VERİ SIKIŞTIRMA ALGORİTMALARININ AĞ İLETİŞİMİ ÜZERİNDEKİ PERFORMANSLARININ DEĞERLENDİRMESİ

Altan MESUT, Aydın CARUS
{altanmesut, aydinc}@trakya.edu.tr
Trakya Üniversitesi, Mühendislik Mimarlık Fakültesi,
Bilgisayar Mühendisliği Bölümü, EDİRNE

ÖZET

Bir sıkıştırma algoritması ağ üzerinden veri iletimini hızlandırmak için kullanılmak isteniyorsa, o algoritmanın sadece sıkıştırma oranı değil, sıkıştırma zamanı ve açma zamanı da önemlidir. Hızlı ağlarda sıkıştırma ve açma zamanları daha önemliyken, yavaş ağlarda sıkıştırma oranı ön plana çıkar. Sıkıştırma oranı tüm cihazlar için sabitken, sıkıştırma ve açma zamanları kullanılan kodlayıcı cihazın hızına göre değişmektedir. Bir sıkıştırma algoritmasının; gönderici tarafta sıkıştırma zamanı, sıkıştırılmış verinin gönderilme zamanı ve alıcı tarafta açma zamanı toplamı, sıkıştırma işlemi yapılmaksızın göndermeye göre daha yavaş ise, bu algoritmanın test edilen cihaz ve bant genişliği için kullanımı elverişsizdir. Fakat aynı algoritmanın daha hızlı bir kodlayıcı cihaz veya daha yavaş bir bant genişliğinde kullanımı elverişli olabilir. Her sıkıştırma algoritmasının, kullanılacağı cihazların performansına bağlı olarak, kazanç sağlayabilecekleri maksimum bant genişlikleri belirlenebilir. Bu çalışmada, seçilen sıkıştırma algoritmalarının 1 Mbit/sn, 10 Mbit/sn ve 100 Mbit/sn gibi farklı iletim hızlarında çalıştırıldığı durumlarda, birbirlerinin alternatifi olabilecekleri saptanmış ve sunulmuştur.

Anahtar Kelimeler: veri sıkıştırma, hızlı veri iletimi, LZW, LZP, LZOP, LZRW, Bzip, Gzip, PPM.

1. GİRİŞ

Verinin boyutunu azaltmak sadece saklama kapasitesini arttırmak amacıyla değil, ağ iletişimini hızlandırmak amacıyla da kullanılır. Verinin iki cihaz arasında hızlı iletimi ağ trafiğinin azalmasına neden olan bir durum olduğu için, daha küçük miktarda verinin gönderilmesi sadece ilgili iki cihaz için kazanç sağlamakla kalmaz, aynı ağa dâhil olan tüm cihazların veri iletiminin hızlanmasını sağlar.

Veri sıkıştırma yöntemleri kayıpsız ve kayıplı olmak üzere iki kategoriye ayrılır. Kayıpsız sıkıştırma yöntemleri ile sıkıştırılan veriler açıldığında orijinal hali ile aynı olarak elde edilebilirken, kayıplı sıkıştırma yöntemleri ile sıkıştırılan veriler açıldığında orijinale göre daha bozulmuş bir haldedir. Genellikle görüntü ve ses verilerini sıkıştırmak için kullanılan kayıplı yöntemler, sesin veya görüntünün algılanmasında büyük oranda etkisi olmayan verileri çıkartarak veya değiştirerek veriyi daha sıkıştırılabilir bir biçime dönüştürürler. Elde edilen büyüklüğü azaltılmış/değiştirilmiş veri (kayıplı veri) daha sonra uygun bir kayıpsız sıkıştırma algoritması ile sıkıştırılır. Açma işlemi yapıldığında ise, ancak büyüklüğü azaltılmış/değiştirilmiş veriye ulaşılabilir, orijinal veri tam olarak geri getirilemez.

