Mérési útmutató a Mobil Kommunikáció és Kvantumtechnológiák Laboratórium méréseihez

Transzport protokollok vizsgálata Ns2 szimulációs környezetben

Mérés helye:

Híradástechnikai Tanszék
Mobil Kommunikáció és Kvantumtechnológiák Laboratórium (MCL)
I.B.113.

Összeállította:
Huszák Árpád

Utolsó módosítás:
2012. augusztus 17.

1. Bevezetés

Az ISO/OSI rétegstruktúrában, a hálózati réteg (network layer) felett elhelyezkedő szállítási réteg feladata a sorrendhelyes, duplikációmentes és megbízható átvitel biztosítása a felhasználói alkalmazástól függően. Számos transzport protokoll áll rendelkezésünkre, amelyek közül az alkalmazás típusától függően választjuk ki a legalkalmasabbat.

2. Transzport protokollok

A szállítási réteg két régi nagy protokollja a TCP (Transmission Control Protocol) [1] és az UDP (User Datagram Protocol) [2]. Ezek a protokollok a mai napig meghatározói a számítógépek valamint számítógép-hálózatok közötti adattovábbításnak, pedig immár 25 éves szabványok, hiszen a TCP-t és az UDP-t is az 1980-as évek elején fejlesztették ki. Ez az oka annak, hogy mindkét szabványt viszonylag kis hibavalószínűségű, vezetékes hálózatra dolgozták ki, azonban a ma egyre szélesebb körben használt vezeték nélküli hálózatok karakterisztikái jelentősen különböznek vezetékes hálózatok adatátviteli tulajdonságaitól. Az eltelt évek alatt a vezetékes számítógépes hálózatok is nagyon sokat fejlődtek. Emiatt a régi protokollokat felül kell vizsgálni, és az új igényeknek megfelelően módosítani kell azokat, illetve újabb szabványok kifejlesztésére van szükség.

A mai Internet protokollokat olyan vezetékes összeköttetésekre dolgozták melyeknek a jellemzőik a következők: nagy sávszélesség, kis késleltetés, kis hibavalószínűség. Az elmúlt időszakban megjelent és egyre népszerűbb vezeték nélküli hálózatok átvitelére azonban pont az ellenkező tulajdonságok jellemzőek: kisebb sávszélesség, nagy késleltetés, nagy hibavalószínűség.

A két legelterjedtebb szállítási rétegbeli protokoll, a TCP és az UDP, nagyon különböző célt szolgálnak. A TCP egy kapcsolatorientált és megbízható adatforgalmat biztosít a felhasználónak, míg az UDP megbízhatatlan. A TCP esetében a megbízhatóság azt jelenti, hogy az elküldött csomagok biztosan megérkeznek, de az esetleges újraküldések miatti késleltetésre nincs garancia, míg UDP esetén a küldő elküldi a csomagot és ezután a hálózaton múlik, hogy megérkezik-e. Újraküldésből adódó késleltetéssel ekkor nem kell számolni. Ezen jellemzői miatt az UDP-t kizárólag olyan esetekben alkalmazzák, ahol a küldött adatok egy részének elvesztése nem okoz működési problémát, sőt, esetleg kívánatos is (pl. torlódás esetén), mint pl. a műsorszórás; vagy ahol ezen hibák korrigálásáról egy magasabb szintű protokoll gondoskodik. Az 1. Táblázat tartalmazza a különböző alkalmazásokhoz leggyakrabban használt transzport protokollokat.

Alkalmazás	Alklam. réteg protokollja	Szállítási réteg
e-mail	SMTP	TCP
távoli hozzáférés	Telnet	TCP
Web	HTTP	TCP
file átvitel	FTP	TCP
távoli file server	NFS	UDP
multimédia	egyedi	UDP, UDPLite, SCTP, DCCP
IP telefónia	egyedi	UDP, UDPLite, DCCP
hálózat menedzsment	SNMP	UDP
útvonalválasztás - routing	RIP	UDP

1. táblázat: Alkalmazások és protokolljaik

A közelmúltban több új transzport protokollt fejlesztettek ki, melyek megpróbálják kiküszöbölni a régebbi protokollok hibáit. Ilyen az UDP módosított változata az UDP Lite (Lightweight User Datagram Protocol) [3] vagy a multimédiás átvitelre szánt megbízható

SCTP (Stream Control Transport Protocol) [4] és a megbízhatatlan DCCP (Datagram Congestion Control Protocol) [5].

