Konformancia vizsgálati eszközök forgalom-analizátor vizsgálathoz

CSORBA J. MÁTÉ, PALUGYAI SÁNDOR, DR. MISKOLCZI JÁNOS
Ericsson Magyarország Konformancia Laboratórium
mate.csorba@eth.ericsson.se

Reviewed

Kulcsszavak: megbízhatóság, alkalmazási feltételek, optimalizálás

Az IP alapú hálózatok korábban elképzelhetetlen méretekben jelennek mindennapjainkban. Ezzel egyidejűleg a hang, videó és egyéb, ez idáig dedikált hálózatokat használó adatok is egyre inkább az Internet Protokollt alkalmazzák. Mindennek következtében lényeges a hálózatok minőségi paramétereinek állandó figyelemmel kísérése. Hálózati eszközök és komplex távközlési rendszerek fejlesztése közben elengedhetetlen a hálózat teljesítményének, a szolgáltatások minőségének folyamatos nyomon követése. E mellett a hálózatok üzemeltetése, karbantartása és az üzemzavarok gyors elhárítása is igényli a forgalom üzem közbeni megfigyelését. Erre a problémára kívánnak megoldást nyújtani a hálózati forgalom-analizátorok.

1. Előszó

A protokollok szabványhoz való hűségének, vagyis konformanciájának vizsgálatára alkalmazott módszerek korábban nem tették lehetővé hálózati forgalom megfigyelésére kifejlesztett, illetve még fejlesztés alatt álló eszközök vizsgálatát. A kidolgozott módszerrel, az alapvetően konformancia vizsgálatra használt TTCN *(Testing and Test Control Notation)* tesztkörnyezettel lehetővé válik a forgalom-analizátor szoftver működésének ellenőrzése.

Az általunk elkészített rutinok lehetővé teszik az analizátor automatizált és központosított tesztelését. A kész tesztkészlet a vizsgálati módszer moduláris felépítése miatt egyszerűen átalakítható és alkalmazható más forgalom-analizátor megoldások vizsgálatára is.

2. Hálózati forgalom-analizátorok

A hálózati forgalom-analizátor általánosságban a hálózati forgalmat megfigyelő olyan egység, amely egyúttal rekonstruálja és értelmezi a protokollok üzeneteit, azokat is, melyeket alacsonyabb szinten esetleg több csomagban szállíthatunk. A forgalom-analizátorok ősének a korai hálózatfelderítő-alkalmazásokat tekinthetjük, amelyek ICMP üzenetek periodikus küldésével végezték a hálózat topológiájának felderítését, majd az így kapott eredményt ábrázolták valamilyen grafikus formában. A legkorábbi megvalósítások a robusztusságukól híres VAX/VMS rendszereken jelentek meg. Napjainkban a hálózat-felügyeletben nagy szerepet kap a felhasználók aktivitásának megfigyelése, a túlterhelések és az illegális használat megakadályozása, valamint az illetéktelen behatolók felfedése *(intrusion detection)* is.

Egy hálózati menedzsment rendszer több, a vezérlést és a menedzsmentet megvalósító komponensből áll. Általában tartozik hozzá egy, a hálózat felépítését megjelenítő grafikus elem, valamint egy valós idejű megfigyelő és jelentéskészítő eszköz. A legfontosabb funkciói közé tartoznak a konfigurálás, a hibakezelés, a teljesítményt befolyásoló, valamint biztonsági beállítások. Ezen kívül általában rendelkezik valamilyen hálózattervezést segítő eszközzel is.

A forgalom megjeleníthető valós időben vagy utólag, esetleg hisztogram formájában. A hisztogramok rajzolása a leggyakrabban a TCP, UDP csatlakozási pontok (rendszer port) figyelésén alapszik, esetleg külön kitérve az ICMP üzenetek különböző típusaira. A megjelenítés kiterjedhet például adott protokollra vonatkozó meghibásodási százalékra, míg a megjelenítendő információ általában lehet bájt vagy csomag alapú. A hirdetési, valamint csoportcímű *(broadcast, multicast)* csomagok és az elveszett csomagok becsült száma is fontos paraméter. A hibás csomagok esetében elemzésre kerülhet a hiba oka, úgy mint CRC ellenőrző-kód hibák száma, csonkolt csomagok, túlméretezett csomagok, ütközések és a helyes sorrendben bekövetkező hibák száma.

A megfigyelés időtartamának megválasztása is körültekintést igényel. A túl hosszú megfigyelési időtartam képes kiátlagolni bizonyos működési rendellenességeket, így ez feltétlenül kerülendő. Általános esetben az egy órás ciklusok megfelelőek [1]. A kihasználtság és a késleltetés változása feltétlenül nyomon követendő. Hirtelen kiugró értékek általában egy kezdődő hálózati probléma jelei lehetnek, ilyen baljós jelenség lehet a csomagvesztés és a vonali hibák megnövekedése, a késleltetés hullámzása vagy a megszaporodott útválasztási forgalom.

A terhelési profil mérés a hálózat hosszú távú megfigyelését igényli. A hálózat felügyeletét végzők segítségével képet alkothatnak nemcsak az egyes végpontok a hálózatra gyakorolt hatásról, hanem arról is, hogy az egyes felhasználói programok a terhelés hány százalékát okozzák, és ez a terhelés hogyan oszlik el egy hosszabb időintervallum alatt. Hasznos lehet nyomon követni, például az Ethernet vagy más technológián alapuló hálózat kihasználtságának átlag- és csúcsértékét

is. A legtöbb esetben ezeket a paramétereket, vagy egyéb hibajelenségeket figyelő automatikus riasztások beállítása is lehetséges.

A forgalom-analizátorok döntő többsége a valós idejű adatokat az Ethernet kártya *promiscuous* üzemmódba kapcsolásával a helyi hálózatból nyeri. Ebben az üzemmódban a kártya gyakorlatilag megkerüli az Ethernet-címzést, ugyanis beolvassa az összes csomagot a hálózatról, nemcsak a közvetlenül neki címzetteket. Annak érdekében, hogy ez működhessen és megfelelő sebességet produkáljon, általában előre le kell foglalni egy bizonyos részt a memóriából a pufferelés számára.

