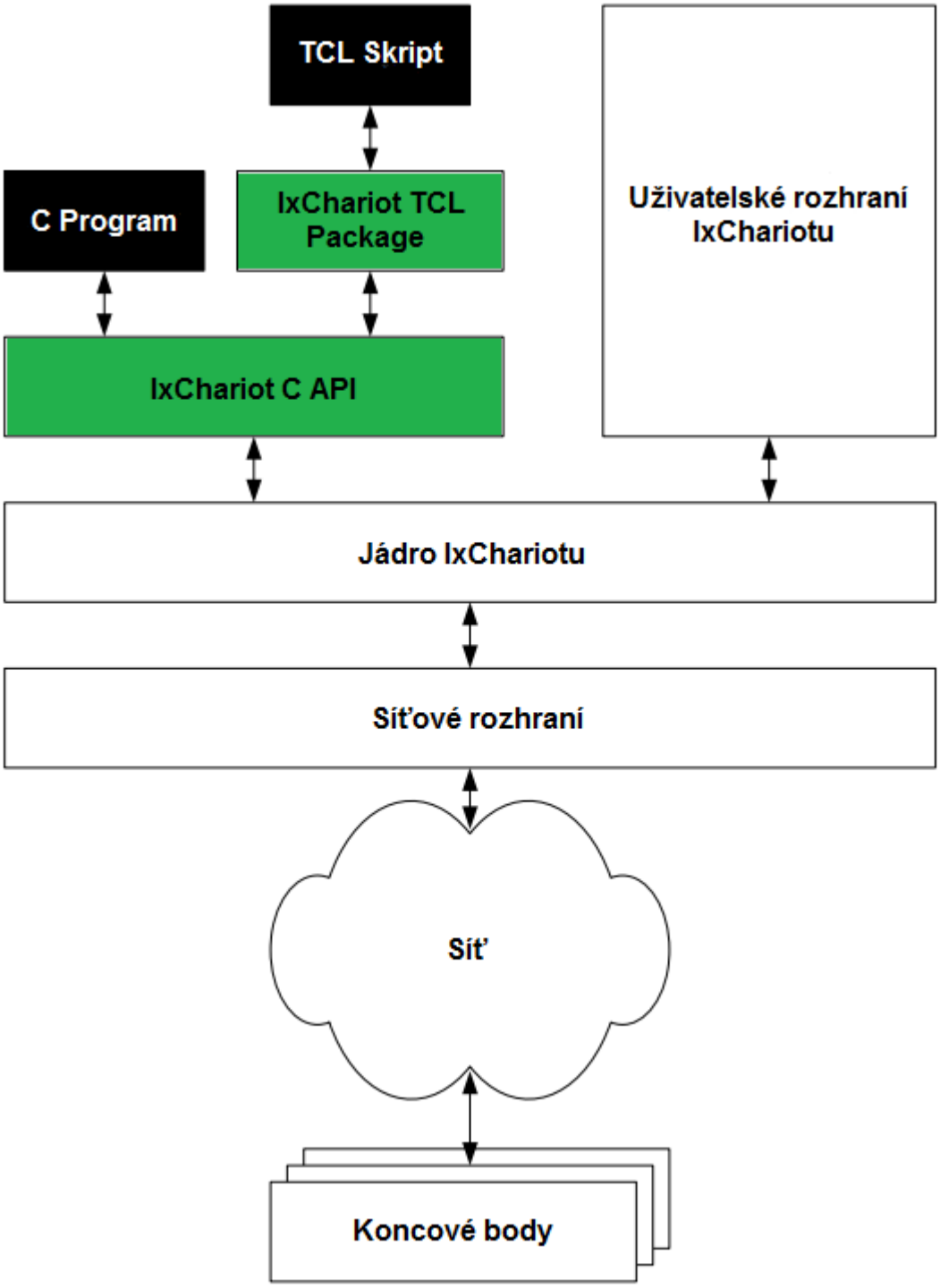
Několik šablon obsahujících konfiguraci QoS používanou v praxi je v IxChariotu předem připravených (VoIP, IPTV, kritická data apod.), ovšem uživatel má i možnost konfigurace vlastních příznaků. Kombinaci DSCP bitů lze nastavit libovolně (vhodné pro experimentální účely) – viz obr. 3.5.



Obr. 3.5 Okno nastavení pole DSCP

3.3. IxChariot API

API (Application Programming Interface) umožňuje tvorbu vlastních programů či TCL skriptů využívajících a rozšiřujících možnosti IxChariotu. Programy a skripty komunikací s API ovládají jádro IxChariotu. [5] TCL skripty ke komunikaci s API využívají nadstavby IxChariot TCL Package.



Obr. 3.6 Rozhraní pro interakci TCL skriptů a C programů s jádrem IxChariotu

Příkazy API umožňují komunikaci s jádrem IxChariotu na stejné úrovni, jako v případě použití uživatelského grafického rozhraní.

4. Spouštění jednoduchého předdefinovaného TCP toku z generátoru IxChariot přes TCL API

Prvním krokem realizace provázání OPNET Modeleru s generátorem síťového provozu IxChariot je ověření možností IxChariot TCL API. Toho je docíleno vytvořením samostatného TCL skriptu, komunikujícího s IxChariot TCL API. Test vytvořený tímto skriptem generuje jednoduchý datový tok bez nastavení identifikátoru kvality služeb. Skript je spouštěn z příkazového řádku prostřednictvím runtime prostředí TCL Shell (např. voláním spouštěcího souboru rozhraní s argumentem reprezentujícím cestu ke skriptu).

TCL skript po svém spuštění vytvoří, nastaví a spustí na pozadí IxChariot test prostřednictvím IxChariot TCL API. Test je po skončení svého průběhu uložen do souboru. V tomto souboru se kromě nastavení nacházejí i výsledky. Soubor je možné otevřít z grafického rozhraní IxChariotu a výsledky analyzovat.

Skript začíná deklarací obecných proměnných. Jako první je nastaven čas v sekundách, po který bude skript čekat na ukončení testu, který byl jeho prostřednictvím spuštěn. Tvůrci IxChariot TCL API doporučují tento čas nastavit na 120 sekund.

```
set timeout <seconds>
```

Následuje načítání dynamických knihoven, které obsahují funkce IxChariot TCL API. Funkcí pro načtení knihoven je v TCL více. V tomto skriptu je použita funkce *load*, která jako argument přebírá název souboru knihovny nebo název požadovaného balíčku nebo obojí. V tomto případě je použit název balíčku. Funkce *package require* kontroluje, zdali byl balíček načten korektně. V negativním případě zahlásí chybu a ukončí skript.

```
load ChariotExt
package require ChariotExt
```

Další sled příkazů využívá funkce z načtených knihoven pro vytvoření nového testu a jeho uložení do proměnné. Poté je vytvořen komunikační pár a je také nastaven soubor, do kterého bude test na závěr uložen. Parametr *filename* určuje jméno souboru (a případně také cestu k tomuto souboru, pokud chceme, aby byl soubor uložen do jiného adresáře, než ve kterém se nachází skript). V případě zadávání řetězce obsahujícího celou cestu k souboru je třeba mít na paměti, že s řetězcem je zacházeno podle norem standardu C99, kde zpětné lomítko uvozuje speciální znaky. Pro zapsání obyčejného zpětného lomítka do řetězce je proto nutné znak zdvojit (zápis zpětného lomítka tedy vypadá takto: „\\“).

```
set test [chrTest new]
set pair [chrPair new]
chrTest set $test FILENAME <filename>
```

Po této úvodní fázi začíná fáze nastavení parametrů testu. Ihned po vytvoření jsou testu nastaveny některé základní parametry. Z tohoto důvodu se nevytváří nová proměnná obsahující konfiguraci testu, ale extrahuje se aktuální (základní) konfigurace pomocí funkce *getRunOpts*.

```
set runOpts [chrTest getRunOpts $test]
```

Jedním z velmi důležitých parametrů testu je doba, po kterou test poběží. Touto dobou je myšlen simulovaný čas, nikoli čas skutečný. Simulovaný a virtuální čas nemusejí nutně korespondovat. Nejprve je třeba nastavit, že test poběží po určitou fixně danou dobu. Poté stačí zadat počet simulovaných sekund. Například při nastavení hodnoty na číslo 5 bude simulován datový tok trvající 5 sekund.

```
chrRunOpts set $runOpts TEST_END FIXED_DURATION
chrRunOpts set $runOpts TEST_DURATION <seconds>
```

Dále je nutné nastavit parametry komunikace. Mezi tyto parametry patří IP adresy komunikujících koncových bodů, použitý komunikační protokol apod. Parametry jsou přiřazeny proměnné *pair*, která reprezentuje jeden komunikační pár. IP adresa *E1_ADDR* odpovídá koncovému bodu na lokálním počítači, *E2_ADDR* reprezentuje vzdálený počítač (toto implicitní nastavení lze v případě potřeby změnit). Pro testovací účely existuje možnost nastavení obou adres na hodnotu *127.0.0.1*, což zapříčiní průběh komunikace přes lokální smyčku.

```
chrPair set $pair E1_ADDR <IP adresa>
chrPair set $pair E2_ADDR <IP adresa>
chrPair set $pair PROTOCOL "TCP"
```

Takto je nastaveno spojení mezi oběma komunikujícími stranami. Je ovšem ještě zapotřebí definovat datový tok a jeho chování. Tato definice je určena speciálním skriptem (soubor formátu SCR). Stačí zadat pouze cestu k tomuto skriptu (v IxChariotu je implicitně definováno několik skriptů, popisujících různé scénáře síťového provozu – FTP přenos, streamování videa apod.).

```
chrPair useScript $pair <skript>
```

Tímto je nastavení parametrů komunikačního páru dokončeno. Pár je třeba přiřadit k testu, v jehož rámci bude komunikovat. Jednomu testu lze přiřadit i více párů s různými konfiguracemi. Po přiřazení komunikačního páru je test připraven ke spuštění funkcí *start*.

```
chrTest addPair $test $pair
chrTest start $test
```

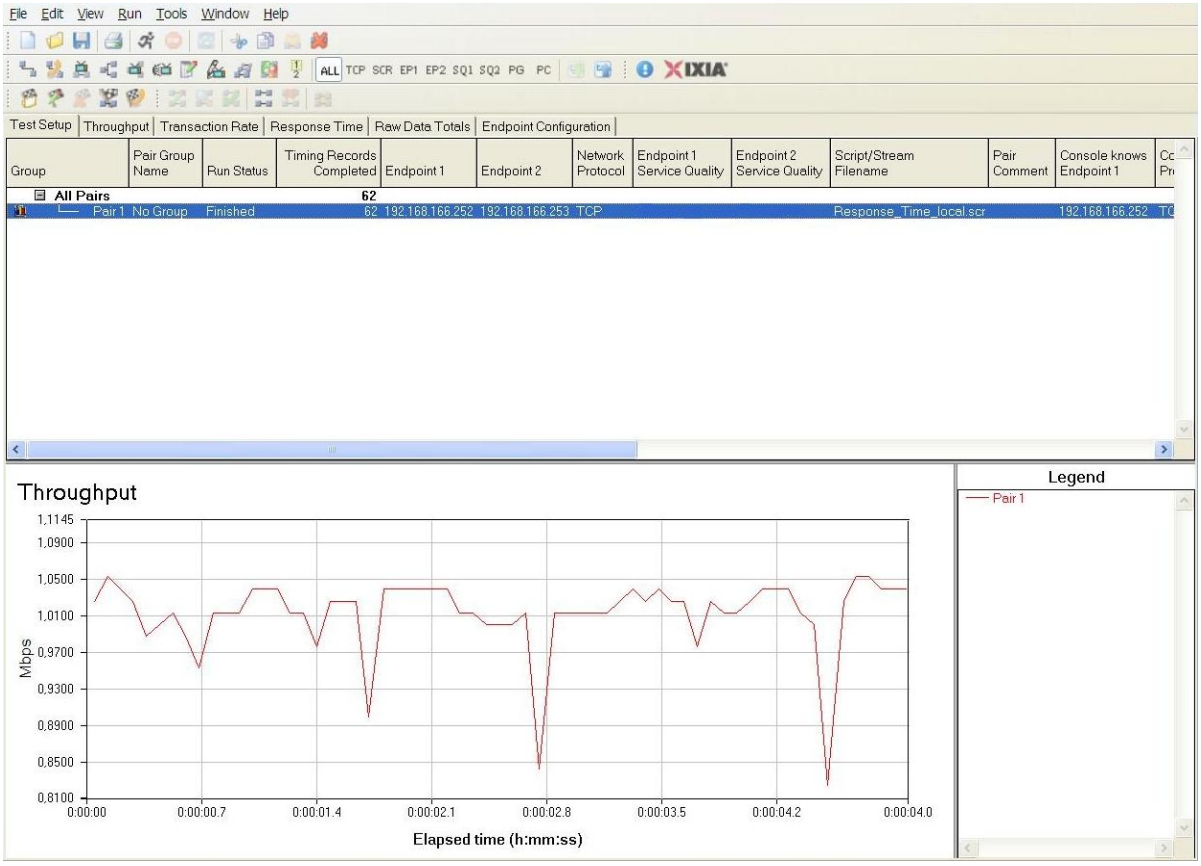
Po svém spuštění test běží po určitou dobu, která závisí na faktorech, jako je výkon a vytížení procesoru, míra stránkování atd. Test nelze uložit, dokud běží, a je tedy nutné nastavit mechanismus pro indikaci ukončení běhu testu. K tomu se používá funkce *isStopped*, která čeká na ukončení testu. Pro zamezení události, že kvůli chybě test poběží donekonečna, je volitelným argumentem této funkce *timeout*, což je čas, po který bude funkce zkoušet, zdali test pořád běží. Pokud test neskončí ani po uplynutí stanovené doby, vrátí funkce *isStopped* hodnotu 0 (*false*) a je volána funkce *stop*, která test „násilně“ ukončí.

```
if {![chrTest isStopped $test $timeout]} {
chrTest stop $test
}
```

Jakmile je zajištěno, že se test nachází ve stavu ukončen (*finished*), zavolá se funkce *save*, která uloží test do souboru, který je definován v parametru *FILENAME* (definice byla provedena na začátku skriptu). Poslední volanou funkcí je *return*, která ukončí běh TCL skriptu.

```
chrTest save $test
return
```

Po skončení skriptu lze vyhledat výsledný soubor a pomocí grafického rozhraní IxChariot jej otevřít. Okno zobrazení je stejné jako v případě testu, který byl konfigurován v grafickém rozhraní. V horní polovině okna je zobrazen definovaný komunikační pár se všemi svými parametry. V dolní polovině lze prohlížet nashromážděné výsledky (viz obr. 4.1).