2.1. TCP (Transmission Control Protocol)

A mai számítógépes hálózatokban 80-90%-ban a TCP-t (Transmission Control Protocol) [1] használják, melynek legnagyobb előnye a megbízhatóság. Mivel az Internet nem központilag vezérelt hálózat, és egyes elemeit más-más tulajdonosok birtokolják, az Internet nem látja el felhasználóit a hálózat állapotát jellemző információkkal. Így az egyes csomagküldő számítógépek csak a saját maguk által kiküldött csomagokat és a vevő által nyugtaként visszaküldött nyugtákból származó információkat használhatják működésükhöz. Csupán a kommunikáció végpontjai szólhatnak bele a csomagáramlás irányításába. Ezt az alapelvet az Internet „végtől végig" (end to end) vezérlőelvének nevezik. A számítógépek túlnyomó része a csomagküldések ütemezésére a fenti elveket leginkább figyelembe vevő algoritmust megvalósító protokollt, a TCP-t (Transmission Control Protocol) használja.

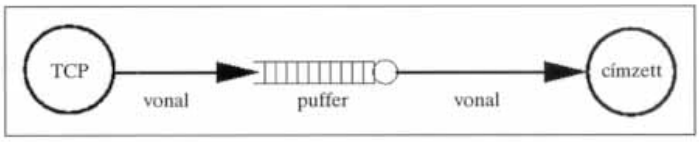
A TCP fejlesztői a hálózatról azt feltételezték, hogy a csatorna hibavalószínűsége minimális, így a csomagvesztés oka többnyire a torlódás. Ennek megfelelően dolgozták ki a protokollt, melynek működése leegyszerűsítve a következő:

A kapcsolat kezdetén a számítógép még nem tud semmit a hálózat állapotáról, ezért először csak egy csomagot küld. Ha sikeresen megérkezik a csomag a címzetthez, akkor az nyugtacsomaggal válaszol. A csomag feladása és a nyugta megérkezése közt eltelt időt körbefordulási időnek, angolul „round trip time"-nak vagy röviden RTT-nek hívják. A TCP állandóan, minden csomag esetén méri és nyilvántartja ezt az időt, mert a hálózat állapotára csak az RTT adatokból tud következtetni. A növekvő RTT értékek mindig arra utalnak, hogy a csomagok egyre hosszabb ideig várakoznak a routerekben, ami a hálózat terheltségének egyik fokmérője. Amikor az első csomagról a nyugta visszaérkezik, a TCP egymás után két csomagot ad fel. Az ezekre érkező nyugták megérkezésekor szintén két-két csomagot ad fel és így tovább. A TCP-nek ezt a működési módját slow-start-nak hívják. Ez a beküldött csomagok számának exponenciális növekedését okozza, a slow-start elnevezés tehát egy kissé félrevezető. Az egymást követő RTT nagyságú időintervallumokban 1, 2, 4, 8... csomag kiküldésére kerül sor. Ez a folyamat addig folytatódik, amíg az egyik feladott csomagra nem érkezik nyugta, ami a csomag elveszését jelzi. A csomagok elvesztését a TCP két módon tudja detektálni:

Az egyik módszer azon alapul, hogy az Interneten két számítógép között a csomagok többnyire ugyanazon az útvonalon haladnak és a csomagok feladásuk sorrendjében érkeznek meg a címzetthez. Amennyiben egy csomag nyugtája hamarabb érkezik meg, mint egy korábban feladott csomagé, akkor feltehető, hogy ez utóbbi csomag elveszett.
A másik módszer azon alapul, hogy az algoritmus figyeli az RTT értékek alakulását. Ha az utoljára mért néhány RTT érték átlagát jelentősen meghaladó időn túl sem érkezik nyugta, az algoritmus a csomagot elveszettnek nyilvánítja.

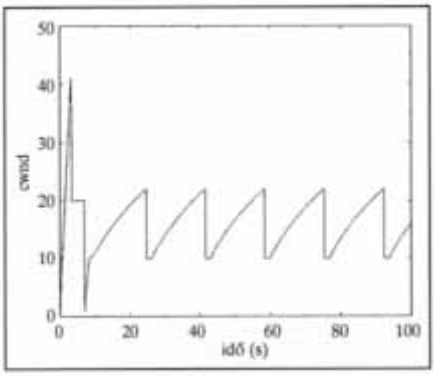
A csomagvesztés után a TCP óvatosabbá válik. Megvárja, amíg a vesztés észlelésekor a hálózatban lévő csomagjainak feléről visszatér a nyugta, és csak ezután növeli ismét – óvatosan – a hálózatban tartott csomagjainak számát. Ennek módszere a következő: az algoritmus használ egy torlódási ablaknak (congestion window vagy röviden cwnd) nevezett változót. Amennyiben a kiküldött, de még nyugtázatlan csomagjainak száma nagyobb, mint a torlódási ablak egész része ([cwnd]), akkor egy nyugta beérkezésekor nem küld ki csomagot, és a cwnd értékét sem változtatja. Ha a hálózatban annyi nyugtázatlan csomag van, mint [cwnd], akkor egy nyugta beérkezésekor kiküld egy csomagot, és a cwnd értékéhez hozzáadja