Ağ iletişimini hızlandırmak için genellikle kayıpsız veri sıkıştırma yöntemleri tercih edilse de, ses ve görüntü verilerinin aktarılmasında kayıplı sıkıştırma yöntemleri de yaygın olarak kullanılmaktadır. Örneğin, İnternet üzerinden veya 3G destekli cep telefonları ile görüntülü konuşma yapılırken hem ses hem de video verileri kayıplı yöntemler ile sıkıştırılarak gönderilmektedir. Düşük hızdaki çevirmeli ağ internet bağlantılarını (56Kbps dial-up gibi) hızlandırmak için de istemci ve sunucu tarafına hem kayıplı hem de kayıpsız veri sıkıştırma algoritmalarını kullanan özel yazılımlar yüklenebilir. İnternet sayfalarındaki resimler genellikle JPEG veya GIF gibi kayıplı görüntü sıkıştırma algoritmaları ile sıkıştırılmış halde olsalar bile, kullanılan bu yazılımlar sayesinde kaliteden ‘istenilen miktarda’ ödün verilerek web sunucusunda daha fazla sıkıştırılıp istemciye gönderilmektedirler. HTML kodu ve diğer veriler ise kayıpsız yöntemlerle sıkıştırılıp gönderilirler.

Ağ iletişimini hızlandırmak için kullanılan kayıpsız veri sıkıştırma algoritmalarının çoğu dinamik

sözlük tabanlı algoritmalardır. Metin türündeki verileri ortalama %50 oranında sıkıştırabilen bu algoritmalar, veriyi hızlı sıkıştırabildikleri için ve verinin tamamını alıcı tarafa iletmeden de iletilen parçayı açabildikleri için tercih edilmektedirler. Dinamik sözlük tabanlı algoritmaların temelini A. Lempel ve J. Ziv tarafından yaratılan ve yayınlandıkları yıllara göre LZ77 [1] ve LZ78 [2] olarak isimlendirilen iki farklı algoritma oluşturmaktadır. Dinamik sözlük tabanlı yöntemlerin dışında Kısmi Eşleme ile Öngörü (PPM: Prediction by Partial Matching) modelini [3] kullanan sıkıştırma algoritmaları da ağ iletişimini hızlandırmak için kullanılabilir. Bu algoritmalar sözlük tabanlı algoritmalara göre daha yavaş çalışıp daha çok bellek kullanmalarına rağmen, sıkıştırma oranlarının daha fazla olması nedeniyle hızlı işlemcileri olan istemci ve sunucular için elverişlidirler.

2. VERİ SIKIŞTIRMANIN AĞ İLETİŞİMİNDE KULLANILMASI

Veri sıkıştırma algoritmaları soket programlama komutları kullanılarak ağ iletişiminde üst katmanlarda uygulanabilecekleri gibi, işletim sisteminin sunduğu ağ olanaklarından faydalanarak veri bağı katmanında da uygulanabilir. Noktadan Noktaya İletişim Protokolünün (PPP: Point to Point Protocol) bir parçası olan *Sıkıştırma Kontrol Protokolü* (CCP: Compression Control Protocol) [4], mevcut sıkıştırma algoritmalarının veri bağı katmanında kullanımı için uygun bir altyapı oluşturmaktır. Noktadan noktaya bağlantının her iki tarafında da veri sıkıştırma algoritmalarının yapılandırılmasından sorumlu olan bu protokol, PPP kısımlarından biri olan Bağlantı Kontrol Protokolü (LCP: Link Control Protocol) ile benzer bir paket yapısı kullanır. Farklı kişi ve kuruluşlar tarafından geliştirilen ve çoğunluğu LZ-türevi olan; Hewlett-Packard PPC, Stac LZS, Microsoft PPC (MPPC), Gandalf FZA, Magnalink MVRCA, Unix BSD, LZS-DCP, Deflate [5], V.42bis [6][7], V.44/LZJH [8] gibi birçok sıkıştırma algoritması CCP ile birlikte kullanılabilmektedir. Unix ve Linux işletim sistemlerinde **pppd** (PPP daemon) kullanılarak BSD ve Deflate algoritmalarının iletişimin her iki tarafında da etkinleştirilmesi veya devre dışı bırakılması sağlanabilir. Microsoft Windows işletim sistemlerinde ise **PPP Ayarları** üzerinden MPPC'nin etkin hale getirilmesi veya devre dışı bırakılması sağlanabilir.