Obr. 4.1 Zobrazení nastavení a výsledků testu v grafickém rozhraní

Pro kontrolu bylo na vzdáleném PC provedeno měření zachytávačem paketů *Wireshark*. Obsah jednoho ze zachycených paketů testu je zobrazen na obr. 4.2. V tomto bodě je podstatné, že komunikace byla na transportní vrstvě řízena protokolem TCP a IP adresy odpovídaly adresám nastaveným v testu. Nastavení QoS nebylo provedeno, hodnota pole DCSP zůstává nulová.

```
⊞ Frame 6113 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: Dell_c8:2e:cb (00:1d:09:c8:2e:cb), Dst: AsustekC_02:ef:cd (00:0e:a6:02:ef:cd)
⊟ Internet Protocol, Src: 192.168.166.252 (192.168.166.252), Dst: 192.168.166.253 (192.168.166.253)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 40
    Identification: 0xb614 (46612)
  ⊞ Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (0x06)
  ⊞ Header checksum: 0x7570 [correct]
    Source: 192.168.166.252 (192.168.166.252)
    Destination: 192.168.166.253 (192.168.166.253)
⊟ Transmission Control Protocol, Src Port: 2156 (2156), Dst Port: ibm-pps (1376), Seq: 304001, Ack: 304001, Len: 0
    Source port: 2156 (2156)
    Destination port: ibm-pps (1376)
    Sequence number: 304001    (relative sequence number)
    Acknowledgement number: 304001    (relative ack number)
    Header length: 20 bytes
  ⊞ Flags: 0x14 (RST, ACK)
    Window size: 0
  ⊞ Checksum: 0xe2cf [correct]
```

Obr. 4.2 Obsah zachyceného paketu testu

Na základě analýzy odchyceného datového toku na vzdáleném PC lze soudit, že test spouštěný skriptem generuje požadovaný datový tok a korektně sbírá informace, které jsou skriptem následně uloženy do souboru pro pozdější analýzu. Analýzou informací v uložených souborech testů se zabývá kapitola 9. Následující kapitola bude rozebírat možnost generování parametrizovaného datového toku s nastavením identifikátoru kvality služeb s využitím IxChariot TCL API.

5. Spouštění parametrizovaného TCP toku z generátoru IxChariot přes TCL API

Dalším dílčím krokem je průzkum možností nastavení identifikátoru kvality služeb v IP záhlaví paketů generovaného datového toku. Skript z kapitoly 4 bude doplněn o toto nastavení. V této kapitole je také popsán mechanizmus předávání argumentů při volání skriptu. Díky tomuto mechanizmu není třeba statické deklarace parametrů datového toku uvnitř kódu skriptu. Jejich deklaraci je možné provést vně zdrojového kódu skriptu, což je z hlediska požadavků následujících kapitol žádoucí.

Oproti jednoduchému toku bude výsledný generovaný tok doplněn o nastavení identifikátoru kvality služeb (QoS). S tímto nastavením jsou generovány pakety, v jejichž IP záhlaví jsou nastavena pole DSCP (6 bitů) a ECN (2bity).

Generátor IxChariot podporuje nastavení QoS pomocí bitových šablon, které jsou uloženy v souboru na disku. Implicitně obsahuje instalace IxChariotu šablony odpovídající všem známým druhům provozu (např. VoIP, IPTV, kritická data atd.). Dále je z grafického rozhraní přístupný editor šablon, který umožňuje vytvářet uživatelsky definované šablony. Tyto šablony mohou obsahovat libovolnou kombinaci nastavených bitů.

Při konfiguraci datového toku přes TCL API nabízí rozhraní funkci k přiřazení šablony ke komunikačnímu páru. Ve skriptu z kapitoly 4 stačí doplnit do definice vlastností páru následující řádek kódu:

```
chrPair set $pair QOS_NAME <template_name>
```

Parametr *template_name* určuje název šablony (nikoli cestu k souboru šablony). Kupříkladu šablona pro kritická data má název *„Critical data“*.

5.1. Spouštění skriptu s argumenty

Spouštěnému TCL skriptu z příkazového řádku lze předávat argumenty. V příkazu pro spuštění se argumenty zapisují za název skriptu. Uvozovky nejsou nutné, všechny argumenty jsou automaticky chápány jako řetězce. Oddělovacím znakem mezi argumenty je mezera. Zápis volání skriptu s parametry datového toku předávanými jako argumenty může vypadat takto:

```
tclsh85 tcp_qos.tcl 192.168.166.252 192.168.166.253 TCP
```

Zevnitř skriptu se k argumentům přistupuje skrz vektor argumentů v podobě proměnné *$argv*, podobně jako v jazyce C. Rozdílem oproti vektoru argumentů v C je skutečnost, že první argument ve vektoru (s indexem 0) je prvním uživatelsky definovaným argumentem z příkazového řádku. V C se na prvním místě vektoru argumentů nachází název a cesta k souboru programu. V TCL se k tomuto implicitnímu argumentu přistupuje přes proměnnou *$argv0*. [2]

K argumentům z výše uvedeného volání skriptu by se přistupovalo následovně:

```
chrPair set $pair E1_ADDR [lindex $argv 0]
chrPair set $pair E2_ADDR [lindex $argv 1]
chrPair set $pair PROTOCOL [lindex $argv 2]
```

Pomocí funkce *lindex* je ze seznamu vybrán požadovaný prvek. Volání funkce musí být uzavřeno v hranatých závorkách, aby překladač pochopil, že číslo za proměnnou *$argv* je dalším parametrem předávaným funkci.

5.2. Předávání identifikátoru třídy služeb argumentem

Předávání identifikátoru třídy služeb pomocí argumentu z příkazového řádku je mnohem komplikovanější záležitost než v případě předchozích parametrů. Rozhraní TCL API nabízí sadu funkcí pro vytváření, editaci či mazání souborů s QoS šablonami. Aby měl uživatel možnost zvolit libovolné nastavení, je ve skriptu použita speciální konstrukce.

```
expr [catch {chrApi newQosTosTemplate CHR_QOS_TEMPLATE_TOS_BIT_MASK \
      <nazev> <mask>} err]

if {$err == "creating QoS template failed: Value is invalid."} \
   {chrApi modifyQosTosTemplate CHR_QOS_TEMPLATE_TOS_BIT_MASK <nazev> \
   <mask>}
```

Funkce *newQosTosTemplate* přebírá argumenty typu šablony (zde je zvolen typ bitové masky), název šablony a bitová maska. Při předávání hodnoty bitovou maskou je pod pojmem maska rozuměno číslo v desítkové soustavě. Toto číslo je funkcí převedeno na číslo binární, podle kterého jsou nastaveny bity (0-7) v poli DSCP. Například pro nastavení hodnoty pole na 10011100 je použito číslo 156 (1 * 128 + 0 * 64 + 0 * 32 + 1 * 16 + 1 * 8 + 1 * 4 + 0 * 2 + 0 * 1).

Pokud ovšem šablona se zadaným názvem již v souboru se šablonami existuje, funkce pro její vytvoření zahlásí chybu a ukončí skript. Z tohoto důvodu je celé její volání umístěno do bloku (pomocí složených závorek) a je předáno funkci *catch*, která případnou navrácenou chybu pouze uloží do proměnné *err*.

V dalším kroku se nachází podmínka, která kontroluje, zdali došlo k chybovému stavu (tedy šablona s daným názvem již existuje). Pokud k chybě došlo, je volána funkce *modifyQosTosTemplate*, která namísto vytvoření nové šablony modifikuje již existující. Argumenty předávané funkci pro modifikaci jsou stejné jako v případě funkce pro vytváření šablony nové.

Po úspěšném vytvoření/modifikaci šablony dle uživatelského zadání je šablona přiřazena ke komunikačnímu páru (viz úvod této kapitoly).

Volání skriptu z příkazového řádku s QoS argumentem může vypadat takto:

```
tclsh85 tcp qos.tcl 192.168.166.252 192.168.166.253 TCP 156
```

Jelikož všechny argumenty z příkazového řádku jsou chápány jako řetězec, musí být argument nastavení DSCP převeden na číslo funkcí *expr* (viz následující příklad).

```
chrApi newQosTosTemplate CHR_QOS_TEMPLATE_TOS_BIT_MASK <nazev> \
      [expr [lindex $argv 3]]
```

5.3. Zachycené pakety

Po skončení testu jsou do souboru opět uložena veškerá nastavení a výsledky. V nastaveních uvnitř souboru je zobrazen i název použité QoS šablony.

Group	Pair Group Name	Run Status	Timing Records Completed	Endpoint 1	Endpoint 2	Network Protocol	Endpoint 1 Service Quality	Endpoint 2 Service Quality	Script/Stream Filename	Pair Comment	Console knows Endpoint 1	Console Protocol	Console Serv. Qual.
All Pairs			59										
Pair 1	No Group	Finished	59	192.168.166.252	192.168.166.253	TCP	OpnetTOS	OpnetTOS	Response_Time_local.scr		192.168.166.252	TCP	n/a

Obr. 5.1 Nastavení páru s QoS šablonou

Na vzdáleném PC se spuštěným zachytávačem paketů Wireshark lze ověřit nastavení bitů QoS (polí DSCP a ECN).

```
⊟ Internet Protocol, Src: 192.168.166.252 (192.168.166.252), Dst: 192.168.166.253 (192.168.166.253)
    Version: 4
    Header length: 20 bytes
  ⊟ Differentiated Services Field: 0x9c (DSCP 0x27: Unknown DSCP; ECN: 0x00)
      1001 11.. = Differentiated Services Codepoint: Unknown (0x27)
      .... ..0. = ECN-Capable Transport (ECT): 0
      .... ...0 = ECN-CE: 0
    Total Length: 152
    Identification: 0xc050 (49232)
  ⊞ Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (0x06)
  ⊞ Header checksum: 0x6a28 [correct]
    Source: 192.168.166.252 (192.168.166.252)
    Destination: 192.168.166.253 (192.168.166.253)
```

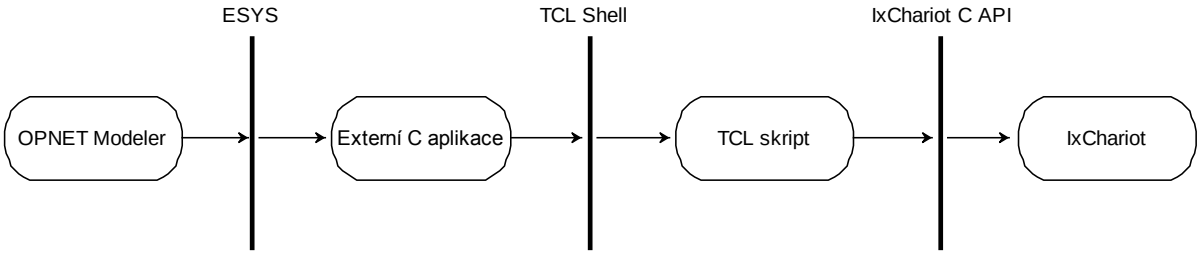
Obr. 5.2 Obsah paketu s nastaveným DSCP polem

Po úpravě skriptu z kapitoly 4 dle popisu v této kapitole je generován datový tok s nastavením identifikátorem kvality služeb. Parametry datového toku již nejsou deklarovány staticky uvnitř zdrojového kódu skriptu, ale jsou předávány při spouštění skriptu z příkazové řádky jako argumenty. V další kapitole bude popsán způsob volání skriptu z této a z předchozí kapitoly z prostředí OPNET Modeleru.

6. Spouštění parametrizovaného TCP toku z OPNET Modeleru přes TCL API

S připravenými TCL skripty, spouštějícími jednoduchý i parametrizovaný datový tok, přichází na řadu vytvoření vrstvy sloužící ke spouštění skriptů zevnitř simulace v OPNET Modeleru. V této kapitole je popsán mechanizmus přenosu dat uvnitř OPNET Modeleru i kód externí aplikace zajišťující extrakci dat ze simulace a spouštějící TCL skripty.

Ke spouštění parametrizovaného toku z OPNET Modeleru je zapotřebí využití dalšího rozhraní a jedné middleware aplikace. Vznikne tak pomyslný řetěz, který extrahuje z OPNET Modeleru data a na jejich základě vytvoří TCL skript, který bude produkovat datový tok s požadovanými parametry (viz obr. 6.1).



Obr. 6.1 Cesta přenášených dat z OPNET Modeleru do IxChariotu

V OPNET Modeleru slouží k extrakci dat rozhraní zvané ESYS (External System Definition). S tímto rozhraním je svázána aplikace kódovaná v jazyce C. Tato aplikace zajišťuje prvotní zpracování dat a jejich následné předání dalšímu článku řetězce.

Runtime prostředí TCL Shell je na obr. 6.1 zobrazeno jako rozhraní. Tato interpretace je velmi nepřesná, nicméně pro pochopení konceptu předávání dat je dostačující a poskytuje velké zjednodušení. V prostředí TCL Shell je spuštěn skript se zadanými parametry, který vygeneruje požadovaný datový tok. Struktura tohoto skriptu odpovídá skriptu z kapitoly 5.