Az Ethernet-kártya mindent beolvasó üzemmódba kapcsolása a forgalom-analizátort futtató gép működését némiképp lassíthatja, főleg abban az esetben, ha egy általános célú számítógépről van szó és nem egy céleszközről. Egy átlagos PC-s hálózati kártya a teljes mértékben leterhelt hálózatról nem képes minden csomagot beolvasni. Különösen ez a helyzet, ha a csomagok követési ideje rendkívül kicsi *(back-to-back bursts)* [2].

A feldolgozási időt természetesen nagyban befolyásolhatja a megfigyelni kívánt forgalom nagysága. Így megkülönböztetésre szorulnak a valós idejű forgalmat analizáló, illetve a késleltetett *(off-line)* feldolgozást végző eszközök. Létezik különálló, speciális kártyát használó megoldás is forgalom-analizátorra *(EtherMeter)*, bár ez napjainkban már nem túl elterjedt megoldás.

Egy protokoll analizátor segítségével a felhasználó számára lehetővé válik a hálózaton keresztül haladó csomag vizsgálata forrás- és célcím, protokoll, alkalmazás, bitminta, csomagméret s egyéb logikai változó alapján. A legtöbb esetben állítható az analizátor működéséhez szükséges néhány paraméter, például a puffertár mérete vagy a csomagok felszabdalásának lehetősége a jobb memória-kihasználás érdekében. Általában a csomagok megjelenítése során is választhatunk a logikai és a hexadecimális nézet között (például a jól ismert *EtheReal* esetében [3]). Néhány szoftver esetében a hibakeresést támogató külön modulokat is használhatunk, és a csomagok vizsgálatához szűrőket is alkalmazhatunk. Ezekkel kapcsolatban fontos, hogy azok alkalmazhatók-e a valós időben megfigyelt hálózati forgalomra, vagy csak egy előre rögzített adathalmazra.

Az analizátor szoftvereket megkülönböztethetjük abból a szempontból is, hogy a protokoll-rétegeket milyen mélységig képesek dekódolni. Esetleg mind a hét réteget vizsgálhatjuk a segítségükkel, vagy csak egy bizonyos részét. Létezik olyan megvalósítás is, melynek a képességeit a felhasználó is bővítheti a saját maga által írt protokollértelmező modulokkal (kitűnő példa erre a lengyel fejlesztésű *ANASIL* elnevezésű analizátor [4]).

A hálózati forgalom bináris tároláskor fontos a nagypontosságú időbélyegek alkalmazása a rekonstruálhatósághoz, kiváltképp fontos ez hosszú időtartamokat átölelő megfigyeléseknél, ahol a hálózat monitorozása akár több mint 24 órán keresztül is folyhat.

Hasznos lehet az összegyűjtött adathalmaz hordozhatósága, hogy a megfigyelés során összegyűjtött adatokat egy általános táblázat- és/vagy adatbázis-kezelővel, esetleg grafikonszerkesztővel is meg lehessen jeleníteni.

3. A Moniq forgalom-analizátor

3.1. Általános tulajdonságok

Vizsgálatunk tárgyát az Ericsson által fejlesztett, Moniq nevű szoftver képezte, amely egy professzionális, passzív hálózat-analizátor. A forgalmat az IP réteg szintjén vizsgálja, elsősorban a csomagkapcsolt mobil hálózatok szolgáltatásainak minőségét biztosítandó. A fejlesztésekor a mobil adathálózatok intelligens, végponttól-végpontig terjedő teljesítmény menedzselését kívánták megoldani, mivel a mobil hálózatok (GPRS, UMTS) minőségbiztosítása és felügyelete sürgető probléma.

A Moniq használata nem igényel speciális hardvert, akár egy közönséges PC-re telepítve csatlakoztatható a megfigyelni kívánt hálózathoz. A szoftver architektúrája lehetővé teszi a TCP/IP struktúra elemzését statisztikus alapokon. Nagysebességű gerinchálózatok vizsgálatára is alkalmazható. Lehetőség van Gigabites sebességtartományban működő hálózatok megfigyelésére is megfelelő hardver csatlakoztatásával.

A statisztikák létrehozásakor az elsődleges szempont, hogy minél teljesebb képet lehessen kapni a végfelhasználó által érzékelt szolgáltatás-minőségről. Az analizátor számos protokoll üzeneteit képes felismerni és feldolgozni, ide értve a következőket: TCP, UDP, ICMP, DNS, RTP, HTTP, FTP, Telnet, SMTP, POP3, IMAP4, WAP, RADIUS. A statisztikák egyrészt lefedik a hálózat teljesítmény-mutatóit, másrészt a felhasználói szint is megfigyelhető és kiértékelhető. A statisztikákat készítő és analizáló képességek köre folyamatosan bővül, ahogy a fejlesztés halad.

A végponttól-végpontig kitétel ebben az esetben a mobil-terminál és a kiszolgáló közti útvonalat jelenti, vagyis a szolgáltatás-minőséget itt a felhasználó szempontjából elemzi a szoftver. Működése passzív, a méréshez külön forgalmat nem hoz létre, csupán a hálózatot használó előfizetők adatátvitelére koncentrál.

A létrehozott statisztikák finomsága változtatható, akár a csomagszintig. A kapott adatok alapján következtethetünk a forgalom összetételére, és elemezhetjük a hálózatban kialakuló tendenciákat. Felhasználói szintű problémák megoldására szűkíthető a megfigyelt tartomány. Megfigyelési pont több helyen is létesíthető, miközben a statisztikai adatbázist egy helyen tárolják, így könnyen elérhető, és a hálózat teljesítménye, valamint a tendenciák nyomon követhetők. Ezzel a módszerrel megfigyelhető például egy nagy léptékű átkonfigurálás hatása a hálózatra.

3.2. A szoftver felépítése

A szoftver igazi erőssége a hálózat teljesítménymutatóinak teljes körű felmérésében, az ok-okozati összefüggések felderítésében, a hálózat tervezés, üzemel-

tetés támogatásában rejlik. Ehhez többféle statisztikát készít, példaképpen említve néhányat: a forgalom eloszlása protokollok szerint, hálózati elakadást jelző üzenetek (pl. *ICMP unreachable)* aránya, tranzakciók száma és időbeli eloszlása, csomagok méretének eloszlása és így tovább.