az 1/[cwnd] számot (cwnd'=cwnd+1/[cwnd]). Ha a [cwnd] értéke nagyobb a nyugtázatlan csomagok számánál, akkor egy nyugta beérkezésekor két csomagot küld ki, és a cwnd értékéhez 1/[cwnd]-t ad hozzá. Ha csomagvesztés lép fel, akkor a cwnd értékét felezi és az így adódó szám egész részére, de legalább 1-re állítja be az új értéket (cwnd'=max(1,[cwnd/2])). A TCP-nek ezt a működési állapotát torlódáskerülő (congestion avoidance) üzemmódnak hívják.
Például, ha kezdetben 5 nyugtázatlan csomagunk van a hálózatban és cwnd=5, akkor a nyugták megérkezésekor a TCP egy-egy csomagot küld ki. Így a nyugtázatlan csomagok száma mindig 5 marad. A cwnd értéke ekkor az 5,2; 5,4; 5,6; 5,8 értékeket veszi fel. Az ötödik nyugta visszaérkezésekor a cwnd értéke 6,0 lesz. Ennek egész része 6, és ekkor a TCP ennek a nyugtának a beérkezésekor két csomagot küld ki 6-ra növelve a nyugtázatlan csomagok számát a hálózatban. Így ebben a módban minden egyes RTT idő alatt eggyel növekszik a nyugtázatlan csomagok száma a hálózatban, azaz az idővel lineárisan, nem pedig exponenciálisan (állandó RTT értéket feltételezve) változik. A torlódáskerülő mód szemléltetésére tekintsük az 1. ábrán látható kapcsolást. Ebben egy számítógép csomagokat küld egy router pufferén keresztül egy másik számítógépnek.



1. ábra: Egyszerű kapcsolás két számítógép között. A bal oldali TCP-vel jelölt számítógép egy pufferrel ellátott routeren keresztül csomagokat küld a címzett számítógépnek. Egyéb csomagforgalom nincs

A TCP egyre növeli a hálózatban tartott csomagok számát. Amennyiben a vonalak csak kismértékben késleltetik a csomagokat, a csomagok a pufferben torlódnak fel. A torlódás addig tart, míg a puffer meg nem telik, túl nem csordul, és a TCP által küldött egyik csomag el nem veszik. Ekkor a TCP megfelezi a hálózatban tartott csomagjainak számát és a feltorlódási folyamat újrakezdődik. Ennek megfelelően ebben az egyszerű rendszerben a cwnd változó időben periodikusan változik a puffer maximális nagyságát jellemző B érték és annak fele (B/2) között. A folyamat közben a pufferben feltorlódó csomagoknak egyre hosszabban kell várakozniuk az előttük várakozó csomagok miatt. Így a körbejárási idő is folyamatosan növekszik. A 2. ábra a cnwd érékének időben periodikus változását mutatja be.



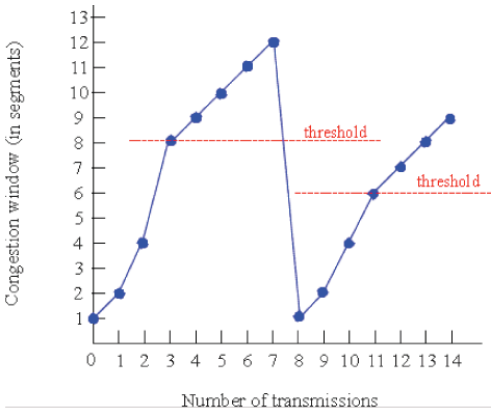
2. ábra: A cwnd időbeli viselkedése az 1 ábrának megfelelő elrendezés esetén. A cwnd a B és B/2 értékek között mozog periodikusan. A cwnd minden RTT alatt eggyel nő, viselkedése mégsem lineáris, mivel az RTT értékek is növekednek a pufferben kialakuló várakozás miatt

Végül a TCP harmadik, „visszafogott" (back-off) működési módja akkor aktivizálódik, amikor a torlódás és csomagvesztés olyan nagy a hálózatban, hogy a cwnd értéke 1-re esik vissza, és az RTT időnkénti egy csomag kézbesítése sem sikerül. Ilyenkor, sikertelen csomagküldés után, a TCP a következő csomagot két RTT várakozás után próbálja elküldeni,

majd ha ez ismét nem sikerül, akkor ehhez hasonlóan 4, 8, 16, 32 és 64 RTT várakozás után próbálja újra elküldeni a csomagot. Ha valamelyik lépésben sikerül elküldenie a csomagot, ismét visszatér a torlódáselkerülő módba.
A valódi algoritmus ennél persze bonyolultabb. Talán meglepő, de ez az egyszerű dinamikus rendszer bevált a mai viszonyok között is, és sikeresen megelőzi, hogy az Internet összeomoljon.
A TCP-nek több változata is létezik, azonban mindegyik az eddig ismertetett alapokon nyugszik. A kutatók a több torlódáselkerülő mechanizmust is kidolgoztak, mint pl. a Tahoe, Reno, New-Reno, SACK, Vegas.