TCL skript komunikuje s rozhraním IxChariot TCL Package, které je nadstavbou univerzálního rozhraní IxChariot C API a slouží k zapouzdření funkcí jazyka C do funkcí TCL.

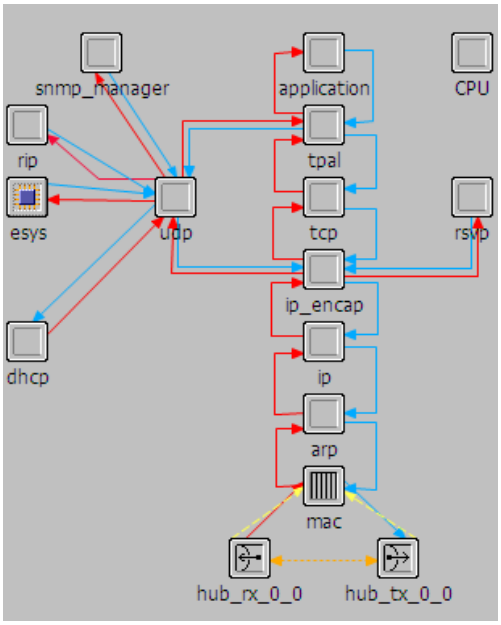
6.1. Simulace v OPNET Modeleru

V simulaci uvnitř OPNET Modeleru je na pracovní ploše umístěna jedna pracovní stanice obsahující proces generující pakety pro SNMP komunikaci. Vývoj tohoto procesu je součástí většího projektu a leží mimo rozsah zaměření této práce (více v [7]). Tok SNMP zpráv má nastavenu cílovou adresu rozhraní stanice, což způsobí vnitřní proudění paketů mezi procesy simulujícími zúčastněné aplikace. Tyto procesy jsou reprezentovány čísly portů transportní vrstvy. Výchozí port odpovídá portu pro SNMP komunikaci (161). K cílovému portu s číslem 162 se v rámci stanice registruje proces, starající se o zápis dat na ESYS rozhraní. Toto nastavení se provádí v atributech uzlu pracovní stanice (viz obr. 6.3).

SNMP	
snmp_manager.Agent Address	192.168.0.250
snmp_manager.Agent port	162
snmp_manager.Community St...	read
snmp_manager.DiffservMIB	(...)
snmp_manager.Local port	161
snmp_manager.Packet Interar...	30
snmp_manager.Start Time	10
snmp_manager.Stop Time	Infinity
snmp_manager.DSCP	2

Obr. 6.3 Nastavení generované SNMP komunikace

Vnitřní struktura stanice je zobrazena na obr. 6.4. Datový tok je generován procesem *snmp_manager* a putuje smyčkou přes procesy *udp*, *ip_encap*, *ip* a zpět do procesu *esys*.



Obr. 6.4 Model uzlu stanice

Data pro ovládání parametrů výsledného datového toku jsou z experimentálních důvodů přenášena v paketech v poli pro hodnotu *version* (pole původně sloužící pro přenos informace o použité verzi SNMP protokolu). Na základě celočíselné hodnoty (v rozmezí 0 – 255) v tomto poli bude vytvořena nová šablona QoS (viz podkapitola 5.2). Hodnota 0 vyjadřuje datový tok bez nastavených parametrů QoS. Hodnota tohoto pole je nastavena jako atribut procesu a povýšena na atribut celého uzlu stanice, jak je vidět na obr. 6.5. Ostatní pole v paketu nejsou podstatná a mohou být ponechána prázdná.

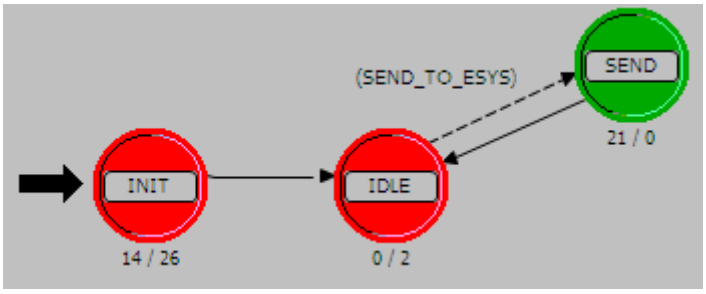
SNMP	
snmp_manager.Agent Address	192.168.0.250
snmp_manager.Agent port	162
snmp_manager.Community St...	read
snmp_manager.DiffservMIB	(...)
snmp_manager.Local port	161
snmp_manager.Packet Interar...	30
snmp_manager.Start Time	10
snmp_manager.Stop Time	Infinity
snmp_manager.DSCP	2

Obr. 6.5 Nastavení pole s informacemi o nastavení DSCP

Ve stavu SEND procesu *snmp_manager* (viz obr. 6.6) je pomocí funkce *op_ima_obj_attr_get* načtena hodnota nastavená uživatelem před začátkem simulace v grafickém rozhraní. Následující volání funkce *op_pk_fd_set* nastaví hodnotu pole v paketu.

```
op_ima_obj_attr_get(my_obj_id, "DSCP", &version);

op_pk_fd_set(paket, 0, OPC_FIELD_TYPE_STRUCT, version, sizeof(version)
      op_prg_mem_copy_create, op_prg_mem_free, sizeof (SNMPPAKET));
```



Obr. 6.6 Stavový automat procesu *snmp_manager*

Z paketu přijatého v procesu *esys* je zjištěna nastavená hodnota pole a je zapsána na ESYS rozhraní. Detailní popis funkce tohoto procesu, rozhraní a zřízení kosimulace lze nalézt v [3]. Po zápisu dat na ESYS rozhraní je pozastaven běh simulace a řízení je předáno externí aplikaci.

6.2. Externí C aplikace

Externí aplikace napsaná v jazyce C je zkompilována do dynamické knihovny, která obsahuje dvě exportované funkce. Zdrojový kód těchto funkcí se nachází v příloze C. Jedna z funkcí musí mít název *esa_main*. Zavaděč OPNET Modeleru hledá v externí aplikaci funkci s tímto názvem. Tato funkce obsahuje volání procedur inicializujících jádro OPNET Modeleru a v kosimulaci a je volána pouze jednou na počátku simulace. Její formát a funkce jednotlivých volání jsou podrobně popsány v [3].

Druhá exportovaná funkce se nazývá *callback* a je volána pokaždé, když jsou na ESYS rozhraní zapsána data ze strany OPNET Modeleru. Zde se nachází kód zpracovávající data a předávající je dalšímu článku v řetězci.

Prvním úkolem funkce *callback* je načtení dat z ESYS rozhraní. K tomu je zapotřebí nejprve voláním funkce *Esa_Interface_Get* z knihovny *esa.h*. Do speciální proměnné *iface* je uložen ukazatel na konkrétní rozhraní obsahující data. K nalezení požadovaného rozhraní je použito hierarchického názvu *<síť.podsíť.uzel.proces.rozhraní>*. [4] V dalším kroku je použita funkce *Esa_Interface_Value_Get* která načítá data z rozhraní a ukládá je do celočíselné proměnné *dscp*.

```
EsaT_Interface iface;
iface = Esa_Interface_Get(esaHandle, "top.office.manager.esys.Version");
Esa_Interface_Value_Get(esaHandle, &status, iface, &dscp);
```

Následující část kódu slouží k definici proměnných, které budou případně předávány skriptům jako parametry. Jsou to IP adresy koncových bodů, transportní protokol a maska QoS. Při zadávání

textových řetězců je třeba mít na paměti také umístění oddělovacích znaků pro argumenty TCL skriptů (mezery). V případě masky je zadána celočíselná hodnota, která je poté převedena na řetězec pomocí standardní funkce *itoa* a vložena do textové proměnné funkcí *sprintf*. V dalším kroku budou všechny parametry vloženy do jedné proměnné společně s cestou k souboru použitím kombinace funkcí *sprintf* a *strcat*.

```
char ip_addr_e1[17] = " 192.168.166.252";
char ip_addr_e2[17] = " 192.168.166.253";
char protocol[5] = " TCP";
int qos_mask = 156;
char qos_char[16];
char qos_char[16];
sprintf(qos char,itoa(qos mask,buf,10));
```

Ve fázi volání skriptu mohou nastat dvě varianty – varianta s nastavováním QoS a varianta bez nastavení QoS. V případě nenulové hodnoty z OPNET Modeleru je volán skript *tcp_qos.tcl* z kapitoly 5, v opačném případě je použit skript *tcp_no_qos.tcl* z kapitoly 4. Aby mohla být hodnota z OPNET Modeleru předána skriptu jako parametr, je třeba ji upravit do textové podoby funkcí *itoa*.

Pomocí funkcí *sprintf* a *strcat* jsou vytvořeny parametry pro volání skriptu. Výsledný řetězec slouží jako argument pro funkci *_popen*, která vyvolá nový proces a v návratové hodnotě vrací ukazatel na standardní výstup (*stdout*) tohoto procesu. V následujícím cyklu *while* je implementován proces zachytávání výstupu. Z proměnné obsahující ukazatel na výstup je přečteno 128 znaků a ty jsou poté standardní funkcí *printf* vypsány do konzole v OPNET Modeleru. Cyklus se opakuje, dokud se na výstupu neobjeví znak EOF (*end of file* = konec souboru) indikující ukončení skriptu. Na závěr je proces ukončen voláním *_pclose*.

Jedním z parametrů, společných pro oba skripty, je řetězec *„2>&1“*. Tímto zápisem je uskutečněno přesměrování chybového výstupu *stderr* na standardní výstup. Případná chybová hlášení tak budou rovněž zachycena a vypsána do konzole.

```
char path[256] = "C:\\tcl8.5.9\\win\\Release_VC9\\tclsh85.exe
                  C:\\tcl8.5.9\\win\\Release_VC9\\";
FILE *output;
char buffer[128];
sprintf(qos_char,itoa(dscp,buf,10));

if(dscp)
{
      sprintf(parameters,"%s","tcp_qos.tcl");
      strcat(parameters,ip_addr_e1);
      strcat(parameters,ip_addr_e2);
      strcat(parameters,protocol);
      strcat(parameters," ");
      strcat(parameters,qos_char);
      strcat(parameters," 2>&1");

      output = _popen(strcat(path,parameters), "rt");

      printf("Running script tcp_qos.tcl...\n");

      while(!feof(output))
      {
            if(fgets(buffer, 128, output) != NULL)
                  printf(buffer);
      }
      _pclose(output);
}
else
{
      sprintf(parameters,"%s","tcp_no_qos.tcl");
      strcat(parameters," 2>&1");

      output = _popen(strcat(path,parameters), "rt");
      printf("Running script tcp_no_qos.tcl...\n");

      while(!feof(output))
      {
            if(fgets(buffer, 128, output) != NULL)
                  printf(buffer);
      }
      _pclose(output);
}
```

Data byla přijata, zpracována a byl vyvolán příslušný skript generující datový tok s požadovanými parametry. Není tedy důvod k pokračování v simulaci. Pro případ, že by v simulaci zůstaly nějaké naplánované události, je volána funkce *Esa_Terminate*, která v OPNET Modeleru vyvolá proces ukončení simulace. Následně je nutné ukončit běh externí aplikace funkcí *exit*.

```
Esa_Terminate(esaHandle, ESAC_TERMINATE_NORMAL);
exit(0);
```

6.3. TCL skript

Vytvoření volaných TCL skriptů se nachází v kapitolách 4 a 5. První skript spouští TCP datový tok bez podpory QoS a druhý s podporou QoS a s předáváním parametrů. Skripty jsou v tomto případě spouštěny na pozadí. Standardní i chybový výstup skriptů je tím pádem zapotřebí zachytávat ve volající aplikaci (viz podkapitola 6.2).

Další variantou je explicitní vypisování do souboru, konfigurované přímo uvnitř skriptu. Voláním funkce *open* s parametrem jména souboru a *„w“* je v daném umístění otevřen soubor (neexistuje-li, je vytvořen) a pomocí ukazatele je možno do něj přesměrovávat výpisy funkce *puts*.

```
set logfile [open "C:\\ixchariot_tcl_api.log" "w"]
puts $logfile "Log skriptu pro ovládání TCL API IxChariotu"
```

Nevýhodou tohoto řešení je nemožnost přesměrování chybového výstupu a decentralizace kontrolních výpisů (konzole OPNET Modeleru a explicitní logovací soubor).

6.4. Shrnutí

V této a předchozích kapitolách byl popisován způsob ovládání generátoru síťového provozu IxChariot pomocí IxChariot TCL API. Kapitoly 4 a 5 popisují vytvoření TCL skriptů pro generování jednoduchého a parametrizovaného datového toku.

Tato kapitola se zaměřuje na spouštění těchto skriptů z prostředí simulace v OPNET Modeleru. K tomu je zapotřebí vytvoření externí aplikace v jazyce C, která má za úkol extrakci dat ze simulace pomocí ESYS rozhraní a spouštění TCL skriptu prostřednictvím TCL Shell. Aplikace také zajišťuje směrování výstupu skriptů do konzole v OPNET Modeleru.

Následující část práce se bude zabývat ovládáním generátoru IxChariot pomocí IxChariot C API a srovnáním výhod a nevýhod obou přístupů (C/TCL).

7. Spouštění parametrizovaného TCP toku přes IxChariot C API

Podobně jako se v kapitolách 4 a 5 vytvářel samostatně spouštěný TCL skript, je v této kapitole popsána konzolová aplikace v jazyce C, která má za úkol vytvoření IxChariot testu generujícího parametrizovaný datový tok (s nastavením identifikátoru kvality služeb).