Yukarıda anlatılan yazılım tabanlı veri sıkıştırma yöntemlerinin dışında, ağ iletişim cihazlarında donanımsal olarak kullanılan algoritmalar da vardır. Daha önce CCP'de kullanılan algoritmalar arasında verdiğimiz V.42bis ve V.44, aslında ITU-T tarafından modemler için geliştirilen ve LZ-türevi algoritmalar kullanan veri sıkıştırma standartlarıdır. Bunların dışında ADLC (Adaptive Lossless Data Compression) [9] gibi yine LZ-türevi olan bazı algoritmalar da veri iletişiminde donanımsal olarak kullanılmıştır [10].

3. KARŞILAŞTIRILAN SIKIŞTIRMA ALGORİTMALARI

Ağ iletişimini hızlandırmada kullanılan veri sıkıştırma algoritmaları genellikle LZ-türevi olan algoritmalar olduğu için, değerlendirmede kullanacağımız algoritmaları belirlerken, hem bu tipte olan algoritmaları, hem de daha fazla sıkıştırma oranına sahip olan fakat genellikle daha uzun sürede sıkıştırma ve açma yapan algoritmaları seçtik. Bu bölümde seçilen algoritmalar kısaca tanımlanmıştır.

LZW: A. Lempel ve J. Ziv tarafından geliştirilen LZ78 algoritması, daha sonra T. Welch tarafından geliştirilerek LZW [11] adını almıştır. Bu algoritma sıkıştırılacak olan verideki karakter katarlarını bir sözlüğe girerek, bu karakter katarı daha sonra tekrar ettiğinde onun sözlük sırasını kodlar. LZW algoritması sadece metin türü verilerde değil, bitmap resim dosyaları ve çalıştırılabilir dosyalar gibi diğer veri tipleri üzerinde de iyi sonuç verir. V.42bis, UNIX Compress, GIF ve TIFF görüntü sıkıştırma formatları LZW tabanlıdır. Karşılaştırmamızda Mark Nelson tarafından geliştirilen LZW algoritmasının C kodu kullanılmıştır (http://marknelson.us/attachments/lzw-data-compression/lzw.c).

LZRW1: R. N. Williams tarafından geliştirilen bu algoritma LZ77 ile aynı mantığı kullanarak sıkıştırma yapar [12]. LZ77, kodlanacak olan girdiden geriye doğru giderek, eğer bu girdi daha önce kodlandıysa bunu bulmaya çalışır. Eğer bulursa, kaç karakter geriye gittiğini ve kaç tane eşleşen ardıl karakter olduğunu kodlar. LZRW1, LZ77'den farklı olarak basit bir hash tablosu kullanıp daha hızlı sıkıştırma ve açma sağlar. Fakat bu hash tablosu nedeniyle sıkıştırma oranında küçük bir azalma olur. R. N. Williams daha sonra daha çok sıkıştırma oranına sahip olan fakat daha yavaş işlem yapan LZRW2 ve LZRW3 algoritmalarını da geliştirmiştir.

PPMd: PPM, kontekst modelleme ve tahminine dayalı uyarlamalı istatistiksel veri sıkıştırma tekniğidir. Bu modelde sıkıştırılmamış verideki sembol kümeleri (konteskt) hafızada saklanarak, kodlanan sembolden önce yer alan kontekste göre, o

sembolün ne olacağı tahmin edilmeye çalışılır. Örneğin 'elek' kontekstinden sonra 't' harfi gelmesi olasılığı ½ ise, bu oran kodlanır. Sıkıştırma oranı en iyi olan yöntemlerden biri olduğu için günümüzde birçok arşivleme programı PPM-türevi algoritmalar kullanmaktadır. PPM'in bir türevi olan PPMd, Dmitry Shkarin tarafından geliştirilen açık kaynak kodlu PPM tabanlı bir veri sıkıştırma algoritmasıdır [13]. Karşılaştırmamızda PPMd vJ rev1 kullanılmıştır.

