Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai tanszék

Róth Ádám Gábor

Erőforrás-hatékony TCP-stack tervezése és megvalósítása FPGA alapú eszközre

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

KONZULENS

Dr. Varga Pál

BUDAPEST, 2016

Tartalomjegyzék

Kivonat

Az alkalmazás-rétegbeli funkciókkal ellátott maghálózatok (pl. az LTE Evolved Packet Core) fejlesztéséhez és üzemeltetéséhez is fontos tudnunk, hogy azok eszközei milyen mennyiségű forgalmat képesek biztonságosan kezelni. Ezért a funkcionális és integrációs tesztek mellett az egyik legfontosabb vizsgálat a terheléses, vagy tömeg-tesztelés.

Napjaink legnagyobb volumenű internetes forgalmát – beleértve a mobil adatforgalmat is – a Facebook és Youtube alkalmazások generálják. Ezen felhasználói szokások kialakulását a szolgáltatás mögötti minőségbiztosítás (QoS) és a felhasználói élmény (QoE) erősítette meg. A megbízható adatátvitelt nyújtó TCP [1] protokoll jelen esetben nagyban támogatja az ezen szolgáltatások nyújtásához szükséges transzport-funkciókat. A hatékony tömeg-tesztelőnek tehát TCP alapú forgalmat kell tudnia generálni és kezelni. A párhuzamos TCP-folyamok állapotának követése azonban nagyon erőforrás-igényes feladat: ez esetben legalább tízezres mennyiségű párhuzamos folyamot kell tudni kézben tartani, nagy sebességgel. Ehhez hardveres támogatásra van szükség: az FPGA-alapú (*Field Programmable Gate Array*) [2], alacsony szinten programozható hálózati eszközök hasonló alkalmazásokban már bizonyítottak.

A nagy hálózati sebesség melletti magas párhuzamosítási igény miatt a hardveres elérésű protokoll megvalósítás a célszerű, mivel ez késleltetés-mentes, és megbízható adatátvitelt is képes biztosítani. A feladat tehát egy olyan TCP forgalom-generátor modul megtervezése és megvalósítása, amely képes egy FPGA-n belül, sok példányban működni. Sajnálatos módon a TCP-kapcsolat létrehozásához és fenntartásához minden szempontból viszonylag nagy erőforrás-igény tartozik. Ennek oka a sorrendhelyes és csomagvesztés-mentes adatátvitel igénye. Ez komoly nehézségeket okozhat nagy mennyiségű párhuzamos kapcsolat forgalmának generálásakor.

Dolgozatomban egy XILINX VIRTEX-5 [3] családba tartozó FPGA-ra fejlesztett TCP implementációt mutatok be. A munka során nagyon fontos volt, hogy a TCP szabvány [1] által meghatározott legfontosabb funkciók le legyenek fedve. Értelemszerűen, a kezdeti kihívást a „felesleges” funkciók megtalálása és elhagyása nyújtotta. Ennek eredményeképp egy, a fenti célhoz illesztett, lecsupaszított TCP-variánst definiáltam, terveztem, és valósítottam meg.

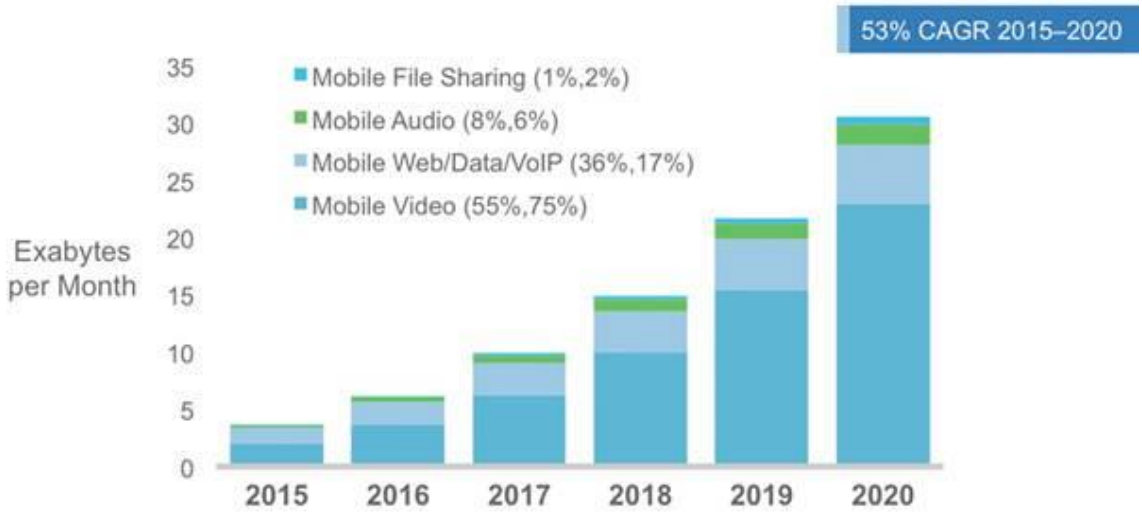
A hardveresen gyorsított TCP-megvalósítást is biztosító terhelés-tesztelőt a mobil gerinchálózatokon túl más területeken is használhatjuk. Egy ilyen eszköz az adatközpontoktól kezdve a nemzetközi gerinchálózati átadó-központokig számos helyen segítheti az operátorok munkáját: mindenütt, ahol nagy mennyiségű, nagy sebességű forgalom szimulálására van szükség.

1 Bevezetés

A XXI. század egyik legjobban fejlődő iparágai közé tartozik az infokommunikációs technológiai terület – ezen belül is a mobil eszközök és az általuk a mobil interneten keresztül elérhető szolgáltatások fejlődnek leginkább. A hatalmas fejlődés hátterében az internetes tartalom szolgáltatásaira való felhasználói igény és az általa generált robbanásszerű technológiai fejlődés áll.

Napjaink „átlagembere" bárhol, bármikor és a lehető leggyorsabban szeretné elérni az internetes tartalmakat, szolgáltatásokat. Ezen jelenlegi felhasználói igények megfelelő kiszolgálásához a mobil internet szolgáltatóknak maximális hatékonyságot kell elérni a hálózataik üzemeltetésénél és karbantartásánál. Ehhez az egyik – ha nem a legfontosabb – alapkövetelmény a hálózatok funkcionális viselkedésének átfogó, részletes ismerete, a másik alapkövetelmény pedig a hálózati elemek teherbírásának részletes ismerete.

Egy, a hálózat túlterheléséből fakadó nagy kiterjedésű üzemszünet akár komoly hitelvesztéssel járhat. Ez hosszútávon a céget kényelmetlenül érintő felhasználó elpártolást okozhat. Ezért is nélkülözhetetlen egy, a konkurensekkel éles versenyt vívó vállalat számára a hálózati elemeken, valamint a hálózat egészén (végponttól végpontig) alkalmazott tömeg-tesztek elvégzése. A terheléses tesztek elvégzéséhez elkerülhetetlen egy jól megtervezett és kivitelezett, a kívánt forgalom generálására képes eszköz. Egy jól konstruált forgalomgenerátor tervezésénél több szempontot is figyelembe kell venni. Ehhez a mobil internetes forgalmak megfigyelése, felhasználói trendek keresése elengedhetetlen. A Cisco VNI Mobile [5] kimutatása alapján elmondhatjuk, hogy a 2015-ös évben a mobil adatforgalmak jelentős, 55%-os részét video tartalmak alkották. A Cisco által kiadott előrejelzések alapján nem csak a teljes mobil adatforgalom mennyisége, de azon belül a video tartalmak is növekedő tendenciát mutatnak. Ezt a növekedést az 1. ábrán láthatjuk, miszerint 2020-ra a video tartalmak 75%-os részt tesznek ki a 30.6 Exabájtra becsült adatforgalomból.



1. ábra – A forgalmi típusok mértékének előrejelzése a Cisco szerint 1

Az imént említett és további más kimutatások [6] és becslések [7] alapján joggal felismerhetjük, hogy napjaink jelentős mobil – és általános internetes – adatforgalmát a Youtube és Facebook alkalmazások generálják. Ezeknek a népszerűségét az általuk nyújtott szolgáltatásra való igény mellett a minőségbiztosítás (QoS) és felhasználói élmény (QoE) tette egyeduralkodóvá az internetes szolgáltatásokat nyújtó alkalmazások között. A megbízható adatátvitelt nyújtó TCP [1] protokoll jelen esetben nagyban támogatja ezen szolgáltatások nyújtásához szükséges transzport-funkciókat.

Az internetes alkalmazások döntő többségben TCP-t használnak; épp az alapvetően nyújtott szolgáltatás-garanciák miatt. Ezért egy hatékony, és valamennyire is valósághű forgalmat generáló tömeg-tesztelőnek TCP alapú forgalmat is kell tudnia generálni és kezelni. Ez azért különösen érdekes követelmény, mert a TCP sokkal összetettebb protokoll, mint az átviteli garanciákat nem nyújtó protokollok (pl. az UDP [8]) – tehát hardveres megvalósítása is jóval összetettebb feladat.

A nagy forgalmi intenzitást és adattömeget generáló tömeg-tesztelőket a hatékonyság növelése érdekében hardveres gyorsítással látják el – pl. FPGA támogatással. A csomagok generálása így kikerüli az összetettebb, szoftver-alapú protokoll-megvalósítást, és az operációs rendszer ütemezőjét is – ezzel jelentős gyorsítást érve el minden egyes csomag generálásakor.

A kihívás tehát adott: a TCP, mint kapcsolat-orientált – emiatt komplex – protokoll implementációja hardver-alapon, esetünkben: FPGA-ban.

2 Szakmai háttér

Ebben a fejezetben a dolgozat témájának környezete kerül bemutatásra:

- az LTE maghálózat, mivel az FPGA-ban megvalósított TCP-stack létrehozásának alapvető motivációja egy olyan forgalomgenerátor, ami az LTE maghálózat vizsgálatát hivatott segíteni;
- konkrét eszközként az SGA-TG forgalomgenerátor, mivel a koncepció bizonyításaként ehhez illesztettem a megoldásom;
- az FPGA-alapú hálózati eszközök, mivel ezen eszközök sajátosságai alapvetően meghatározzák a feladat megvalósítási módszertanát.

2.1 LTE maghálózat – Evolved Packet Core

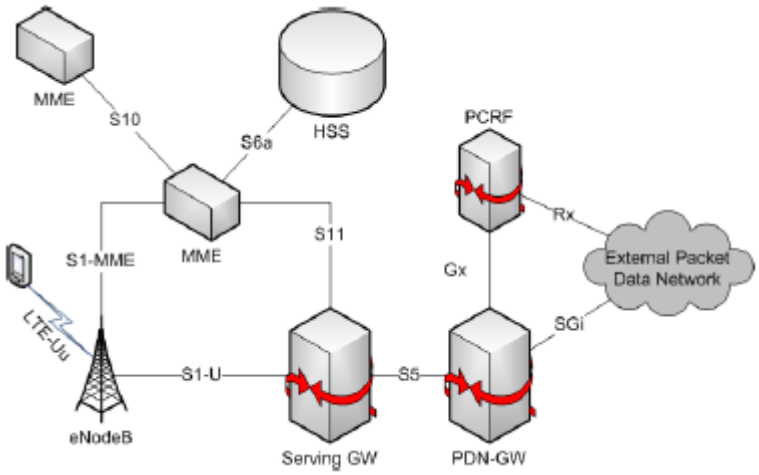
Az LTE [9] (Long Term Evolution) egy negyedik generációs mobil adatátviteli szabvány, amelyet a 3GPP Release 8 [10] ír le. Célja a „három és feledik" generációs HSDPA [11] hálózat továbbfejlesztése az olyan nagy felbontású video tartalmak és egyéb, nagy sávszélességet igénylő szolgáltatások számára.

A korábbi GSM és UMTS hálózatokhoz képest az LTE nem áramkörkapcsolt architektúrára támaszkodik, hanem egy úgynevezett „csomag kapcsolt" hálózati felépítésre. Ez azt jelenti, hogy az adatokat csomagok szállítják az áramkörkapcsolt hálózat helyett. A 3GPP döntése alapján, az EPS hálózaton IP csomagok szállítják az összes adatot.

Az EPS hálózat tervezésekor az alap koncepció az egyszerű architektúra volt, melyben az adatok szállítása hatékonyabb teljesítmény és költség szempontjából. Kevés csomópontot érintve haladnak át a csomagok a hálózaton, valamint elkerülendők a protokoll konverziók. Arról is döntés született, hogy megtartják a felhasználói adatok (user plane) és a jelzések (control plane) szétválasztását a független skálázhatóság érdekében. Ennek a funkcionális szétválasztásnak köszönhetően az operátorok könnyedén méretezhetik a hálózataikat.

A 2. ábrán az EPS (Evolved Packet-switched) alapvető hálózati architektúrája látható, egy felcsatlakozott felhasználói készülékkel az E-UTRAN-on (LTE hozzáférési hálózat) keresztül. Az EnodeB az LTE rádiós bázisállomása. Ezen az ábrán az EPC

hálózat (LTE maghálózat) négy hálózati elemből épül fel: a Serving Gateway-ből, a PDN Gateway-ből, az MME-ből és a HSS-ből. Az EPC a külső IP hálózatra csatlakozik, amely magába foglalja az IP Multimédia maghálózatot (IMS-IP Multimedia Core Network Subsystem [12]).



2. ábra - LTE architektúra: főbb elemek és interfészek

- HSS:

Alapvetően a HSS (Home Subscriber Server) egy olyan adatbázis, amely a felhasználókra és az előfizetőkre vonatkozó adatokat tárolja. Támogatja a mobilitás menedzsmentet, hívás és munkamenet beállításokat, felhasználó hitelesítését, hozzáférés engedélyezést. Lényegében a HSS valósítja meg az UMTS hálózatban alkalmazott HLR (Home Location Register) és AuC (Authentication Center) feladatait.

