

TDK-dolgozat

Nagy Zoltán

Debreceni Egyetem, Informatikai Kar,
Informatikai Rendszerek és Hálózatok Tanszék

Nagysebességű TCP kapcsolatok teljesítmény-analízise
hangolható pufferek alkalmazásával

Nagy Zoltán
III. évfolyamos hallgató

Témavezető:

Dr. Orosz Péter, egyetemi adjunktus

Tartalomjegyzék

1. Bevezetés	4
2. Mérési környezet	5
2.1 Konfiguráció	5
2.2 A használt TCP torlódásvezérlő algoritmusok	8
2.2.1 TCP Reno	8
2.2.2 Highspeed TCP	8
2.2.3 Scalable TCP	8
2.2.4 TCP BIC	9
2.2.5 TCP CUBIC	9
2.2.6 TCP Hybla	9
2.2.7 Hamilton TCP	10
2.2.8 TCP Illinois	10
2.2.9 TCP Low Priority	10
2.2.10 TCP Vegas	10
2.2.11 TCP Veno	10
2.2.12 Yeah TCP	10
3. Mérési eredmények	11
3.1 Alkalmazott matematikai eszközök	11
3.1.1 Tapasztalati eloszlás	11
3.1.2 Exponenciális eloszlás	11
3.1.3 Maximum likelihood becslés	12
3.1.4 Általánosított Pareto eloszlás (GPD)	12
3.1.5 Közvetlen mintavételű L-momentum becslés	13
3.1.6 Anderson-Darling teszt	14
3.1.7 Említett matematikai eszközök konkrét megvalósítása	15
3.2 LAN mérések	15
3.2.1 Átviteli sebesség	15
3.2.2 Torlódási ablakméret	19
3.2.3 Következtetések	24
3.3 WAN mérések	26
3.3.1 Átviteli sebesség	26
3.3.2 Torlódási ablakméret	32
3.3.3 Következtetések	38
4. Összegzés	41
5. Tovább lépési lehetőségek	41
6. Irodalomjegyzék	42
7. Saját munka leírása	43
8. Köszönetnyilvánítás	43

1. Bevezetés

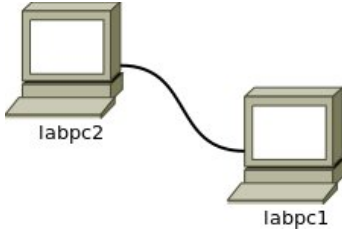
Az Ethernet technológia nagy fejlődésen ment keresztül létrejött, a 70-es évek óta. Kezdetben 2,94 Mbit/s-os sebességgel tudott üzemelni, de olyan sikeres volt, hogy a DEC (Intel és a Xerox) 1983-ban egy 10 Mbit/s-os vastag koaxiális kábelezésű Ethernet szabványban állapodott meg. Ebből jött létre később az IEEE 802.3 szabványcsalád. 1990-ben megjelent a 802.3i, ami 10 Mbit/s sebességet volt képes elérni sodrott érpár felett. Két évvel később megjelent a Fast Ethernet szabvány, mely 100 Mbit/s átviteli sebességgel rendelkezett. 1995-ben a 802-es bizottság egy még gyorsabb, 1 Gbit/s-os szabványon kezdett el dolgozni, ami a Gigabit Ethernet nevet kapta. A tízszer gyorsabb, 10 Gigabit Ethernet szabvány pedig 2002-ben került publikálásra [1]. Az egyre növekvő adatátviteli sebességnek köszönhetően nagyobb valószínűséggel történhet túlterhelés a forgalom statisztikus változása miatt. Ezért egyre nagyobb hangsúlyt kap a torlódásvezérlő algoritmusok használata a TCP alapú adatátvitel során. A klasszikus TCP torlódásvezérlő algoritmus már negyvenéves, így nem nyújt minden esetben hatékony működést. Megjelentek új igényekhez alakított TCP variánsok, de nincs egységesen elfogadott jó algoritmus nagy sebesség esetében. A Linux a CUBIC-ot, míg a Windows rendszerek egy saját fejlesztésű torlódásvezérlő algoritmust használnak. Ma már a technikai fejlődés miatt 1, 10, sőt ennél nagyobb gigabit/sec sávszélességű kapcsolatok is léteznek akár szerveroldali, végponti linkek formájában. Ilyen esetekben fontossá válik annak vizsgálata, hogy a variánsok hogyan viselkednek nagy átviteli sebességen a végponti pufferméret függvényében. Jelentek meg konferenciacikkék ezen vizsgálatokkal kapcsolatban, de a TCP-igazságosság (TCP-fairness) és a TCP-együttműködés (TCP-friendliness) szempontjainak figyelembe vétele mellett [2] [3] [4] [5]. Jelen dolgozatomban az átviteli teljesítményre helyeződik a hangsúly, ugyanis egy adatfolyam vizsgálatát végzem el, így nincs szükség párhuzamos TCP kapcsolatok szembeni méltányosság figyelembe vételére. Ezen okokból átviteli sebesség és ablakméret mérésére is szükség van. Az ablakméret ugyanis a kapcsolat átviteli performanciáját határozza meg. Minél nagyobb az ablakméret, annál nagyobb a küldési ráta, így annál nagyobb az átviteli teljesítmény. Így egyértelmű célom, hogy maximalizáljam az ablakméretet és minimalizáljam a veszteséget, ami pufferhangolással érhető el. További célom, hogy konkrét mérési eredmények leíró statisztikai elemzése segítségével igazoljam, hogy ezekben az implementációkban mely TCP torlódásvezérlő algoritmus és mely puffermemória-méretetek nyújtják a leghatékonyabb adatátvitelt egy TCP adatfolyam esetében.

2. Mérési környezet

2.1 Konfiguráció

A mérések közvetlen összeköttetéssel zajlottak le két általános célú PC munkaállomás között.

	labpc1	labpc2
CPU	Intel Core i7 870	Intel Core i5 3450
RAM	4 GB DDR3	8 GB DDR3
Hálózati eszköz	Intel 10G PCIe NIC	



The diagram illustrates a network connection between two desktop computers. On the left, a computer labeled 'labpc2' is shown. On the right, a computer labeled 'labpc1' is shown. A black cable connects the two computers, representing a network link.

1. táblázat :Laborgépek hardver konfigurációja

A gépeken Fedora 17 64-bites operációs rendszer futott 3.5.3 kernellel [6]. A forgalom generálásához 2.0.5 verziójú *iperf*-et használtam, ami egy - a hálózati útvonalak kapacitásának mérésére szolgáló - speciális szoftver [7]. Ennek köszönhetően a végponttól végpontig terjedő IP útvonalon elérhető aktuális bitrátát lehet tesztelni TCP, illetve UDP kapcsolatok esetében is. A labpc1 laborgép volt a forgalomgenerátor (*iperf* kliens), a fogadó (*iperf* szerver) pedig a labpc2 laborgép. A kettő között közvetlen 10 Gbit/s-os Ethernet kapcsolat került kialakításra [8]. A szegmensméret változások nyomon követése céljából a mérések alatt a *tcpprobe* kernel modul is engedélyezésre került [9].

A TCP kapcsolat átviteli karakterisztikájának javítása érdekében mindkét oldalon a küldő és fogadó puffereket hangoltam.

Módosításra került kernel paraméterek:

- `rmem_default`: Socket fogadó memória alapértelmezett mérete byte-ban.
- `rmem_max`: Socket fogadó memória maximális mérete byte-ban.
- `wmem_default`: Socket küldő memória alapértelmezett mérete byte-ban.
- `wmem_max`: Socket küldő memória maximális mérete byte-ban.
- `max_backlog`: A feldolgozásra váró csomagok maximális száma (CPU-nként).
- `tcp_rmem`: TCP csomagok fogadására felhasználható memória minimális, kezdeti és maximális mérete
- `tcp_wmem`: TCP csomagok küldésére felhasználható memória minimális, kezdeti

és maximális mérete

Első esetben a Linux kernel alapértelmezett kernel paramétereivel, a másodikban extrém nagy memória méretek beállításával indítottam a méréseket. A harmadik esetben pedig $B \times D$ szorzat szerinti hangolást alkalmaztam. Ennek megfelelően a TCP kernel paramétereiket a második esetben maximális méretre állítva, harmadikban $B \times D$ szorzat szerint számolt BDP (Bandwidth Delay Product) értékek szerint finomhangoltam. A két kártya közötti közvetlen összeköttetés késleltetésének meghatározására ping-et használtam, mellyel 120,712 μ s átlagos RTT értéket tapasztaltam. Ezt az értéket felezni kellett, mert a $B \times D$ érték meghatározásánál csak az egyik irányt kell figyelembe venni, ugyanis mindkét oldalon megvan a saját puffere (1).

$B \times D$ 10 Gbit/s esetben:

$$B = 10\,000\,000\,000 \text{ bit/s} \quad RTT = 0,000120712 \text{ s} \quad (RTT = 2 \times D)$$
$$2 \times B \times D = 1\,207\,120 \text{ bit} = 150\,890 \text{ byte} \quad (1)$$

A net.core.wmem_default, net.core.rmem_default paraméterek a kapott értékhez közeli 2 hatványát kapták. A net.core.wmem_max, net.core.rmem_max paraméterek pedig ennek a kétszeresét.

Paraméter	Alapértelmezett	Max. hangolt	B×D hangolt
net.core.wmem_max	131071	16777216	327680
net.core.rmem_max	131071	16777216	327680
net.core.wmem_default	126976	8388608	163840
net.core.rmem_default	126976	8388608	163840
net.core.netdev_max_backlog	1000	50000	10000
net.ipv4.tcp_wmem	4096 16384 4194304	4096 65536 16777216	4096 81920 33554432
net.ipv4.tcp_rmem	4096 87380 4194304	4096 87380 16777216	4096 81920 33554432

2. táblázat: Alkalmazott kernel paraméterek értékei a 3 LAN környezet esetében

Nagy távolságú WAN kapcsolat emulálása céljából netem modul használatával késleltetést indukálva is elvégeztem a méréseket mindhárom kernel paraméterezéssel. A Debrecen-Kalifornia között mért késleltetés nagyjából 180 ms volt, ennek a fele a Debrecen-New York

kapcsolaton.

A NetEm (Network Emulator) modul a Linux forgalomszabályozó lehetőségeihez tartozik. Segítségével késleltetést, csomagvesztést, átrendeződést és még számos hálózati paramétert lehet az adott interfészen kimenő csomagokhoz rendelni. 180 ms késleltetés előidézéséhez mindkét oldalon *netem delay 90ms 2ms 50%* került beállításra, mely 90 ms késleltetést eredményez 2 ms szórással, és a következő véletlenszerű elem 50%-os valószínűséggel függ az előző delay értéktől. 90 ms késleltetés esetén pedig *netem delay 45ms 2ms 50%* értékek kerültek a kommunikációban részt vevő interfészekhez.

A késleltetés növekedése miatt, a $B \times D$ paraméterek is növekedtek.

90 ms késleltetés esetén:

$$B = 10\,000\,000\,000\,bit/s$$

$$RTT = 0,0901293878\,s$$

(90 ms + vonalon mért késleltetés)

$$2 \times B \times D = 901\,293\,878\,bit = 112\,661\,735\,byte \quad (2)$$

180 ms késleltetés esetén:

$$B = 10\,000\,000\,000\,bit/s$$

$$RTT = 0,1801293878\,s$$

(180 ms + vonalon mért késleltetés)

$$2 \times B \times D = 1\,801\,293\,878\,bit = 225\,161\,735\,byte \quad (3)$$

Paraméter	B×D 90 ms delay	B×D 180 ms delay
net.core.wmem_max	225323470	450323470
net.core.rmem_max	225323470	450323470
net.core.wmem_default	112661735	225161735
net.core.rmem_default	112661735	225161735
net.core.netdev_max_backlog	10000	10000
net.ipv4.tcp_wmem	4096 8192 112661735	4096 8192 225161735
net.ipv4.tcp_rmem	4096 8192 112661735	4096 8192 225161735

3. táblázat: Alkalmazott kernel paraméterek értékei WAN környezetek esetében

Tizenkettő darab 2.2 fejezetben bemutatásra kerülő TCP torlódásvezérlő algoritmus vizsgálatával foglalkoztam három féle puffermemória méret alkalmazásával, így LAN környezetben 36, WAN környezetben pedig 72 mérés zajlott le. Minden mérés egy 10 perces *iperf* session volt, ahol egy adott TCP variánst állítottam be szimmetrikusan mindkét oldalon.

A mérések között legalább 1 perces szünet volt, hogy a pufferek kiürülhessenek.

2.2 A használt TCP torlódásvezérlő algoritmusok

2.2.1 TCP Reno

Az AIMD (Additive Increase Multiplicative Decrease) algoritmust használja. Ez az algoritmus egy szegmenssel növeli az ablakméretet minden beérkezett TCP nyugta esetén. Csomagvesztés bekövetkeztekor viszont felezi azt (4).

$$\begin{aligned} cwnd &= cwnd + 1 && \text{minden megérkezett nyugta után torlódásmentes} \\ &&& \text{időszakban} \\ cwnd &= cwnd - \frac{1}{2} * cwnd && \text{csomagvesztés esetén} \end{aligned} \quad (4)$$

Ez a torlódásvezérlő algoritmus nagysebességű kapcsolatok esetén nem nyújt hatékony működést a lassú ablaknövelés - és csomagvesztés esetén annak felezése - miatt.

2.2.2 Highspeed TCP

Módosításra került a torlódási ablak értékét megadó függvény, aminek két paramétere van: $a(w)$ és $b(w)$. Az első azt adja meg, hogy mennyivel növelje a torlódási ablak méretét, amikor nyugta érkezik, ami $a(w)/(w)$ RTT-nként. A második pedig, hogy csomagvesztés-detektálás esetén mennyivel csökkentse azt: $(1 - b(w))w$ három duplikált nyugta érkezésével. Kis ablakméretek esetében ezek a paraméterek a TCP Reno AIMD algoritmusával használt értékekkel egyeznek meg $a(w) = 1$ és $b(w) = 0,5$ így ilyenkor a Highspeed TCP a Reno-val megegyező módon működik. Ha a torlódási ablakméret egy bizonyos határérték fölé kerül, az $a(w)$ értéke növekszik és a $b(w)$ értéke csökken. Így az ablakméret gyorsabban nő, és csomagvesztés esetén kevésbé fog csökkenni [10].

2.2.3 Scalable TCP

Más mértékben változtatja a torlódási ablak méretét, mint a hagyományos TCP. A rossz ablakdinamika javítása érdekében módosított AIMD-t használ, melynek segítségével az ablakméret növelése gyorsabb, míg csomagvesztés miatti csökkenése kevésbé drasztikus. Módosított algoritmus a MIMD (Multiplicative Increase Multiplicative Decrease)

$$\begin{aligned} cwnd &= cwnd + 0,01 && \text{minden érkező nyugta után torlódásmentes} \\ &&& \text{időszakban} \end{aligned} \quad (5)$$

$$cwnd = 0,875 * cwnd \quad \text{csomagvesztés esetén}$$

További előnye, hogy a szegmensvesztés utáni helyreállási idő csak az RTT-vel arányos, a küldési sebességgel viszont nem [11].

2.2.4 TCP BIC

Nagysebességű, nagytávolságú kapcsolatokhoz kialakított variáns. A Linux 2.6.8 – 2.6.18-as verziójú kernel alapértelmezett TCP torlódásvezérlő mechanizmusa. Bináris kereső algoritmust alkalmaz, amely csomagvesztés esetén egy multiplikatív faktorral csökkenti a torlódási ablakot, ami a csomagvesztés előtt a maximum értékre van állítva, a csomagvesztés után pedig a minimum értékre. A BIC TCP ekkor egy bináris keresést hajt végre felhasználva a két paramétert és a torlódási ablakot. A két paraméter által meghatározott intervallum közéértékére állítja a torlódási ablakot. Ez a keresés addig ismétlődik, míg a növekmény értéke kisebb nem lesz egy előre megadott konstansnál. Ekkor a torlódási ablakot az aktuális maximum értékre állítja a protokoll, így az ablakméret logaritmikusan növelhető lesz [12].

2.2.5 TCP CUBIC

A CUBIC a BIC egy kevésbé agresszív változata, egyben a Linux kernel 2.6.19-es verziójától kezdődően alapértelmezett TCP torlódásvezérlő algoritmus [13]. A torlódási ablak legutolsó ablakméret-csökkentés óta eltelt idő egy harmadrendű függvénye:

$$W(t) = C(t - K)^3 + W_{max}$$

$$K = \sqrt[3]{W_{MAX} \beta / C}$$
(6)

Ahol:

- t : a legutolsó ablakméret csökkenés óta eltelt idő.
- C : skálázási faktor.
- W_{max} : az ablakméret a legutóbbi csökkentés előtt.
- β : egy konstans szorzási csökkenési faktor

2.2.6 TCP Hybla

Nagytávolságú vagy műholdas kapcsolatokra tervezték. Célja a magas RTT okozta hátrányok mérséklése, hagyományos TCP teljesítményének javítása. A TCP paramétereket a torlódási ablak dinamikájának vizsgálatával változtatja [14].