LZP: C. Bloom tarafından geliştirilen LZP algoritması PPM-tipi kontekst modelleme ile LZ77-tipi karakter eşleme tekniklerini bir araya getirir [14]. Bu hızlı ve etkili algoritma, geçerli kontekstin en son tekrar ettiği durumu bulur ve bu konteksti takip eden karakter dizisini kodlanacak girdi ile karşılaştırır. İki karakter dizisi arasındaki eşleşme uzunluğu bulunur ve kodlanır. Eğer bu uzunluk sıfırsa, PPM gibi bir yöntem ile uyuşmayan karakter kodlanır. LZP'nin hız ve sıkıştırma oranı değişen 4 farklı uyarlamasından karşılaştırmamızda en hızlısı olan LZP1 kullanılmıştır.

Bzip: Bzip veri sıkıştırma programı Burrows-Wheeler blok sıralamalı kayıpsız veri sıkıştırma algoritması [15] ve aritmetik kodlama [16] kullanarak sıkıştırma yaparken, yeni sürüm olan Bzip2 ise blok sıralama işleminden sonra Huffman kodlaması [17] kullanır (aritmetik kodlamadan vazgeçilmesinin nedeni patent kısıtlamalarıdır). Sıkıştırma ve açma hızı LZ ailesine göre yavaş olan Bzip'in sıkıştırma oranı çok daha iyidir (PPM'e yakındır). Karşılaştırmamızda Bzip2 v1.0.4 kullanılmıştır.

Gzip: Gzip, LZ77 ile Huffman Kodlamasını bir arada kullanan Deflate algoritması ile sıkıştırma yapar. Deflate ilk olarak P. Katz tarafından kendi geliştirdiği PKZIP sıkıştırma programının 2. sürümünde kullanılmış ve daha sonra "IETF Network Working Group RFC 1951" olarak yayınlanmıştır [5]. Patent kısıtlaması olmaması nedeniyle kısa sürede yaygın olarak kullanılmaya başlanan Deflate, PKZIP dışında Gzip ve Winzip gibi birçok sıkıştırma programı ve PNG görüntü sıkıştırma formatı tarafından kullanılmaktadır. Karşılaştırmamızda Gzip v1.2.4 kullanılmıştır.

LZOP: M. Oberhumer tarafından geliştirilen LZOP algoritması yine Oberhumer tarafından ANSI C kullanılarak geliştirilen LZO veri sıkıştırma kütüphanesini kullanır. Taşınabilir bir kayıpsız veri sıkıştırma kütüphanesi olan LZO oldukça hızlı sıkıştırma ve çok hızlı açma sağlar. Gzip ile benzerlik gösteren LZOP'un hız avantajı olmasına rağmen sıkıştırma oranı Gzip'e göre daha düşüktür. Sıkıştırma oranına göre farklı sıkıştırma seviyeleri bulunan LZOP, sıkıştırma oranı yükseldikçe sıkıştırma hızını yavaşlatsa da, açma hızı bu durumdan etkilenmez. Karşılaştırmamızda LZOP v1.02 kullanılmıştır.

4. KARŞILAŞTIRMA YÖNTEMİ

Veri sıkıştırma algoritmaları ağ iletişimini hızlandırmak amacıyla kullanıldığında, sıkıştırma oranı dışında sıkıştırma ve açma hızları da önem taşır. Eğer bir algoritma yavaş sıkıştırma/açma yapıyorsa, sadece yavaş ağlarda kullanımı kazanç sağlayabilir. Hızlı çalışan bir algoritmanın sıkıştırma oranı kötüyse, bu algoritma da yavaş ağlarda verimsiz olacaktır. Önemli olan sıkıştırma ve açma yapan cihazların hızına ve ağın iletim hızına uygun olan bir algoritmayı seçmektir.

