BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Híradástechnikai Tanszék

Lencse Gábor:

Kommunikációs rendszerek hatékony szimulációjának egyes kérdései

Ph.D. értekezés

Témavezető:
dr. Németh Gábor

Konzulens:
dr. Pongor György

Budapest, 2000.

Tartalomjegyzék

1. Bevezetés

A távközlési piac liberalizációjával kialakuló versenyhelyzet Magyarországon is megköveteli, hogy a szolgáltatók optimális erőforrás kihasználtságra törekedve költség hatékonyan biztosítsák a megfelelő minőségi (QoS) paramétereket. Új hálózatok építése, meglevő hálózatok bővítése, új szolgáltatások bevezetése esetén előre meg kell győződni arról, hogy kielégíthetők-e a minőségi követelmények. Hálózatok üzemeltetése során is elengedhetetlen, hogy a forgalmi trendeket figyelembe véve előre jelezhető legyen a hálózat valamely szűk keresztmetszete, és így a szükséges bővítéssel a minőségromlást meg lehessen előzni.

Kommunikációs rendszerek teljesítőképesség vizsgálatának (Jain, 1991) - az analitikus módszer és a valóságos rendszeren való mérés mellett - nélkülözhetetlen eszköze a diszkrét idejű szimuláció. Ennek két fajtája az idővezérelt és az eseményvezérelt szimuláció, melyek közül az utóbbi az általánosabb, sokkal szélesebb körben alkalmazzák, én is ezt tettem vizsgálataim tárgyává.

Nagy méretű és bonyolultságú rendszerek szimulációjánál számos megoldandó probléma merül fel. Ezek közül választottam ki néhányat:

- nagy erőforrás igény mind memória-, mind számításigény tekintetében
- nagy eseményszámnál a jövőbeli események halmaza (Future Event Set) méretének növekedésével a halmaz műveleteinek időigénye jelentősen megnőhet
- a statisztikagyűjtéshez - különösen ritka események esetén - igen nagy eseményszám és ezért hosszú végrehajtási idő lehet szükséges.

Az elsőnél a párhuzamosítás lehetőségeivel foglalkozom, egy kevéssé ismert, de kedvező tulajdonságai miatt ígéretes szinkronizációs módszer, a statisztikai szinkronizáció (Statistical Synchronisation Method) (Pongor, 1992) olyan továbbfejlesztését mutatom be amely időben változó rendszerek párhuzamos szinkronizációs módszereként is alkalmazható (SSM-T) (Lencse, 1998). Az általam kidolgozott módszer tulajdonságait vizsgálom szimulációval és analitikus úton is, a felmerülő új kérdésekre keresek választ.

A másodiknál a FES tárolására alkalmazható fontosabb adatszerkezeteket hasonlítom össze a műveleteik lépésszámának aszimptotikus viselkedése szempontjából, mind elméleti vizsgálattal, mind végrehajtási idő méréssel, különböző peremfeltételek mellett.

A harmadik probléma kapcsán egy általam kifejlesztett új eljárást (Traffic Flow Analysis) mutatok be, amely a sok mintát igénylő statisztikagyűjtés helyett eleve statisztikákkal működő módszer: a szimuláció és numerikus illetve analitikus módszerek kombinációja.

A három kérdéskörrel három külön fejezetben foglalkozom, fejezetenként irodalmi áttekintés és a kérdéskörbe való bevezetés után bemutatom saját vizsgálati módszereimet és új kutatási eredményeimet.

Az utolsó fejezetben közlöm eredményeim általam ismert alkalmazását. Munkámat irodalomjegyzékkel zárom, melyben felsorolom az értekezésemben hivatkozott összes külső és saját publikációt.

2. A párhuzamosítás kérdései

2.1. A párhuzamos szimulációban született korábbi eredmények

A párhuzamos diszkrét idejű szimuláció (parallel discrete-event simulation, PDES) témájában jó áttekintést ad (Fujimoto, 1990). Ennek a cikknek az alapján mutatom be a párhuzamosítás problémáját és a megoldására alkalmazott két közismert szinkronizációs módszert.

2.1.1. A feladat

Diszkrét idejű szimuláció esetén a vizsgált rendszer állapotváltozásai diszkrét időpillanatokban mennek végbe, vagy legalábbis így vesszük őket figyelembe. A rendszer modellje egy *esemény* hatására kerül egyik állapotból egy másik állapotba. Egy kommunikációs hálózat szimulációja során esemény lehet például egy csomag megérkezése egy csomópontba, egy csomag adásának megkezdése, egy elem meghibásodása, stb. Azzal az esettel foglalkozunk, amikor a szimulációs modell *aszinkron*, vagyis az események nincsenek egy globális órához szinkronizálva, hanem tetszőleges időpillanatban bekövetkezhetnek, azaz nem idővezérelt, hanem eseményvezérelt szimulációról van szó. A rendszer különböző pontjain bekövetkező eseményeket szeretnénk párhuzamosan végrehajtani, de ez *szinkronizációs* problémákat vet fel, amit az eseményvezérelt diszkrét idejű szimuláció algoritmusának a segítségével lehet bemutatni. Az algoritmus a következő:

```
inicializálás, benne bizonyos események felrakása FES-be;
repeat
  első esemény levétele FES-ből, onnan törlése;
  MOST := a levett esemény ideje;
  esemény feldolgozása, benne esetleg újabb események
    felrakása FES-be, de legalább MOST+0 időre;
until  kiürült a FES, vagy MOST>egy beolvasott határ,
       vagy valami miatt le kell állni
```

A szimuláció fő ciklusában mindig a legkorábbi, azaz a legkisebb időbélyegű eseményt kell kivenni a jövőbeli események halmazából (FES). A modell idő (más szóhasználattal virtuális vagy szimulációs idő) felveszi a FES-ből kivett esemény időbélyegének értékét. A kivett esemény feldolgozása során újabb esemény keletkezhet, amelynek időbélyegéről általános esetben csak annyit tudunk, hogy nem kisebb a jelenlegi modell időnél. Éppen ebből adódik a párhuzamosítás alapvető problémája: legyen a legkorábbi esemény E_1 időbélyege t_1, a második legkorábbi esemény E_2 időbélyege t_2. Végrehajtható-e az E_2 esemény feldolgozása az E_1 feldolgozásával párhuzamosan? Mi a feltétele annak, hogy végrehajtható legyen? Ez az az alapkérdés, amire a különböző PDES szinkronizációs módszerek eltérő választ adnak. A klasszikus módszerek alapvetően két fő osztályba sorolhatók. A konzervatív megközelítés

szerint E_2 csak akkor hajtható végre E_1 végrehajtásával párhuzamosan, ha biztosított, hogy ennek következtében nem fog megsérülni a kauzalitás. Az optimista megközelítés ilyen megkötést nem tesz, hanem a kauzalitás tényleges megsértését kezeli megfelelő módon. A két megközelítés tárgyalásához vezessünk be néhány alapfogalmat.

2.1.2. A logikai processzek módszere

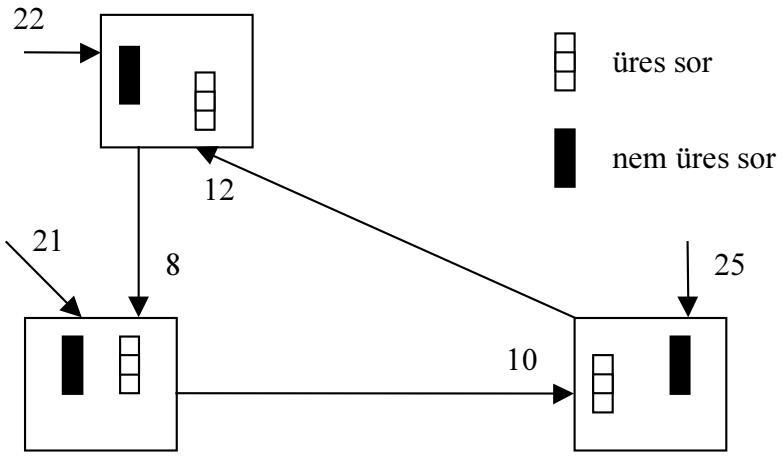
Ennél a megközelítésnél a szimulációt N darab logikai processzre bontjuk: LP_0, LP_1, …, LP_{N-1}. $Clock_i$ az *i*. processz szimulációja során elért modellbeli időt jelenti; egy esemény feldolgozásakor a processz órája automatikusan felveszi az esemény időbélyegének az értékét. Ha LP_i a szimuláció során küldhet üzenetet LP_j-nek, akkor azt mondjuk, hogy egy *link* vezet LP_i-ből LP_j-be. A szimulátor állapotát a logikai processzek állapotváltozóinak összessége adja. Az állapotváltozók nem lehetnek megosztottak, minden logikai processz csak a saját állapotváltozóit látja, más processz állapotváltozóihoz csak üzenetküldésen keresztül férhet hozzá. Az üzeneteket azok küldője időbélyeggel látja el. A szimuláció kielégíti a *lokális kauzalitási feltétel*t, ha minden LP nemcsökkenő időbélyeg sorrendben dolgozza fel az eseményeket. A lokális kauzalitási feltétel elégséges, de nem szükséges feltétele annak, hogy a szimulációban ne forduljon elő kauzalitási hiba. Azért nem szükséges, mert egy LP-n belül két esemény lehet független, így nem sérti meg a kauzalitást, ha feldolgozásuk sorrendje nem időbélyeg szerinti.

2.1.3. A konzervatív megközelítés

A konzervatív megközelítés szerint a szimuláció csak *biztonságos* eseményeket dolgozhat fel. Egy t_i időbélyegű E_i esemény akkor biztonságos, ha garantált, hogy az LP már biztosan nem fog t_i-nél kisebb időbélyegű eseményt kapni, mert ekkor garantáltan nem fogja megsérteni a lokális kauzalitási feltételt. Amelyik LP nem rendelkezik biztonságos eseménnyel, annak a végrehajtása blokkolódik. Ez holtponthoz (deadlock) is vezethet megfelelő elővigyázatosság hiányában. Miért? Egy korai konzervatív algoritmus (Chandy and Misra, 1979) statikusan specifikálja a processzek lehetséges kommunikációját kifejező linkeket. Az üzenetek sorrendje linkenként kötelezően monoton nemcsökkenő, így egy linken az utolsó megérkezett üzenet időbélyege alsó korlát a később érkező üzenetek időbélyegére. A beérkező üzeneteket linkenként FIFO sorrendben tárolják (ami természetesen időbélyeg sorrendet is jelent egyben). Egy bejövő linkhez tartozó óra a link sorában elöl álló esemény időbélyegével egyenlő, ha a sor nem üres, ha pedig üres, akkor a linken legutoljára érkezett esemény időbélyegének felel meg. Az 1. ábra egy holtpont helyzetet mutat: a három processz mindegyike várakozni kényszerül az üres bemenő sorok és az időbélyegük miatt, bár kívülről kaptak feldolgozásra váró üzeneteket.

A holtpont elkerülésére úgynevezett *null üzenetek*et alkalmaznak, amelyek nem részei a szimulációnak, csak egy ígéretet jelentenek: ha LP_A egy t_i időbélyegű null üzenetet küld LP_B-nek, akkor ezzel vállalja, hogy ezután már nem fog t_i-nél kisebb időbélyegű üzenetet küldeni. (Ilyen ígéret természetesen csak az adott szimuláció tulajdonságaiból következően adható.) A null üzenetek küldése nagy többletmunkát (overhead) jelenthet, kidolgoztak olyan megoldást

is, ahol csak kérésre küldenek null üzenetet (Misra, 1986). Egy másik megoldás az, hogy megengedik a holtpont létrejöttét, viszont detektálják és megszüntetik azt (Chandy and Misra, 1981).



1. ábra Holtpont kialakulása konzervatív szinkronizáció esetén

A konzervatív megközelítés esetén nagyon fontos fogalom a *lookahead*. Ez azt fejezi ki, hogy milyen távolra látunk a jövőbe: ha LP_i jelenlegi virtuális ideje $\mathbf{Clock_i}$ és $\mathbf{Clock_i+L}$ virtuális időig minden általa generált üzenetet biztosan előre tudunk jelezni, akkor azt mondjuk, hogy a LP_i lookaheadje **L**. Kedvező lookahead tulajdonságokkal rendelkező szimulációk esetén a konzervatív szinkronizációt használó párhuzamos szimulációval jó *gyorsulás* (speed-up) érhető el, ennek hiányában azonban a módszer nem képes kiaknázni a szimulációban egyébként meglévő párhuzamosságot. A kommunikációs rendszerek vizsgálata során oly gyakran alkalmazott Poisson eloszlású tranzakció generálás vagy exponenciális eloszlású kiszolgálás esetén például 0 a lookahead. Ugyanez a helyet preemptív rendszereknél is. Ezekben az esetekben tehát gyorsulás helyett a szinkronizációs módszer által okozott többletmunka miatt lassulást érünk el. Ezektől eltérő esetben is a módszer nagy hátránya az, hogy a konzervatív szinkronizációt használó párhuzamos szimuláció megírása sokkal bonyolultabb feladat, mintha közönséges szekvenciális szimulációt készítenénk.

2.1.4. Az optimista megközelítés

Az optimista megközelítés megengedi, hogy a szimuláció végrehajtása során megsérüljön a kauzalitás, de képes ezt detektálni és megfelelő módon helyrehozni. A legnépszerűbb optimista módszer a Time Warp, ennek lényege a következő: a kauzalitás megsértését akkor detektálja, amikor egy processz az aktuális virtuális idejénél kisebb időbélyegű üzenetet kap egy másik processztől. Ekkor elindul a *rollback* folyamat, aminek a során visszaállítja a processz állapotát (ennek érdekében az állapotot előzőleg rendszeresen menti) és a más processzeknek hibásan kiküldött üzenetek törlését ellen-üzenetek (*anti-messages*) küldésével végzi el, amelyek természetesen szintén rollback folyamatot indíthatnak el. A módszer hatékonyságát kölönböző megoldásokkal igyekeznek javítani, ezek közül érdekességként a *lazy*

*cancellation*t említem meg: ennek alapötlete az, hogy a rollback végrehajtása után sok esetben ugyanazok az üzenetek kerülnek elküldésre, amit már korábban is elküldött és aztán törölt a processz: nem küldi el tehát a kauzalitási hiba detektálása után rögtön az ellenüzeneteket, hanem csak akkor küldi el a *t* virtuális idővel generált ellenüzenetet, ha az újra végrehajtás során az óra már elhagyta *t*-t és az ellenüzenetnek megfelelő eredeti üzenet most nem jött létre. A megoldás az adott szimulációtól függően persze ronthatja is és javíthatja is a hatékonyságot!

Az optimista módszerekkel sok esetben jó gyorsulást értek el, de ennek is vannak árnyoldalai:

- az állapotmentés sok memóriát igényelhet, ami esetleg nem áll rendelkezésre
- ha az állapotmentés átlagos költsége megközelíti az egy esemény feldolgozásának átlagos költségét, akkor már nem várhatunk gyorsulást
- a sok rollback miatt nagyon időigényes lehet a szimuláció nyomkövetése
- nagy odafigyelést igényel a Time Warp szimulációs kernel írása – kis ügyetlenség tönkreteheti a gyorsulást
- dinamikus memóriafoglalás esetén egyáltalán nem triviális probléma az állapotmentés – szimulációs kernelnek legalábbis tudnia kell róla, mit művel a szimulációs kernel felhasználója.

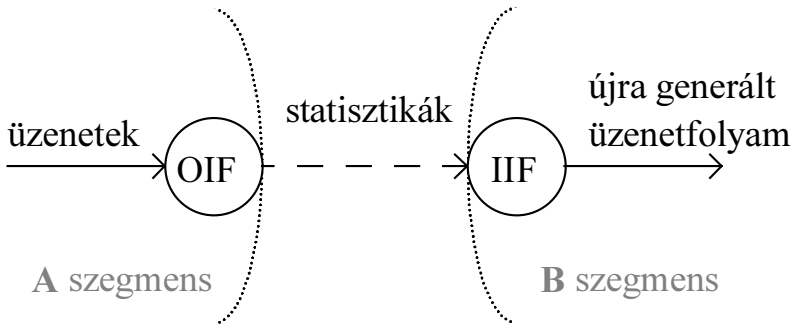
Az állapotmentések költségének csökkentésére hardver támogatást javasolnak: ez természetesen segíthet, de nagy árat kell fizetni érte: általános célú számítógépek helyett a speciálisan optimista szinkronizációt támogató gépek a kis sorozatszám miatt várhatóan sokkal drágábbak lesznek.

2.1.5. Az eredeti statisztikai szinkronizációs módszer

Kevéssé közismert de nagyon szellemes megoldás a Statisztikai Szinkronizációs Módszer (Pongor, 1992). Ennek működése a következő. Más párhuzamos szimulációs módszerekhez hasonlóan a rendszer modelljét – ami többé vagy kevésbé pontos reprezentációja a valóságos rendszernek – szegmensekre osztják, ahol a szegmensek általában a valóságos rendszer funkcionális egységeinek viselkedését írják le. Az egyes szegmensek szimulációjának a végrehajtását külön processzorok végzik. A szegmensek egymás közti kommunikációját különböző üzenetek küldésével és fogadásával lehet kifejezni. Az SSM számára az egyes szegmenseket input és output interfészekkel látják el. Az egy adott szegmensben generált, de egy másik szegmensben feldolgozandó üzeneteket nem továbbítják oda, hanem az *output interfészek* statisztikát gyűjtenek róluk. A szegmensek számára az *input interfészek* generálják a bejövő üzeneteket a megfelelő output interfészek által gyűjtött statisztikák alapján (2. ábra).

Az input és output interfészekkel kiegészített szegmensek szimulációja külön végrehajtható, és statisztikailag helyes eredmények kaphatók. Azonban az egyik szegmensben keletkező üzeneteknek a másik szegmensekben kifejtett hatása **nem** azonos azzal, ami az eredeti modellben volt, így az SSM segítségével végzett szimulációban gyűjtött eredmények nem pontosak. Pontosságuk függ a szegmentálástól, a statisztikagyűjtés pontosságától, a pro-

cesszorok közti statisztikacsere gyakoriságától. Vannak esetek, amikor az SSM egyáltalán nem alkalmazható, ilyen például a digitális áramkörök logikai szimulációja. Ekkor a hagyományos, eseményenkénti szinkronizáció elkerülhetetlenül szükséges.



2. ábra SSM összeállítás

Az SSM a következő kedvező tulajdonságokkal rendelkezik:

- a processzorok kommunikációja az eseményenkénti szinkronizációhoz képest nagyságrendekkel ritkább, ekkor sem cserélnek nagy adatmennyiségeket, így lazán csatolt rendszereken is alkalmazható
- különböző sebességű, sőt architektúrájú processzorok is alkalmazhatók
- már létező moduláris egyprocesszoros szimuláció kódja kevés módosítással vagy anélkül felhasználható
- a gyorsulás közel a processzorok számával arányos (speciális esetekben lineárisnál jobb is elérhető (Pongor, 1994))
- processzoronként a szükséges memóriaméret lényegesen kisebb, mint általában más módszereknél
- dinamikus terhelésmegosztás alkalmazható és az képes nagyon hatékony lenni.

Mindezek a tulajdonságok ígéretessé teszik a módszert, és indokolják tulajdonságainak mélyebb vizsgálatát. Tisztázni kell az alkalmazhatóság kritériumait, a statisztikacsere körülményeit (mi triggereli) és a statisztikagyűjtés, üzenetregenerálás kérdéseit.

Az SSM a rendszer állandósult állapotbeli viselkedésének meghatározására alkalmas. Felmerül a kérdés, hogy alkalmassá tehető-e a rendszerben lévő lassú tranziens vizsgálatára is. Ezekre a kérdésekre adnak választ az SSM-mel kapcsolatos kutatásaim.

2.2. Az SSM továbbfejlesztése és alkalmazhatósági kritériumai

2.2.1. Motiváció

A statisztikai szinkronizáció (SSM) eredetileg publikált formájában csak állandósult állapotban alkalmazható párhuzamos szimuláció szinkronizációs módszereként.

A módszert úgy fejlesztettem tovább, hogy az az időben lassan változó állapotú rendszer vizsgálatára is alkalmas legyen. Az eredeti cikk nem részletezi, hogy meddig kell a statisztikagyűjtést folytatni és pontosan mi triggereli a statisztikacserét. Két fajta statisztikacserét említ: szinkronnak nevezi azt az esetet, amikor "addig késleltetjük a statisztikák cseréjét, amíg mindegyik output interfész be nem fejezi a statisztikagyűjtést, és az új statisztikákat egyszerre kezdik használni az input interfészek" és aszinkronnak hívja azt a megoldást, ahol "amikor egy output interfész elkészül a statisztikagyűjtéssel, az új értékeket rögtön alkalmazni fogja a megfelelő input interfész". A szerző a továbbiakban és későbbi cikkében (Pongor, 1994) az aszinkron megoldással foglalkozik, ennek kedvező tulajdonságaként mutatja ki az automatikus importance sampling-et. Ez értékes tulajdonság, de létének az ára az, hogy az egyes szegmensek virtuális ideje nagy mértékben eltérő. Vannak olyan rendszerek, ahol a szegmensek virtuális idejének eltérése nem jelent problémát, és a módszer kiválóan alkalmazható. A távközlési rendszerek egy nagyon jelentős részének vizsgálatánál azonban ez hátrányos tulajdonság. Mind a klasszikus telefóniában (PSTN), mind az adathálózatoknál közismert tény a forgalom intenzitásának a napon belüli változása. Ezt támasztják alá az adathálózati terhelésvizsgálatban az Elassys Consulting Kft-nél szerzett személyes munkatapasztalataim, valamint a távközlési szolgáltatók díjszabási gyakorlata is. Az SSM továbbfejlesztésének fontos célja tehát a kommunikációs hálózatok vizsgálatára való alkalmasságának fokozása, hiszen ez az alkalmazásának egy nagyon jelentős területe.

2.2.2. A statisztikacsere feltételei

Vizsgáljuk meg először egy két szegmensből álló szimulációs modellnél, hogy mi válthatja ki a statisztikacserét. A (Lencse, 1998a) cikkemben vezettem be az SSM két változatára az SSM-C és az SSM-T jelöléseket. A C a *counter driven* kifejezésből származik, ami azt jelenti, hogy a statisztikacserét az adott mintaszám, vagyis a kívánt statisztikagyűjtési pontosság elérése váltja ki. A T a pedig *time driven* kifejezésből jön, ahol az adott virtuális idő elérése okozza a statisztikacserét. Kutatásaimhoz az eredetileg a BME Híradástechnikai Tanszékén kifejlesztett, azóta nemzetközi együttműködésben továbbfejlődő OMNeT++ diszkrét idejű szimulációs rendszert (Varga, 2000) használtam. Az OMNeT++ lehetővé teszi szimulációk elosztott végrehajtását PVM fölött, ekkor az egyes szegmensek saját eseményhalmazt (FES) használnak, a szegmensek lokális virtuális időinek (LVT) szinkronizálására csak úgy van lehetőség, ha a szinkronizáció (globális) virtuális ideje előre meghatározott. Ez a szinkronizáció "egyoldalas", ami azt jelenti, hogy ha az **A** szegmens üzenetet akar küldeni a **B** szegmensnek az általa előre ismert t_i virtuális időben, akkor ezt előre jelzi a **B** szegmens számára legkésőbb az előző azaz (i-1)-edik **A**-tól **B** felé irányuló üzenet küldésekor. (A legelső üzenet bejelentése a szimuláció indításakor történik.) A szimulátor ilyenkor garantálja, hogy a **B** szegmens lokális virtuális ideje nem fogja elkerülni a t_i-t, míg az **A**-tól jövő üzenet meg nem érkezik. Ez biztosítja, hogy a kauzalitás ne sérüljön meg. Természetesen ebben az esetben az **A** szegmens lokális virtuális ideje messze megelőzheti a **B** szegmensét. Ha az a célunk, hogy a két szegmens lokális virtuális idői "találkozzanak" t_i-ben, akkor mindkét irányban alkalmaznunk kell a szimulátor által nyújtott szinkronizációt. Ezt a megoldást neveztem *laza időszinkronizáció*nak (*loose synchronization*).

Az **A** és **B** szegmensek közötti laza időszinkronizáció formális definíciója a következő:

Legyenek t_1, t_2, … t_i, … t_n szinkronizációs időpontok. Jelölje t_A és t_B az **A** illetve **B** szegmens lokális virtuális idejét. Az **A** és **B** szegmens között laza időszinkronizáció van, ha:

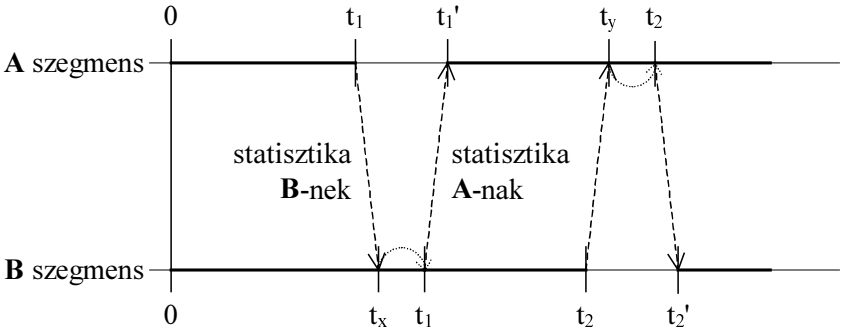
$((t_A < t_i) \Rightarrow (t_B \le t_i)) \wedge ((t_A > t_i) \Rightarrow (t_B \ge t_i)),\ i = 1, 2, ... , n.$

A két szegmens közötti laza időszinkronizáció azt jelenti, hogy egyik sem hagyhat el egy szinkronizációs időpontot, míg a másik el nem érte azt.

A szegmensek t_i-ben kicserélhetik egymással azokat a statisztikáikat, amelyeket t_i előtt gyűjtöttek, és az újakat használhatják a $[t_i, t_{i+1}]$ időintervallumban. A t_i szinkronizációs időpontok megválasztásával biztosítható, hogy az egyik szegmensben t-ben bekövetkező változás hatása a másik szegmenst legkorábban t, legkésőbb t+Δt virtuális időben érje el, ahol:

$$\Delta t \le 2 \min_i (t_{i+1} - t_i)$$

(Bővebb magyarázat a 2.2.3. alfejezetben (4. ábra).) A laza időszinkronizációt használó SSM-T-nek az eseményenkénti szinkronizációval szemben nagy előnye az, hogy a t_i szinkronizációs időpontok között a szegmensek szimulációja egymástól függetlenül végrehajtható.



3. ábra Az SSM-T működésének illusztrációja

A formális definíción túl lássunk most egy egyszerű példát! Az 3. ábrán az **A** és a **B** szegmens első statisztikacseréjének virtuális időpontja t_1. Az ábrán a vékony vízszintes vonalak a szegmenseket végrehajtó processzorok valós idejét (*wall-clock time*) jelzik, a vastag vízszintes vonalak pedig szegmensek virtuális idejét. Mindkét szegmens az előre megállapodott t_1 lokális virtuális időben küldi el a saját statisztikáját a másiknak. A példában a **B** szegmens a $t_x<t_1$ virtuális időben kapja meg az **A** szegmenstől származó statisztikát, ami az eseményhalmazában t_1 időpontra kerül felidőzítésre, és ekkor fogja feldolgozni. Az **A** szegmens végrehajtása pedig t_1-ben felfüggesztésre kerül mindaddig, amíg **B**-től meg nem érkezik a megígért statisztika. A második szinkronizációnál a helyzet teljesen hasonló, azzal az eltéréssel, hogy most a **B** szegmens kényszerül várakozásra. Természetesen a módszer akkor hatékony, ha az idő nagy részében mindkét szegmens végrehajtása folyik és csak kisebb mértékben kell egymásra várniuk. A hatékonyság kérdéseit a 2.3. fejezetben részletesen tárgyalom.

Az SSM-T és a laza időszinkronizáció alapvetően két egymással összekapcsolt (azaz az eredeti, szekvenciális modellben egymásnak üzeneteket küldő) szegmens időszinkronizációját

fejezi ki; az összekapcsolt szegmensek közt páronként más-más szinkronizációs időpontok is választhatók. Ha a szinkronizációs időpontokat az egész rendszerben egységesen választjuk meg, akkor speciális esetként kapjuk a (Pongor, 1992) cikkben szinkronnak nevezett megoldást.

A legegyszerűbb esetben a szinkronizációs időpontokat megválaszthatjuk úgy, hogy köztük egyenlő idő teljen el, azaz, t_i=i*UI, ahol UI az *Update Interval*. Ezt a megoldást alkalmaztam a (Lencse, 1998a) cikkemben. A szinkronizációs időpontok megválasztásával a 2.5. fejezetben részletesen foglalkozom.

A teljesség kedvéért említem meg, hogy az SSM-C módszer pedig megfelel a (Pongor, 1992) cikk aszinkronnak nevezett megoldásának.

Mivel a lokális kauzalitási feltétel biztosított, az általam kidolgozott SSM-T módszer alkalmas párhuzamos diszkrét idejű szimuláció szinkronizációs módszerének.

2.2.3. Az alkalmazhatóság feltételei

Az SSM-T alkalmazhatósági kritériumait először informálisan adom meg, a dőlt betűs részek a bizonyítás során nyerik el formális tartalmukat. A módszer alkalmazható, és értelmes eredményeket ad, ha az alábbi feltételek mindegyike fennáll:

(a) A szimulált rendszer szegmensekre osztható úgy, hogy a szegmensek közötti forgalomban csak az üzenetfolyam statisztikai jellemzői számítanak, az egyes üzenetek maguk a rendszer működése és az általunk megfigyelt kimeneti jellemzők szempontjából nem fontosak.

(b) Az üzenetfolyam statisztikai jellemzőinek közelítésében egy *kis hiba* a kimenetben is *kis hibát* okoz, ami *csak a közelítés hibájának a mértékétől függ*.

(c) A modell paraméterei változhatnak a szimuláció alatt, de a szegmensek közötti üzenetfolyamok statisztikai jellemzőinek változásai *kellően ritkák*.

(d) Mivel egy üzenetfolyam statisztikai jellemzőinek változása csak akkor jut el más szegmensekhez, amikor a kimeneti interfész elküldi az általa a megváltozott statisztikai jellemzők szerint gyűjtött statisztika csomagot, a negyedik kritérium az, hogy ez a késleltetés csak a késleltetés alatt, vagy még legfeljebb a késleltetés idejével arányos ideig okoz hibát a szimuláció kimenetében.