Az alkalmazás moduláris jellegéből adódóan több részből épül fel. A legfontosabbnak tekinthető részek a következők voltak:

- *Moniqdump* – a forgalom rögzítéséhez;
- *Moniqparse* – a csomagok elemzéséhez;
- *ReadBin* – az eredmények ember által olvasható formába öntéséhez.

A működés első lépcsőjeként létrejön az úgynevezett forgalom *(trace)* állomány, amely tartalmazza a megfigyelés alatt a hálózaton áthaladt csomagokat. Ennek az állománynak több megjelenési formáját is alkalmazhatjuk. A legegyszerűbb, ha a Unix alapú operációs rendszerek részét képező *tcpdump* programot használva készítjük el ezt az állományt. Ennél kifinomultabb megoldást jelent a szoftver részét képező *Moniqdump* program használata, amely a forgalom állomány elkészítése közben titkosítja az előfizetői adatokat és belső IP címeket, úgy, hogy ez az adatok későbbi feldolgozását nem befolyásolja. A *Moniqdump* bemenete lehet a működő hálózat, de képes egy előre elkészített forgalom állományt is átalakítani. Annak érdekében, hogy egy hosszabb megfigyelést követően is kezelhető maradjon az állomány mérete, a csomagok fejrészét követő adattartalom tetszőleges mértékben leválasztható, azaz nem szükséges teljes csomagokat eltárolni, a későbbi kiértékelést ez nem befolyásolja. Az adattartalom leválasztásának helye azért kell, hogy változtatható legyen, mivel különböző protokollok, bizonyos esetekben további hosszú fejléceket helyezhetnek el a hálózati réteg adatait követően. (Kitűnő példa erre a WAP protokoll, mely az UDP fejléc után következő adatrészbe helyezi el fejléc-információit, időnként a többi IP protokolltól eltérően rendkívül hosszan.)

A tárolt forgalmat feldolgozható formába a *Moniqparse* program alakítja, melynek a kimenete több bináris napló állomány. Tartalmuk röviden:

- Státusz állomány;
  Valós idejű mérésnél a mérésre vonatkozó adatokat tárolja, mint például időbélyegek, aktuális időbeli felbontás.
- Rövid távú globális statisztika;
  Tartalma a halmozódó forgalmi adatok kis részletességgel és nagy időléptékben.
- Hosszú távú globális statisztika;
  Ebben az állományban nagyon részletesen, de kis időbeli felbontással tárolódnak az adatok.
- Tranzakciók naplója;
  Minden kliens-szerver közti adatátvitelt tartalmaz. Új bejegyzés akkor kerül bele, amikor egy tranzakció lezárul.
- Felhasználói kapcsolatokat tároló napló állomány;
  Bejegyzést tartalmaz minden lezárult felhasználói kapcsolatról.
- Egy perces felhasználói kapcsolat napló;
  A felhasználók által érzékelt átviteli kapacitás becslésére szolgáló statisztikákat tartalmaz.
- Tranzakció számláló napló állomány;
  Tartalma az egy időegységre eső tranzakciók száma.
- Felhasználói kapcsolatokat számláló állomány;
  Hasonlóan az előző állományhoz, az időegységre eső kapcsolatok számát naplózza.

A *Moniqparse* által előállított állományokból program modulok segítségével kinyert különböző statisztikák jeleníthetők meg egy kliens-szerver kapcsolaton keresztül. A felhasználó gépén futó grafikus felhasználói felületen (GUI) láthatók a különböző statisztikai elemzések eredményei. (A grafikus felületen keresztüli felhasználást a továbbiakban nem részletezzük, mivel annak tesztelése nem tartozott a feladataink közé.)

Az adatok kinyerésének másik módja a *ReadBin* program használata, melynek segítségével a bináris állományokból lekérdezéseket hajthatunk végre, amelyek eredményét szöveges kimenetként kapjuk. A program kimenetét parancssori kapcsolók használatával formálhatjuk. Így lehetséges szűrési feltételek beállítása, valamint azt is szabályozhatjuk, hogy az adatbázis mely mezőire vagyunk kíváncsiak.

4. A TTCN-3 nyelv

A TTCN-3 egy konformancia vizsgálatokra általánosan használt, magasszintű programozási nyelv (valójában nincs korlátozva kizárólag konformancia tesztelésre). Vizsgálható a segítségével együttműködési képesség, robosztusság, végezhető rendszer- és integrált teszt. A nyelv általában tesztelési metódusoktól és protokolloktól független tesztkészletek specifikálására hivatott. Mindezen tulajdonságok mellett alkalmas távvezérelt tesztek lebonyolítására is, amelyekben az IUT irányítása is TTCN program segítségével történik. A felhasználás más egyéb jellemző területei: a szolgáltatások tesztelése, CORBA alapú platformok, API-k tesztelése [5].

A TTCN-3 fordító protokoll független C++ forráskódot generál. Vagyis szükség van valamilyen kiegészítésre, ami megteremti a kapcsolatot a végrehajtható tesztkészlet és a tesztelendő között. Ez a tesztport, ami egy C++ nyelven írt szoftver-könyvtár. A teszt-csatoló rutin (tesztport) egy adott protokoll üzeneteinek, csomagjainak kezeléséhez szükséges, így egy adott protokoll minden verziójához külön meg kell írni. Ehhez nyújt némi segítséget, hogy a TTCN-3 fordító segítségével elő lehet állítani a tesztport alapját képező, C++ sablont, amely definiálja a szükséges függvényeket, azonban a végleges kódot a felhasználónak kell megírnia. A tesztport gyakorlatilag egy végtelen FIFO sorral modellezhető, amely egész addig tárolja a beérkező üzeneteket, míg a TTCN komponens, melyhez tartozik, ki nem olvassa azokat. Természetesen egy TTCN komponens több tesztport felett is rendelkezhet, ezt a tesztkörnyezet ki is használja [6].