2.2.7 Hamilton TCP

Optimalizált torlódásszabályozó, csomagvesztés-alapú algoritmus nagysebességű, magas késleltetéssel rendelkező LFN (Long Fat Network) hálózatokhoz. Két működési módja van, hasonlóan, mint a Highspeed TCP-nek. Alacsonyabb sebességű környezetben hagyományos TCP-ként viselkedik, nagysebességű kapcsolatok esetén pedig egy hatékonyabb és rugalmasabb torlódásszabályozást hajt végre. Torlódáskor az ablakcsökkenés a körülfordulási időtől és a lefoglalt sávszélességtől függ. Az ablakméret-növelés pedig a torlódás óta eltelt időtől függ [15].

2.2.8 TCP Illinois

Nagy BDP értékkel rendelkező hálózatokhoz kifejlesztett variáns. Az ablakméret-változtatás irányának meghatározásához a csomagvesztést, a változtatás mértékének a sorbaállási késleltetést figyeli. A hagyományos AIMD algoritmus torlódásszabályzásánál használt konstans paraméterek itt az átlagos sorbaállási késleltetés függvényében változnak. Jobb átlagos átviteli teljesítményt biztosít a hagyományos TCP-nél [16].

2.2.9 TCP Low Priority

Osztott algoritmust alkalmaz, amelynek célja, hogy a többlet hálózati sávszélességet használja fel. Mechanizmusa csomagok késleltetését tartja számon a torlódás érzékelésére, és annak elkerülésére [17].

2.2.10 TCP Vegas

Tisztán késleltetés alapú protokoll. A késleltetés változása alapján próbálja megbecsülni a küldési rátát. Egy BaseRTT értéket határoz meg, ami azon körülfordulási idő, amikor még nincs torlódás a hálózaton. Teljesítménye nagymértékben függ attól, hogy milyen pontosan sikerül ezt az értéket megbecsülni. Ha a késleltetés csökken, nagyobbra veszi a küldési rátát, ha viszont nő, akkor csökkenti azt [18].

2.2.11 TCP Veno

A vezeték nélküli hálózatoknál jelentkező véletlenszerű csomagvesztés a TCP átviteli teljesítményt jelentősen lerontja. Ezt a problémát szándékozták kiküszöbölni a TCP Veno esetében [19].

2.2.12 Yeah TCP

Több TCP variáns teszteléséből szerzett tapasztalatok alapján hozták létre. Két eltérő üzemmódja van. A gyors üzemmódban egy STCP eljárást alkalmazó agresszív algoritmus

szerint növekszik a torlódási ablak. A lassú üzemmódban pedig a TCP Reno változathoz hasonlóan működik. A mért RTT értékek alapján becslést végez a sorban álló csomagok számának meghatározásához. Az alkalmazott üzemmódot ezen szám alapján választja meg [20].

Mivel egy TCP adatfolyam vizsgálatával foglalkozunk, ezért csak a bitrátára helyeződik a hangsúly a TCP-együtműködéssel (TCP-Fairness) szemben. A mérések során ezen okból csak átviteli sebességet és torlódási ablakméretet mértem.

3. Mérési eredmények

3.1 Alkalmazott matematikai eszközök

A statisztikai számításokhoz és grafikához az ehhez fejlesztett R project nevű programnyelvet és statisztikai programkörnyezetet használtam [21].

3.1.1 Tapasztalati eloszlás

Legyen $X^* = (X_1^*, X_2^*, \dots, X_n^*)$ n darab rendezett torlódási ablakméret minta

$$F_n^*(x) = \begin{cases} 0 & \text{ha } x \leq X_1^* \\ \frac{i}{n} & \text{ha } X_i^* \leq x \leq X_{i+1}^* \\ 1 & \text{ha } X_n^* \leq x \end{cases} \quad (7)$$

Az $F_n^*(x)$ lépcsős függvény lesz a becsült tapasztalati eloszlás. Az R `ecdf {stats}` függvényének segítségével határoztam meg az ablakméretek tapasztalati eloszlásfüggvényeit.

3.1.2 Exponenciális eloszlás

Egy ζ valószínűségi változót λ paraméterű exponenciális eloszlásúnak nevezünk, ha eloszlásfüggvénye:

$$F(x) = \begin{cases} 0, & \text{ha } x \leq 0 \\ 1 - e^{-\lambda x}, & \text{ha } x > 0 \end{cases} \quad \text{ahol } \lambda > 0 \text{ rögzített} \quad (8)$$

3.1.3 Maximum likelihood becslés

A módszer segítségével nyert becslés lényege, hogy azt a paramétert használjuk, amely esetén az aktuálisan bekövetkezett minta valószínűsége maximális. [22]
Tegyük fel egy Φ valószínűségi családról, hogy

$$A_0 : P_{\nu_1} \neq P_{\nu_2} \text{ ha } \nu_1 \neq \nu_2$$

$$A_\mu : \Phi \text{ dominált egy } \sigma\text{-véges } \mu \text{ mértékkel}$$

Jelöljük az eloszlást $p_\nu(x)$ -el, így a minta sűrűségfüggvénye:

$$p_{\nu_1}(x_1, \dots, x_n) = \prod_{i=1}^n p_{\nu_1}(x_i)$$

Maximum likelihood becslésnek nevezzük a t_n becslést, ha rögzített $(x_1, \dots, x_n)$ minta esetén

$$p_{t_n}(x_1, \dots, x_n) = \sup_{\nu} p_{\nu}(x_1, \dots, x_n) \quad (9)$$

Az R `fitdist` {`fitdistrplus`} függvényének segítségével határoztam meg az ablakméret tapasztalati eloszlásához illesztett exponenciális eloszlás λ paraméterét.

3.1.4 Általánosított Pareto eloszlás (GPD)

Kumulatív eloszlásfüggvénye:

$$GPD_{\xi, \alpha, \kappa}(x) = \begin{cases} 1 - \left(1 + \kappa \frac{x - \xi}{\alpha}\right)^{-\frac{1}{\kappa}}, & \text{ha } \kappa \neq 0 \\ 1 - \exp\left(-\frac{x - \xi}{\alpha}\right), & \text{ha } \kappa = 0 \end{cases} \quad (10)$$

ahol ξ : a helyzetmutató paraméter
 α : a skálaparaméter
 κ : az alakmutató paraméter

R-ben az `quagpa` {`lmomco`} függvény segítségével határoztam meg az adott tapasztalati eloszláshoz illesztett ξ , α , κ paraméterű, Pareto eloszlású eloszlásfüggvényt [23].

3.1.5 Közvetlen mintavételű L-momentum becslés

Az R `lmom.ub` `{lmomco}` függvénye segítségével határozhatóak meg $(x_1, \dots, x_n)$ minta esetén az L-momentumok [24]:

$$\begin{aligned}\lambda_1 &= \binom{n}{1}^{-1} \sum_{i=1}^n x_{(i)} \\ \lambda_2 &= \frac{1}{2} \binom{n}{2}^{-1} \sum_{i=1}^n \left\{ \binom{i-1}{1} - \binom{n-i}{1} \right\} x_{(i)} \\ \lambda_3 &= \frac{1}{3} \binom{n}{3}^{-1} \sum_{i=1}^n \left\{ \binom{i-1}{2} - 2 \binom{i-1}{1} \binom{n-i}{1} + \binom{n-i}{2} \right\} x_{(i)} \\ \lambda_4 &= \frac{1}{4} \binom{n}{4}^{-1} \sum_{i=1}^n \left\{ \binom{i-1}{3} - 3 \binom{i-1}{2} \binom{n-i}{1} + 3 \binom{i-1}{1} \binom{n-i}{2} - \binom{n-i}{3} \right\} x_{(i)}\end{aligned}\tag{11}$$

$$\text{Ha } \tau_r = \frac{\lambda_r}{\lambda_2}, \quad r = 3, 4, \dots\tag{12}$$

akkor τ_r értéket skálázott r-edik L-momentumnak nevezzük.

Az általánosított Pareto eloszlás megfelelő ξ , α , κ paramétereit az R `pargpa` `{lmomco}` függvénye segítségével határoztam meg, ami egy adatsor L-momentumai segítségével végzi el a paraméterbecslést a következő összefüggéseket kihasználva [25]:

$$\begin{aligned}\lambda_1 &= \zeta + \frac{\alpha}{1 + \kappa} & \lambda_2 &= \frac{\alpha}{(1 + \kappa)(2 + \kappa)} \\ \tau_3 &= \frac{1 - \kappa}{3 + \kappa} & \tau_4 &= \frac{(1 - \kappa)(2 - \kappa)}{(3 + \kappa)(4 + \kappa)}\end{aligned}\tag{13}$$

A (13) egyenletekből ξ , α , κ paraméterek a következőképpen számolhatóak ki

$$\kappa = \frac{1 - 3\tau_3}{1 + \tau_3} \quad \alpha = (1 + \kappa)(2 + \kappa)\lambda_2 \quad \xi = \lambda_1 - (2 + \kappa)\lambda_2 \quad (14)$$

3.1.6 Anderson-Darling teszt

Mintaelemekre alkalmazható illeszkedésvizsgálattal foglalkozó (Goodness of fit) statisztikai teszt, mellyel meghatározható, hogy a minta adatsor nevezetes eloszlásból származik-e.

Tegyük fel, hogy $x_1 \leq \dots \leq x_n$ egy $F_R(x)$ tapasztalati eloszlásfüggvényből vett n darab független rendezett minta. A teszt a következő null hipotézis vizsgálatán alapszik:

$$H_0 : F_R(x) = F(x, \Theta),$$

ahol:

F : egy nevezetes eloszlásfüggvény,

Θ : egy a megfelelő paramétereket tartalmazó vektor.

A teszt statisztikája egy A értéket határoz meg (15) képlet alapján.

$$A^2 = -n - \frac{1}{n} \sum_{i=1}^n [(2i-1) \ln[F(x_i, \Theta)] + (2n+1-2i) \ln[1-F(x_i, \Theta)]] \quad (15)$$

A kritikus érték az Anderson-Darling teszt esetében a vizsgált nevezetes eloszlástól függ.

Ezen értéknél, ha nagyobb a (15) egyenlet során kapott A érték, a null hipotézist el kell vetni [26].

A tesztet az R `gofGENPARTest {nsRFA}` függvénye segítségével végeztem el, ami eredményül a teszt A^2 értékét, és P valószínűségét adja, például, ha $P(A^2) = 0,9$ a vizsgált minta 10%-ban nem illeszkedett a megfelelő paraméterű nevezetes Pareto eloszlásfüggvényhez (10).

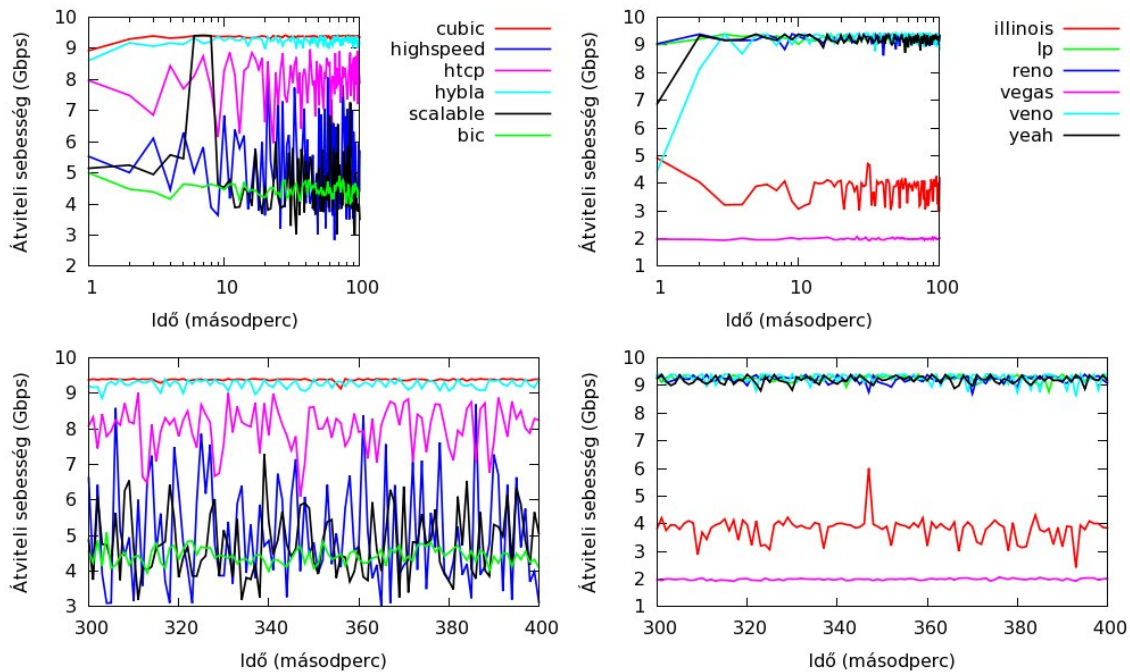
3.1.7 Említett matematikai eszközök konkrét megvalósítása

A torlódásvezérlő algoritmusok ablakméret-növelő függvényeinek jellemzésére a legjobb eszköz, ha az eloszlásfüggvényét (ECDF) sikerül meghatározni. Az, hogy mennyire skálázható egy variáns, az eloszlásfüggvényből képzett komplementer függvény (CCDF) lecsengésének vizsgálatával dönthető el, azaz rövid lecsengésű (sort-tail) vagy hosszú lecsengésű (long-tail) lesz. Ezt a vizsgálatot úgy lehet elvégezni, hogy megpróbáljuk illeszteni a CCDF függvényeket megfelelő paraméterbecsléssel (9) az exponenciális eloszlásra (8) mint rövid lecsengésűre, illetve a Pareto eloszlásra (10) mint klasszikus hosszú lecsengésűre. A nagy elemszám miatt az illeszkedés statisztikai igazolására a Kolmogorov-Smirnov helyett az Anderson-Darling tesztet végeztem el (*3.1.6 fejezet*). A vizsgálatról azt vártam, hogy bizonyos variánsok rövid lecsengésű eloszlást produkálnak az ablaknövelés tekintetében, míg mások hosszú lecsengésűt. Ez bizonyítja, hogy mennyire volt jó dinamikája az ablakméret-növelő függvénynek, ami jól fog látszani ezekből az eloszlásfüggvényekből.

3.2 LAN mérések

3.2.1 Átviteli sebesség

A tízperces mérések elvégzése után a másodpercenként mintavételezett átviteli sebesség értékeket grafikonon ábrázoltam. A TCP variánsokat két csoportra osztottam a jól láthatóság érdekében. Az első oszlopba a nagy sebességű, a másikba az egyéb, *2.2 fejezetben* bemutatott TCP torlódásvezérlő algoritmusok kerültek. Az első ábrázolásnál logaritmikus skálázású 1-től 100 másodpercig tartó időtengelyt alkalmaztam, így láthatóvá váltak az esetleges felfutási sebességek. A másik ábrázolásnál lineáris időskálázás valósult meg 300-tól 400 másodpercig tartó intervallumban. Az átviteli sebességértékekre alkalmazva a maximum, a minimum, az átlag és a tapasztalati szórás leíró statisztikák eredményeit táblázat segítségével szemléltettem. Ezeket a statisztikai függvényeket a mérés tizedik másodpercétől kezdve alkalmaztam, így az esetleges felfutási idők okozta alacsony értékek elkerülésével úgynevezett steady-state állapotot elemeztem segítségükkel.