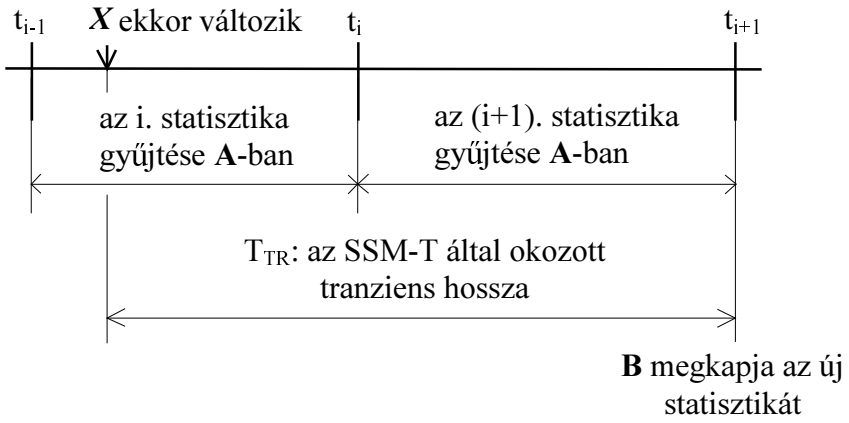
Számítsuk ki az SSM-T szinkronizációt használó szimuláció kimenetének hibáját! Az egyszerűség kedvéért tekintsünk csupán egyetlen egy üzenetfolyamtól függő egy megfigyelt kimenetet. Jelölje az üzenetfolyam statisztikai jellemzőit az $\boldsymbol{X}$ valószínűségi vektorváltozó, a közelítését pedig $\boldsymbol{X}^*$. A közelítés hibája is egy valószínűségi vektorváltozó: $h_X=\boldsymbol{X}^*-\boldsymbol{X}$. A szimuláció általunk megfigyelt kimenetét jelölje $\boldsymbol{O}$. Az üzenetfolyamok statisztikai jellemzőinek közelítésének h_X hibája a kimenetben $h_O=\boldsymbol{O}^*-\boldsymbol{O}$ hibát okoz. Ekkor a (b) feltétel formális definíciója:

(b') $h_O \le f(h_X, \boldsymbol{X}^0)$, - azaz h_O felülről becsülhető egy olyan értékkel, ami csak h_X-től függ, $\boldsymbol{X}$ értékétől nem - és $\forall\varepsilon>0\ \exists\delta: \|h_X\|<\delta \Rightarrow \|h_O\|<\varepsilon$.

Jelöljük az egy statisztikacsomag (*sample*) gyűjtéséhez felhasznált elemek (*observations*) számát *n*-nel. $\forall\delta>0\ \exists N: n>N \Rightarrow \|h_X\|<\delta$. N értéke függ mind δ-tól, mind az alkalmazott statisztikagyűjtési módszer konvergenciasebességétől. Néhány jól ismert statisztikagyűjtési módszer konvergenciasebességét részletesen megvizsgáltam a (Lencse 1998b) cikkben, az eredményeimet a 2.4. fejezetben ismertetem.

A stacionárius állapotban tehát $\|h_O\|<\varepsilon$ az N értékének megfelelő megválasztásával biztosítható.

Vizsgáljuk meg a rendszerben lejátszódó tranziens hatását. Ha az **A** szegmensben az üzenetfolyam statisztikai jellemzőit leíró ***X*** valószínűségi vektorváltozó eloszlása megváltozik az i. statisztikagyűjtési intervallumban levő t_c virtuális időpontban, akkor az új, pontos statisztikák a **B** szegmensbe a t_{i+1} szinkronizációs időpontban érkeznek meg. (4. ábra) Ha T_I felső korlát bármely $(t_{i+1}-t_i)$ intervallum hosszára, akkor az SSM-T által okozott tranziens hossza $T_{TR}<2T_I$.



4. ábra Az SSM-T által okozott tranziens

A (d) feltétel szerint a tranziens a kimenetben annak T_{TR} időtartama alatt és utána még annak hosszával arányos c_eT_{TR} ideig okozhat hibát. (Ahol c_e egy alkalmas konstans.) Így a szimuláció kimenete az SSM-T által okozott tranziens miatt legfeljebb $c_e'T_I$ ideig tartalmazhat hibát, ahol $c_e'=2(1+c_e)$.

Jelölje T_{QST} a legfeljebb $c_e'T_I$ hosszúságú hibás kimenetet adó időintervallum végétől az ***X*** következő változásáig eltelő kvázistacionárius állapot időtartamát. A (c) feltételben a *"kellően ritka"* kifejezés empirikus jelentése: $c_e'T_I << T_{QST}$. Tegyük ezt most formálisabbá: Legyen H_O felső korlát a kimeneti hiba normájára az SSM-T által okozott tranziens alatt, azaz $\|h_O\|< H_O$. Most megmutatom, hogy a kimeneti hiba átlagának normája tetszőlegesen kicsivé tehető, azaz $\left\|\bar{h}_O\right\|<\varepsilon$ biztosítható $\forall\varepsilon>0$ esetén.

Válasszuk meg δ'-t úgy, hogy: $\|h_X\|<\delta' \Rightarrow \|h_O\|<0.5\varepsilon$ a kvázistacionárius állapot T_{QST} időtartama alatt. Jelölje N_{TR} és N_{QST} a $c_e'T_I$ illetve T_{QST} időintervallum alatt gyűjtött szimulációs kimeneti értékek számát. A korábban említett *"kellően ritka"* feltétel így:

(c") $$N_{QST} \geq \frac{2N_{TR}H_O}{\varepsilon} - N_{TR}$$

Ekkor az átlagos kimeneti hiba normája:

$$\left\|\bar{h}_O\right\| \le \frac{N_{TR} H_O}{N_{QST} + N_{TR}} + \frac{N_{QST}\, 0.5\varepsilon}{N_{QST} + N_{TR}} \le 0.5\varepsilon + 0.5\varepsilon = \varepsilon$$

Az N_{QST}-re adott korlát azonban meglehetősen szigorú, és nem mindig szükséges. Mint a figyelmes Olvasó bizonyára észrevette, a szimuláció kimenetében az SSM-T tranziense által okozott hibát úgy küszöböltük ki, hogy kellően sok értékből számoltunk átlagot. Ha azonban ***X*** változásának idejét előre ismerjük, vagy a változást detektálni tudjuk, egyszerűen kihagyhatjuk a statisztikagyűjtésből az N_{TR} számú hibás kimeneti értéket. Ezáltal jelentős mennyiségű virtuális idő (és ezáltal természetesen végrehajtási idő is) megtakarítható, mert így csak a c_e'T_N virtuális időintervalum veszik el, az N_{QST}-re vonatkozó feltétel pedig ugyanaz lesz, mint a hagyományos, eseményenkénti szinkronizáció esetén:

(c") N_{QST}-nek elég nagynak kell lennie a szimuláció kimenetéből való kívánt pontosságú statisztikagyűjtéshez.

Megmutattam, hogy az alkalmazhatósági kritériumok teljesülése esetén az egyprocesszoros szimuláció eredményeit tetszőleges pontossággal közelíteni tudja az SSM-T módszer.

Megjegyzés: az elvesztegetett c_e'T_N és a hasznos T_{QST} virtuális idők aránya nagyon fontos. A c_e'T_N virtuális időt az SSM-T által okozott tranziens miatt veszítjük el. Ennek a végrehajtási idejéhez adódik még a ki- és bemeneti interfészek működéséhez, valamint a köztük lévő kommunikációhoz felhasznált végrehajtási idő. Ha mindezek összege jóval kisebb, mint a T_{QST} hasznos virtuális időhöz tartozó végrehajtási idő, és a modell partícionálása olyan, hogy a processzorok terhelése kiegyensúlyozott, akkor a párhuzamos szimulációval jó gyorsulást érhetünk el. Mindezekkel a 2.3. fejezetben részletesen foglalkozom.

2.2.4. Példák az alkalmazhatósági feltételek ellenőrzésére

2.2.4.1. Műholdas adatátviteli rendszer energiafogyasztása – pozitív példa az alkalmazhatóságra

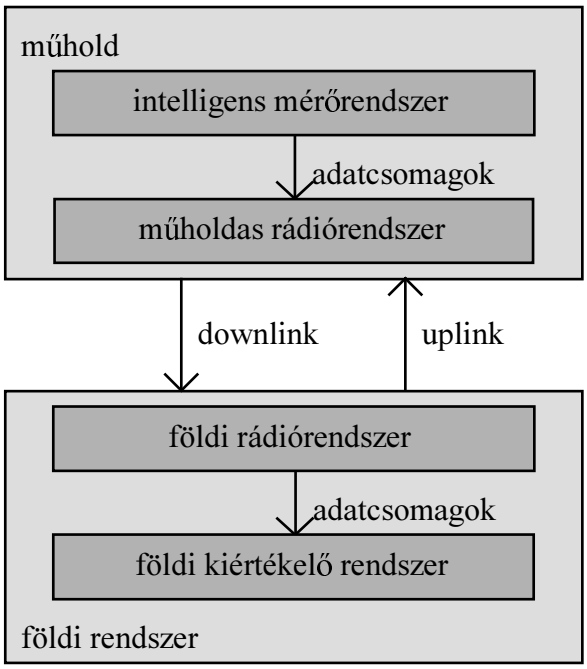
Tekintsük a következő illusztratív példát. Egy geostacionárius pályájú meteorológiai műhold hurrikán előrejelzés céljából adatokat gyűjt az atmoszféra állapotáról és bizonyos szintű feldolgozás után továbbküldi azokat a földi állomásnak kiértékelésre. Az egész rendszer három fő funkcionális egységből áll:

1. Az *intelligens mérőrendszer* vezérli az érzékelőket és kiértékeli a tőlük jövő jeleket, mérési adatokat gyűjt és előfeldolgozást végez rajtuk. A kimenete egy változó sebességű adatcsomag folyam. A csomagok gyakorisága (*packet rate*) olyan környezeti viszonyoktól függ, mint például az atmoszféra állapota, hurrikán gyanús megfigyelések, stb.
2. Az *adatátviteli rendszer* szállítja az adatcsomagokat az intelligens mérőrendszertől a földi kiértékelő rendszerhez. Az adatátviteli rendszer két rádiós linkből épül fel: egy downlink szállítja az adatcsomagokat, és egy uplink segítségével történik a nyugtázás. A műhold

adóteljesítményét a rendszer a csomagvesztés arányának függvényében szabályozza. Az egy bit átviteléhez felhasznált teljesítmény függ a környezeti viszonyoktól (atmoszféra állapota, pálya eltérések). Energiatakarékosság céljából a downlink vivőjét a rendszer kikapcsolja, mikor nincs átvitel.

3. A *földi kiértékelő rendszer* felelős a gyűjtött mérési adatok végső kiértékeléséért.

A 5. ábra mutatja a rendszer blokkdiagramját.



5. ábra Műholdas meteorológiai mérőrendszer

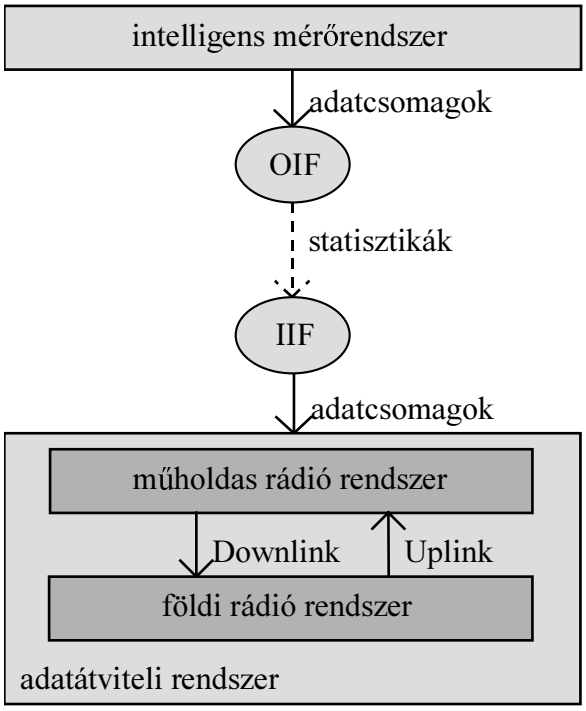
A szimuláció feladata: meghatározni a műholdas rádiórendszer energiafogyasztásának viselkedését a napelem és akkumulátor méretezéséhez. A kérdéses energiafogyasztás mind az intelligens mérőrendszertől származó adatfolyam csomaggyakoriságától, mind a rádiócsatorna terjedési viszonyaitól függ. Mindkét tényező olyan összetett környezeti feltételek függvénye, amelyek analitikus kezelése túl bonyolult lenne. A környezeti feltételek egy része (pl. az atmoszféra állapota) befolyásolja mind a csomaggyakoriságot, mind a szükséges adóteljesítményt, így a két alrendszer nem szimulálható egymástól függetlenül. Az azonban korábbi kísérletekből már ismert, hogy a környezeti feltételek a csomaggyakorisághoz képest nagyon lassan változnak, azaz két egymást követő lényeges változás között tipikusan adatcsomagok millióit viszi át a rendszer. Az is ismert, hogy a csatornakapacitás több, mint kétszerese a legnagyobb csomaggyakoriság esetén szükségesnek is, így gyakorlatilag nincs pufferelés attól eltekintve, hogy (az esetleg szükséges újraadás érdekében) a csomagokat tárolják, amíg nyugta nem érkezik rájuk. A feladat megoldására a következő párhuzamos szimulációt dolgoztam ki:

- A földi kiértékelő rendszert a szimulációból kihagyjuk, mert annak nincs hatása a vizsgált energiafogyasztás alakulására.

- A megmaradó rendszert két szegmensre osztjuk: az egyik az intelligens mérőrendszer, a másik az adatátviteli rendszer. Mindkét szegmensben modellezzük azokat a környezeti feltételeket, amelyek a szegmens működése szempontjából lényegesek.

- A két szegmenst két processzor hajtja végre, a szinkronizációs módszer SSM-T. Az intelligens mérőrendszer kimeneti interfésze statisztikát gyűjt a csomagok érkezési időköz (*inter-arrival time*) eloszlásáról, és az eredményt eljuttatja az adatátviteli rendszer bemeneti interfészének, amely a statisztikai jellemzők alapján újra előállítja a csomagfolyamot.

A szimulációs modell blokkdiagramja a 6. ábrán látható.



6. ábra A mérőrendszer szimulációs modellje

A modell valóban kielégíti az SSM-T alkalmazhatósági kritériumait:

(a) A két szegmens határán valóban nem az egyes csomagok, hanem csak a csomaggyakoriság számít az adatátviteli rendszer fogyasztása szempontjából.

(b) A csomaggyakoriság (vagy érkezési időköz eloszlás) becslésében elkövetett kis hiba szintén kis hibát okoz az energiafogyasztás számításánál, és az utóbbi hiba csak az előbbi hiba mértékétől függ, nem függ a csomaggyakoriság konkrét értékétől.

(c) A rendszerben az energiafogyasztás szempontjából lényeges változások kellően ritkák ahhoz, hogy kvázistacionárius állapotban elégséges számú megfigyelést végezhessünk a rendszer kimenetén.

(d) Mivel gyakorlatilag nincs pufferelés, a csomaggyakoriság változásának késleltetése csak addig okoz hibát, amíg az új, helyes statisztika meg nem érkezik.

Az alkalmazhatósági kritériumok ellenőrzésére adott példákkal az Olvasót igyekeztem segíteni az SSM-T alkalmazhatóságának problémakörében való nagyobb jártasság megszerzésében.

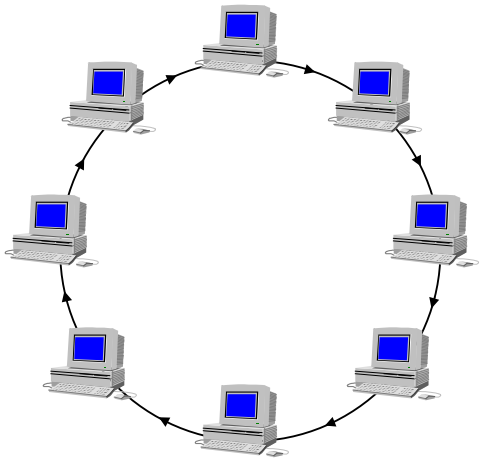
2.2.4.2. Módosított műholdas adatátviteli rendszer szimulációja – negatív példa az alkalmazhatóságra

A korábban említett műholdas rendszert most a következőképpen fogjuk módosítani. A műholdon két adó van, teljesítményszabályozás nincs. A nyugtázatlan csomagokat most is újra kell adni, míg csak nyugta nem érkezik. Az elsődleges adó mindig üzemel, a másodlagos, csak akkor, ha nagy csomaggyakoriság vagy kedvezőtlen csatorna viszonyok (nagy arányú csomagvesztés) miatt szükséges. A földi állomás nem rendelkezik börszt demodulátorral, így a második adó bekapcsolásakor jelentős szinkronizációs overhead lép fel. Éppen ezért a vivőt nem kapcsolják ki rögtön, mikor egy csomag átvitele befejeződik, hanem csak bizonyos késleltetéssel, ha közben nem jön újabb csomag. Ez azt is jelenti, hogy már néhány (elszórtan érkező) csomagnak a másodlagos adó segítségével történő továbbítása is jelentős fogyasztásnövekedéssel járhat.

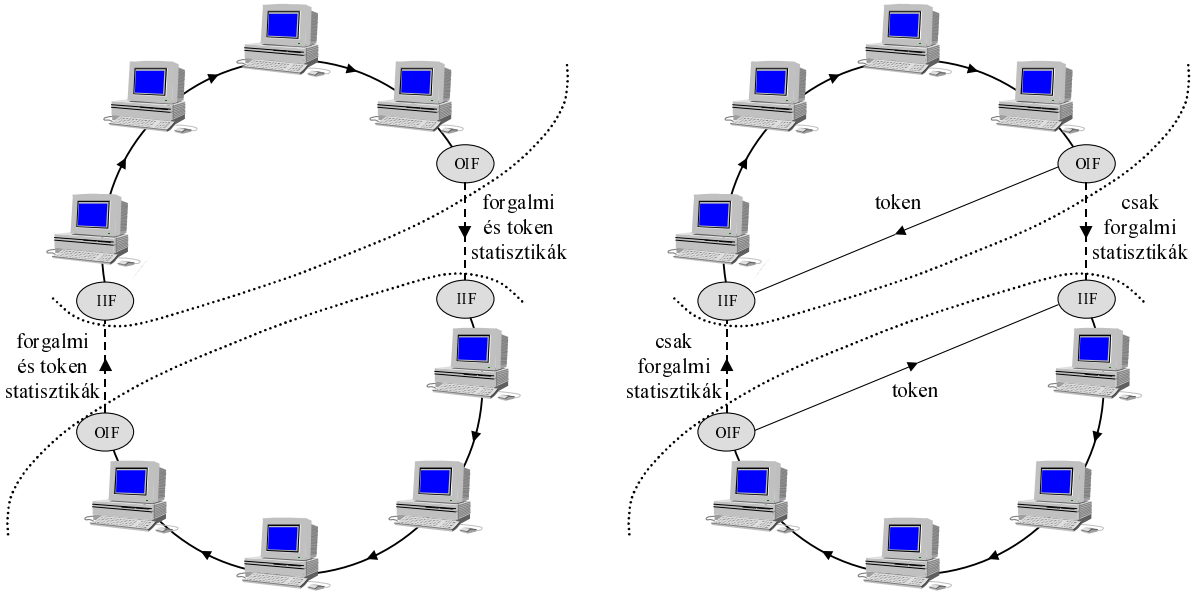
Tetszőleges csatorna viszonyok esetén kiszámítható, hogy mekkora az a csomaggyakoriság, amely éppen kimeríti az elsődleges adó kapacitását. Ha az intelligens mérőrendszertől származó csomagfolyam csomaggyakorisága elég közeli ehhez az értékhez, akkor a csomaggyakoriság mérésében elkövetett tetszőlegesen kis hiba is komoly hibát okozhat a szimuláció kimenetében. Most a (b) feltételt nem elégíti ki a rendszer.

2.2.4.3. FDDI gyűrű szimulációja – negatív példa az alkalmazhatóságra

Egy FDDI (ANSI X3.139, 1987) gyűrű teljesítőképességét kell szimulációval megvizsgálnunk. A 7. ábra egy 8 állomásból felépülő példa gyűrűt mutat, a nyilak az elsődleges gyűrű irányát jelezik, szimulációnkban csak ezt vizsgáljuk. Ha a gyűrűt két (vagy több) szegmensre vágjuk szét, és az egyes szegmensek végrehajtását rendeljük külön processzorokhoz, a szegmensek határán a vezérjel (*token*) tényleges továbbadását nem lehet mechanikusan a token felbukkanásának statisztikáival helyettesíteni (8. ábra). Ez az FDDI MAC (Media Access Control) protokolljának a megsértését jelentené (token vesztés, több token, stb). Itt az SSM-T nem alkalmazható, mert a rendszer nem elégíti ki az (a) feltételt. Ez a megállapítás természetesen nem zárja ki, hogy a problémára létezzen ad-hoc megoldás: a szegmensek kimeneti interfészétől a token közvetlenül eljuttatható a bemeneti interfészhez, ami a másik gyűrű forgalma és az FDDI MAC protokoll ismeretében a megfelelő időben újra beadhatja tokent a szegmensben lévő gyűrű részbe. Ez a trükk megoldhatja a token problémáját, azonban a gyűrűnek az adatforgalomra való zárása további feladat, amivel a feladat partikuláris volta miatt nem foglalkozom; a példa tanulsága éppen az, hogy itt az egyszerű, mechanikus megoldás nem működik.



7. ábra Egy FDDI gyűrű – hogyan lehet ezt szétvágni?



8. ábra Az FDDI gyűrű mechanikus és trükkös szétvágása

2.2.4.4. Két összekapcsolt FDDI gyűrű szimulációja – pozitív példa az alkalmazhatóságra

A (Lencse 1998a) cikkemben az SSM-T módszert két összekapcsolt FDDI gyűrű szimulációjára alkalmaztam. A hálózat topológiája a 9. ábrán látható (22. oldal). A két gyűrű alkotta a két szegmenst, amelyek között a szinkronizációt az SSM-T segítségével oldottam meg. A szimulációs modell részletes leírása a kettős gyűrű párhuzamos szimulációjával elért gyorsulásról szóló 2.3.2. alfejezetben található, az ott közöltek alapján az érdeklődő Olvasó ellenőrizheti, hogy a rendszer tényleg kielégíti az SSM-T alkalmazhatósági kritériumait.

2.3. Az elérhető gyorsulás

2.3.1. Mikor érhető el jó gyorsulás?

Az SSM-T alkalmazásával elérhető gyorsulás szempontjából nagyon fontos, hogy a szinkronizációs módszer mekkora többletmunkát okoz. Ennek megvizsgálásához definiáljuk most a statisztikatovábbítási irányított gráfot a következőképpen:

- csomópontjait a szimulációs modell szegmensei alkotják
- irányított élei a statisztikatovábbításnak a szegmenstől szegmensig vezető útjai

Ha a gráfban nincs irányított kör, akkor tekintsünk egy leghosszabb utat. Ez egy olyan *pipeline*ra emlékeztet, amelynek állomásai (*stage*) az útban szereplő szegmensek. (Maga a szimulációs modell persze ennél összetettebb, a pipeline egyes szakaszaihoz – esetleg szövevényes – kerülőutak is tartozhatnak.) – Ez az eset kommunikációs hálózatok vizsgálatánál egyáltalán nem mondható tipikusnak. Éppen ellenkezőleg, a valóságos hálózatok részei közti kapcsolat és így a modell szegmensei közti adatáramlás is általában kétirányú. Emiatt volt indokolt az, hogy a laza időszinkronizációt szimmetrikusra terveztem.

Megfontolásainkhoz tekintsünk egy minimális méretű ilyen rendszert, ez két szegmensből áll, amelyek egymással statisztikákat cserélnek. Történjen ez a statisztikacsere minden T hosszú virtuális időintervallum végén. A szegmenseket jelöljük **A**-val és **B**-vel. Az őket végrehajtó processzorok legyenek egyformák és a szimuláción kívül más terhelésük ne legyen. Egy T hosszúságú virtuális időintervallum végrehajtási idejét jelölje az egyes szegmensek esetén τ_A és τ_B. A szinkronizáció miatt szükséges kommunikáció idejét jelölje τ_C. Ez az idő magában foglalja a hálózatra küldendő adatcsomag összeállításának és az adatok leadásának idejét, a terjedési késleltetést, és a címzettnél a hálózatról kapott adatcsomag kibontásák idejét. Így a T hosszú virtuális időhöz tartozó végrehajtási idő két processzoron:

$$\tau_2=\max(\tau_A,\tau_B)+\tau_C$$

A statisztikagyűjtés és az üzenetfolyamnak a statisztikákból való regenerálása által felhasznált végrehajtási időt jelölje τ_{OIF} és τ_{IIF}. Ezek benne vannak a két processzoros, SSM-T szinkronizációt alkalmazó szimulációra vonatkozó τ_A és τ_B mindegyikében, így ezeket ki kell vonnunk, ha az egyprocesszoros (nem párhuzamos) szimuláció idejét kívánjuk meghatározni. A T hosszú virtuális időhöz tartozó egy processzoros végrehajtási idő:

$$\tau_1=\tau_A-\tau_{OIF\text{-}A}-\tau_{IIF\text{-}A}+\tau_B-\tau_{OIF\text{-}B}-\tau_{IIF\text{-}B}$$

Az I/O interfészek együttes végrehajtásának idejét jelölje τ_{IF}.

$$\tau_{IF}=\tau_{OIF\text{-}A}+\tau_{IIF\text{-}A}+\tau_{OIF\text{-}B}+\tau_{IIF\text{-}B}$$

Az egyprocesszoros szimulációhoz képesti gyorsulás két processzoron:

$$s=\frac{\tau_A+\tau_B-\tau_{IF}}{\max(\tau_A,\tau_B)+\tau_C}$$

Ez az érték közelíthet a 2-höz, ha $\tau_A \approx \tau_B$, $\tau_C << \tau_A$ és $\tau_{IF} << \tau_A$, azaz a processzorok közti terhelés jól kiegyensúlyozott, a kommunikáció, valamint a statisztikagyűjtés és üzenetfolyam regenerálás által okozott többletmunka kicsi. A következő fejezetben fogok egy valós példát mutatni arra, hogy ez nagyon is lehetséges.

Egy összetett kommunikációs hálózatban általában több olyan pont is van, ahol az SSM-T alkalmazhatósági kritériumai kielégíttetnek. Ez azt jelenti, hogy ezeken a pontokon az SSM-T interfészek behelyezése nem rontja lényegesen a szimuláció kimenetének a pontosságát. Ezek közül a pontok közül a szimuláció írójának kell gondosan kiválasztania azokat, amelyek a szimulációs modellt hasonló komplexitású részekre osztják, abban az értelemben, hogy az egyes szegmensek végrehajtási ideje azonos nagyságrendbe essék. Vegyes architektúrájú / teljesítményű rendszer esetén a részek méretének meghatározásánál figyelembe veendő az egyes processzorok (esetleg a köztük lévő eltérő hálózati összeköttetés) teljesítménye is.

A statisztikagyűjtési módszer kiválasztása is fontos, ezek konvergenciasebességét és az I/O interfészek végrehajtási ideje szempontjából fontos algoritmikus komplexitását a 2.4. fejezetben tárgyalom.

Ebben a fejezetben megadtam, hogy mikor érhető el jó gyorsulás (speed-up) SSM-T esetén, tehát mikor van értelme alkalmazni.

2.3.2. Elért gyorsulás egy valós életből vett hálózat pontos modelljén

2.3.2.1. A modellezésre kiválasztott hálózat

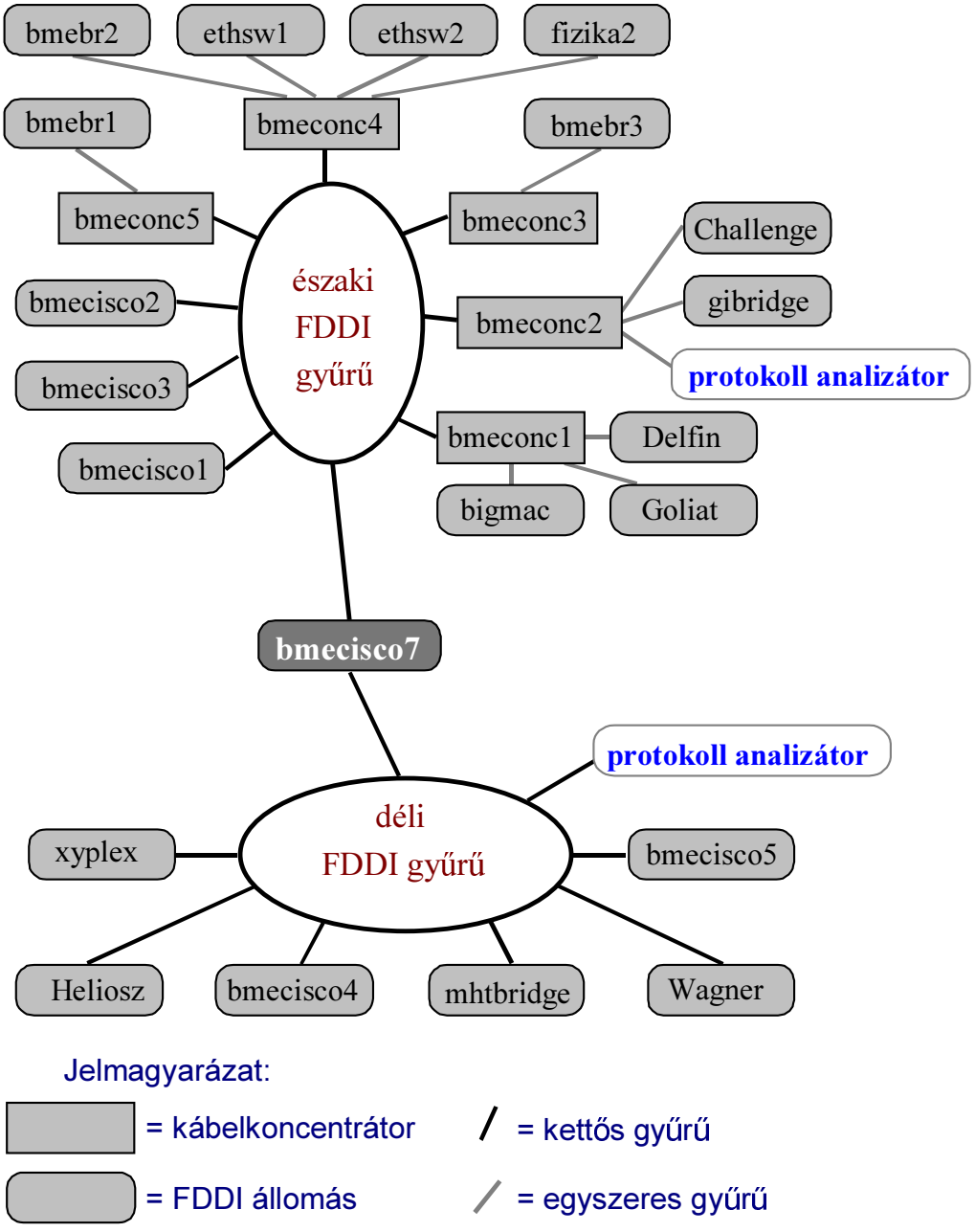
1996-ban azt a célt tűztem magam elé, hogy a statisztikai szinkronizációs módszert egy realisztikus szimulációban vizsgáljam meg, mert addig csak egy illusztratív példa alkalmazás volt ismert (Pongor, 1992). Annak érdekében, hogy a módszer jellemzőit a gyakorlati élet szempontjából fontos esetben vizsgálhassam, a szimuláció tárgyául az akkor korszerűnek számító FDDI-t választottam. A Fiber Distributed Data Interface egy 100 Mbps-os optikai hálózati szabvány (ANSI X3.139, 1987). Kettős gyűrű topológiája van, ami kábelkoncentrátorok segítségével faszerkezetekkel egészíthető ki. Az úgynevezett időzített tokenes protokollt (*Timed-Token access protocol*) használja közeghozzáférési célokra.