2.1.1. TCP Tahoe

A Tahoe slow-start mechanizmusa hasonlóan a 2.1 fejezetben ismertetetett eljáráshoz, szintén exponenciálisan növeli a hálózatban lévő csomagok számát. Ezt addig teszi, amíg vagy csomagvesztés nem történik, vagy el nem ér egy küldési sebesség küszöbértéket. A küszöb elérése után ismét csak egyesével növeli a hálózatban lévő csomagok számát. Ha csomagvesztés történik a torlódási ablak értékét egyre állítja, az új küszöbérték a küldési sebesség fele lesz, és a slow-start kezdődik előről.

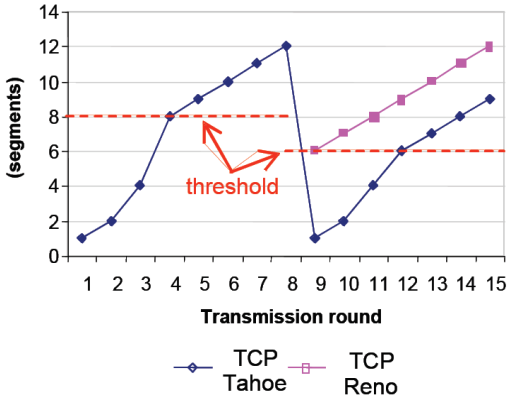


3. ábra: TCP Tahoe torlódási ablak mérete

Fontos megjegyezni, hogy ez a változat az elküldött csomagokhoz rendelt időzítő alapján dönt a csomag újraküldéséről. Ehhez a késleltetéshez adódik, a kumulatív nyugtázásból (egyszerre több csomagot nyugtáz, nem pedig külön minden csomagot) adódó késleltetés.

2.1.2. TCP Reno

Ez a változat a csomagvesztés gyorsabb detektálását teszi lehetővé a Fast Retransmit algoritmus alkalmazásával. Ennek lényege, hogy szinte azonnal nyugta érkezzen a küldőhöz, ahogy a vevő megkapta a csomagot. Amennyiben három duplikált nyugta érkezik a forráshoz, az csomagvesztést érzékel és újraküldi a csomagot, anélkül, hogy az időzítő lejárna.
Másik módosítás, hogy csomagvesztés esetén nem állítja a torlódási ablak méretét egyre, hanem megfelezi annak értékét, valamint a TCP ablakot is azonos erre az értékre állítja. Ezután pedig már egyesével növeli a hálózatban lévő csomagok számát.



4. ábra: TCP Reno torlódási ablak mérete

A TCP Reno nem működik hatékonyan, ha túl nagy a csomagvesztés aránya, mivel a csomagvesztést detektáló algoritmusa nem tudja megkülönböztetni a többszörös hibákat.

2.1.3. TCP SACK

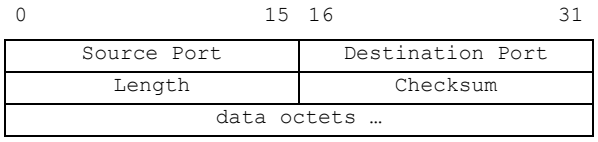
Ez a változat a Selective Acknowledgement módszerrel kiterjesztett TCP Reno-t takarja. A TCP SACK megoldást ad a többszörös csomagvesztés detektálására, oly módon, hogy egyszeres nyugtázást alkalmaz. Egy csomagra egy nyugta érkezik.

2.1.4. TCP Vegas

A TCP Vegas a szintén a TCP Reno módosított változata. Az eddig bemutatottak mindegyike a csomagvesztés, vagyis a torlódás után tesz lépéseket a torlódás megszüntetésére. A TCP Vegas a hálózat aktuális paraméterei alapján kívánja megelőzni a hálózat túlterhelését. Az RTT mérésével becsülni lehet, hogy az elküldött csomagra, mikor kell a nyugtának megérkeznie. A TCP Vegas egy algoritmus alapján számolja a becsült küldési sebességet, majd összehasonlítja az aktuális értékkel. Ha a két érték nagyon közel van egymáshoz, akkor a csökkenti a sebességet.

2.2. UDP (User Datagram Protocol)

Az IP protokoll csak két gép közötti adattovábbítást biztosítja. Nem teszi lehetővé az alkalmazások vagy a felhasználók azonosítását. Az UDP [2] szállítási protokoll biztosítja, hogy egy gépen egyidejűleg futó több alkalmazói program egymástól függetlenül küldhessen és fogadhasson csomagokat.
Az UDP sokkal gyorsabb protokoll, mint a TCP protokoll, viszont nem megbízható adatátvitel szempontjából. Nem kapcsolat orientált, nincs hibajavítás, nincs nyugtázás. Tulajdonképpen az IP szint által biztosított szolgáltatásokat nyújtja felfelé. Akkor szokták használni, ha az adatátvitel sebessége a legfontosabb és a csomagvesztés megengedett, minden többi feladatot a felette elhelyezkedő réteg lát el. Tipikusan a DNS-ek (Domain Name Server), real-time alkalmazások, játékok szokták használni. Egy játékban vagy real-time hangátvitel esetén, ha egy csomag rossz vagy hiányzik, akkor ott felhasználó legfeljebb döccenést észlel, de ez még mindig kisebb baj, mintha az adott pontnál megállna, és onnantól elkezdené újra adni a csomagokat. Az UDP estén ugyanis nem kell várni az újraküldésre így az esetleges csomaghibák nem blokkolják a küldő oldalt, hiszen az csupán elküldi a csomagokat egymás után, és nem foglalkozik azzal, hogy mi történ azokkal. A szegényesebb szolgáltatásból adódóan sokkal egyszerűbb az UDP fejléc.