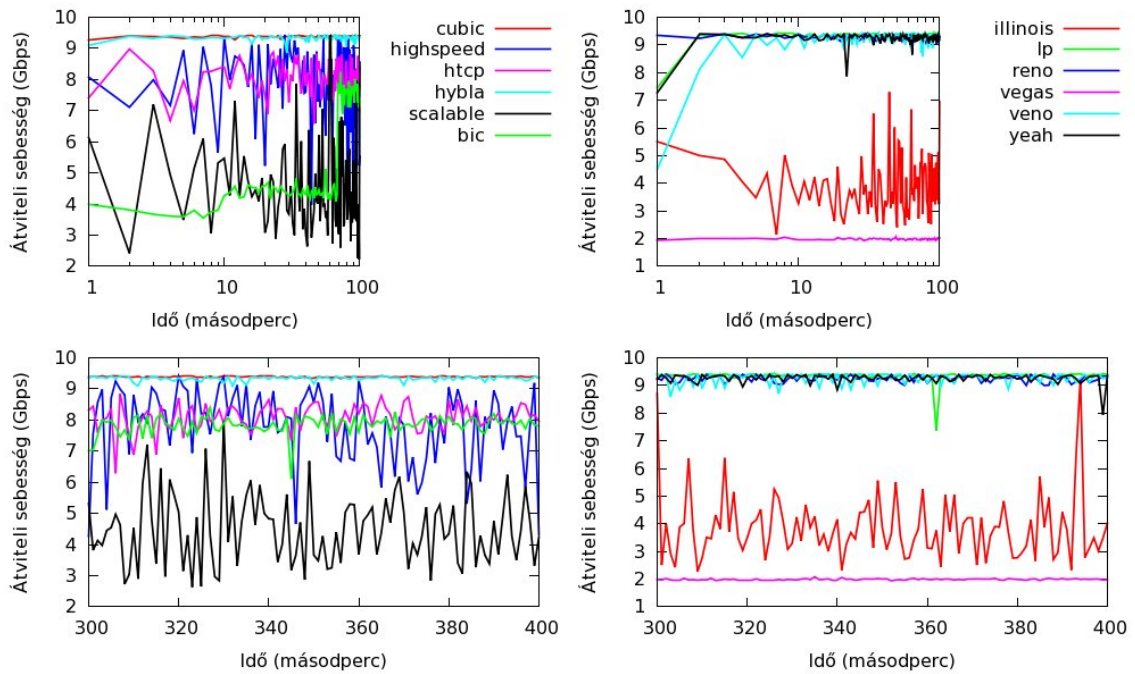


1. ábra: Átviteli sebességek LAN környezetben alapértelmezett pufferméretekkel

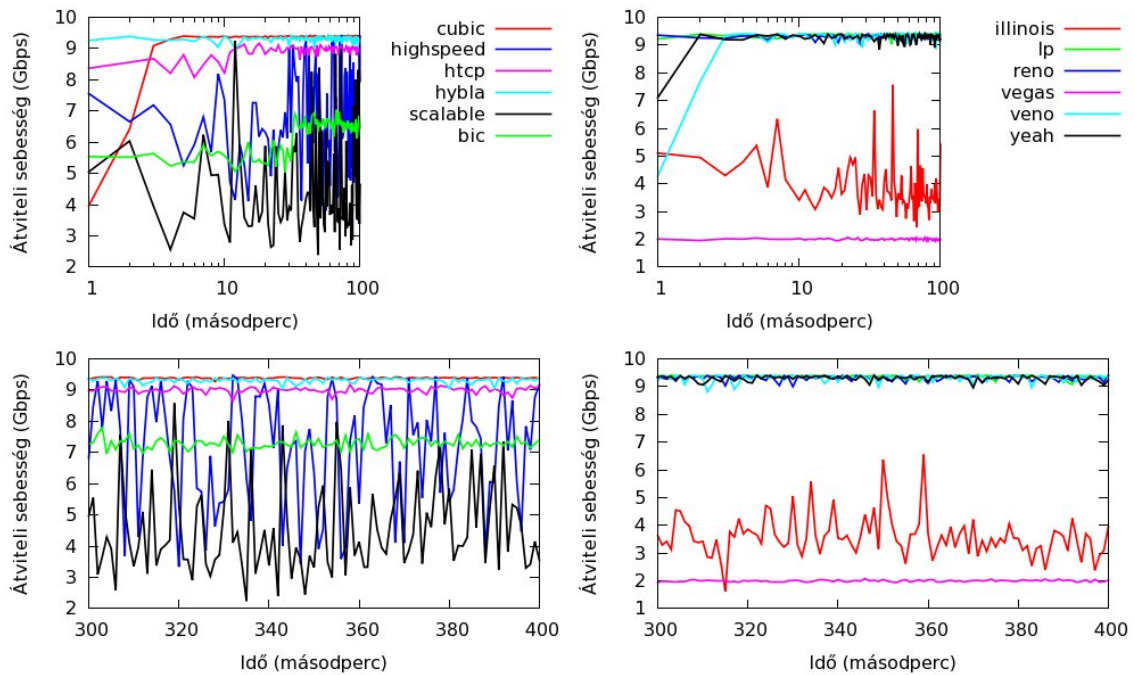
Az alacsony pufferméretek ellenére a CUBIC és a Hybla ideálisan viselkedett a nagysebességű variánsok közül. Magas átlagot értek el alacsony szórással. Viszont a BIC bináris keresőalgorithmusa nem volt hatékony. A HTCP variáns is elfogadható átlagértéket ért el, de nagy ingadozással. A Highspeed és a Scalable variánsok esetében az átviteli bitráta átlagok a maximálisan elérhetőnek csak az 50%-át érték el, nagy szórással. A Reno - mint klasszikus TCP torlódásvezérlő algoritmus - is megfelelően működött. Az alacsony késleltetés miatt nem történt csomagvesztés. Az LP variáns használata hasonlóan jó eredményeket eredményezett. A Veno és a Yeah variánsok is - bár hosszabb felfutási idővel - ideális átviteli sebességet biztosítottak.

variáns	min	max	átlag	szórás
bic	3,95	5,09	4,432119	0,2002132
cubic	8,92	9,42	9,380034	0,03736207
highspeed	2,24	8,7	5,124856	1,355026
htcp	4,41	9,08	8,029679	0,6513033
hybla	8,62	9,42	9,251627	0,1157381
illinois	2,33	7,26	3,752318	0,4486007
lp	8,76	9,41	9,252183	0,107641
reno	8,6	9,41	9,241542	0,1241822
scalable	2,93	8,91	4,674247	0,9425185
vegas	1,9	2,07	1,994129	0,02730016
veno	8,32	9,42	9,265601	0,1802097
yeah	8,26	9,4	9,198409	0,1454199

4. táblázat: Bitráta statisztikák (Gbit/s) LAN környezetben alapértelmezett méretű pufferekkel



2. ábra Átviteli sebességek LAN környezetben nagy méretű pufferekkel



3. ábra Átviteli sebességek LAN környezetben $B \times D$ szerint hangolt pufferekkel

	Nagyméretű pufferek				B×D szerint hangolt pufferek			
variáns	min	max	átlag	szórás	min	max	átlag	szórás
bic	4,09	8,4	7,425854	1,068737	2,15	8,19	7,038528	0,5273211
cubic	9,2	9,42	9,383824	0,02459701	9,2	9,42	9,383621	0,02435789
highspeed	2,72	9,44	7,84308	1,237851	2,72	9,48	7,057377	1,818459
htcp	5,29	9,05	7,98824	0,6092924	8,22	9,18	8,982538	0,1068215
hybla	4,33	9,43	9,318799	0,2260923	8,29	9,41	9,307073	0,09317048
illinois	1,42	9,45	4,070169	1,454312	1,62	8,08	3,607851	0,7524241
lp	7,37	9,44	9,369746	0,09307608	8,99	9,43	9,358663	0,05601217
reno	8,62	9,41	9,262927	0,1102696	8,74	9,42	9,307479	0,08805375
scalable	1,63	9,42	4,492555	1,357594	0	9,31	4,53956	1,450391
vegas	1,92	2,1	1,988257	0,02632602	1,93	2,08	2,001929	0,02817022
veno	8,25	9,42	9,261286	0,19093	8,4	9,42	9,319323	0,1285323
yeah	7,87	9,41	9,252284	0,1563103	7,94	9,42	9,290897	0,1200185

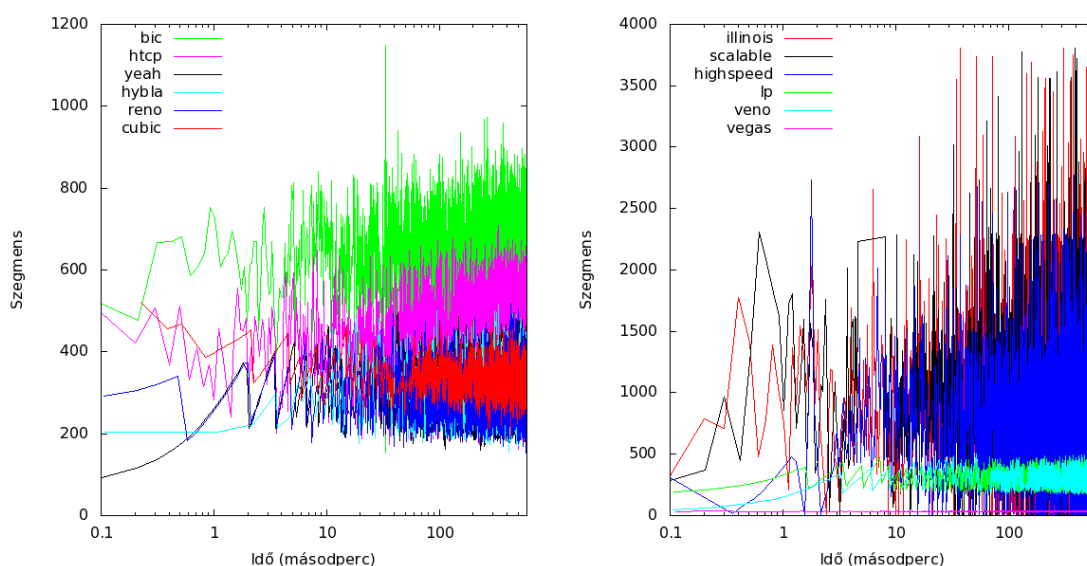
5. táblázat: Bitráta statisztikák (Gbit/s) LAN környezetben nagyméretű és $B \times D$ szerint hangolt pufferekkel

Az extrém nagyméretű puffer beállításokkal a BIC már magasabb átlagos átviteli sebességet tudott elérni a mérés során, de a szórása is megnőtt. Az alapértelmezett kernel paraméterek mellett hatékony variánsok szintén jó eredményt produkáltak. A Highspeed átviteli karakterisztikája is jelentősen javult. Átlagos átviteli sebessége szinte megegyezik a HTCP-vel, a szórása viszont kétszerese. A Scalable és az Illinois továbbra is gyenge hatékonysággal teljesítettek.

A beállított nagyméretű pufferekhez képest nagyságrenddel kisebb puffermemóriát használ a $B \times D$ szerint hangolt eset (5. táblázat). Ennek ellenére hasonlóan jó átviteli sebesség statisztikákkal rendelkeztek a mérési eredmények. A nagysebességű TCP variánsok esetében ismét a CUBIC nyújtotta a legjobb eredményeket annak ellenére, hogy viszonylag alacsony bitráta értékkel indította a TCP kapcsolatot. A Hybla átviteli teljesítménye is megmaradt, sőt az értékek szórása jelentősen csökkent. A HTCP által vezérelt kapcsolat karakterisztikája is javult átlag és szórás esetén is, viszont a Highspeed esetében mindkettő romlott. A Scalable torlódásvezérlő algoritmus hatékonysága továbbra is gyenge maradt. A további variánsok esetében az LP, a Reno, a Veno és a Yeah ugyanolyan mértékű átviteli sebességű kommunikációt eredményezett, mint nagyméretű pufferek esetén, sőt az értékek szórása mindenhol csökkent. Az Illinois és Vegas teljesítménye továbbra is gyenge maradt.

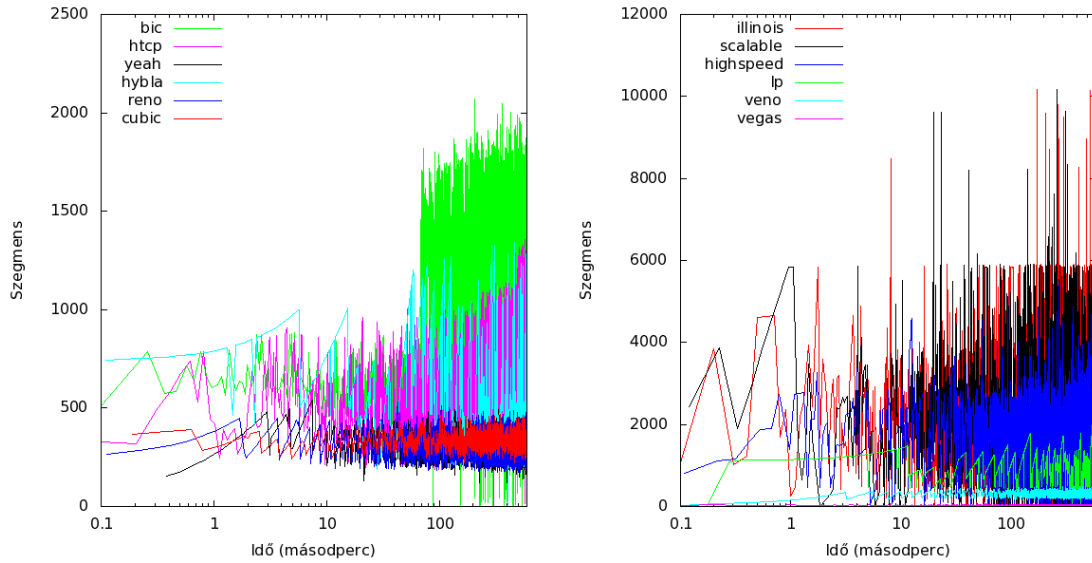
3.2.2 Torlódási ablakméret

A *tcpprobe* modul kimenetén akkor jelennek meg értékek, amikor változás történik valamely figyelt paraméternél. Ezért 10 Gbit/s linken gigabyte méretű is lehet a letárolt torlódási ablakméret érték, aminek feldolgozása problémákat eredményez. Ezen okból a rajzoló programba egy *awk* scriptet ágyaztam be, mely segítségével csak azokat az értékeket rajzoltattam ki, amelynek az időbélyege az előző eleméhez képest legalább 0,1 másodperccel nagyobb. Így az ábrázoláshoz megfelelő mennyiségű adat maradt, és a rajzolásnál sem okozott fennakadást. Logaritmikus időskálán ábrázoltam a szegmensméret változásokat. A TCP variánsokat szintén két részletre osztottam a jobb szemléltethetőség érdekében.

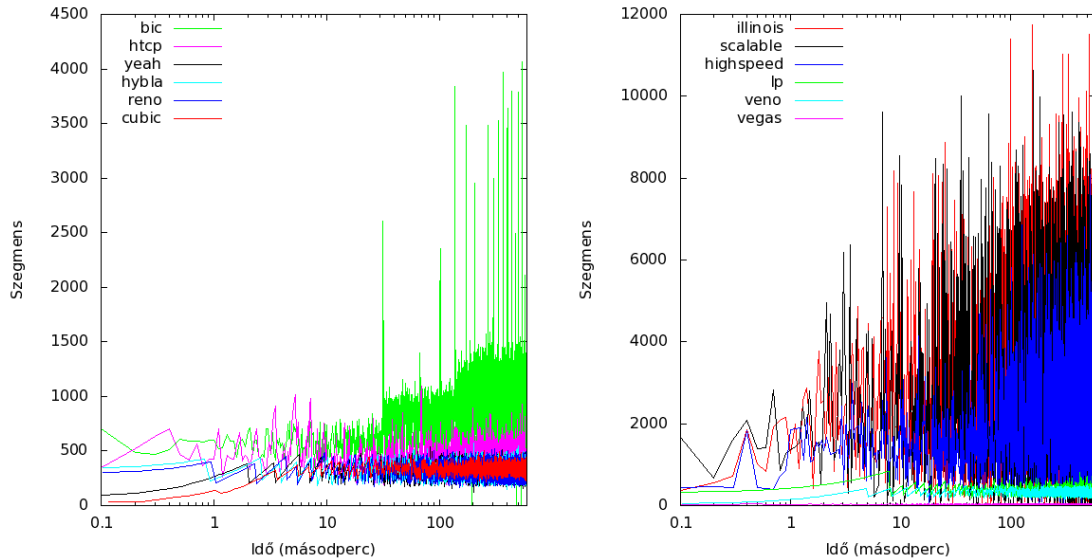


4. ábra: Ablakméretek LAN környezetben alapértelmezett méretű pufferekkel

A jó átviteli sebességet eredményező CUBIC, Hybla, Reno, LP, Veno, és Yeah algoritmusok hasonlóan 200 és 500 szegmensméret között ingadoztak. Nagyobb ablakméreteket ért el a BIC és a HTCP. Legnagyobb ingadozást a rossz átviteli sebesség értékeket eredményező Illinois és a Scalable ért el. 3 500 szegmens fölötti adat küldése ment végbe TCP nyugta fogadása nélkül. A Vegas algoritmus nem tudott felskálázni, így 30 szegmens méretnél maradt az ablakméret.