A lehetőségeim határáig minden erőfeszítést megtettem, hogy olyan szimulációs modellt építsek, ami nagyon közel áll egy valóságos hálózathoz. A Magyarországon létező FDDI hálózatok közül a BME FDDI gerinchálózatához volt hozzáférésem. Ez a hálózat az 1996/97-es tanévben, amikor a szimuláció bemenetét adó méréseket végeztem, a következőképpen épült fel: A hálózat két gyűrűből állt. Az úgynevezett északi gyűrű az egyetem gerincét adta és 15 FDDI állomás volt benne, összesen 5 koncentrátorral összekapcsolva. A déli gyűrű pedig a Villamosmérnöki és Informatikai Kar gerinceként szolgált és 7 FDDI állomásból állt. A két gyűrűt egy router kötötte össze. A topológia az 9. ábrán látható. Az ábrán megjelöltem azokat a pontokat, ahol a hálózat forgalmát mértem. Ezt az EIK Network General gyártmányú, Sniffer nevű protokoll analizátorával végeztem. Az elsődleges gyűrűn (normál működés esetén forgalom csak itt van, a másodlagos gyűrű tartalék) az összes csomag első 32 byte-ját lementettem, ennek alapján meghatároztam és tároltam a mintavételezés

időtartama alatti csomagokról (pontosabban keretekről, mivel a 32 byte kevés volt ahhoz, hogy a hálózati szint fejléce is beleférjen) az alábbi jellemzőket:

- eredeti hossz FDDI MAC réteg szintjén
- követési idő (*inter-arrival time*)
- forrás MAC cím
- cél MAC cím

Ezekből már egy nagyon részletes forgalmi mátrix nyerhető $T=[t_{ij}]$ (*traffic matrix*), ahol *i* a forrás *j* pedig a cél állomás indexe, a t_{ij} elemek pedig az i. állomástól a j. állomás felé irányuló csomagok 2 dimenziós eloszlásai, csomaghossz és követési idő, mint dimenziók szerint.



9. ábra A BME FDDI gerinchálózata az 1996/97-es tanévben

2.3.2.2. Az FDDI hálózat modellje

Az FDDI szabványnak csak a szimuláció szempontjából lényeges részeit modelleztem. A kettős gyűrűből csak az elsődleges gyűrűt, mert a másodlagos gyűrű normál működés közben nem szállít forgalmat. Minden FDDI állomás tartalmaz egy MAC (*Media Access Control*) entitást. Ennek a normál működését a *Timed-Token Ring Protocol*t pontosan követtem. A különböző gyűrű inicializálási mechanizmusokat (*ring initialization and ring recovery*) teljesen kihagytam, mert a gyűrű normál működését vizsgáltam. A szimuláció kezdetén minden állomásnál beállításra került a TTRT (*Target Token Rotation Time*) operatív értéke (*T_Opr*), a tokent pedig egy kiválasztott állomás helyezte be a gyűrűbe. A működés közben hibát nem modelleztem, ilyent a mérések közben sem tapasztaltam. A kábelhosszak pontos értékét a mérési jegyzőkönyvek alapján állítottam be. Az állomáskésleltetés (*station latency)* értékét a (MIL 3, 1996) forrásból vettem. A T_Opr paraméter értékét az FDDI állomásokból kérdezték le számomra. A kábelkoncentrátor hatását konstans késleltetéssel modelleztem.

A szimulációs modellt a korábban említett OMNeT++ diszkrét idejű szimulátorhoz készítettem el. (A modell példaprogramként jelenleg is szerepel az OMNeT++ disztribúcióban (Varga, 2000).)

2.3.2.3. A forgalom modellezése

Amint már korábban említettem, a $T=[t_{ij}]$ forgalmi mátrix az FDDI gyűrűn végzett forgalmi mérésekből származtatható. Ez a mátrix a gyűrűnek azon pontján jellemzi a forgalmat, ahova a protokoll analizátort behelyeztem. T nem azonos a $D=[d_{ij}]$ igény mátrix-szal (*demand matrix*), ahol d_{ij} a külvilágból az *i*. állomáshoz (mint forrás állomáshoz) érkező, általa a gyűrűn a *j*. állomásnak elküldendő igények két dimenziós (csomaghossz és érkezési időköz, mint dimenziók fölötti) igény eloszlását jelenti. A forgalomgeneráláshoz a D mátrixra van szükségünk, de csak a T-t tudjuk egyszerűen mérni. A (Lencse, 1997) cikkben megmutattam, hogy miért használható a forgalomgeneráláshoz T a D helyett annak ellenére, hogy a két mátrix nem azonos.

Mivel a megfigyelt FDDI hálózat átlagos kihasználtsága 5% alatt volt, egy *Load Multiplier*-nevű faktort vezettem be a nagyobb terhelés modellezése érdekében. Ezzel az értékkel elosztva a mért érkezési időköz értékeket a terhelés Load Multiplier szorosára nő. Ennek a faktornak az értékét változtatva a rendszer forgalmi terhelése könnyen módosítható, miközben a forgalom természete ugyanolyan marad.

A két gyűrűt egy router köti össze, amelynek egy-egy portja van mindegyik gyűrűben. Minden olyan csomagot, amelynek a címzettje az egyik gyűrűben a router abban levő portja, a router a másik gyűrűben lévő portjához továbbít. Mivel a méréskor a routolási információkat nem gyűjtöttem, a cél gyűrűben lévő router port statisztikai alapon választja ki a címzettet a (Lencse, 1997) cikkben ismertetett algoritmus szerint.

2.3.2.4. A kísérletek és eredményeik

Mint korábban említettem, a hálózatnak egyetlen olyan pontja van, ami kielégíti az SSM-T alkalmazhatósági kritériumait: a két gyűrűt összekapcsoló router. A szimuláció célja a gyorsulás vizsgálata volt, ezért egy hagyományos egyprocesszoros (SSM-T nélküli) szimuláció-

hoz, mint referenciához képest vizsgáltam az SSM-T alkalmazásának különböző eseteit. A következő 4 modellt működtettem azonos terheléssel a mérésekhez elegendő 2s virtuális ideig:

1. A két gyűrű között direkt összeköttetés van, az egész hálózat szimulációja egyetlen eseményhalmaz használatával egy processzben történik, így a két gyűrű lokális virtuális ideje így azonos.
2. A két gyűrűt az **SSM-T statisztikai interfészei kapcsolják össze**, de az egész hálózat szimulációja egyetlen eseményhalmaz használatával egy processzben történik, így a két gyűrű lokális virtuális ideje azonos.
3. A két gyűrűt az SSM-T statisztikai interfészei kapcsolják össze, és **a két gyűrű szimulációja két saját eseményhalmazzal rendelkező processzben történik, amelyek lokális virtuális idői között laza időszinkronizáció van**. A két processz végrehajtását ugyanaz a processzor végzi.
4. A két gyűrűt az SSM-T statisztikai interfészei kapcsolják össze, és a két gyűrű szimulációja két saját eseményhalmazzal rendelkező processzben történik, amelyek lokális virtuális idői között laza időszinkronizáció van. **A két processz végrehajtását két külön processzor végzi**.

Természetesen a 2., 3. és 4. szimuláció kimenete azonos, csak a végrehajtási idejük eltérő.

Mindegyik kísérlethez azonos konfigurációjú PC-(ke)t használtam: 100MHz Intel Pentium processzor, 16MB RAM, 512KB cache. A 4. modell esetén a két számítógép közötti összeköttetést egy 150 Mbit/s sebességű ATM hálózat biztosította. A processzek Linux operációs rendszer alatt futottak (kernel verzió 2.0.25) és a gcc 2.7 verziójával a "-O3" kapcsolóval (teljes optimalizáció) fordítottam le őket. Az OMNeT++ sim könyvtárának verziója 145, a PVM verziója pedig 3.3 volt.

A processzorokon csak az általam vizsgált szimuláció futott, a hálózaton más forgalom nem volt. A végrehajtási időket a "time" UNIX paranccsal mértem, az úgynevezett falióra időt (*wall clock time*) véve figyelembe. (A falióra idő a processz elindítása és befejeződése között a külső világban eltelt fizikai időt jelenti.)

Az 1. táblázat mutatja az egyes esetekben mért végrehajtási időket. Látható, hogy sem az SSM-T interfészek behelyezése (2. modell), sem a laza időszinkronizációhoz szükséges processzek közti kommunikáció (3. modell) nem okoz szignifikáns többletmunkát a referenciához képest (1. modell). A 4. modell végrehajtási ideje az 1. modelléhez képest a statisztikacsere gyakoriságától függően (update interval 100ms illetve 10ms virtuális idő) 1.91-szeres illetve 1.86-szoros gyorsulást mutat. A 4. modellt végrehajtottam UI=1ms mellett is, ez 5 perc 33 másodpercig tartott, ami 1.75-szörös gyorsulást jelent. Még ez is jó érték, de az UI=1ms paramétert a gyakorlatban valószínűleg nem fogják alkalmazni, mert túlságosan kicsi a statisztikagyűjtésre az FDDI által használt TTRT nagyságrendjéhez képest, ami tipikusan 10ms körüli. A kimenet hibája az 1.91-szeres gyorsulást adó esetben az északi gyűrű esetén 0.63% a déli gyűrű esetén pedig 3.56% volt az egyprocesszoros szimulációhoz képest.

modell sorszáma	szinkronizáció	FES	processzorok száma	UI = 100 ms	UI = 10 ms
1	-	1	1	9:42	
2	SSM-T	1	1	9:43	9:46
3	SSM-T	2	1	9:45	9:50
4	SSM-T	2	2	5:04	5:13

1. táblázat. Az egyes modellekhez tartozó végrehajtási idők [perc:másodperc]

Két összekapcsolt FDDI gyűrű párhuzamos szimulációjánál a szimulációt SSM-T alkalmazásával két lazán csatolt processzoron futtatva, 1.91 illetve 1.86 szoros gyorsulást értem el az egy processzoros szimulációhoz képest.

Ebből arra következtetek, hogy az SSM-T szinkronizációs módszer lehetővé teszi (az alkalmazhatósági kritériumokat kielégítő) kommunikációs rendszerek hatékony párhuzamos szimulációt munkaállomás fürtökön (*clusters of workstations*).

2.4. A statisztikagyűjtés körüljárása

A (Lencse, 1998b) cikkemben az SSM-T számára különböző statisztikagyűjtési eljárások erőforrásigényét és pontosságát (konvergenciasebességét) vizsgáltam szimulációval a kommunikációs hálózatok szimulációja során előforduló fontosabb időinvariáns eloszlások esetén. A dolgozatomnak ez a fejezete a cikk megfelelő részeinek szabad magyar fordításán alapul.

A megvizsgált erőforrásigények a következők:

- a statisztikagyűjtés és az üzenetfolyam regenerálás számításigénye
- a statisztikagyűjtés és az üzenetfolyam regenerálás tárigénye
- a szegmensek közti kommunikációs igény

A megvizsgált módszerek:

- relatív gyakoriság (*relative frequency*)
- egyenlő osztásközű hisztogram (*equidistant histogram*)
- Barron estimate
- egyenlő cellavalószínűségű hisztogram (*equiprobable bin histogram*)
- kvázi egyenlő cellavalószínűségű hisztogram (*semi equiprobable bin histogram*)

A konvergenciasebességet néhány jól ismert eloszlás és két mérésen alapuló empirikus eloszlás esetén vizsgáltam.

Szóhasználat: a statisztikacsomag vagy minta (*sample*) N darab megfigyelésből (*observation*) áll.

2.4.1. A választott hibakritérium

Az a feladatunk, hogy az X valószínűségi változó ismeretlen eloszlását megbecsüljük. Jelölje az X eloszlásfüggvényét *F(x)*=Pr(X<x) és (folytonos esetben) a sűrűségfüggvényét *f(x)=F'(x)*. Az *f(x)*-et *g(x)*-el közelítjük. A közelítés hibájának mértékét például az alábbi hibakritériumok valamelyikével írhatjuk le:

$$L_p = \int_{-\infty}^{\infty} \left| f(x) - g(x) \right|^p dx, \quad p \geq 1$$

Egyszerű analitikus kezelése miatt a legnépszerűbb hibakritérium az L_2 (négyzetes hiba). Jelen esetben azonban nem megfelelő, mert elnyomja az eloszlás szélein (farok, *tail*) jelentkező hibákat a kis értékek négyzetre emelésével. Az L_1 hibakritériumot választottam, mert számunkra az a fontos, hogy események valószínűségét milyen jól tudjuk becsülni, azaz az *f(x)* függvénynek egy adott intervallumra vett integrálja számít, nem maga az *f(x)*. Másképpen szólva: Egy tetszőleges Borel halmazra mennyire tér el a mi *g(x)* becslésünk valószínűségi mértéke az elméleti *f(x)* valószínűségi mértékétől. Ennek az eltérésnek a szuprémuma a Scheffè tétel szerint kifejezhető az L_1 hibával, amint ez (Devroye and Győrfi, 1985.) 2. oldalán is található:

$$\sup_{B \in \mathsf{B}} \left| \int_B f(x)dx - \int_B g(x)dx \right| = \frac{1}{2} \int_{-\infty}^{\infty} \left| f(x) - g(x) \right| dx = \frac{1}{2} L_1$$

Az L_1 abszolút hibának egy másik hasznos tulajdonsága a skálázás monoton folytonos változásaira való invarianciája, ami (Scott, 1992.) 41. oldalán olvasható. L_1 dimenziómentes mennyiség, és könnyű belátni, hogy $0 \leq L_1 \leq 2$.

Diszkrét valószínűségi változókra az L_1 hiba integrál helyett összegként számítható. Vegye fel az X diszkrét valószínűségi változó az $x_1, x_2, \ldots x_k, \ldots$ értékeit $p_1, p_2, \ldots p_k, \ldots$ valószínűségekkel. Jelölje $q_1, q_2, \ldots q_k \ldots$ X értékeinek becsült relatív gyakoriságát.

$$L_1 = \sum_k \left| p_k - q_k \right|$$

2.4.2. A megvizsgált statisztikagyűjtési eljárások

Mivel a szimulációt általában olyan összetett feladatok megoldására használják, amelyek az analitikus megoldása nem ismert, az esetek többségében a szimulációban megfigyelt eloszlások sem ismertek, ezért csak a nemparametrikus módszerekkel foglalkoztam. Megvizsgáltam az eljárások számítási-, tár- és kommunikációs igényeit. A továbbiakban használom az algoritmuselmélet (Aho et al. 1975) és a C programozási nyelv jelöléseit. (A C nyelvű kódrészleteket `courier`-ral írom.)

2.4.2.1. Relatív gyakoriság

Ha egy valószínűségi változó diszkrét eloszlású és nem túl nagy a lehetséges értékeinek száma, ráadásul ezek előre ismertek, akkor a statisztikagyűjtés legegyszerűbb módja az, hogy megszámoljuk, hogy az egyes lehetséges értékei hányszor fordulnak elő. Ezt hívják empirikus eloszlásnak. Jelölje az X diszkrét valószínűségi változó lehetséges értékeit: x_1, x_2, … x_k, … x_M. Ha egy N méretű mintában X az x_k értéket n_k-szor veszi fel, akkor annak relatív gyakorisága $q_k = n_k/N$. A statisztikagyűjtéshez szükségünk van a számlálóknak (`int`) egy M méretű tömbjére. Az általános esetben rendezetten kell tárolnunk az x_k értékeket (ez egy másik M méretű `double` típusú tömböt kíván) és minden beérkező X_i megfigyelésre ($1 \le i \le N$) bináris kereséssel kell meghatározni, hogy melyik j-re: $X_i=x_j$ ezután a j. számláló értékét kell növelni. Speciális esetben, amikor külön információval rendelkezünk az X lehetséges értékeiről, lehetséges, hogy a bináris keresés elkerülhető hash függvény vagy akár direkt leképzés alkalmazásával. (például: Legyen X értékkészlete 0, 10, 20, … 1000. Ekkor X/10 használható a számlálók tömbjének indexeként.)

Ha előre ismerjük és rendezetten tároljuk a lehetséges x_k értékeket, akkor egy N méretű minta gyűjtésének algoritmikus bonyolultsága O(N*log(M)) az általános esetben, és a statisztikacsere érdekében csak a számlálókat kell átvinnünk, ennek mérete: M*`sizeof(int)`.

Ha azonban nem ismerjük, vagy nem kívánjuk tárolni az X lehetséges értékeit (például túl sok van belőlük és egy részük nagyon kis valószínűséggel fordul elő), akkor az x_k értékek rendezett tömbjét a statisztikagyűjtés közben kell felépítenünk. Egy új elem behelyezése a tömb bizonyos számú elemének mozgatását is igényelheti, ezek száma arányos a tömb méretével. Így a statisztikagyűjtés algoritmikus bonyolultsága: $O(N*\log(M')+(M')^2)$, ahol M' az X különböző elemeinek számát jelenti az adott N méretű mintában. Természetesen az X lehetséges értékei (a számlálókkal együtt) tárolhatók valamilyen garantáltan logaritmikus beszúrási költségű adatstruktúrában, pédául: AVL-fa, 2-3-fa, B-fa, stb. Ebben az esetben a hozzáadódó $O((M')^2)$ költséget csökkentettük O(M'*log(M'))-re. A kommunikációs költség mindenképpen tartalmazni fogja az M' érték átvitelét is: M'*`sizeof(double)`.

A statisztika regenerálás helyén a tárigény szintén M*`sizeof(int)`. A gyűjtött gyakoriság értékek alapján a véletlenszám-generálás O(M) számú összeadást igényel a triviális algoritmussal: Generáljunk egy R véletlenszámot egyenletes eloszlás szerint a [0, K) intervallumban, ahol:

$$K = \sum_{i=1}^{M} n_i$$

Aztán adjuk össze az n_1, n_2, ...n_j gyakoriságokat úgy, hogy megállunk, mikor az összeg nagyobb lesz, mint R. A generált véletlenszám az x_j lesz. Az algoritmus egy valóban használható változata a következő: számítsuk ki és tároljuk el az összes sum(j) értéket ($1 \le j \le M$).

$$sum(j) = \sum_{i=1}^{j} n_i$$

Ez M*`sizeof(int)` tárigényű, és az eredeti n_i értékekre már nincsen szükség. A megfelelő **j** index meghatározására bináris keresést használunk, aminek műveletigénye O(log(M)).

2.4.2.2. Egyenlő osztásközű hisztogram

Ha egy valószínűségi változó folytonos eloszlású, vagy diszkrét eloszlású ugyan, de a lehetséges értékeinek száma igen nagy, akkor elfogadható valamilyen fajta hisztogram használata a statisztikagyűjtésre.

Az egyenlő osztásközű hisztogramot a t_0 kezdőpontjával h cellaméretével és M cellaszámával jellemezhetjük. Minden cellájához egy számlálóra van szükség, összesen közelítőleg M*`sizeof(int)` a tár- és kommunikációs költsége. A statisztikagyűjtés nagyon egyszerű, ha a:

`counter_of_bin[int((X-t0)/h)]++;`

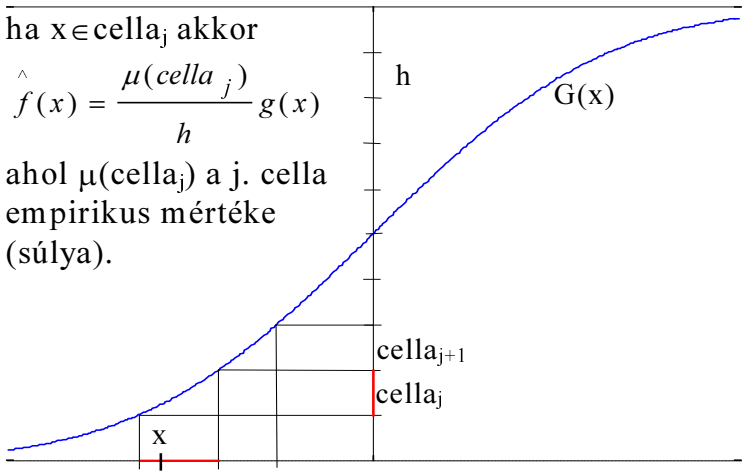
algoritmust használjuk, így az N méretű minta gyűjtésének komplexitása csak O(N).

A gyűjtött hisztogram alapján való véletlenszám-generáláshoz ki kell választani egy cellát, és a választott cellán belül egyenletes eloszlás szerint kell sorsolni. A cella kiválasztását éppen úgy (és olyan költséggel) végezhetjük, ahogy a relatív gyakoriság módszerénél láttuk.

Meg kell azonban jegyeznünk, hogy a hisztogramnak ezt a megközelítését csak akkor használhatjuk, ha előre ismerjük az eloszlás (elég jó) alsó és felső korlátját. Bizonyos gyakorlati esetekben ezeket a korlátokat csak mint "nagy valószínűségű korlátokat" ismerjük, azaz, kis valószínűséggel az X értékei kívül eshetnek a korlátokon. Ebben az esetben elterjedt gyakorlat az underflow és/vagy overflow cellák használata ezeknek az értékeknek a számlálására.

2.4.2.3. Barron estimate

A Barron estimate (Barron et al. 1992) egy hisztogram alapú valószínűségi sűrűség becslési módszer, ami a következőképpen működik: Az X valószínűségi változóból származó bejövő megfigyeléseket transzformáljuk az előre rögzített $G(x)$ függvénnyel, amely valamennyire közelíti $F(x)$-et, azaz X eloszlásfüggvényét. A transzformált mintákból a [0,1] intervallumban egy M cellaszámú h=1/M cellaszélességű hisztogramot készítünk. A becsült sűrűségfüggvény számítását a 10. ábra mutatja.



10. ábra Barron estimate

Megjegyzés: g(x) helyett $g(x_j)$-t használva, ahol x_j egy alkalmas rögzített pont a j. partícióban, egy közel egyenlő cellavalószínűségű hisztogram kapható, ha *G(x)* közel van *F(x)*-hez.

Az erőforrás igények az egyenlő osztásközű hisztograméiból származtathatók, ehhez jön hozzá a minták transzformálásának a számításigénye.

2.4.2.4. Egyenlő cellavalószínűségű hisztogram

Az *F(x)* eloszlásfüggvény ismeretében a cellahatárok megválaszthatók úgy, hogy a cellák egyenlő valószínűségűek legyenek. A statisztikagyűjtésnél az egyes megfigyelésekhez kell a megfelelő cellát megtalálni. Ez bináris kereséssel megoldható, így egy N elemű minta gyűjtésének számításigénye O(N*log(M)), éppen úgy, mint a relatív gyakoriság módszernél. A tárigény magában foglalja mind a cellahatárok, mind a számlálók tárolásához szükséges tárkapacitást:

M*(`sizeof(double)+sizeof(int)`).

A többi erőforrásigénye hasonló az egyenlő osztásközű hisztograméhoz.

Mivel *F(x)* nem ismert, ezt a módszert csak referenciának használtam a következőhöz.

2.4.2.5. Kvázi egyenlő cellavalószínűségű hisztogram

Úgy határozzuk meg a cellahatárokat, hogy statisztikailag ekvivalens blokkokat nyerjünk. Az $F_N(x)$ (az empirikus eloszlásfüggvény) szerint a partíció egyenlő valószínűségű. A hisztogram cellahatárait tehát mindig a gyűjtött mintából számítjuk. Azt várjuk, hogy ha a minta mérete, N elég nagy, akkor a cellahatárok elég közel lesznek az optimálishoz, és a statisztika L_1 hibája közel lesz a fent említett referencia módszer statisztikájának L_1 hibájához.

Ehhez a módszerhez a következő algoritmust javaslom: tároljuk el az X valószínűségi változó mind az N megfigyelését. Azután rendezzük a mintákat és húzzuk meg az M darab cella határait úgy, hogy mindegyik cellába ugyanannyi megfigyelés kerüljön.

A rendezés költsége O(N*log(N)), az igényelt tárkapacitás nagy lehet, fő komponense az N*`sizeof(double)`, emellett a cellahatárok tárolása már elhanyagolható. Elvileg elég lenne csak a cellahatárok átvitele a szegmensek között (körülbelül M*`sizeof(double)` byte), mivel a számláló értékek azonosak minden cellára. A valóságban azonban ezek eltérőek is lehetnek különböző okok miatt:

- a megfigyelések száma nem egész számú többszöröse a cellák számának
- ha a megfigyelt valószínűségi változó kvantált, a megfigyelések között egyenlőek lehetnek, és így ugyanabba a cellába kerülnek.

Ilyen esetben a számlálókat is átvisszük.

2.4.2.6. Más lehetséges statisztikagyűjtési eljárások

A következő két módszer a megfigyelések tárolása nélkül működik: A P^2 (*pi-square*) módszer (Jain and Clamtach 1985) kvantiliseket számol. A K-split módszer (Varga 1997) pedig bizonyos kritériumok alapján történő cella aláosztásokat alkalmaz a cellák számának optimális szinten tartásához az eloszlás jellegének és a megfigyelések számának megfelelően.

Más nemparametrikus eljárások még: a frequency polygon, az averaged shifted histogram és a különböző kernel módszerek. (Scott 1992)

Ezekkel a módszerekkel nem foglalkoztam; az SSM-T szempontjából a vizsgált módszerek kielégítő megoldást adnak, a további módszerek vizsgálata várhatóan nem vezetne új megállapításokhoz.

2.4.2.7. Az erőforrás igények figyelembe vétele

Mikor kiválasztjuk, hogy adott esetben melyik statisztikagyűjtési módszert alkalmazzuk, akkor figyelembe kell vennünk a módszerek fent megvizsgált erőforrás igényeit is, ezeket a 2. táblázatban foglaltam össze. Szükség lehet arra is, hogy megbecsüljük a szegmensek közti üzenetek végrehajtási időegységre jutó mennyiségét. A statisztikagyűjtési módszerek pontossága elsődleges fontosságú, de az is lényeges, hogy megvizsgáljuk, hogy az algoritmusok számítási, tár és kommunikációs igényei a szimuláció lényeges sebességcsökkenése nélkül kielégíthetők-e, ellenkező esetben az SSM-T alkalmazása esetleg nem gyorsulást, hanem lassulást eredményezhet.

Módszer	Számításigény	Tárigény	Kommunikációs igény
Relatív gyakoriság	O(N*log(M)) vagy O(N*log(M')+(M')^2)	M*(sizeof(int)+sizeof(double)) vagy M'*(sizeof(int)+sizeof(double))	M*sizeof(int) vagy M'*(sizeof(int)+sizeof(double))
Egyenlő osztásközű hisztogram	O(N)	M*sizeof(int)	M*sizeof(int)
Barron estimate	O(N)	M*sizeof(int)	M*sizeof(int)
Egyenlő cella-valószínűségű hisztogram	O(N*log(M))	M*(sizeof(int)+sizeof(double))	M*(sizeof(int)+sizeof(double))
Kvázi egyenlő cellavalószínű-ségű hisztogram	O(N*log(N))	N*sizeof(double)+ M*(sizeof(int)+sizeof(double))	M*sizeof(double) vagy M*(sizeof(int)+sizeof(double))

2. táblázat A tanulmányozott statisztikagyűjtési módszerek erőforrásigényei

2.4.3. Diszkrét idejű szimulációban előforduló eloszlások fajtái

Diszkrét idejű szimulációban a valószínűségi változók sok esetben diszkrét eloszlásúak. Egy kommunikációs hálózat szimulációja esetén például a csomaghossz, sorhossz, aktív állomások száma, stb. nemcsak diszkrét, hanem egész értékű. A másik oldalon az események időpontját általában folytonos eloszlású valószínűségi változóként kezelik. Vezessük még be a kvantált (*quantized*) valószínűségi változókat is a következő okokból:

- a számítógépek lebegőpontos változói véges pontosságúak
- a szimulált rendszerben létezhet egy legkisebb (alap) időegység
- ha a szimuláció bemeneti adatai egy valóságos rendszeren való mérésből származnak, kvantáltak lehetnek

- a szimulációban lehetnek vegyes eloszlások (amelyek diszkrét és abszolút folytonos eloszlásokból tevődnek össze)

2.4.4. A statisztikagyűjtés pontossága

Mint már korábban említettem, a parametrikus eljárások általában nem alkalmazhatóak, mivel legtöbbször nem tudjuk pontosan, hogy milyen eloszlás fordul elő az adott szimuláció adott pontján. Ettől azonban még megvizsgálhatjuk a különböző statisztikagyűjtési eljárások viselkedését néhány gyakran használt eloszlásra, részben, mert egyes esetekben az eloszlás jellegét előre tudhatjuk, részben, mert ez ad egy képet a módszer pontosságáról (konvergenciasebességéről), ami segíthet abban, hogy adott szimuláció esetén melyik módszert válasszuk.

2.4.4.1. Exponenciális eloszlás

Az exponenciális eloszlás sokszor előfordul szimulációs modellekben. Poisson eloszlású igény generálás esetén az érkezési időköz eloszlása exponenciális. Kiszolgálási időt is szoktak exponenciális eloszlással modellezni. Az exponenciális eloszlás valószínűségi sűrűségfüggvénye:

$$f(x) = \lambda e^{-\lambda x}$$

A λ paraméter értékét a kísérletekben 1-nek választottam. Kérdésünk az, hogy mi az a legkisebb L_1 hiba egy N megfigyelést tartalmazó mintából való statisztikagyűjtésnél, ami az egyes módszerek alkalmazásával paramétereik optimális megválasztása esetén elérhető?

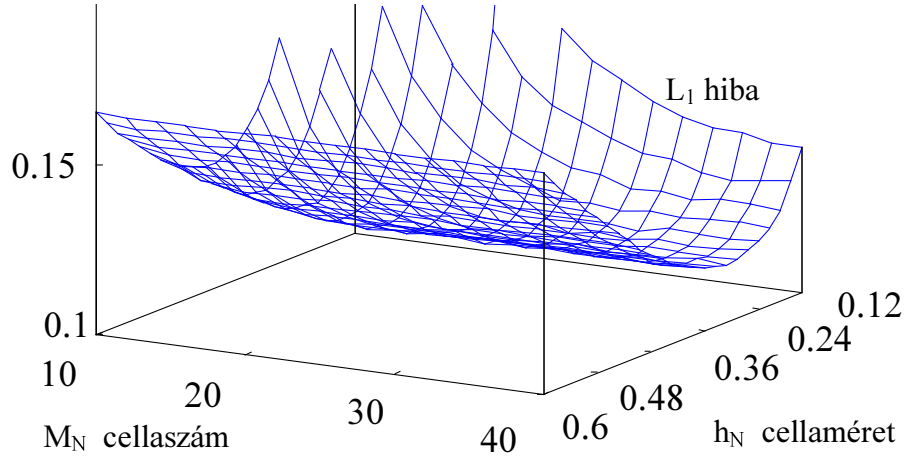
2.4.4.1.1. Egyenlő osztásközű hisztogram

A megfigyelések N számán kívül az L_1 hiba függ a hisztogram paramétereitől is. A hisztogram kezdőpontja nyilvánvalóan t_0=0. A h_N cellaméret és az M_N cellaszám különböző értékei mellett szimulációs kísérleteket hajtottam végre. A megfigyelések száma N=1000 volt, a kísérleteket 1000-szer hajtottam végre, az L_1 hibára kapott értékeket átlagoltam. Az eredmények a 11. ábra ábrán láthatók. A tengelyek irányát és beosztását úgy választottam meg, hogy az L_1 hiba viselkedése a paraméterek függvényében jól megfigyelhető legyen. (A különösen érdeklődő Olvasók számára a pontosabb vizsgálat lehetőségének érdekében a grafikon alapját képező adatokat a 3. táblázatban közlöm.)

A 11. ábra mutatja, hogy adott M_N cellaszám esetén létezik egy h_{Nopt} optimális cellaméret. Ha $h_N > h_{Nopt}$ akkor a felbontás túl durva, ami növekvő hibát okoz, amint h_N nő. Ha $h_N < h_{Nopt}$ akkor a megfigyelések cellánkénti száma N=1000 mellett túl kicsi (túl nagy a fluktuáció), ezért a hiba nő, amint h_N csökken. A paraméterek optimalizálásának érdekében h_N-t változtattam, miközben M_N-t úgy választottam meg, hogy a hisztogram tartománya elég széles legyen, azaz:

$$M_N = \left\lceil \frac{7}{\lambda h} \right\rceil$$

A $7/\lambda$ tartomány elég széles, mert az exponenciális eloszlás ezen tartományon kívül eső farkának mértéke kisebb, mint 0.001, ami a mért L_1 hibákhoz képest elhanyagolható.