Jak už samotný název napovídá, IxChariot C API je rozhraní založené na funkcích standardu jazyka C. Kromě TCL skriptů je tedy možné toto API ovládat také z programů v jazyce C. Postup pro vytvoření a spuštění datového toku je přibližně stejný, jako v případě TCL skriptu, nicméně se liší v drobných detailech. Je zapotřebí definovat vlastní funkci, ošetřující případné chyby při volání funkcí rozhraní, a mechanismus načítání a inicializace API je také odlišný.

Funkce IxChariot C API jsou do programu zahrnuty přidáním hlavičkového souboru *chrapi.h* pomocí direktivy *#include*. Při kompilování programu je třeba přidat do cesty linkeru adresáře, obsahující tento hlavičkový soubor a statickou knihovnu, na kterou se tento soubor odkazuje.

```
#include "chrapi.h"
```

Deklaraci proměnných obsahujících nastavení testu je vhodné provést globálně. Nastavení se za běhu programu měnit nebude a proměnné je tedy vhodné deklarovat s klíčovým slovem *static*. V těchto proměnných se uchovává jméno souboru, do kterého bude výsledný test uložen, IP adresy koncových bodů, transportní protokol, název, délka a maska QoS šablony, použitý komunikační skript a hodnoty pro časovače sloužící k detekci ukončení testu.

```
static CHR_STRING testFile = "opnettest.tst";
static CHR_STRING e1Addr = "192.168.166.252";
static CHR_STRING e2Addr = "192.168.166.253";
static CHR_PROTOCOL protocol = CHR_PROTOCOL_TCP;
static CHR_STRING qos_name = "opnettos";
static int qos_len = strlen(qos_name);
static CHR_BYTE qos_mask = 156;
static CHR_STRING script =
      "C:\\tcl8.5.9\\win\\Release_VC9\\Response_Time_local.scr";
static CHR_COUNT timeout = 5;
static CHR_COUNT maxWait = 120;
```

7.1. Hlavní funkce

Funkce main obsahuje výkonnou část kódu. Zajišťuje vytvoření, nastavení, spouštění, ukončení a uložení IxChariot testu. Pokud při volání funkcí z IxChariot C API dojde k chybě, je volána funkce *show_error* pro ošetření chyby (viz dále) a běh programu je ukončen.

Na počátku jsou deklarovány lokální proměnné. První dvě plní funkci ukazatelů na test a komunikační pár. Třetí proměnná *errorInfo* má charakter textového pole a slouží k ukládání informací o chybě (pokud nastane). Délka řetězce s informacemi o chybě je ukládána do čtvrté proměnné *errorLen*. Proměnná *isStopped* má za úkol indikaci ukončení běhu testu, zatímco hodnota proměnné

timer je používána časovačem při odpočítávání času běhu testu. Poslední proměnná *rc* slouží k zachycení návratového kódu volaných funkcí z IxChariot C API z důvodu detekce chyb.

```
CHR_TEST_HANDLE test;
CHR_PAIR_HANDLE pair;
char errorInfo[CHR_MAX_ERROR_INFO];
CHR_LENGTH errorLen;
CHR_BOOLEAN isStopped = CHR_FALSE;
CHR_COUNT   timer = 0;
CHR_API_RC rc;
```

První volaná funkce z IxChariot C API je *CHR_api_initialize*, která inicializuje celé rozhraní. Pokud její volání skončí neúspěšně, nelze volat specializovanou funkci pro zpracování chyb, jelikož se tato funkce opírá o konstrukce inicializovaného rozhraní. [5] V tomto případě jsou vypsány veškeré dostupné informace funkcí *printf*.

```
rc = CHR_api_initialize(CHR_DETAIL_LEVEL_ALL, errorInfo,
                        CHR_MAX_ERROR_INFO, &errorLen);
if (rc != CHR_OK)
{
      printf("Initialization failed: rc = %d\n", rc);
      printf("Extended error info:\n%s\n", errorInfo);
      exit(255);
}
```

K vytvoření nové instance testu slouží funkce *CHR_test_new*. Jejím jediným parametrem je odkaz na proměnnou, ve které bude nově vzniklá instance uložena. Návratový kód funkceje uložen do pomocné proměnné, která je následně podrobena testu na úspěšnost (tedy rovná-li se hodnotě *CHR_OK*). V negativním případě je volána funkce *show_error* pro ošetření chyby a ukončení programu. Tato procedura se repetitivně opakuje při každém volání funkce z IxChariot C API a dále již nebude zmiňována.

```
rc = CHR_test_new(&test);
if (rc != CHR_OK)
      show_error((CHR_HANDLE)NULL, rc, "test_new");
```

Nově vytvořenému testu je funkcí *CHR_test_set_filename* přiřazeno jméno souboru, do kterého bude později uložen. Pokud je funkcím z IxChariot C API předáván argument typu textového řetězce, je zapotřebí dalším argumentem definovat jeho délku. K tomuto účelu velmi dobře poslouží funkce *strlen* z knihovny *string.h*.

```
CHR_test_set_filename(test, testFile, strlen(testFile));
```

Dalším důležitým krokem je vytvoření komunikačního páru a přiřazení parametrů provozu. Páru jsou přiřazeny adresy koncových bodů, protokol a soubor s komunikačním skriptem.

```
CHR_pair_new(&pair);
CHR_pair_set_e1_addr(pair, e1Addr, strlen(e1Addr));
CHR_pair_set_e2_addr(pair, e2Addr, strlen(e2Addr));
CHR_pair_set_protocol(pair, protocol);
CHR_pair_use_script_filename(pair, script, strlen(script));
```

K vytváření nové šablony QoS slouží funkce *CHR_api_new_qos_tos_template*. Podobně jako v případě funkce ve skriptu TCL i zde existuje možnost vytvoření šablony na základě různého typu dat. Stejně jako v TCL skriptu byla i zde zvolena metoda bitové masky. V případě, že v seznamu již existuje šablona s požadovaným názvem, vrací funkce hodnotu *CHR_VALUE_INVALID*. Pokud k tomuto případu dojde, je zavolána funkce *CHR_api_modify_qos_tos_template* s totožnými parametry, která způsobí přepis existující šablony.

```
CHR_api_new_qos_tos_template(CHR_QOS_TEMPLATE_TOS_BIT_MASK, qos_name,
                             qos_len, qos_mask);
if (rc == CHR_VALUE_INVALID)
      CHR_api_modify_qos_tos_template(CHR_QOS_TEMPLATE_TOS_BIT_MASK,
                                      qos_name, qos_len, qos_mask);
CHR_pair_set_qos_name(pair, qos_name, qos_len);
```

Vytvořená QoS šablona se přiřadí k páru funkcí *CHR_pair_set_qos_name* s druhým parametrem odpovídajícím jménu šablony. Tímto je dokončeno nastavení páru a lze jej přiřadit k testu voláním *CHR_test_add_pair*. Takto nastavený test je již připraven ke spuštění, které je provedeno funkcí *CHR_test_start*.

```
CHR_pair_set_qos_name(pair, qos_name, qos_len);
CHR_test_add_pair(test, pair);
CHR_test_start(test);
```

Po zavolání startovací funkce test začne generovat síťový provoz. Běh programu mezitím nadále pokračuje. V programu je proto nutné implementovat čekací mechanizmus, který zajistí, že se program nebude snažit ukládat test v jeho průběhu. V podmínkách cyklu *while* se testuje proměnná, určující stav běhu testu (ukončen/stále běží), a velikost proměnné *timer* oproti předdefinované maximální hodnotě (viz globální proměnné na začátku reportu). V těle cyklu je volána funkce *CHR_test_query_stop*, která čeká určitý čas na základě argumentu *timeout* a poté vrátí kód určující stav testu. Vrátí-li hodnotu *CHR_OK*, test v zadaném časovém intervalu skončil a proměnná *isStopped* je nastavena na pravdivou hodnotu *CHR_TRUE*, což v dalším cyklu zapříčiní ukončení cyklování. Návratová hodnota *CHR_TIMED_OUT* indikuje stav, kdy test nadále pokračuje v běhu. K proměnné *timer* je přičtena doba, po kterou se v tomto cyklu čekalo, a pokračuje se dalším cyklem. Pokud funkce vrátí jinou hodnotu než tyto dvě, znamená to, že došlo k chybě a je zavolána funkce pro ošetření chyb.

```
while(!isStopped && timer < maxWait)
{
      rc = CHR_test_query_stop(test, timeout);
      if (rc == CHR_OK)
            isStopped = CHR_TRUE;
      else if (rc == CHR_TIMED_OUT)
      {
            timer += timeout;
            printf("Waiting for test to stop... (%d)\n", timer);
      }
      else
            show_error(test, rc, "test_query_stop");
}
```

Za cyklem se nachází další podmínka, kontrolující ukončení testu. Pokud i v této chvíli test stále běží, znamená to, že u časovače došlo k překročení maximální hodnoty čekací doby, a je vyvolána chyba.

Byl-li test úspěšně ukončen, může jej program uložit na disk (jméno a cesta k souboru odpovídají nastavení provedenému funkcí *CHR_test_set_filename*). Programem vytvořená šablona je ze souboru šablon vymazána a program končí s návratovým kódem indikujícím úspěch.

```
if (!isStopped)
      show_error(test, CHR_TIMED_OUT, "test_query_stop");

CHR_test_save(test);
CHR_api_delete_qos_template(qos_name, qos_len);
exit(EXIT_SUCCESS);
```

7.2. Funkce ošetřující chyby

Funkce ošetřující chyby, použitá v tomto programu, byla převzata z ukázkových kódů z IxChariot SDK. Její struktura bude popsána pouze velmi stručně a bez ukázek kódu (zdrojový kód funkce lze najít v příloze C).

Argumenty této funkce jsou ukazatel na objekt, který chybu vyvolal (např. instance testu, páru apod.), návratový kód sdělující druh chyby a název funkce, jejíž volání chybu způsobilo. V kódu se nachází volání IxChariot C API, které se dotazuje na zprávu popisující chybu. Tato zpráva je poté tisknuta na výstup.

Kromě běžných zpráv o chybě poskytuje IxChariot C API i rozšířené informace. Tyto informace (pokud jsou k dispozici) funkce načítá a ukládá do logovacího souboru.

7.3. Výstup

Výstupem tohoto programu je vygenerování datového TCP toku s nastavením odpovídajícím toku generovanému TCL skriptem v kapitole 5. Hodnota QoS masky je definována staticky uvnitř zdrojového kódu.

Výsledkem kompilace programu je konzolová aplikace. Do konzole jsou tisknuty kontrolní výpisy a případné informace o chybách. Obr. 7.1 ukazuje modelový příklad výstupu, kdy byla špatně zadána IP adresa jednoho z koncových bodů.

```
c:\Documents and Settings\ixchariot\Dokumenty\Visual Studio 2008\Projects\chariotcapi\Deb...
Create the test...
Create a pair...
Set pair attributes...
Run the test...
Wait for the test to stop...
Waiting for test to stop... (5)
Waiting for test to stop... (10)
Waiting for test to stop... (15)
Waiting for test to stop... (20)
Pair 1, Console to 192.168.166.252
Error was detected by the Console.
CHR0225: The TCP address is unreachable, the endpoint or phone is unavailable,
or the endpoint does not support the requested function.

Error was detected at Fri Apr 22 13:28:27 2011
Error was detected by the Console.
The return code was 4.
CHR0225: The TCP address is unreachable, the endpoint or phone is unavailable,
or the endpoint does not support the requested function.
```

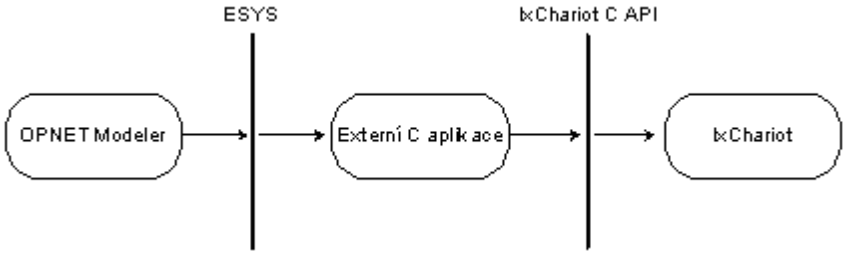
Obr. 7.1 Konzolový výstup programu s kontrolními výpisy

V této kapitole byla popsána jednoduchá konzolová aplikace, sloužící k ovládání IxChairot C API. Následující kapitola bude pojednávat o začlenění funkcí použitých v této aplikaci do kosimulace v OPNET Modeleru.

8. Spouštění TCP toku z OPNET Modeleru přes IxChariot C API

Tato kapitola se zabývá spouštěním datového toku pomocí IxChariot C API z OPNET Modeleru. Popisuje princip volání funkcí z konzolové aplikace z předchozí kapitoly. Závěr kapitoly je věnován experimentu, testujícímu kompatibilitu fyzických síťových zařízení, podporujících QoS, s nastavením DSCP pole provedeným generátorem IxChariot.

Mechanizmus spouštění TCP datového toku z OPNET Modeleru pomocí IxChariot C API má stejnou strukturu, jako při využití TCL API. Použití C API namísto TCL API přináší ovšem v případě této architektury značné výhody. Především se nabízí možnost volání funkcí z C API přímo z externí aplikace, extrahující data z OPNET Modeleru, neboť aplikace i ovládací funkce jsou kódovány v jazyce C. Dochází ke zkrácení a k zjednodušení řetězu předávání dat z kapitoly 6 o jeden článek (viz obr. 8.1). Odstraněním mezistupně volání TCL Shellu a skriptu jsou data zpracovávána rychleji a jsou kladeny menší nároky na softwarové vybavení hostujícího PC (rozhraní TCL Shell není zapotřebí).