Ağın iletim hızı farklılığının yarattığı etkiyi bir örnek ile açıklamak gerekirse; 100 MB büyüklüğünde bir veri dosyası 100 Mbit/sn hızında gönderildiğinde yaklaşık olarak 8 sn sonra karşı tarafa ulaşır. Eğer bu dosya sıkıştırıldığında boyutu yarıya inebiliyorsa, 4 sn sonra karşı tarafa gönderilebilecektir. Eğer sıkıştırma ve açma zamanlarının toplamı 4 sn'den fazla ise sıkıştırma yapmadan göndermek daha mantıklı olacaktır. Fakat iletim hızı 10 Mbit/sn ise durum değişecektir. Bu defa, sıkıştırmadan gönderme zamanı 80 sn iken, sıkıştırma yaparak gönderme zamanı 40 sn olacaktır. Bu durumda, 100 MB büyüklüğünde veriyi %50 oranında sıkıştırabilen ve sıkıştırma ve açma zamanlarının toplamı 40 saniyeden düşük olan tüm algoritmaların kullanımı kazanç sağlayacaktır. Algoritmaların sıkıştırma oranı ve zamanı farklı ise; Örneğin X algoritması 100 MB veriyi 20 sn'de 30 MB büyüklüğe sıkıştırıp 10 sn'de açıyorsa, daha hızlı fakat daha az oranda sıkıştıran Y algoritması ise aynı veriyi 10 sn'de 50 MB büyüklüğe sıkıştırıp 5 sn'de açıyorsa, 10 Mbit/sn ağ iletim hızı için hesaplandığında yavaş işlem yapan X ile göndermek 1 sn daha hızlı olacaktır:

X ile gönderim:

Sıkıştırma 20 sn + İletim 24 sn + Açma 10 sn = 54 sn

Y ile gönderim:

Sıkıştırma 10 sn + İletim 40 sn + Açma 5 sn = 55 sn

1 sn fark az gibi görünse de ağ iletim hızı düştükçe fark artacaktır. Örneğin aynı hesaplama 1Mbit/sn hızında bir ağ için yapılırsa, aradaki gönderim zamanı farkı 145 sn olacaktır:

X ile gönderim:

Sıkıştırma 20 sn + İletim 240 sn + Açma 10 sn = 270 sn

Y ile gönderim:

Sıkıştırma 10 sn + İletim 400 sn + Açma 5 sn = 415 sn

Belirli bir sıkıştırma algoritmasının belirli bir cihazdaki sıkıştırma ve açma hızları biliniyorsa, o algoritmanın hangi ağ iletim hızına kadar kullanımının kazanç sağlayacağı hesaplanabilir. Toplam gönderim zamanı (t) sıkıştırma, gönderme ve açma sürelerinin toplamı olarak kabul edilirse, aşağıdaki denklem ile ifade edilebilir:

$$t = \frac{boyut}{SH} + \frac{s.boyut}{İH} + \frac{boyut}{AH} \quad (1)$$

boyut	gönderilecek verinin büyüklüğü
s.boyut	sıkıştırılmış verinin büyüklüğü
SH	sıkıştırma hızı
İH	iletim hızı
AH	açma hızı (sıkıştırılmamışa göre)

Algoritmanın kullanımının kazançlı olması için t değeri sıkıştırılmamış verinin gönderilme zamanından az olmalıdır:

$$\frac{boyut}{SH} + \frac{s.boyut}{İH} + \frac{boyut}{AH} < \frac{boyut}{İH} \quad (2)$$

Aşağıdaki sadeleştirmeler yapılarak iletim hızı denklemin sol tarafında yalnız bırakılır ve bu sayede algoritmanın verim sağlayacağı maksimum iletim hızı belirlenmiş olur:

$$\frac{boyut}{SH} + \frac{boyut}{AH} < \frac{boyut - s.boyut}{İH}$$

$$İH < \frac{boyut - s.boyut}{\left(\frac{boyut}{SH} + \frac{boyut}{AH}\right)}$$

$$İH < \frac{1 - \frac{s.boyut}{boyut}}{\left(\frac{1}{SH} + \frac{1}{AH}\right)} \quad (3)$$

Sıkıştırılmış veri büyüklüğünün sıkıştırılmamış veri büyüklüğüne oranı "sıkıştırma oranı" olarak kabul edildiği için Denklem 3 aşağıdaki şekilde yazılabilir:

$$İH_{\max} = \frac{1 - s.oranı}{\left(\frac{1}{SH} + \frac{1}{AH}\right)} \quad (4)$$