- Serving Gateway:

A Serving Gateway -kiszolgáló átjáró- a kapcsolódási pont a rádiós interfész és a csomagkapcsolt maghálózat között (EPC). Ahogy a neve is mondja, ez az eszköz szolgálja ki a felhasználói készülék kimenő és beérkező IP csomagjainak az útválasztását. Emellett ez a kapcsolódási pont más 3GPP hálózatokkal (GSM, UMTS) is.

- PDN Gateway:

Ez az adathálózati átjáró valósítja meg a kapcsolatot az EPC hálózat és a külső IP hálózat között. Az ilyen hálózatokat Packet Data Network-nek hívjuk, innen a „PDN” [13] elnevezés. A PDN GW határozza meg a PDN-ek ki- és beérkező csomagjainak útvonalát, IP cím kiosztást végez, adatcsomag szűrést, adatforgalom alapú számlázást.

Tehát, ezen a két átjárón keresztül haladó adatcsomagok szállítják az IP adatforgalmat a felhasználói készülék és a külső hálózatok között. A gyakorlatban ezeket az átjárókat szinte egy dobozban, egy egységként valósítják meg a hálózat szolgáltatók.

- MME:

Az MME (Mobility Management Entity) vagyis a Mobilitás vezérlő egység dolgozza fel a jelzéseket. Ez kezeli a mobilitással kapcsolatos jelzések és az E-UTRAN-hoz való biztonságos kapcsolódás jelzéseinek valamennyiét. Az MME felel a készenléti állapotban lévő mobil készülékek követéséért és kereséséért.

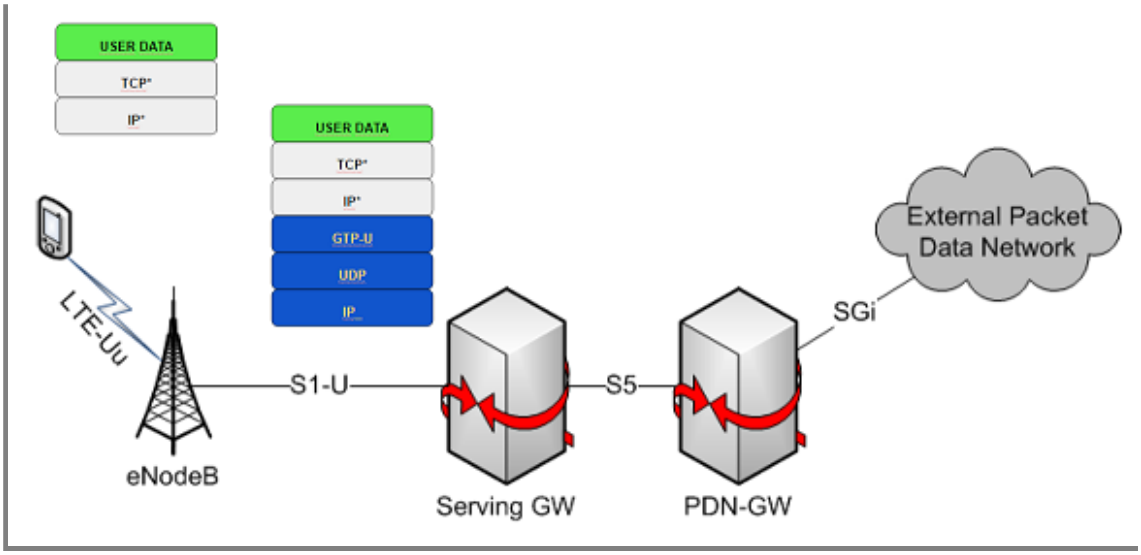
- S1_MME link:

Ezen a linken kizárólag jelzési üzenetek haladnak, mégpedig az eNodeB és az MME között. Ezek a jelzési üzenetek lehetnek kezdeti konfigurációs beállítások, előfizető fel- vagy lecsatlakozását szolgáló jelzések és persze a mobilitás vezérlő üzenetek is ezen a vonalon haladnak.

- S1_U link:

Az eNodeB és a Serving Gateway-t összekötő vonalat nevezzük S1_U-nak. Ezen az interfészen haladnak át az eNodeB felől érkező felhasználói kérések és a külső hálózat szerverei felöl érkező adatok, egy szóval minden felhasználói adat.

Számomra az S1_U vonal és az ezen megvalósított protokoll-stack fontos, mivel itt kell az általam generált forgalmat átengedni. Ahogy a 3. ábrán is látható az általam megvalósított TCP*/IP* csomagokat az eNodeB GTP-U [14] protokoll, UDP protokoll, valamint egy második IP felett továbbítja az Sx interfészeken. Ezzel a jelenlegi munka során nem foglalkoztam, az ábrán kékkel szereplő protokoll-stack korántsem megközelíthető kihívás, mint az általam megvalósított TCP-stack.



3. ábra -

2.2 Forgalomgenerátorok az LTE maghálózat vizsgálatára

Dr. Varga Pál és Olaszi Péter -LTE core network testing using generated traffic based on models from real-life data [15] közös cikkében leírt. Az Evolved Packet Core hálózatok terhelés vizsgálatára már létező megoldásokat mutatom be röviden.

A Polaris system [16] teszt esetei rugalmasan konfigurálhatóak, de a forgalmi variációkat korlátozza az általános a forgalmi terhelés mértéke.

TeraVM megoldásának [17] lenyűgöző képessége, az akár 1 Tbps-os adatsebesség. A teszt után támogatja a flow-kénti vizsgálatot, amelyet a Capture-ölés megjelenítésének lehetőségei szabnak határt.

IXIA teszt berendezés [18] biztosítja a szolgáltatás és a felhasználók közti szétválasztást, habár nem foglalkozik az általános forgalmi felépítéssel és a modellezés problémáival.

Az ng4T tester [19], a Torrent6100 [20], az Aricent megoldás [21], és a Nethawk EAST [22] mind biztosítanak nagyon változatos jelzési folyam variációkat, de mindből hiányzik az élethű variancia a felhasználói adatfolyamok esetében.

NS-3 [23] Discrete-event Network Simulator is használható forgalom generálására – habár ezt elsődlegesen kutatási célokra tervezték, így az élő berendezésekkel való összeköttetése kihívást jelent.

2.3 SGA-TG: forgalomgenerátor a mobil maghálózathoz

Munkám implementációs része jelenleg az SGA-TG forgalomgenerátorhoz kapcsolódik szorosan, ehhez illesztettem a TCP-stack-et és ezzel végeztem a verifikációt. Maga a TCP-stack azonban általánosan felhasználható más alkalmazásokban is; ehhez adott esetben az illesztések átalakítására lehet szükség.

A jelzési üzeneteket szükség esetén hálózati, szolgáltatási, és felhasználói paraméterekkel töltik ki. A gyakorlatban viszonylag nagy része ezeknek az üzeneteknek nem változik: számos hívás-beállítás, esemény vagy mobilitás vezérlés ugyanazzal az értékkel szerepel. Hálózati és szolgáltatási paraméterek (pl. állomás kódok, elérhető szolgáltatások, várható QoS paraméterek) egyáltalán vagy csak kissé változnak; felhasználókkal kapcsolatos adatok (pl. végpont azonosítók, ideiglenes azonosítók), viszont eléggé változóak. Ebből a szempontból a protokoll üzenetek felépítéséhez minden párbeszéd esetében használható egy előre meghatározott sablon. Ezekben a sablonokban

- rögzített értékekkel kitöltött adatok,

- meghatározott határok közti változó paraméterek (számos szabály alapján), és

- protokoll által meghatározott változó paraméterek (azaz ideiglenes azonosítók, szekvencia számok stb.).

A felhasználói adatfolyamot összetett modell alapján generált egyéni csomagok alkotják. A tényleges csomagméretek és átviteli késleltetések is a használatban lévő összetett modell szabja meg.

Eddig az SGA-TG forgalomgenerátor leginkább UDP-alapú forgalom kibocsátására volt képes nagy tömegben és sok IP-címről; a TCP-alapú forgalom kibocsátása korlátozott volt: hiányzott az, hogy százas, vagy ezres nagyságrendben tudjon párhuzamosan futó TCP folyamokat karban tartani a kívánt sebességen. Jelen munka erre a konkrét feladatra is megoldást kíván nyújtani.

Az UDP implementáció 1 Gbit/s sebességű változatában a csomagok összeállítását a szoftver végzi, a kész üzeneteket továbbításra adja át az erre a feladatra használt hálózati kártya számára. A gyorsabb, 10 Gbps-os sebességgel üzemelő megoldásban, viszont a szoftver csak az adatméretet és a csomaghoz kapcsolódó engedélyező jelet szolgáltatja. A csomag összeállítását a hálózati kártya végzi.

A kollégáim és jómagam által készített TCP-stack, szoftveres vezérlése, a forgalmi profilok (payload hosszok, késleltetéshez szükséges belső számlálóértékek), IP forrás és cél címek, Socketek száma, TCP port címek küldésében merül ki PCIe interfészen keresztül a kártyára. A forgalmazással kapcsolatos többi feladatot teljes mértékben a kártya hajtja végre.

2.4 FPGA-alapú hálózati eszközök

A NetFPGA SUME [24] eszköze használható hálózati kártyaként (NIC-Network Interface Controller), többportos switch-nek, tűzfalként, teszt és mérési környezetként, stb.

NetFPGA CML [25] ideális rendkívül komplex, kis sávszélességű alkalmazások fejlesztéséhez. Ethernet Interfész loopback teszt, referencia NIC, learning Switch, referencia router megvalósítására használják.

NetFPGA 10G [26]: Ideális nagy sávszélességű alkalmazások számára. 10G/1G Ethernet interfész loopback teszthez, referencia NIC 10G/1G

NetFPGA 1G [27] alkalmazások: Ethernet switch, Buffer monitoring router, Packet generator, End-to-end Ethernet authorization, Tuneling OpenFlow switch with ICMP, stb.

C-Gep [28]: Nagy pontosságú 64 bites időbélyeg 4/8 ns-os felbontással szinkronizálva atomórával vagy PTP protokoll használatával. Kifinomult csomag műveletek: elemzés, szűrés, irányítás, osztályozás. Veszteségmentes csomag megjelenítés, azonnali fejléce levágással. Kizárólag a fejlécek megjelenítése: konfigurálható protokoll réteg mélységgel, dekódolás rögtön a hardveren. Csomag pufferelés és TCP adatfolyam monitorozás. Paraméterezett, vonali sebesség támogatású csomag/folyam generátor aktív mérésekhez.

C-BOARD [29]: 64 bites időbélyeg 4/8 ns-os felbontással. Veszteségmentes csomag megjelenítés korlátozva a fogadó PC gyorsasága és erőforrásai által. Csak fejléc megjelenítés: konfigurálható protokoll mélységgel hardver által egyből dekódolva. Teljesen PCAP/WINPCAP kompatibilis interfész. Paraméterezett, vonali sebesség képes csomag/folyam generátor aktív mérésekhez. RFC 2455 tesztelő implementáció

SGA-10GED [30]: A firmware mellett az AITIA számos alkalmazás alapot kínál segítségül, a fejlesztők saját kódjaihoz. Ultra nagy sebességű (akár 10Gb/s) Ethernet

hálózati forgalom elemzés. Hálózat tesztelés, karbantartás. Tűzfal, útválasztási funkciók. Forgalom generátor.

2.4.1 Megfigyelés és Analízis

Szinte minden mérnöki munkát egy jól meggondolt, átfogó, információ- és adatgyűjtéssel érdemes kezdeni – nincs ez másképp a forgalomgenerátorok világában sem. Ebben a speciális esetben a valós hétköznapi forgalmak rögzítésére és analízisére van szükség – a felhasználói csomagokból és a jelzésüzenetekből egyaránt.

A munkám során volt lehetőségem belehallgatni konkrét S1_U és S1_MME linkek forgalmaiba. Szerencsére esetemben emellett megfelelő rálátást és információ-tartalmat biztosítottak más általános internetes TCP forgalmak vizsgálatai is.

A rendelkezésemre álló Wireshark [31] .pcap fájlok alapján arra a következtetésre jutottam, hogy a *Facebook* és *Youtube* forgalmi minták generálása a legésszerűbb a felhasználói szokások szimulálása szempontjából. A továbbiakban olyan forgalom generálása a cél, amely – a hálózati eszközök számára – nehezen megkülönböztethető az eredeti, valós forgalmi mintáktól.

2.4.2 Modell alkotás

Egy jó forgalmi modell alkotásához nélkülözhetetlenek a lényeges paraméterek és üzenetminták, melyek különböző felhasználói szokású forgalmi mintákból épülnek fel. Ismernünk kell az általunk megvalósított protokoll fejlécének felépítését, és a protokoll minden funkcióját. Ezen ismeretek jó alapot adnak az eredményes modell-alkotáshoz.

Jelen esetben az kezdeti feladatom a TCP protokoll megismerése volt, az előzőekben ismertetett szempontok alapján. Az erőforrás-felhasználás minimalizálása érdekében a modellalkotást a feleslegesen „drága” funkciók keresésével és az ezekhez tartozó protokoll-elemek (fejléc mezők, időzítők, állapotok, algoritmusok) megengedhető egyszerűsítésével folytattam. Emellett a kapcsolatokat azonosító TCP port és IP cím mezőinek forgalom-generátoron belüli optimalizálására is nagy hangsúlyt fektettem. Ezen az úton jutottam el a jelenlegi rendszer tervéhez.

2.4.3 Megvalósítás

A Modell alapján meghatározott paraméterű, hálózati feladatot megvalósító eszköz implementálása, megvalósítása.

A teljes forgalomgenerátor egészét alkotó különálló kisebb egységek prototípusait, kezdetlegesen megvalósítottam. A modulok megfelelő működésnek ellenőrzésére és a hibák detektálására a fejlesztőkörnyezetbe integrált szimulációs ISim tool-t használtam. A teljes forgalom generátor összerakásánál nagy nehézséget okozott a modulok összereszelésére, amiben a szintén sok segítséget nyújtott a fejlesztőkörnyezet által nyújtott szimulációs lehetőség.