Obr. 8.1 Diagram cesty dat z OPNET Modeleru do IxChariotu

8.1. OPNET Modeler

Konfigurace uvnitř simulace v OPNET Modeleru zůstává stejná jako v kapitole 6. V atributech stanice se nachází nastavení celočíselné hodnoty dat a ta jsou vnitřní smyčkou směrována do procesu zápisujícího na ESYS rozhraní. K ESYS rozhraní je přidružena externí aplikace v podobě dynamické knihovny, která v momentě zápisu dat na rozhraní přebírá řízení kosimulace. [3]

8.2. Externí C aplikace

Struktura externí aplikace zůstává rovněž prakticky stejná jako v kapitole 6. Obsahuje dvě exportované funkce *esa_main* a *callback*, kde funkce *esa_main* zajišťuje inicializaci kosimulace a registraci *callback* funkce k danému ESYS rozhraní a ve funkci *callback* se nachází výkonná část kódu, načítající a zpracovávající data a spouštějící datový tok.

Do hlavičky aplikace přibývá načtení knihoven IxChariot C API pomocí direktivy *#include* a deklarace globálních proměnných pro nastavení testu (stejné jako v kapitole 7). Další změnou oproti aplikaci z kapitoly 6 jsou dvě vnitřní funkce (tj. nejsou exportovány makrem *DLLEXPORT*), z nichž jedna se stará o vytvoření, konfiguraci a spuštění testu a druhá o ošetření chyb (viz kapitola 7).

Funkce *start_test*, ovládající test voláním funkcí z IxChariot C API, přebírá jako argument hodnotu z OPNET Modeleru. Tato hodnota je poté použita jako argument reprezentující bitovou TOS masku pro funkce *CHR_api_new_qos_tos_template* a *CHR_api_modify_qos_tos_template*.

```
void start_test(int qos);
```

Ve funkci *callback* je definována podmínka volání funkce. Provádí se kontrola relevantnosti hodnoty přijaté z OPNET Modeleru, tj. pohybuje-li se v intervalu od 0 do 255.

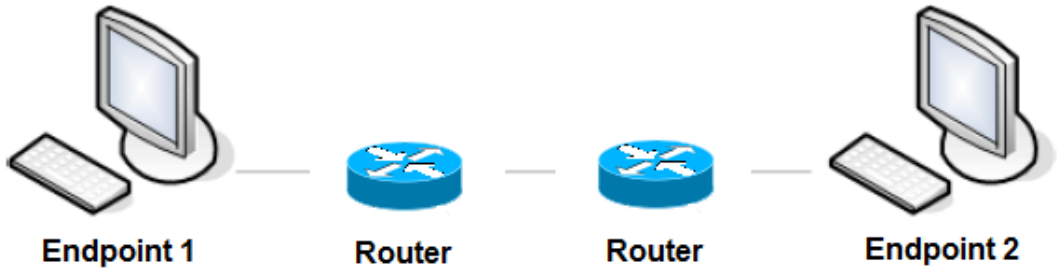
```
if(dscp => 0 && dscp < 256) start_test(dscp);
```

Uvnitř funkce *start_test* je oproti hlavní funkci z kapitoly 6 drobná změna v podobě podmínky pro nastavení QoS šablony pouze v případě, kdy je hodnota z OPNET Modeleru nenulová (nula indikuje datový tok bez QoS).

```
if (qos)
{
    rc = CHR_api_new_qos_tos_template(CHR_QOS_TEMPLATE_TOS_BIT_MASK,
                                      qos_name, qos_len, qos);
    if (rc == CHR_VALUE_INVALID)
        CHR_api_modify_qos_tos_template(CHR_QOS_TEMPLATE_TOS_BIT_MASK,
                                        qos_name, qos_len, qos);
    CHR_pair_set_qos_name(pair, qos_name, qos_len);
}
```

8.3. Výsledky testu průchodu datového toku fyzickým směrovačem s podporou QoS

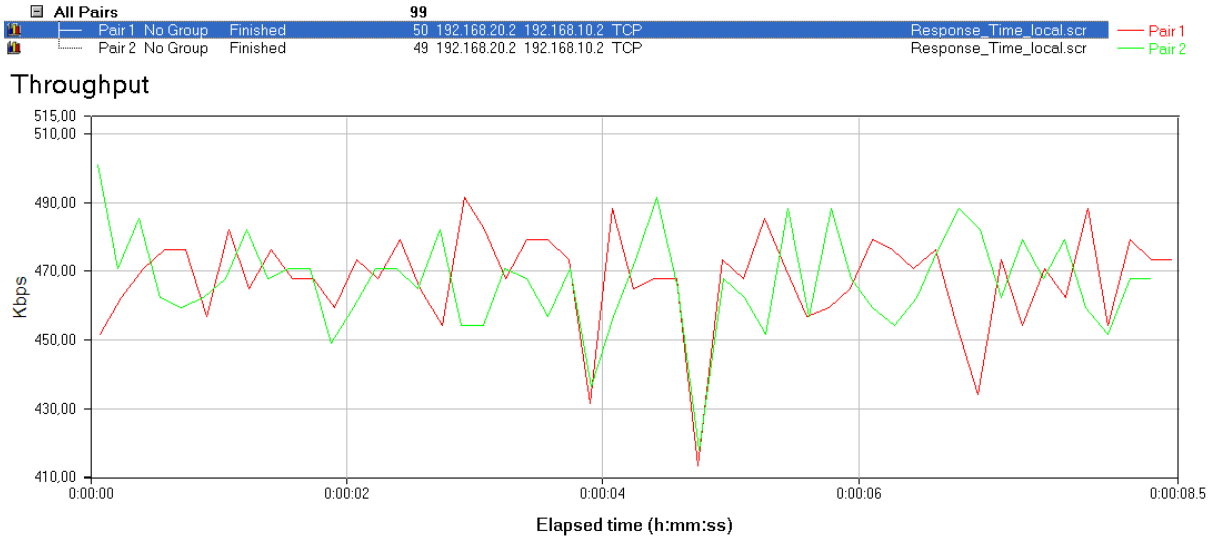
Za účelem prověření korektního nastavení QoS generátorem IxChariot byl vytvořen test obsahující dva komunikační páry ovládané skriptem *Reponse_Time_Local*. Koncové body těchto párů se nacházely na stejných stanicích propojených přes dva Cisco směrovače (viz obr. 8.2).



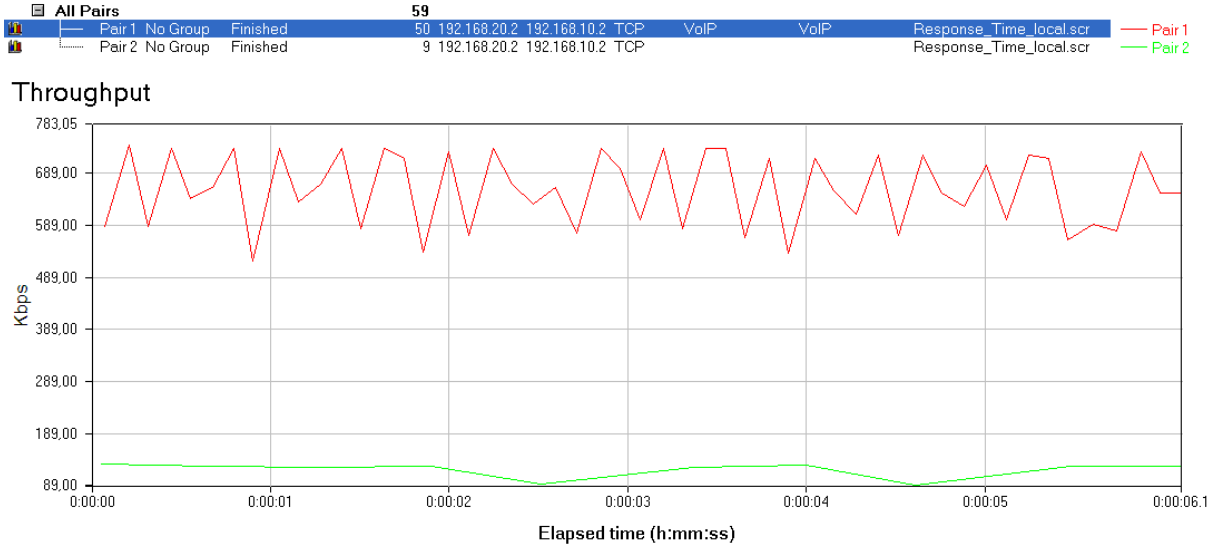
Obr 8.2 Propojení endpointů se zakomponovanými směrovači

Test byl spuštěn dvakrát; poprvé byly generovány oba datové toky bez nastavených příznaků QoS, podruhé byl jednomu z toků nastaven příznak *Expedited Flow* (VoIP). Obr. 8.3 ukazuje

propustnost sítě pro oba toky bez nastavených příznaků, na obr. 8.4 je zobrazena propustnost sítě při preferenci jednoho datového toku na úkor druhého.



Obr 8.3 Propustnost sítě pro datové toky bez nastavení QoS



Obr 8.4 Propustnost sítě pro datové toky s nastavením QoS

Druhý graf ukazuje stav, kdy datovým tokům bez nastavených QoS příznaků je přidělena šířka pásma pouze 100 kb/s. Propustnost směrovačů pro „zelený“ pár nepřesáhne tuto limitní hodnotu a tím pádem je „červenému“ páru poskytnuta větší šířka pásma, což se projeví vzrůstem propustnosti pro tento pár.

8.4. Shrnutí

V této části práce byly popisovány možnosti využití IxChariot C API pro ovládání datového toku z prostředí OPNET Modeler. Kapitola 7 popisovala vytvoření jednoduché konzolové aplikace v jazyce C. Tato aplikace je spouštěna samostatně prostřednictvím vlastního spouštěcího souboru.

V této kapitole byl popsán způsob sjednocení funkcí z konzolové aplikace a externí aplikace z kapitoly 6. Konzolová aplikace zaniká a funkce se stávají součásti externí aplikace v podobě dynamické knihovny, která je volána při spuštění simulace. Výstup funkcí je směrován do konzole v OPNET Modeleru.

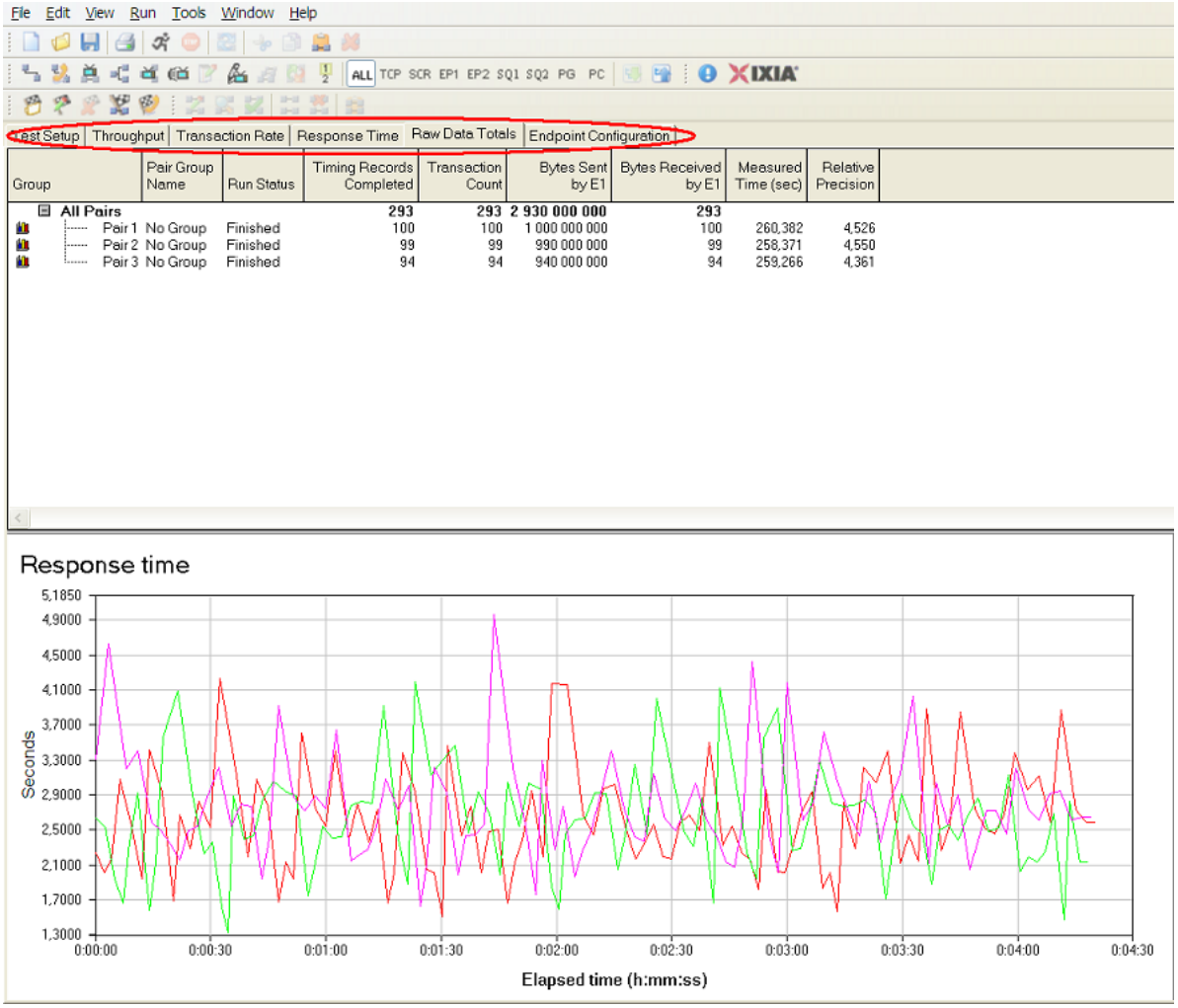
V kapitole 9 bude popsán princip zobrazování a analýzy výsledků testů prostřednictvím nástrojů generátoru IxChariot, především grafického uživatelského rozhraní. Pro případ potřeby zpracování výsledků pomocí jiných aplikací (např. MATLAB) je stručně objasněn princip exportu dat do obecných formátů.