11. ábra Az egyenlő osztásközű hisztogram L_1 hibája exponenciális eloszlásból származó 1000 megfigyelés esetén a h_N cellaméret és az M_N cellaszám függvényében

	M_N cellaszám értéke															
$\downarrow h_N \downarrow$	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
0.60	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165
0.58	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161
0.56	0.157	0.157	0.157	0.157	0.157	0.157	0.157	0.157	0.157	0.157	0.157	0.157	0.157	0.157	0.157	0.157
0.54	0.154	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153	0.153
0.52	0.151	0.149	0.149	0.149	0.149	0.149	0.149	0.149	0.149	0.149	0.149	0.149	0.149	0.149	0.149	0.149
0.50	0.147	0.145	0.145	0.145	0.145	0.145	0.145	0.145	0.146	0.146	0.146	0.145	0.145	0.146	0.146	0.146
0.48	0.144	0.142	0.141	0.142	0.141	0.142	0.141	0.142	0.141	0.142	0.142	0.142	0.142	0.141	0.142	0.142
0.46	0.142	0.138	0.138	0.138	0.139	0.138	0.139	0.138	0.139	0.138	0.138	0.138	0.138	0.138	0.138	0.138
0.44	0.140	0.136	0.134	0.135	0.135	0.135	0.135	0.135	0.135	0.135	0.135	0.135	0.135	0.135	0.135	0.135
0.42	0.138	0.133	0.131	0.131	0.131	0.131	0.132	0.132	0.132	0.131	0.132	0.132	0.132	0.132	0.132	0.131
0.40	0.137	0.131	0.129	0.128	0.128	0.128	0.128	0.129	0.128	0.129	0.129	0.129	0.129	0.128	0.129	0.129
0.38	0.137	0.129	0.127	0.125	0.126	0.126	0.125	0.126	0.126	0.126	0.126	0.126	0.126	0.126	0.126	0.126
0.36	0.138	0.128	0.124	0.123	0.123	0.123	0.124	0.123	0.123	0.124	0.123	0.124	0.124	0.123	0.124	0.123
0.34	0.140	0.128	0.124	0.121	0.121	0.121	0.121	0.121	0.121	0.122	0.121	0.121	0.121	0.122	0.121	0.121
0.32	0.143	0.129	0.123	0.120	0.118	0.119	0.119	0.119	0.119	0.119	0.120	0.119	0.119	0.120	0.119	0.120
0.30	0.148	0.131	0.123	0.119	0.118	0.117	0.118	0.118	0.118	0.117	0.118	0.118	0.118	0.118	0.118	0.118
0.28	0.155	0.135	0.125	0.120	0.117	0.117	0.117	0.117	0.117	0.117	0.117	0.117	0.117	0.117	0.117	0.117
0.26	0.166	0.141	0.128	0.121	0.118	0.116	0.116	0.116	0.116	0.116	0.117	0.116	0.116	0.117	0.117	0.116
0.24	0.179	0.151	0.135	0.126	0.120	0.117	0.116	0.116	0.116	0.117	0.116	0.116	0.117	0.117	0.117	0.117
0.22	0.194	0.162	0.144	0.132	0.125	0.120	0.118	0.118	0.118	0.117	0.118	0.117	0.118	0.118	0.118	0.119
0.20	0.216	0.178	0.155	0.140	0.132	0.125	0.122	0.120	0.119	0.119	0.119	0.118	0.120	0.120	0.120	0.120
0.18	0.243	0.201	0.173	0.153	0.142	0.134	0.129	0.126	0.123	0.123	0.122	0.123	0.122	0.123	0.123	0.123
0.16	0.276	0.228	0.196	0.173	0.157	0.146	0.139	0.134	0.131	0.130	0.127	0.128	0.126	0.128	0.127	0.126
0.14	0.318	0.266	0.228	0.200	0.180	0.167	0.154	0.147	0.143	0.140	0.136	0.136	0.133	0.132	0.133	0.134
0.12	0.369	0.314	0.270	0.239	0.214	0.193	0.180	0.168	0.160	0.156	0.150	0.147	0.145	0.144	0.143	0.143

3. táblázat Az egyenlő osztásközű hisztogram L_1 hibája exponenciális eloszlásból származó 1000 megfigyelés esetén a h_N cellaméret és az M_N cellaszám függvényében

h_N	0.13	0.17	0.21	0.25	0.29	0.33	0.37
$E(L_1)$	0.1393	0.1253	0.1194	0.1166	0.1173	0.1193	0.1242
$\sigma(L_1)$	0.0178	0.0164	0.0154	0.0137	0.0120	0.0105	0.0100

4. táblázat Az L_1 hiba a cellaméret függvényében (20 cella, exponenciális eloszlás, 1000 megfigyelés, 1000 kísérlet)

Ahogyan a 4. táblázat mutatja, h_N-re jó választás a 0.25. Az is látható, hogy h_N kis változása nem okoz szignifikáns változást az L_1 hibában. A cellaszámot (M_N) 20-ra választottam, mert a kevesebb cella lényegesen nagyobb L_1 hibát okozott, a nagyobb cellaszám pedig nem eredményezett lényegesen kisebb L_1 hibát.

Hasonlóan más N mintaméretre is meghatároztam az optimális cellaméretet és cellaszámot. A 5. táblázat mutatja az eredményeket a hozzájuk tartozó L_1 hibával együtt. Megjegyzés: a hisztogramgyűjtés tartománya (M_N*h_N) az $5/\lambda$ nagyságrendjében van.

N	500	1000	2000	4000	8000	16000	32000
M_N	17	20	24	33	41	56	67
h_N	0.30	0.25	0.20	0.15	0.12	0.10	0.08
$E(L_1)$	0.1444	0.1166	0.0952	0.0765	0.0627	0.0485	0.0402
$\sigma(L_1)$	0.0188	0.0136	0.0090	0.0067	0.0049	0.0035	0.0025

5. táblázat Az egyenlő osztásközű hisztogram L_1 hibája a megfigyelések számának függvényében (exponenciális eloszlás, optimális cellaszám és cellaméret)

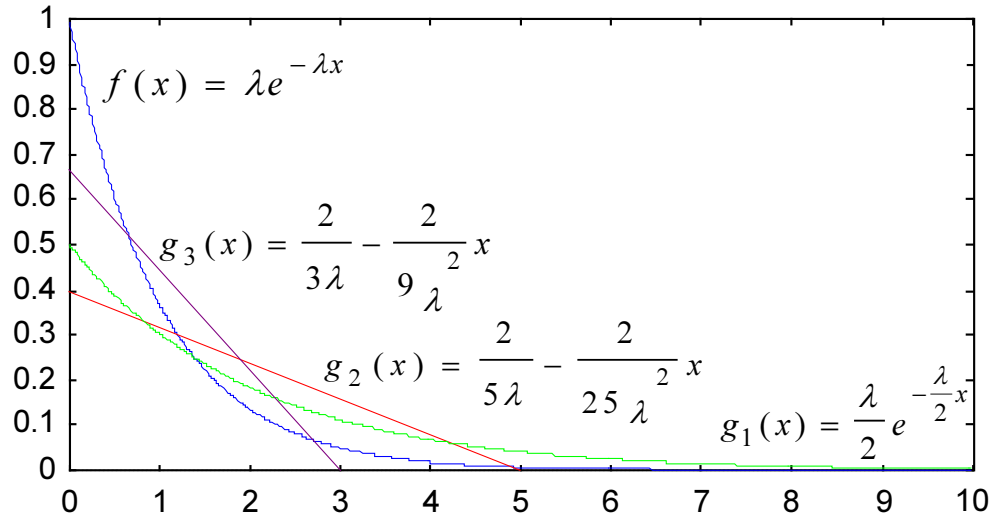
2.4.4.1.2. Barron Estimate

Természetesen a *G(x)≡F(x)* és M_N=1 választás mellett a módszer 0 L_1 hibát eredményez, ezért vigyázni kell rá, hogy becsületesen járjunk el a *G(x)* függvény megválasztásánál.

Vizsgáljuk meg a következő három példát:

$$G_1(x) = F\left(\frac{x}{2}\right), \quad G_2(x) = \frac{2}{5\lambda}x - \frac{1}{25\lambda^2}x^2, \quad G_3(x) = \frac{2}{3\lambda}x - \frac{1}{9\lambda^2}x^2$$

Az első szintén exponenciális eloszlású, de $\lambda/2$ a paramétere, a második és a harmadik pedig a sűrűségfüggvény különböző lineáris közelítését adja. (12. ábra)



12. ábra Sűrűségfüggvények a Barron estimate-hez

Az M optimális értékét a szokott módon határoztam meg, az L_1 hibára vonatkozó eredményeket az 6. táblázat mutatja.

G	$G_1(x)$		$G_2(x)$		$G_3(x)$	
N	1000	8000	1000	8000	1000	8000
M_N	9	20	10	23	5	14
$E(L_1)$	0.0865	0.0438	0.1028	0.0588	0.1716	0.1402
$\sigma(L_1)$	0.0141	0.0062	0.0127	0.0052	0.0133	0.0053

6. táblázat A Barron estimate L_1 hibája különböző G(x) függvényekre (exponenciális eloszlás, optimális cellaszám)

A $G_1(x)$ és a $G_2(x)$ függvények használatával a Barron estimate jobb eredményeket adott, mint az egyenlő osztásközű hisztogram. Ha azonban az eloszlásról nincs előzetes ismeretünk, akkor a módszer nem alkalmazható, $G_3(x)$ egy negatív példa.

2.4.4.1.3. Egyenlő cellavalószínűségű hisztogram

A [0, 5/λ) intervallumot használtam statisztikagyűjtésre, és a cellahatárokat a következőképpen állítottam elő: A [0,1) intervallumot M_N egyenlő részre osztottam. Az osztópontokat az exponenciális eloszlás módosított inverz függvényével transzformáltam, úgy, hogy az utolsó határvonal az 5/λ-ra essen:

$$F^{-1}(x) = \frac{-\ln(1 - xe^{-5})}{\lambda}$$

A cellák M_N számát a következőképpen optimalizáltam: M_N-t addig növeltem, míg a további növelés már nem okozott jelentős változást az L_1 hiba nagyságában. A 7. táblázat mutatja az eredményeket a megfigyelések számának, N-nek néhány értéke mellett.

N	500	1000	2000	4000	8000	16000	32000
M_N	9	12	15	19	24	30	40
$E(L_1)$	0.1871	0.1495	0.1218	0.0989	0.0803	0.0656	0.0529
$\sigma(L_1)$	0.0201	0.0157	0.0105	0.0073	0.0054	0.0039	0.0028

7. táblázat Az egyenlő cellavalószínűségű hisztogram L_1 hibája a megfigyelések számának függvényében (exponenciális eloszlás, optimális cellaszám)

Az egyenlő cellavalószínűségű hisztogram jó megoldásnak tűnt annak a problémának a kezelésére, hogy a kevés megfigyelést tartalmazó cellák relatíve nagy bizonytalansággal (és így hibával) becsülhetők. Az eredményeim viszont azt mutatják, hogy az L_1 hiba értéke az egyenlő cellavalószínűségű hisztogramnál nagyobb, mint azt egyenlő osztásközűnél. Ez egészen meglepő és váratlan. A szimulációs eredményeim ellenőrzése érdekében összehasonlítottam az ideális egyenlő osztásközű és egyenlő cellavalószínűségű hisztogramok L_1 hibáját. Az "ideális" alatt azt értem, hogy a hisztogramokat a valószínűségi sűrűségfüggvény alapján határoztam meg a véges számú megfigyelés helyett. A hisztogamokat a [0, 5/λ) tarto-

mányra képeztem, az L_1 hibába belevettem az $e^{-5} \approx 0.0067$ hibát is, amit az exponenciális eloszlás $[5/\lambda, \infty)$ farkának elhagyása okoz mindkét hisztogram esetén. A 8. táblázat mutatja az eredményeket, amelyek igazolják az intuitív feltételezésemet: az egyenlő cellavalószínűségű hisztogramnak kisebb az L_1 hibája, de a különbség egyre csökken, amint a cellák száma (M) nő.

M	2	4	8	16	32	64
L_1(EqD)	0.5801	0.3106	0.1611	0.0842	0.0455	0.0261
L_1(EqP)	0.5282	0.2846	0.1523	0.0818	0.0450	0.0260

8. táblázat Az ideális egyenlő osztásközű (EqD) és egyenlő cellavalószínűségű (EqP) hisztogramok számított L_1 hibája a cellaszám (M) függvényében exponenciális eloszlás esetén.

Kísérleteket végeztem a két fajta hisztogrammal N=8000 megfigyelés és változó cellaszám mellett. A kísérleteket 1000-szer hajtottam végre, átlagot és szórást számoltam. Az eredmények (9. táblázat) azt mutatják, hogy a véges számú megfigyelésből készített hisztogrammok esetében az egyenlő cellavalószínűségű valóban jobb eredményeket az egyenlő osztásközűnél, ha nagyon kicsi a cellaszám. Nagyobb cellaszám esetén azonban az egyenlő osztásközűnek kisebb az L_1 hibája. A "miért" kérdést nyitva hagyom a matematikusok számára.

M	2	4	8	16	32	64
L_1(EqD)	0.5802	0.3109	0.1625	0.0898	0.0634	0.0676
$\sigma(L_1)$	0.0002	0.0003	0.0008	0.0019	0.0042	0.0062
L_1(EqP)	0.5283	0.2853	0.1557	0.0948	0.0769	0.0841
$\sigma(L_1)$	0.0014	0.0013	0.0022	0.0043	0.0060	0.0063

9. táblázat Az egyenlő osztásközű (EqD) és egyenlő cellavalószínűségű (EqP) hisztogramok L_1 hibája a cellaszám (M) függvényében exponenciális eloszlás esetén. (N=8000 megfigyelés, 1000 kísérlet)

2.4.4.1.4. Kvázi egyenlő cellavalószínűségű hisztogram

A cellák számát nem optimalizáltam, hanem átvettem az egyenlő cellavalószínűségű hisztogramtól. Az eredmények N néhány értéke mellett a 10. táblázatban láthatók. Ez a módszer körülbelül akkora L_1 hibát ad, mint a "tökéletes" egyenlő cellavalószínűségű hisztogram. (Természetesen az L_1 hiba nem kisebb, ilyent semmiképpen sem állíthatunk a szórás ismeretében!)

N	500	1000	2000	4000	8000	16000	32000
M_N	9	12	15	19	24	30	40
$E(L_1)$	0.1845	0.1482	0.1227	0.1019	0.0853	0.0716	0.0588
$\sigma(L_1)$	0.0216	0.0164	0.0113	0.0083	0.0059	0.0042	0.0030

10. táblázat A kvázi egyenlő cellavalószínűségű hisztogram L_1 hibája a megfigyelések számának függvényében (exponenciális eloszlás, cellaszámok az egyenlő cellavalószínűségű hisztogramtól átvéve)

2.4.4.2. Gamma eloszlás

Az *n* számú λ paraméterű exponenciális összege (*n*, *λ*) paraméterű gamma eloszlást követ, aminek a valószínűségi sűrűségfügvénye:

$$f(x;\lambda,n)=\begin{cases}\dfrac{\lambda^n}{(n-1)!}x^{n-1}e^{-\lambda x}, & x>0\\ \\ 0, & x\le 0\end{cases}$$

A kísérleteimhez n=4 and λ=1 paramétereket választottam. A gamma eloszlású véletlenszámokat az eloszlás definíciója alapján generáltam n=4 darab λ=1 paraméterű exponenciális eloszlású véletlenszám összegeként. A kétdimenziós optimalizálást elkerülendő, statisztikagyűjtésre a [0, 11) intervallumot használtam, elhagyva az eloszlás [11, ∞) farkát, aminek mértéke kisebb, mint 0.005.

2.4.4.2.1. Egyenlő osztásközű hisztogram

A megfigyelések N számának értékeihez meghatároztam az M_N optimális cellaszámot. A cellaméret mindig $h_N=11/M_N$. A kísérleteket most is 1000-szer végeztem el. Az eredmények a 11. táblázatban láthatók.

N	500	1000	2000	4000	8000	16000	32000
M_N	13	15	19	24	28	37	45
$E(L_1)$	0.1012	0.0839	0.0678	0.0551	0.0459	0.0369	0.0308
$\sigma(L_1)$	0.0174	0.0122	0.0085	0.0060	0.0042	0.0031	0.0022

11. táblázat Az egyenlő osztásközű hisztogram L_1 hibája a megfigyelések számának függvényében (gamma eloszlás, optimális cellaszám)

Terjedelmi okok miatt célszerű volt elhagyni a Barron estimate és az egyenlő cellavalószínűségű hisztogram vizsgálatát. Ezeket gyakorlati esetben úgysem választanánk, hiszen ismeretlen eloszlásokat kell közelítenünk.

2.4.4.2.2. Kvázi egyenlő cellavalószínűségű hisztogram

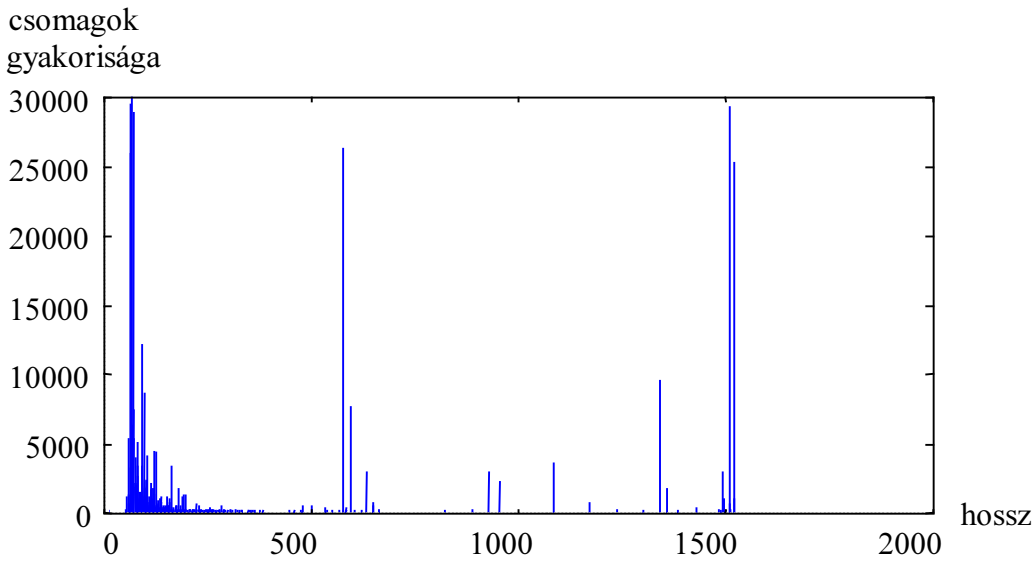
A megfigyelések N számának értékeihez meghatároztam az M_N optimális cellaszámot. Az eredmények a 12. táblázatban láthatók. Az egyenlő osztásközű hisztogram eredményeivel összehasonlítva látható, hogy az egyenlő osztásközű 20-25%-kal kisebb L_1 hibát produkált. Semmi nem igazolja hát a kvázi egyenlő cellavalószínűségű hisztogram alkalmazását. Ha a lehetséges értékek tartománya nem ismert, és a tárkapacitás megengedi, akkor érdemes eltárolni a megfigyeléseket, aztán belőlük megállapítva a hisztogramgyűjtés tartományát, egyenlő osztásközű hisztogramot készíthetünk, és azt küldjük át a megfelelő szegmensnek.

N	500	1000	2000	4000	8000	16000	32000
M_N	12	16	20	23	28	37	45
$E(L_1)$	0.1279	0.1060	0.0888	0.0758	0.0649	0.0535	0.0449
$\sigma(L_1)$	0.0221	0.0168	0.0098	0.0080	0.0053	0.0039	0.0030

12. táblázat A kvázi egyenlő cellavalószínűségű hisztogram L_1 hibája a megfigyelések számának függvényében (gamma eloszlás, optimális cellaszám)

2.4.4.3. Csomaghossz eloszlás egy FDDI gerinchálózaton

Annak érdekében, hogy gyakorlati esetet vizsgáljunk, tekintsük egy létező hálózat csomaghossz eloszlását. A választott hálózat a BME (2.3.2.1. alfejezetben bemutatott) FDDI gerinchálózata. Az adatokat az északi gyűrűből (9. ábra) gyűjtöttem protokoll analizátorral (Lencse 1997). A csomaghossz eloszlás áttekintő képe a 13. ábrán látható. Ez egy egész értékű diszkrét eloszlás a [61-1521] intervallumbeli értékekkel, tehát a lehetséges értékek száma 1461.



13. ábra Csomaghossz eloszlás egy FDDI gerinchálózaton. (A hossz=67-nél lévő oszlop erősen csonkított, a valódi magassága körülbelül 130000.)

Mivel az eloszlást magát nem ismerjük az N=500-32000 méretű mintában mért relatív gyakoriságokat hasonlítottam a teljes rendelkezésemre álló adatmennyiséghez (500000 megfigyelés). Az N méretű mintákat véletlenszerűen választottam ki a teljes adathalmazból. A 13. táblázat mutatja az eredményeket.

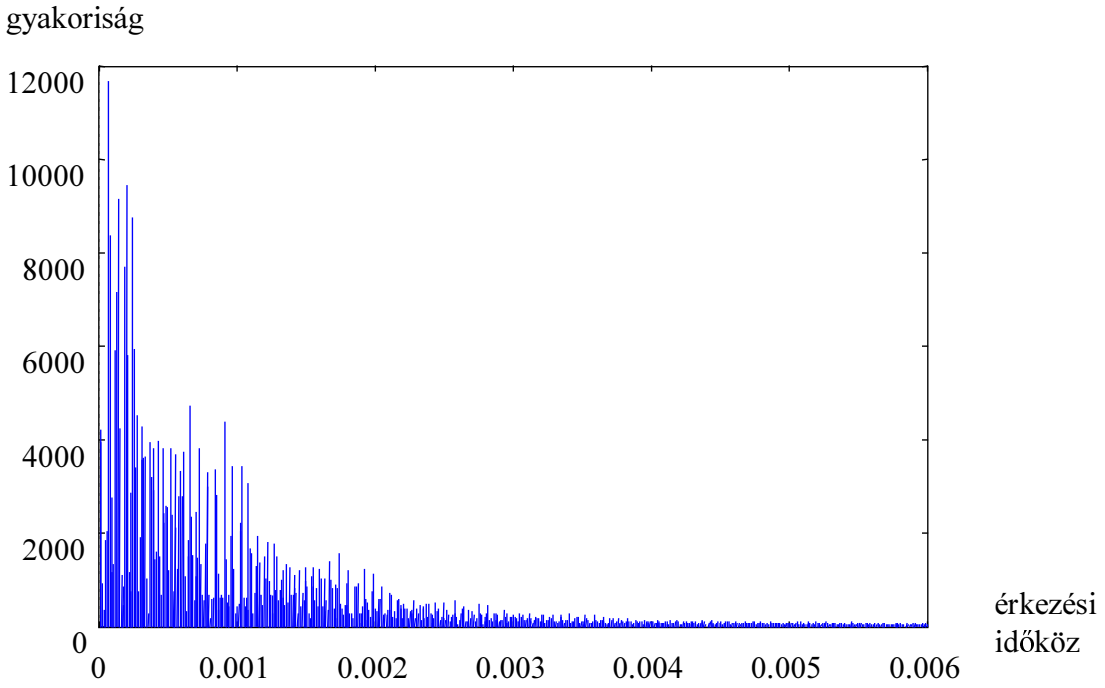
N	500	1000	2000	4000	8000	16000	32000
$E(L_1)$	0.3874	0.2978	0.2242	0.1691	0.1246	0.0904	0.0654
$\sigma(L_1)$	0.0230	0.0189	0.0137	0.0093	0.0062	0.0048	0.0026

13. táblázat A relatív gyakoriság módszerének L_1 hibája (csomaghossz eloszlás egy FDDI gerinchálózaton)

Ennél az eloszlásnál az L_1 hiba a korábbiakhoz képest nagy az N kis értékeire, de a konvergenciasebesség közel $c_1/N^{1/2}$. Ez gyorsabb, mint a többi eloszlásnál tapasztalt $c_2/N^{1/3}$, ami a hisztogramok garantált konvergenciasebessége.

2.4.4.4. Érkezési időköz eloszlás egy FDDI gerinchálózaton

A korábban említett hálózaton a csomagok érkezési időközeit is mértem. Sajnos a protokoll analizátor az időt csak 0.00001s pontossággal tudta rögzíteni. Mivel az FDDI hálózat adatsebessége 100Mbit/s, ez a 10μs 1000 bit ideje. Így az érkezési időköz kvantált valószínűségi változó, a relatív gyakoriság módszerét alkalmaztam.



14. ábra Érkezési időköz eloszlás egy FDDI gerinchálózaton (másodpercekben mérve)

A 14. ábra csak az eloszlás egy részének a közelítő képét mutatja, az eloszlás nincs a [0, 0.006] intervallumra korlátozva. Az eloszlás burkológörbéje az exponenciális eloszlásra emlékeztet. A statisztikagyűjtés tartománya a [0, 0.01] intervallum, az eloszlás elhanyagolt far-

kának mértéke 0.012, ami a 2000 fölötti mintaméretek esetén már az L_1 hiba szignifikáns (10% fölötti) része. Az eredmények a 14. táblázatban láthatók.

N	500	1000	2000	4000	8000	16000	32000
$E(L_1)$	0.1911	0.1478	0.1160	0.0918	0.0702	0.0552	0.0438
$\sigma(L_1)$	0.0243	0.0171	0.0113	0.0084	0.0061	0.0049	0.0034

14. táblázat A relatív gyakoriság módszerének L_1 hibája (érkezési időköz eloszlás egy FDDI gerinchálózaton)

2.4.5. A statisztikagyűjtéssel kapcsolatos megállapításaim

Különböző statisztikagyűjtési eljárásokat hasonlítottam össze többféle tipikus eloszlás esetén, hogy meghatározzam, melyeket célszerű az SSM-T esetén alkalmazni. Mind az erőforrás igényüket, mind a pontosságukat (konvergenciasebességüket) vizsgáltam.

A szimuláció eredményeinek pontossága érdekében a módszerek pontossága elsődleges fontosságú, de az erőforrás (számítás, tár és kommunikáció) igényüket is figyelembe kell venni a szimuláció végrehajtásának hatékonysága érdekében.

A statisztikagyűjtési módszerek hibájának mérésére az L_1 hibakritériumot célszerű alkalmazni.

Abszolút folytonos eloszlásokra a Baron estimate adja a legkisebb L_1 hibát, de az eloszlásról a-priori információra van szüksége, ami az általános esetben nem áll rendelkezésünkre.

Számítás útján megmutattam, hogy exponenciális eloszlás esetén az egyenlő valószínűségű cellájú hisztogram kisebb L_1 hibát ad, mint az egyenlő osztásközű. Az optimálisnál jóval kisebb számú cella esetén ezt mutatta a gyakorlat is, **elegendő számú cella esetén azonban az egyenlő osztásközű hisztogram bizonyult jobbnak, ezért abszolút folytonos eloszlásokra ezt a módszert javaslom**, ennek ráadásul erőforrás igénye is kedvezőbb.

A valós életből vett diszkrét és kvantált eloszlásoknál elfogadható eredményeket kaptam az L_1 hibára a relatív gyakoriság módszerének alkalmazásával.

2.5. A statisztikacsere körüljárása

A (Lencse, 1999b) cikkemben az SSM-T statisztikacseréjének feltételeivel foglalkoztam. Olyan statisztikacsere vezérlő algoritmust mutattam be, amely előrejelzést és szinkronizációs időpont törlést használ. Bevezettem az úgynevezett büntetőfüggvényeket, amelyek a különböző statisztikacsere vezérlő algoritmusok jóságának a mérésére adnak matematikai kritériumot. A problémát mind analitikusan, mind – egy speciális esetre vonatkozóan – szimulációval kezeltem. A dolgozatomnak ez a fejezete a cikk megfelelő részeinek szabad magyar fordításán alapul.

2.5.1. Mikor cseréljünk statisztikákat?

A statisztika “csere” kifejezés azt implikálja, hogy a statisztika átadás két szegmens között kölcsönös, abban a virtuális időpontban, amikor az **A** szegmens output interfésze elküldi a gyűjtött statisztikáit a **B** szegmens input interfészének, a **B** szegmens output interfésze is elküldi a gyűjtött statisztikáit az **A** szegmens input interfészének. Az adatáramlás kommunikációs hálózatok esetén a szegmensek között általában valóban kétirányú, de a statisztikacsere optimális ideje nem feltétlenül esik egybe a két irányt illetően. Most csak egy irányt vizsgálunk, az **A** szegmens küld statisztikát a **B** szegmensnek, a 2. ábra szerinti összeállításban. (Ez a 10. oldalon található.)

2.5.1.1. A laza időszinkronizációból következő feltétel

Az **A** szegmensnek az (i-1)-edik statisztikacsomag elküldésekor közölnie kell a **B** szegmenssel, hogy az i-edik statisztikacsomag küldésének t_i virtuális idejét. A t_i virtuális időpontig így a **B** szegmens az **A**-tól függetlenül végrehajtható. Mint azt már korábban láttuk, t_i-ben a **B** szegmensnek vagy várnia kell az **A** szegmenstől jövő statisztika csomagra, vagy az már megérkezett, ez a két szegmens végrehajtásának és az azokat végrehajtó processzorok közti kommunikációs csatornának a sebességétől függ. A lényeg, hogy a statisztikát a **B** szegmens a t_i virtuális időpontban veszi figyelembe.

2.5.1.2. A szimuláció pontosságát biztosító feltétel

A kívánt pontosság elérése érdekében, azaz, hogy az L_1 hiba a kívánt határ alatt legyen, a statisztikának megfelelő N számú megfigyelésen kell alapulnia, ahol N függ a statisztikagyűjtési módszertől, a megfigyelések valószínűségi eloszlásától és az L_1 hiba előírt szintjétől, amint ezt az előző fejezetben láttuk. Ezt a követelményt egyszerűen kielégítené a korábban SSM-C-nek nevezett megoldás.

2.5.1.3. A javasolt megoldás

A statisztikacserének tehát két fajta triggerfeltételét láttuk, az egyik az adott virtuális időpont elérése, a másik az adott számú megfigyelés beérkezése. A két feltétel közti ellentét feloldására a következő megoldást dolgoztam ki: A kapcsolat az N számú üzenet és aközött a $\Delta t_i=t_i-t_{i-1}$ hosszúságú $T_i=[t_{i-1}, t_i)$ virtuális időintervallum között, amelyben ezeket megfigyeltük, az *i-edik intervallumban mérhető üzenetgyakoriság* (*message rate in the i-th interval*) $R_i=N/\Delta t_i$. Természetesen az általános esetben R_i még nem ismert a t_{i-1} virtuális időpontban, csak előre jelezhető a korábbi R_j, $j<i$ értékek alapján. A legegyszerűbb *előrejelzés* (*prediction*) az $R_i=R_{i-1}$, amely jó, ha az *üzenetfolyam sebessége* (*message rate*) lassan változik. Természetesen az előrejelzés adathiánnyal küzd a szimuláció kezdetén, ebben segíthetnek az a-priori ismereteink (korábbi szimulációk, analitikus közelítések, stb.).