2.4.4 Ellenőrzés és tökéletesítés

A forgalomgenerátor kiajánlása és éles üzembe helyezése előtt fontos, hogy megfelelő figyelmet szenteljünk a tesztek elvégzésére. Ennek a forgalmi modellezéshez kapcsolódó része a valós és a szintetizált forgalmak statisztikai paramétereinek összevetése. A (feladat szempontjából) lényeges eltérések minimalizálása érdekében a paraméterek finomítása is feladatunk.

A már korábban említett szoftveres szimulációk mellett több hardveres tesztet is lefolytattam. Ezt az AITIA International ZRt. által készített C-Board egy FPGA-jának felhasználásával végeztem el. A teszthez felhasznált Virtex-5 családba tartozó XC5VLX110T típusú FPGA-ra példányosítottam egy szerver- és egy kliens-oldali forgalomgenerátort. Az eredményül szolgáló adatokat a két oldal által használt portok kitükrözésével nyertem ki. Az így kapott adatok értelmezéséhez a nagyon sok kényelmi funkcióval rendelkező, a hálózati forgalom vizsgálatára szolgáló Wireshark szoftvert használtam.

Emellett egy ismert próbahálózaton is érdemes tesztelni a megoldást – erre a dolgozat befejezése előtt sajnos nem nyílt lehetőségem. Bár ezt a lépést eddig nem volt lehetőségem elvégezni, de így is rendelkezésemre állt elfogadható mennyiségű teszteset. Ezek alapján egyértelműen megalapozott az energia, erőforrás befektetése a próbahálózati tesztelésre, majd ha az sikeres: a valós hálózati bevezetésre.

2.4.5 Bevezetés (Telepítés)

Végezetül a teljesen kész eszközt magára a tesztelendő (próba-, vagy pilot-) hálózatra kell telepíteni, amikor már megbizonyosodtunk a forgalomgenerátor megfelelő működéséről.

3 A TCP stack optimalizációja a forgalomgenerátor feladathoz

3.1 Követelmények

A TCP protokoll megbízható mivoltából fakadóan több a biztonságos adatátvitelt biztosító funkcióra FPGA kompatibilis megoldást kellett találnunk, az RFC [4] által meghatározott TCP szabványból. A három utas kézfogással megvalósuló kapcsolat felépítésből és bontásból egyáltalán nem engedtünk az ehhez szükséges üzenettípusokkal együtt (SYN, SYNACK, FIN, FINACK). A sorrendhelyes és a hibamentes adatszállítás nyújtásához a helyes sorszámozás és nyugtázás implementálása elkerülhetetlen, már csak a valódinak tűnő fejléc szempontjából is. A nyugták elmaradásából adódó újraküldések megvalósításához minden csomag kibocsájtásánál egy újraküldési időzítő társítása szükséges a funkció eredményességéhez. Ezzel együtt szükségünk van arra, hogy minden flow számára egy előre meghatározott átviteli sebességet tudjunk biztosítani. Ezt, hibák esetén egy leegyszerűsített torlódáskezelő algoritmussal tudjuk befolyásolni, amely eredményeképp a szokásos TCP fűrészfog alakú adatsebesség – idő diagramot kell, kapjuk. Mivel a mi egységeink állnak a hálózat végpontjain, ezért nem kell általános - bonyolult- torlódáskezelő algoritmusokat implementálnunk, elég ez az egyszerűsített torlódáskezelő algoritmus. A csúszó ablak megvalósításával, és az ebből fakadó problémák kezelésével abszolút nem kell foglalkoznunk.

A projekt folyamán a saját egyszerűsítésekkel bíró TCP implementációt TCP Budapest néven nevezem el.

3.2 Egyszerűsítések az üzenet-fejlécben

Értelemszerűen az erőforrás-felhasználás csökkentése érdekében a szegmens méretének csökkentése volt a legegyszerűbben megvalósítható cél. Másképp fogalmazva: mely módon lehetne olyan funkciókat megvalósító mezőket elhagyni a fejlécből, amelyeket luxus lenne a mi szempontunkból benne hagyni.

Miután több megoldással kísérleteztem elméleti szinten, végül a következő megoldásra jutottam. A keretből teljes mértékben elhagyásra került az Option és a Padding mező, amellyel a szegmens méretét 32 bittel, azaz 4 bájttal tudtuk csökkenteni.

Ez az eredmény nem túlságosan nagy bravúr, de ezzel is tudtam csökkenteni a tárolandó adat mennyiséget. Emellett a TCP fejléc maradék mezőiből meghatároztam néhányat, amelyek funkciói számomra elhanyagolhatóak. Ezáltal a Header Length, Reserved, URG, PSH, RST fejléc részeket tetszőleges adattal töltjük fel.

3.3 Ésszerűsítések a kapcsolat-kezelésben

A kapcsolat-felépítés és bontás folyamatában nagyon kevés egyszerűsítési lehetőség van. Emellett megtehetjük, hogy a kezdő sorszámokat nem random választjuk, hanem nullától kezdjük a számozást. Az Nyugtázási és Szekvencia sorszám jelenlegi megoldásomban csak 16 biten van értelmezve a szabvány által maximált 32 helyett, ezt 20-20-ig lehet növelni „ingyen”. Ha nagyobb sorszámokat szeretnénk, ahhoz extra erőforrásokra lesz szükség.

Az adat üzenetekre a következő megkötések vannak:

- A payload hossza 16 byte égésszeresei lehetnek csak.
- Minden üzenetet külön nyugtáz a szimulátor.

Egy forrás és célpont közötti kommunikációnál egyértelműen az IP cím és a TCP port szám határoz meg egy kapcsolatot. Viszont, számunkra ezen azonosítási metódus feleslegesen nagy méretnek bizonyult, ezért a modul periféria-egységei egy úgynevezett TCPID-t képeznek le az IP cím és a TCP port számból; ezzel azonosítunk egy kapcsolatot. Ezzel a megoldással a modulunkon belüli kapcsolat-azonosításhoz sikerült 10 bitre redukálnunk a folyam-azonosításra eredetileg használt 96 bitet. Felmerülhet a kérdés, hogy akkor a kimeneten milyen fejléccel fog szerepelni egy adott kapcsolathoz tartozó csomag. Ezt egy viszonylag egyszerű megoldással orvosoltuk. A kimeneti periféria egységben tárolunk IP cím és TCP port szám párokat, amelyek adott TCPID-khoz vannak rendelve, így a hálózat felé a kimeneten már a hagyományos fejléc formátumot kapjuk.

3.4 Ésszerűsítések a folyam-szabályozásban

Az új TCP-implementációnál azzal a megfontolással éltünk a folyamszabályozás során, miszerint a jelenlegi technikai eszközeinkben az aktuális célállomás puffere nem valószínű, hogy túltelítődik. Ezzel azért számolhatunk, mert az általunk megcélzott alkalmazások (pl. forgalomgenerátor) architektúrájában a célállomás is hasonló TCP implementációval rendelkezik, sőt, feltehetjük, hogy vezérlése és konfigurációja a

kezünkben van. A célállomás pufferét így, hogy a kezünkben van a tervezése, lényegében tekinthetjük nagyon jó közelítéssel végtelennek. Ezáltal a forgalom szabályozásra nem készítettem külön, ezt a funkciót megvalósító implementációt.

Végezetül, a munkánk során talán a legnagyobb kihívásokat az újraküldés időzítése, és a torlódás kezelése okozták.

A TCP kapcsolat alapvető tulajdonsága, hogy az adatot tartalmazó üzeneteihez egy időzítőt társít azért, hogy garantálni tudja a veszteségmentes kommunikációt. Az újraküldési időzítés implementálásának alapvető megoldása az lehetne, hogy minden egyes flow üzenetei számára elindítunk a küldés pillanatában egy időzítőt, ennek a számlálónak a meghatározott értéke alapján és a beérkező ACK üzenetek alapján tudnánk eldönteni, hogy szükséges-e egy fiktív kapcsolat és a hozzátartozó adott üzenet újraküldése. Ezt egy elég innovatív erőforrást felhasználó módon, egy számláló és több BRAM[33] memória felhasználásával implementáltuk. Ezzel a megoldással hatalmas erőforrás-spórolást értünk el, mivel egy időzítőhöz társítunk több flow-t, nem minden flow-hoz társítunk külön időzítőt. Ezt a funkciót „rotációs FIFO” néven illettük, és a későbbiekben részletesen ismertetem a működését.

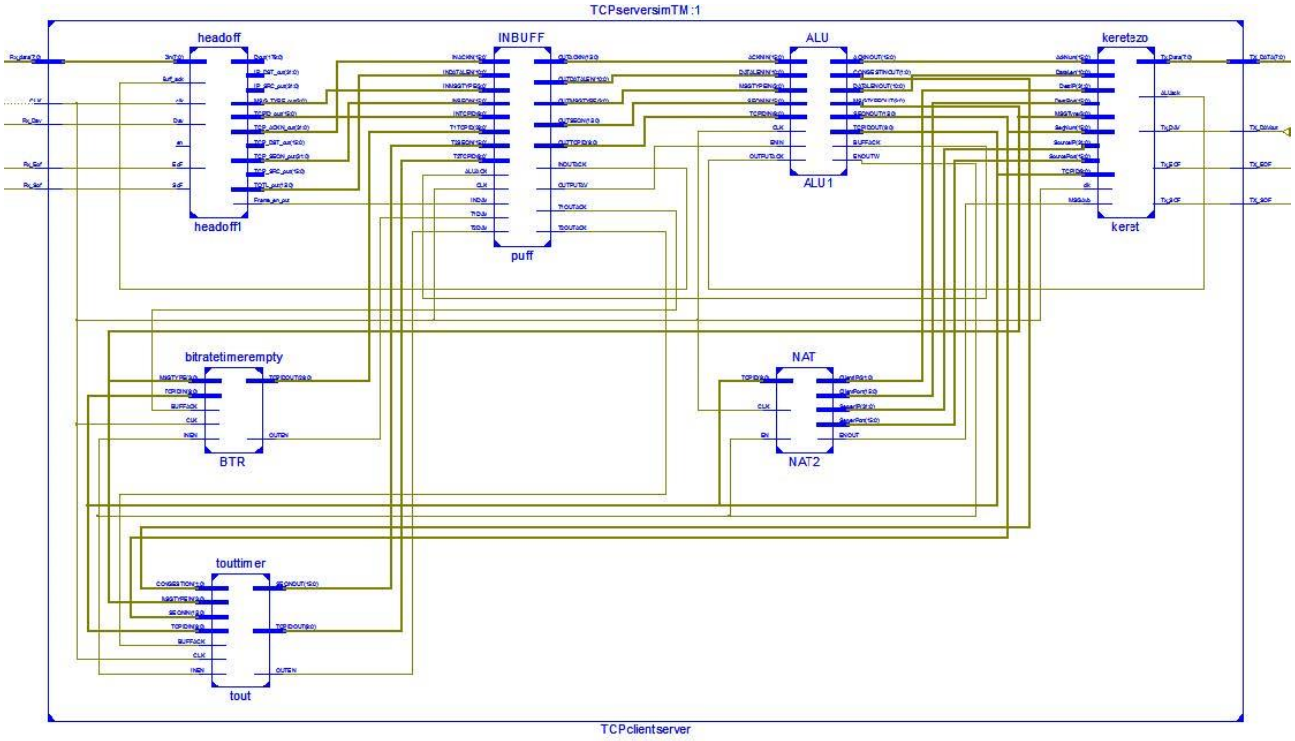
Összefoglalásul, az egyszerűsítések és elhanyagolások legjobb szemléltetése érdekében, a 4. ábrán látható táblázatot készítettem el.

Problémás funkciók	RFC TCP	TCP Budapest
Tcp fejléc felépítése :	Lehetőség opciók használatára.	Teljes opciók mező elhagyása.
Kapcsolat azonosítás:	IP cím és TCP port alapú azonosítás.	Kapcsolatonként egyedi 10 bites TCPID, TCP/URGP mezőben továbbítva.
Payloda méret:	További rétegbeli megkötéseken belül akármekkora méret.	További rétegbeli megkötéseken belül 16 Byte vagy ennek egésszám többszörösei.
Nyugtázási módok:	Többszörös ACK, SACK	Menden üzenetet külön nyugtáz.
Számozások:	Szekvencia és Nyugta számozás random kezdőértéktől. 32-32 biten.	Szekvencia és Nyugta számozás nulla kezdőértéktől. 16-16 biten, amely növelhető 20-20-ra.
Forgalom szabályozás - Újraküldés időzítés:	Számos újraküldési időt meghatározó algoritmus,	Közösen megvalósított Újraküldési időzítés és Torlódás kezelés: Időzítés alapú újraküldés, további csomagvesztésnél az újraküldési időzítés növelésével az átviteli sebesség változtatása. Fűrészfog közeli sebesség változás.
Forgalom szabályozás - Torlódás kezelés:	Többféle torlódáskezelést megvalósító implementáció. (Reno, Tahoe, Vegas)	
SACK, URG, PUSH opciók; Windowsize; Reset	Implementáció függő.	Teljesen kihagyott funkciók.