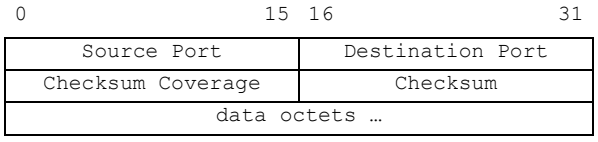


5. ábra: UDP fejléc

Az UDP esetében is felmerül a mobilitásból adódó változó hibaarány, ami időnként a kapcsolat teljes megszakadásához vezet. A nyugtázásra ugyan nem kell várni, de a változó csatornaminőség és a cellaváltások itt is komoly gondot okoznak.
A hibaarány növekedéséből adódó csomagvesztésre legegyszerűbb módon úgy lehet védekezni, hogy a hatékonyabb FEC (Forward Error Connection) hibavédő kódolást alkalmazunk. Ezzel azonban növeljük az átviendő adatmennyiséget, nagyobb lesz az overhead, és a hálózati terhelés is növekszik.

2.2.1. UDP Lite (Lightweight User Datagram Protocol)

Az UDP módosításával létrehoztak egy újabb protokollt az UDP-Lite-t [3] amely valójában az új valósidejű multimédiás szolgáltatások kiszolgálására jött létre. Az új protokoll csupán annyiban változott az eredeti UDP-hez képest, hogy egy új részleges ellenőrző-összeget vezettek be. Amennyiben a csomagnak abban részében keletkezik bithiba, amelyet a részleges ellenőrző-összeg lefed, akkor a vevő érzékeli a hibát és eldobja a csomagot, míg ha a hiba olyan helyen van, amit a részleges ellenőrzőösszeg nem fed le, akkor nem dobja el. Ebben az esetben az alkalmazásnak kell kezelnie a hibás csomagot. Ezzel az eljárással kb. 40%-kal csökken a csomag eldobások száma. Abban az esetben, ha az UDP ellenőrzőösszege az egész csomagra kiterjed, az UDP Lite működése megegyezik a hagyományos UDP működésével.



6. ábra: UDP-Lite fejléc

Egyik legjelentősebb ok, amiért létrehozták az UDP Lite protokollt, mert az alkalmazások egy csoportja kezelni tudja a hibás csomagokat is. A felhasználó által megfigyelhető minőség jobb lesz, ha a hibás csomagok nem kerülnek eldobásra, hanem az alkalmazásig eljutnak. Több hang és video codec is ehhez az alkalmazás csoportba tartozik: ITU-H.263, ITU-H.265, MPEG-4 video codec (ISO-14496). Ezek a kódolók jobb minőséget nyújtanak hibás csomagok kezelésével, mint ha egyáltalán nincs is csomag.
A szállítási réteg hibaérzékelő mechanizmusának, tehát védenie kell az alapvető információkat, mint a fejléc, de szükség szerint az adatok egy részét vagy egészét is. Annak meghatározása, hogy mely adatokat kell lefedni ellenőrző összeggel, a küldő oldali alkalmazás feladata.

2.3. DCCP (Datagram Congestion Control Protocol)

A DCCP egy megbízhatatlan transzport protokoll, amely torlódásszabályozási algoritmus használatára, valamint sorrendhelyes csomagtovábbításra is alkalmas a TCP-hez hasonlóan. UDP esetén a torlódás elleni védekezést az alkalmazásoknak kellett megoldaniuk, míg DCCP esetén ez már a protokoll szerves része. A DCCP-t úgy próbálták alakítani, hogy a TCP és az UDP előnyeit egy protokollként valósítsák meg. A DCCP fejlécben így ráismerhetünk az előbbi protokollokból ismert fejlécmezőkre.
A fejléc hossza minimálisan 12 byte, maximálisan pedig a 1024 byte-t is elérheti, ha az opcionális mezőket, és az egyes csomagtípusok esetén használt pótlólagos mezőket is

használjuk. Általános esetben az ellenőrzőösszeg (Checksum), az összes adatot lefedi, de az UDP Lite-hoz hasonlóan, a DCCP is lehetővé teszi az adatok részleges lefedését ellenőrzőösszeggel. Ez lehetőséget ad arra, hogy azok az alkalmazások, amelyek képesek kezelni a sérült adatokat, hatékonyabban működjenek.
A kapcsolatorientált DCCP a kapcsolat felépítése során megbízható protokollként működik. A torlódásszabályozással kapcsolatos üzenetek szintén megbízható adatfolyamként kerül továbbításra. Jelenleg két torlódásszabályozó algoritmust specifikáltak:

1. TCP-like Congestion Control [CCID 2]
2. TFRC (TCP-Friendly Rate Control) Congestion Control [CCID 3]

A torlódásszabályozási algoritmust a kapcsolat felépítés során kerül meghatározásra. Az adatfolyam során azonban mind a vevő, mind az adó oldal kezdeményezheti a torlódásszabályozási algoritmus megváltoztatását. Jelenleg ugyan csak két ilyen algoritmus van specifikálva, de a DCCP protokollt felkészítették esetleges újabb torlódáskezelési módszerek bevezetésére is. A torlódásszabályozó algoritmusokat a fejléc CCID (Congestion Control Identifier) mezőjében különböztetjük meg.
A TCP-Like algoritmus a 2.1 fejezetben ismertetett TCP torlódáselkerülési technikáját alkalmazza, amire jellemző a hirtelen sebességcsökkenés. A TFRC egy képlet (1) alapján határozza meg a küldési sebességet, aminek köszönhetően nincsenek drasztikus sebességváltozások.

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + 4R(3\sqrt{\frac{3p}{8}} \cdot p \cdot (1+32p^2))} \quad (1)$$

A képletben szereplő változók: T – küldési sebesség; s – csomagméret; R – körbefordulási idő (RTT); p – csomagvesztési arány.
Az adatfolyam ugyan megbízhatatlan maradt, de az adó oldal értesül a vevő által fogadott csomagok helyes megérkezéséről. A vevő nyugtát küld az érkezett csomagokról. A torlódáskezelő algoritmus egyben azt is meghatározza, hogy milyen gyakran érkeznek nyugták. TCP-like (CCID 2) esetben körülbelül két elküldött csomag után érkezik nyugta, míg a TCP-Friendly Rate Control (CCID 3) esetén körbefordulási időnként (Round Trip Time) küld egy nyugtát a vevő.
A DCCP képes annak meghatározására is, hogy milyen okból történt csomagvesztés. Ez az opció fontos lehet a torlódásszabályozó algoritmus számára, hiszen abban az esetben, ha például bithiba keletkezik, vagy a vevőoldali buffer túlcsordulása miatt kerül sor csomageldobásra, nincs szükség torlódásszabályozó algoritmus beavatkozására.
A DCCP-t olyan alkalmazások számára fejlesztették ki, mint például a streaming médiaalkalmazások, amelyek ki tudják használni a DCCP beépített szabályozási módszereit. Annak érdekében, hogy a DCCP hatékonyan vegye fel a versenyt a gyors UDP-vel, a DCCP csomagok fejlécét próbálták a lehető legkisebbre méretezni. A protokoll bizonyos feladatok estén még így is túlságosan bonyolult, ezért kifejlesztettek egy egyszerűsített DCCP protokollt is, melynek neve DCCP-Lite [6].

2.4. SCTP (Stream Control Transport Protocol)

Az SCTP [4] egy megbízható szállítási rétegbeli protokoll, mely hibamentes kommunikációt biztosít nyugták használatával. A protokoll torlódáselkerülő algoritmust is használ, amely nem teljesen azonos a TCP algoritmusával.
Bizonyos alkalmazások esetén megengedhető a nem sorrendhelyes átvitel is, amelyre az SCTP ad megoldást. A protokoll lehetővé teszi mind a sorrendhelyes, mind a nem

sorrendhelyes átviteli módot. Fontos megjegyezni, hogy egy SCTP csomagon belül több folyamot is meg lehet különböztetni. Ez streaming alkalmazások esetén fontos, amennyiben a felhasználóhoz több audió/videó folyamot szeretnénk elkülönítve továbbítani.

3. Transport protokollok Ns2-ben

Az Ns2 nagyon jó lehetőséget biztosít a szállítási réteg protokolljainak vizsgálatára, hiszen a protokollok nagy része már implementálva van, így a felhasználónak csupán ezeket kell alkalmaznia a számára megfelelő beállításokkal.

A vizsgálatokat a topológia és a kommunikáció eseményeinek megadásával kell kezdeni, amelyet OTcl nyelven tehetünk meg. Először a csomópontokat kell létrehozni, majd a linkeket, időzítési-, nyomkövetési feladatok leírását kell megadni.

A létrehozott csomópontokra megadott nevükkel hivatkozhatunk a továbbiakban (n0, n1, n2). Abban az esetben, ha a csomópont nem egy router, hanem egy forrás vagy nyelő végpont, forgalomszabályozó ügynököt (traffic agent) kell a végponthoz rendelni. Ez az ügynök valósítja meg a szállítási réteg funkcióit, tehát valójában a transzport protokollt kell megjelölni (TCP, UDP, stb.). Emellett a forrás típusát is meg kell adni (pl. FTP (File Transfer Protocol), CBR (Constant Bit Rate), VBR (Variable Bit Rate), stb.).
Leggyakrabban használt forgalomszabályozó ügynökök a TCP és UDP ügynökök, melyeknek számos típusa érhető el az alap Ns2-ben. Legismertebb TCP változatok: TCP Tahoe, TCP Reno, TCP SACK, TCP Vegas.