Ha a megfigyelések száma kisebb, mint N, akkor a gyűjtött statisztikák pontossága nem kielégítő, így nem tudjuk garantálni a szimuláció eredményeinek megkívánt pontosságát. Ha a megfigyelések száma lényegesen nagyobb, mint N, akkor a statisztikák átvitelét szükségtelenül késleltettük az adott virtuális időpontig, és az SSM-T által okozott tranziens hosszabb, mint amilyen az optimális esetben lett volna. Ez megint csak az eredmények pontosságát rontja.

Az eredmények pontosságának biztosítása érdekében vezessük be a következő módszert: Az **A** szegmens **B** szegmensnek a statisztikaküldés idejére korábbi virtuális időpontot üzen, mint ami az előrejelzésből jön. Ha a statisztikacsomag elkészül erre a virtuális időpontra, akkor elküldi azt, egyébként pedig a **B** szegmenst értesíti arról az új előrejelzett virtuális időpontról, ameddig az a szimulációval elmehet, törölve ezzel az aktuális szinkronizációs időpontot. Ezt addig folytatja, amíg el nem készül a kívánt pontosságú statisztikacsomag, amit elküldhet a **B** szegmensnek. Annak elküldésekor az előrejelzésben már felhasználja az újonnan szerzett tapasztalatokat is. Ezzel a módszerrel a kívánt pontosságot biztosítottuk. A törölt szinkronizációs időpontok számát azonban alacsonyan kell tartani, különben a szimuláció által nyújtott gyorsulás nem lesz kielégítő.

Az optimális esettől tehát 3 fajta eltérés van:

1. túl sok a törölt szinkronizációs időpont
2. a statisztikák túl kevés megfigyelésen alapulnak
3. a tranziens túl hosszú (túl sok a megfigyelés)

Mindegyiknek megvan a maga negatív hatása a szimuláció eredményeire vagy végrehajtási idejére.

2.5.1.4. A büntetőfüggvények

Egy *statisztikacsere vezérlő algoritmus* (*statistics exchange control algorithm*) tehát megmondja, mikor kell elküldeni a statisztikákat, létrehozni vagy törölni egy szinkronizációs időpontot. Egy ilyen algoritmus jóságának a mérésére vezessük be a *büntetőfüggvényeket* (*penalty function*s). Ezek azt fejezik ki, hogy mennyire tartjuk rossznak a korábban említett három fajta eltérést az optimálistól. Az eltérés büntetését a mintában lévő megfigyelések *n* számának függvényében adják meg.

A szinkronizációs időpontok által okozott többletmunka arányos azok számával, azaz fordítottan arányos a mintában lévő megfigyelések *n* számával. A büntetőfüggvény:

(1) $P_{SP}(n) = c_1 / n, \quad n > 0$

Amint korábban láttuk, ha hisztogramot használunk statisztikagyűjtésre, az L_1 hiba a legrosszabb esetben $n^{-1/3}$-nal arányos. A büntetőfüggvény meghatározása érdekében ismernünk kell azt a függvényt, ami a szimuláció kimenetének hibáját adja meg a statisztikák L_1 hibájának függvényében. Természetesen ez a függvény nem adható meg általánosságban, csak az adott szimuláció ismeretében. Ha az a követelményünk, hogy az L_1 hiba kisebb legyen egy előre definiált értéknél, és ezen kívül nem foglalkozunk a pontos értékével, egy nagyon egyszerű büntetőfüggvény a következő:

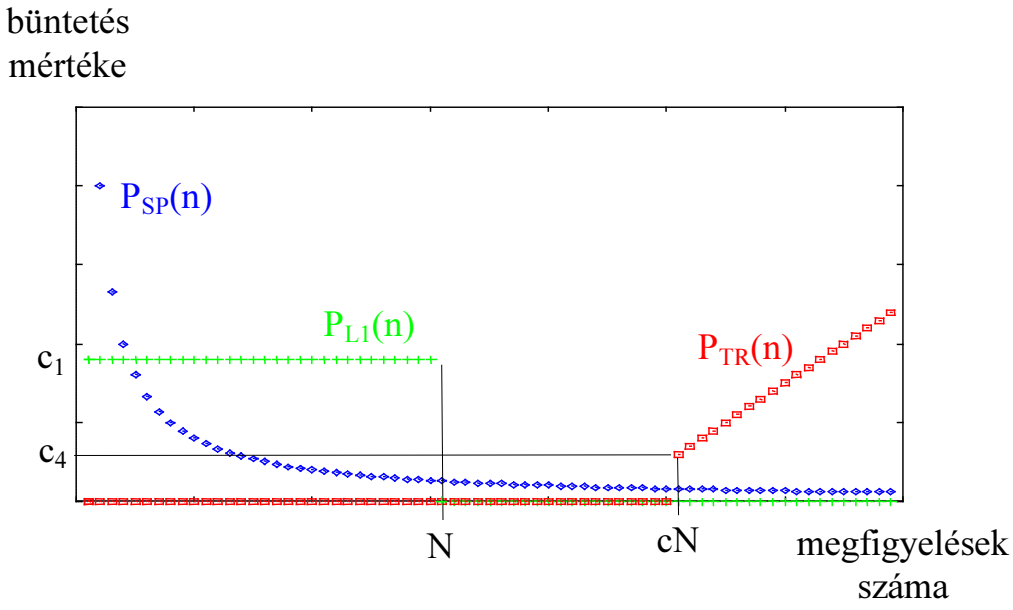
(2) $$P_{L_1}(n) = \begin{cases} c_2, & 0 < n < N \\ 0, & n \geq N \end{cases}$$

Hasonlóan, a tranziens által okozott hiba, és így a büntetés is függ az adott rendszertől, amit szimulálunk. Egy logikus választás a következő: Ha *n* csak egy kicsit nagyobb, mint *N*, az

nem érdemel túl nagy büntetést, de a lényegesen nagyobb értékek egyre nagyobb büntetést érdemelnek. Egy példa:

$$(3)\quad P_{TR}(n) = \begin{cases} 0, & 0 < n < cN \\ c_3 n + c_4, & n \geq cN \end{cases}$$

Az említett példa büntetőfüggvények a 15. ábrán láthatók.



15. ábra Példa büntetőfüggvények

A *teljes büntetőfüggvény* (*overall penalty function*) a három büntetőfüggvény összege. Természetesen ez az összeg ekvivalens a lineáris kombinációval, mert a büntetőfüggvényeknek van olyan szabadsági foka, ami alkalmas a súlyozásra. Az összeget azért tartottam jobbnak a szorzatnál, mert így a büntetőfüggvények egymástól függetlenül hangolhatók; a nagyon kis vagy 0 értékek is megengedettek.

$$(4)\quad P_{\Sigma}(n) = P_{SP}(n) + P_{L1}(n) + P_{TR}(n)$$

2.5.2. Statisztikacsere vezérlő algoritmusok tervezési kritériuma

Egy optimális statisztikacsere vezérlő algoritmusnak minimalizálnia kell a büntetés várható értékét. A büntetés várható értéke a következőképpen számítható ki. Tételezzük fel, hogy a megfigyelések (azaz a csomagok) érkezési időköz eloszlása exponenciális λ paraméterrel. Az output interfészhez érkező csomagok száma egy [0,T] virtuális időintervallumban egy λ intenzitású Poisson folyamattal írható le.

$$(5)\quad p_T(n) = \frac{(\lambda T)^n}{n!} e^{-\lambda T}$$

A legegyszerűbb esetben, amikor nincs szinkronizációs időpont törlés, hanem a gyűjtött statisztikákat minden szinkronizációs időpontban továbbítják, a statisztikacsere vezérlő algoritmus feladata, hogy úgy válassza meg T-t, hogy a P(T) büntetés várható értéke minimális legyen.

$$(6)\quad \sum_{n=1}^{\infty}\frac{(\lambda T_k)^n}{n!}e^{-\lambda T}\frac{c_1}{n}$$

Ha megengedjük a szinkronizációs időpontok törlését, akkor a teljes büntetőfügvényt módosítani kell:

$$(7)\quad P_{k\Sigma}(n,n') = P_{SP}(n) + I_k P_{L_1}(n') + I_k P_{TR}(n'),$$

ahol I_k=1, ha a statisztikacsomagot tényleg elküldik a k. szinkronizációs időpontban, és I_k=0 különben; *n* a megfigyelések száma a legutolsó (esetleg törölt) szinkronizációs időponttól, *n′* pedig az utolsó statisztika csomag elküldéstől a megfigyelések kumulált száma.

Jó lenne olyan K és T_1, T_2, ..., T_k, ..., T_K, értékeket találni, amelyek minimalizálják a P(K, ..., T_k, ...) büntetés várható értékét.

$$(8)\quad E\{P(K,...,T_k,...)\} = \sum_{k=1}^{K}\sum_{n=1}^{\infty}p_{T_k}(n)P_{SP}(n) + \sum_{n'=1}^{\infty}p_T(n')(P_{L_1}(n') + P_{TR}(n')), \text{ ahol } T = \sum_{k=1}^{K}T_k$$

A fejezet elején exponenciális érkezési időközt tételeztünk fel. Ha ez nem állna is fent, a Poisson folyamat akkor is jó közelítés N nagy értékei esetén. Mint a 2.4. fejezetben láttuk, N tipikus értékei 1000 körüliek, vagy nagyobbak, ami már elég nagy. A közelítés esetén a λ paramétert a következőképpen kell megválasztani:

$$(9)\quad \lambda = \frac{1}{\text{átlagos érkezési időköz}}$$

Használjuk az (1) és (2) által definiált büntetőfüggvényeket, és az analitikus kezelhetőség érdekében legyen:

$$(3')\quad P_{TR}(n) = c_3 n^2$$

A gyorsulás büntetés várható értéke egyetlen T_k intervallumban, *n* megfigyelés esetén a következőképpen számítható:

$$(10)\quad \sum_{n=1}^{\infty}p_{T_k}(n)P_{SP}(n) = \sum_{n=1}^{\infty}\frac{(\lambda T_k)^n}{n!}e^{-\lambda T}\frac{c_1}{n} = c_1\mathrm{E}\left\{\frac{1}{U}I_{\{U\geq 1\}}\right\} \leq c_1\left(\frac{1}{\lambda T_k} + \frac{3}{(\lambda T_k)^2}\right) = c_1\frac{\lambda T_k + 3}{(\lambda T_k)^2}$$

Az egyenlőtlenség bizonyítása megtalálható (Beirlant and Győrfi, 1998).

Az L_1 hiba büntetése a teljes T intervallumtól és a megfigyelések kumulált *n′* számától függ:

$$(11)\quad \sum_{n'=1}^{\infty}p_T(n')P_{L_1}(n') = \sum_{n'=1}^{\infty}\frac{(\lambda T)^{n'}}{n'!}e^{-\lambda T}c_2 I_{n'<N} = c_2\sum_{n'=1}^{N-1}\frac{(\lambda T)^{n'}}{n'!}e^{-\lambda T}$$

A tranziens büntetés pedig:

$$(12)\quad \sum_{n'=1}^{\infty} p_T(n')P_{TR}(n') = \sum_{n'=1}^{\infty} \frac{(\lambda T)^{n'}}{n'!} e^{-\lambda T} c_3 n'^2 = c_3 \mathrm{E}\{U^2\} = c_3\left((\lambda T)^2 + \lambda T\right)$$

A teljes büntetés várható értéke:

$$(8')\quad E\{P(K,...T_k...)\} = c_1 \sum_{k=1}^{K} \frac{\lambda T_k + 3}{(\lambda T_k)^2} + c_2 \sum_{n'=1}^{N-1} \frac{(\lambda T)^{n'}}{n'!} e^{-\lambda T} + c_3\left((\lambda T)^2 + \lambda T\right), \text{ ahol } T = \sum_{k=1}^{K} T_k$$

Értelmezzük az eredményt! A c_1, c_2 és c_3 konstansok megválasztása egy kompromisszumot jelent az eredmények pontosságának garantált szintje és a párhuzamos szimuláció gyorsulásának mértéke között. A $c_2 \to \infty$ választással garantálható, hogy az L_1 hibára való kritériumunk mindig teljesül. Ebben az esetben azonban fel kell adnunk egy szabadsági fokot, K-t. A statisztikacsomagot csak akkor küldhetjük el, ha már megvan a szükséges számú megfigyelés, különben a szinkronizációs időpontot törölnünk, a statisztikagyűjtést pedig folytatnunk kell. Ez a megoldás minden esetben szigorúan biztosítja az L_1 hiba megfelelő szintjét, de van egy negatív következménye: az algoritmusnak ez a változása befolyásolja a teljes büntetés várható értékének másik két komponensét is (gyorsulás büntetés és tranziens büntetés). Így ebben az esetben nem használhatjuk az analitikus eredményeinket. Erre az esetre szimulációs eredményeket fogok bemutatni a következő fejezetben.

2.5.3. Szimulációs eredmények speciális esetekre

Mint az előző fejezetben említettem, egyáltalán nem engedjük meg, hogy a statisztikacsomag L_1 hibája nagyobb legyen egy előre meghatározott értéknél, azaz mindig megköveteljük, hogy a statisztikáink legalább N számú megfigyelésen alapuljanak.

Tételezzük fel, hogy az előrejelzésünk szerint a megfigyelések *n'* kumulált száma T idő alatt éri el N-et. Hogyan válasszuk meg a T_i virtuális időintervallumok hosszát, hogy minimalizáljuk a büntetést? Válasszunk néhány numerikus értéket: N=1000, λ=1, T=1000, c_1=10, c_3=10^{-6}. A szinkronizációs időpontok virtuális idejének meghatározására két módszert teszteltem:

a) $t_i = i\alpha T, \quad i \geq 1$

b) $t_i = \begin{cases} \beta \dfrac{T}{2}, & i = 1 \\ T_{i-1} + \beta \dfrac{T}{2^i}, & i \geq 2 \end{cases}$

Mindkét esetben addig folytattam a statisztikagyűjtést, míg el nem értem: n'≥N. Kísérleteket végeztem α és β különböző értékeire. Az egymást követő megfigyelések (üzenetek) közt eltelt időt λ paraméterű exponenciális eloszlás szerint sorsoltam. Az i-edik szinkronizációs időpontban (t_i) a statisztikacsomagot akkor tekintettem késznek, ha legalább N megfigyelésből állt (n'≥N). Különben a szinkronizációs időpontot töröltnek tekintettem és a statisztikagyűjtést (azaz az üzenetek számlálását) folytattam. A 15. táblázat 16. táblázat tartalmazza az eredményeket. Minden érték 10000 kísérleten alapul.

α	gyorsulás büntetés		tranziens büntetés		teljes
	átlag	szórás	átlag	szórás	büntetés
0.05	4.1838	0.2720	1.0507	0.0315	5.2345
0.10	1.0599	0.0796	1.1026	0.0675	2.1624
0.15	0.4733	0.0262	1.1228	0.0728	1.5961
0.20	0.2760	0.0320	1.2138	0.1716	1.4898
0.25	0.1807	0.0244	1.2755	0.2309	1.4562
0.30	0.1338	0.0041	1.4400	0.0841	1.5738
0.35	0.0875	0.0083	1.1510	0.1686	1.2385
0.40	0.0752	0.0022	1.4411	0.0848	1.5163
0.50	0.0499	0.0111	1.6003	0.5662	1.6502
0.60	0.0334	0.0010	1.4426	0.0839	1.4759
0.70	0.0286	0.0008	1.9618	0.1057	1.9905
0.80	0.0250	0.0006	2.5615	0.1288	2.5866
0.90	0.0222	0.0006	3.2410	0.1610	3.2632
1.00	0.0150	0.0053	2.4818	1.4280	2.4968

15. táblázat A büntetési értékek az "a" módszer szerinti szinkronizációs időpont generálással

β	gyorsulás büntetés		tranziens büntetés		teljes
	átlag	szórás	átlag	szórás	büntetés
1.10	0.3453	0.7768	1.0591	0.0390	1.4044
1.15	0.1825	0.0790	1.0747	0.0467	1.2572
1.20	0.1251	0.0343	1.1136	0.0598	1.2387
1.50	0.0401	0.0016	1.2672	0.0754	1.3073
2.00	0.0201	0.0103	1.6186	0.5668	1.6387

16. táblázat A büntetési értékek a "b" módszer szerinti szinkronizációs időpont generálással

Az első esetben az α=0.35 érték tűnik a legjobb választásnak, bár a tranziens büntetés szórása olyan nagy, hogy csak annyit mondhatunk, hogy egy 0.15 és 0.60 közötti érték jó választásnak tűnik α-ra. Megfigyelhető, hogy α>0.25 esetén a tranziens büntetés értékei α függvényében ingadozást mutatnak. A magyarázat a következő: A t_i értékek összemérhetők T-vel. Az n' érje el N-et a K-adik szinkronizációs időpontban. K értéke ekkor körülbelül:

$$K = \left\lceil \frac{T}{T_i} \right\rceil = \left\lceil \frac{T}{\alpha T} \right\rceil = \left\lceil \frac{1}{\alpha} \right\rceil$$

A tranziens büntetés arányos $(K\alpha)^2$-tel, például α=0.40-re K=3 és a tranziens büntetés 1.2^2=1.44. Természetesen az 1 alsó korlátja a tranziens büntetésnek.

A második esetben β-ra, az 1.15 és 1.5 közötti értékek tűnnek jónak, azonban ebben az esetben a gyorsulás büntetés szórása okoz bizonytalanságot. Óvatosaknak kell lennünk a β értékeinek megválasztásával, mert a t_i értékek βT-hez konvergálnak. Ha β nagyon közel van 1-hez, akkor az nagyon nagy számú (törölt) szinkronizációs időpontot, és így nagy tranziens büntetést eredményezhet.

Számunkra most nem az α és β optimális értéke a fontos, hanem a lényeg az, hogy: a szimulált rendszertől, a c_i konstansoktól, és a t_i szinkronizációs időpontok meghatározására használt módszertől függően meghatározható az optimális eset, de ez függ a c_i konstansok értékeitől, azaz a szimuláció készítőjének kell megtalálnia a különböző fajta hibák között a számára megfelelő egyensúlyt.

Egy másik nagyon fontos megállapítás, hogy az α és β paraméterek széles tartományára közel optimális eredményeket kaptunk, így az SSM-T kellően robusztus ahhoz, hogy toleráljon bizonyos pontatlanságot a megválasztásukban.

2.5.4. A statisztikacserével kapcsolatos megállapításaim

Előrejelzés és szinkronizációs időpont törlés segítségével működő statisztikacsere vezérlő algoritmust dolgoztam ki a statisztikacsere virtuális időpontjának meghatározására.

Büntetőfüggvényeket vezettem be, amelyekkel mérni tudom a statisztikacsere virtuális időpontjának az optimálistól való eltérésének különböző hatásait. Ezek felhasználásával analitikusan meghatároztam a teljes büntetés várható értékét.

A javasolt statisztikacsere vezérlő algoritmus használata esetén a konstansok megválasztása kompromisszumot jelenthet az eredmények pontosságának garantált szintje és a párhuzamos szimuláció gyorsulásának mértéke között.

Szimulációs vizsgálattal arra a következtetésre jutottam, hogy az adott feltételek mellett az SSM-T kellő robusztusságot mutat, azaz jól tolerálja paramétereinek az optimálistól való eltérését.

3. A jövőbeli események halmazának kezelése

A (Lencse, 1995) cikkemben diszkrét idejű szimulátorokban a jövőbeli események halmazának (FES: *Future Event Set*) tárolására alkalmazott adatszerkezetekkel foglalkoztam. 7 adatszerkezetet vizsgáltam meg szimulációval és hasonlítottam össze abból a szempontból, hogy melyik a legalkalmasabb a FES tárolására. A kulcsösszehasonlítások és pointerhivatkozások számát, valamint a felhasznált CPU időt mértem. Megvizsgáltam a különböző processzor architektúrák hatását. A modell számos paraméterét ortogonálisan változtattam, hogy az adatszerkezetek algoritmusainak jellemzőit meghatározzam.

3.1. Bevezetés

Minden diszkrét idejű szimulátor tartalmaz valamilyen adatszerkezetet, amit a már felidőzített, a jövőben bekövetkezendő események tárolására használ. Régebben esemény listának is hívták, ami elárul valamit a szokásos implementációs megoldásokról. Jó elnevezésnek tartom a *jövőbeli események halmaza* kifejezést, mert ez elfedi az implementációt. Angolul több elnevezés is ismert, a *Future Event Set* (FES) ennek a pontos megfelelője, de találkoztam már a *Pendig Event Set* illetve egyszerűen csak *Event Set* kifejezésekkel is. A továbbiakban a FES rövidítést fogom alkalmazni. Nagy méretű modellek esetén egy szimulátor sebességét jelentősen befolyásolja, hogy a FES tárolására milyen adatszerkezetet használnak.

A legismertebb adatszerkezeteket előttem már sokan tanulmányozták, műveleteik teljesítmény jellemzőit meghatározták (Aho et al. 1975; Knuth 1973; Wirth 1976). Ezekben a vizsgálatokban egy adott adatstruktúra valamilyen műveletének költségét (pl. kulcsösszehasonlítások száma) az adatszerkezetben tárolt elemek számának függvényében fejezik ki. A hivatkozott vizsgálatok azt feltételezik, hogy a kulcsok véletlenszerűek (egyenletes eloszlás), a keresés műveletek száma jóval magasabb, mint a beszúrásoké, vagy törléseké, illetve a keresendő vagy törlendő elemek kiválasztása is véletlenszerű. A FES esetében azonban a kulcsok nem egyenletes eloszlás szerintiek, hanem növekvő tendenciát mutatnak; az esetek többségében a legelső (legkisebb kulcsú) elemet törlik ki és beszúrnak néhány (0 vagy több) új elemet. Előfordul néha véletlenszerűen választott elemek törlése, de keresési műveletre egyébként nincs szükség. A FES igényei tehát eltérnek az adatbázis kezelésnél megszokottaktól. Reeves (Reeves 1984) megvizsgálta a láncolt listák és a kupac (*heap*) variánsait, de a különböző faszerkezetek és a skip list (Pugh 1990) további vizsgálatot igényel a FES tárolásának szempontjából.

3.2. A vizsgálati modell

Annak érdekében, hogy a vizsgálataimhoz alkalmas modellt alkothassak, az eseményvezérelt diszkrét idejű szimuláció korábban ismertetett algoritmusából indultam ki. (Az olvasó a 2.1.1. fejezetben találja meg az algoritmust.)

Az algoritmusban az “első esemény” a legkisebb időbélyegű eseményt jelenti.

A FES modellje egy absztrakt adatstruktúra, *beszúrás* és *első* műveletekkel, ami új elem beszúrását illetve a legkisebb időbélyegű elem kivételét (törlést is beleértve) jelenti. Ezeken kívül szükség van még egy harmadik műveletre is: néha szükséges egy véletlen elem *törlés*e a FES-ből. (Ez például time-out kezeléséhez kell.)

Nem egy létező szimulátort vizsgáltam, amely egy adott rendszer végrehajtását végzi, hanem terhelési modellt alkottam, ami FES számára olyan igénybevételt jelent, ami tipikus egy szimulátorban. A modell megalkotásakor a következő feltételezésekkel éltem:

1. feldolgozása közben egy esemény egy vagy több új eseményt generálhat
2. az aktuális esemény időbélyege és az újonnan generált esemény időbélyege közötti időintervallum valamilyen valószínűség eloszlást követ
3. a *törlés* műveletben az események kiválasztása a FES-ből véletlenszerű, egyenletes eloszlással

A különböző adatstruktúrák és algoritmusaik teljesítményének vizsgálatához a következő jellemzőket mértem:

- a szükséges kulcsösszehasonlítások száma
- a szükséges pointerhivatkozások száma
- a felhasznált processzoridő

Természetesen az utóbbi függ a szimuláció végrehajtásához használt processzor architektúrájától.

A modell viselkedését számos paraméter együttese határozza meg.

3.2.1. A modell paraméterei

3.2.1.1. A FES állapota

Két tipikus állapotot vizsgáltam:

- a szimuláció állandósult állapotában a FES-ben az események száma állandó: a modellben minden esemény pontosan egy új eseményt termel
- a tranziens állapotban a FES-ben az események száma növekszik: a modellben minden esemény két új eseményt termel

3.2.1.2. Az események száma a FES-ben

A vizsgálataim különösen az egyes adatszerkezetek aszimptotikus viselkedésére irányultak, ezért mérés sorozatokat hajtottam végre a FES-ben tárolt események *n* számának változtatása mellett. Az *n* értékeit úgy választottam meg, az egymást követő értékek logaritmusainak különbsége közel állandó legyen. A használt *n* értékek a következők voltak: 10, 21, 46, 100, 215, 464, 1000, 2154, 4641, 10000.

3.2.1.3. A szimulációs lépések száma

Az előbbiekből adódóan 2*n* darab *beszúrás*t és *n* darab *első* műveletet hajtottam végre a tranziens állapotban. Az állandósult állapotbeli első és beszúrás műveletek *m* számát pedig úgy határoztam meg, hogy az kellően nagy legyen a mérések pontosságának biztosításához.

3.2.1.4. Az érkezési időköz eloszlása

A feldolgozás alatt álló esemény és az újonnan generált esemény időbélyegének különbsége a második feltételezés szerint valamilyen valószínűség eloszlást követ. A kísérletekben az alábbi eloszlásokat alkalmaztam:

- egyenletes a [0, 1] intervallumban
- exponenciális eloszlás $\lambda=1$ intenzitással
- normális eloszlás: várható érték=1, szórás=0.3
- normális eloszlás: várható érték=1, szórás=0.1

(ahol a normális eloszlások a $t \geq 0$-ra csonkoltak)

3.2.1.5. A törlések aránya

A FES-ből az elemek nagyrészt az *első* művelettel kerülnek kivételre. (A kivétel alatt mindig az onnan való törlést is értem.) A harmadik feltételezés szerint az események bizonyos hányadát a *törlés* művelettel vesszük ki. Az alkalmazott értékek 0%, 5%, 10%, 20%, 50%. (A 0% azt jelenti, hogy minden elemkivétel az *első* művelettel történik, az 50% pedig azt, hogy a véletlenszerűen kiválasztott elemek törlésének száma egyenlő a legkisebb kulcsúként kivett elemek számával.)

3.2.1.6. A processzor típusa

Mint korábban említettem, a műveletek ideje függ az alkalmazott CPU típusától. Egy olyan processzoron, amelyik nem rendelkezik hardver lebegőpontos támogatással a kulcsösszehasonlítások száma dominál. De ha a lebegőpontos műveleteket a hardver támogatás felgyorsítja, akkor a pointerhivatkozások és az adatstruktúra karbantartásához szükséges egyéb utasítások hatása jobban befolyásolja a FES műveleteinek idejét. Az összehasonlítás céljából három különböző processzort használtam: DEC ALPHA, Intel 80486-DX, Intel 80486-DLC. (További részletek a 3.2.2.2. alfejezetben.)

3.2.1.7. A megvizsgált adatszerkezetek

A FES tárolására alkalmazott 7 adatszerkezet:

- rendezett, egyirányban láncolt lista
- bináris fa
- AVL-fa
- B-fa
- 2-3-fa
- kupac

- skip-list

(Aho et al. 1975; Knuth 1973; Pugh 1990; Wirth 1976)

3.2.1.8. A paraméterek ortogonalitása

A leírt hét paraméter közül hatot ortogonálisan változtattam, azaz értékük összes lehetséges kombinációját tekintettem. Kivétel az *m*, az állandósult állapotban végzett szimulációs lépések száma.

3.2.2. A vizsgálathoz használt szimuláció

3.2.2.1. A vizsgálat algoritmusa

A hét adatszerkezet korábban említett teljesítmény karakterisztikáinak a vizsgálatához egyszerű diszkrét idejű szimulációk sorozatait hajtottam végre. Eseményeket szúrtam be a FES-be és azokat az *első* illetve a *törlés* műveletekkel vettem ki. A diszkrét idejű szimuláció korábban említett algoritmusa szerint jártam el, azaz:

A FES-ből egy esemény kivétele a *törlési arány*-nak megfelelő valószínűséggel a *törlés*, és (1-*törlési arány*)-nak megfelelő valószínűséggel az *első* művelettel történt. A "MOST" változóba csak az *első* művelet esetén írtam bele a kivett esemény idejét. A FES-ből történő minden elemkivétel után tranziens állapotban két, stacionárius állapotban pedig egy új eseményt szúrtam be MOST+(az aktuális eloszlás szerinti véletlenszám) időbélyeggel.

Az egyes kísérletek között változtattam a paramétereket, és minden használt paraméter kombinációval 100 kísérletet végeztem. A kulcsösszehasonlítások és pointerhivatkozások számából, valamint a felhasznált CPU időből a 100-100 eredményre átlagot és szórást számoltam. Az empirikus szórás képletét alkalmaztam:

$$\sigma = \sqrt{\frac{\sum(x^2) - N \cdot \bar{x}^2}{N-1}}$$

ahol N=100 a kísérletek száma, $\bar{x}$ pedig a 100 darab x érték átlagát jelenti.

3.2.2.2. A kísérletek reprodukálásához szükséges részletek

Minden program C++ nyelven íródott, kihasználva az *incomplete Boolean evaluation* és az *operator overloading* előnyeit. A C++ nyelv standard 'double' típusát használtam az események időbélyegének, azaz kulcsának a tárolásához.

A szimulációt három különböző számítógépen futtattam, ezek a kísérletek végzése idején (1995-ben) korszerű, illetve tipikus konfigurációk voltak.

Az első egy DEC ALPHA workstation volt (150MHz, Evolution 4 processor, 512 KB cache, 64MB RAM, DEC 2000, model 300, DEC OSF/1 V3.0 operációs rendszer). A programokat gcc (v2.6) segítségével fordítottam le, teljes optimalizációval (-O3 flag).

A második egy Intel 80486 DX2 volt (66MHz, 256KB cache, 20MB RAM, DOS 6.2). A Borland C++ 3.1 fordítót használtam, és a fordítást a kód sebességére optimalizáltam. A "Compact" memóriamodellt használtam.

A harmadik egy Intel kompatibilis Texas 80486 DLC volt (40MHz, 128KB cache, 4MB RAM, matematikai ko-processzor nélkül, DOS 6.2). A fordító és az opciói megegyezek az előző esetben használtakkal, annak kivételével, hogy a lebegőpontos műveletek emulációja be volt kapcsolva, mert ez a számítógép (az előző kettővel ellentétben) nem rendelkezett hardver lebegőpontos támogatással.

Sajnos, a gépek óráinak felbontása rendre csak 1ms, 55ms, 55ms volt így a kísérleteket kis *n* értékek esetén többször is meg kellett ismételni egy-egy időmérési intervallumon belül, hogy elfogadható pontosságú eredményeket kapjak.

Az állandósult állapotbeli események *m* számát az egyes processzorok esetén úgy választottam meg, hogy elég nagy legyen a kielégítő pontosságú időméréshez, rendre: 10000, 10000 és 1000.

A véletlenszámok generálására a (Jain 1991) forrásban közölt véletlenszám-generátort alkalmaztam, hogy a szimulációban használt véletlenszámok minőségét biztosítsam.

AVL-fa	(Wirth 1976)
B-fa	(Wirth 1976)
kupac	(Gonnet 1984)
Skip-list[1]	(Pugh 1990)

17. táblázat Adatszerkezetek és algoritmusaik implementációjához felhasznált források

A egyes adatstruktúrák pontos algoritmusait a 17. táblázatban közölt forrásokból vettem. Mivel a 2-3-fa a B-fa speciális esete, a kódot a B-fa alapján írtam a megfelelő egyszerűsítésekkel. A bináris fa és a láncolt lista algoritmusai triviálisak, azokat önállóan implementáltam.

A kupacot általában úgy szokták implementálni, hogy az elemeit egy tömbben tárolják. Ez azonban lelassítja a műveleteit, mikor egy elemet mozgatni kell. Ezért csak az elemek (események) pointereit tároltam a tömbben.