4. ábra – RFC és TCP Budapest összehasonlítása

4 TCP-Stack megvalósítása forgalom generálás céljára

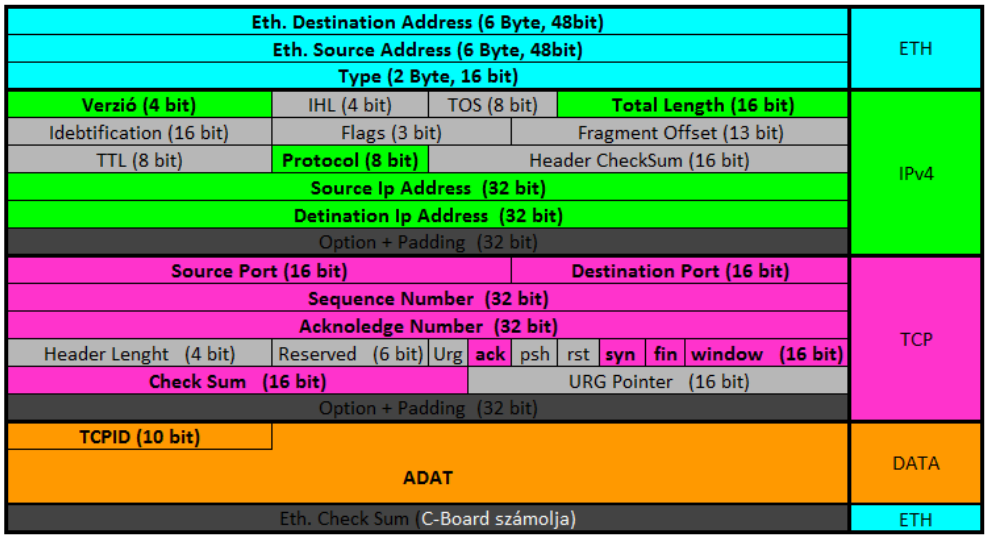
A TCP-Stack összeállításában kollégáimmal sikerült elérnünk a cél érdekében legjobban optimalizált modul összeállítást, a TCP-vel kapcsolatos követelmények kielégítése mellett. A koncepció lényege és az architektúra alapvető elemei a [32] forrásban részletesebben is megtalálhatók. Dolgozatomban kizárólag az architektúra felépítésével és az alkotóelemek működésének ismertetésével összefoglalás szintjén foglalkozom. Az 5. ábrán látható a forgalom generátort felépítő belső modulok és összeköttetéseik.



5. ábra - Forgalom generátor architektúra

4.1 Periféria modulok

A következő perifériamodulok által feldolgozott, és előállított csomagformátum a 6. ábrán látható. A képen feketével a teljesen elhagyott, szürkével a tetszőleges értékkel – jelen esetben nullával – kitölthető mezőket jelöljük.



6. ábra – Teljes fejléccel ellátott szegmens

4.1.1 Demodulátor (headoff)

Ez a modul felelős azért, hogy a hálózati interfészén beérkező bájtfolyamot kezelje: a kimenetén az aktuálisan feldolgozott csomag fontos részeit kategorikusan szétválassza, és egy speciális üzenet típust előállítva továbbítsa a logikai egység számára. A projekt jelenlegi állapota szerint az IP fejléc forrás, cél címét és teljes hosszát tartalmazó mezőt – utóbbiból az IP és TCP fejléc hosszát, azaz 40 bájtot kivonva – továbbítja. Fontos működésbeli jellegzetesség még, hogy a TCP fejlécből a Szekvencia számon és Nyugta számon túl egy speciális üzenettípust is átad az ALU számára. Ezt a belső működést segítő üzenettípust a TCP fejléc flag mezőjében található ACK, SYN, FIN bitek határozza meg. Emellett az információk mellett a flow-k azonosítására létrehozott a flow-k azonosítására használt TCPID-t is elküldi – az adatok mellett. Ennek az információnak a továbbítására két megoldás is született az elhelyezkedését illetően. Az első elgondolás alapján a TCP fejléc Urgent Pointer mezőjében továbbítottuk a 10 bites azonosítót. Jelenleg a legfrissebb verzióban az adat rész első két bájtjában továbbítjuk. Erre a változtatásra a hálózati eszközök miatt volt szükség, mert ezeknek megengedett az Urgent Pointer átírása.

4.1.2 Keretező és címfordító (NAT)

Keretező: A processzor utasításaiból készít OSI kompatibilis üzeneteket, és vezérli a MAC-et. Itt lényegében egy flow-hoz tartozó teljes fejlécet kell összeállítania a modulnak és a kimenetén a tesztelni kívánt hálózat felé bájtonként kiadni. A keret összeállításánál ismét lehet egyszerűsítésekkel élni. Az Ethernet fejléc forrás és cél MAC

címeit ebben az esetben konstansnak vettük, valamint a forgalom generálás folyamán IPv2 és TCP kikötése okán az Ethernet fejléc típus mezőjét fixen az IPv2-t azonosító 0800 hexadecimális értékre az IP fejléc protokoll mezőjét 06 hexadecimális értékre állítjuk be. Ennél sokkal kényesebbek a minden esetben előállítandó IP, TCP ellenőrző összeg és TCP Flag-ek. Az ellenőrző összegeket a vonatkozó RFC[4] által meghatározott módon helyesen lett implementálva. Magához a kerethez és a számolni kívánt adatokhoz az ALU-n kívül szükségünk van a processzor flow azonosítóhoz IP cím és Port párok rendelésére. Ezt a műveletet a NAT modul valósítja meg. Szerencsére itt is alkalmazhatóak bizonyos egyszerűsítések, mint az, hogy a szerver oldali IP címet és a Port párokat állandó értékekre állítottuk be.

4.2 Processzor moduljai

4.2.1 INBUFFER

A Timeout, Bitrate timer és a MAC bemenet üzeneteinek sorosítására, átmeneti tárolására szolgáló modul. A sorosítás mellett fontos feladata, hogy a felsorolt moduloktól érkező információt megfelelő formába hozza az ALU számára. Fontos a megfelelő működéshez, hogy mindegyik modul időben sorra kerüljön az ALU-nál. A kiéheztetés elkerülése érdekében az INBUFFER egy 36 kilobit tárolási kapacitású FIFO-t tartalmaz, melynek a bemenetére multiplexeltük a demodulátort és a két időzítő modult. A bemenet vezérlését egy statikus arbiter, a kimenet vezérlését pedig az ALU végzi.

4.2.2 ALU

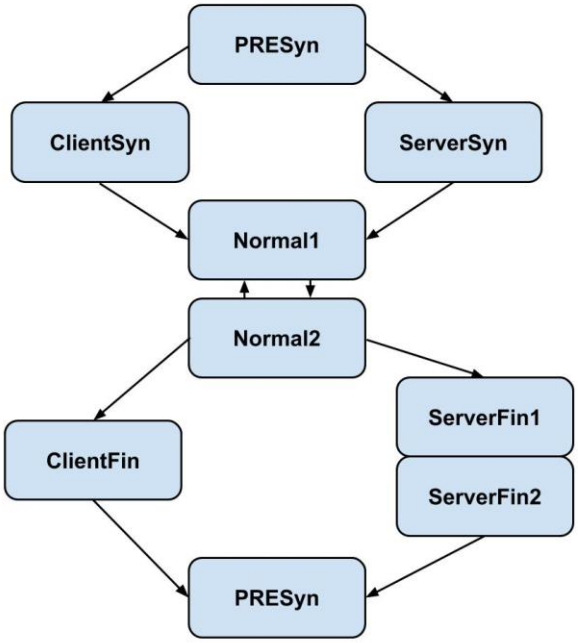
A generátor agya, a beérkező üzenetekből és a flow-k eltárolt adataiból, kódolja a TCP protokollnak megfelelő válasz üzeneteket, ill. állapotváltozásokat. Beérkező üzenetek lehetnek egy másik generátor által küldött üzenetek (pl. a kapcsolat felépítésével, bontásával, vagy adatátvitellel kapcsolatos üzenetek). Ezentúl a központi logikai egység kezeli az általános adatsebesség és újraküldési időzítő üzeneteit is. Ezeknek az üzenettípusoknak megfelelő válaszüzenetet állít elő. A belső működést is nagyban segítő üzenet típusokat a 1. táblázat foglalja össze.

Állapotok	SYN	SYN-ACK	ACK	DATA	FIN	TOUT	BRT	FIN-ACK
Bináris kód	0000	0001	0010	0011	0100	0110	0111	0101

1. táblázat - Forgalom generátor üzenettípusai

Az ALU fő belső állapotai és közöttük fennálló kapcsolatokat a 7. ábrán láthatjuk. PreSYN a fogalom generátor inicializálás előtti alapállapota, bemeneten érkező SYN üzenet – amely ServerSyn állapotba viszi a rendszert –, vagy egy Wakeup üzenet – amely ClientSyn állapotba viszi a rendszert– ébreszthetí fel ebből az állapotból.

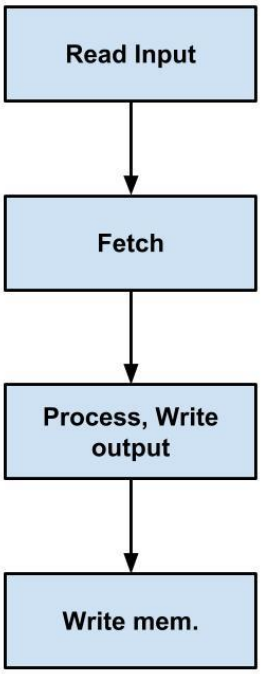
A Wakeup üzenet célja, hogy a forgalomgenerátort elindítsuk, kapcsolat felépítésre késztessük. A kliens oldali egységbe példányosított speciális Bitrate időzítővel érjük el a Wakeup üzenetet. Ezután a Kliens szinkronizációs (ClientSyn) állapotba kerülünk. Ebben az állapotban a SYN üzenet kiküldése után várjuk a másik oldalról érkező SYN ACK választ, amire normális esetben egy ACK üzenettel válaszolunk, ezzel elérve a három-utas kapcsolat felépítést, majd Normál1 állapotban folytatjuk a működést. A SYN ACK válasz elmaradása etében az újraküldési időzítő lejárta után egy újabb SYN üzenet kiadásával maradunk ebben az állapotban.



7. ábra – ALU működéséhez használt fő állapotok

A Normal1 és Normal2 állapotokat a flow-k normális adatátvitelének biztosítására hoztuk létre. Ezekben az állapotokban került implementálásra a beérkező adatcsomagokra küldendő nyugták előállítása. Ezekben az állapotokban kezeljük továbbá az alap adatsebesség eléréséhez szolgáló BRT üzeneteket, a csomagvesztést jelzésére szolgáló TOUT üzeneteket. Előbbi abban az esetben jut érvényre, ha minden korábbi csomag nyugtázva lett, utóbbi elmaradt nyugtázás következményeként újraküldést eredményez. Hasonlóképpen a kapcsolat felépítéséhez a Normal1-es és 2-es állapotokból CLOSESOCKET vagy FIN üzenet vezérli az ALU-t a kapcsolat bontására szolgáló Kliens illetve Szerver FIN állapotokba. A kapcsolat bontására használt állapotokban teljesen hasonlóan zajlik le a kapcsolat bontás, mint a kapcsolat felépítés állapotaiban. A három-utas kézfogás elvégzése után a kezdeti PreSyn állapotban találjuk magunkat.

A működés helyessége érdekében a flow-k adatainak tárolásához két darab 36 kilobites BRAM-ot használunk. A TCPID-val címezzük a memóriákat ezért egy ALU 1024 flow kezelését képes egyszerre ellátni. A memóriában tároljuk a mindenkori Nyugta számot, utoljára elküldött csomag adat hosszát, torlódásvezérlő biteket és az aktuális flow állapotot.



8. ábra – Processzor feldolgozási lépések

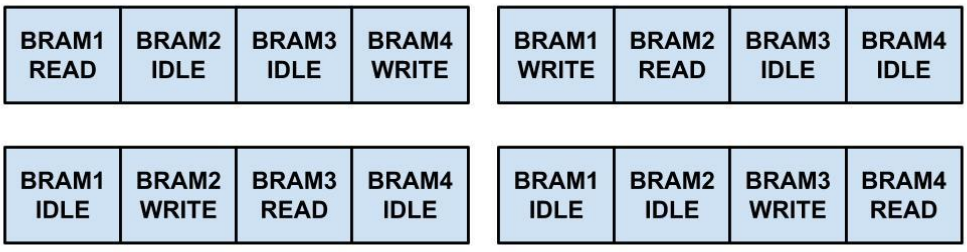
Az aritmetikai-logikai egység a 8. ábrán látható módon hajtja végre az utasítás feldolgozást. Ezt csak 6 órajel alatt képes elvégezni, a memória olvasás késleltetése miatt.

Amint a keretező visszajelez, hogy az aktuális üzenethez tartozó bájtfolyam elkészült, az ALU elkezdi a következő üzenet feldolgozását.

4.2.3 Congestion Timeout Timer

A TCP socketek fontos tulajdonságai közé tartozik, hogy minden adatot tartalmazó üzeneteihez egy újraküldési időzítőt társít. Ezzel garantálni tudja az esetleges üzenetvesztés esetén szükséges újraküldést. Ezt a problémát egy visszacsatolással oldjuk meg. A Keretező és az ALU közötti adatbuszon hallgatózik és a rá vonatkozó üzenetek adatait eltárolja. Ezt a mindenkori torlódáshoz tartozó idő után az Inbufferen keresztül továbbítja az ALU számára. A flow-k azonosításához a TCPID-kat, ezen belül az üzenetek azonosításához Szekvencia számot, valamint a torlódás (congestion) jelzésére szolgáló változót továbbít. Legutóbbit ezt úgy implementáltuk, hogy csomagvesztéssel detektáljuk a hálózat torlódását. Ez esetben növeljük a CONG számláló értékét. A Tout Timer modul minden egyes memória olvasás esetén megvizsgálja a congestion értékét. A vektor nulla értéke esetén egyből, nagyobb értékek esetén eggyel csökkentve újra visszatöltjük a BRAM-ba [33]. Ezzel a megoldással a torlódás kezelését az átviteli sebesség csökkentésével érjük el.

A modul megvalósításához 4 darab 18 kilobites BRAM példányt használtunk fel. Egy számláló segítségével a Block RAM példányok közti kiválasztásokkal érjük el az időzítés funkciót. Ezzel a BRAM-ok egy FIFO-t^{TM} alkotnak. Minden időpillanatban egyből olvasunk, egybe írunk és kettő idle. A működést a 9. ábra szemlélteti.