Eğer veri iletim hızı ve gönderilecek verinin boyutu biliniyor ise, bir veri sıkıştırma algoritması kullanmanın sıkıştırma yapmadan göndermeye göre en çok ne kadar zaman kazandırabileceği Denklem 2'den yola çıkılarak hesaplanabilir:

$$t_{\max} = \frac{boyut}{İH} - \left(\frac{boyut}{SH} + \frac{s.boyut}{İH} + \frac{boyut}{AH}\right)$$

$$t_{\max} = boyut \times \left(\frac{1}{İH} - \left(\frac{1}{SH} + \frac{s.orani}{İH} + \frac{1}{AH}\right)\right)$$

$$t_{\max} = boyut \times \left(\frac{1 - s.orani}{İ.H.} - \frac{1}{S.H.} - \frac{1}{A.H.}\right) \quad (5)$$

5. ALGORİTMALARIN KARŞILAŞTIRILMASI

Karşılaştırma için test verisi olarak "ODTÜ Türkçe Derlemi" [18] içindeki XML'e özgü karakterler çıkartılarak kullanılmıştır (yeni boyutu 15.828.594 byte olmuştur). Seçilen veri sıkıştırma algoritmaları en hızlı sıkıştırma yaptıkları durumları ile Intel Core 2 Duo 1.83 MHz işlemci, 2 GB hafıza, 120 GB sabit disk içeren ve işletim sistemi olarak Microsoft Windows XP Professional kullanan bir bilgisayarda çalıştırılmıştır. Elde edilen sıkıştırma oranı (karakter başına bit olarak verilmiştir: bpc), sıkıştırma süresi ve açma süresi değerleri sıkıştırma oranına göre sıralanmış halde Tablo 1'de verilmiştir.

Tablo 1. Sıkıştırma oranı ve sıkıştırma/açma süreleri

Algoritma (parametre)	Sıkıştırma Oranı	Sıkıştırma Süresi	Açma Süresi
PPMd Var J (-m256)	2,07 bpc	2,04 sn	2,28 sn
Bzip2 (-1)	2,73 bpc	3,70 sn	1,37 sn
Gzip (-1)	3,60 bpc	0,90 sn	0,34 sn
LZW	4,39 bpc	0,90 sn	0,40 sn
LZOP (-1)	4,69 bpc	0,37 sn	0,24 sn
LZP1	4,98 bpc	0,70 sn	0,73 sn
LZRW1	5,11 bpc	0,64 sn	0,65 sn

Tablo 1'deki değerler ile Denklem 4 kullanılarak her algoritma için test edilen sistemde kazanç sağlayabileceği *maksimum iletim hızı* hesaplanmış ve sonuçlar sıralı olarak Tablo 2'de verilmiştir.

Tablo 2. Maksimum iletim hızları

Algoritma (parametre)	Max İ.H.
LZOP (-1)	86 Mbit/sn
Gzip (-1)	56 Mbit/sn
LZW	44 Mbit/sn
LZRW1	35 Mbit/sn
LZP1	33 Mbit/sn
PPMd Var J (-m256)	22 Mbit/sn
Bzip2 (-1)	16 Mbit/sn

Eğer alıcı ve verici cihazlar test bilgisayarımız ile aynı hızda ise; tüm algoritmaların 10 Mbit/sn hızında bir iletim hızında kazanç sağlayabileceği fakat hiçbirinin 100 Mbit/s hızında kazanç sağlayamayacağı Tablo 1'de görülmektedir. Denklem 4'ten anlaşılacağı üzere; *maksimum iletim hızı*, sıkıştırma ve açma hızları ile doğru orantılıdır. 5 kat yavaş alıcı/verici kullanılırsa, tablodaki değerler 5 kat küçük olacaktır.