A B-fa paramétere 20, azaz minimum 20, maximum 40 kulcs tárolható egy csomópontban. Itt szintén csak az események pointereit tároltam a csomópont tömbjében.

3.3. Az eredmények kiértékelése

3.3.1. Az eloszlástól való függés

Az eredmények mennyisége olyan nagy, hogy szükségesnek látszik valamilyen redukció. Vizsgáljuk meg először, hogy a FES implementációjára használt különböző adatszerkezetek

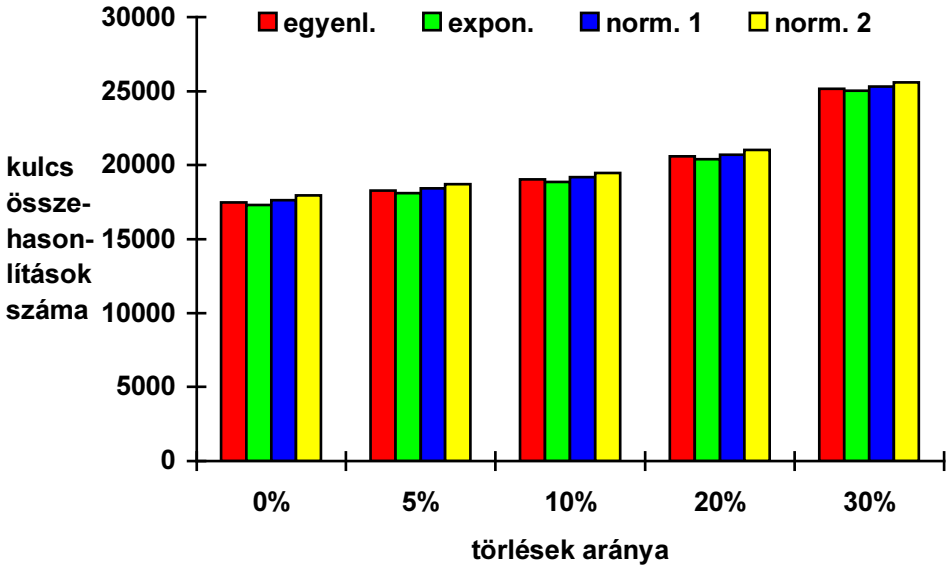
[1] A forráskódot anonymous FTP-vel töltöttem le az `ftp.cs.umd.edu/pub/skipLists` helyről

teljesítménye függ-e az alkalmazott eloszlástól. (Mert ha nem, akkor elegendő lenne egyetlen eloszlás esetét vizsgálni.)

A kulcsösszehasonlítások száma alkalmas erre a célra, mivel nem architektúrafüggő, így elegendő egyetlen processzoron végzett futtatások eredményeit vizsgálni.

3.3.1.1. A kiegyensúlyozott fák

Az AVL-fa, a B-fa és a 2-3-fa nagyon hasonlóan viselkedik: a használt kulcsösszehasonlítások száma gyakorlatilag független az alkalmazott eloszlástól, de növekszik a törlési arány függvényében. A 16. ábra a B-fára vonatkozó eredményeket mutatja, az ábra a másik kettő esetén is teljesen hasonló lenne.



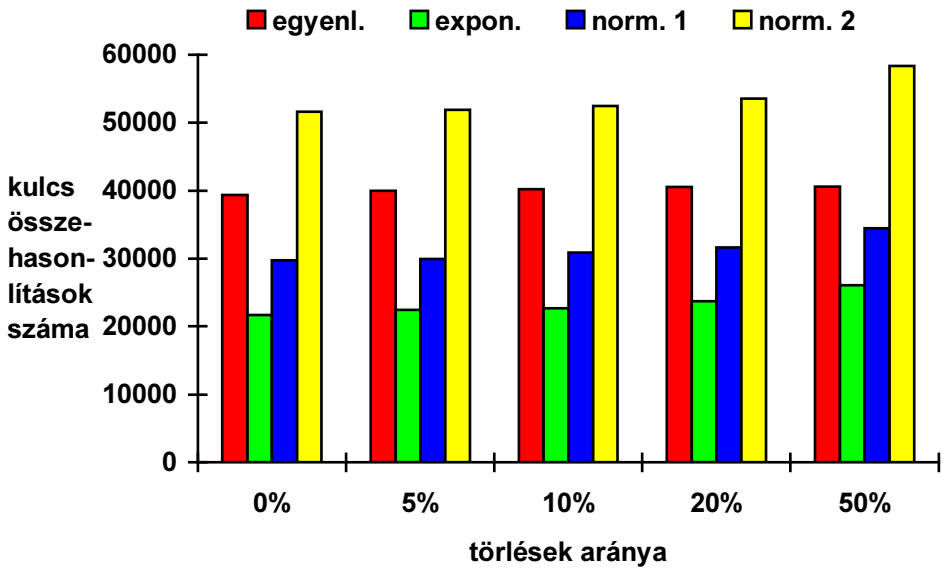
*16. ábra A kulcsösszehasonlítások számának az érkezési időköz eloszlásától és a törlések arányától való függése **B-fa** esetén (FES feltöltése 1000 eseményig)*

Az eloszlásfüggetlenség nem meglepő, mert mindhárom fa kiegyensúlyozott. A törlési arány hatása pedig azzal magyarázható, hogy az *első* művelet nem igényel kulcsösszehasonlítást, míg a *törlés* művelet igen.

3.3.1.2. A bináris fa

A bináris fa (17. ábra) eloszlásfüggőséget mutat. Ez az adatszerkezet az eloszlások közül ez exponenciálisnál adja a legjobb teljesítményt. Ha összehasonlítjuk a két normális eloszlásra kapott eredményt, akkor láthatjuk, hogy a kisebb szórású rosszabb eredményeket produkál a másiknál. Egy intuitív magyarázat, hogy az *első* művelet a bal oldaláról fogyasztja a bináris fát, a *beszúrás* pedig főleg a jobb oldaláról építi. Ez még inkább így van a kis szórású normális eloszlás esetén, ami közelít ahhoz, mintha determinisztikus konstans késleltetést alkalmaznánk. Az exponenciális eloszlás azonban nagyobb valószínűséggel produkál 0-hoz közeli késleltetés értékeket, ezáltal bal oldaláról is építve a fát.

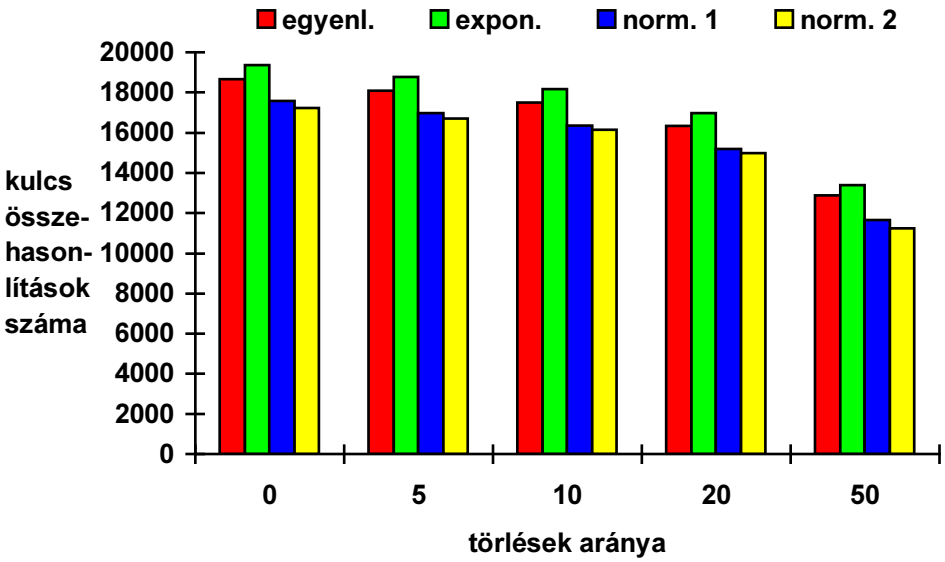
Egy másik fontos megfigyelés, hogy itt a törlési aránynak sokkal kisebb hatása van a kulcsösszehasonlítások számára, mint azt korábban láttuk.



*17. ábra A kulcsösszehasonlítások számának az érkezési időköz eloszlásától és a törlések arányától való függése **bináris fa** esetén (FES feltöltése 1000 eseményig)*

3.3.1.3. A kupac

A kupac viselkedését a 18. ábra mutatja. Ennek van némi eloszlásfüggése, de olyan kicsi, hogy elhanyagolhatjuk. A többiekkel ellentétben a kupac kevesebb kulcsösszehasonlítást kíván, ha a törlési arány növekszik.

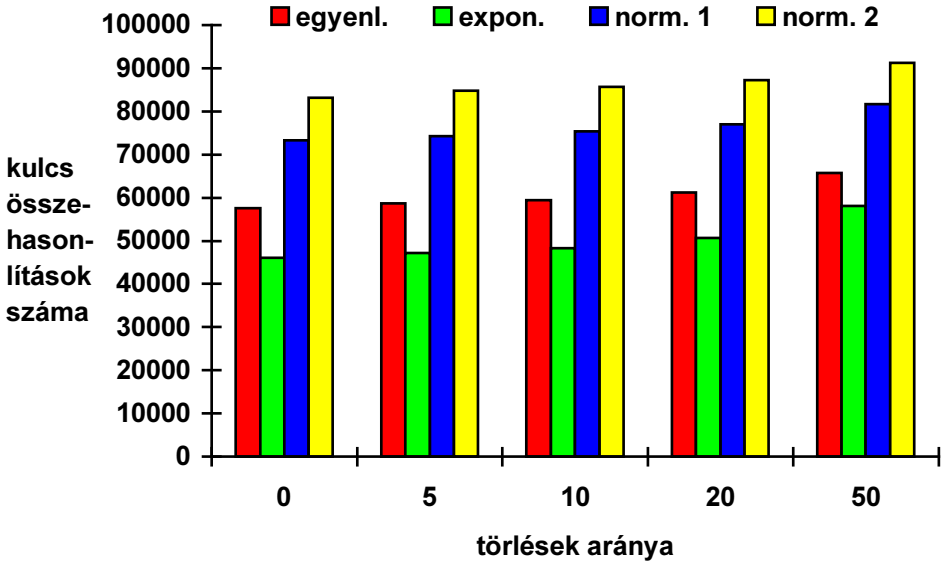


*18. ábra A kulcsösszehasonlítások számának az érkezési időköz eloszlásától és a törlések arányától való függése **kupac** esetén (FES feltöltése 1000 eseményig)*

3.3.1.4. A rendezett, egyirányban láncolt lista

A lista szimulációja olyan lassú volt, hogy nem tudtam n=1000-re futtatni, a 19. ábra n=215 melletti eredményeket mutat. (Ezeket az eredményeket tehát nem szabad a többi adat-

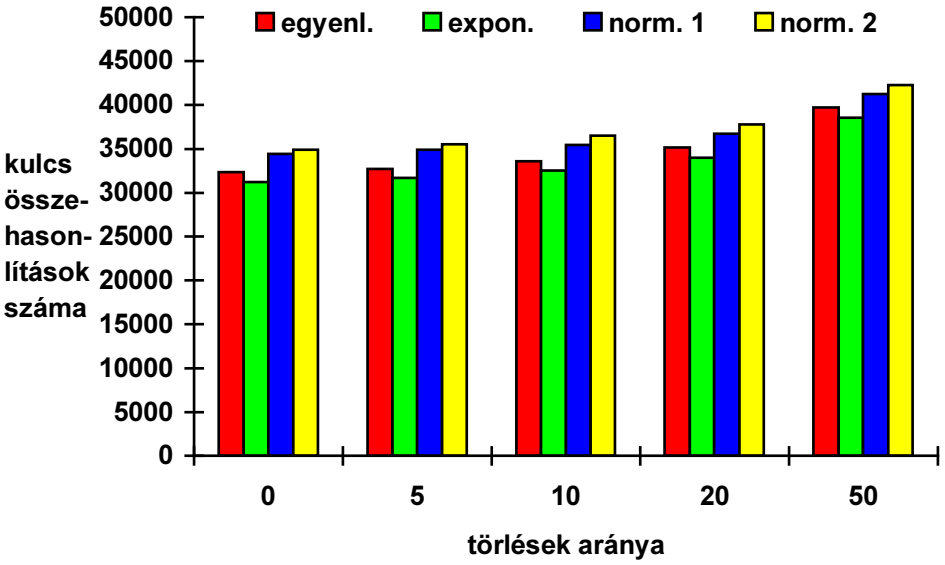
szerkezet n=1000 esetén kapott eredményeivel közvetlenül összehasonlítani!) Eloszlásfüggése hasonló a bináris fa eloszlásfüggéséhez. Hasonlóan, a kulcsösszehasonlítások száma is gyenge növekedést mutat a törlési arány függvényében.



*19. ábra A kulcsösszehasonlítások számának az érkezési időköz eloszlásától és a törlések arányától való függése **rendezett, egyirányban láncolt lista** esetén (FES feltöltése **csak 215** eseményig)*

3.3.1.5. A skip-list

Végül nézzük a skip-list jellemzőit: 20. ábra. Az eloszlásfüggés elhanyagolható, de a kulcsösszehasonlítások száma növekszik a törlések számának függvényében.



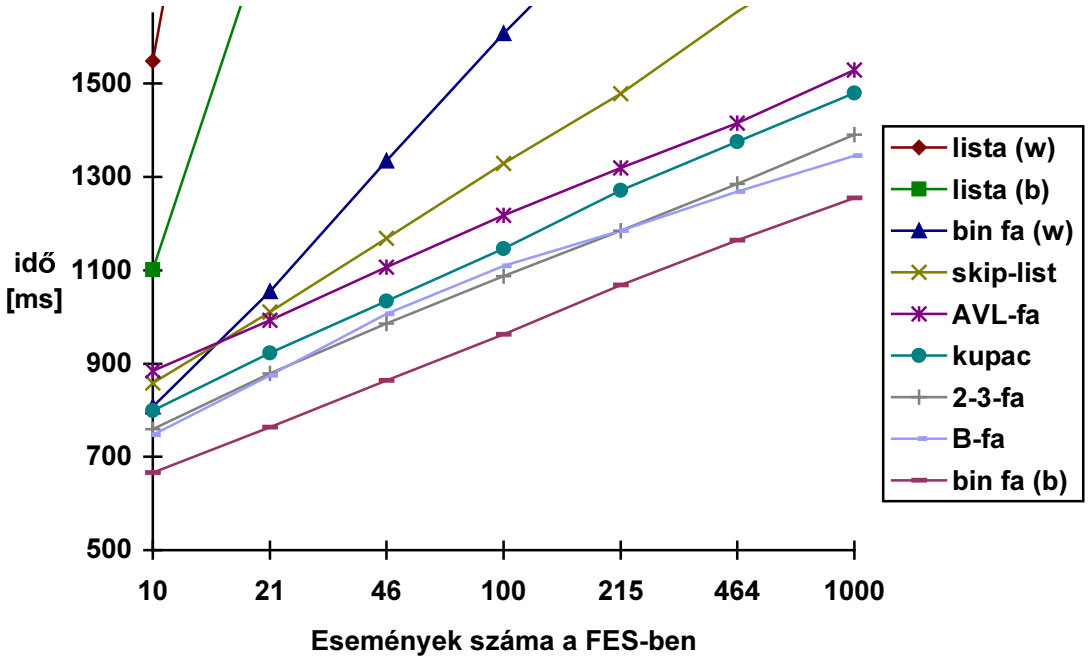
*20. ábra A kulcsösszehasonlítások számának az érkezési időköz eloszlásától és a törlések arányától való függése **skip-list** esetén (FES feltöltése 1000 eseményig)*

3.3.2. Az adatszerkezetek teljesítményének összehasonlítása

Amint fent tárgyaltuk, mind az eloszlásfüggés, mind a törlési arány befolyásolhatja a vizsgált adatszerkezetek teljesítményét. Ez nagyon megnehezíthetné az összehasonlításukat, ha nem találnánk elfogadható kompromisszumot jelentő közös nevezőt. Használjunk 10% törlési arányt – ez egy realisztikus (valószínűleg felső) becslés. Az eloszlásfüggés problémájának a megoldására pedig az erősen eloszlásfüggő esetekben (bináris fa, láncolt lista) használjunk két grafikont. Az egyik a legkedvezőbb esetben mért teljesítményt mutatja (ezt "b"-vel jelöljük – *best case*), a másik pedig a legkedvezőtlenebb eset teljesítményét ("w" – *worst case*). A többi 5 adatszerkezet esetén egyenletes eloszlást használunk.

3.3.2.1. Hardver lebegőpontos támogatás nélkül

Hasonlítsuk össze az adatszerkezetek időjellemzőit! Először azt az esetet vizsgáljuk meg, amikor a processzorban nincs hardver támogatás a lebegőpontos műveletekhez: 21. ábra. (Figyelem: a vízszintes tengely logaritmikusan skálázott!)



21. ábra Az adatszerkezetek időjellemzőinek összehasonlítása i486 DLC processzoron (állandósult állapot, 1000 szimulációs lépés, törlési arány 10%)

A kiegyensúlyozott fák (2-3-fa, B-fa és AVL-fa) jó teljesítményt nyújtottak, mert hatékonyan minimalizálták a kulcsösszehasonlítások számát, ami ezen az architektúrán dominálja a végrehajtási időt. A kupac is jó teljesítményt nyújtott, mert az is viszonylag kevés kulcs-összehasonlítást igényel, amint korábban láttuk.

A bináris fa volt a leggyorsabb, mikor a "*best case*" teljesítményét nyújtotta. Azonban, mivel egyáltalán nem garantálható, hogy a szimulációk nagy többségében az eloszlás exponenciális lesz, a bináris fa rosszabb teljesítmény jellemzőit (is) figyelembe kell vennünk.

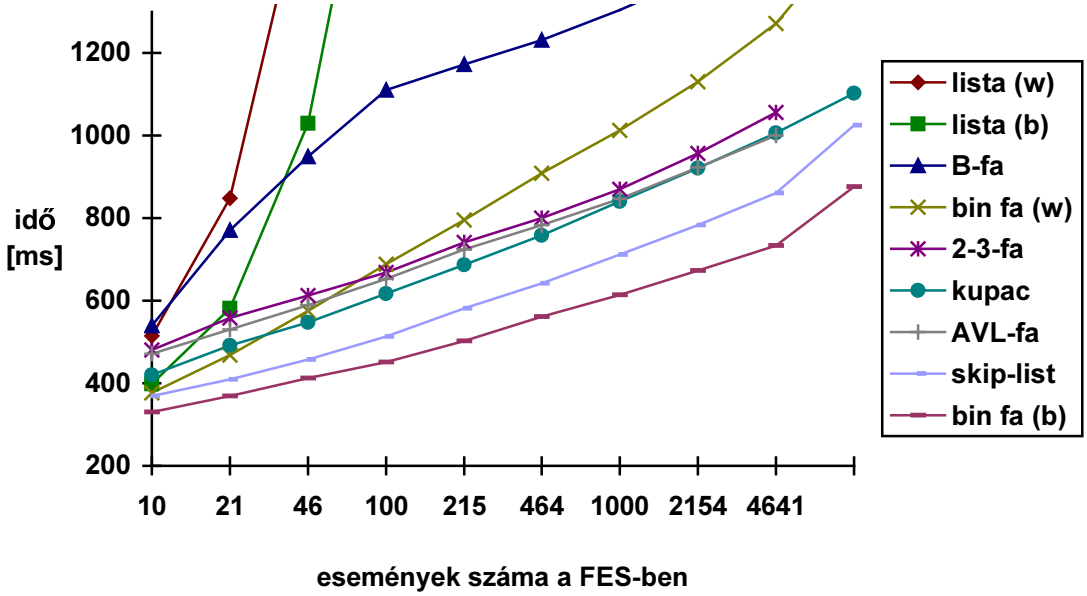
A skip-list ebben az esetben határozottan rosszabb volt a kiegyensúlyozott fáknál.

A rendezett, egyirányban láncolt lista természetesen a leggyengébb teljesítményt nyújtotta.

3.3.2.2. Közepes hardver lebegőpontos támogatással

Most vizsgáljuk meg a másik Intel processzoron kapott eredményeket! Ebben a processzorban beépített matematikai ko-processzor van.

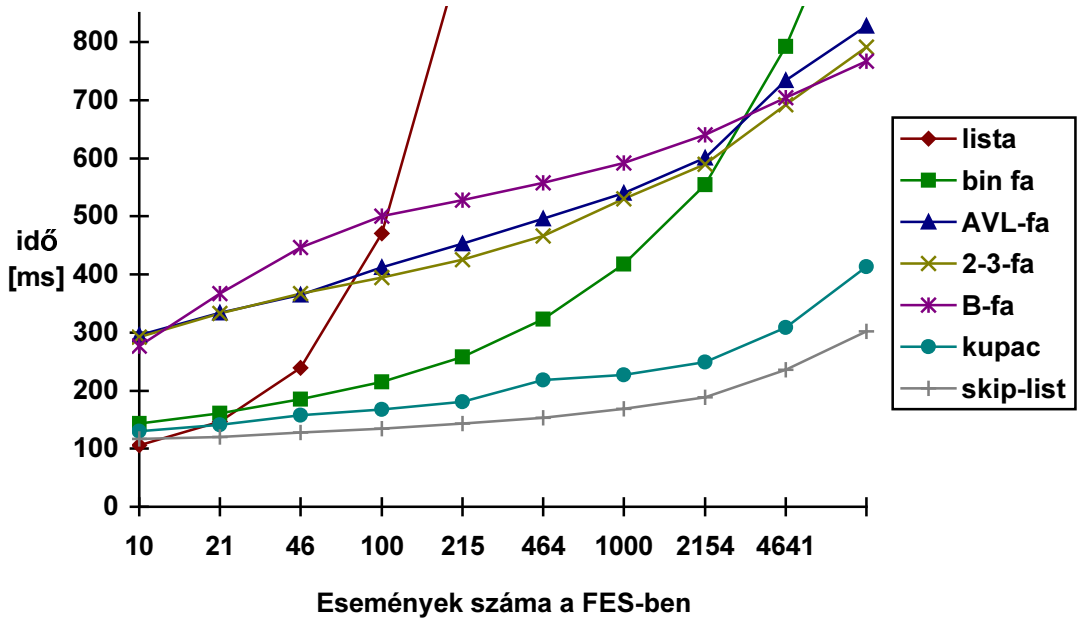
Amint a 22. ábrán látható, az AVL-fa, a 2-3-fa és a kupac megtartották a jó helyezésüket, a B-fa lényegesen rosszabb. A bináris fáról és a láncolt listáról ugyanaz mondható el, mint az előző processzor esetén. Természetesen a láncolt lista csak egy bizonyos *n* FES-beli eseményszám fölött a legrosszabb. Ezen a platformon a skip-list megelőzte a kiegyensúlyozott fákat és a kupacot.



22. ábra Az adatszerkezetek időjellemzőinek összehasonlítása i486 DX processzoron (állandósult állapot, 10000 szimulációs lépés, törlési arány 10%)

3.3.2.3. Erős hardver lebegőpontos támogatással

Végül nézzük az ALPHA processzoron kapott futási eredményeket: 23. ábra. Ez a processzor rendelkezik a 3 közül a lebegőpontos műveletek legjobb hardver támogatásával. Itt a törlési arány 0% és az új események késleltetése minden adatszerkezet esetén egyenletes eloszlást követ. Az ábrán a többitől élesen elkülönül a két legjobb választást jelentő adatszerkezet: a kupac és különösen a skip-list.



23. ábra Az adatszerkezetek időjellemzőinek összehasonlítása DEC ALPHA processzoron (állandósult állapot, 10000 szimulációs lépés, törlési arány 0%)

3.4. A FES implementációjával kapcsolatos megállapításaim

Hét adatszerkezetet (rendezett, egyirányban láncolt lista, bináris fa, AVL-fa, B-fa, 2-3-fa, kupac, skip-list) vizsgáltam meg szimuláció segítségével, hogy melyiket célszerű diszkrét idejű szimulátorokban a jövőbeli események halmazának (FES) a tárolására alkalmazni. A problémát az események számának függvényében aszimptotikusan vizsgáltam, számos paraméter összes lehetséges kombinációja mellett.

A kapott eredmények alapján arra a következtetésre jutottam, hogy mind a szimulációt végrehajtó processzor architektúrája, mind a szimulációban egymást követő események időeloszlása jelentősen befolyásolhatja, hogy melyik adatszerkezetnek a legjobbak a végrehajtási idő jellemzői.

Megvizsgálva a kulcsösszehasonlítások számát, azt találtam, hogy a kiegyensúlyozott fák és a kupac adta a legkisebb értékeket. Ezért azokban az esetekben, ahol az adatszerkezet műveleteinek processzoridejét a kulcsösszehasonlítások dominálják (ahol nincsen hardver lebegőpontos művelet támogatás), a kiegyensúlyozott fák és a kupac jelentik a legjobb választást. A közönséges (kiegyensúlyozatlan) bináris fa jó eredményeket mutatott exponenciális eloszlás esetén, de más eloszlásokra eredményei rosszabbak a kiegyensúlyozott fák és a kupac eredményeinél.

Erős hardver lebegőpontos támogatás esetén a skip-list és a kupac is jobb időjellemzőket adott, mint a kiegyensúlyozott fák. Ebben az esetben tehát a skip-list vagy a kupac alkalmazását javaslom, ezek algoritmusai egyszerűbbek is mint a kiegyensúlyozott fákéi.

4. A traffic-flow analysis nevű új módszer

A (Lencse, 2000) cikkemben a traffic-flow analysis (TFA) nevű kommunikációs hálózatok teljesítőképesség-vizsgálatára alkalmazható, általam kidolgozott módszert ismertettem. Ez a megoldás a szimulációnak és numerikus módszereknek a kombinációja, a hálózat részletes szimulációjához képest lényegesen kisebb számításigénnyel – így sokkal gyorsabban – közelítő eredményeket ad. Működése során a forgalmat megfelelően választott méretű egységekben a hálózat tényleges routolási algoritmusának felhasználásával osztja szét a rendszerben. A hálózat véges kapacitásainak a forgalom időeloszlására gyakorolt hatását a forgalom térbeli elosztása után veszi figyelembe.

4.1. Bevezetés

Kommunikációs hálózatok teljesítőképességének vizsgálatára gyakran használunk diszkrét idejű szimulációt. Ennek során mintegy lejátsszuk a hálózat működését, azaz forgalomforrások a modellezett hálózatra jellemző idő- és hosszeloszlással csomagokat generálnak, amelyek a hálózat vonalain és csomópontjain keresztül a hálózatra jellemző forgalomirányítási (routing) algoritmus szerint jutnak el a címzetthez. A rendszer számunkra érdekes jellemzőit a szimuláció során gyűjtött statisztikákból határozzuk meg. Az eredmények pontossága függ a forgalommodellnek, a rendszer működését leíró modellnek és az alkalmazott statisztikagyűjtési módszernek a pontosságától. Kellő statisztikai pontosság eléréséhez megfelelően hosszú virtuális (azaz modellbeli) időintervallumban kell a szimulációt végrehajtani. A kívánt pontosság meghatározza a szimuláció műveletigényét, így végrehajtási idejét is. Nemcsak az eredmények pontosságának a fokozása lehet a cél, hanem a végrehajtási idő drasztikus csökkentése is, elfogadható mértékű pontosság mellett. Ez a helyzet például egy hálózat tervezésének korai szakaszában, amikor elképzeléseink, ötleteink helyességét, vagy a változtatások hatását szeretnénk nagyon gyorsan megvizsgálni nagy számú változat esetére. Ebben nyújt segítséget a **traffic-flow analysis** (TFA), ami a szimuláció és az analitikus és/vagy numerikus módszerek kombinációja. A részletes szimulációval szemben a forgalmat nem csomagonként, hanem nagyobb, alkalmasan választott méretű egységekben osztja szét a rendszerben, a forgalom időeloszlását pedig a térbeli eloszlás meghatározásától bizonyos értelemben elkülönítve kezeli. Az eljárás számításigénye – így végrehajtási ideje is – a részletes szimulációhoz viszonyítva sokkal kisebb lehet.

A továbbiakban ismertetem a módszer működését, bemutatom a választott forgalommodell statisztikáinak összeadását, részletesen kitérek egyes kritikus részekre, majd bemutatok egy konkrét TFA implementációt. Végül ismertetek egy traffic-flow analysis – analitikus vizsgálat összehasonlítást.

4.2. A módszer általános leírása

4.2.1. A traffic-flow analysis hálózati modellje

A traffic-flow analysis (a továbbiakban TFA) a hálózat modelljéről feltételezi, hogy az (valamilyen protokoll szerint működő) **csomópontok**ból (routerek, switchek, stb.) és azokat összekötő **vonalak**ból áll. A hálózat forgalmát a csomópontokhoz kapcsolódó **alkalmazásmodellek** generálják. Az alkalmazásmodellek forgalmának mértékét és időeloszlását matematikai modellel, (tipikusan statisztikákkal) írjuk le.

A TFA egy általános módszer, amely többféle forgalommodellel illetve hálózati útvonalválasztási algoritmussal is működhet. A forgalommodellre vonatkozó követelményeket a TFA alapvető működésének ismertetése után a 4.2.3. alfejezetben adom meg.

4.2.2. A módszer alapvető működése

Egy program, ami egy hálózati modellen TFA-t végez, két fázisban működik.

4.2.2.1. Első fázis: a forgalom térbeli eloszlásának meghatározása.

Az alkalmazásmodellek az általuk generált forgalmat paraméterezhető méretű **routolási egységek**ben küldik el a megfelelő cél címzettnek. A routolási egységek a forgalom alapegységei, amelyeket az algoritmus egyben kezel. A csomópontok a routolási egységeket a modellezett hálózat útvonalválasztási algoritmusának megfelelően továbbítják, miközben mind a vonalak, mind a csomópontok rögzítik a rajtuk áthaladó forgalmat. A csomópontok a routoláshoz természetesen felhasználhatják az egyes vonalak kapacitás adatait, de a forgalom időeloszlását a forgalom térbeli szétosztásának fázisában a vonalak és a csomópontok kapacitásai nem befolyásolják. A vizsgált hálózat teljes modellezett forgalmának szétosztása után minden vonalon és csomópontban rendelkezésre áll a rajta áthaladt forgalom naplója.

4.2.2.2. Második fázis: a forgalom időeloszlásának meghatározása.

A második fázisban a vonalakon és csomópontokon áthaladó ***routolási egységeket össze kell adni***. Az összeadás a forgalommodell összeadási műveletével történik. Az összes összeadandó összeadása után, a legvégén kell ***figyelembe venni a véges kapacitás hatását***. Ezeket majd a választott forgalommodellnél a 4.3. fejezetben részletesen tárgyalom.

4.2.3. A forgalommodellre vonatkozó követelmények

A forgalommodellnek meg kell felelnie a következő követelményeknek

- a forgalom mindig adott csomóponttól adott csomópont felé irányul (azaz a TFA nem kezel üzenetszórást vagy többesküldést)
- A forgalom mértéke és időeloszlása matematikai modellel leírható. A továbbiakban ezt nevezzük **forgalommodell**nek.

- A forgalommodell az összeadás műveletére nézve zárt, azaz az összeadás eredménye ugyan olyan típusú lesz, mint az összeadandók.
- A csomópontok és vonalak véges kapacitásának a forgalom időeloszlására való hatása kiszámítható. (Nem követelmény a zártság.)
- Forrás- és cél csomópont páronként az azonos típusú alkalmazások azonos cél csomópont felé irányuló forgalma együttesen, de akár tetszőleges méretű egységekben is kezelhető. (aggregált forgalommodell)
- Az egyes forrás- és cél csomópont párok forgalma tetszőleges sorrendben kezelhető

4.2.4. Mire képes a traffic-flow analysis?

A módszer segítségével a hálózat csomópontjain és vonalain olyan klasszikus, a szimulációban általánosan használt statisztikai jellemzőket határozhatunk meg, mint például a kihasználtság, késleltetés, stb.