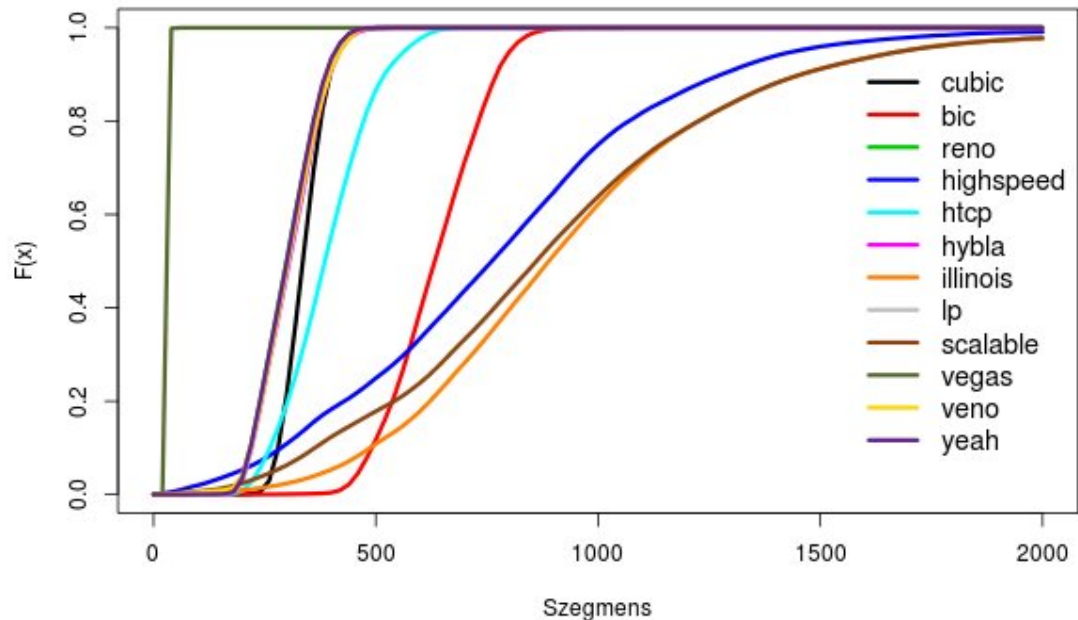
5. ábra: Ablakméretek LAN környezetben nagyméretű pufferekkel



6. ábra: Ablakméretek LAN környezetben $B \times D$ szerint hangolt pufferekkel

A puffermemóriák extrém nagy méretre növelésével azon variánsok, melyek alapértelmezett esetben alacsony szegmensméreteket eredményeztek, szintén nem voltak a kommunikáció során 500 fölötti értékek. A BIC - mely a hangolásnak köszönhetően jelentősebb átviteli sebességgel rendelkezett - ablakméretben is jóval nagyobb értékeket ért el. $B \times D$ szerinti hangolással jelentkeztek extrém nagy értékek is az átlaghoz képest. Az Illinois, a Scalable és a Highspeed esetében jelentősen nagyobb értékek is megjelentek az alapértelmezett esethez képest.

A torlódási ablakméret értékekből (7) egyenlet alapján képzett tapasztalati eloszlásfüggvények jól szemléltetik a variánsok közötti különbségeket, és a további statisztikai elemzéseknek alapjául is szolgálnak.



7. ábra: Ablakméretek eloszlásfüggvényei alapértelmezett méretű pufferek esetében

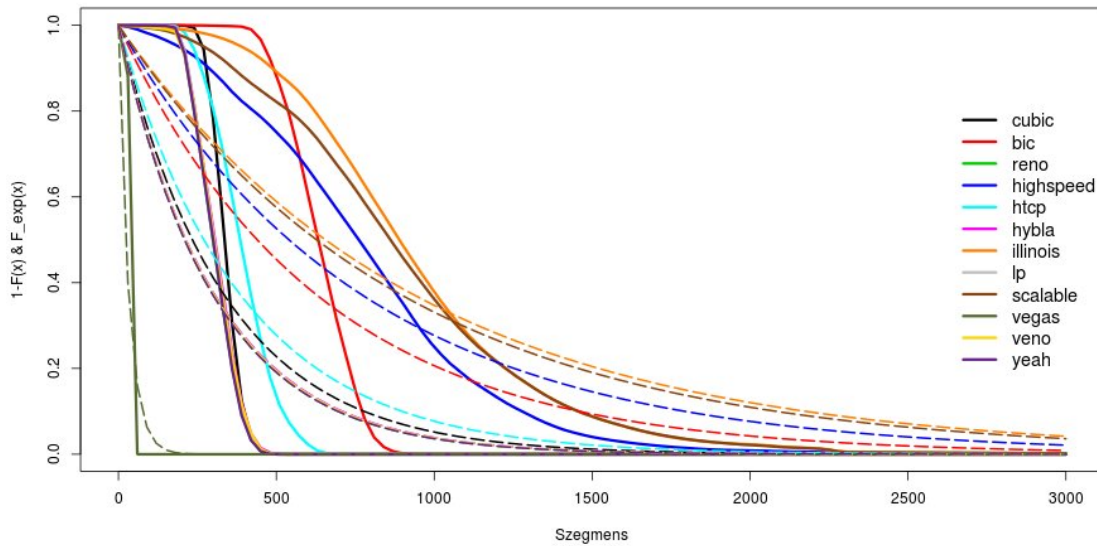
Ha ezen eloszlásfüggvények komplementereit ábrázoljuk, lecsengés szempontjából további vizsgálatokat lehet elvégezni annak kiderítése érdekében, hogy az adott függvény hosszú

lecsengésű (long-tail), vagy rövid lecsengésű (short-tail). Tehát annak a valószínűsége, hogy az adott variáns által vezérelt TCP kapcsolat során jelentkeznek extrém nagy ablakméretek, nulla vagy sem. Maximum likelihood becslés segítségével (9) meghatározható egy exponenciális eloszlás λ paramétere (8). Ha ezen λ paraméterű exponenciális eloszlásfüggvényre illeszkedik az adott tapasztalati eloszlásfüggvényünk, akkor rövid lecsengésű lesz a görbe. Megbecsülhető (11) egy ξ , α , κ paraméterű Pareto eloszlás (10) is, ami illeszkedés esetén

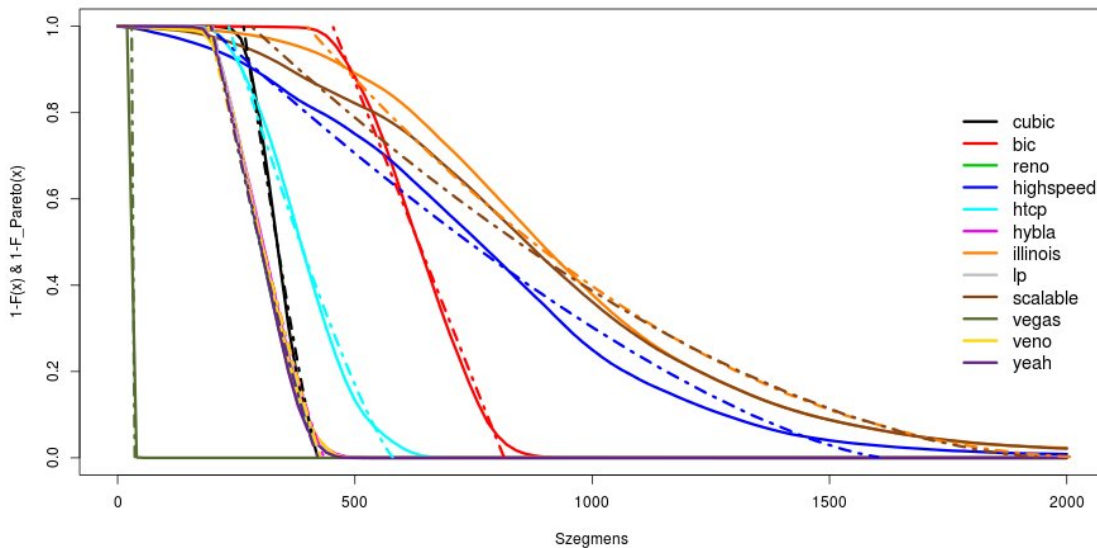
	Alapértelmezett		Max. méretű		B×D szerint hangolt	
	A^2	P	A^2	P	A^2	P
bic	6518	1	1137	1	19490	1
cubic	465	1	435.8	1	870.8	1
highspeed	9148	1	3075	1	16890	1
hybla	660.4	1	118.1	1	1668	1
htcp	11970	1	3850	1	22190	1
illinois	14770	1	7087	1	17050	1
lp	665.4	1	351.4	1	2417	1
reno	646.6	1	501.2	1	1866	1
scalable	14760	1	11420	1	27570	1
vegas	8132	1	8861	1	16310	1
veno	421.1	1	338	1	1051	1
yeah	686.1	1	692.8	1	1925	1

6. táblázat: Anderson-Darling teszt A^2 értékei és azok valószínűségei a variánsok Pareto-ra illeszkedésének vizsgálata során LAN környezetben

hosszú lecsengést igazol.

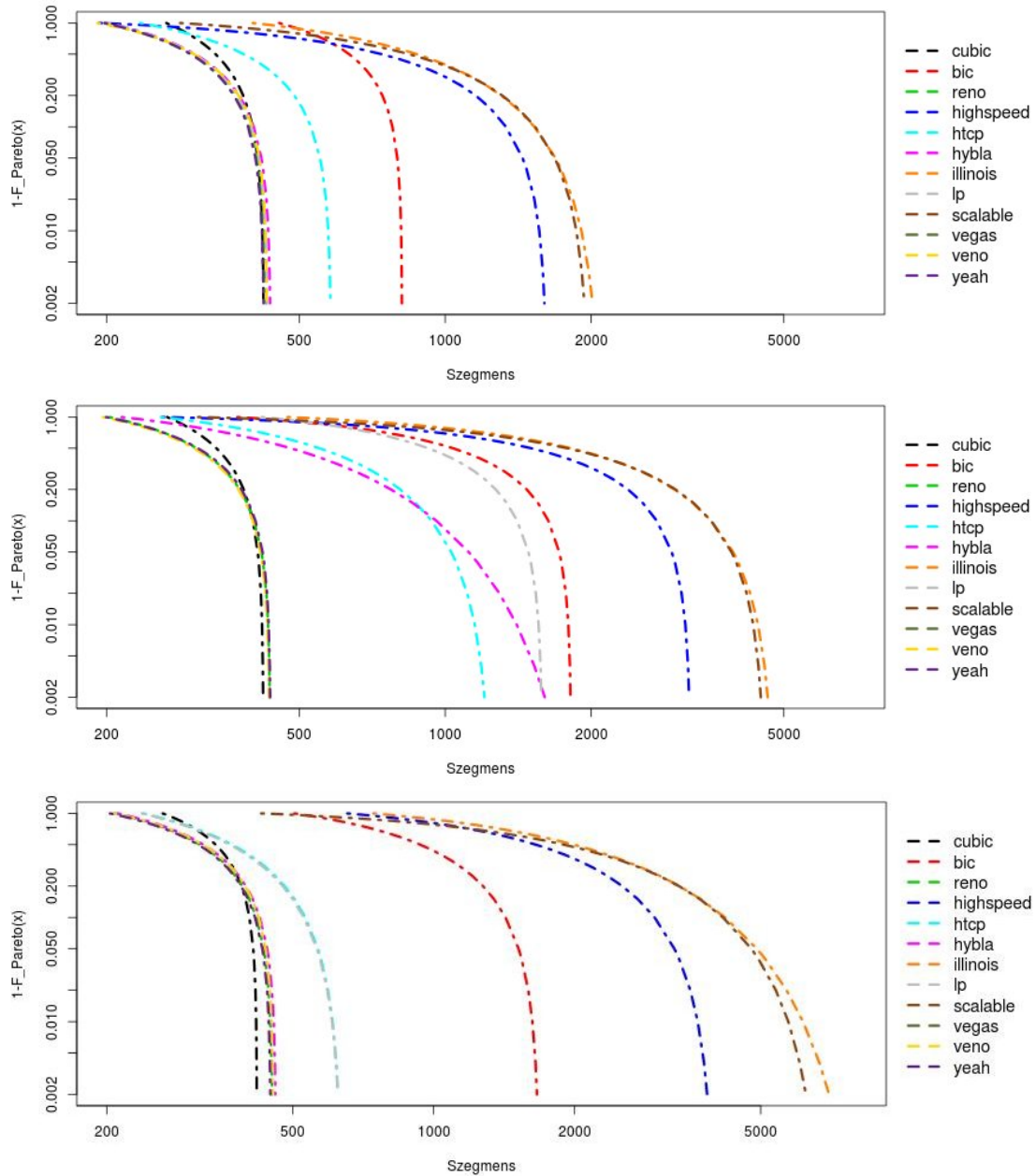


8. ábra: Komplementer ablakméret eloszlásfüggvények és az illesztett exponenciális eloszlásfüggvények alapértelmezett méretű puffer memória méretekkel



9. ábra: Komplementer ablakméret eloszlásfüggvények és az illesztett Pareto eloszlásfüggvények alapértelmezett méretű puffer memória méretekkel

Az illesztések során egyértelműen kiderült, hogy a variánsok a Pareto görbéhez igazodnak, ami long-tail tulajdonságukat igazolja. Ezt a 6. táblázat támasztja alá, ugyanis a 3.1.6 fejezetben tárgyalt Anderson-Darling illeszkedésvizsgálatot elvégző `gofGENPARTest` R függvény P értéke az összes variáns és puffermemória méret kombinációjában 1 volt, ami igazolja a null hipotézist az összes esetben.



10. ábra: Illesztett Pareto eloszlás görbék logaritmusos idő és szegmensméret skálán alapértelmezett, nagyméretű és $B \times D$ szerint hangolt puffér esetekben.

A logaritmusos skálán ábrázolt Pareto görbéken jól látható, hogy mennyire gátat tudnak szabni az ablakméret növekedésének a korlátozott puffermemóriák. Alapértelmezett méretű puffereknél az Illinois és a Scalable teljesített a legjobban, 1% alatti valószínűséggel, de használtak nagyobb ablakméretet 2 000 szegmensnél a kapcsolat során. A Highspeed is kellően nagy méreteket eredményezett. Az illesztett eloszlásfüggvények paramétereit a 7. táblázat tartalmazza

A pufferek nagy méretre növelésének eredményeként számos torlódásvezérlő algoritmusnál növekedett jelentősen a nagyobb ablakméretek valószínűsége. A BIC, a HTCP, a Hybla és az LP 5%-os valószínűséggel érték el 1 000 szegmens fölötti ablakméretet, holott hangolt esetben egyiknek sem sikerült. Az alapértelmezett esetben jól viselkedő három variáns ablakméret értékei jelentősen megnövekedtek: 50%-ban volt 2 000 szegmens fölötti az adatátvitel során.

A $B \times D$ szerinti puffer memória hangolás eredményeként a HTCP, a Hybla és az LP nagy ablakméret értékei eltűntek, és a hangolatlan esetben produkált méreteket eredményezték. A BIC és a Highspeed variánsokra nem volt hatással, a nagyméretű pufferek esetéhez képest ugyanazt a teljesítményt eredményezték. Az Illinois és a Scalable algoritmusok esetében a $B \times D$ hangolásnak köszönhetően jelentek meg 1%-os valószínűséggel extrém nagy ablakméretek.

3.2.3 Következtetések

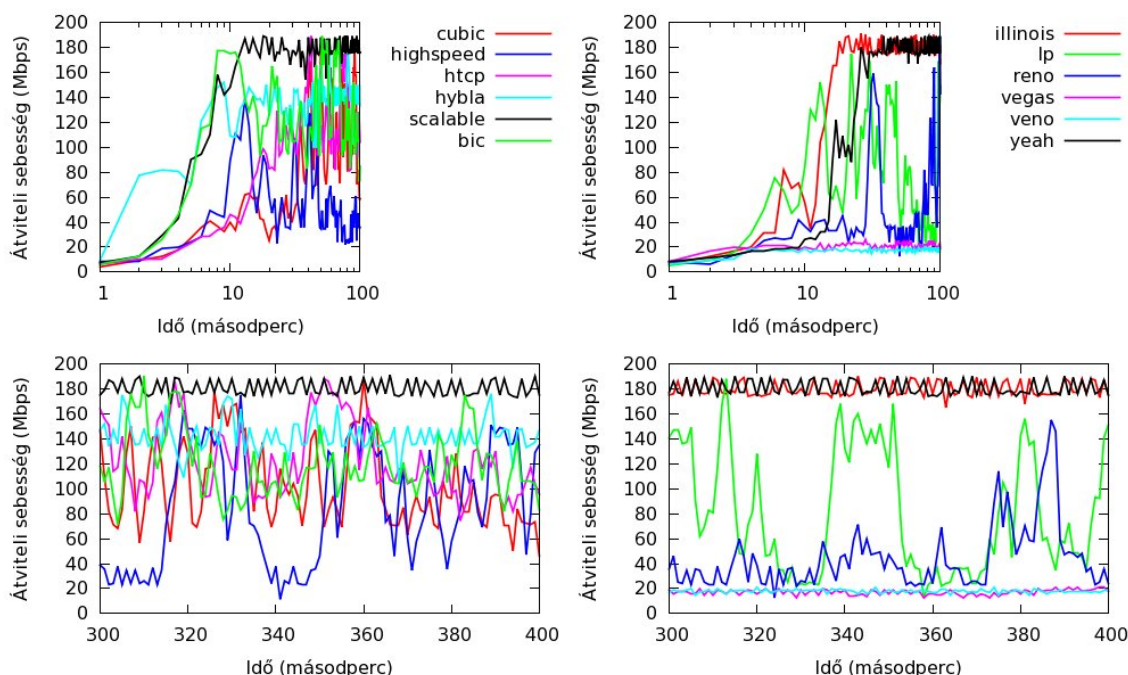
A LAN mérések során volt olyan TCP torlódásvezérlő algoritmus, melyek a rendelkezésre álló linkkapacitáshoz közeli teljesítményt nyújtottak már alapértelmezett puffer memóriákkal is, így a hangolás nem volt rájuk hatással. Ezek a variánsok a Hybla, a Reno, a Veno, a Yeah voltak, és végig kiemelkedő teljesítményt nyújtott a CUBIC átviteli sebességben. A Reno nem skálázható, de ebben az esetben nem is fontos, ugyanis nagyon kis mértékű volt a késleltetés, és ezért nagyobb ablakméretekre sincs szükség. Az alapértelmezett esetben rosszul viselkedő BIC variáns a hangolásnak köszönhetően jelentős átviteli teljesítményjavulást ért el, és ablakméretben is növekedést eredményezett. Az ablakméret tapasztalati eloszlásfüggvényeit sikerült illeszteni egy adott ξ , α , κ paraméterű Pareto eloszlásfüggvényre. A $B \times D$ szerint hangolt puffermemóriák esetén jelentkeztek a legnagyobb torlódási ablakméretek a mérések során. Az Illinois torlódásvezérlő algoritmus esetén 14 500 szegmens méretű adat került küldésre nyugtázatlanul, ami 21,75 Mbyte-nak felel meg, ha 1 500 byte-os szegmenseket veszünk az alapértelmezett MTU miatt. De az Illinois és a Scalable sem tudott elfogadható átviteli sebességet produkálni. A Highspeed variáns által vezérelt kapcsolat ért el elfogadható sebességet és magas ablakméret-eredményeket a kernel paraméterek hangolását követően. Összességében a CUBIC torlódásvezérlő algoritmus mondható a leghatékonyabbnak - függetlenül a puffer memória méretektől - közvetlen összeköttetésű LAN kapcsolatok esetében.