İletim hattının iki tarafında da test bilgisayarımız ile aynı hızda olan sistemlerin bulunduğu ve iletim hızının 1 Mbit/sn, 10 Mbit/sn ve 100 Mbit/sn olduğu 3 farklı durum için, 10 MB (80 Mbit) büyüklüğünde bir verinin iletiminin sıkıştırma yapmadan iletime göre en fazla kaç saniye kâr veya zarar sağlayacağı Denklem 5 kullanarak hesaplanmış ve sonuçlar sıralı olarak Tablo 3, 4 ve 5'te verilmiştir.

Tablo 3. 1 Mbit/sn iletim hızındaki kazançlar

Algoritma (parametre)	Kazanç
PPMd Var J (-m256)	56,58 sn
Bzip2 (-1)	49,49 sn
Gzip (-1)	43,24 sn
LZW	35,24 sn
LZOP (-1)	32,73 sn
LZP1	29,33 sn
LZRW1	28,04 sn

Tablo 4. 10 Mbit/sn iletim hızındaki kazançlar

Algoritma (parametre)	Kazanç
Gzip (-1)	3,62 sn
PPMd Var J (-m256)	3,20 sn
LZOP (-1)	2,93 sn
LZW	2,78 sn
LZP1	2,12 sn
LZRW1	2,07 sn
Bzip2 (-1)	2,07 sn

Tablo 5. 100 Mbit/sn iletim hızındaki zararlar

Algoritma (parametre)	Kazanç
LZOP (-1)	-0,05 sn
Gzip (-1)	-0,34 sn
LZW	-0,46 sn
LZRW1	-0,53 sn
LZP1	-0,60 sn
PPMd Var J (-m256)	-2,14 sn
Bzip2 (-1)	-2,68 sn

Tablo 1 ve Tablo 3'teki değerlere dikkat edilirse en iyi sıkıştırma oranına sahip olan algoritmaların sıralaması ile 1 Mbit/sn hızında en çok kazanç sağlayan algoritmaların sıralamasının aynı olduğu görülür. Bu sonuç, yavaş ağlarda sıkıştırma oranının sıkıştırma ve açma hızlarından daha önemli olduğunu göstermektedir. Tablo 5'teki değerlere bakıldığında ise sıkıştırma ve açma hızları iyi olan algoritmaların daha az zarar getirdiği görülmektedir. Tablo 1'de yer alan sıkıştırma ve açma süreleri toplanarak küçükten büyüğe sıralanacak olursa, Tablo2 ve Tablo 5'teki algoritmaların sıralaması ile aynı olacağı görülecektir. Bu sonuç ise, hızlı ağlarda sıkıştırma ve açma hızının sıkıştırma oranından daha önemli olduğunu göstermektedir.

Tablo 4'te yer alan sonuçlar ise, iletim hızının orta seviyede olduğu durumlarda, ya çok hızlı olan, ya çok iyi sıkıştırma oranına sahip olan, ya da her iki parametrede de en iyi olmasa da başarılı olan algoritmaların daha çok kazanç sağladığını göstermektedir. Tablo 3'te üçüncü, Tablo 5'te ikinci sırada olan Gzip, Tablo 4'te birinci sırada yer alırken, Tablo 3'ün birincisi PPMd Tablo 4'te ikinci, Tablo 5'in birincisi LZOP ise Tablo 4'te üçüncü olmuştur.

6. SONUÇ VE DEĞERLENDİRME

Ağ iletişimini hızlandırmak için genellikle sıkıştırma ve açma hızları iyi olan fakat sıkıştırma oranları orta seviyede olan LZ-türevi algoritmaların tercih edildiğine Bölüm 1 ve Bölüm 2'de değinmiştik. Yaptığımız karşılaştırmanın sonuçları göstermiştir ki: Yavaş ağlarda LZ-türevi algoritmaları kullanmak yerine PPM-türevi çok iyi sıkıştırma oranına sahip olan algoritmaları kullanmak daha fazla kazanç sağlamaktadır. Bize göre ağ iletişimini hızlandırmak en iyi yöntem: Hem LZ-türevi hem de PPM-türevi iki farklı algoritmanın sisteme yüklenmesi, sistemin iletim hızı, göndericinin sıkıştırma hızı ve alıcının

açma hızına göre her iletim öncesi küçük bir hesaplama yapılarak daha fazla kazanç sağlayacak olan algoritmanın belirlenmesi ve bu algoritma ile verinin sıkıştırılarak gönderilmesi olacaktır.