9. ábra - FIFO^{TM} működése

4.2.4 Bitrate Timer

A TCP forgalom generálásához szükségünk volt egy olyan modulra, amely képes az adatüzeneteket egy előre meghatározott átviteli sebességgel kiadni. Erre a feladatra ad

megoldást a Bitrate timer modul. Ez a modul a másik számlálóhoz hasonlóan szintén az ALU és keretező közti buszon hallgatózik. A rá vonatkozó üzeneteket eltárolja, majd a pontosan kiszámolt időzítés után az Inbufferen keresztül az ALU számára visszaadja BRT üzenetként.

A modul elvi működése teljesen megegyezik a Tout Timerével, viszont ennél a modulnál kizárólag a TCPID-t tároljuk el. A rövidebb bejegyzés hossz miatt élhetünk azzal a helytakarékossági megfontolással, hogy egy memória rekeszbe három TCPID-t helyezünk el. Ezáltal hármasával adjuk át az Inbuffer számára az adatokat, amely bemenete erre a hármas adatáramlásra van tervezve. A memória szegmensek kiválasztása, olvasása és írása teljesen azonos a Tout Timer-ével. Annyiban különbözik, hogy itt egy darab 18 kilobites BRAM példányt használtunk fel. Ezt négy egyenlő részre osztva képeztük a különböző szegmenseket.

5 Konfiguráció

A forgalomgenerátor különböző beállításai során kihasználjuk az FPGA és a VHDL hardver leíró nyelv által nyújtott lehetőségeket. Ilyenek többek között a könnyű példányosítás és a hardverszintű, rendkívül gyors applikáció-működés. Az alapvető elképzelés az, hogy két teljes forgalom generátort, egy szervert és egy klienst példányosítunk. Ezek a vizsgálni kívánt hálózat két különböző hozzáférési pontján helyezkednek el. A két példány között annyi különbség van, hogy a kliens oldali generátor Bitrate timer moduljának memóriáját alapértelmezetten TCPID-kal töltöttük fel 0-tól 1023-ig. Ezzel elértük a konfiguráció szerinti automatikus kapcsolat-felépítést. A jelenleg verifikált verzióban mind a két forgalomgenerátor a konfigurálható paraméterek mellett - az egyszerűsítések végett - rendelkezik több fixen megválasztottal is.

Elsőként az a Szerver IP címét kell kiemelni, amely fixen 209.0.1.18-ra van beállítva. Emellett minden esetben a kliens port címe 80-as, a szerver port címe 120-as értékkel szerepel a TCP fejlécekben.

A konfigurálható paraméterek közé tartozik a NAT modulban található Block RAM memória tartalma, itt tároljuk TCPID-khoz tartozó a kliens oldali IP címeket. Ezen címek alapértelmezett értékei 254.0.0.0 - 254.0.3.255 között vannak megadva. A címeken túl a kapcsolat számának és az időzítések mértékének meghatározásával is élhetünk a forgalom generátor üzembe helyezésekor.

A kapcsolat-számokat úgy határozhatjuk meg, hogy a kliens oldali Bitrate timer modulban a BRAM szegmensein belül használt kiolvasási cím határokat adjuk meg. Lényegében ezzel azt határozzuk meg, hogy egy blokk szegmensből hány memória rekeszt olvasunk ki. Itt figyelmesen kell eljárni, mert egy memória rekesz három darab TCPID-t tartalmaz. Csak hármasával tudjuk megszabni a kapcsolat számokat, egészen a maximális 1024-ig. Ezt a 10. ábra szemlélteti.

```
---TCPID number/segment ;
SIGNAL FF1 : STD_LOGIC_VECTOR (6 downto 0):=B"1111111"; --Full:3*127=381db Id
SIGNAL FF2 : STD_LOGIC_VECTOR (6 downto 0):=B"1111111"; --Full:3*127=381db Id
SIGNAL FF3 : STD_LOGIC_VECTOR (6 downto 0):=B"1011000"; --88*3-2=262db Id
SIGNAL FF4 : STD_LOGIC_VECTOR (6 downto 0):=B"0000000"; --Empty-full of zeroes
                                                        ----------------------
                                                        --Sum:1024
```

10. ábra - Kapcsolatszám meghatározás

Fontos szempont a forgalom generátor működésénél az időzítések meghatározása, amely a teljes adat átvitelisebesség számításának alapjául szolgál.

Kezdésként a 11. ábrán összesített táblázat első sorában szereplő újraküldési időkből indultunk ki különböző „cong" (torlódás) értékek esetén. Elsőként 250ms-onként, legrosszabb esetben pedig egy másodpercenként küldjük újra a nyugtázatlan csomagokat. Az idők felezésével 16 különböző konfigurációs esetet határoztunk meg az újraküldés időzítésnek a beállítására.

Tout Timer			Cong = 00 1xTokenCTR	Cong = 01 2xTokenCTR	Cong = 10 3xTokenCTR	Cong = 11 4xTokenCTR	**Bitrate Timer**		
TokenCTR (bitszélesség)	Késleltetéshez szükséges belső számláló értéke		Delay [s]	Delay [s]	Delay [s]	Delay [s]	TokenCTR (bitszélesség)	Késleltetéshez szükséges belső számláló értéke	Delay [s]
25	31 250 000	1.eset	0,25	0,5	0,75	1	28	268 435 456	2,147484
24	15 625 000	2. eset	0,125	0,25	0,375	0,5	27	134 217 728	1,073742
23	7 812 500	4. eset	0,0625	0,125	0,1875	0,25	26	67 108 864	0,536871
22	3 906 250	5. eset	0,03125	0,0625	0,09375	0,125	25	33 554 432	0,268435
21	1 953 125	6. eset	0,015625	0,03125	0,046875	0,0625	24	16 777 216	0,134218
20	976 563	7.eset	0,0078125	0,015625	0,0234375	0,03125	23	8 388 608	0,067109
19	488 281	8. eset	0,00390625	0,0078125	0,01171875	0,015625	22	4 194 304	0,033554
18	244 141	9.eset	0,001953125	0,00390625	0,005859375	0,0078125	21	2 097 152	0,016777
17	122 070	11.eset	0,000976563	0,001953125	0,002929688	0,00390625	20	1 048 576	0,008389
16	61 035	12.eset	0,000488281	0,000976563	0,001464844	0,001953125	19	524 288	0,004194
15	30 518	13. eset	0,000244141	0,000488281	0,000732422	0,000976563	18	262 144	0,002097
14	15 259	14. eset	0,00012207	0,000244141	0,000366211	0,000488281	17	131 072	0,001049
13	7 629	15. eset	6,10352E-05	0,00012207	0,000183105	0,000244141	16	65 536	0,000524
12	3 815	16.eset	3,05176E-05	6,10352E-05	9,15527E-05	0,00012207	15	32 768	0,000262

$$\frac{1}{fclk} * 2^{bitszám} = Tclk * 2^{bitszám} = Delay \text{ [s]}$$

$$\text{Max. TokenCTR} = 2^{bitszám} - 1$$

11. ábra – Időzítési beállítások

Minden eset első időzítési értékből, ahol cong="00" (pl. 0,25 s) az ábra alján látható egyszerű képlet segítségével számítható a belső számláló, értéke, vagyis ameddig szükséges elszámolunk (pl.: 31 250 000).

A következő feladat az ehhez szükséges Tout Timer számláló bitszélességének meghatározása. Ezután az esetekhez társítunk egy Bitrate Timer számláló bitszélességet. A két belső számlálási határ között három nagyságrendi különbség már teljesen

biztonságos működést eredményez. Ezáltal megkaptuk a maximális bitszélességeket, amely a Tout Timer esetében 25 és a Bitrate Timer esetében 28 bit.

Az egyszerűség kedvéért egységesen a BRT T. egy 28, a Tout T. egy 25 bites számlálóval rendelkezik, ezzel általánosíthatjuk az időzítési beállítást. Konfiguráláskor csak a táblázatban szereplő számlálási értékeket adjuk át a modulnak - ezzel elérve a kívánt időzítéseket. Ezek az értékek csak ajánlások, a felhasználó e határokon belül bármilyen értékeket használhat, viszont figyelmesen kell megválasztania ezeket. A Bitrate Timer modul számláló határának minden esetben legalább két nagyságrenddel nagyobbnak kell lennie, mint a Tout Timer modulban meghatározottnak. A táblázatban szereplő legnagyobb adatsebességet és leggyorsabb újraküldést eredményező paraméterek esetében. A két csomag között eltelt idő 262 µs, amelyhez az újraküldési időzítés 30,52 és 122,07 µs-os intervallumon belül mozog. Így a forgalomgenerátor üzembehelyezése előtt a fenti szabályok betartása mellett testreszabhatjuk az időzítési paramétereket. Ezzel a nagyobb csomagsebességet és rövidebb újraküldési időt eredményező beállítástól, a kisebb csomagsebességet eredményező lassabb újraküldési időzítesig.

Hozzá kell tennünk, hogy ezek kizárólag elméleti értékek, az időzítést a korábban már részletezett rotációs FIFO™ végzi. Ezért nem tudunk pontos értékekről beszélni, csak közelítve tudjuk garantálni a különböző időzítéseket. A valóságban előforduló késleltetés értékének intervallumát a következő módon számíthatjuk:

$$MAX(TokenCTR) * T.clk * 0.5 < delay < MAX(TokenCTR) * T.clk * 0.75$$

5.1 Forgalom profilok

Az élethűnek tűnő forgalmi minták elérése érdekében a generátorban szükségünk van jól megkülönböztethető forgalmi profilok beállíthatóságára. Esetünkben leginkább azon van a hangsúly, hogy az általunk előállított forgalom csomagméretei különbözőek legyenek meghatározott profilok alapján.

Már itt az elején tisztázandó, hogy nem célunk a konkrét szolgáltatások részletes adatelemzése. Egy teljes forgalmi elemzés egy sokkal komplexebb folyamat, amely akár külön kutatási téma lehet – kívül mutat e munka keretein. Ezért a jelenlegi munka során kizárólag csak a közelítésekkel dolgozom. A profilok meghatározásánál szempont volt,

hogy alapvető statisztikai elemezéssel is be lehessen osztályozni egy forgalmat a megnevezett forgalmak (Facebook, Youtube) közé.

Az adathosszakon túl az egyes profilokban az adatcsomagok közötti késleltetések beállításával is érdemes foglalkozni. Ezzel még jobban erősíthetjük a generált forgalom egyediségét. A profilok időzítésével kapcsolatos beállításokról a fejezet végén foglalkozom.

Elsőként két különböző profilt határoztam meg, melyekben öt különböző hasznos adat (payload) méret szerepel. Ez az öt csomagméret fog egymás után ismétlődni a forgalmazás ideje alatt. A maradék két profilban egységesen egy payload méret szerepel, ami az implementációt is nagyban leegyszerűsíti. Ezekkel más típusú forgalmi eseteket szimulálunk.

5.1.1 Böngészés

Ebben a forgalmi profilban úgy határoztam meg a csomagméreteket, hogy egy-két kisebb és több közepes méretű csomag szerepeljen. Ennek célja, hogy egy változatos csomagméretekből felépített, általános internetes böngészés adatforgalmát szimuláljuk. Itt a feltételezett adatokat képek, szövegek, teljes weblapok alkothatják.

A kritériumok alapján az általam kiválasztott adathosszak részletezve a 12. ábrán szerepelnek.

Böngészés

	Eth. header [Byte]	IPv4/header length [Byte]	IPv4/total length [Byte]	TCP header [Byte]	Payload [Byte]	Teljes hossz (eth.,ip,tcp,payload)
SYN	14	20	40	20	0	54
SYN,ACK	14	20	40	20	0	54
ACK	14	20	40	20	0	54
DATA1	14	20	85	20	45	99
DATA2	14	20	74	20	34	88
DATA3	14	20	466	20	426	480
DATA4	14	20	264	20	224	278
DATA5	14	20	267	20	227	281
DATA1...						

12. ábra - böngészés forgalom generátor modell

5.1.2 Chat

A chat profil létrehozásánál a népszerű interneten keresztüli írásos beszélgetés modellezése motivált. Itt több kisebb, és egy-két, az előző profilnál sokkal nagyobb csomagméreteket választottam ki. Az itt meghatározott szegmensméretekkel lefedhetjük az átvitt karakterek, emoji-k, hangulatjelek és matricák által generált csomagokat. A chat felületen átvihető nagyobb méretű tartalmakat, mint képeket, videókat, animált matricákat és GIF-eket nem tartalmazza. A forgalom generátorba implementált adathosszakat a 13. ábrán szereplő táblázatban láthatjuk.

Chat

	Eth. header [Byte]	IPv4/header length [Byte]	IPv4/total length [Byte]	TCP header [Byte]	Payload [Byte]	Teljes hossz (eth.,ip,tcp,payload)
SYN	14	20	40	20	0	54
SYN,ACK	14	20	40	20	0	54
ACK	14	20	40	20	0	54
DATA1	14	20	78	20	38	92
DATA2	14	20	369	20	329	383
DATA3	14	20	74	20	34	88
DATA4	14	20	135	20	95	149
DATA5	14	20	459	20	419	473
DATA1...						

13. ábra – Chat forgalom generátor modell

5.1.3 Video

Egy közel maximális payload hosszal rendelkező profil meghatározása könnyen elképzelhető az internetes videók korában. Ezzel egy az online video nézés által létrehozott adatforgalmat szeretném szimulálni. Az itt meghatározott szegmens méretek a 14. ábrán láthatóak.

Video

	Eth. header [Byte]	IPv4/header length [Byte]	IPv4/total length [Byte]	TCP header [Byte]	Payload [Byte]	Teljes hossz (eth.,ip,tcp,payload)
SYN	14	20	40	20	0	54
SYN,ACK	14	20	40	20	0	54
ACK	14	20	40	20	0	54
DATA1	14	20	1450	20	1410	1464
DATA2	14	20	1450	20	1410	1464
DATA3	14	20	1450	20	1410	1464
DATA4	14	20	1450	20	1410	1464
DATA5	14	20	1450	20	1410	1464
DATA1...						