	λ	ξ	$\alpha,$	κ	
bic	0,001577975	453,5228	359,6605	0,9958875	Alapértelmezett
cubic	0,002959815	265,3918	134,537	0,8565238	
highspeed	0,001283808	191,0537	998,2384	0,6980334	
htcp	0,002563067	233,8797	283,7931	0,8159529	
hybla	0,003246248	199,7177	199,9502	0,8457461	
illinois	0,001057889	399,067	803,2324	0,4705502	
lp	0,003265029	200,4291	196,0936	0,8526163	
reno	0,003308682	197,9439	190,6694	0,8282401	
scalable	0,001105169	281,6569	982,8529	0,5771512	
vegas	0,03061667	29,6109	5,556955	0,8213306	
veno	0,003278862	190,9675	219,1132	0,9217711	
yeah	0,003312939	197,5441	194,1268	0,8611873	
	λ	ξ	$\alpha,$	κ	
bic	0,0009498322	369,6092	1292,246	0,8914378	Nagyméretű
cubic	0,002950387	266,4563	136,1002	0,8777026	
highspeed	0,0006401304	255,5034	2344,415	0,7941794	
htcp	0,001695119	258,1511	498,9446	0,5038504	
hybla	0,001810023	214,114	413,9066	0,2232548	
illinois	0,0005043252	467,6225	2336,601	0,5420818	
lp	0,001061174	415,9637	956,9861	0,8180212	
reno	0,003225748	202,4117	199,8038	0,8570182	
scalable	0,0005177761	303,7083	2630,12	0,6159217	
vegas	0,03065641	29,5322	5,583843	0,8085846	
veno	0,003266908	196,1123	204,5798	0,8600279	
yeah	0,003215001	199,0313	212,4297	0,8965131	
	λ	$\xi,$	$\alpha,$	κ	
bic	0,001029287	502,1479	779,8453	0,6613719	B×D szerint hangolt
cubic	0,002928128	263,0323	158,1996	1,015723	
highspeed	0,000560436	649,9536	1722,223	0,518218	
htcp	0,002598561	238,8664	231,1629	0,5837214	
hybla	0,003074425	210,5608	212,4456	0,8521316	
illinois	0,000431652	738,7499	1965,675	0,2457295	
lp	0,002622718	237,2628	225,4461	0,5653696	
reno	0,003126172	207,5074	207,67	0,8480479	
scalable	0,0004582547	422,7925	2418,959	0,3748771	
vegas	0,03047701	29,58356	5,663	0,7543059	
veno	0,003091965	206,2178	222,5398	0,8987859	
yeah	0,003156246	202,7131	212,9399	0,8659467	

7. táblázat: Az illesztett exponenciális és Pareto eloszlások paraméterei LAN kapcsolat esetében

3.3 WAN mérések

3.3.1 Átviteli sebesség

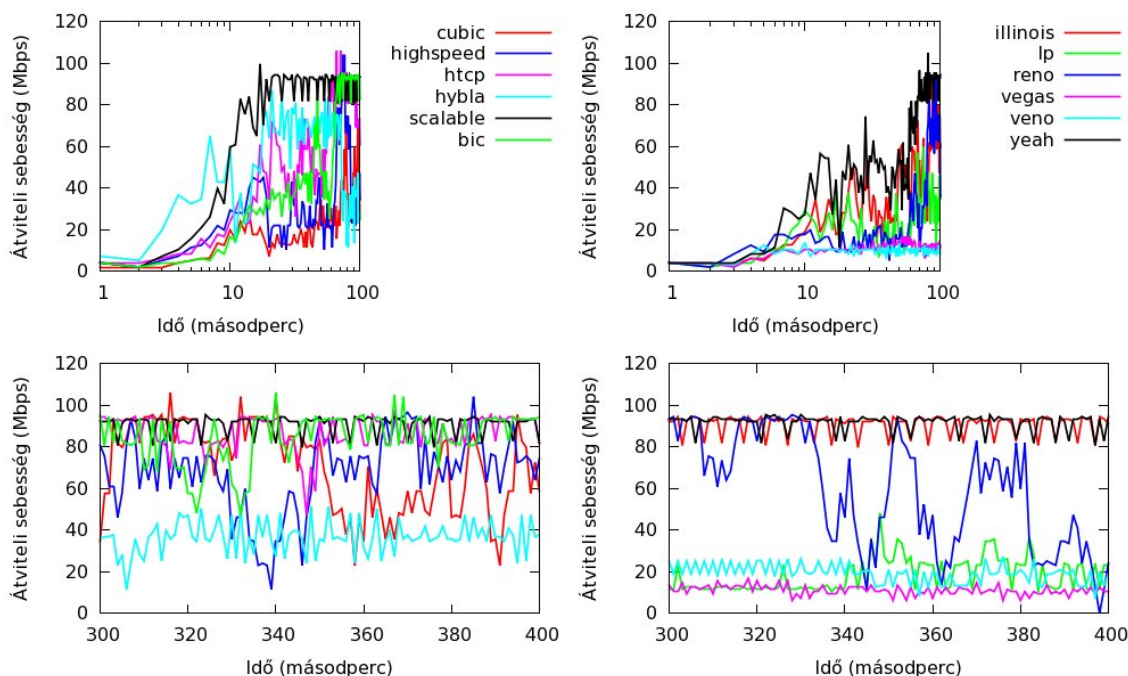


11. ábra: Átviteli sebességek 90 ms késleltetéssel alapértelmezett méretű pufferekkel

Az átviteli sebességek a nagy késleltetés miatt jelentősen lecsökkentek a maximálisához képest. A Scalable és az Illinois variánsok - melyek a közvetlen méréseknél alacsony átviteli sebességet tudtak csak elérni - a legmagasabb átlagsebességet érték el alacsony szórással. Hasonlóan jól működött a Yeah algoritmus is. Mindhárom variáns - egy bizonyos felfutási idő után - egy állandó érték körül ingadozott alacsony szórással. A BIC, a CUBIC és a HTCP variánsok tudtak még elfogadható átviteli sebesség értékeket produkálni, de nagyobb szórással. A Reno - bár ért el magas bitráta értékeket - átlagos átviteli teljesítménye ugyanakkor gyengének bizonyult.

variáns	min	max	átlag	szórás
bic	71,3	191	124,3328	26,96616
cubic	25,2	190	103,3716	34,9088
highspeed	10,5	175	63,08156	39,6526
htcp	38,8	190	126,5293	30,67839
hybla	75,5	191	142,1887	14,56371
illinois	34,6	192	178,942	12,02731
lp	12,6	188	76,50525	44,15128
reno	11,5	189	56,06633	37,69739
scalable	148	193	179,8832	6,637968
vegas	12,6	27,3	19,28003	2,550568
veno	14,7	25,2	18,03469	1,46787
yeah	26,2	192	177,3812	18,63318

8. táblázat: Bitráta statisztikák (Mbit/s) 90 ms késleltetésű WAN környezetben alapértelmezett méretű pufferekkel

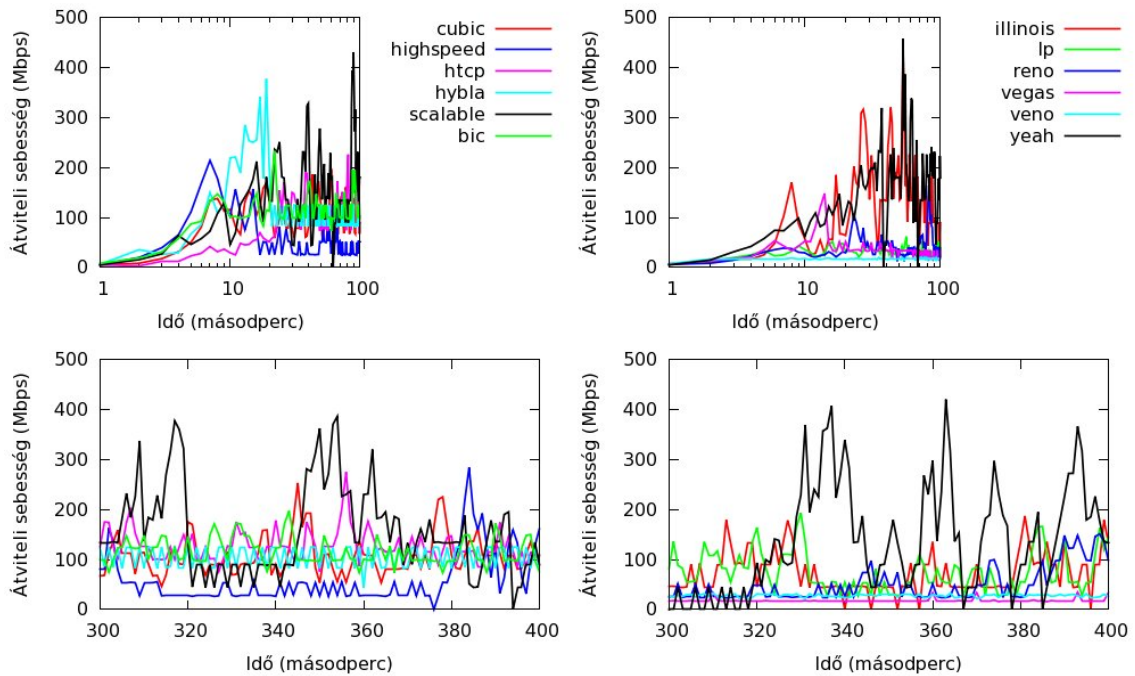


12. ábra: Átviteli sebességek 180 ms késleltetéssel alapértelmezett méretű pufferekkel

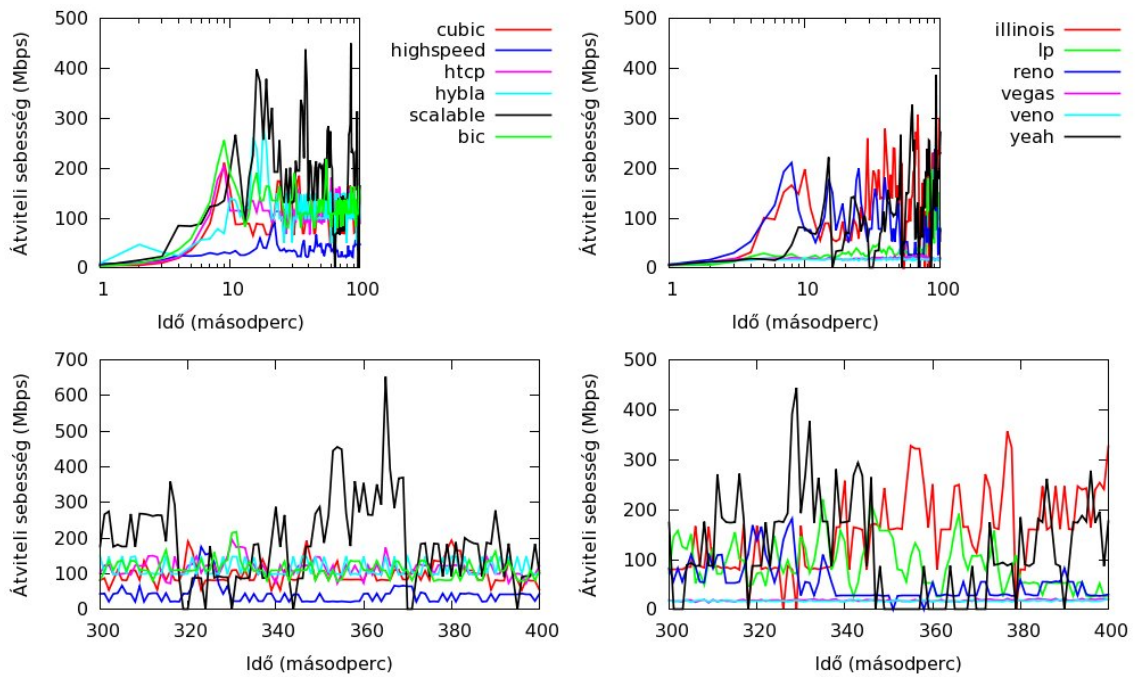
180 ms-os késleltetés esetén az átviteli sebességértékek tovább csökkentek. Mivel a késleltetés a kétszeresére nőtt, a legtöbb variánsnál feleződtek az értékek a 90 ms-os késleltetés során kapott eredményekhez képest. Továbbra is a Scalable érte el a legjobb átlagos értékeket alacsony szórással. A 12. ábrán jól látszik a TCP Reno AIMD algoritmus (4) ablakcsökken-tő mechanizmusának hatása az átviteli sebességre. A mérés során tapasztalt maximális értéket is elérte, de szórás szempontjából a legnagyobbak közé került, ugyanis a csomagvesztést követő meredek esés miatt nehezen tudott ismét magasabb bitráta tartományba kerülni.

variáns	min	max	átlag	szórás
bic	14,7	106	77,22369	18,85213
cubic	7,34	106	47,60159	24,6663
highspeed	0	105	45,97107	26,71731
htcp	23,1	106	83,35533	14,53618
hybla	11,5	89,1	39,76091	12,65516
illinois	12,6	103	82,41235	19,07352
lp	0	95,4	21,74426	11,83121
reno	0	106	55,46538	26,02785
scalable	59,8	105	90,72064	5,294597
vegas	6,29	16,8	10,99345	2,185419
veno	6,29	50,3	16,60154	6,186024
yeah	11,5	105	87,78054	13,78362

9. táblázat: Bitráta statisztikák (Mbps) 180 ms késleltetésű WAN környezetben alapértelmezett méretű pufferekkel



13. ábra: Átviteli sebességek 90 ms késleltetésnél nagyméretű pufferekkel



14. ábra: Átviteli sebességek 90 ms késleltetésnél $B \times D$ szerint hangolt pufferekkel

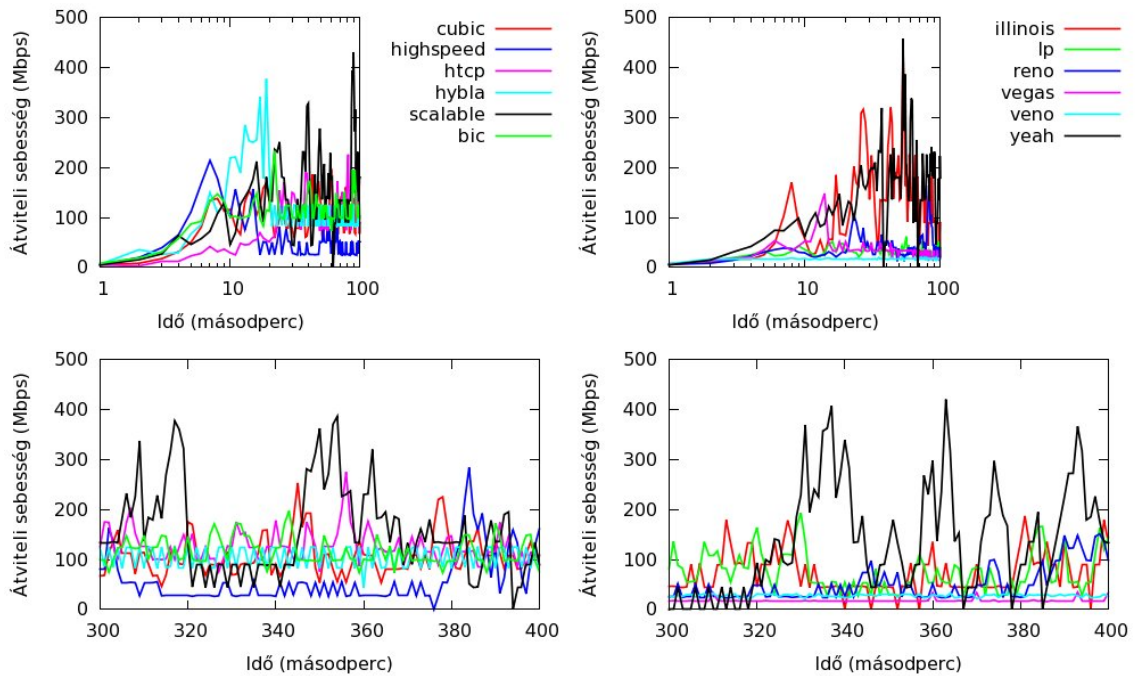
	Nagyméretű pufferek				B×D szerint hangolt pufferek			
variáns	min	max	átlag	szórás	min	max	átlag	szórás
bic	8,39	233	119,1988	28,40821	3,15	256	120,5523	30,55871
cubic	5,24	253	99,33488	35,24786	0	234	98,48247	38,83651
highspeed	0	284	59,04923	40,95387	0	196	54,77315	35,08446
htcp	3,15	275	113,982	30,86608	5,24	230	118,4242	28,39138
hybla	6,29	377	104,742	30,97009	9,44	284	117,6321	29,92355
illinois	0	431	84,91073	57,39205	0	553	150,4295	88,84601
lp	0	238	62,52507	39,1015	0	223	58,39388	41,33843
reno	0	208	47,99713	33,1225	0	238	62,67273	44,16841
scalable	0	510	167,8292	91,48467	0	653	167,7304	99,74587
vegas	0	148	20,9324	10,35227	5,24	26,2	19,04407	2,088778
veno	7,34	75,5	26,37207	11,69053	7,34	21	17,53123	1,344568
yeah	0	555	171,8542	107,6046	0	512	133,5308	101,2736