- Agent/TCP – a Tahoe TCP
- Agent/TCP/Reno – Reno TCP
- Agent/TCP/Sack1 – TCP selective acknowledgement
- Agent/TCP/Vegas – TCP Vegas

Az UDP-nek is van módosított változata, az UDPLite, ez azonban nem része az alap Ns2-nek, ennek ellenére már az UDPLite patch már letölthető, és használható.

A leggyakoribb forrásmodellek és alkalmazástípusok:
- Application/FTP – adatfolyam, melyet TCP továbbít
- Application/Traffic/CBR – állandó csomagküldési sebességet generál
- Application/Traffic/Exponential – olyan On-Off modell, ahol a küldési periódus és a néma periódus hossza exponenciális eloszlást mutat
- Application/Traffic/Trace – a forgalom egy un. trace fájl alapján alakul, melyben a csomagok mérete és a küldési időpontok szerepelnek

A következő példában az **n0** csomóponthoz rendelünk hozzá egy CBR forrást, amely UDP protokollt használ, az `attach-agent` parancs használatával:

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$udp0 set packet_size_ 1000
$cbr0 set rate_ 1000000
```

Az ügynököknek különböző konfigurációs paraméterei is vannak, mint pl. a CBR esetén a küldési sebesség, vagy UDP és TCP esetén a csomagméret byte-ban.
Hasonlóan az előzőhöz, most az **n1** csomóponthoz egy TCP alapú FTP alkalmazást kapcsolunk.

```
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$tcp1 set packet_size_ 1000
```

Eddig a forrásokról volt szó, azonban vevő oldali csomópontot is létre kell hozni. A TCP nyelő az Agent/TCPSink osztályban van definiálva, míg az UDP nyelő az Agent/Null osztályban.
A következő példában az **n2** csomóponthoz rendelünk egy nyelő ügynököt, majd létrehozzuk a kapcsolatot a forrás és a nyelő között

```
set null [new Agent/Null]
$ns attach-agent $n2 $null
$ns connect $udp0 $null
```

Az általános TCP nyelő minden helyesen megérkezett csomagra nyugtával válaszol. Az előzőekhez hasonlóan hozzuk létre a kapcsolatot a TCP forrás és nyelő között:

```
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp1 $sink
```

Amennyiben a nyomkövető (trace) fájlban nem találhatóak meg az általunk vizsgálni kívánt értékek, a különböző komponensek által használt változókat is követhetjük. Pl. TCP esetén a (ns-allinone-2.29\ns-2.29\tcp\tcp.cc) fájlban megtalálhatjuk, mely változókhoz férhetünk hozzá olvasásra:

```
#ifdef TCP_DELAY_BIND_ALL
#else /* ! TCP_DELAY_BIND_ALL */
     bind("t_seqno_", &t_seqno_);
     bind("rtt_", &t_rtt_);
     bind("srtt_", &t_srtt_);
     bind("rttvar_", &t_rttvar_);
     bind("backoff_", &t_backoff_);
     bind("dupacks_", &dupacks_);
     bind("seqno_", &curseq_);
     bind("ack_", &highest_ack_);
     bind("cwnd_", &cwnd_);
     bind("ssthresh_", &ssthresh_);
     bind("maxseq_", &maxseq_);
     bind("ndatapack_", &ndatapack_);
     bind("ndatabytes_", &ndatabytes_);
     bind("nackpack_", &nackpack_);
     bind("nrexmit_", &nrexmit_);
     bind("nrexmitpack_", &nrexmitpack_);
```

```
        bind("nrexmitbytes_", &nrexmitbytes_);
        bind("necnresponses_", &necnresponses_);
        bind("ncwndcuts_", &ncwndcuts_);
        bind("ncwndcuts1_", &ncwndcuts1_);
#endif /* TCP_DELAY_BIND_ALL */
```

Vannak egyéb paraméterek is, amelyeket mi magunk is állíthatunk. Ezek tetszőleges állítása a következő módon zajlik (pl. vevő oldali ablak méretének beállítása):

```
Agent/TCP set window_ 100000
```

Amikor egy csomag beérkezik egy csomóponthoz, az egy várakozási sorban tárolódik. A soroknak azonban több típusa is létezik (DropTail, RED, SFQ, stb.), attól függően, hogy a csomópont hogyan kezeli a bejövő csomagokat. Van azonban olyan paraméter, amely minden sorra megadható, mégpedig a várakozási sor mérete. DropTail típusú sor esetén ezt így adhatjuk meg:

```
Queue/DropTail set limit_ 10
```

A mérésekben többször lesz szükség a torlódási ablak (cwnd) és az aktuális sávszélesség vizsgálatára, amit a következő módon tehetünk meg:

```
proc record {} {
global sink0 f0 f1
      #Get an instance of the simulator
         set ns [Simulator instance]
      #Set the time after which the procedure is called again
         set time 0.5
      #How many bytes have been received by the traffic sinks?
         set bw0 [$sink0 set bytes_]
      #Get the current time
         set now [$ns now]
      #Calculate bandwidth (MBit/s) and write it to the files
         puts $f0 "$now [expr $bw0/$time*8/1000000]"
      #Reset the bytes_ values on the traffic sinks
         $sink0 set bytes_ 0
      #Congestion window size
         set cwnd [$tcp set cwnd_]
         puts $f1 "$cwnd"
      #Re-schedule the procedure
         $ns at [expr $now+$time] "record"
}
```

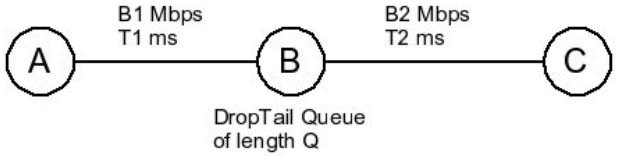
Az ismertetett módszerek csupán töredékét képezik az Ns2 lehetőségeinek. Jól használható segítséget jelenthet az Ns2 témánkénti leírása és használati útmutatója, amely az ”ns-allinone-2.35\ns-2.35\doc\” könyvtárban található.

4. Ellenőrző kérdések

1. Sorolja fel az ön által ismert transzport protokollokat, és azok variánsait! Milyen típusú alkalmazások használják a felsorolt protokollokat?
2. Miben különbözik az UDP-Lite az UDP-től?
3. Milyen működési módjai vannak a TCP-nek? Röviden ismertesse ezeket!
4. Mi a ”slow-start”? Ismertesse működését!
5. Milyen TCP változatokat ismer? Miben különböznek egymástól?
6. Miben különbözik a DCCP két torlódáskezelő algoritmusa?

5. Mérési feladatok

1. Írja meg a következő elrendezéshez tartozó OTcl scriptet:



7. ábra: Topológia

Hozzon létre FTP/TCP kapcsolatot az A és C csomópont között. A csomagok mérete 1250 byte, valamint B1=25, B2=10, T1=10, T2=20, Q=10.
Az 50 másodpercig tartó szimulációt végezze el TCP Tahoe, TCP Reno és TCP Vegas szállítási protokollal. Állítsa a TCP vevő oldali ablak méretét 100000-re, annak érdekében, hogy ne a vevő oldal legyen a hálózat szűk keresztmetszete:

```
Agent/TCP set window_ 100000
```

a) Ábrázolja a torlódási ablakot az idő függvényében 0,1 másodperces időfelbontásban!
b) Ábrázolja a sávszélesség alakulását az idő függvényében 0,1 másodperces időfelbontásban!

2. A TCP Tahoe újraküldésének vizsgálatára generáljon véletlenszerű 5%-os csomagvesztést a B-C csomópont közé. Megtartva a 7. ábrán bemutatott topológiát, ahol most:
B1=2, B2=1, T1=10, T2=40, Q=5
Állítsa a TCP vevő oldali ablak méretét 100000-re, annak érdekében, hogy ne a vevő oldal legyen a hálózat szűk keresztmetszete:

```
Agent/TCP set window_ 100000
```

a) Hasonlítsa össze a teljes átvitt adatmennyiséget a hibamentes és az 5%-os csomagvesztés estén! Az adatmennyiség aránya 95%? Miért vagy miért nem?
b) Ábrázolja a teljes átvitt adatmennyiséget a késleltetés függvényében! A csomagvesztés B-C között maradjon 5%, a link késleltetése pedig legyen: 10ms, 60ms, 110ms és 160ms.
c) Ábrázolja a teljes átvitt adatmennyiséget a csomagvesztési arány függvényében! Vegye alapul a 7. ábrán bemutatott topológiát, a csomagvesztési arány pedig legyen: 25%, 5%, 1%, 0,2%.

6. Hivatkozások

[1] J. Postel: ”Transmission Control Protocol”, RFC-793, September 1981.
[2] J. Postel: ”User Datagram Protocol”, RFC-768, August 1980.
[3] Larzon, Degermark, Pink: “The Lightweight User Datagram Protocol”, RFC-4340, July 2004.
[4] R. Stewart: ” Stream Control Transmission Protocol”, RFC-2960, October 2000
[5] Kohler, Handley, Floyd: “Datagram Congestion Control Protocol”, RFC-, March 2006
[6] Phelan: ”Datagram Congestion Control Protocol - Lite (DCCP-Lite)”, draft-phelan-dccp-lite-00.txt, August 2003
[7] Ns2 /www.isi.edu/nsnam/ns/index.html