A hálózatba kapcsolt tesztelést végző számítógépek futtatás alatti viselkedése tesztesetek (*Test Case*) formájában kerül kifejezésre. A nyelv hatékonyan tudja kezelni a különböző viselkedési alternatívákat, úgy, mint különböző adatok fogadása a tesztportokon keresztül, időzítő események bekövetkezése stb. A tesztesetekben kerül sor az eredményt jelentő ítéletek meghozatalára, ezek mellett a sokszor rendkívül hasznos naplóállomány készítésre is lehetőség van.

5. A Moniq vizsgálata

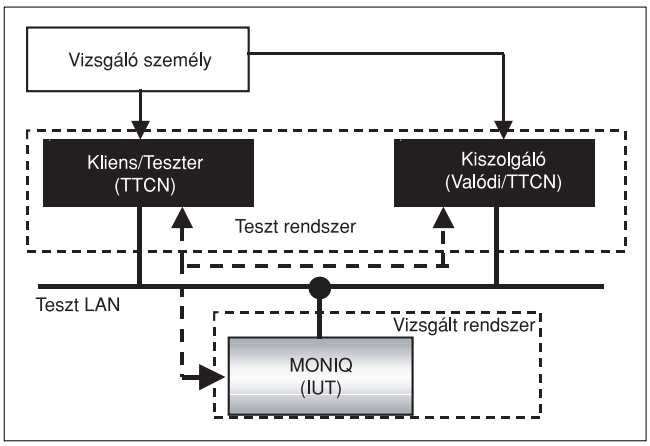
A Moniq vizsgálatához a konformancia vizsgálatok körében hagyományosnak tekinthető távoli tesztelési elrendezést alkalmaztuk, némileg módosítva azt. A teszter oldal a konformancia vizsgálati elrendezéseknek megfelelően mindig egy TTCN nyelvű program volt. Azonban a vizsgált rendszer oldalán a legtöbbször szintén egy TTCN komponens állt vagy pedig egy valódi kiszolgáló szoftver, míg az IUT a forgalom szempontjából passzív hálózati analizátor volt *(1. ábra)*.

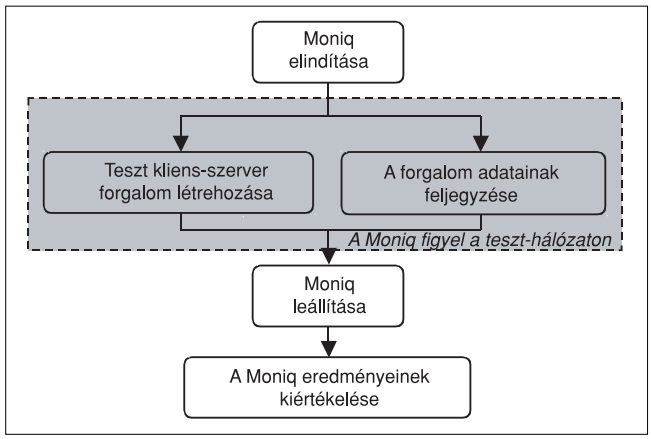
A kliens és a kiszolgáló megvalósítása nagyban hasonlít egy konformancia teszt kifejlesztéséhez, mivel a közöttük lejátszódó kommunikációt a vonatkozó ajánlások (RFC-k) alapján kell elkészíteni. Mindemellett szükség van az ajánlástól eltérő, hibás vizsgálati sorozatok előállítására is (hasonlóan a konformancia vizsgálatokhoz).

A szimulációra a kiszolgáló TTCN segítségével azért volt szükség, mert helyes és helytelen vizsgálati sorozatokat egyaránt elő kellett állítani (mivel az IUT-nek mindkét esetben helyes analízissel kell szolgálnia). A fentiekből látható, hogy itt egy speciális konformancia vizsgálati módszerrel állunk szemben: a teszter úgy viselkedik, mintha a kiszolgálót vizsgálná helyes és helytelen üzenetekkel, s ez alatt a Moniq által szolgáltatott forgalmi adatokat ellenőrzi.

A cél egy olyan minősítő rendszer kialakítása volt, amely a valódi használat körülményeit szimulálva, automatizáltan teszi lehetővé a Moniq legfontosabb moduljainak (*Moniqdump, Moniqparse, ReadBin*) ellenőrzését. A vizsgálat megvalósításakor tehát alapvetően két funkciót kellett megkülönböztetnünk egymástól. Egyrészt ki kellett alakítanunk egy környezetet, melynek segítségével a Moniq automatikusan vezérelhetővé vált, mintha egy operátor ténylegesen használná [7]. Másrészt létre kellett hoznunk egy speciális forgalmat a hálózaton, amely a szabványoknak megfelelően szimulálta a vizsgált protokoll működését. Ezek után rendelkezésünkre állt egy jól definiált forgalom egy szeparált hálózaton, tehát a tesztek befejező lépése a kiértékelés kellett legyen.

1. ábra A tesztrendszer és az IUT viszonya





2. ábra A tesztprogram működése

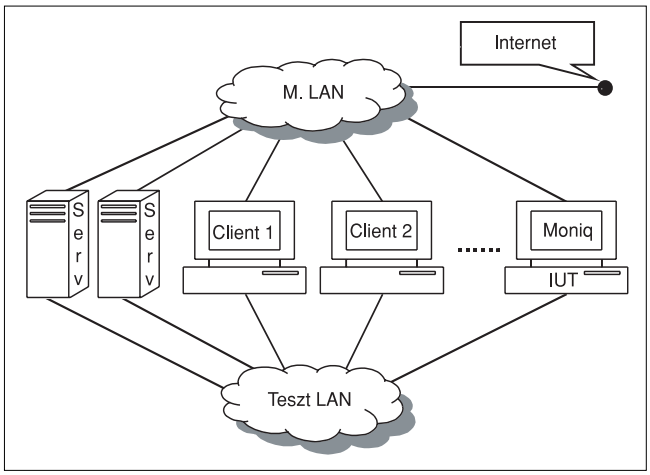
A kiértékelés során annak tudatában, hogy mi zajlott le a hálózaton, meg kellett nézni, hogy a Moniq helyesen értékelte-e a hálózaton történteket. Ezeket a lépéseket láthatjuk a *2. ábrán,* amely magába foglalja a Moniq elindítását a tesztforgalom elemzéséhez, a Moniq leállítását, a mért eredmények tárolását, illetve annak kiértékelését, valamint a mérés folyamatáról egy dátummal és a Moniq-ra vonatkozó azonosító információkkal rendelkező adatállomány előállítását.