9. Grafické zobrazení výsledků pomocí funkcí generátoru IxChariot

IxChariot testy během svého průběhu pravidelně sbírají informace o datovém toku a o síti, jíž tento tok prochází. Měřené atributy jsou například míra propustnosti sítě, velikost zpoždění atd. Po skončení je test aplikací uložen do souboru. Tento soubor obsahuje nejen veškerá nastavení, která byla aplikací provedena, ale zároveň také nasbírané statistiky. V grafickém rozhraní IxChariotu má pak uživatel možnost tento soubor otevřít a statistiky prohlédnout.

Za účelem demonstrace zpracování výsledků byl vytvořen test obsahující tři standardní komunikační páry ovládané skriptem *High_Performance_Throughput.scr*, které během testu komunikovaly paralelně.

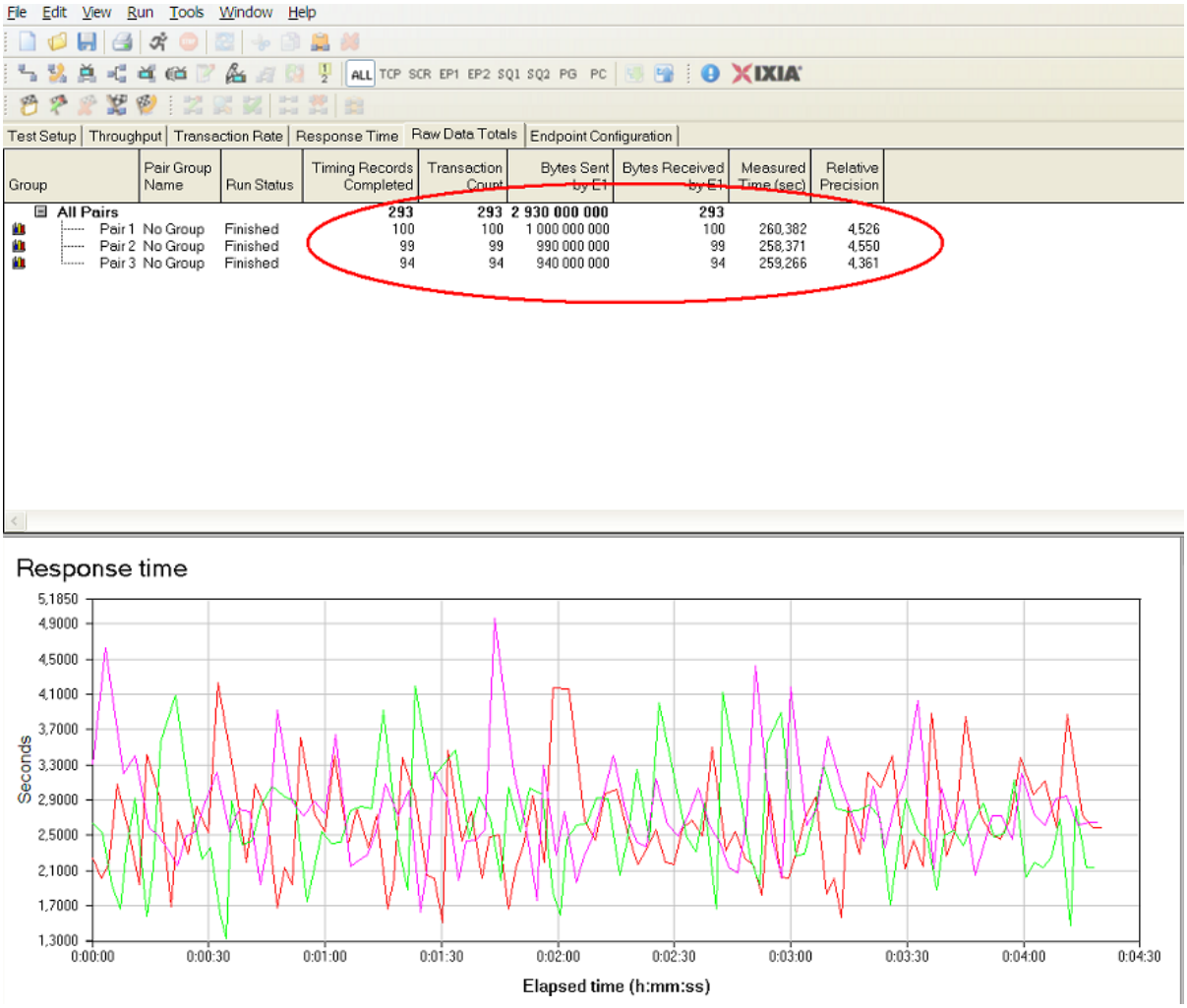
Graficky zpracované statistiky jsou v grafickém uživatelském rozhraní zobrazovány ve spodní části okna. Přepínání mezi nimi je řešeno záložkami umístěnými v horní části okna nad sekcí s konfigurací komunikačních párů (viz obr. 9.1).



Obr. 9.1 Umístění záložek pro výběr zobrazovaných statistik

Konkrétní informace, jako např. minimální/maximální hodnoty, objem přenesených dat apod. jsou zobrazovány v prostřední části v sekci s komunikačními páry (viz obr. 9.2). Hodnoty, jejichž

globální součet poskytuje relevantní informace, jsou v rámci skupin párů sumarizovány. Přepínání mezi zobrazovanými statistikami opět podléhá volbě záložky.



Obr. 9.2 Výpis naměřených individuálních hodnot pro každý pár a pro celou skupinu

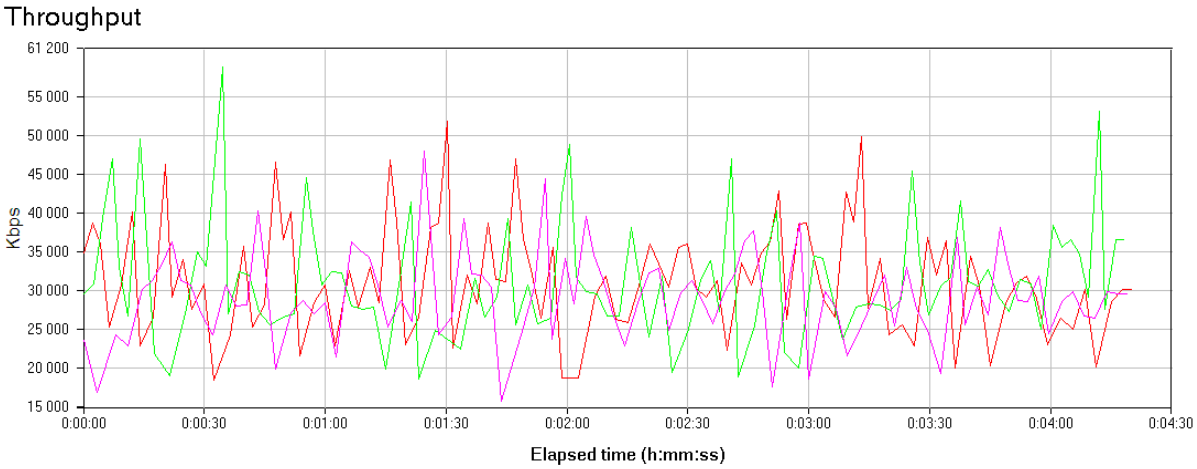
U základního typu páru jsou sbírány statistiky popisující propustnost sítě, počet zpracovaných transakcí, dobu odezvy a celkový objem přenesených dat.

9.1. Propustnost

Propustnost sítě je vlastnost vyjadřující průměrné množství dat úspěšně přenesených sítí. Měří se většinou v bitech za sekundu (b/s), ovšem vyskytují se i jednotky jako pakety za sekundu či pakety za časový rámec (timeslot).

Uvedené hodnoty představují maximální, minimální a průměrnou míru propustnosti v kilobitech za sekundu. Graf reprezentuje kolísání hodnot v průběhu testu.

	Pair Group Name	Run Status	Timing Records Completed	95% Confidence Interval	Average (Kbps)	Minimum (Kbps)	Maximum (Kbps)	Measured Time (sec)	Relative Precision	
All Pairs			**293**		**87 893,821**	**15 757,363**	**58 784,806**			
Pair 1	No Group	Finished	100	-1 357,957 : +1 357,957	30 003,997	18 469,269	51 910,304	260,382	4,526	Pair 1
Pair 2	No Group	Finished	99	-1 362,002 : +1 362,002	29 935,154	18 645,587	58 784,806	258,371	4,550	Pair 2
Pair 3	No Group	Finished	94	-1 235,137 : +1 235,137	28 325,159	15 757,363	48 076,928	259,266	4,361	Pair 3



Obr. 9.3 Statistiky míry propustnosti sítě

Na obr. 9.3 lze u párů vidět kromě výše uvedených údajů také hodnoty *95% Confidence Interval* (interval spolehlivosti) a *Relative Precision* (relativní přesnost). Jde o statistické veličiny popisující konzistenci naměřených a vypočtených hodnot. Jejich význam je takový, že devadesát pět procent naměřených hodnot spadá do oblasti průměru plus/mínus interval spolehlivosti. Relativní přesnost vyjadřuje procentuální odchylku od průměrné hodnoty. [1]

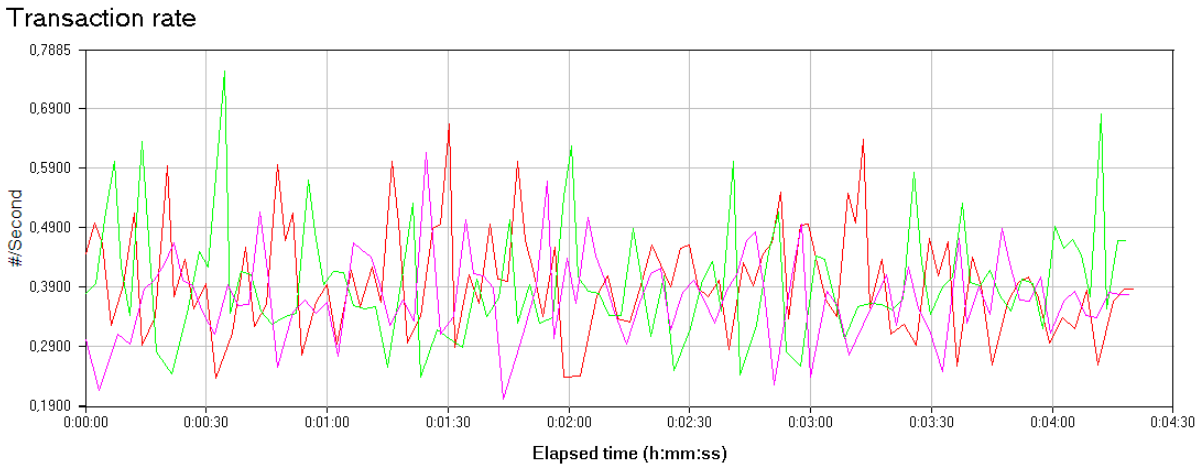
Aplikací těchto poznatků na naměřené výsledky lze v tomto případě získat informaci, že propustnost sítě pro komunikační pár číslo jedna měla hodnotu 30 003,997 kb/s +/- 1 357,957 kb/s nebo +/- 4,526%.

9.2. Počet zpracovaných transakcí

Údaje této statistiky vypovídají o míře transakcí provedených sítí za sekundu. Jednotkou je transakce za sekundu.

Data se sestávají z maximální, minimální a průměrné hodnoty, přičemž graf sleduje kolísání těchto hodnot.

	Pair Group Name	Run Status	Timing Records Completed	95% Confidence Interval	Transaction Rate Average	Transaction Rate Minimum	Transaction Rate Maximum	Measured Time (sec)	Relative Precision	
All Pairs			**293**		**1,130**	**0,202**	**0,752**			
Pair 1	No Group	Finished	100	-0,017 : +0,017	0,384	0,236	0,664	260,382	4,526	Pair 1
Pair 2	No Group	Finished	99	-0,017 : +0,017	0,383	0,239	0,752	258,371	4,550	Pair 2
Pair 3	No Group	Finished	94	-0,016 : +0,016	0,363	0,202	0,615	259,266	4,361	Pair 3



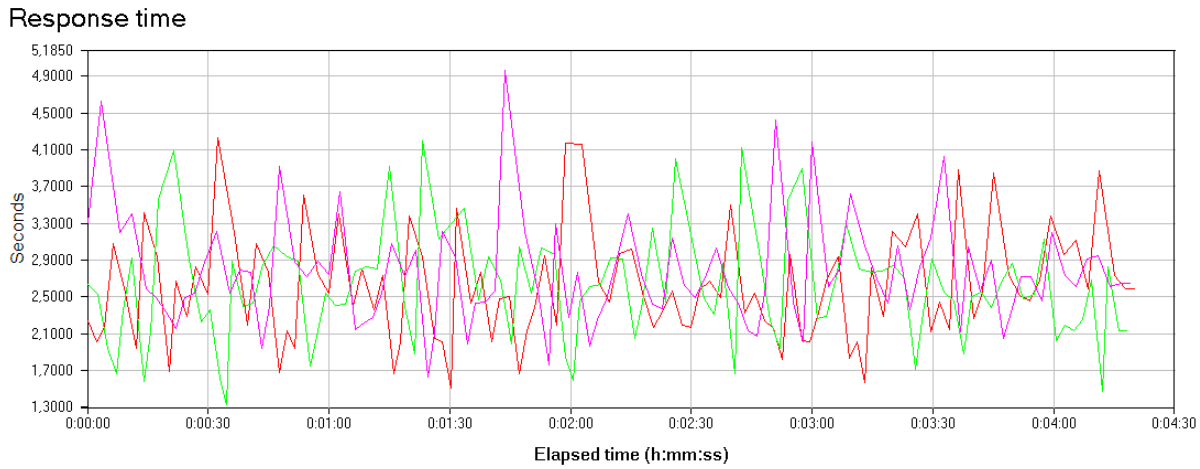
Obr. 9.4 Statistiky počtu transakcí zpracovaných sítí

9.3. Doba odezvy

Tato charakteristika určuje dobu mezi odesláním žádosti a přijetím odpovědi. Její velikost je ovlivněna rychlostmi mezilehlých linek, stupněm jejich agregace a případným zahlcením mezilehlých směrovačů.