14. ábra – Online videózás forgalom generátor modell

5.1.4 Maximális csomagméret

Végül, de nem utolsó sorban egy maximális csomagmérettel operáló forgalmi profil létrehozásra esett a választás. A profilban kizárólag 1514 bájtos csomagok találhatóak, mint az a 15. ábrán is látható.

Max. szegmens

	Eth. header [Byte]	IPv4/header length [Byte]	IPv4/total length [Byte]	TCP header [Byte]	Payload [Byte]	Teljes hossz (eth.,ip,tcp,payload)
SYN	14	20	40	20	0	54
SYN,ACK	14	20	40	20	0	54
ACK	14	20	40	20	0	54
DATA1	14	20	1500	20	1460	1514
DATA2	14	20	1500	20	1460	1514
DATA3	14	20	1500	20	1460	1514
DATA4	14	20	1500	20	1460	1514
DATA5	14	20	1500	20	1460	1514
DATA1...						

15. ábra – Maximális szegmens méret forgalom generátor modell

Értelemszerűen minden egyes profilban különböző csomagok közti időzítéseket célszerű alkalmazni, ez a különböző szolgáltatásokhoz tartozó különböző felhasználói aktivitás miatt is szükséges.

A böngészésnél elég kiegyensúlyozott időközönként érkeznek a csomagok.

Chat-elés esetén jóval változóbb ez a trend, ahol a felhasználói aktivitástól függően lehet robbanásszerű sebesség növekedés, de átlagban sokkal nagyobb a csomagok közti késleltetés.

Az online video nézés és a maximális szegmens mérettel operáló szolgáltatások által létrehozott adatforgalmaknál folyamatosan és sűrűn érkeznek a közel maximális méretű csomagok.

Az elméleti feltételezések alapján a 16. ábrán elhelyezkedő táblázatot készítettem. Itt az átlagos csomagok közötti késleltetés meghatározása után, kiszámoltam a már 11. ábráról jól ismert képlet segítségével a forgalom generátor számára szükséges paramétereket.

	Átlagos késleltetés [ms]	Bitrate timer		Tout timer	
		Késleltetéshez szükséges belső számláló értéke	TokenCTR (bitszélesség)	Késleltetéshez szükséges belső számláló értéke	TokenCTR (bitszélesség)
Böngészés	0,32025	40 030	16	8 191	13
Chat	6,3593	794 912	20	131 071	17
Video	0,1552	19 399	15	4 095	12
Max. Szegmens	0,106	13 249	14	2 047	11

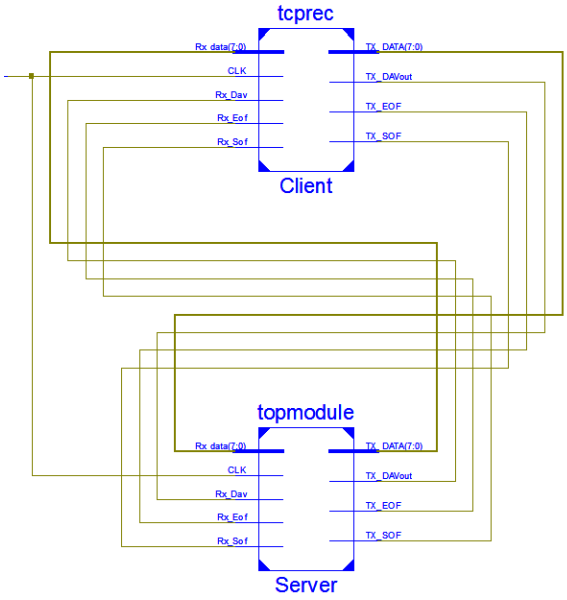
16. ábra - Forgalmi profilok időzítés beállításai

6 Verifikáció

6.1 Építőelemek működésének vizsgálata

A forgalomgenerátor megvalósításában nagy segítséget nyújtott a fejlesztő környezet (Xilinx ISE Design Suite [34]) beépített Isim [35] szoftveres szimulátora. A legfontosabb, hogy a fejlesztés során a teljes rendszert alkotó egységeket külön-külön tudjuk vizsgálni, mielőtt a kész projektet az FPGA-ra töltve ellenőriznénk. Habár FPGA-ra töltés után Chip Scope [36] segítségével akár egyesével is vizsgálhatjuk a modulokat – viszont ez a művelet jelentősen lassabb, körülményesebb. A szimuláció lehetőséget nyújt a futási idő könnyű meghatározására – akár egy adott pillanatban megállításra-, belső jelek és vektorok megjelenítésére.

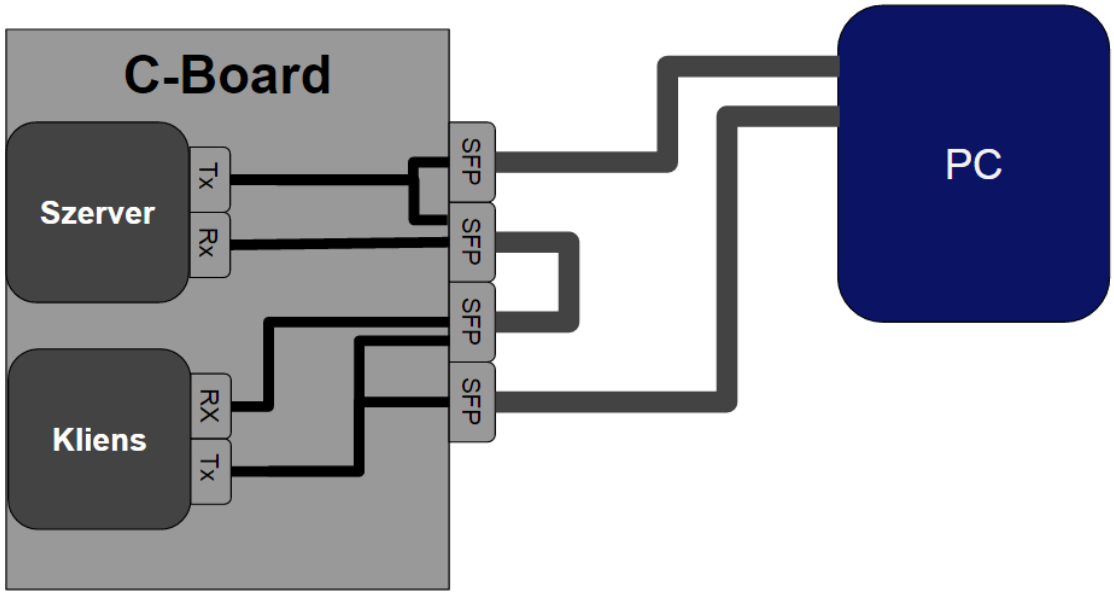
A sokrétű lehetőségek kihasználásával sikerült elérnünk az egyes modulok tervezési szempontjait kielégítő, helyes működést. Mivel az FPGA-s teszt kimeneteléül szolgáló Wireshark-adatsor megjelenítése nem adott elegendő információt a hiba okával kapcsolatban, ezért a végső hardveres teszt elrendezés szimulálására egy FPGA topmodul-ban két forgalomgenerátor példányt közöttünk egymással szembe. Ezt a 17. ábra szemlélteti. Az ezutáni megfigyelésekkel könnyedén be tudtuk korlátozni a hibás modult, vagy esetenként azon belül egy adott kódrészt is.



17. ábra – ISim teszt eérendezés

6.2 Hardveres tesztelés C-Boardon

Az AITIA International Zrt.-től kapott C-Board-hoz [37] készített illesztő kódba két teljes forgalomgenerátor modult példányosítottunk egy klienst és egy szervert, ami a 18. ábrán szerepel. Ezek között az egyetlen különbség, hogy a kliens Bitrate Timer BRAM-ját SYN üzenetekkel töltöttük fel alapértelmezettként, a szerver oldalit üresen hagytuk. Így elértük az automatikus kapcsolat-felépítés kezdeményezését a kliens oldalról, ezzel is életszerűbbé téve a tesztelést. Mind a két példány bemeneteit és kimeneteit a C-Board SFP csatlakozóihoz (1Gbit/s Ethernet) vezetékeztük. A mérés során négy darab ilyen SFP interfész használtunk fel, kettőre a két generátor példány kimeneteit és bemeneteit kötöttük rá. Ezt a két csatlakozási pontot egy rövid UTP kábellel kötöttük össze, itt zajlik a szerver és kliens oldal közötti adatátvitel. A maradék két interfészre kitükröztük mind a két oldal Tx_adat buszát, Wireshark-kal ennek a két interfésznek a forgalmát rögzítve megjelenítettük a teljes szerver és kliens forgalmat.



18. ábra - C-Board implementáció

6.2.1 Teljesítmény

A forgalomgenerátor jelenlegi periféria elemei 1 Gbit/sec-os hálózati sebességet képesek kezelni. Ezt a nyolc bites be- és kimenetek, valamint a 125 MHz-es órajel kombinációjával értük el. Ezen a sebességen az processzor egységei – kritikus az ALU - 1024 socketet képesek külön kezelni. A kapcsolatok számának ellenőrzéséhez a Wireshark conversations funkcióját használtam, ez a 19. ábrán látható. Az ábráról

leolvasható továbbá a kapcsolatonkénti 14 kbit/ sec-os átviteli sebesség is, amelyet egy alapbeállítással értünk.

Ethernet · 1 | IPv4 · 1024 | IPv6 | TCP · 1024 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
209.0.1.18	120	254.0.2.26	80	137	21 k	68	17 k	69	4554	0.134645000	9.246200	14 k	
209.0.1.18	120	254.0.2.27	80	137	21 k	68	17 k	69	4554	0.134647000	9.246200	14 k	
209.0.1.18	120	254.0.2.28	80	137	21 k	68	17 k	69	4554	0.134648000	9.246200	14 k	
209.0.1.18	120	254.0.2.29	80	137	21 k	68	17 k	69	4554	0.134649000	9.246200	14 k	
209.0.1.18	120	254.0.2.30	80	137	21 k	68	17 k	69	4554	0.134651000	9.246199	14 k	
209.0.1.18	120	254.0.2.31	80	137	21 k	68	17 k	69	4554	0.134652000	9.246200	14 k	
209.0.1.18	120	254.0.2.32	80	137	21 k	68	17 k	69	4554	0.134654000	9.246199	14 k	
209.0.1.18	120	254.0.2.33	80	137	21 k	68	17 k	69	4554	0.134661000	9.246193	14 k	
209.0.1.18	120	254.0.2.34	80	137	21 k	68	17 k	69	4554	0.134662000	9.246194	14 k	
209.0.1.18	120	254.0.2.35	80	137	21 k	68	17 k	69	4554	0.134664000	9.246193	14 k	
209.0.1.18	120	254.0.2.36	80	137	21 k	68	17 k	69	4554	0.134665000	9.246193	14 k	
209.0.1.18	120	254.0.2.37	80	137	21 k	68	17 k	69	4554	0.134666000	9.246193	14 k	
209.0.1.18	120	254.0.2.38	80	137	21 k	68	17 k	69	4554	0.134667000	9.246198	14 k	
209.0.1.18	120	254.0.2.39	80	137	21 k	68	17 k	69	4554	0.134668000	9.246199	14 k	
209.0.1.18	120	254.0.2.40	80	137	21 k	68	17 k	69	4554	0.134670000	9.246198	14 k	
209.0.1.18	120	254.0.2.41	80	137	21 k	68	17 k	69	4554	0.134672000	9.246197	14 k	
209.0.1.18	120	254.0.2.42	80	137	21 k	68	17 k	69	4554	0.134674000	9.246196	14 k	
209.0.1.18	120	254.0.2.43	80	137	21 k	68	17 k	69	4554	0.134679000	9.246192	14 k	
209.0.1.18	120	254.0.2.44	80	137	21 k	68	17 k	69	4554	0.134681000	9.246192	14 k	
209.0.1.18	120	254.0.2.45	80	137	21 k	68	17 k	69	4554	0.134682000	9.246192	14 k	
209.0.1.18	120	254.0.2.46	80	137	21 k	68	17 k	69	4554	0.134684000	9.246191	14 k	
209.0.1.18	120	254.0.2.47	80	137	21 k	68	17 k	69	4554	0.134685000	9.246191	14 k	
209.0.1.18	120	254.0.2.48	80	137	21 k	68	17 k	69	4554	0.134686000	9.246192	14 k	
209.0.1.18	120	254.0.2.49	80	137	21 k	68	17 k	69	4554	0.134688000	9.246191	14 k	

19. ábra – Wireshark conversations megjelenítése

Amennyiben ennél több kapcsolatot szeretnénk működtetni ugyanazon az FPGA eszközön belül, a processzor egység párhuzamosítására lesz szükség. A kapcsolatok azonosítására használt adatrész első két bájtját használjuk, ez bőven elég. Szükség van egy dekódoló modulra, ami a 16 bites TCPID felső három bitje alapján el tudja dönteni, hogy melyik processzor számára továbbítsa az üzenet adatait. Egy olyan keretező modul fejlesztése is szükséges, amely képes több processzort kezelni. Több ezer flow esetén viszont az 1Gbit/s-s sávszélesség nem elégséges, ezért ezeknek a perifériáknak 10Gbit/s-s vagy nagyobb sávszélességet is el kell tudniuk érni.

6.2.2 Válaszidők

A válaszidők vizsgálatához beállítottam a Wireshark szűrést úgy, hogy egy socket csomagjai jelenjenek meg. Ez a 20. ábrán látható.