A teszteket tesztcélok formájában dokumentáltuk, szabványos alakban megadva a teszt nevét, egy rövid leírást, utalva az adatbázis mezejére, amit vizsgálunk és egy bővebb leírást a működésről. A dokumentálást elősegítendő minden egyes teszt futása után három állományt tárol a tesztrendszer: egyrészt a TTCN program futásakor keletkező naplót (amely a mérés során bekövetkező összes eseményről tartalmaz bejegyzést), másrészt a Moniq által felvett, hálózati forgalmat tároló bináris állományt, valamint a Moniq szöveges kimenetét.

6. A rendszer vizsgálata

6.1. Vizsgálati környezet

A forgalom-analizátor vizsgálatának alapját a támogatott protokollok működésének szimulációja képezte. Mivel a vizsgált protokollok működése kliens-kiszolgáló elrendezésben folyik, ki kellett alakítani néhány kiszolgálót és a felhasználói oldalt szimuláló kliens gépeket. Az első lépés tehát a vizsgáló hálózat kiépítése, majd azt követően a megfelelő szoftverek telepítése.



3. ábra A teszthálózat

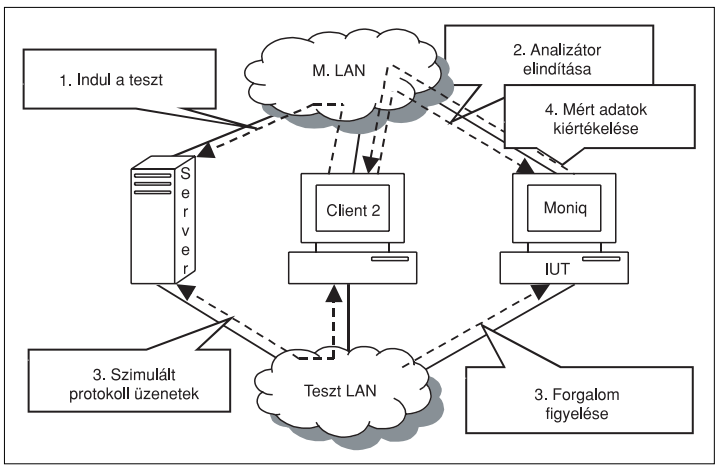
Az *3. ábrán* láthatóan két alhálózatot alakítottunk ki, melyek egymástól elkülönítve működtek. Szükség volt egy könnyen kézben tartható forgalmú ellenőrző-alhálózatra, melyen minden automatikus és/vagy felesleges kommunikációt letiltottunk, hogy kizárólag a tesztek által előállított csomagok használhassák.

A második, menedzsment hálózat *(M. LAN)* a tesztek működéséhez szükséges vezérlő üzenetek és a munkához szükséges egyéb forgalom lebonyolítására szolgált. A Moniqot vezérlése teljes egészében a hálózaton keresztül történt, vagyis tulajdonképpen a Moniq-ot kezelő felhasználó parancsait is egy TTCN programrészlet valósította meg.

6.2. A Moniq forgalom-analizátor vizsgálata

Méréseink megvalósítása a rendelkezésre álló hálózaton a *4. ábrán* látható. A példában a teszt(ek) futásának koordinálását és kiértékelését a *Client2* névvel jelzett gép végzi. A teszt indításakor jelez a kiszolgáló oldalt megvalósító *Server2* gépnek, majd elindítja a vizsgált forgalom-analizátort. Ezt követően az IUT már folyamatosan monitorozza a *Teszt LAN*-t, miközben a vizsgáló program elkezd egy adott protokollnak megfelelően kommunikálni a kiszolgálóval (például végrehajt egy FTP letöltést vagy egy levélküldést az SMTP protokollnak megfelelően). Fontos, hogy minden vezérlési információ az elkülönített menedzsment hálózaton bonyolódik, így a forgalom-analizátor ezeket a csomagokat nem érzékeli. Amikor a kívánt működés szimulációja véget ér, az IUT által mért adatok begyűjtése és kiértékelése szintén automatikusan történik, ismét az *M. LAN* igénybevételével.

4. ábra A Moniq tesztelési lépéseinek megvalósítása



6.3. Kommunikáció a csatlakozási pontokon keresztül

A TTCN-3 kódnak szüksége van egy C++ nyelven írt csatoló rutinra (port-ra), amely lehetővé teszi számára a kommunikációt a vizsgálandó objektummal. A mérési elrendezésünkben három különböző típusú csatoló rutin fordult elő.

A STORE csatoló rutin

Szükség volt először is egy speciális csatoló rutinra a mért adatok átmeneti tárolásához és kiértékeléséhez. Ez a csatoló rutin a *STORE_Port* nevet kapta. A feladatai közé tartozott a Moniq által mért adatok és a számított, tehát az elvileg helyes adatok adatbázisszerű tárolása egy-egy teszteset futása során. A csatoló rutin ezt az igen egyszerű adatbázist a memóriában tárolta, tehát az csak a teszt futása során volt hozzáférhető, ami elegendő is volt, hiszen csupán a tesztek kiértékelésénél volt rá szükség. Az adatbázis felépítése az *1. táblázatban* látható.

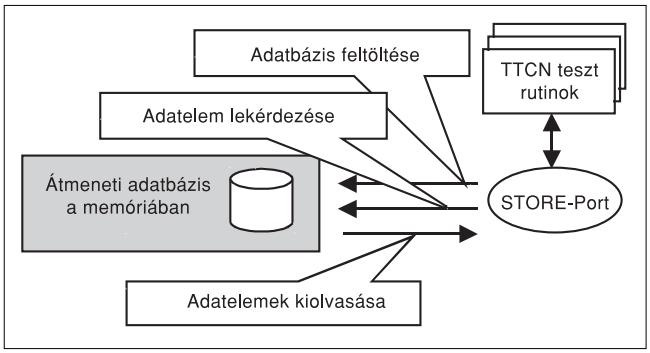
Mező neve	Tartalma
SType	Napló állomány típusa
TNO	Tranzakció sorszáma
Name	Vizsgált adatbázis mező neve
Data	A tesztelt mező várt értéke
MoniqData	A mező Moniq által mért értéke

1. táblázat A mérés kiértékeléséhez használt adatbázis rekordjainak felépítése

A csatoló rutinon keresztül ugyanúgy lehetséges volt az üzenetküldés és -fogadás, mint egy hagyományos kommunikációs ponton keresztül. Azzal a különbséggel, hogy az üzenetek nem a hálózatra kerültek ki, hanem a memóriában található adatbázissal lehetett kommunikálni a csatlakozáson keresztül. Az adatbázis elemeinek három lehetséges típusa volt:

- egész szám, különböző adatok, például csomagok számlálására;
- lebegőpontos szám, jellegzetesen idő (másodperc alapú) mérésére;
- szöveg, általában státuszinformációk tárolására.