A TFA ***pillanatképet ad*** a hálózat forgalmi viszonyairól. Ez azt jelenti, hogy a TFA eredményeként kapott forgalmi jellemzők abban az adott virtuális időpontban állnak fenn, amelynek megfelelőek az alkalmazásmodell paramétereinek értékei.

Az alkalmazásmodellek együttesen kezelik paraméterként megadott számú azonos típusba tartozó alkalmazás forgalmát, amelyek egyenként is nagy számú küldött/fogadott csomagot tartalmazhatnak. Mivel a TFA eleve statisztikákat visz át a hálózaton, nem kell azokat nagy számú megfigyelés (áthaladó csomag) alapján gyűjtenie. Ezáltal a TFA ***végrehajtási ideje*** az adott rendszer hagyományos, részletes szimulációval történő vizsgálatához szükséges időnek a ***töredéke lehet***. Természetesen a routolási egység méretének és az alkalmazott statisztikák jellemzőinek megválasztása a sebesség és a pontosság közötti kompromisszumot jelenti.

A TFA a vonalkapacitásnak a forgalom időeloszlására gyakorolt hatását csak lokálisan, a forgalom térbeli szétosztása után veszi figyelembe. Mi ennek a következménye? Tekintsünk egy vég-vég útvonalat, amelyben van egy kis kapacitású vonal. A valóságos rendszerben a vonalnak a forgalom időeloszlását megváltoztató hatása az út további részén is megjelenik, a TFA esetén pedig nem. Emiatt a TFA nem is ad pontos képet a hálózat forgalmi viszonyairól. Alkalmas viszont annak a kérdésnek a megválaszolására, hogy ***van-e a hálózatban szűk keresztmetszet*** (kritikus terhelés, késleltetés, pufferméret, stb) ***és ez hol van***. Ez olyan fontos kérdés, hogy hatékony megválaszolása indokolja a TFA létjogosultságát.

4.3. A választott forgalommodell

A forgalom mértékének és időeloszlásának leírására elvileg bármi használható, ami a fenti feltételeknek megfelel, én a **throughput eloszlás**t és a **késleltetés eloszlás**t használom. Ezek értelmezése a következő: Ha a throughput eloszlás sűrűségfüggvényének értéke egy x helyen $p(x)$, ez azt jelenti, hogy $p(x)$ annak a valószínűsége, hogy a throughput értéke x. Hasonlóan a késleltetés eloszlás sűrűségfüggvénye azt mutatja meg, hogy mi az adott késleltetés kialakulásának a valószínűsége.

A forgalommodellnek alkalmasnak kell lennie arra, hogy a forgalmat mind a vonalakon, mind a csomópontokban megfelelően jellemezze. Erre alkalmazhatjuk például a bit-throughput és a packet-throughput eloszlásokat. Az alkalmazásmodellek csak throughput eloszlásokat generálnak, a késleltetés eloszlások majd a hálózat elemein történő összeadást követő kapacitás korlátozás során keletkeznek.

4.3.1. A throughput származtatása

A throughput alatt az időegység alatt átvitt bitek vagy csomagok számát értem. Mérésére alkalmasan választott T időintervallumban megszámolom az átvitt biteket/csomagokat, és a kísérletet kellő számúszor ismételve az eredményből statisztikát (hisztogramot) készítek. A T paraméter megválasztása drasztikusan befolyásolja a kapott throughput eloszlási statisztikát. Ezzel kapcsolatos megfontolásaimat a vonalak bit-throughputjának a példáján végzem.

Egy vonalnak csak két állapota lehet: vagy éppen van rajta forgalom, vagy nincsen (a különböző multiplexelt csatornákat külön vonalaknak tekintjük). Ha tehát $T \to 0$ akkor a vonal bit-throughput eloszlása két impulzusból fog állni, az egyik 0-nál a másik éppen a vonalkapacitásnál helyezkedik el. Magasságuk pedig megadja annak a valószínűségét, hogy a vonal éppen üres, illetve, hogy éppen forgalom van rajta. Ha pedig $T \to \infty$, akkor egyetlen impulzusból fog állni a vonal bit-throughput eloszlása, és ez a forgalom nagyságának a várható értékénél fog elhelyezkedni. A TFA számára mindkét eset használhatatlan eredményt ad.

Olyan T-t kell választani, amelynek értéke mellett a kapott bit-throughput eloszlás a legjobban jellemzi a generáló alkalmazást, (azaz a legtöbb információt hordozza róla). Ezt objektíven mérni persze nehéz. A T paraméter értékének meghatározásához abból kell kiindulnunk, hogy az analízissel milyen kérdésre szeretnénk választ kapni. Egy kulcskérdés lehet: milyen felbontással érdekel bennünket a vonalon kialakuló késleltetéseloszlás? A válasz a TFA céljától függ. Lássunk erre két példát:

- Ha például a vonalhoz tartozó véges kapacitású puffer méretének elégséges voltát szeretnénk ellenőrizni, akkor fontos a puffer feltöltési ideje:

  $$T_f = \frac{\textit{pufferméret}}{\textit{vonalkapacitás}}$$

  Ekkor T-t úgy választjuk meg, hogy az a T_f néhányadrésze legyen, például $T=T_f/10$.

- Ha pedig a vég-vég késleltetés értékét vizsgáljuk, hogy az alatta van-e az alkalmazás által tolerált szintnek, akkor T-t úgy választjuk meg, hogy az adott vonalra még megengedett T_d késleltetés néhányadrésze legyen.

4.3.2. Összeadás

A hálózat adott pontján a throughput eloszlásokat össze kell adni. Először a vonal vagy kapcsológép kapacitásának figyelembevétele nélkül összeadjuk az összes throughput eloszlást, végül elvégezzük a kapacitást figyelembe vevő korrekciót.

Vizsgáljuk meg most, hogyan kell összeadni két throughput eloszlást!

Jelölje *p(x)* és *q(x)* a két összeadandó throughput eloszlás sűrűségfüggvényét. Az eredményben az adott *y* értékhez tartozó *r(y)* valószínűsűget gyarapítja mindaz az eset, amikor az egyik forrásból jövő throughput értéke *x*, a másikból jövőé pedig *(y-x)*, a forrásokat függetleneknek feltételezve ezen throughput értékek együttes előfordulásának valószínűsége a két valószínűség szorzata, azaz:

$$r(y) = \int_{x=0}^{y} p(x)q(y-x)dx$$

Ami azt jelenti, hogy a throughput eloszlások sűrűségfüggvényei konvolúcióval adhatók össze.

A throughput eloszlás elméletileg abszolút folytonos sűrűségfüggvényét a gyakorlatban egy hisztogrammal közelítjük, melynek egy cellája azt mutatja meg, hogy mi annak a valószínűsége, hogy a throughput a cella alsó és felső határa közé essen. A konvolúció ekkor:

$$r[l] = \sum_{k=0}^{l} p[k]q[l-k]$$

A forgalommodellhez használt hisztogramok gyakran valóságos rendszeren végzett méréseken alapulnak. Az adaptív on-line statisztika gyűjtő módszerek, mint a P^2 (Jain and Clamtac, 1985) és a k-split (Varga and Fakhamzadeh, 1997) különösen hasznosak erre a célra, bár természetesen bizonyos határokig lehetőség van a mért adatok tárolására és utólagos off-line feldolgozására. A TFA céljára a k-split jobb, mint a P^2, mert a k-split módszerrel gyűjtött statisztika információvesztés nélkül egyenlő osztásközű hisztogrammá transzformálható. (A k-split esetén a cellaméretek egy legkisebb cellaméret többszörösei; a P^2 nem rendelkezik ezzel a tulajdonsággal.) Más fontos statisztikagyűjtési módszerekkel már a 2.4. fejezetben foglalkoztam.

4.3.3. Véges sávszélesség miatti korrekció

A vonal- illetve csomópont kapacitást túllépő forgalom késleltetést szenved, kiszolgálásuk pedig a később beérkező forgalomhoz hozzáadódó terhelést jelent a rendszer számára (puffereit esetet feltételezve). A problémát kapcsológép és packet-throughput eloszlás, mint példa esetén vizsgálom meg, de ügyelek arra, hogy eredményeim a vonalkapacitásra is alkalmazhatóak legyenek.

Annak a valószínűsége, hogy ***T* idő alatt éppen *k* csomag érkezik** a kapcsológéphez $p_T[k]$. Indítsuk a rendszert üres várakozási sorral, így az első *T* hosszúságú időszeletben nincsenek várakozó csomagok. Az egyszerűség kedvéért fogadjuk el azt, hogy mivel a kapcsológép *T* idő alatt *K* darab csomag kapcsolására képes, ha $k>K$ darab csomag érkezik, akkor abból *K* darab tud átmenni, (*k*-*K*) darab várakozik és hozzáadódik a következő *T* intervallumban érkezőkhöz, ha pedig $k \leq K$ darab érkezik, akkor mind a *k* darab átmegy. (A valóságban persze ez nem szükségszerűen teljesül, hanem függ a csomagok érkezésének a *T* intervallumon belüli eloszlásától, de most az egyszerű kezelhetőség kedvéért ettől a problémától eltekintünk.) Annak a valószínűsége, hogy nem marad csomag a következő időrésre:

$$p_0 = \sum_{k=0}^{K} p_T[k]$$

Annak a valószínűsége pedig, hogy marad valahány várakozó csomag: p_1=1-p_0. Erre az esetre nézve határozzuk meg, hogy hány csomag igényel kiszolgálást a második időrésben. Ekkor az eloszlásnak a (*K+1*) darab csomag érkezésének valószínűségét leíró értéktől kezdődő farkát (tail) hozzá kell adni a csomagok következő időrésbeli beérkezését leíró sűrűségfüggvényhez. Az összeadást konvolúcióval kell végezni:

$$q_T[k] = \sum_{l=0}^{k} p_T[l+K]p_T[k-l]$$

Az összeadás futó változója, *l* azt fejezi ki, hogy hány darab várakozó csomag adódik a későbbi időrésben érkező csomagokhoz.

Az új sűrűségfüggvény ($p_T^2[k]$, ami a második időrésbeli helyzetet írja le) tehát az üres sorba való érkezést leíró $p_T[k]$ és sorbanálló csomagoknak az újonnan érkezettekhez hozzáadódását leíró (konvolúció eredményeként kapott) $q_T[k]$ súlyozott összegeként adódik. Az összegzés itt egyszerű cellánkénti összeadás, azaz:

$$p_T^{\,2}[k] = p_0 p_T[k] + q_T[k]$$

Megjegyzés: $q_T[k]$ már tartalmazza a p_1 súlyt, mert a farokrész sűrűségfüggvénye nem volt normált.

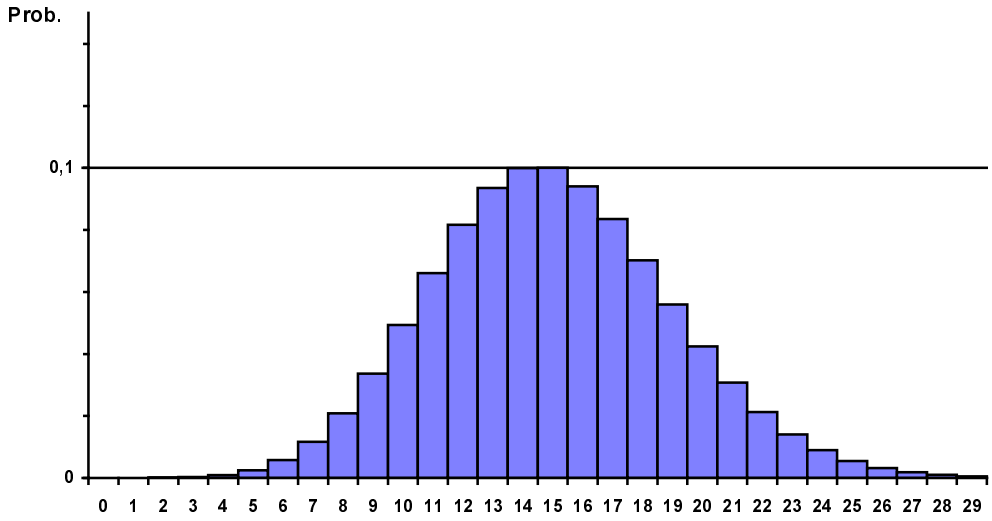
Az így kapott eloszlásra megint kiszámolhatjuk, hogy mekkora a valószínűsége, hogy nem marad, illetve, hogy marad csomag a következő időrésre. Ezután az előzőhöz teljesen hasonlóan nyerjük $p_T^3[k]$ -t, majd az eljárást ismételve $p_T^4[k]$ -t, … stb. Végül kialakul a rendszer állandósult állapotát (steady state) leíró sűrűségfüggvény: $\boldsymbol{p_T^*[k]}$, ami megadja, hogy mi a valószínűsége annak, hogy **állandósult állapotban éppen *k* darab csomag igényel kiszolgálást *T* idő alatt**. (Az *igényel* szó hivatott elfedni azt a különbséget, hogy újonnan érkezett-e a csomag, vagy már korábban jött és várakozott.) Jelölje $\boldsymbol{P_T[k]}$ azt a sűrűségfüggvényt, ami **megadja, hogy mi a valószínűsége annak, hogy *T* idő alatt éppen *k* darab csomag kerül kiszolgálásra**. Ez a kiszolgálás igényt leíró eloszlás sűrűségfüggvényéből ($p_T^*[k]$) számítható: a (*K+1*)-től kezdődő farka (közönséges matematikai + művelettel) hozzáadódik a *K*. cellához:

$$P_T[k] = \begin{cases} p_T^{\,*}[k], & k < K \\ \sum_{l=K}^{\infty} p_T^{\,*}[l], & k = K \\ 0, & k > K \end{cases}$$

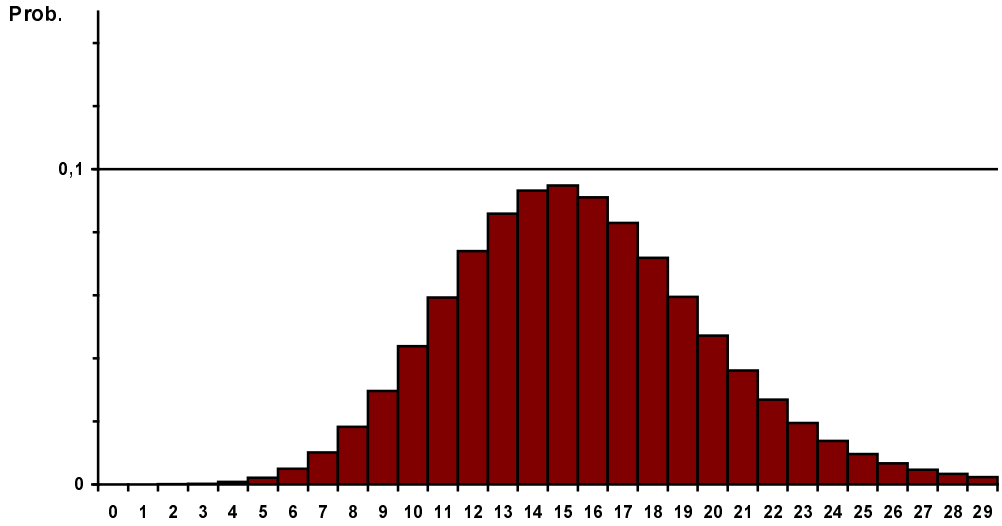
A $p_T^*[k]$ sűrűségfüggvény (*K+1*). cellájától kezdődő farok része által leírt forgalom tehát késleltetést szenved. Vizsgáljuk meg, milyen lesz a kapacitáskorlátozás okozta késleltetés eloszlás. Jelölje $\boldsymbol{D_T[i]}$ **annak a valószínűségét, hogy a csomagok *iT* időrésnyi késleltetést szenvednek**.

$$D_T[i] = \begin{cases} \sum_{k=0}^{K} p_T^{*}[k], & i = 0 \\ \sum_{k=iK+1}^{(i+1)K} p_T^{*}[k], & i \geq 1 \end{cases}$$

Most lássunk egy példát! A paraméterek értékeit úgy választottam meg, hogy a kapacitáskorlátozás hatása jól látható legyen.



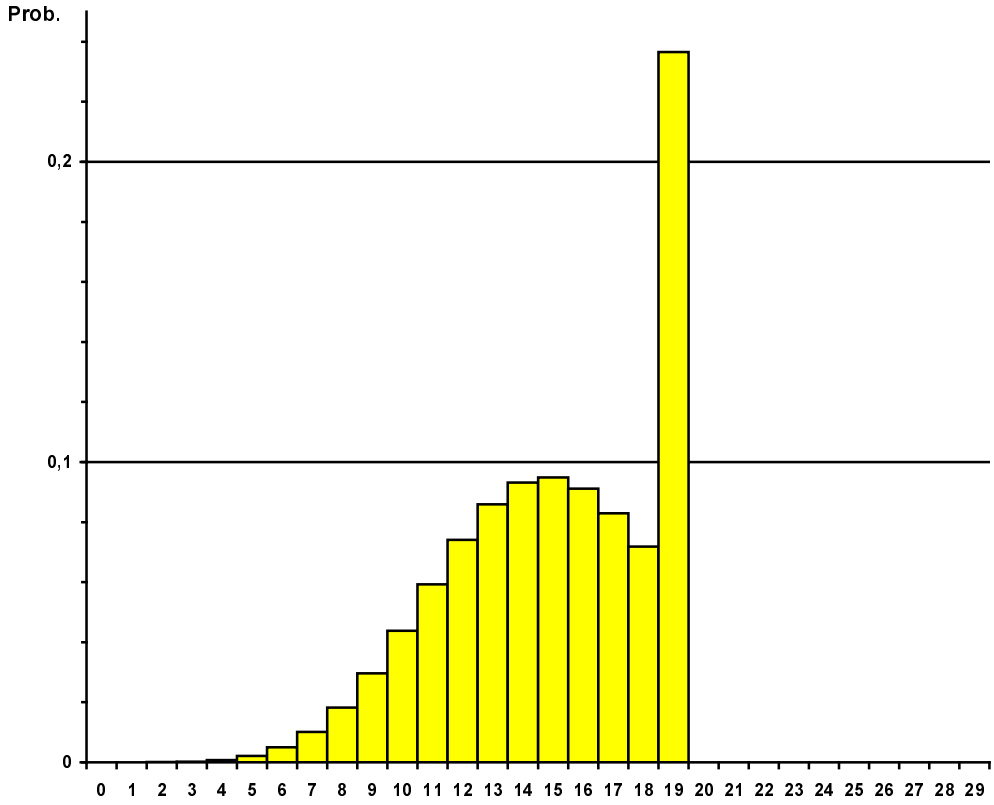
24. ábra A $p_T[k]$ eredeti sűrűségfüggvény



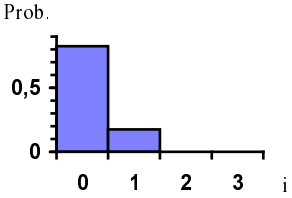
25. ábra A konvolúciók végeredményeként kapott $p_T^*[k]$

A 24. ábrán látható az eredeti eloszlás. A 25. ábra mutatja a konvolúciók végeredményét. Megfigyelhetjük, hogy az ismételt konvolúciók hatására az eloszlás súlypontja jobbra tolódott. A 26. ábrán találjuk a kapacitáskorlátozott eloszlást. Az eloszlást 20 cellára korlátoztam, ezek 0-tól 19-ig sorszámozottak. Az esetek több, mint 23%-ában a kapacitás kihasználtsága teljes. A 19-es számú cella eredeti értéke a 25. ábrán 0.059 volt, ehhez jött hozzá a levá-

gott részből 0.177. A 27. ábrán látható a késleltetés eloszlás alakulása. Az 1*T késleltetés valószínűsége körülbelül 17.7%.



26. ábra A kapacitás korlátozás eredményeként kapott $P_T[k]$



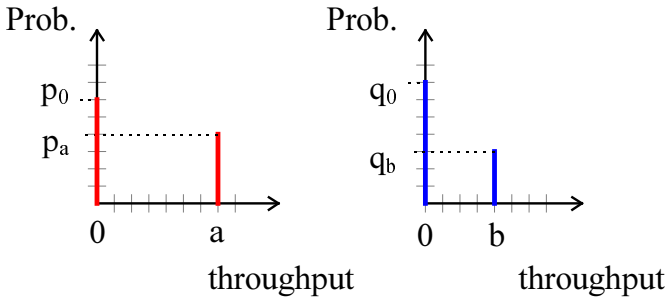
27. ábra A $D_T[i]$ *késleltetés eloszlás alakulása*

A fenti eljárásban az egyszerűbb megfogalmazás érdekében kihasználtam azt, hogy kapcsológépbeli packet-throughput eloszlásról volt szó, ahol a hisztogram cellák méretének szélessége 1 volt. A bit-throughput eloszlásnál és néha gyakorlati okok miatt a packet-throughput eloszlásnál is szélesebb cellákat használunk. Ekkor a véges cellaszélesség hatását figyelembe vevő módosított konvolúciót kell alkalmazni, amit a 4.4.1. alfejezetben fogok bemutatni.

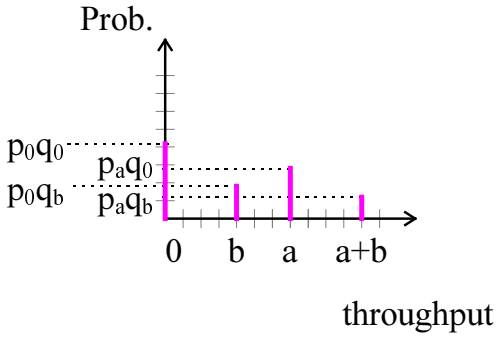
4.4. Fontos részletkérdések megvitatása

4.4.1. A véges cellaszélesség hatása

Ha a hisztogram celláinak szélessége elhanyagolható, akkor a konvolúció triviális. (28. és 29. ábrák)

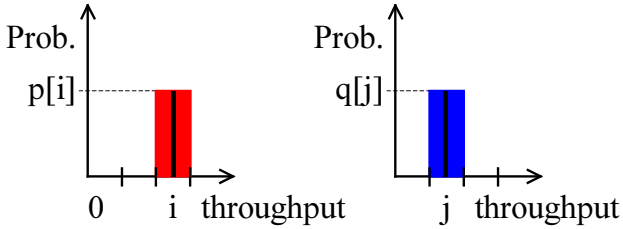


28. ábra Összeadandó sűrűségfüggvények – ideális eset

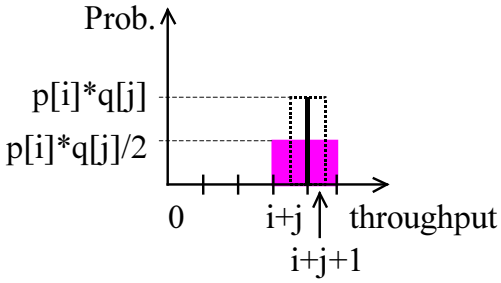


29. ábra A konvolúció eredménye – ideális eset

Ha a hisztogram celláinak szélessége nem elhanyagolható, akkor a 30. és 31. ábrákon látható jelenség következik be: a konvolúció eredményeként kapott cellák elterülnek. Ennek magyarázata a következő: Jelölje a cellák szélességét *w*. A cellák sorszámozását a C programozási nyelvben megszokott módon 0-tól kezdjük. Az *i*. cella helye: [*iw, (i+1)w*], a *j*. cella helye: [*jw, (j+1)w*], a cellák közepének helye pedig: *(i+0.5)w* és *(j+0.5)w* Az eredménycella helye ekkor: [*(i+j)w, (i+j+2)w*], ami éppen az *(i+j)*. és az *(i+j+1)*. cellának felel meg. Mivel a cellaszélesség a duplájára nőtt, a magasságot felezni kell, hogy az eredmény továbbra is sűrűségfüggvény legyen.



30. ábra Összeadandó sűrűségfüggvények – a cellaszélesség nem hanyagolható el



31. ábra A konvolúció eredménye – a cellaszélesség nem hanyagolható el

4.4.2. A forgalomadagonkénti routolás

A routolási egység méretének (S_{RU} – Size of Routing Unit) megválasztásakor két ellentétes irányba ható követelményre kell tekintettel lennünk: Minél nagyobb S_{RU}, az első fázisban annál kevesebb üzenetet kell elroutolni, a második fázisban pedig annál kevesebb forgalommodell összeadást kell végezni, így annál gyorsabb az analízis. Ha azonban S_{RU} túl nagy, akkor a forgalom szétosztása durván eltérhet a részletes szimuláció (és a valóság) esetén kialakuló térbeli eloszlástól. Ha S_{RU} kicsi, a forgalom szétosztása pontos lehet, de a túl sok routolandó üzenet és forgalommodell összeadás lassítja az analízist. ***S_{RU} megválasztása tehát az adott rendszer ismeretében kialakítandó ésszerű kompromisszum***. Erre látunk majd példát az analitikus vizsgálattal való összehasonlításról szóló 4.6. fejezetben.

A TFA még egy fontos tulajdonságát kell itt kiemelnem. Az a tény, hogy S_{RU} az egyes alkalmazástípusokra külön-külön megválasztható és $S_{RU}<1$ is lehet, a következő előnnyel jár: Ha valamely típusú alkalmazásból a többiekhez képest nagyságrendileg kevesebb van, de forgalmának térbeli eloszlása valamilyen oknál fogva mégis fontos a számunkra, akkor részletes szimuláció esetén olyan hosszú ideig kell a vizsgálatot végeznünk, hogy abból a forgalomból is kellő számú mintán alapuló statisztika álljon rendelkezésünkre. Közben más típusú alkalmazások forgalmából az elegendően pontos statisztikagyűjtéshez szükséges forgalom sokszorosát kell a hálózaton átvinnünk. Ezzel szemben TFA esetén ha S_{RU} méretét alkalmazástípusonként jól választjuk meg, akkor csak a kellő pontosságú statisztikagyűjtéshez szükséges forgalom néhányszorosát kell átvinnünk, ami óriási megtakarítással jár. A TFA ilyen módon az **importance sampling** megvalósítása is.

4.5. Példa a TFA alkalmazására

Példaként most tekintsünk egy kereskedelmi hálózatot, amelyen sokféle alkalmazás működhet, például: POS (Point Of Sale), ATM (Automatic Teller Machine), file transzfer, riasztó rendszerek, stb. Ezeket a TFA számára a paraméterezhető **általános alkalmazásmodell** (General Application Model, GAM) írja le. Az általános alkalmazásmodell legfontosabb paramétere az általa képviselt alkalmazások **típus**a (type). Számos, típusonként megadott paraméter határozza meg a GAM forgalmát.

Mint már korábban említettem, az ***alkalmazásmodell aggregált jellegű***, azaz képes kifejezni az adott csomóponthoz kapcsolt összes, a saját típusának megfelelő alkalmazás forgalmát.

Ha a routolási algoritmus szimmetrikus, azaz ugyanolyan A-tól B-ig, mint B-től A-ig, akkor az A és B egymás közti forgalma együtt kezelhető, és elegendő az A-tól B-ig végigvinni a mindkét irányt leíró statisztikákat. Például: egy-egy "kliens" a forgalmát leíró statisztikákat egy vagy több "szerver" felé küldi, amely a csomagokat fogadja, de forgalmat (választ) nem generál, mert a vizsgálat során a kliens által küldött statisztikák leírják mind az alkalmazás által küldött, mind az általa vett (azaz visszafele irányuló) forgalmat.

4.5.1. A GAM típusonkénti paraméterei

A példaként tekintett kereskedelmi hálózat tervezése esetén megbecsüljük, hogy hány alkalmazás fog futni a rendszerben, azoknak milyen lesz a területi eloszlása, illetve azt, hogy milyen lesz az alkalmazások aktivitásának napon belüli időeloszlása. A TFA számára pedig meg kell adnunk, hogy mekkora forgalomadagot szabad a routolás szempontjából egyben kezelni. A modell típusonkénti paraméterei tehát a következők:

- $N_{App}(type)$: Number of Applications – alkalmazásszám: az adott típusbeli összes alkalmazás száma a rendszerben.
- $N_{AppPN}(type, node)$: Number of Applications Per Node – alkalmazások node-onkénti száma: az adott típusbeli alkalmazások száma az adott node-on.
- $P_{Ac}(type, day, time)$: Probability of Activity – aktivitásvalószínűség: annak a valószínűsége, hogy egy adott típusú alkalmazás az adott napon és adott időben éppen aktív.
- $S_{RU}(type)$: Size of Routing Unit – routolási egység mérete: az adott alkalmazástípus esetén legfeljebb ekkora lehet a routolás szempontjából együttesen kezelt forgalom.

A figyelmes olvasó észrevehette, hogy a felhasználói viselkedés szempontjából figyelembe vesszük a napon belüli időt, sőt megkülönböztettük a hét napjait is. Ez utóbbira főleg a munkanapokon és munkaszüneti vagy szabadnapokon való eltérő felhasználói viselkedés modellezése miatt van szükség. Természetesen lehetnek olyan alkalmazások, amelyeknek a használata a napnak a hónapon belüli helyétől, vagy az évszaktól is függ, ekkor azt is figyelembe kell venni, de mivel elvi újdonságot nem hoz, ezért elkerüljük a feleslegesen sok paraméter használatát.

Az alkalmazásmodell tehát a hálózatban való szétosztás (elroutolás) céljából

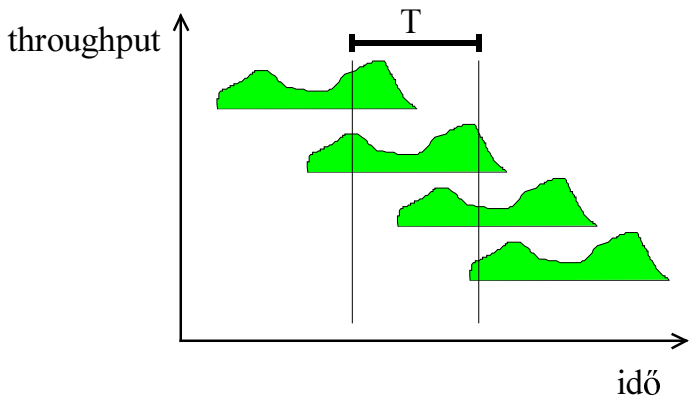
$$N_{App}(type, node, day, time) = \frac{N_{AppPN}(type, node) P_{Ac}(type, day, time)}{N_{App}(type)}$$

alkalmazás forgalmát bocsátja ki, legfeljebb $S_{RU}(type)$ méretű egységekben, ahol N_{App} és S_{RU} nem feltétlenül egész szám.

Az alkalmazásmodell forgalmát típustól függően adott számú **tranzakciótípus**ba (*transaction type*, rövidítve: *tt*) sorolható tranzakciók alkotják. Modellünkben a tranzakciók alkotják a forgalom tovább nem struktúrált alapegységét.

4.5.2. A tranzakciók időbeli lefutásának kiejtése

Modellünkben feltételezzük, hogy a tranzakciók létrejötte intenzitásának időbeli változása a tranzakciók időtartamához képest lassú, ezért úgy tekintjük, hogy egy *T* időintervallumban létrejövő tranzakciók kezdete az időintervallumban egyenletesen van elosztva, sőt a *T* intervallumot megelőző időben is hasonló volt a tranzakciók létrejöttének intenzitása. Ilyen módon azt mondhatjuk, hogy egy alkalmasan választott *T* időintervallumhoz éppen annyi tranzakció *teljes forgalmát* rendeljük, amennyi a *T* időintervallum alatt jön létre. Ezt illusztrálja a 32. ábra.