Opět jsou zde uvedeny maximální a minimální hodnota a vypočtená průměrná hodnota doby odezvy. Graf zobrazuje vývoj hodnot v čase.

	Pair Group Name	Run Status	Timing Records Completed	95% Confidence Interval	Response Time Average	Response Time Minimum	Response Time Maximum	Measured Time (sec)	Relative Precision	
All Pairs			**293**		**2,657**	**1,329**	**4,958**			
Pair 1	No Group	Finished	100	-0,118 : +0,118	2,604	1,505	4,230	260,382	4,526	Pair 1
Pair 2	No Group	Finished	99	-0,119 : +0,119	2,610	1,329	4,190	258,371	4,550	Pair 2
Pair 3	No Group	Finished	94	-0,120 : +0,120	2,758	1,625	4,958	259,266	4,361	Pair 3



Obr. 9.5 Statistiky doby odezvy sítě

9.4. Celkový objem přenesených dat

Zde jsou uloženy informace o objemu dat zahrnutých do výpočtu statistik. Je zobrazen počet bytů odeslaných koncovými body E1 a E2 u jednotlivých párů, počet provedených průběžných záznamů a transakcí.

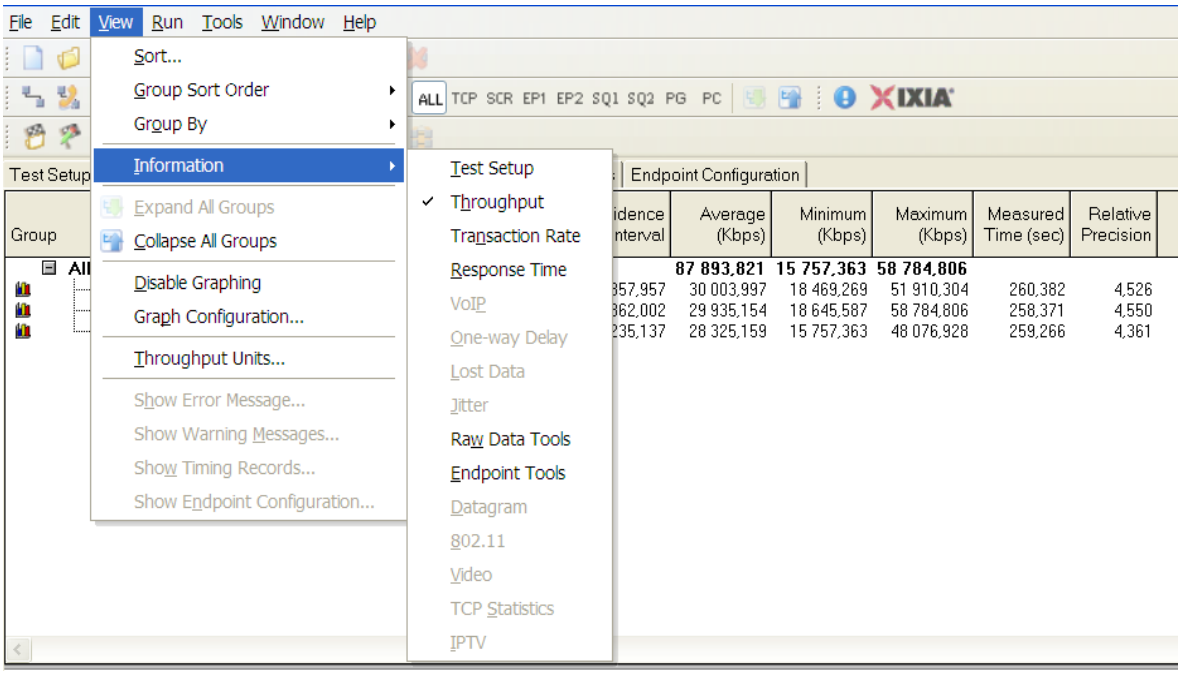
Tato statistika neobsahuje žádná data, jejichž vynášení do grafu by přinášelo relevantní informaci. Z tohoto důvodu je na obr. 9.6 vyobrazena pouze sekce s komunikačními páry.

	Pair Group Name	Run Status	Timing Records Completed	Transaction Count	Bytes Sent by E1	Bytes Received by E1	Measured Time (sec)	Relative Precision	
All Pairs			**293**	**293**	**2 930 000 000**	**293**			
Pair 1	No Group	Finished	100	100	1 000 000 000	100	260,382	4,526	Pair 1
Pair 2	No Group	Finished	99	99	990 000 000	99	258,371	4,550	Pair 2
Pair 3	No Group	Finished	94	94	940 000 000	94	259,266	4,361	Pair 3

Obr. 9.6 Údaje o celkovém objemu přenesených dat

9.5. Statistiky speciálních párů

Kromě standardních párů existují v IxChariotu také páry se specifickým účelem (VoIP, IPTV). Tyto páry mají rozšířené nastavení (např. o použitý zvukový kodek) a také jejich statistiky obsahují více informací. Průzkum statistik těchto specifických komunikačních párů se ovšem nachází mimo oblast zaměření této práce. Na obr. 9.7 je zobrazeno menu obsahující výčet veškerých dostupných statistik. Zobrazení statistik pro VoIP a IPTV páry je u standardních párů nedostupné.

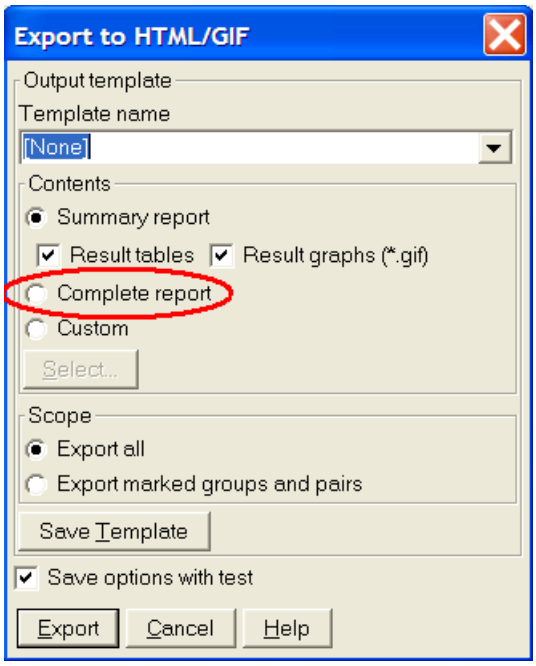


Obr. 9.7 Menu zobrazování statistik

9.6. Export statistik

Zobrazené statistiky je možno z testu exportovat do jiných datových formátů. Podporované formáty jsou čistý text (TXT soubor), soubor webové stránky (HTML), soubor typu comma-separated values (CSV – kompatibilní s MS Excel) a PDF dokument.

Volby exportu se nacházejí v menu File – Export. Obr. 9.8 ukazuje okno exportu do HTML formátu. Zatržením volby *Complete report* jsou do výsledného souboru exportována i data z grafů, která lze využít pro vytvoření vlastních grafů např. v programech MATLAB či MS Excel.



Obr. 9.8 Okno exportu výsledků

Formáty CSV a TXT obsahují pouze čistá data. PDF a HTML podporují i zobrazování grafických informací. Grafy jsou v tomto případě převedeny na soubory typu GIF a jsou uloženy do cílového umístění společně s hlavním souborem.

10. Závěr

Cílem práce bylo vytvoření dvou middleware aplikací pro ovládání generátoru síťového provozu IxChariot na základě dat ze simulace v OPNET Modeleru. Důvod k vytvoření dvou aplikací je existence dvou API generátoru podporujících standardy jazyka C nebo TCL.

Aplikace pracující s rozhraním na bázi jazyka TCL funguje na principu extrakce dat z OPNET Modeleru prostřednictvím ESYS rozhraní do dynamické knihovny v jazyce C. V této knihovně se nachází mechanizmus volání TCL skriptu a předání dat z OPNET Modeleru jako parametrů. TCL skript se stará o vytvoření IxChariot testu pomocí funkcí rozhraní, jeho konfiguraci a spouštění a následné uložení výsledků. Výstup skriptu je směrován do konzole simulace v OPNET Modeleru. Aplikace se skládá ze dvou zdrojových souborů: soubor v jazyce C pro dynamickou knihovnu a textový soubor obsahující TCL skript.

Aplikace využívající C API se celá nachází v dynamické knihovně. Vytváření, konfiguraci, spouštění testu i ukládání výsledků zajišťují vnitřní funkce této knihovny. Oproti první aplikaci má jednodušší strukturu, menší počet volání rozhraní mezi architekturami a centralizovanější správu (jediný zdrojový soubor).

Pro využití propojení generátoru síťového provozu IxChariot s aplikací OPNET Modeler je jednoznačně výhodnější použití aplikace pracující s IxChariot C API. Druhá varianta s sebou přináší několik komplikací včetně zvýšení požadavků na softwarové vybavení pracovní stanice (TCL Shell). OPNET Modeler je nástroj pracující na bázi standardu C99 a jeho úroveň kompatibility s IxChariot C API je vysoká. Použití komunikace přes IxChariot TCL API by přinášela výhody pouze v případě komunikace s aplikací, která je založena na bázi TCL (jako např. síťový simulátor NS-2).

11. Literatura

[1] IXIA. *IxChariot User Guide, Release 7.0. 913-0843 Rev. A.* July 2009

[2] *SourceForge.net* [online]. Geeknet, Inc., 2000, Last updated: 9 Jan 2009 [cit. 2011-05-07]. Tcl Reference Manual. Dostupné z WWW: <http://tmml.sourceforge.net/doc/tcl/index.html>.

[3] BARTL, M. *Aplikace zpracovávající reálný síťový provoz v prostředí OPNET Modeler.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 31 s.

[4] OPNET TECHNOLOGIES. *OPNET Modeler Documentation Set, Release 16.0.* OPNET Technologies Inc. July 2010

[5] IXIA. *IxChariot API Guide, Release 7.10. 913-0954-02 Rev. A.* June 2010

[6] BEDNÁRIK, J. *Modelovanie komunikácie proprietárnym protokolom, určeným pre výmenu informácií s podporovanou technológiou QoS, v prostredí Opnet Modeler*, Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2007. 35 s.

[7] HOŠEK, J.; RŮČKA, L.; MOLNÁR, K.; BARTL, M.; MATOCHA, T. *Integration of Real Network Components into OPNET Modeler Co- simulation Process. WSEAS TRANSACTIONS on COMMUNICATIONS*, 2010, volume 9, issue 9, p. 553-562. ISSN: 1109- 2742.

Příloha A – Zdrojový kód TCL skriptů

Skript tcp_no_qos.tcl

```
set timeout 120

puts "loading libraries"
load ChariotExt
package require ChariotExt

puts "creating test"
set test [chrTest new]

puts "creating pair variable"
set pair [chrPair new]

chrTest set $test FILENAME "C:\\tcl8.5.9\\win\\Release_VC9\\opnettest.tst"

set runOpts [chrTest getRunOpts $test]

chrRunOpts set $runOpts TEST_END FIXED_DURATION
chrRunOpts set $runOpts TEST_DURATION 5

puts "setting pair options"
chrPair set $pair E1_ADDR "192.168.166.252"
chrPair set $pair E2_ADDR "192.168.166.253"

chrPair set $pair PROTOCOL "TCP"

puts "loading script"
chrPair useScript $pair "C:\\tcl8.5.9\\win\\Release_VC9\\Response_Time_local.scr"

chrTest addPair $test $pair

puts "starting test"
chrTest start $test

puts "stopping test"
if {![chrTest isStopped $test $timeout]} {
puts "ERROR: Test didn't stop in 2 minutes!"
chrTest stop $test
}

puts "saving test"
chrTest save $test

puts "test finished"
return
```

Skript tcp_qos.tcl

```
set timeout 120

puts "loading libraries"
load ChariotExt
package require ChariotExt
```

```
puts "creating test"
set test [chrTest new]

puts "creating pair variable"
set pair [chrPair new]

chrTest set $test FILENAME "C:\\tcl8.5.9\\win\\Release_VC9\\opnettest.tst"

set runOpts [chrTest getRunOpts $test]

chrRunOpts set $runOpts TEST_END FIXED_DURATION
chrRunOpts set $runOpts TEST_DURATION 5

puts "setting pair options"
chrPair set $pair E1_ADDR [lindex $argv 0]
chrPair set $pair E2_ADDR [lindex $argv 1]

chrPair set $pair PROTOCOL [lindex $argv 2]

puts "creating QoS template"
expr  [catch  {chrApi  newQosTosTemplate  CHR_QOS_TEMPLATE_TOS_BIT_MASK  "OpnetTOS"
[expr [lindex $argv 3]]} err]
if {$err == "creating QoS template failed: Value is invalid."} \
       {chrApi modifyQosTosTemplate CHR_QOS_TEMPLATE_TOS_BIT_MASK "OpnetTOS" \
       [expr [lindex $argv 3]]}
chrPair set $pair QOS_NAME "OpnetTOS"

puts "loading script"
chrPair useScript $pair "C:\\tcl8.5.9\\win\\Release_VC9\\Response_Time_local.scr"

chrTest addPair $test $pair

puts "starting test"
chrTest start $test

puts "stopping test"
if {![chrTest isStopped $test $timeout]} {
puts "ERROR: Test didn't stop in 2 minutes!"
chrTest stop $test
}

puts "saving test"
chrTest save $test

puts "test finished"
return
```