Egy adat lekérése tehát a következő információk megadásával volt lehetséges: tranzakció sorszáma (TNO), ezzel például egy TCP kapcsolatot lehetett kiválasztani a monitor által felvett több kapcsolat közül. Egy kapcsolathoz azonban az analizátor több napló állományt is készíthe-



5. ábra A STORE-port működése

tett, tehát ki kell választani az adott kapcsolathoz tartozó, a számunkra érdekes mezőt tároló naplót (*SType*). Valamint természetesen meg kell adnunk, hogy név szerint mely mezőre vagyunk kíváncsiak (Name). A lekérdezés eredményeként a *STORE_Port* két értéket ad vissza: a tesztprogram által helyesnek ítélt *Data* mezőt és a Moniq által mért *(Moniqdata)* értéket. Ezt a két értéket ezek után könnyedén össze lehet hasonlítani.

A Telnet csatoló rutin

Ez a rutin (port) több helyen is rendkívül fontos szerepet tölt be. Az általunk írt TTCN függvény-gyűjtemény e kommunikációs port használatával csatlakozik a forgalom-analizátort futtató számítógéphez, és szabványos Telnet kapcsolaton keresztül, egy valódi felhasználót szimulálva parancsokat ad ki a gépnek, és értelmezi a válaszokat.

A másik fontos felhasználása a vizsgáló sorozatok előállításában, vagyis a protokollok működésének szimulálásában volt. Tekintve, hogy a POP3, IMAP és SMTP protokollok nem alkalmaznak külön saját csomagformátumot, hanem egy TCP kapcsolat felett, karakteres alapon működnek.

A TCP csatoló rutin

A Telnet port-nál is alkalmazott stream-csatoló *(stream socket)* egy alternatívája az úgynevezett datagram-csatoló *(datagram socket)*, amely UDP kapcsolatok létrehozására használható, vagyis nem megbízható és nem kapcsolat-orientált átvitelre.

Bizonyos teszteknél azonban nem alkalmazható sem a *stream*, sem a *datagram* csatoló *(socket)*. Ezekben az esetekben a forgalom előállításakor az alacsonyabb rétegben lévő protokollokat is befolyásolni szeretnénk. Erre például akkor lehet szükség, amikor olyan üzeneteket szeretnénk előállítani, amelyeket a rendelkezésre álló kiszolgáló szoftverből csak nehezen vagy egyáltalán nem tudunk kisajtolni.

Egy konkrét példát említve: szükség volt a tesztelés során olyan tesztesetre, mely különböző működési fázisokba juttatja az IMAP kiszolgálót, és megvizsgálja, hogy a Moniq megfelelően ismeri-e fel a kiszolgáló állapotát. Azt az állapotot azonban, amikor az IMAP kiszolgáló már nem tud több kapcsolatot fogadni, mert telített, és emiatt rögtön a TCP kapcsolat felépítése után visszautasítja a klienst, a rendelkezésre álló erőforrásokkal nehéz lett volna megvalósítani. A megoldást az jelentette, hogy a protokoll működését alacsonyabb szinten kellett modellezni.

Valamint nem lehetett használni a valódi kiszolgáló szoftvert, így a működését egy TTCN programmal kellett szimulálni. Ezen tényezők miatt szükség volt az úgynevezett *raw socket* használatára, az ezt használó port pedig a TCP port nevet kapta. A tesztek között voltak olyan alacsony szintű mérések is, mint például egy adott protokollhoz tartozó átvitt bájtok összege vagy csomagok száma, amelyekhez elkerülhetetlenül egy alacsonyabb szinten dolgozó port-ot kellett használni.

A raw-csatolót *(raw socket-et)* használó kommunikációs port esetében a teljes csomag összeállításáról nekünk kell gondoskodni, vagyis minden protokoll-rétegre a fizikai réteg felett oda kell figyelni. Tehát ezeknél a teszteknél egy csomag a következőképpen nézett ki: <Ethernet fejrész> <IP fejrész> <TCP fejrész> <TCP adatrész>. A *raw socket* használatának hátránya, hogy az így modellezett kapcsolatok esetében a szállítási protokoll működését is teljes egészében meg kell valósítani TTCN-ből, kezdve a kapcsolat-felépítéstől a bezárásáig, a TCP szabványnak megfelelően.

6.4. A Moniq tesztelését végző TTCN rutinok

A Moniq szoftver vezérlése és méréseinek kiértékelése egységesen történt. A Moniq-ot kezelő segédfüggvényeken kívül a legfőbb funkciókat a következő négy elem tartalmazta.

A moniq_start függvény

A függvény meghívása után bejelentkezik a tesztgépről az IUT-re (Moniq) és ellenőrzi, hogy fut-e a vizsgált forgalom-analizátor valamelyik részegysége. Erre azért van szükség, mivel a szoftver ellenőrzését egyszerre többen is végezhetik, ugyanezt a függvénykönyvtárat *(MoniqFunctions)* használva. Azonban a mérések tisztaságának érdekében a *moniq_start* függvény egyszerre csak egy virtuális felhasználót engedélyez az IUT-n. Abban az esetben, ha már valaki más is használja a forgalom-analizátort, a teszt futása félbeszakad.

Amint a szükséges erőforrások rendelkezésre állnak, a függvény elindítja a hálózat monitorozását végző *moniqdump* programot, de hibakereséshez segítségül tudja hívni egyúttal a Linux disztribúció részét képező *tcpdump* programot is.