Bu çalışmada verilen sonuçlar, sıkıştırılacak olan verinin tamamen sıkıştırıldıktan sonra gönderilmeye başlanması, açma işleminin de sıkıştırılmış olan veri tamamen alındıktan sonra başlaması yaklaşımına göre hesaplanmıştır. Ağ iletişimini hızlandırmak için kullanılan yazılımsal ve donanımsal sıkıştırma yöntemlerinin çoğu, veriyi belirli büyüklüklerde parçalara ayırıp göndererek, bir sonraki parçanın sıkıştırması sürerken bir önceki parçanın gönderimini ve açmasını sağlayarak toplam gönderim zamanını (t) azaltırlar. Eğer böyle bir yaklaşım kullanılarak denemeler yapılırsa, bu çalışmada bulunan sonuçlardan daha iyi sonuçlar elde edilebilir. Fakat parçalar çok küçük tutulursa, gönderim sayısı ve buna bağlı olarak paket başlıklarının artması ile gönderim hızı yavaşlayabilir. Parçaların büyüklüğüne göre aktarım zamanları değişiklik gösterecek olsa da, algoritmaların başarı sıralamasında büyük bir değişim olmayacağı düşünülmektedir.

REFERANSLAR

1. Ziv, J., Lempel, A., "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, vol. 23(3), 1977, pp. 337-343.
2. Ziv, J., Lempel, A., "Compression of Individual Sequences via Variable-Rate Coding", IEEE Transactions on Information Theory, vol. 24(5), 1978, pp. 530-536.
3. Cleary, J. G., Witten, I. H., "Data Compression Using Adaptive Coding and Partial String Matching", IEEE Transactions on Communications, vol. 32(4), 1984, pp. 396-402.
4. Rand, D., "The PPP Compression Control Protocol (CCP)", Internet Engineering Task Force Network Working Group RFC 1962, June 1996.
5. Deutsch, P., "DEFLATE Compressed Data Format Specification Version 1.3", Internet Engineering Task Force Network Working Group RFC 1951, May 1996.
6. ITU-T Recommendation V.42bis, 1990.
7. Thomborson, C., "The V.42bis Standard for Data-Compressing Modems", IEEE Micro, vol. 12(5), 1992, pp. 41-53.
8. ITU-T Recommendation V.44, 2000.
9. ISO/IEC 15200: "Information technology -- Adaptive Lossless Data Compression algorithm (ALDC)", 1996.
10. Craft, D. J., "A fast hardware data compression algorithm and some algorithmic extensions", IBM journal of research and development, vol. 42(6), 1998, pp. 733-745.
11. Welch, T. A., "A Technique for High-Performance Data Compression", IEEE Computer, vol. 17(6), 1984, pp. 8-19.
12. Williams, R. N., "An extremely fast Ziv-Lempel Data Compression Algorithm", Proceedings of IEEE Data Compression Conference, Snowbird, Utah, 1991, pp. 362-371
13. Shkarin, D., "PPM: One Step to Practicality", Proceedings of IEEE Data Compression Conference, Snowbird, Utah, 2002, pp. 202-211.
14. Bloom, C. R., "LZP: A New Data Compression Algorithm", Proceedings of IEEE Data Compression Conference, Snowbird, Utah, 1996, pp. 425.
15. Burrows, M., Wheeler, D. J., "A Block-sorting Lossless Data Compression Algorithm", Technical report, Digital Equipment Corporation, Palo Alto, California, 1994.
16. Witten, I. H., Neal, R. M., Cleary, R. J., "Arithmetic Coding for Data Compression", Communications of the ACM, vol. 30, 1987, pp. 520-540.
17. Huffman, D. A., "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of IRE, vol. 40(9), 1952, pp. 1098-1101.
18. Say, B., Zeyrek, D., Oflazer, K. and Özge, U., "Development of a Corpus and a Treebank for Present-day Written Turkish", Proceedings of the Eleventh International Conference of Turkish Linguistics, 2002, pp. 183-192.