Příloha B – Konzolová aplikace komunikující s IxChariot C API

Globální deklarace

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "chrapi.h"

static
CHR_STRING testFile = "opnettest.tst";
static
CHR_STRING e1Addr = "192.168.166.252";
static
CHR_STRING e2Addr = "192.168.166.253";
static
CHR_PROTOCOL protocol = CHR_PROTOCOL_TCP;
static
CHR_STRING qos_name = "OpnetTOS";
static
int qos_len = strlen(qos_name);
static
CHR_BYTE qos_mask = 156;
static
CHR_STRING script = "C:\\tcl8.5.9\\win\\Release_VC9\\Response_Time_local.scr";
static
CHR_COUNT timeout = 5;    /* 5 sekund */
static
CHR_COUNT maxWait = 120;  /* 2 minuty */
static
CHR_STRING logFile = "pairsTest.log";

static void
show_error(
    CHR_HANDLE handle,
    CHR_API_RC code,
    CHR_STRING where);
```

Hlavní funkce

```
int main(int argc, char** argv)
{
    CHR_TEST_HANDLE test;
    CHR_PAIR_HANDLE pair;

    char errorInfo[CHR_MAX_ERROR_INFO];
    CHR_LENGTH errorLen;

    CHR_COUNT index;

    CHR_BOOLEAN isStopped;
    CHR_COUNT   timer = 0;

    CHR_API_RC rc;
```

```
rc = CHR_api_initialize(CHR_DETAIL_LEVEL_ALL, errorInfo,
                        CHR_MAX_ERROR_INFO, &errorLen);
if (rc != CHR_OK) {
    printf("Initialization failed: rc = %d\n", rc);
    printf("Extended error info:\n%s\n", errorInfo);
    exit(255);
}

printf("Create the test...\n");
rc = CHR_test_new(&test);
if (rc != CHR_OK)
    show_error((CHR_HANDLE)NULL, rc, "test_new");

rc = CHR_test_set_filename(test, testFile, strlen(testFile));
if (rc != CHR_OK)
    show_error(test, rc, "test_set_filename");

char comment[CHR_MAX_PAIR_COMMENT];

printf("Create a pair...\n");
rc = CHR_pair_new(&pair);
if (rc != CHR_OK)
    show_error((CHR_HANDLE)NULL, rc, "pair_new");

printf("Set pair attributes...\n");

rc = CHR_pair_set_e1_addr(pair, e1Addr, strlen(e1Addr));
if (rc != CHR_OK)
    show_error(pair, rc, "pair_set_e1_addr");

rc = CHR_pair_set_e2_addr(pair, e2Addr, strlen(e2Addr));
if (rc != CHR_OK)
    show_error(pair, rc, "pair_set_e2_addr");

rc = CHR_pair_set_protocol(pair, protocol);
if (rc != CHR_OK)
    show_error(pair, rc, "pair_set_protocol");

rc = CHR_pair_use_script_filename(pair, script, strlen(script));
if (rc != CHR_OK)
    show_error(pair, rc, "pair_use_script_filename");

rc = CHR_api_new_qos_tos_template(CHR_QOS_TEMPLATE_TOS_BIT_MASK, qos_name,
    qos_len, qos_mask);

if(rc == CHR_VALUE_INVALID)
    rc = CHR_api_modify_qos_tos_template(CHR_QOS_TEMPLATE_TOS_BIT_MASK,
    qos_name, qos_len, qos_mask);

if (rc != CHR_OK)
    show_error(pair, rc, "pair_create_qos_tos_template");

rc = CHR_pair_set_qos_name(pair, qos_name, qos_len);
if (rc != CHR_OK)
    show_error(pair, rc, "pair_set_qos_name");
```

```
    rc = CHR_test_add_pair(test, pair);
    if (rc != CHR_OK)
        show_error(test, rc, "test_add_pair");

    printf("Run the test...\n");
    rc = CHR_test_start(test);
    if (rc != CHR_OK)
        show_error(test, rc, "start_ test");

    printf("Wait for the test to stop...\n");
    isStopped = CHR_FALSE;
    timer = 0;

    while(!isStopped && timer < maxWait) {
        rc = CHR_test_query_stop(test, timeout);

        if (rc == CHR_OK)
            isStopped = CHR_TRUE;

        else if (rc == CHR_TIMED_OUT) {
            timer += timeout;
            printf("Waiting for test to stop... (%d)\n", timer);
        }

        else
            show_error(test, rc, "test_query_stop");
    }

    if (!isStopped)
        show_error(test, CHR_TIMED_OUT, "test_query_stop");

    printf("Save the test..\n");
    rc = CHR_test_save(test);
    if (rc != CHR_OK)
        show_error(test, rc, "test_save");

      CHR_api_delete_qos_template(qos_name, qos_len);
      printf("Test completed.");
      getchar();
      exit(EXIT_SUCCESS);
}
```

Funkce pro ošetření chyb

```
static void
show_error(
    CHR_HANDLE handle,
    CHR_API_RC code,
    CHR_STRING where)
{
    char msg[CHR_MAX_RETURN_MSG];
    CHR_LENGTH msgLen;

    char errorInfo[CHR_MAX_ERROR_INFO];
    CHR_LENGTH errorLen;
```

```
    FILE *fd;
    time_t currentTime;
    char *timestamp;

    CHR_API_RC rc;

    rc = CHR_api_get_return_msg(code, msg, CHR_MAX_RETURN_MSG, &msgLen);

    if (rc != CHR_OK) {
        printf("%s failed\n", where);
        printf("Unable to get message for return code %d, rc = %d\n", code, rc);
    }
    else {
        printf("%s failed: rc = %d (%s)\n", where, code, msg);
    }

    if ((code == CHR_OPERATION_FAILED || code == CHR_OBJECT_INVALID) && handle !=
        (CHR_HANDLE) NULL) {
        fd = fopen(logFile, "a+");

        currentTime = time(NULL);
        timestamp = ctime(&currentTime);

        rc = CHR_common_error_get_info(handle, CHR_DETAIL_LEVEL_ALL, errorInfo,
                                       CHR_MAX_ERROR_INFO, &errorLen);

        if (rc == CHR_OK) {
            if (fd != NULL) {
                fprintf(fd, "%s %s failed\n", timestamp, where);
                fprintf(fd, "%s %s\n", timestamp, errorInfo);

                fclose(fd);
            }
        }
    }

    getchar();
    exit(EXIT_FAILURE);
}
```

Příloha C – Externí aplikace

Funkce callback pro IxChariot TCL API

```
extern "C" DLLEXPORT

void callback(void *state_ptr, double time, va_list vararg)
{
	int status;
	EsaT_Interface iface;
	int dscp;

	iface = Esa_Interface_Get(esaHandle, "top.office.manager.esys.Version");
		Esa_Interface_Value_Get(esaHandle, &status, iface, &dscp);

	printf("[EXT] Data prijata v externi aplikaci: %d\n", dscp);

	char ip_addr_e1[17] = " 192.168.166.252";
	char ip_addr_e2[17] = " 192.168.166.253";
	char protocol[5] = " TCP";
	char qos_char[16];
	char buf[5];
	char parameters[64];

	sprintf(qos_char,itoa(dscp,buf,10));

	FILE *output;
	char buffer[128];

	if(dscp)
	{
		sprintf(parameters,"%s","tcp_qos.tcl");
		strcat(parameters,ip_addr_e1);
		strcat(parameters,ip_addr_e2);
		strcat(parameters,protocol);
		strcat(parameters," ");
		strcat(parameters,qos_char);
		strcat(parameters," 2>&1");

		output = _popen(strcat(path,parameters), "rt");

		printf("Running script tcp_qos.tcl...\n");

		while(!feof(output))
		{
			if(fgets(buffer, 128, output) != NULL)
				printf(buffer);
		}
		_pclose(output);
	}
	else
	{
		output = _popen(strcat(path,"tcp_no_qos.tcl"), "rt");
		printf("Running script tcp_no_qos.tcl...\n");

		while(!feof(output))
		{
			if(fgets(buffer, 128, output) != NULL)
				printf(buffer);
		}
		_pclose(output);
	}
	Esa_Terminate(esaHandle, ESAC_TERMINATE_NORMAL);
	exit(0);
}
```

Funkce callback pro IxChariot C API

```
extern "C" DLLEXPORT

void callback(void *state_ptr, double time, va_list vararg)
{
	int status;
	EsaT_Interface iface;
	int dscp;

	iface = Esa_Interface_Get(esaHandle, "top.office.manager.esys.Version");
		Esa_Interface_Value_Get(esaHandle, &status, iface, &dscp);

	printf("[EXT] Data prijata v externi aplikaci: %d\n", dscp);

	if(dscp => 0 && dscp < 256) start_test(dscp);

	Esa_Terminate(esaHandle, ESAC_TERMINATE_NORMAL);
	exit(0);
}
```

Funkce esa_main

```
extern "C" DLLEXPORT

int esa_main(int argc, char *argv[], int stat)
{
	EsaT_Interface callback_interface;
	int status;
	double cas = 0.0;
	int udalost;

	WSADATA ws;
	WSAStartup(0x0101,&ws);

	Esa_Init(argc, argv, ESAC_OPTS_NONE, &esaHandle);

	Esa_Load(esaHandle, ESAC_OPTS_NONE);

	if((callback_interface = Esa_Interface_Get(esaHandle,
		"top.office.manager.esys.Version")) == ESAC_INTERFACE_NULL)
	{
		MessageBox (NULL, (LPCWSTR)"Chyba pri ziskavani ID interface",
				NULL,0);

		Esa_Terminate(esaHandle, ESAC_TERMINATE_NORMAL);

		exit(1);
	}

	Esa_Interface_Callback_Register(esaHandle, &status, callback_interface,
		callback, (EsaT_Interface_Array_Callback_Proc)NULL,(void *)NULL);

	Esa_Execute_Until(esaHandle, &status, ESAC_TIME_MAX, ESAC_UNTIL_EXCLUSIVE,
		&cas, &udalost);

	printf("\nSimulace ukoncena v case: %d", cas);

	exit(0);
  }
```

Funkce start_test

```
void start_test(int qos)
{
	CHR_TEST_HANDLE test;
	CHR_PAIR_HANDLE pair;

	char errorInfo[CHR_MAX_ERROR_INFO];
	CHR_LENGTH errorLen;

	CHR_COUNT index;

	CHR_BOOLEAN isStopped;
	CHR_COUNT   timer = 0;

	CHR_API_RC rc;

	rc = CHR_api_initialize(CHR_DETAIL_LEVEL_ALL, errorInfo,
		CHR_MAX_ERROR_INFO, &errorLen);
	if (rc != CHR_OK)
	{
	 printf("Initialization failed: rc = %d\n", rc);
	 printf("Extended error info:\n%s\n", errorInfo);
	 exit(255);
	}

	printf("Create the test...\n");
	rc = CHR_test_new(&test);
	if (rc != CHR_OK)
		show_error((CHR_HANDLE)NULL, rc, "test_new");

	rc = CHR_test_set_filename(test, testFile, strlen(testFile));
	if (rc != CHR_OK)
		show_error(test, rc, "test_set_filename");

	printf("Create a pair...\n");
	rc = CHR_pair_new(&pair);
	if (rc != CHR_OK)
		show_error((CHR_HANDLE)NULL, rc, "pair_new");

	printf("Set pair attributes...\n");

	rc = CHR_pair_set_e1_addr(pair, e1Addr, strlen(e1Addr));
	if (rc != CHR_OK)
		show_error(pair, rc, "pair_set_e1_addr");

	rc = CHR_pair_set_e2_addr(pair, e2Addr, strlen(e2Addr));
	if (rc != CHR_OK)
		show_error(pair, rc, "pair_set_e2_addr");

	rc = CHR_pair_set_protocol(pair, protocol);
	if (rc != CHR_OK)
		show_error(pair, rc, "pair_set_protocol");

	if(qos)
	{
		rc = CHR_api_new_qos_tos_template(CHR_QOS_TEMPLATE_TOS_BIT_MASK,
			qos_name, qos_len, qos);

		if(rc == CHR_VALUE_INVALID)
			CHR_api_modify_qos_tos_template(CHR_QOS_TEMPLATE_TOS_BIT_MASK,
				qos_name, qos_len, qos);

		rc = CHR_pair_set_qos_name(pair, qos_name, qos_len);
		if (rc != CHR_OK)
			show_error(pair, rc, "pair_set_qos_name");
	}
```

```
	rc = CHR_pair_use_script_filename(pair, script, strlen(script));
	if (rc != CHR_OK)
		show_error(pair, rc, "pair_use_script_filename");

	rc = CHR_test_add_pair(test, pair);
	if (rc != CHR_OK)
		show_error(test, rc, "test_add_pair");

	printf("Run the test...\n");
	rc = CHR_test_start(test);
	if (rc != CHR_OK)
		show_error(test, rc, "start_ test");

	printf("Wait for the test to stop...\n");
	isStopped = CHR_FALSE;
	timer = 0;

	while(!isStopped && timer < maxWait)
	{
		rc = CHR_test_query_stop(test, timeout);

		if (rc == CHR_OK)
			isStopped = CHR_TRUE;

		else if (rc == CHR_TIMED_OUT)
		{
			timer += timeout;
			printf("Waiting for test to stop... (%d)\n", timer);
		}

		else
			show_error(test, rc, "test_query_stop");
	}

	if (!isStopped)
		show_error(test, CHR_TIMED_OUT, "test_query_stop");

	printf("Save the test..\n");
	rc = CHR_test_save(test);
	if (rc != CHR_OK)
		show_error(test, rc, "test_save");

		return;
}
```