A moniq_end függvény

A *moniq_end* függvény meghívására akkor kerülhet sor, ha a vizsgáló sorozatot már kiküldtük a hálózatra, vagyis a forgalom-analizátornak már nem szükséges figyelnie a hálózatot. Ekkor a függvény leállítja Telnet kapcsolaton keresztül a monitorozást, és meghívja a csomagok elemzését végző *moniqparse* programot, figyelembe véve a felhasználó által használt konfigurációs állományt.

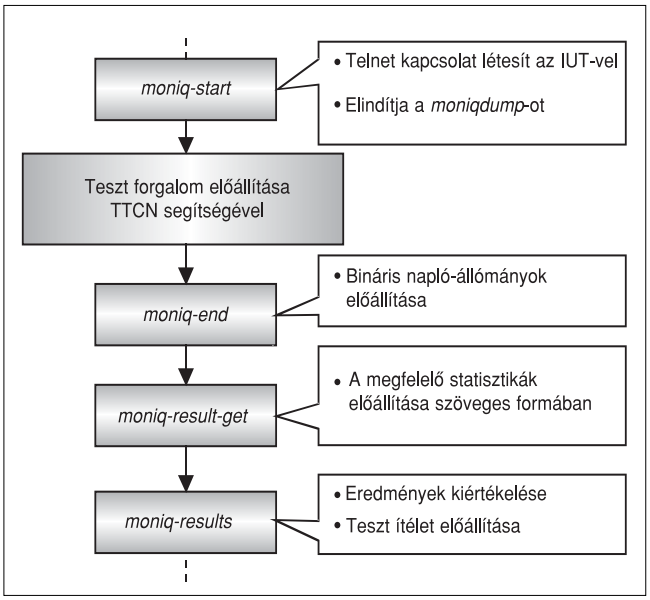
A moniq_result_get függvény

Ez a komponens a *ReadBin* program meghívásával teszi lehetővé a *moniqparse* segítségével létrehozott bináris napló állományok átalakítását. Természetesen ebben az esetben is a tesztelést végző pontosan beállíthatja a *ReadBin* program paramétereit. Az átalakítás eredményeként előálló szöveges kimenetet a függvény a *STORE_Port*-on keresztül tárolja a kiértékelés idejére, valamint szöveges állományként is menti egy megadott könyvtárba a központi kiszolgálón. Ezen kívül a hálózat monitorozása során létrejött bináris állományt is ugyanazon könyvtárba tárolja.

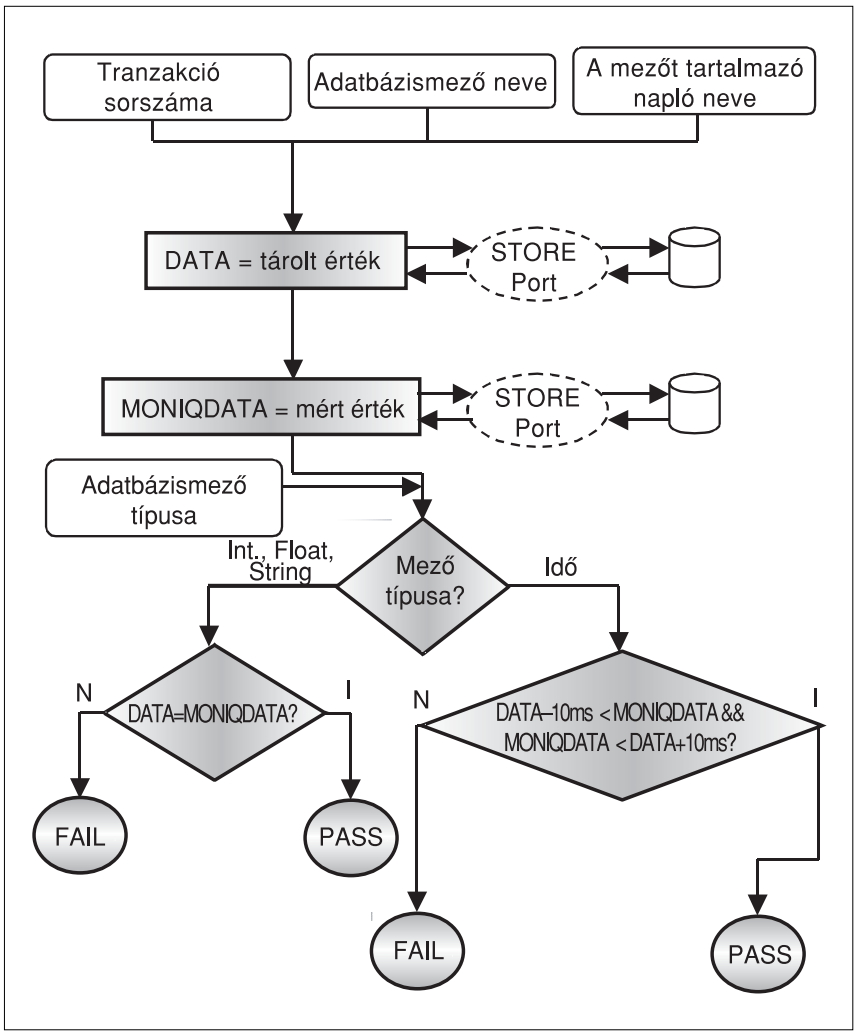
A moniq_results függvény

Amint a *moniq_result_get* függvény tárolta szöveges formában az adott teszt szempontjából érdekes napló állományt, az eredményeket a *moniq_results* segítségével értékeli.

A függvény két értéket kérdez le a *STORE_Port*-on tárolt adatbázisból. A kiolvasott két érték a tesztelő program által helyesnek ítélt és a Moniq által mért adat. A függvény bemeneti paraméterei a következők: a vizsgált adatbázismező neve, a mezőt tartalmazó statisztika neve, egy tranzakció sorszám és az adatbázismező típusa, mely négyféle lehet. A teszt ezek után átment *(pass)* ítélettel zárul, ha a kapott két érték megegyezik. Amennyiben az adat típusa egész szám, valós szám vagy szöveg, teljes egyezőséget vizsgál, ha a típusa idő, 10 ms-os pontossággal értékel ki a program.