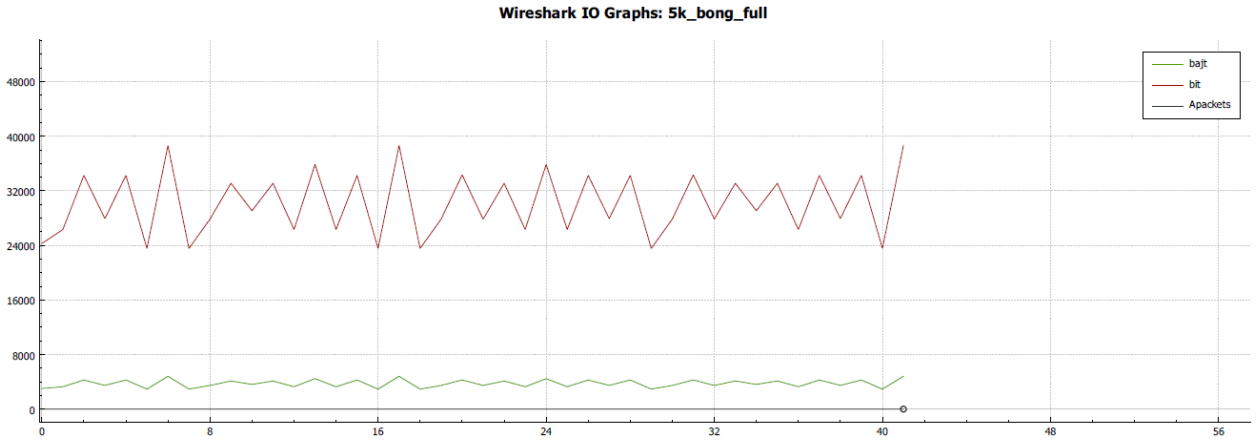
	Time	Source	Destination	Protocol	Length	Info
803	-0.000020	254.0.1.136	209.0.1.18	TCP	66	80 → 120 [SYN] Seq=0 Win=4369 Len=0
806	0.000020	209.0.1.18	254.0.1.136	TCP	66	120 → 80 [SYN, ACK] Seq=0 Ack=1 Win=4369 Len=0
1559	0.000001	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1 Win=4369 Len=0
3273	0.000001	209.0.1.18	254.0.1.136	TCP	134	120 → 80 [<None>] Seq=1 Win=4369 Len=80
3294	0.000045	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=81 Win=4369 Len=0
4664	0.000002	209.0.1.18	254.0.1.136	TCP	480	120 → 80 [<None>] Seq=81 Win=4369 Len=426
4728	-0.000311	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=507 Win=4369 Len=0
6330	0.000061	209.0.1.18	254.0.1.136	TCP	278	120 → 80 [<None>] Seq=507 Win=4369 Len=224
6331	-0.000060	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=731 Win=4369 Len=0
7715	0.000057	209.0.1.18	254.0.1.136	TCP	281	120 → 80 [<None>] Seq=731 Win=4369 Len=227
7727	-0.000052	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=958 Win=4369 Len=0
9107	0.000082	209.0.1.18	254.0.1.136	TCP	134	120 → 80 [<None>] Seq=958 Win=4369 Len=80
9109	0.000001	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1038 Win=4369 Len=0
10450	0.000001	209.0.1.18	254.0.1.136	TCP	88	120 → 80 [<None>] Seq=1038 Win=4369 Len=34
10453	0.000010	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1072 Win=4369 Len=0
11767	0.000028	209.0.1.18	254.0.1.136	TCP	480	120 → 80 [<None>] Seq=1072 Win=4369 Len=426
11768	-0.000026	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1498 Win=4369 Len=0
13131	-0.000035	209.0.1.18	254.0.1.136	TCP	278	120 → 80 [<None>] Seq=1498 Win=4369 Len=224
13148	0.000001	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1722 Win=4369 Len=0
14572	0.000017	209.0.1.18	254.0.1.136	TCP	281	120 → 80 [<None>] Seq=1722 Win=4369 Len=227
14589	-0.000013	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1949 Win=4369 Len=0
16042	0.000080	209.0.1.18	254.0.1.136	TCP	134	120 → 80 [<None>] Seq=1949 Win=4369 Len=80
16045	-0.000079	254.0.1.136	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=2029 Win=4369 Len=0

20. ábra – Általános teszt eredmény Wire shark-ban

Sajnos a nem minden esetben pontos sorrendben jeleníti meg a csomagokat, ezt a két interfészről való adatgyűjtés okozza. A hálózati kártya és az operációs rendszer késleltetései alapján csak közelítéssel tudunk beszélni a csomagok közti késleltetésről. Így az elméleti számításokkal közel megegyezően átlagosan 1-80 µsec. a nyugtázás utáni következő csomag érkezésének ideje. A késleltetés ingadozását az Inbufferben felhasznált FIFO és a Bitrate Timerben található memóriabeli elhelyezkedés okozza. A nyugták érkezésének ideje ennél kicsit kevesebb, de azonos nagyságrendbe tartoznak.

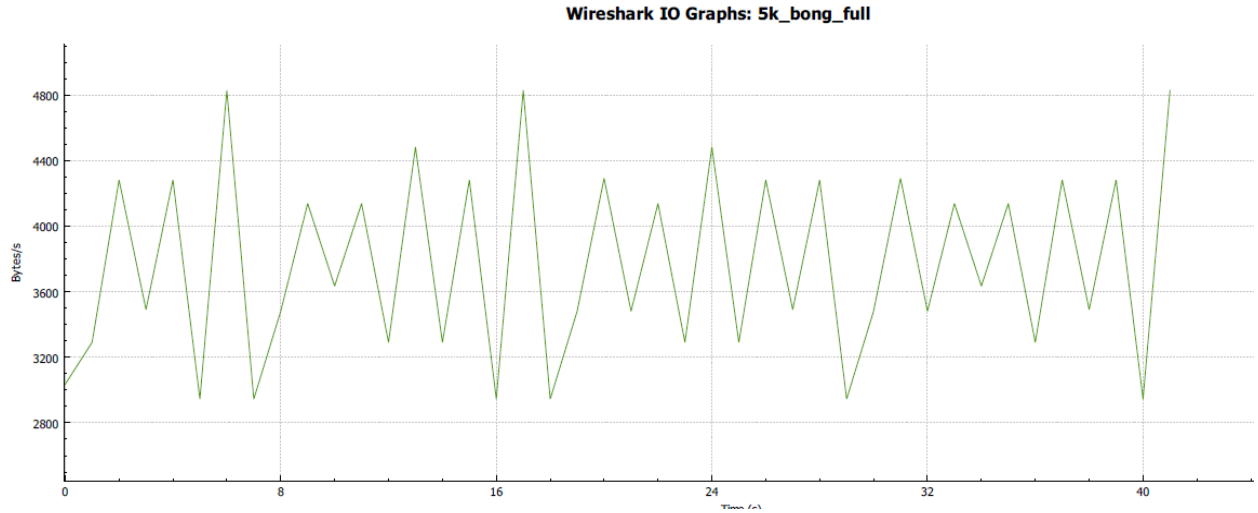
6.2.3 Átviteli sebesség (I/O gráf)

Az átviteli sebesség szemléltetésére a Wireshark „I/O graph” funkcióját használtam. Ez lehetőséget nyújt csomag/s, bájt/s és bit/s függvények kirajzolására, akár egy kapcsolatot forrásul véve. Kiválasztva egyet a legtöbb adat átvitelét lebonyolító kapcsolatok közül, általánosan a 21. ábrán látható diagram-típust kaptam; a számértékek a mérések során az ábrán látható módon szóródtak.



21. ábra – Wire Shark I/O gráf

Értelemszerűen a bit/s és bájt/s közötti kapcsolat miatt kizárólag csak nagyságrendi különbség van a két diagram között, ezért a látványosabb szemléltetés miatt kiválasztottam bájt alapút. Ebből kiválóan látszik a működés helyessége, felismerhető a 22. ábráról a TCP közismertnek mondható fűrészfog alakú sebességgörbéje.



22. ábra – Adatsebesség diagram

6.2.4 Forgalmi profilok

A korábban a forgalmi profilokkal foglalkozó fejezetben taglaltak alapján elvégeztünk még további, az FPGA működését vizsgáló teszteket. Mindegyiknél a vártaknak megfelelő eredményt kaptunk. Ezért is most csak kettőt ismertetek részletesen a két alapvető – változó csomagméretű (böngészés, chat) és állandó csomagméretű (video, max. szegmens) – típus közül. Az elsőből a chat-et a másodikból a maximális szegmens mérettel operáló profilt választottam ki.

A chat profil egy kapcsolatának csomagjai a 23. ábrán láthatóak Az ábrán láthatóak a változó csomagméretek, és a köztük eltelt idő. Itt továbbra sem tudunk komoly következtetést levonni a plusz késleltetést okozó operáció rendszer és hálózati kártya miatt. Emellett a Wireshark hibái miatt nem teljesen korrektek az időbélyegek. Azonban azt látjuk, hogy a várakozásoknak eleget téve a µs-os nagyságrendbe esnek a késleltetések.

Time	Source	Destination	Protocol	Length	Info
0.000009	209.0.1.18	254.0.3.39	TCP	92	120 → 80 [<None>] Seq=497 Win=4369 Len=38
0.000008	254.0.3.39	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=535 Win=4369
0.000004	209.0.1.18	254.0.3.39	TCP	383	120 → 80 [<None>] Seq=535 Win=4369 Len=329
-0.000219	254.0.3.39	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=864 Win=4369
0.000001	209.0.1.18	254.0.3.39	TCP	88	120 → 80 [<None>] Seq=864 Win=4369 Len=34
0.000001	254.0.3.39	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=898 Win=4369
0.000001	209.0.1.18	254.0.3.39	TCP	149	120 → 80 [<None>] Seq=898 Win=4369 Len=95
-0.000641	254.0.3.39	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=993 Win=4369
0.000001	209.0.1.18	254.0.3.39	TCP	473	120 → 80 [<None>] Seq=993 Win=4369 Len=419
-0.000132	254.0.3.39	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1412 Win=4369
0.000001	209.0.1.18	254.0.3.39	TCP	92	120 → 80 [<None>] Seq=1412 Win=4369 Len=38
0.000001	254.0.3.39	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1450 Win=4369
0.000002	209.0.1.18	254.0.3.39	TCP	383	120 → 80 [<None>] Seq=1450 Win=4369 Len=329
0.000002	254.0.3.39	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1779 Win=4369
0.000001	209.0.1.18	254.0.3.39	TCP	88	120 → 80 [<None>] Seq=1779 Win=4369 Len=34
0.000002	254.0.3.39	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1813 Win=4369

23. ábra – Forgalomgenerátor Chat profil forgalma

A 24. ábrán egy általam gyűjtött létező internetes chat szolgáltatás által generált forgalmának egy részlete szerepel. Szinte egyből észrevehető az payload hosszak és az időzítések között fennálló hasonlóság. Ez nem véletlen, ezzel igazolhatjuk, hogy egy jó modellt alkottunk a chat profil tekintetében. Azonban a mi egyszerűsített modellünkből fakadóan vannak nyilvánvaló különbségek is, mint például az, hogy eredeti összevont nyugtákat alkalmaz.

Time	Source	Destination	Protocol	Length	Info
0.000134	31.13.84.36	192.168.100.3	TLSv1.2	88	Application Data
0.005658	31.13.84.36	192.168.100.3	TLSv1.2	92	Application Data
0.000076	192.168.100.3	31.13.84.36	TCP	54	61690 → 443 [ACK] Seq=14475 Ack=17394 Win=657 Len=0
0.000427	31.13.84.36	192.168.100.3	TCP	64	443 → 61690 [ACK] Seq=17394 Ack=14475 Win=2037 Len=0
0.000001	31.13.84.36	192.168.100.3	TLSv1.2	88	Application Data
0.043969	192.168.100.3	31.13.84.36	TCP	54	61690 → 443 [ACK] Seq=14475 Ack=17428 Win=657 Len=0
0.021300	31.13.84.36	192.168.100.3	TLSv1.2	305	Application Data
0.000002	31.13.84.36	192.168.100.3	TLSv1.2	85	Application Data
0.000123	192.168.100.3	31.13.84.36	TCP	54	61690 → 443 [ACK] Seq=14475 Ack=17710 Win=655 Len=0
0.000209	31.13.84.36	192.168.100.3	TLSv1.2	488	Application Data
0.000001	31.13.84.36	192.168.100.3	TLSv1.2	85	Application Data
0.000050	192.168.100.3	31.13.84.36	TCP	54	61690 → 443 [ACK] Seq=14475 Ack=18175 Win=654 Len=0
0.377371	192.168.100.3	31.13.84.36	TLSv1.2	348	Application Data
0.000106	192.168.100.3	31.13.84.36	TLSv1.2	865	Application Data
0.014398	31.13.84.36	192.168.100.3	TLSv1.2	88	Application Data
0.043392	31.13.84.36	192.168.100.3	TCP	64	443 → 61690 [ACK] Seq=18209 Ack=15580 Win=2043 Len=0

24. ábra – Internetes Chat applikáció forgalma

Végezetül a maximális szegmensmérettel rendelkező profilunk által lebonyolított forgalom adatait (25. ábra) hasonlítjuk össze a 26. ábrán látható internetes video megosztó alkalmazás forgalmával.