10. táblázat: Bitráta statisztikák (Mbit/s) 90 ms késleltetésű WAN környezetben nagyméretű és B×D szerint hangolt pufferekkel

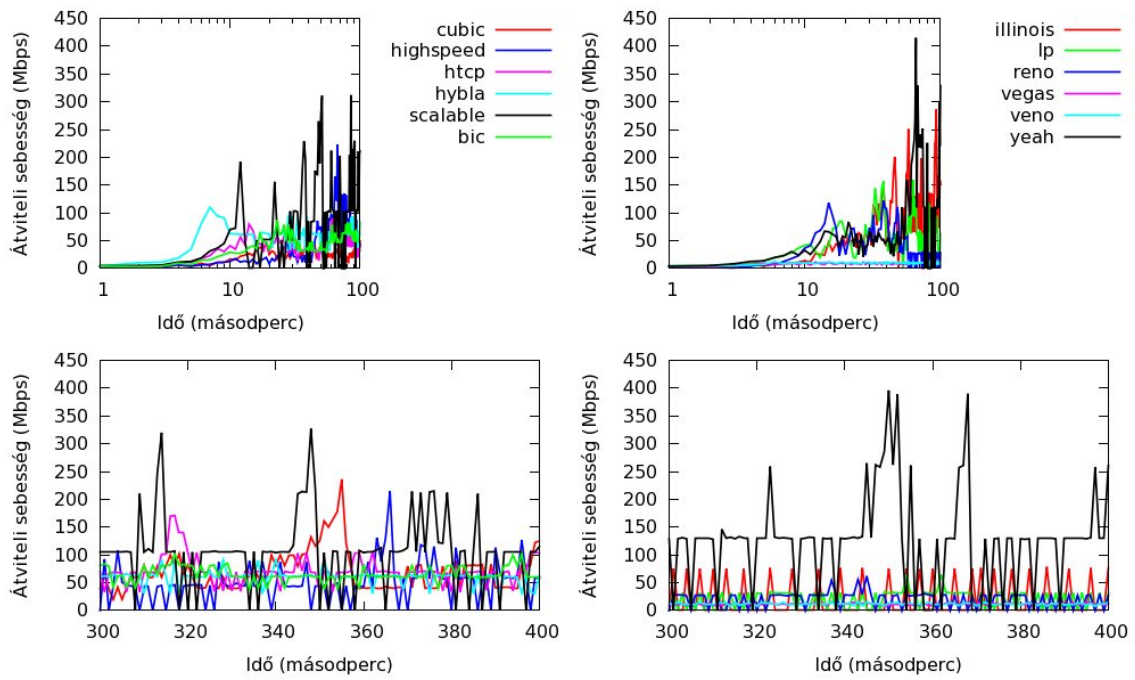
A puffermemóriák nagy méretre növelésével 90 ms késleltetés esetén magasabb bitráta értékeket értek el a variánsok a kapcsolat során, de jelentek meg 0 értékek is, így az átlag átviteli sebesség szinte mindenhol csökkent. A Scalable és a Yeah variánsok esetében a maximális átviteli sebesség 500 Mbit/s felett is volt, de a szórásuk is jelentősen emelkedett, így nem tudtak jobb átlagos átviteli sebességet produkálni az alapértelmezett esethez képest. A BIC és a CUBIC variánsok nem voltak érzékenyek a pufferméret változásokra, szinte ugyanazokat az eredményeket érték el, mint alapértelmezett puffermemória esetén.

A $B \times D$ szerint hangolt kernelparaméterekkel mért eredmények nem mutattak lényeges teljesítménynövekedést a nagyméretű pufferekkel mért esethez képest - az Illinois kivételével, ami a legjobb átlagos átviteli sebességet érte el a variánsok közül. A Scalable 650 Mbit/s fölötti értéket is elért. Ez több mint háromszorosa az alapértelmezett puffermemória esetén mért maximum eredményeknek.

90 ms késleltetésű kapcsolatnál nem lehet hosszú távon az átviteli sebességen javítani a puffer memóriák növelésével. Pillanatnyi magas értékek elérhetőek hangolással, de ez nem eredményez teljesítménynövekedést a kommunikáció során. A Scalable és a Yeah algoritmusok tudják a legnagyobb átviteli sebesség átlagot elérni ekkora késleltetésű kapcsolaton.



15. ábra: Átviteli sebességek 180 ms késleltetésnél nagyméretű pufferekkel



16. ábra: Átviteli sebességek 180 ms késleltetésnél $B \times D$ szerint hangolt pufferekkel

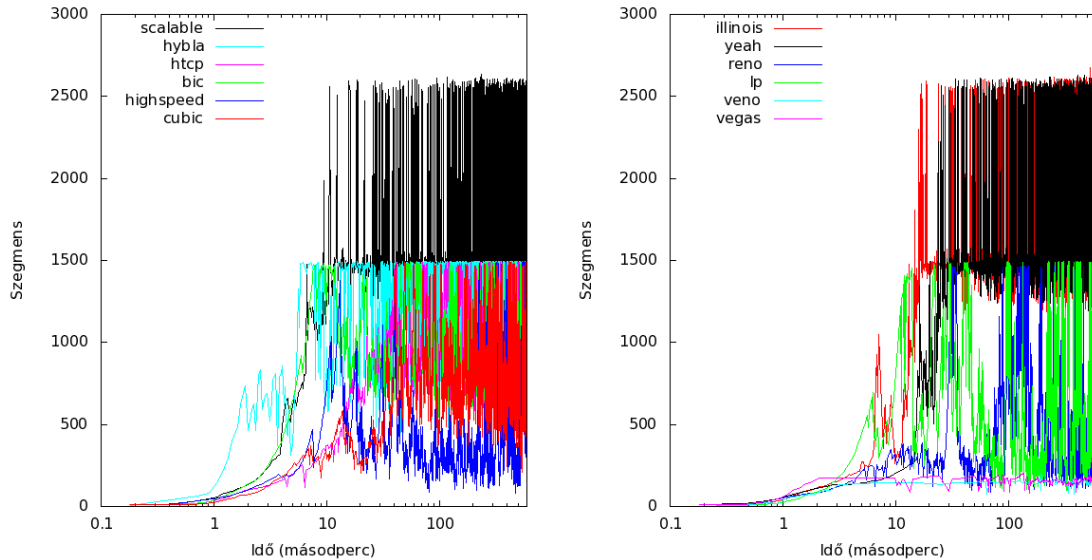
	Nagyméretű pufferek				B×D szerint hangolt pufferek			
variáns	min	max	átlag	szórás	min	max	átlag	szórás
bic	30,4	163	71,63621	24,78837	28,3	117	65,34416	17,63912
cubic	0	171	52,09949	26,56609	0	236	54,14203	26,8421
highspeed	0	209	33,03739	27,52064	0	223	47,41805	32,5642
htcp	11,5	184	67,04129	27,49423	14,7	171	65,78359	24,31246
hybla	0	90,2	54,6022	12,29912	29,4	97,5	59,36633	14,06839
illinois	0	194	47,76919	33,04453	0	286	40,26481	50,67047
lp	0	141	31,60198	25,4589	0	159	26,6049	25,82688
reno	0	142	22,90745	21,16438	0	122	23,08037	20,14479
scalable	0	274	71,05178	43,96764	0	590	97,4828	75,90196
vegas	5,24	17,8	10,52633	2,206701	5,24	16,8	10,61303	2,066065
veno	0	143	35,63418	28,95525	5,24	33,6	12,54008	4,515304
yeah	0	332	103,5328	62,88502	0	442	114,7086	85,08246

11. táblázat: : Bitráta statisztikák (Mbit/s) 180 ms késleltetésű WAN környezetben nagyméretű és B×D szerint hangolt pufferekkel

180 ms-os késleltetés esetén már voltak variánsok, amelyek esetében teljesítménynövekedést eredményezett a puffermemóriák megnövelése. A Yeah torlódásvezérlő algoritmusnál a 9. táblázatban szereplő átlagos átviteli sebességhez képest jelentős javulást eredményezett a $B \times D$ szerinti pufferhangolás. A maximálisan elért átviteli sebesség a négyszeresére, szórása szintén többszörösére nőtt az alapértelmezett méretű pufferek esetében mérthez képest. A Scalable közel 600 Mbit/s sebességet is elért az adott másodpercben, de átlagban csak kis mértékben javult. A többi variáns esetében nem történt javulás a nagyméretű pufferek esetéhez képest.

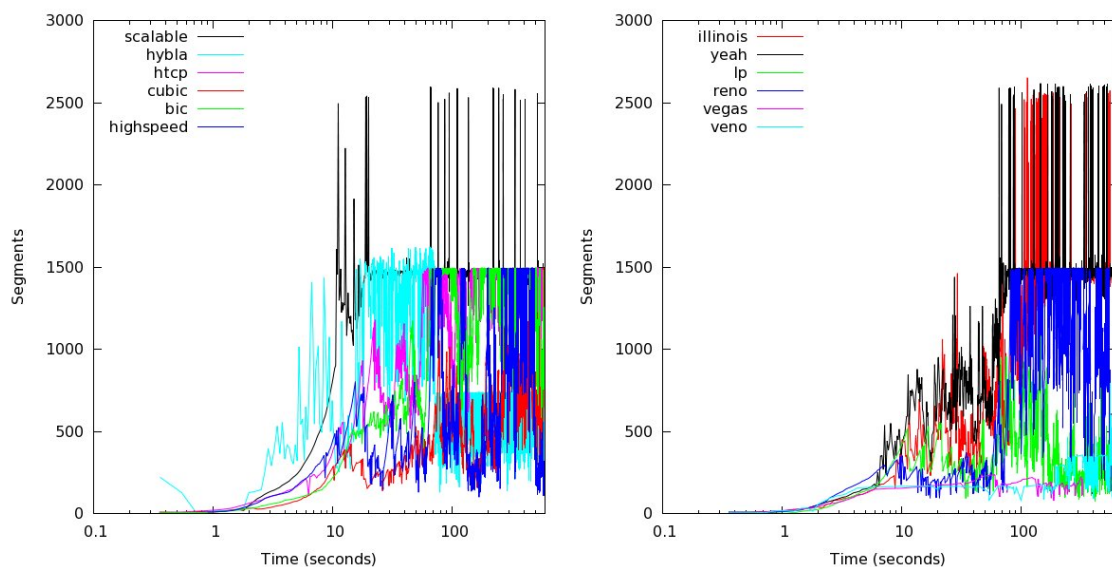
A 90 ms késleltetésű alapértelmezett puffermemóriával rendelkező kapcsolathoz képest fele akkora átlagos átviteli sebességeket eredményezett a 180 ms késleltetésű esetben. A Scalable és a Yeah nagyobb átviteli sebességet tudott elérni a hangolásnak köszönhetően.

3.3.2 Torlódási ablakméret

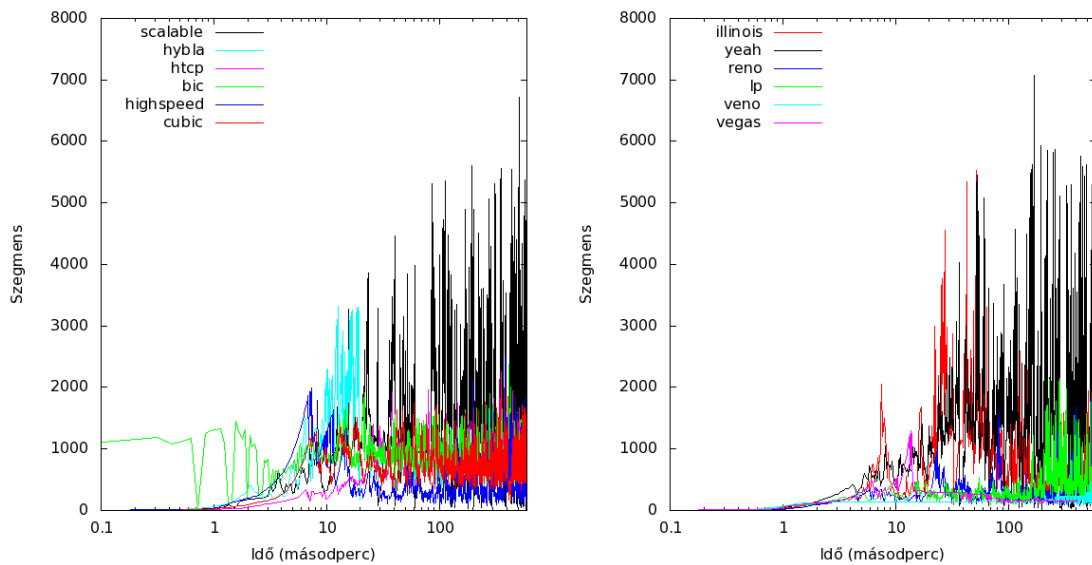


17. ábra: Ablakméretek WAN környezetben 90 ms késleltetéssel alapértelmezett méretű pufferekkel

A 90 ms késleltetésű kapcsolaton - alapértelmezett pufferekkel - kiemelkedő átviteli sebességet eredményező Scalable, Illinois és a Yeah, tíz másodperces felfutási idő után ablakméretben is nagyobb intervallumban mozgott, mint a többi variáns. 1 400-tól 2 600-as szegmens méreteket is elértek, szemben a többivel, ahol 1 500 fölé nem sikerült az ablakméretet emelniük. 180 ms késleltetés esetén is hasonló értékek tapasztalhatóak. A három jól teljesítő variáns - bár kisebb gyakorisággal - ugyanakkora ablakméreteket is elért.