7. ábra A Moniq-ot kezelő függvények használata



6. ábra A moniq_results függvény működése

7. A mérések értékelése

A mérési elrendezés kidolgozásánál fontos szempont, hogy a vizsgálati módszer rugalmas, könnyen átalakítható legyen. Annál is inkább, mivel a Moniq forgalom-analizátor szoftver fejlesztése a teszteléssel egyidejűleg folyt, így többször szükség volt a tesztelési eljárás kis mértékű átalakítására. Éppen ezért a forgalom-analizátor kezelését végző, tehát a Moniq-specifikus rutinokat jól elkülöníthetően kellett kialakítani.

Ennek a moduláris felépítésnek köszönhetően egy másik forgalom-analizátor megvalósítás tesztelése sem igényelne óriási beavatkozást a tesztrendszerbe, mivel csak a Moniq specifikus részek cseréjére lenne szükség. A teszteléshez használt protokollok körét a modularitásból adódóan úgyszintén könnyedén lehet bővíteni.

A kidolgozott módszer működéséből látható, hogy a felhasznált konformancia vizsgálati eszközök rugalmasságuknak köszönhetően más, a konformancia vizsgálatoktól eltérő feladatok megoldására is alkalmazhatók. A TTCN-3 nyelv használatával olyan vizsgálatok is elvégezhetőek,

melyek más módszerrel nem lehetségesek, például logikailag hibás vizsgálati sorozatok előállítása vagy adott értékű válaszidők szimulálása.

A teljes tesztkészlet futtatására a vizsgált szoftver minden újabb verziójának *(build)* létrejöttekor lehetőség van. Ekkor az összes teszteset emberi beavatkozás nélküli futtatása után a tesztek által készített naplóállományokból kiválaszthatók a hibás *(fail)* ítélettel zárult tesztek. Ezután az ilyen naplók részletes elemzésével megtudható, hogy a kiértékelés során melyik adatbázis mező tartalmazott hibás értéket, illetve következtethetünk a hiba okára is.

Amint a vizsgálat során egy hibára fény derül, az azonnal hibajelentésként *(Trouble Report)* továbbítható a fejlesztőkhöz, és a hiba akár már a következő fejlesztői változatban kiküszöbölhető.

Irodalom

[1] Stine, R.: FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices [RFC1147]
[2] Bradner, S., McQuiad, J.: Benchmarking Methodology for Network Interconnect Devices [RFC 2544]
[3] A GNU/GPL értelmében ingyen hozzáférhető Ethereal programcsomag weboldala: http://www.ethereal.com/
[4] A lengyel A plus C Ltd. által fejlesztett Anasil programcsomag weboldala: http://www.anasil.net/
[5] HTE Online könyv: Távközlő hálózatok és informatikai szolgáltatások
[6] Szabó, J.Z. (Ericsson Mo. Konformancia Laboratórium): User Documentation for the TTCN-3 Test Executor
[7] Agilent Technologies: Test Automation for Network Routing Devices [Technical Report ; http://www.agilent.com]

Hírek

Az **Oracle a kaliforniai San Diegoban zajló Apps World konferenciáján** több újdonságot jelentett be termékeiről, az alkalmazásfejlesztés legújabb irányairól és várható fejleményeiről.
Az Oracle betekintést nyújtott új üzleti alkalmazásegyüttesének, az Oracle® E-Business Suite 11i.10 jellemzőibe. A nemsokára megjelenő új változatban komoly továbbfejlesztéseket hajtottak verge az integrációs rétegben, és jelentősen bővült az ágazatspecifikus üzleti folyamatokat támogató funkciók köre. Az Oracle alkalmazásait használó szervezetek több mint 85 százaléka jelenleg az Oracle E-Business Suite 11i verziót használja, így az ügyfelek többségének nem okoz majd gondot az áttérés a 11i.10-re. Az új verzió az Open Applications Group (OAG) által definiált nyílt szabványú interfészeket is támogatja, amelyek egységes szabványokat biztosítanak az üzleti alkalmazások integrálásához. A 11i.10 változat több, mint 150 szabványos OAG üzleti objektumot támogat, amelyek például rögzítik, hogyan kell definiálni egy beszerzési megrendelést.
Egy másik újdonsága az integrációs interfészek katalógusa, amely az Oracle E-Business Suite közzétett API-jait írja le. Emellett az Oracle Application Server 10g képességeit is kihasználja az integrációhoz más fejlesztők alkalmazásaival és az üzleti partnerekkel.
Továbbfejlesztett automatizálási és felügyeleti szolgáltatások az E-Business Suite Outsourcing-ban. A továbbfejlesztések megkönnyítik a rendszeradminisztrációt, automatizálják a szoftverfelügyelet egyes fontos folyamatait, proaktív rendszermonitoringot biztosítanak, és csökkentik a karbantartási költségeket.
Az ügyfél változó üzleti követelményeinek rugalmas kiszolgálásához az Oracle egy- vagy többéves szerződéseket kínál 30 napos felmondással; az ügyfél megválaszthatja, hol üzemeljen a hardver, és a kihelyezéses szolgáltatásokat az Oracle kínálatában szereplő bármely termékhez igénybe veheti.

A **Linux World Konferencián bejelentett újdonságok** három fő csoportba sorolhatók.
Az első az új generációs asztali technológiák, amelynek keretében megjelenik a Sun Java Desktop System új verzója Linuxon. Ebben bővülnek a szoftver felügyeleti funkciói, valamint megjelenik egy új háromdimenziós, Java alapú PC desktopfelület.
A második csoportba a vállalati szoftverek és hardverek sorolhatók: a Java Enterprise System, integrált infrastruktúraszoftver-megoldás, és támogatni fogja a Linux operációs rendszert Intel Xeon rendszereken és az AMD Opteron processzor alapú x86 szervereken is.
A harmadik csoportot a Linuxos fejlesztőeszközök képezik: a Sun bemutatott egy asztali megoldást, amely a Sun új Java Studio Creatorát, egy egyszerűen használható Java-alkalmazáskészítőt. A Sun a tervek szerint 2004 végéig fejlesztőeszközeinek teljes sorát megjelenteti Linuxra is.