32. ábra Tranzakciók időbeli lefutásának kiejtése

4.5.3. A GAM tranzakcióinak paraméterei

A GAM által generált tranzakciókat is paraméterekkel jellemezzük. A paramétereknek le kell írniuk, hogy az adott típusba tartozó tranzakciók mekkora terhelést jelentenek a vonalak és csomópontok számára, naponta mennyi történik belőlük, és a napon belül milyen az időbeli eloszlásuk. A tranzakciókat jellemző paraméterek tehát:

- $N_b^{\{s|r\}}(tt)$: Number of Bits Sent | Received – küldött | fogadott bitek száma: egy adott típusba tartozó tranzakció alatt az alkalmazás által küldött | fogadott bitek száma
- $N_p^{\{s|r\}}(tt)$: Number of Packets Sent | Received – küldött | fogadott csomagok száma: egy adott típusba tartozó tranzakció alatt az alkalmazás által küldött | fogadott csomagok száma
- $N_c(tt)$: Number of Connection Setups – kapcsolatfelépítések száma: egy adott típusba tartozó tranzakció alatt történő kapcsolatfelépítések száma, tipikusan 0 vagy 1, ha egyáltalán lehetséges (csak kapcsolat orientált protokollok esetén)
- $N_{TrPD}(tt, day)$: Number of Transactions Per Day – tranzakciók száma naponként: a rendszerben a hét adott napján megtörténő, az adott típusba tartozó összes tranzakciók száma.
- $I_{Tr}(tt,day,time)$: Intensity of Transactions – tranzakciósűrűség: a hét adott napján az adott típusú tranzakciók időbeli eloszlását leíró sűrűségfüggvény értéke adott virtuális időpontban.

Megjegyzések:

1. Kapcsolatorientált hálózati protokollok esetén az esetleges tranzakciónkénti kapcsolatfelépítés és bontás a kapcsológépekben bizonyos esetben jelentős terhelést okozhat, például egy kapcsolatfelépítés terhelése megegyezik 10 darab csomag kapcsolásának terhelésével. Ezért vezettem be az N_c paramétert.
2. Az I_{Tr} értelmezve van a hét minden forgalommal bíró napjára, sűrűségfüggvény lévén:
   $\int_{time=0}^{24x3600} I_{Tr}(tt, day, time) = 1$ (ahol az időt másodpercben mérjük).
3. Az I_{Tr} függvény a teljes rendszerre vonatkozik, nem veszi figyelembe a területenként eltérő felhasználói viselkedést.

4.5.4. Poisson eloszlású tranzakciógenerálás

Modellünkben az aktív alkalmazások λ paraméterű Poisson folyamat szerint generálnak tranzakciókat, vagyis az érkezési időköz eloszlása λ paraméterű exponenciális eloszlású valószínűségi változó. Tehát annak a valószínűsége, hogy T idő alatt k tranzakció keletkezik:

$$p_T[k] = \frac{(\lambda T)^k}{k!} e^{-\lambda T}$$

Adott nap adott időpontjában egy node-on lévő alkalmazások által generált tranzakciók létrejöttének intenzitása:

$$\lambda(tt, node, day, t) = \frac{N_{AppPN}(type, node) \cdot N_{TrPD}(tt, day)}{N_{App}(type)} I_{Tr}(tt, day, time)$$

Megjegyzés: mivel a tranzakciótípus meghatározza az alkalmazástípust is, ezért a *tt* megadása esetén a *type* elhagyható.

A forgalommodell tehát tranzakciótípusonkénti λ-ból, N_c-ből és az adott típusú tranzakciókra jellemző N_b^s, N_b^r, N_p^s, N_p^r paraméterekből áll.

4.5.5. Az összeadás

Használjuk ki a modell tulajdonságaiból adódó egyszerűsítési lehetőségeket!

Mivel két Poisson eloszlású valószínűségi változó összege is Poisson eloszlást követ, tetszőleges hálózati elemen az adott tranzakciótípushoz tartozó (akár egyazon forrásból származó, de külön részletben routolt, akár külön forrásból származó) λ értékeket numerikusan összeadhatjuk. Az egyes elemek tehát tranzakciótípusonként egy-egy skaláris változó segítségével gyűjthetik a rajtuk áthaladó forgalmat. Ez azért fontos eredmény, mert így Poisson eloszlás esetén a forgalom térbeli elosztása során a skaláris intenzitás értéket ($\lambda(tt)$) tranzakciótípusonként összeadva jelentős számú konvolúciót küszöbölhetünk ki.

Míg az egy típusba tartozó tranzakciók intenzitás értékeit egyszerűen összeadhattuk, a különböző típusú tranzakciók intenzitásával ezt nem tehetjük meg, hiszen azok forgalmi jellemzői

(N_b^s, N_b^r, N_p^s, N_p^r, N_c) eltérnek. Példaként tekintsük az adott vonalon egy irányban küldött bitek forgalmát. A bit-throughput hisztogramot megkapjuk, ha a $\lambda(tt)$ paraméterű Poisson eloszlás által meghatározott hisztogram celláinak szélességét szorozzuk, magasságát pedig osztjuk az $N_b^s(tt)$ mennyiséggel. Az így nyert küldött bit-throughput eloszlásokat konvolúcióval összegezzük az összes tranzakciótípusra. A vonalnak az adott irányban való bitforgalmának a meghatározásának érdekében a fenti értékhez hozzá kell adnunk a másik irányban megfigyelt fogadott bitforgalmat, amit teljesen hasonlóan számíthatunk. A kapacitáskorlátozást a korábban ismertetett módon végezzük el.

4.5.6. A hatékony implementáció kérdései

A program futását gyorsítandó alkalmaztam néhány esetben paraméterként megadható ε értéket is annak kifejezésére, hogy mi a felhasználó által elvárt pontosság. Ezek a következők:

- ε_P**: Epsilon of Poissonian Tail** – Akkor használjuk, mikor az összegzett λ értékből létrehozzuk a Poisson eloszlás statisztikáját. Ha valamely k-ra $p_T[k]<\varepsilon_P$, akkor az eloszlás k-tól kezdődő farkát elhanyagoljuk.
- ε_A**: Epsilon of Addition** – A fentiekhez hasonlóan az összeadáshoz használt konvolúció eredményét korlátozzuk így. Ennek hiányában sok összeadás esetén a kis valószínűségű farokcellák száma igen megnőne, ami megnövelné a konvolúció műveletigényét.
- ε_C**: Epsilon of Change** – A kapacitáskorlátozásnál kiszámítjuk a legelső korrigált eloszlás $p_T^2[k]$ és az eredeti eloszlás $p_T[k]$ különbségét: firstChange. Minden korrekció után kiszámítjuk az előzőhöz képesti változást, azaz $p_T^i[k$ és $p_T^{(i-1)}[k]$ különbségét: actualChange. Megállunk, ha:

  $actualChange < \varepsilon_C * firstChange$.

A hatékonyság kérdései közt a végrehajtási időn kívül igen fontos szerepet kap az analízis elvégzése érdekében felhasznált emberi munka mennyisége. Ennek csökkentése érdekében objektum orientált TFA könyvtárat készítettem. A különböző objektumok, például forgalom generátor, vonal, csomópont, forgalom nyelő, stb. egy-egy C++ osztály, amelynek néhány tiszta virtuális függvényét a 3. szintű protokollok implementációjakor kell megírni. Ez a megoldás kisebb munkát ró a konkrét modell elkészítőjére, mintha a modellt neki kellene megírnia és csupán egy előre definiált TFA függvénykönyvtár állna rendelkezésére.

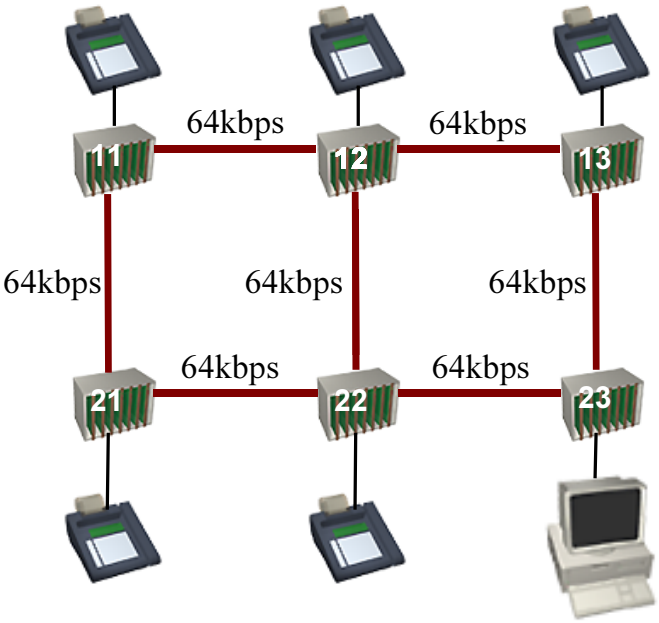
Nem használtam ugyan, de a konvolúció kiváltható FFT és FFT^{-1} transzformációval és a Fourier transzformáltak közötti szorzással. Ez különösen akkor kifizetődő, ha nagy a felbontás és a konvolúciók száma.

4.6. A TFA és az analitikus vizsgálat összehasonlítása

A két módszer összehasonlítására használjunk egy nagyon egyszerű, 6 csomópontból álló hálózatot. A csomópontokat 64kbps-os vonalak kötik össze a 33. ábra szerinti topológiában. Az egyik csomóponthoz egy szerver kapcsolódik, amelyet az 5 másik csomópont valamelyikéhez kötött terminálok használnak. Csomópontonként legyen 150 terminál, melyek 60% va-

lószínűséggel üzemelnek, és ha be vannak kapcsolva, akkor átlagosan percenként két tranzakciót hajtanak végre. A terminálok egy tranzakció során 5 csomagot küldenek a szervernek, és 4-et kapnak. A csomagok hosszeloszlása egyenletes a [140 byte, 180 byte] intervallumban.

A hálózatot olyan X.25 protokoll szerint működő kapcsológépek építik fel, amelyek nem fix routing táblát használnak, hanem node-onként megkeresik a legkisebb költségű útvonalat: Egy kapcsológépen való áthaladás költsége C_{Sw} (switching cost), egy vonalon való áthaladás költsége pedig arányos a rajta fennálló kapcsolatok számával, a szorzótényező C_{pC} (cost per connection).



33. ábra A példahálózat topológiája (Iminet[2] grafika)

Állapítsuk meg a TFA paramétereit! Egyetlen egy alkalmazástípus van, és ahhoz is csak egyetlen tranzakciótípus tartozik, így ezeket a továbbiakban elhagyjuk. Az alkalmazások száma a rendszerben: $N_{App} = 5 \times 150 = 750$. Az 5 csomópont mindegyikére az alkalmazások node-onkénti száma: $N_{AppPN} = 150$, az aktivitásvalószínűség: $P_{Ac} = 0.6$. Egy tranzakció alatt az alkalmazás által küldött bitek száma: $N_b^s = 6400$ a fogadott bitek száma: $N_b^r = 5120$, a küldött csomagok száma: $N_p^s = 5$, a vett csomagok száma: $N_p^r = 4$, a kapcsolat felépítések száma: $N_c = 1$, a nap folyamán létrejövő összes tranzakciók száma pedig:

$$N_{TrPD} = N_{App} \times P_{Ac} \times (1/30) \times 24 \times 3600, \text{ azaz}$$

$$N_{TrPD} = 15 \times 86400$$

Mivel a feladat szövege a tranzakciók napi eloszlásáról nem nyilatkozik, tételezzünk fel egyenletes eloszlást, azaz a sűrűségfüggvény értéke egész nap:

[2] Az Iminet az Elassys Consulting Kft. hálózati szakértői rendszere.

$I_{Tr} \equiv 1/86400$

A TFA fontos paramétere a routolási egység mérete S_{RU}. Ennek többféle értéke mellett is elvégeztem az analízist.

Az 18., 19., és 20. táblázat mutatja az egyes vonalak forgalmának várható értékére kapott eredményeket S_{RU} =0.1, 1, 10 esetére. Összehasonlításképpen megadom az egyes vonalakra számítás útján kapott ideális értékeket, amelyek természetesen mindhárom táblázatban azonosak. Helytakarékossági okokból minden vonalon csak a terminálóktól a szerver felé irányuló forgalmat közöljük, a másik irány a forgalom térbeli szétosztásának szempontjából úgysem hordoz további információt. Mivel az egyes routolási egységek kibocsájtása közt adott intervallumban egyenletes eloszlással sorsolt véletlen virtuális idő telik el, az analízis eredménye függ egy véletlenszám sorozat adott reprezentációjától. Az analízist többször végrehajtva az eredményeket átlagoltam, minimumot, maximumot és szórást számoltam.

vonal	elvi	átlag	szórás	min	max
11→12	15360	15337	66.5	15104	15488
12→13	23040	23009	108.8	22699	23232
21→22	23040	23063	127.5	22912	23296
22→23	53760	53792	153.9	53568	54101
11→21	3840	3863	167.6	3712	4096
12→22	11520	11528	178.3	11328	11626
13→23	42240	42209	198.0	41899	42432

18. táblázat Vonalkihasználtság S_{RU} *= 0.1 mellett*

vonal	elvi	átlag	szórás	min	max
11→12	15360	15181	378.4	14080	15787
12→13	23040	22859	533.1	21333	23680
21→22	23040	23219	653.7	22613	24320
22→23	53760	53941	753.9	53120	55467
11→21	3840	4019	843.5	3413	5120
12→22	11520	11522	881.1	10453	12160
13→23	42240	42059	957.8	40533	42880

19. táblázat Vonalkihasználtság S_{RU} *= 1 mellett*

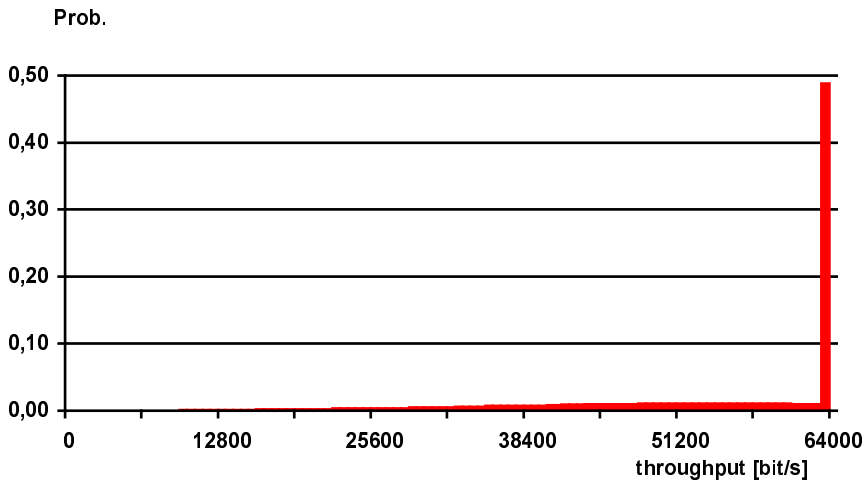
Az eredményekről megállapíthatjuk, hogy S_{RU} = 0.1 mellett a szórás mindenhol a várható érték 5%-a alatt van, nagy terheltségű vonalak esetén pedig 1%, sőt 0.5% alatt is. A minimális és maximális értékek közötti eltérés is kicsi, így ebben az esetben egyetlen kísérlet eredménye is nagy valószínűséggel jó közelítést ad. S_{RU} = 10 mellett az átlagok a 11→21 vonal kivételé-

vel még mindig jól közelítik az elvi értéket, de a minimum és maximum tekintetében már jelentősebb eltérések mutatkoznak, így mindenképpen csak több kísérlet átlagát szabad figyelembe venni. Vizsgáljuk meg, mi okozza a 11→21 vonal esetén a jelentős eltérést. Az elméleti vizsgálat során kiderült, hogy ezen a vonalon várható értékben 18 tranzakció halad át. Ennek ismeretében érthető, hogy 10-es routolási egység mellett ez az érték nem közelíthető jól. A problémát természetesen kezelhetjük a routolási egység méretének csökkentésével és/vagy a kísérletek számának növelésével, de ez nem mindig szükséges. Jelen esetben például, ha a vonalkapacitást semmiképpen nem akarjuk 64k alá csökkenteni, és csak arra a kérdésre keressük a választ, hogy a 64k mindenhol elég-e, számunkra teljesen közömbös a kis terheltségű vonalak kihasználtságának pontos értéke, elég azt tudnunk, hogy az a vonalkapacitásnál sokkal kisebb.

vonal	elvi	átlag	szórás	min	max
11→12	15360	14229	1096	10667	17067
12→13	23040	22101	1834	19200	23467
21→22	23040	24171	2136	21333	27733
22→23	53760	54699	2593	53333	57600
11→21	3840	4971	2815	2133	8533
12→22	11520	11328	3000	8533	12800
13→23	42240	41301	3341	38400	42667

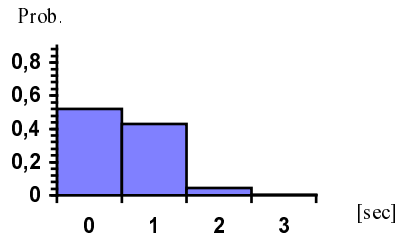
20. táblázat Vonalkihasználtság S_{RU} *= 10 mellett*

A várható értéken túl vizsgáljuk meg az egyes vonalak bit-throughput eloszlását és késleltetés eloszlását! Válasszunk ki két érdekes vonalat! Legyen az első a 22→23 vonal. Ennek a számított terhelése 53 760 bps, ami meghaladja a 64k-nak az ökölszabály szerint biztonságosnak tekintett 80%-át.



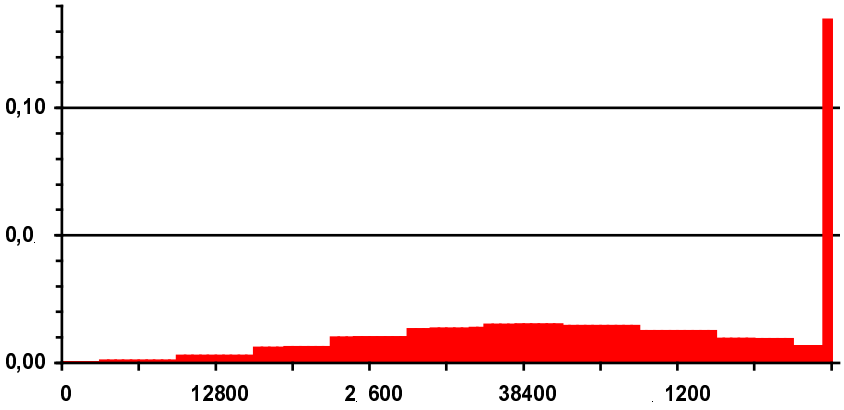
34. ábra A 22→23 vonal bit-throughput eloszlása

A 34. ábrán látható a 22→23 vonal bit-throughput eloszlása. Szembetűnő, hogy a vonal majdnem 50% valószínűséggel teljesen telített. Ennek az árát a 35. ábrán látható késleltetés-eloszlással fizetjük meg. Ebben az esetben megfontolandó egy 128k kapacitású vonal alkalmazása.

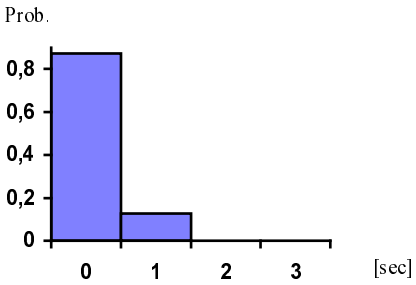


35. ábra A $D_T[i]$ késleltetés eloszlás alakulása a 22→23 vonalon

A 36. ábrán látható a 13→23 vonal bit-throughput eloszlása. Itt már lényegesen kisebb annak a valószínűsége, hogy a vonal teljesen telített legyen. A 37. ábrán látható késleltetéseloszlást már elfogadhatónak tekintem.



36. ábra A 13→23 vonal bit-throughput eloszlása



37. ábra A $D_T[i]$ késleltetés eloszlás alakulása a 13→23 vonalon

A többi vonalnál nincs lényeges valószínűsége annak, hogy a forgalom elérje a vonalkapacitást, elemzésüknek számunkra nincs elvi jelentősége.

Ezzel a példával azt kívántam szemléltetni, hogyan lehet a TFA segítségével a hálózatban lévő szűk keresztmetszeteket feltárni.

4.7. A TFA módszerrel kapcsolatos megállapításaim

Két fázisban működő (forgalom térbeli szétosztása, időeloszlás meghatározása a véges kapacitások figyelembevételével) módszert (traffic-flow analysis) alkottam, amely a részletes szimulációhoz képest sokkal gyorsabban közelítő képet ad a hálózat forgalmi viszonyairól.

Megadtam a módszer alkalmazhatósági kritériumait.

Throughput-eloszlás, mint forgalommodell esetén kidolgoztam a kapacitáskorlátozás és a késleltetés eloszlás számítását.

Bemutattam egy konkrét TFA implementációt. Egyszerű példahálózat esetén a TFA eredményei jól közelítik az analitikus módszer eredményeit.

A TFA hatékony módszer lehet kommunikációs hálózatok gyors, közelítő elemzésére.

A további kutatás iránya lehet az, hogy a TFA módszert a részletes szimulációval hasonlítjuk össze összetettebb, analitikusan egyszerű módon nem végigszámolható esetben, mind az eredmények, mind a végrehajtáshoz szükséges idő tekintetében.

Megvizsgálandóak még a térbeli szétosztás, kapacitáskorlátozás pontossági és konvergencia kérdései.

5. Az eredmények alkalmazása

A jövőbeli események halmazával kapcsolatos eredményeimet felhasználták az OMNeT++ diszkrét idejű szimulátor implementációjakor: a FES az eredetei láncolt lista helyett kupac lett.

Az általam kidolgozott TFA módszert az Elassys Consulting Kft. alkalmazza az Iminet hálózati szakértői rendszerében.

Az SSM-T példaként megjelenik az OMNeT++ rendszerben, az Iminetben pedig az alkalmazására irányuló fejlesztés folyamatban van.

Az SSM-T módszer az ausztráliai Monash Universityvel való együttműködés keretében a közeljövőben szuperszámítógépen is kipróbálásra kerül.

6. Irodalomjegyzék

Aho, A. V.; J. E. Hopcroft; and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1975.

ANSI X3.139. "Fiber Distributed Data Interface (FDDI) Token Ring Media Access Control (MAC)". American National Standards Institute, New York, NY, 1987.

Banks, J.; J. S. Carson; B. L. Nelson *Discrete-Event System Simulation.* Prentice Hall, Upper Saddle River, New Jersey, 1996.

Barron, A. R.; L. Győrfi; E. C. Meulen. "Distribution Estimation Consistent in Total Variation and in Two types of Information Divergence" *IEEE Transactions on Information Theory* Vol 38, No. 5, September 1992, pp. 1437-1454.

Beirlant, J.; L. Győrfi. "On the Asymptotic Normality of the L_2-Error in Partitioning Regression Estimation" *Journal of Statistical Planning and Inference*, 71, (1998) pp. 93-107.

Bratley P.; B. L. Fox; and L. E. Schrage *A Guide to Simulation*. Springer-Verlag, New York, 1986.

Chandy, K.M., and J. Misra "Distributed simulation: A case study in design and verification of distributed programs" *IEEE Transactions on Software Engineering SE-5*, 5, September 1979, pp. 440-452.

Chandy, K.M., and J. Misra "Asynchronous distributed simulation via a sequence of parallel computations" *Communications of the ACM* 24, no. 11, November 1981, pp. 198-205.

Devroye, L.; L. Győrfi. *Nonparametric Density Estimation: The L1 view*. John Wiley, New York, 1985.

Fujimoto, R. M. "Parallel Discrete Event Simulation". *Communications of the ACM* 33, (1990.) no 10, 31-53

Gonnet, G. H. *Handbook of Algorithms and Data Structures.* Addison-Wesley, 1984.

Jain, R.; and I. Chlamtac "The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms Without Storing Observations". *Communications of the ACM* **28** (1985), no. 10, pp. 1076-1085.

Jain, R. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, 1991.

Jávor, A. *Simulation in Research and Development.* North-Holland, Amsterdam, 1985.

Jávor, A. "Petri Nets in Simulation" *EUROSIM Simulation News Europe*, 1993, no. 9, pp. 6-7.

Jefferson, D; B. Beckman; F. Wieland; L. Blume; M. DiLoreto; P. Hontalas; P. Laroche; K. Sturdevant; J. Tupman; V. Warren; J. Vedel; H. Younger and S. Bellenot. "Distributed Simulation and the Time Warp Operating System". *Proceedings of the 12th SIGOPS - Symposium on Operating System Principles*, (1987) pp. 73-93.

Knuth, D. E. *The Art of Computer Programming. Vol. 3. (Sorting and Searching)* Addison-Wesley, 1973

Lencse, G. "Investigation of Event Set Algorithms" *Proceedings of the 1995 European Simulation Multiconference (ESM 95)* (Prague, Czech Republic, June 5-7). SCS Europe, pp. 821-825.

Lencse, G. "Performance of Future Event Set Implementations" *Journal on Communications*, Vol. XLVII, December 1996. (pp. 33.)

Lencse, G. "Efficient Simulation of Large Systems - Transient Behaviour and Accuracy" *Proceedings of the 1997 European Simulation Symposium (ESS'97)* (Passau, Germany, Oct. 19-23). SCS Europe, 660-665.

Lencse, G. "Efficient Parallel Simulation with the Statistical Synchronization Method" *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation (CNDS'98)* (San Diego, CA. Jan. 11-14). SCS International, 3-8.

Lencse, G. "Statistics Collection for the Statistical Synchronisation Method" *Proceedings of the 1998 European Simulation Symposium (ESS'98)* (Nottingham, UK. Oct. 26-28). SCS Europe, 46-51.

Lencse, G. "Applicability Criteria of the Statistical Synchronization Method" *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation (CNDS'99)* (San Francisco, CA. Jan. 17-20). SCS International, pp. 159-164.

Lencse, G. "Design Criterion for the Statistics Exchange Control Algorithm used in the Statistical Synchronization Method" *Proceedings of the Advanced Simulation Technologies Conference (ASTC 1999)* part of the 32nd Annual Simulation Symposium, April 11-15 1999, San Diego, CA, USA (pp. 138-144).

Lencse, G. "Traffic-Flow Analysis for Fast Performance Estimation of Communication Systems" - *elfogadva a Journal of Computing and Information Technology-nál*

MIL 3. "OPNET Example Models Manual", Release 3. (1996.) (Chapter FDDI) MIL 3, Inc.

Misra, J. "Distributed discrete-event simulation" *ACM Computing Survey 18*, 1, March 1986, pp. 39-65.

Pongor, Gy. "Statistical Synchronization: a Different Approach of Parallel Discrete Event Simulation". Proceedings of the 1992 European Simulation Symposium (ESS 92) (Dresden, Germany. Nov. 5-8) SCS Europe, pp. 125-129.

Pongor, Gy. "Multiple Virtual Times in Parallel Discrete Event Simulation". Proceedings of the Parallel Processing Wokshop (Technical Univ. of Budapest, Budapest, Hungary, Febr. 10-11, 1994.)

Pugh, V. "Skip Lists: A Probabilistic Alternative to Balanced Trees" *Communications of the ACM,* 33, no. 6, (June) 1990. 668-676

Reeves, C. M. "Complexity Analyses of Event Set Algorithms" *The Computer Journal, 27.* (1984.) no. 1, 72-79

Scott, D. W. *Multivariate Density Estimation* John Wiley & Sons, Inc., 1992.

Tanenbaum, A. S. *Computer Networks.* Second Edition, Prentice Hall Inc., 1989.

Varga, A.; B. Fakhamzadeh. "The K-Split Algorithm for the PDF Approximation of Multi-Dimensional Empirical Distributions without Storing Observations*". Proceedings of the 9th European Simulation Symposium (ESS'97)* (Passau, Germany, Oct. 19-22, 1997.) SCS Europe, pp.94-98.

Varga, A. "K-split – On-Line Density Estimation for Simulation Result Collection". *Proceedings of the European Simulation Symposium* (Nottingham, UK. Oct. 26-28, 1998.). SCS Europe, 41-45.

Varga, A. "OMNeT++ Discrete Event Simulation System" (2000) http://www.hit.bme.hu/phd/vargaa/omnetpp.htm

Wirth, N. *Algorithms + Data Structures = Programs.* Prentice Hall, 1976

A. függelék: az értekezésben használt rövidítések

Az értekezésben használt rövidítések magyarázatát fejezetenként ABC sorrendben adom meg.

1. Bevezetés

rövidítés	angol megfelelő	magyar megfelelő
FES	Future Event Set	jövőbeli események halmaza
QoS	Quality of Service	a szolgáltatás minősége
SSM	Statistical Synchronisation Method	statisztikai szinkronizációs módszer
SSM-T	the time-driven version of the Statistical Synchronisation Method	statisztikai szinkronizációs módszer idővezérelt változata

2. A párhuzamosítás kérdései

rövidítés	angol megfelelő	magyar megfelelő
ANSI	American National Standards Institute	Amerikai Szabványügyi Testület
ATM	Asynchronous Transfer Mode	<hálózati szabvány neve>
EIK	-	Egyetemi Információs Központ (BME)
FES	Future Event Set	jövőbeli események halmaza
FDDI	Fiber Distributed Data Interface	<hálózati szabvány neve>
FIFO	First In First Out	az első bejövő távozik először (sorrend)
IIF	Input Interface	bemeneti interfész
LP	Logical Process	logikai processz
LVT	Local Virtual Time	helyi virtuális idő
MAC	Media Access Control	közeghozzáférési (alréteg)
OIF	Output Interface	kimeneti interfész
OMNeT++	Objective Modular Network Testbed	<eseményvezérelt diszkrét idejű szimulációs rendszer neve>
PDES	Parallel Discrete-Event Simulation	párhuzamos diszkrét idejű szimuláció
PSTN	Public Switched Telephone Network	nyilvános kapcsolt telefonhálózat
PVM	Parallel Virtual Machine	virtuális párhuzamos számítógép
SSM	Statistical Synchronisation Method	statisztikai szinkronizációs módszer

SSM-C	the counter-driven version of the Statistical Synchronisation Method	statisztikai szinkronizációs módszer számláló-vezérelt változata
SSM-T	the time-driven version of the Statistical Synchronisation Method	statisztikai szinkronizációs módszer idő-vezérelt változata
TTRT	Target Token Rotation Time	megcélzott token körbejárási idő
T_Opr	the operative value of TTRT	a TTRT operatív értéke (amiben megállapodtak az állomások)
UI	Update Interval	frissítési intervallum

3. A jövőbeli események halmazának kezelése

rövidítés	**angol megfelelő**	**magyar megfelelő**
FES	Future Event Set	jövőbeli események halmaza

4. A traffic-flow analysis nevű új módszer

rövidítés	**angol megfelelő**	**magyar megfelelő**
ATM	Automatic Teller Machine	pénzkiadó automata
GAM	General Application Model	általános alkalmazásmodell
POS	Point of Sale	elektronikus fizetőhely
S_{RU}	Size of Routing Unit	routolási egysége mérete
TFA	Traffic-Flow Analysis	forgalomfolyam analízis

Megjegyzés: a példa-alkalmazás paramétereinek értelmezését a 4.5. fejezetben megadtam, azokat itt nem soroltam fel újra.

B. függelék: ábrák és táblázatok jegyzéke

Ábrák

Táblázatok