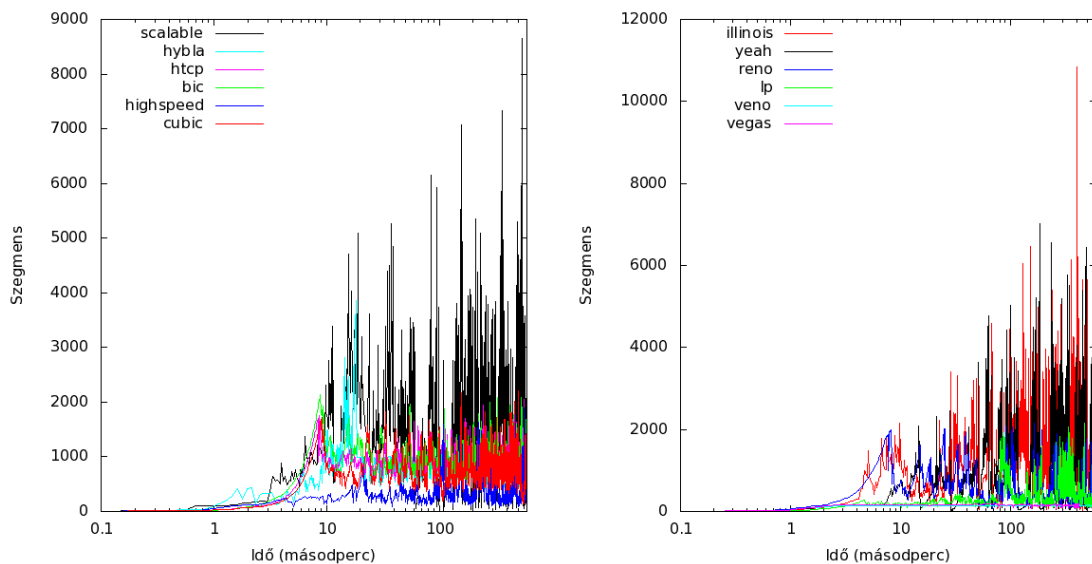


18. ábra: Ablakméretek WAN környezetben 180 ms késleltetéssel alapértelmezett méretű pufferekkel



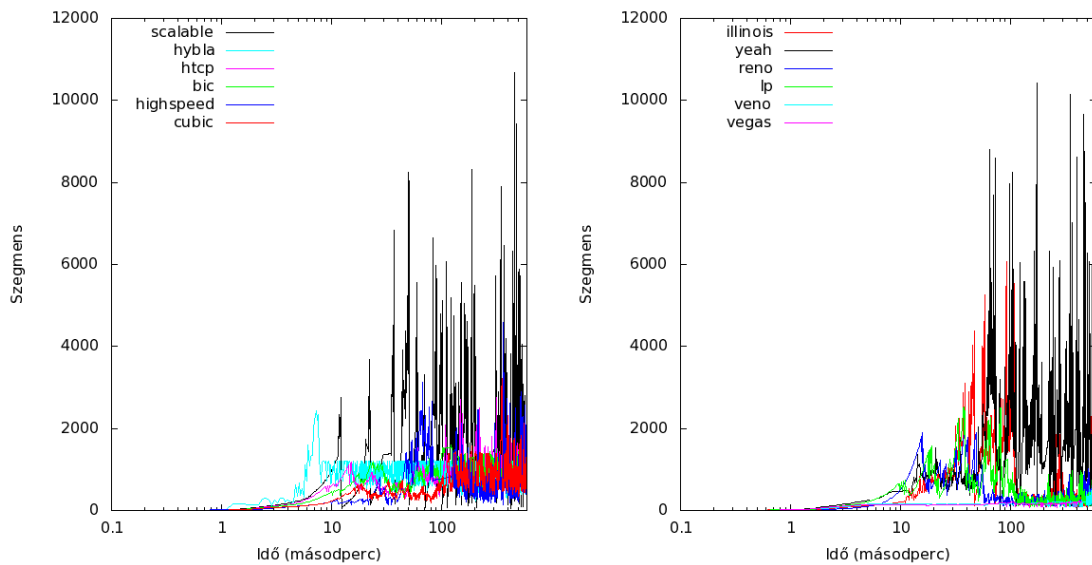
19. ábra: Ablakméretek WAN környezetben 90 ms késleltetéssel nagyméretű pufferekkel

Nagyméretű pufferek esetében a Scalable és a Yeah 6 000 szegmens fölötti ablakméreteket is elért. A Hybla és az Illinois tudott még magas eredményeket produkálni. A többi variáns 2 000 szegmens alatti ablakméreteket tudott alkalmazni a mérés során.



20. ábra: Ablakméretek WAN környezetben 90 ms késleltetéssel $B \times D$ szerint hangolt pufferekkel

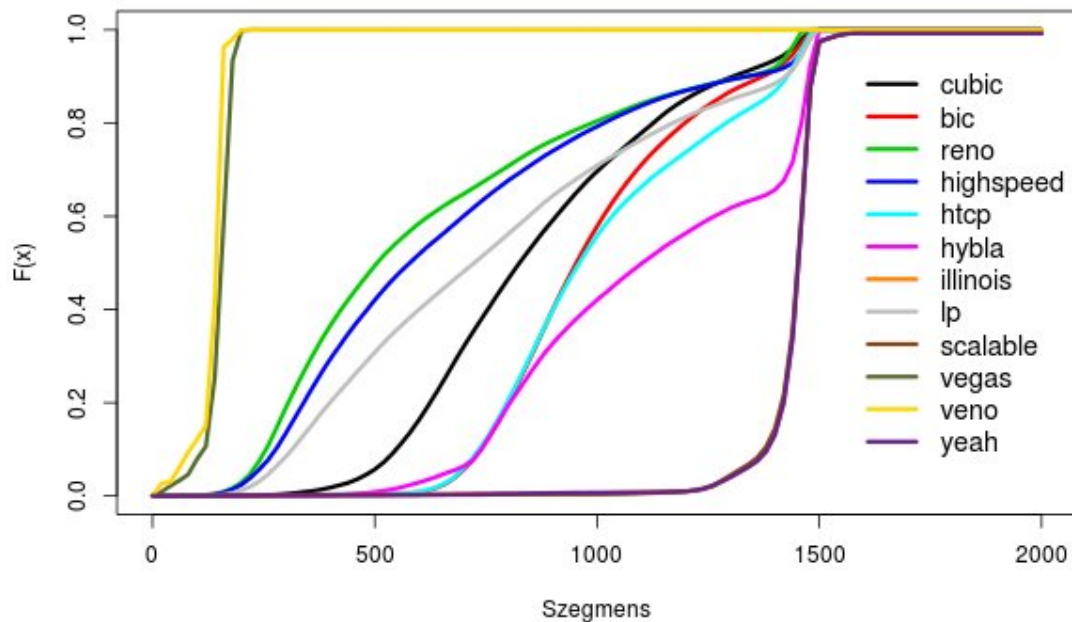
A $B \times D$ szerint hangolt pufferméretek alkalmazása mellett megjelentek extrém nagy ablakméret értékek is a Scalable és az Illinois torlódásvezérlő algoritmusok esetében. A Yeah variáns esetében nem történt növekedés a nagyméretű pufferekkel mért esethez képest.



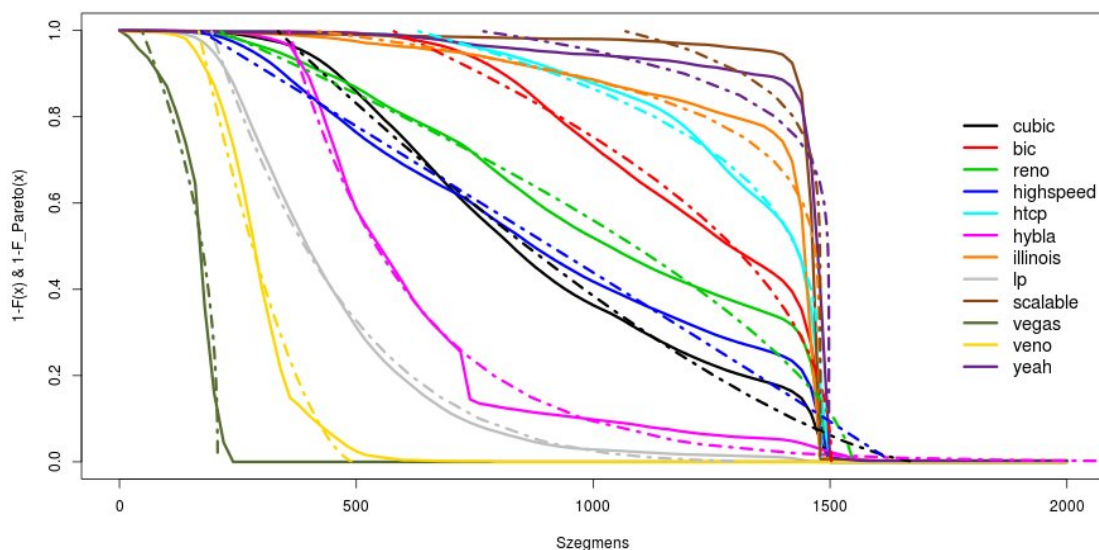
21. ábra: Ablakméretek WAN környezetben 180 ms késleltetéssel $B \times D$ szerint hangolt pufferekkel

A 180 ms késleltetésű $B \times D$ szorzatot a (3) egyenlet határozza meg. A nagy késleltetésnek és sebességnek köszönhetően 200 Mbyte fölötti puffermemóriákat határoz meg. Ennek köszönhetően a Scalable és a Yeah torlódásvezérlő algoritmusok esetében jelentek meg extrém nagy 10 000 szegmens fölötti torlódási ablakméret értékek is.

A variánsok ablakméret minta értékeiből tapasztalati eloszlásfüggvényt készítettem (7) a két késleltetés és a három pufferméret eset összes kombinációjával.



22. ábra: Ablakméretek tapasztalati eloszlásfüggvényei 90 ms késleltetéssel alapértelmezett méretű pufferekkel

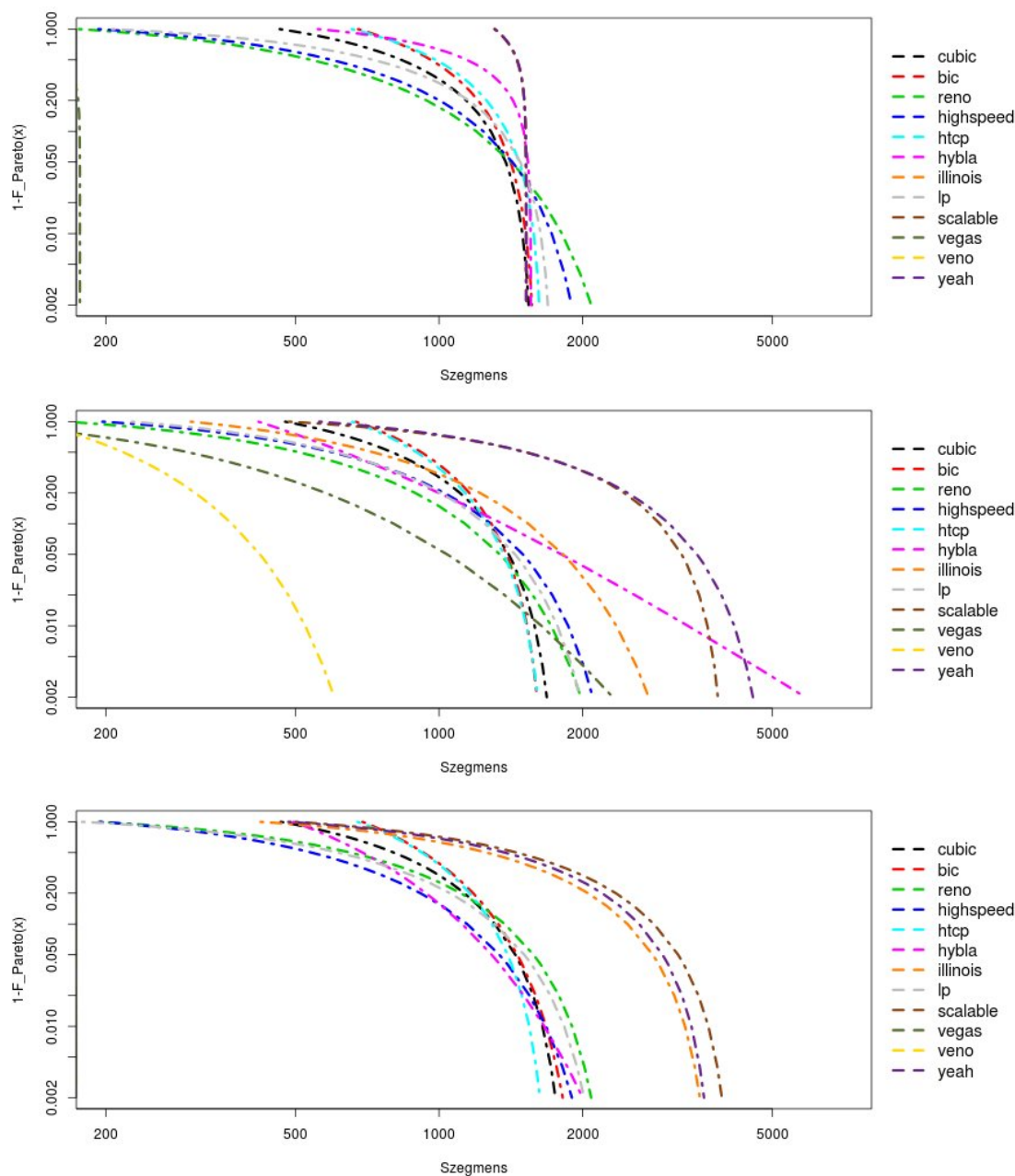


23. ábra: Komplementer ablakméret eloszlásfüggvények és az illesztett Pareto eloszlásfüggvények alapértelmezett méretű puffermemória méretekkel 180 ms késleltetésnél

	90 ms késleltetés						180 ms késleltetés					
	Alapértelmezett		Max. méretű		B×D szerint hangolt		Alapértelmezett		Max. méretű		B×D szerint hangolt	
	A2	P	A2	P	A2	P	A2	P	A2	P	A2	P
bic	2071	1	6897	1	5089	1	799.10	0.96	2327	1	3941	1
cubic	1809	1	2244	1	2750	1	549.5	1	771.2	1	2800	1
highspeed	318.9	1	235.1	1	558.9	1	357.2	1	84.95	1	1238	1
hybla	1016	1	406.7	1	1400	1	257.7	1	197.8	1	566.9	1
htcp	1357	1	3051	1	4433	1	14170	1	1138	1	3325	1
illinois	11520	1	1678	1	8218	1	3792.0	0.49	587.5	1	124	1
lp	285.4	1	685.8	1	185.2	1	472.9	1	58.13	1	173.5	1
reno	293.4	1	181.2	1	288.6	1	615.6	1	153.1	1	246.2	1
scalable	11610	1	7086	1	9902	1	5764.0	0.51	933.2	1	6175	1
vegas	173.4	0.99	26.42	1	289.6	0.95	99.75	0.99	176	1	176.8	1
veno	63.880	0.60	199.5	1	125.4	0.57	268.1	1	81.33	1	185.7	1
yeah	11330	1	6881	1	9167	1	7877.0	0.53	1797	1	6922	1

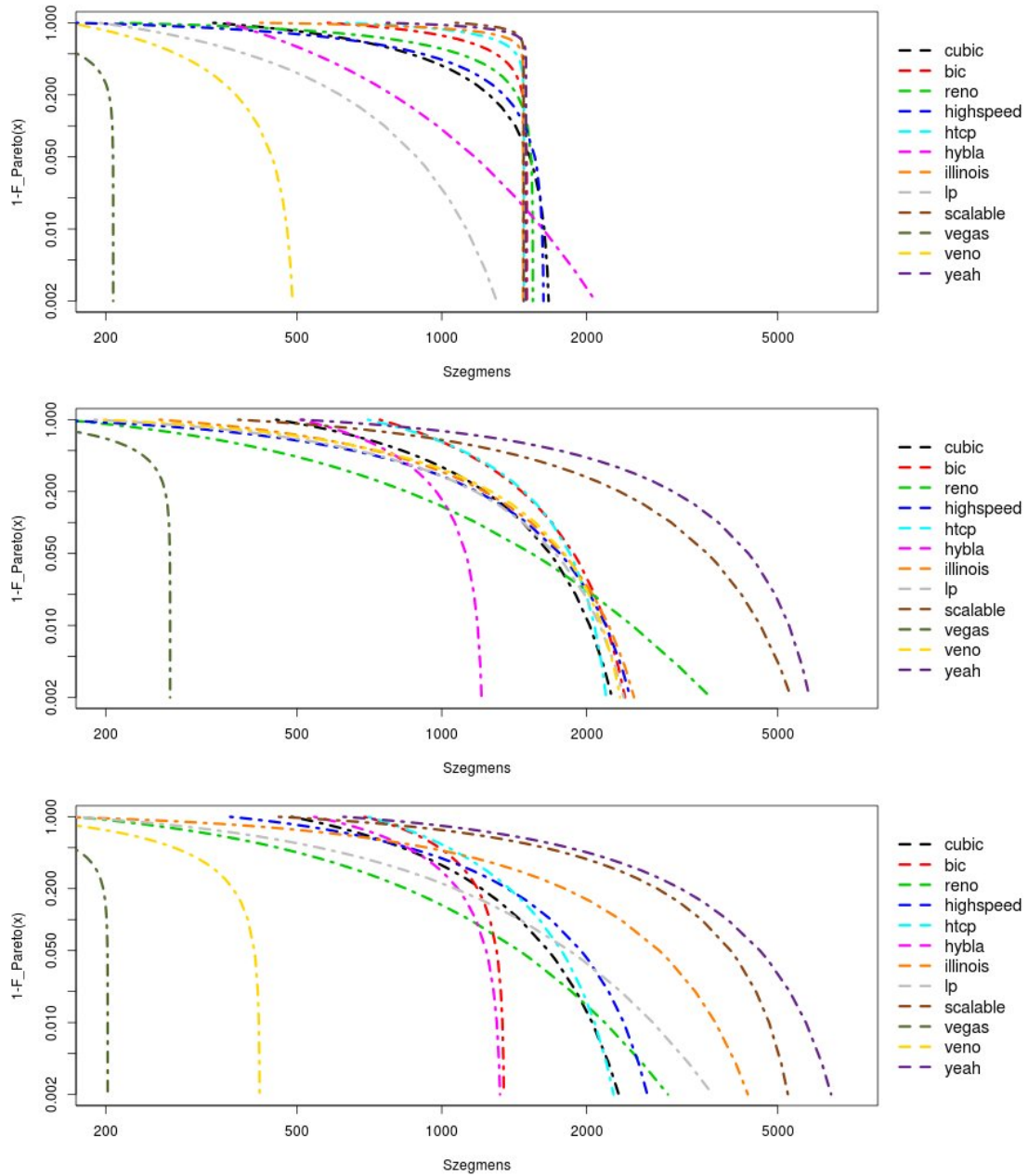
12. táblázat: Anderson-Darling teszt A^2 értékei és azok valószínűségei a variánsok Pareto-ra illeszkedésének vizsgálata során, WAN környezetben

A 23. ábra szemlélteti, hogy a komplementer ablakméret eloszlásfüggvények a nevezetes ξ , α , κ paraméterű Pareto eloszlást követnek, amit statisztikailag a 12. táblázat igazol. Kijelenthető, hogy a komplementer eloszlásfüggvények hosszú lecsengésűek. Az Illinois a Scalable és a Yeah esetében 180 ms késleltetésnél alapértelmezett puffermemóriákkal az eloszlásfüggvények a megfelelő paraméterű Pareto görbékhez képest meredekebben csengnek le a memória korlátok miatt, ezért csak 50% az illeszkedés. A hangolást követően egyértelműen hosszú lecsengésűnek bizonyultak. Az illesztett eloszlásfüggvények paramétereit a 13. táblázat és a 14. táblázat tartalmazza.