Time	Source	Destination	Protocol	Length	Info
0.000000	254.0.3.1	209.0.1.18	TCP	66	80 → 120 [SYN] Seq=0 Win=4369 Len=0
0.023090	209.0.1.18	254.0.3.1	TCP	66	120 → 80 [SYN, ACK] Seq=0 Ack=1 Win=4369 Len=0
0.000073	254.0.3.1	209.0.1.18	TCP	66	[TCP Spurious Retransmission] 80 → 120 [SYN] Se
-0.109208	254.0.3.1	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1 Win
0.131089	209.0.1.18	254.0.3.1	TCP	1514	120 → 80 [<None>] Seq=1 Win=4369 Len=1460
-0.000077	254.0.3.1	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=1461
0.000007	209.0.1.18	254.0.3.1	TCP	1514	120 → 80 [<None>] Seq=1461 Win=4369 Len=1460
-0.000255	254.0.3.1	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=2921
0.000016	209.0.1.18	254.0.3.1	TCP	1514	120 → 80 [<None>] Seq=2921 Win=4369 Len=1460
0.000003	254.0.3.1	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=4381
0.000001	209.0.1.18	254.0.3.1	TCP	1514	120 → 80 [<None>] Seq=4381 Win=4369 Len=1460
-0.000234	254.0.3.1	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=5841
0.000269	209.0.1.18	254.0.3.1	TCP	1514	120 → 80 [<None>] Seq=5841 Win=4369 Len=1460
-0.000273	254.0.3.1	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=7301
0.000001	209.0.1.18	254.0.3.1	TCP	1514	120 → 80 [<None>] Seq=7301 Win=4369 Len=1460
-0.000301	254.0.3.1	209.0.1.18	TCP	66	[TCP Keep-Alive] 80 → 120 [ACK] Seq=0 Ack=8761

25. ábra – Forgalom generátor maximális szegmens

A kiragadott részleteken ismét szembetűnő a hasznos adathosszak, valamint a késleltetések közötti alapvető hasonlóság. Ezekkel a méréssorozatokkal belátjuk, hogy ez a forgalomgenerátor modellünk a követelményeknek megfelelő mértékben képes jól szimulálni a nagy mennyiségű, gyors adatátvitelt igénylő szolgáltatásokat. A leglényegesebb különbség azonban az, hogy a valós alkalmazás ACK üzenetekben továbbítja a hasznos adatait, ezzel egy még hatékonyabb átvitelt megvalósítva. Ezt értelemszerűen mi is elérhetnénk a generátor kisebb változtatásaival. Ebben az esetben eltérnék az általánosított implementációnktól.

Time	Source	Destination	Protocol	Length	Info
0.001090	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=1461 Ack=1 Win=3626 Len=1460
0.000047	10.0.0.60	23.6.113.19	TCP	54	54224 → 80 [ACK] Seq=1 Ack=2921 Win=461 Len=0
0.001133	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=2921 Ack=1 Win=3626 Len=1460
0.001104	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=4381 Ack=1 Win=3626 Len=1460
0.000208	10.0.0.60	23.6.113.19	TCP	54	54224 → 80 [ACK] Seq=1 Ack=5841 Win=467 Len=0
0.000886	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=5841 Ack=1 Win=3626 Len=1460
0.001260	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=7301 Ack=1 Win=3626 Len=1460
0.000054	10.0.0.60	23.6.113.19	TCP	54	54224 → 80 [ACK] Seq=1 Ack=8761 Win=467 Len=0
0.001946	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=8761 Ack=1 Win=3626 Len=1460
0.001265	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=10221 Ack=1 Win=3626 Len=1460
0.000028	10.0.0.60	23.6.113.19	TCP	54	54224 → 80 [ACK] Seq=1 Ack=11681 Win=473 Len=0
0.001101	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=11681 Ack=1 Win=3626 Len=1460
0.001319	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=13141 Ack=1 Win=3626 Len=1460
0.000047	10.0.0.60	23.6.113.19	TCP	54	54224 → 80 [ACK] Seq=1 Ack=14601 Win=473 Len=0
0.000947	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=14601 Ack=1 Win=3626 Len=1460
0.001095	23.6.113.19	10.0.0.60	TCP	1514	80 → 54224 [ACK] Seq=16061 Ack=1 Win=3626 Len=1460

26. ábra – Video megosztó alkalmazás forgalma

6.3 Az FPGA kihasználtsága

A kódunk FPGA-n lévő erőforrások kihasználásáról jó és részletes információt nyújt számunkra a Xilinx ISE Design Summary funkciója.

Tudjuk, hogy a Xilinx Virtex-5 családba tartozó 110T egy viszonylag komoly kapacitású eszköz. Amint a 27. ábra is mutatja, az illesztő kóddal együtt egy egymással kommunikáló szerver és kliens oldali forgalomgenerátor példány nem telíti túlságosan az FPGA építőelemeit. Okkal gondolkozhatunk el azon, hogy vajon hány ilyen forgalomgenerátor pár fér el egy 110T típusú eszközön.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	5358	69120	7%
Number of Slice LUTs	6635	69120	9%
Number of fully used LUT-FF pairs	3293	8700	37%
Number of bonded IOBs	36	640	5%
Number of Block RAM/FIFO	11	148	7%
Number of BUFG/BUFGCTRLs	3	32	9%

27. ábra – Teljes projekt erőforrás felhasználása

Ahhoz, hogy meg tudjuk határozni hány darab ilyen forgalomgenerátor fér el egy 110T-n a felhasznált regiszterek, LUT-ok és BRAM/FIFO-k számát kell megvizsgálnunk. A 28. ábrán kiragadtam az alábbi fontos paramétereket és külön választottam a kizárólag a forgalom generátort tartalmazó és a teljes illesztőprogrammal együtt szintetizált projektet. Ezeket, ahogy az ábráról is kiderül lefordítottam a Virtex-6 255T típusú FPGA-ra is, hogy látszódjon a még erősebb hardver bevonásából fakadó lehetőség.

	V-5-110T: generátor + illesztővel			V-5-110T: csak a generátor			V-6-255T: generátor + illesztővel			V-6-255T: csak a generátor		
	Felhasznált	Elérhető	ELFÉR	Felhasznált	Elérhető	ELFÉR	Felhasznált	Elérhető	ELFÉR	Felhasznált	Elérhető	ELFÉR
Regiszterek	5358	69120	12,90	3161	69120	21,87	5303	316800	59,74	3181	316800	99,59
LUT-ok	6635	69120	10,42	4874	69120	14,18	6841	158400	23,15	5029	158400	31,50
BRAM/FIFO	11	148	13,45	9	148	16,44	13	516	39,69	13	516	39,69

28. ábra – Erőforrás felhasználás

A 28. ábrán szereplő táblázatból jól látszik, hogy a LUT-ok felhasználásának száma korlátozza több forgalomgenerátor példányosításának lehetőségeit. Elméletileg egy Virtex-5 110T-n tíz generátor példányt hozhatunk létre. Ezzel a tízezres nagyságrendbe tartozó flow számot is elérhetünk, ezáltal jóval felülmúlva a célként

kitűzött több százas, ezres mennyiséget. Oda kell figyelni viszont arra, hogy a valóságban minél jobban telítünk egy FPGA-t annál nehezebb dolga van a fordítónak a vezetékezésnél, ezáltal nem biztos, hogy kihasználható a teljes vagy közel teljes kapacitás.

7 Összefoglalás

Ez a dolgozat egy olyan TCP-stack kialakításának követelményeit, tervezési lépéseit, implementációját és verifikációját mutatja be, ami a nagysebességű hálózatok tömeges tesztelésekor ad megoldást a TCP-forgalom generálásának hardveres gyorsítására.

A feladat követelményeinek megismerésével együtt áttekintettem a kapcsolódó irodalmat, és a dolgozatban bemutattam az LTE maghálózat felépítését a várható forgalmi mintákból adódó tesztelési igényeket. Mivel az életszerű tömeg-teszteléshez TCP-forgalom generálására is szükség van, bemutattam, hogy FPGA-alapú hardveres gyorsítással lehetséges a korábbinál több nagyságrenddel nagyobb, 1024 párhuzamos folyamszámú TCP-forgalom generálása egyetlen berendezéssel, akár linkenként 10Gbit/s hálózati sebességen is.

Mivel az FPGA fizikai korlátai és a kívánt forgalmi mennyiség generálása nem lehetséges a nyers erő módszerével, ezért a következő újításokat kellett bevezetnem:

- A TCP szabvány alapos megvizsgálásakor olyan lehetőségeket kerestem és találtam, amelyekkel úgy egyszerűsíthető a TCP-stack megvalósítása, hogy a feladat környezetében ez ne sértse a TCP-től elvárt hálózati viselkedéssel kapcsolatos alapelveket: átalakítottam slow-start, congestion avoidance, elhagytam a fast retrnsmit és a fast recovery algoritmusokat.
- A megvalósítást modularizáltam, az egyes funkciókhoz speciális állapotgépeket terveztem úgy, hogy azok a funkcionális igényeket és az időkritikus követelményeket is teljesítsék.
- A TCP stack példányosításánál kézi optimalizációs módszereket is kellett használnom az FPGA-n belül.

A hardveresen gyorsított, FPGA-alapú TCP-stack-et szimulált és valós környezetben is validáltam; vizsgálataim eredményét is közzétettem jelen dolgozatban.

Irodalomjegyzék

[1] BME-VIK, TMIT tanszék:A LAN/MAN interfészek szállítási rétege (TCP rész), http://alpha.tmit.bme.hu/meresek/lantcp.htm (2016. Augusztus)

[2] Xilinx: Field Programmable Gate Array (FPGA) ismertetése, definíciója, http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm (2016. Szeptember)

[3] Xilinx: Virtex 5 FPGA család dokumentációja: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf (2015. Augusztus)

[4] Network Workin Groupe: RFC4614 TCP szabvány leírása, https://tools.ietf.org/html/rfc4614 (2016.)

[5] Cisco: Visual Networkin Index: Global Mobile Data Traffic Forecast Update, 2015-2020 White Papper, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html (2016. Február)

[6] International Telecommunication Union (ITU): UMTS Forum Report 44 Mobile traffic forecasts 2010-2020 : http://groups.itu.int/Portals/17/SG5/WP5D/2-3%20UMTS%20Forum%20presentation%20at%20IMT-%20WS%20at%20AWG%20210311_final_v1.pdf (2011. Január)

[7] Internet Society: Global Internet Report 2015- „Mobil Evolution and Development of the Internet”, http://www.internetsociety.org/globalinternetrep1ort/assets/download/IS_web.pdf (2015.)

[8] Network Sorcery Inc.: UDP, User Datagram Protocol leírás- http://www.networksorcery.com/enp/protocol/udp.htm (2012.)

[9] Wikipedia: LTE, https://hu.wikipedia.org/wiki/LTE (2016. Szeptember)

[10] 3GPP: The Evolved Pocket Core, http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core (2016. Szeptember)

[11] Wikipedia: HSDPA: https://hu.wikipedia.org/wiki/HSDPA (2015. Október)

[12] 3GPP: IP Multimedia Core Network Subsystem, http://www.3gpp.org/technologies/keywords-acronyms/109-ims (2016. Szeptember)

[13] tutorialspoin.com: LTE Network Architecture-Packet Data Network (PDN), https://www.tutorialspoint.com/lte/lte_network_architecture.htm (2016. Szeptember)

[14] 3GPP: TS 29.281-GPRS Tunneling Protocol for User plane (GTPv1-U), http://www.arib.or.jp/IMT-2000/V720Mar09/5_Appendix/Rel8/29/29281-800.pdf (2012. December)

[15] Varga Pál és Olaszi Péter: „LTE core network testing using generated traffic based on models from real-life data", IEEE ANTS, Chennai, India, 2013, https://www.researchgate.net/publication/259464868_LTE_core_network_testing_using_generated_traffic_based_on_models_from_real-life_data (2014 Március)

[16] Polaris, "LTE EPC Load Tester," Termék leírás, http://www.polarisnetworks.net/support-load-tester.html (2013. Augusztus)

[17] Shenick, "Testing LTE/4G - Evolved Packet Core with TeraVM," Shenick megoldásának kivonata, http://www.shenick.com/media/files/LTE EPC testing with TeraVMv061213.pdf (2013. Június)

[18] IXIA, "IxLoad Access," Dokumentáció, http://www.ixiacom.com/pdfs/datasheets/ixload lte access.pdf (2013. Május)

[19] ng4T, "NG40-EPC-S1," Adatlap, http://www.ng4t.com/Datasheets/datasheet NG40-EPC-S1.pdf (2013.)

[20] MobileMetrics, "Torrent 6100 LTS," Termék leírás, http://www.mobilemetrics.net/lte-test.htm (2012.)

[21] Aricent, "Total Testing for LTE," Termék leírás, http://www.aricent.com/pdf/Aricent Solution Brief LTE Testing.pdf (2013.)

[22] EXFO, "Nethawk EAST EPC for Evolved Packet-Core Testing," Termék leírás, (2011.)

[23] NS-3, "ns-3 Discrete-event Network Simulator", http://www.nsnam.org/g (2013.)

[24] NetFPGA: SUME, Adatlap, http://netfpga.org/site/#/systems/1netfpga-sume/details/ (2016. Október)

[25] NetFPGA: CML, Adatlap, http://netfpga.org/site/#/systems/2netfpga-1g-cml/details/ (2016. Október)

[26] NetFPGA: 10G, Adatlap, http://netfpga.org/site/#/systems/3netfpga-10g/details/ (2016. Október)

[27] AITIA: 1G, Adatlap, http://netfpga.org/site/#/systems/4netfpga-1g/details/ (2016. Október)

[28] AITIA: C-GEP, Adatlap, http://www.fpganetworking.com/160616/c_gep/index.html (2016. Október)

[29] AITIA: C-Board, Adatlap, http://www.fpganetworking.com/160616/c_board/index.html (2016. Október)

[30] AITIA: SGA-10GED, Adatlap, http://www.fpganetworking.com/160616/10ged/index.html (2016. Október)

[31] WireShark: Hálózati adatforgalmo rögzítésére szolgáló szoftver, https://www.wireshark.org/#learnWS (2016. Október)

[32] Nagy Balázs, Kotnyek Bence, Róth Ádám: Nagymennyiségű TCP kapcsolat implementáció FPGA alapú eszközökre (2016. Augusztus)

[33] Xilinx: BRAM overview, https://www.xilinx.com/products/intellectual-property/block_ram.html (2016. Október)

[34] Xilinx: ISE Design Suite, https://www.xilinx.com/products/design-tools/ise-design-suite.html (2016. Október)

[35] Xilinx: ISE Simulator (ISim), https://www.xilinx.com/products/design-tools/isim.html (2016. Október)

[36] Xilinx: Chip Scope Pro 11.4 Software and Cores, https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/chipscope_pro_sw_cores_ug029.pdf (2016. Október)

[37] Plósz Sándor, Moldován István, Dr. Varga Pál, Kántor László: „Depandability of a Network Monitorin Hardware", IEEE Conference on Dependability (DEPEND), 2010, https://www.researchgate.net/publication/232632738_Dependability_of_a_Network_Monitoring_Hardware (2010.)