24. ábra: Illesztett Pareto eloszlás görbék logaritmikus idő és szegmensméret skálán 90 ms késleltetéssel alapértelmezett, nagyméretű és $B \times D$ szerint hangolt puffer esetekben.

A küldő és fogadó oldali memóriák növelésével nagytávolságú WAN kapcsolat esetén is nagyobb torlódási ablakméret érhető el. Különösen látványos változást eredményezett a Hybla, az Illinois, a Scalable és a Yeah esetében. A nagyméretű pufferek megfelelőbbek voltak a Hybla és a Yeah variánsok számára, a $B \times D$ szerint hangoltak viszont az Illinois-nak kedveztek. A Scalable ablaknövelő (5) algoritmus mindkét esetben hasonlóan magas ablakméreteket eredményezett.



25. ábra: Illesztett Pareto eloszlás görbék logaritmikus idő és szegmensméret skálán 180 ms késleltetéssel alapértelmezett, nagyméretű és $B \times D$ szerint hangolt puffer esetekben.

180 ms-os késleltetésű kapcsolat esetében is nagyobb valószínűséggel voltak magasabb torlódási ablakméretek a növelt puffermemóriák esetében. A Hybla kivételével az összes variáns esetében nőtt a nagyobb értékek valószínűsége. A Scalable és az Illinois eredményezték ismét a legnagyobb ablakméret valószínűségeket. A $B \times D$ szerint hangolt esetben a Yeah algoritmus 5%-os valószínűséggel ért el 5 000 szegmens fölötti ablakméret értékeket.

3.3.3 Következtetések

A WAN mérések során az átlagos átviteli sebességértékek jelentősen elmaradtak a vonalon elérhető maximum sávszélességhez képest. A 180 ms késleltetésű kapcsolat esetében arányosan csökkent az átviteli sebesség a 90 ms-os esethez képest, mivel a *netem* modul konstans késleltetést biztosított nagyon kis szórással.

90 ms-os esetben nem feltétlen eredményez átviteli teljesítménynövekedést a küldő és fogadó oldali puffermemóriák méretének növelése. A maximálisan elért átviteli sebességértékek növekedtek, de a szórásuk is jelentősen fokozódott, így a teljesítmény a növekedés helyett gyengülő tendenciát mutatott, ugyanis nagyobb ablakméretek használata során csomagvesztéskor nagyobb adatmennyiség újraküldését kell megvalósítani. 180 ms-os esetben viszont teljesítmény-növekedés volt a kernel paraméterek hangolásának köszönhetően. Legnagyobb mértékű, 30%-os növekedést a Yeah algoritmus eredményezett $B \times D$ szerint hangolt esetben. Így a nagyobb késleltetés okozta sebességcsökkenés mérséklődött a hangolásnak köszönhetően. Összességében a leghatékonyabb torlódásvezérlő algoritmusnak a Scalable és a Yeah bizonyultak átviteli sebesség szempontjából.

A torlódási ablakméretet nem befolyásolta a késleltetés mértéke, sokkal inkább a puffermemóriák mérete. Ezt igazolja a 25. ábra, ahol a Yeah variáns eloszlásfüggvényéhez illesztett Pareto görbe a leghosszabb lecsengésű. A küldő oldal 10 430 szegmens méretű adatot is tudott egyetlen nyugtával küldeni 180 ms késleltetésű kapcsolat esetén $B \times D$ szerint hangolt kernel paraméterekkel, ami több mint 15 Mbyte, ha 1 500 byte-os szegmensmérettel számolunk. Torlódási ablakméret szempontjából szintén a Scalable és a Yeah bizonyult a leghatékonyabbnak: kiemelkedő ablakméret értékeket eredményeztek a többi variánshoz képest. A puffermemória méretek növelésével a különbség csak tovább nőtt.

Nagyávolságú WAN kapcsolatok esetén a nagy késleltetés okozta esetleges csomagvesztések miatt nagymértékben romlik a sávszélesség kihasználtsága. Ezt a minőségromlást a Scalable vagy a Yeah torlódásvezérlő algoritmusok használatával mérsékelhetjük $B \times D$ szorzat szerint növelt puffermemóriákkal.

	λ	ξ	$\alpha,$	κ	
bic	0.001003334	674.9872	493.3506	0.5336211	Alapértelmezett
cubic	0.001146259	461.736	652.8043	0.5896199	
highspeed	0.001470373	191.5449	652.1698	0.3348965	
htcp	0.0009809719	654.3892	576.9379	0.580617	
hybla	0.000895427	554.0339	1280.662	1.275713	
illinois	0.0006930503	1304.525	379.7974	1.744764	
lp	0.001282107	204.2375	926.4117	0.6091126	
reno	0.001565914	174.5632	567.1707	0.2222413	
scalable	0.0006932122	1309.904	345.1819	1.602093	
vegas	0.006693106	78.3988	256.8794	2.617579	
veno	0.007488943	48.19023	414.2464	3.854072	
yeah	0.0006928406	1306.2	376.4486	1.74512	
	λ	ξ	$\alpha,$	κ	
bic	0.001049226	678.4774	411.881	0.4998966	Nagyméretű
cubic	0.001152247	481.1031	512.0992	0.3240532	
highspeed	0.001407619	215.835	640.0291	0.2940739	
htcp	0.001032098	673.9078	388.3661	0.316531	
hybla	0.001211867	523.156	477.7887	0.5819924	
illinois	0.001544885	229.047	543.8594	0.3003207	
lp	0.001285568	212.1455	848.2774	0.4994627	
reno	0.001464595	206.0843	649.6681	0.3628492	
scalable	0.0009726996	293.6013	1357.69	0.8485418	
vegas	0.007041975	62.40679	290.4971	2.649515	
veno	0.001540007	208.9087	548.7958	0.2460201	
yeah	0.0009719383	297.0379	1384.114	0.8912953	
	λ	$\xi,$	$\alpha,$	κ	
bic	0.00108096	640.662	471.4481	0.6574524	B×D szerint hangolt
cubic	0.001227018	453.986	621.8692	0.7226405	
highspeed	0.001196187	443.0858	496.7716	0.26436	
htcp	0.001075292	664.0592	405.4663	0.5247616	
hybla	0.001340529	488.8751	374.9788	0.4584983	
illinois	0.0009791026	326.0308	1300.125	0.8698418	
lp	0.002849733	71.65177	284.8368	0.01997605	
reno	0.001571714	204.0488	669.3169	0.5486299	
scalable	0.0009863133	325.9015	1209.533	0.758106	
vegas	0.006794549	91.53143	190.0725	2.415782	
veno	0.007879068	30.84992	537.2592	4.592451	
yeah	0.0009859885	313.3685	1246.105	0.7780118	

13. táblázat: Az illesztett exponenciális és Pareto eloszlások paraméterei 90 ms késleltetésű WAN kapcsolat esetében

	λ	ξ	$\alpha,$	κ	
bic	0.000826937	575.2982	2003.718	2.160519	Alapértelmezett
cubic	0.001116883	332.9441	963.3473	0.7129082	
highspeed	0.001101748	170.1166	1490.263	1.020609	
htcp	0.0007634074	624.6921	3413.495	3.981572	
hybla	0.001594952	357.2278	265.749	-0.01483372	
illinois	0.0007472911	404.7691	7195.857	6.709317	
lp	0.0022268	194.0087	303.5674	0.190151	
reno	0.000979392	210.297	2060.879	1.541958	
scalable	0.0006957192	1061.014	3746.599	8.955152	
vegas	0.006302998	47.24899	366.6182	2.29084	
veno	0.003425045	167.1795	200.8435	0.6094846	
yeah	0.0007047584	756.1345	6246.342	8.424295	
	λ	ξ	$\alpha,$	κ	
bic	0.0008740867	719.1312	590.6431	0.39001	Nagyméretű
cubic	0.001011859	438.0877	783.451	0.423959	
highspeed	0.001186103	235.4836	762.8179	0.2554324	
htcp	0.0008500114	714.6298	717.2319	0.5530384	
hybla	0.001124835	432.5398	878.8763	0.925337	
illinois	0.001237062	333.1038	750.874	0.5799119	
lp	0.001233852	188.4369	856.9138	0.3776014	
reno	0.001989628	202.8455	253.0491	-0.1558303	
scalable	0.0009764742	465.5199	942.9035	0.6880587	
vegas	0.004637107	98.73436	358.4081	2.065483	
veno	0.001315211	235.5207	610.2645	0.1628219	
yeah	0.001063432	449.5851	737.1241	0.5019848	
	λ	$\xi,$	$\alpha,$	κ	
bic	0.0009414575	555.0875	796.3078	0.5703313	B×D szerint hangolt
cubic	0.001084652	370.0273	775.3828	0.404864	
highspeed	0.001058946	380.6465	766.1938	0.3592491	
htcp	0.0008701738	731.9843	561.8894	0.3467738	
hybla	0.001251881	498.8868	362.1215	0.2074286	
illinois	0.0005362975	442.5368	1480.577	0.04112049	
lp	0.001251298	213.0408	614.0469	0.04763076	
reno	0.00173581	230.6961	358.5463	0.0380494	
scalable	0.0006358582	506.3384	1180.216	0.1067923	
vegas	0.005742395	62.87108	436.3985	2.921898	
veno	0.002095965	193.2053	307.5333	0.08323816	
yeah	0.0004504651	583.4142	2034.873	0.2434197	

14. táblázat: Az illesztett exponenciális és Pareto eloszlások paraméterei 180 ms késleltetésű WAN kapcsolat esetében

4. Összegzés

Összességében a mérések során nem csak a TCP variánsok viselkedését sikerült karakterizálni nagy sebességen, hanem azok pufferméret függését is az adott hálózati környezetben. LAN implementáció esetében az alacsony késleltetés miatt nincs szükség a küldő és fogadó oldali memóriák növelésére, ugyanis voltak olyan variánsok, melyek alapértelmezett pufferek mellett a vonalon lehetséges maximum sávszélességet is elérték. Ilyen algoritmus volt a CUBIC, a Hybla, a Veno a Yeah és az alacsony késleltetésnek köszönhetően a skálázásra nem alkalmas klasszikus torlódásvezérlő algoritmus: a Reno is. Azon variánsoknál, melyek alapesetben nem nyújtottak jó teljesítményt, a pufferméret növelésével teljesítménynövekedés volt elérhető, így például a BIC esetében. Annak ellenére, hogy nem volt szükség magas ablakméretekre a vizsgált hálózati környezetben, a $B \times D$ szorzat szerint hangolt puffermemóriáknak köszönhetően ablakméret növekedés volt tapasztalható a skálázható variánsoknál. Nagy késleltetésű WAN kapcsolat esetében viszont már jelentős sebességnövekedést lehet elérni, ezen szorzat szerint hangolt küldő és fogadó oldali memóriák mellett. A nagyobb ablakméreteknek köszönhetően mérséklődhet a nagy távolság okozta sebességcsökkenés, ha megfelelő TCP torlódásvezérlő algoritmust alkalmazunk a kommunikáció során. Ezen algoritmusok közé tartozik a Scalable és a Yeah. Sikerült azt is megállapítani, és statisztikailag is igazolni, hogy alapvetően a torlódásvezérlő algoritmusok ablakméret növelő függvényei long-tail típusú Pareto eloszlással jellemezhetőek, csak más-más paraméterezéssel.

5. Tovább lépési lehetőségek

A dolgozat tapasztalatai alapján tervezem a CUBIC TCP variáns módosítását a kutatócsoport által kifejlesztendő ultra nagysebességű monitoring rendszer által támasztott követelményeknek megfelelően. Továbbá az eredmények alapul szolgálhatnak TCP variánsok matematikai modellezéséhez és ultra nagysebességű TCP adatátvitel szimulációs vizsgálatához.

6. Irodalomjegyzék

- [1] „IEEE 802.3 ETHERNET WORKING GROUP,” <http://www.ieee802.org/3/> [Letöltve: 2012 október].
- [2] L. Xu, „Binary increase congestion control (BIC) for fast long-distance networks”, INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, 7-11 March 2004, pp 2514- 2524.
- [3] Z. Yue, „The performance evaluation and comparison of TCP-based high-speed transport protocols,” Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference, 22-24 June, pp 509-512.
- [4] W. Lv, „Research of TCP optimization technology for long-distance and high bandwidth-delay private network,” in Computer Science and Information Processing (CSIP), 2012 International Conference, 24-26 August 2012, pp 381-384.
- [5] H. Jung, „Adaptive delay-based congestion control for high bandwidth-delay product networks,” INFOCOM, 2011 Proceedings IEEE, 10-15 April 2011. pp 1885-2893.
- [6] „Fedora Projekt honlap” <http://fedoraproject.org/> [Letöltve: 2012 október]
- [7] „Iperf” Available: <http://iperf.sourceforge.net/> [Letöltve: 2012 október]
- [8] „Intel® Ethernet Converged Network Adapter,” <http://www.intel.com/content/www/eu/en/network-adapters/10-gigabit-network-adapters/ethernet-x540-t2-brief.html?wapkw=10g+pcie+nic> [Letöltve: 2012 október].
- [9] „The Linux Foundation” <http://www.linuxfoundation.org/collaborate/workgroups/networking/tcpprobe> [Letöltve: 2012 október].
- [10] S. Floyd, RFC 3649: HighSpeed TCP for Large Congestion Windows.
- [11] T. Kelly, „Scalable TCP: improving performance in highspeed wide area networks,” ACM SIGCOMM Computer Communication Review, 2003.
- [12] S. Ha, „BIC and CUBIC” <http://research.csc.ncsu.edu/netsrv/?q=content/bic-and-cubic>. [Letöltve: 2012 október]
- [13] L. X. Injong Rhee „CUBIC: A New TCP-Friendly High-Speed TCP Variant”, ACM SIGOPS Operating Systems Review, July 2008, 64-74.
- [14] R. F. Carlo Caini, TCP Hybla: a TCP enhancement for heterogeneous networks, International Journal of Satellite Communications and Networking, 5 2004, 547-566.
- [15] R. S. D. Leith, H-TCP: TCP for high-speed and long-distance networks.
- [16] T. B. R. S. Shao Liu, „TCP-Illinois: a loss and delay-based congestion control algorithm for high-speed networks”.
- [17] „TCP LP's Homepage,” <http://www.ece.rice.edu/networks/TCP-LP/> [Letöltve: 2012 október]
- [18] „Vegas Home Page,” <http://www.cs.arizona.edu/projects/protocols/> [Letöltve: 2012 október]
- [19] C. P. Fu, „TCP Veno: TCP enhancement for transmission over wireless access networks” Selected Areas in Communications, IEEE Journal on, 2003, pp 216-228.
- [20] A. P. C. ., F. V. Andrea Baiocchi, YeAH-TCP: Yet Another Highspeed TCP.
- [21] „The R Project for Statistical Computing,” <http://www.r-project.org/> [Letöltve: 2012

október]

- [22] Terdik György, Előadások a matematikai statisztikából.
- [23] J. R. M. W. J. R. Hosking, „Parameter and Quantile Estimation for the Generalized Pareto Distribution”.
- [24] J. R. M. Hosking, „L-Moments: Analysis and Estimation of Distributions Using Linear Combinations of Order Statistics”.
- [25] Q. J. Wang, Direct Sample Estimators of L Moments, 1996.
- [26] Laio, F. (2004), Cramer – von Mises and Anderson-Darling goodness of fit tests for extreme value distributions with unknown parameters, Water Resour. Res., 40, W09308

7. Saját munka leírása:

Magát a dolgozatot, teljes egészében önállóan készítettem a témavezető szakmai irányítása mellett.

8. Köszönetnyilvánítás

Köszönettel tartozom Dr. Orosz Péter témavezetőmnek, aki végig rendelkezésemre állt, és hasznos tanácsokat adott, a dolgozattal kapcsolatban, valamint Skopkó Tamásnak a technikai segítségnyújtásért.